
The University Of Sheffield.

# Automated Reverse Engineering of Agent Behaviors

**Wei Li**

A thesis presented for the degree of
Doctor of Philosophy

Department of Automatic Control and Systems Engineering
The University of Sheffield

30th September 2015

# Abstract

This thesis concerns the automated reverse engineering of agent behaviors. It proposes a metric-free coevolutionary approach—*Turing Learning*, which allows a machine to infer the behaviors of agents (simulated or physical ones), in a fully automated way.

*Turing Learning* consists of two populations. A population of models competitively coevolves with a population of classifiers. The classifiers observe the models and agents. The fitness of the classifiers depends solely on their ability to distinguish between them. The models, on the other hand, are evolved to mimic the behavior of the agents and mislead the judgment of the classifiers. The fitness of the models depends solely on their ability to 'trick' the classifiers into categorizing them as agents. Unlike other methods for system identification, *Turing Learning* does not require any predefined metrics to quantitatively measure the difference between the models and agents.

The merits of *Turing Learning* are demonstrated using three case studies. In the first case study, a machine automatically infers the behavioral rules of a group of homogeneous agents through observation. A replica, which resembles the agents under investigation in terms of behavioral capabilities, is mixed into the group. The models are executed on the replica. This case study is conducted with swarms of both simulated and physical robots. In the second and third case studies, *Turing Learning* is applied to infer deterministic and stochastic behaviors of a single agent through controlled interaction, respectively. In particular, the machine is able to modify the environmental stimuli which the agent responds to. For the deterministic behaviors, the machine can construct complex patterns of stimuli that facilitate the learning process, while for the stochastic behaviors, it can interact with the agent on the fly through dynamically changing patterns of stimuli. This allows the machine to explore the agent's hidden information and thus reveal its entire behavioral repertoire. This interactive approach proves superior to learning only through observation.

# Acknowledgments

I consider the process of pursuing my PhD as a journey to learn not only about science, but also about life. Firstly, I would like to thank my supervisor, Dr. Roderich Gross for their support in this journey. This journey could not be perfect without his supervision. He helped me from toddling to walking on the pathway in this research field by motivating, teaching and passing their experience to me without any reservation.

Second, many people have contributed to this journey, especially people in the Natural Robotic Lab: Melvin Gauci, Jianing Chen, Yuri, Stefan, who makes the lab a warm, supportive and fun environment. Special thanks to Melvin Gauci and Jianing Chen. To Melvin, I considered myself very lucky that I met him from the start of this unforgettable journey. I have learned much from him on doing research and life. We not only had many collaborations on research, but we also shared the experience outside science. To Jianing Chen, I appreciated his help.

Outside the research group, there are also a number of people who helped me during my stay in Sheffield, especially Xiao Chen, Yuzhu Guo. This journey would not be completed without their encouragement.

To my parents, thank you for giving me a free environment to grow up and always supporting me.

To my wife, Yifei.

# Contents

# 1 Introduction

## 1.1 Motivation and Objectives

Over the last 50 years, robotic and automation systems have transformed our world and greatly enhanced the quality of our daily life. With the development of science and technology, many intelligent systems that integrate machines, electronics, control and information technologies have emerged. Such systems can accomplish numerous tasks originally performed by humans and often prove superior in terms of precision, speed and cost. They can replace humans in the tasks that require repetitive and monotonous operations. For example, in the automotive industry, robotic and automation systems have been widely used for machining, welding, painting and assembling. From an industrial perspective, these systems have lowered the product cost and improved the efficiency of production, thus greatly increasing the speed of industrialization.

Robotic and automation systems also contribute to scientific research. For example, in outdoor environments such as those that may be beyond humans' capabilities of reach (e.g., other planets), robots can be sent to collect samples or conduct experiments. In 2004, two robots—*Spirit* and *Opportunity* were sent to Mars by NASA in a mission to explore the geology of this planet [1]. The primary goal of this mission was to analyze the rocks and soils on Mars and explore the activity of water in the past. In the indoor laboratories, robotic and automation systems have also played an important role. With the help of these systems, researchers can collect data much faster than ever before. For instance, high-throughput screening (HTS) systems [2], which are widely used in drug discovery, allow the researchers to conduct millions of experiments in a very short time. Such systems consist of several components, including sensing units, robotic manipulator, control system, etc. Besides data collection, robotic and automation systems can also help analyze the data using intelligent software, which provides an effective tool for data analysis in scientific research and frees researchers from the tedious and monotonous

process of data analysis if done manually. This accelerates the development of scientific research to a great extent.

A particular scientific area in which robotic and automation systems play a significant role is *ethology*—the study of animal behavior [3]. There are four types of questions to be investigated in ethology: questions concerning causes, functions, development and evolution [3]. Causes refer to the mechanisms of animals that are innate as well as the external/internal stimuli that affect such behavior. Functions concern what is the purpose of this behavior such as mating, aggregation or foraging. The development of animal behavior concerns how animals learn such behavior during their life and how such behavior is affected by experience, while evolution relates to how the behavior changes over generations in the course of natural evolution. Over centuries, these four questions have been investigated by ethologists primarily in well-controlled (indoor) laboratories or outdoor environments. Ethology 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. For instance, models of animal decision-making can be used to predict their behavior in novel environments, which can help in making ecological conservation policy [4]. Knowledge about animal behavior has also been applied for solving computational problems [5], and for constructing biologically-inspired robotic agents [6].

Before the emergence of computers, ethologists needed to observe the animals and analyze the data manually. They also needed to learn how to control the environmental conditions in a meaningful way to extract most of the information from the animals under investigation. Such process of analysis sometimes is time-consuming and tedious. With the help of intelligent and automation systems, nowadays researchers can conduct experiments much more efficiently. However, these systems are often secondary, and in most of the cases they are merely accomplishing mechanical and repetitive work. The question is whether we can build a machine/system that can accomplish the whole process of scientific investigation and automatically analyze experimental data, search for correlations between different elements, generate new hypotheses and devise new experimental conditions to be investigated. In other words, can we build a system that is able to automatically conduct scientific research without (or with minimal) human intervention? Recently, the development of "robot scientists" shows that such systems may be within reach [7, 8, 9]. Following this motivation, this thesis aims to pave the way for further development in science automation [8], especially in the area of ethology. The ultimate goal of this thesis is to contribute to the study of animal behavior through

developing an automated system identification/modeling method.

System identification is a process of modeling natural or artificial systems with observed data. It has drawn a large interest among researchers for decades [10, 11]. One application of system identification is the reverse engineering of agent behavior (biological organisms or artificial agents). Many studies have investigated how to deduce rules of agent behavior using system identification techniques based on various models [12], one of which is an agent-based model [13], which simulates the complex behavior of a group of agents with relatively simple behavioral rules. The global behavior emerging from interaction within agents and between agents and environments is used for refining the model. Evolutionary computation which draws inspiration from biological evolution has proven to be a powerful method to automate the generation of models, especially for behaviors that are hard to formulate [14, 15, 16]. Evolutionary computation also provides a potential realization for automation science, as models evolve in an autonomous manner. It is the main technique that is investigated in this thesis for performing system identification.

A limitation of existing system identification methods is that they rely on predefined metrics, such as the sum of squared error, to measure the difference between the output of the models and that of the system under investigation. Model optimization then proceeds by minimizing the measured differences. However, for complex systems, defining a metric can be non-trivial and case-dependent. It may require sufficient prior information about the systems. Moreover, an unsuitable metric may not well distinguish between good and bad models, or even bias the identification process. This thesis aims to overcome the limitation by introducing a metric-free system identification method that allows a machine to infer agent behavior automatically.

## 1.2 Problem Statement

The investigated (agent) behaviors in this thesis are simulated by a computer or physical robots. The agent to be studied is put in an environment. Its behavior depends on interaction with the environment and with other agents in a group (if any). The machine will observe the agent's motion. It is assumed that it is possible to track the position and orientation of the agent at discrete steps in time. In general, one could monitor a range of variables including the agent's motion, heart rate, morphology, body temperature, etc. Furthermore, the machine can control environmental stimuli that the agent may

respond to. The system identification task is to infer the observed behavior, in other words, the agent's behavioral rules.

Three case studies are presented in this thesis. The first case study is about inferring swarm behaviors, which are emergent behaviors that arise from the interactions of numerous simple individuals [17]. Learning about behaviors that are exhibited in a collective manner is particularly challenging, as the individuals not only interact with the environment but also with each other. Typically their motion appears stochastic and is difficult to predict [18]. For instance, given a swarm of simulated fish, one would have to evaluate how close its behavior is to that of a real fish swarm, or how close the individual behavior of a simulated fish is to that of a real fish. Comparing behaviors at the level of the swarm (i.e., emergent behaviors) is challenging [19]. It may require domain-specific knowledge and not discriminate among alternative individual rules that exhibit similar collective dynamics [20]. Comparing behaviors at the level of individuals is also difficult, as even the same individual fish in the swarm is likely to exhibit a fundamentally different trajectory every time it is being looked at. In this case study, the machine observes the motion of each individual in the swarm and needs to automatically infer the behavioral rules of the swarming agents.

The second case study is about inferring the deterministic behaviors of a single agent, and investigating how the agent responds to the environmental stimuli. The machine has full control over the environmental stimuli that the agent may respond to, and at the same time observes the agent's motion. We investigate a particular type of agent behavior; from the perspective of system identification, the agent behavior has low observability. That is, randomly generated sequences of inputs (stimuli) are not sufficient to extract all the hidden information of the agent and thus infer its behavior. Instead, in order to reveal all the agent's behavioral repertoire, the machine needs to construct complex patterns of stimuli in particular ways that facilitate the learning process.

In the third case study, we investigate how to infer the stochastic behaviors of a single agent. The agent still responds to the environmental stimuli; however, its behavior is not only determined by the environmental stimuli but also by probability. In other words, constructing a fixed sequence of stimuli may not trigger all the agent's behavioral repertoire as mentioned in the second case study. The machine needs to interact with the agent during the experimental process and dynamically change/control the environmental stimuli based on the agent's observed motion to reveal its hidden behavior. Inferring such stochastic behaviors through predefined metrics is challenging, as given the same

sequence of stimuli, the agent is likely to exhibit different behaviors.

## 1.3 Contributions

The main contribution of this thesis is a novel system identification approach—*Turing Learning*—which allows a machine to infer agent behavior in an autonomous manner. *Turing Learning* uses a coevolutionary algorithm comprised of two populations. A population of models competitively coevolves with a population of classifiers. The classifiers observe the models and agents. The fitness of the classifiers depends solely on their ability to discriminate between them. Conversely, the fitness of the models depends solely on their ability to 'trick' the classifiers into categorizing them as agents. Unlike other system identification methods, *Turing Learning* does not rely on predefined metrics to gauge the difference between the behaviors of agents and models.

The specific contributions are as follows:

1) A metric-free approach to automatically infer the behavioral rules of a group of homogeneous agents. Both the inferred model parameters and emergent global behaviors closely matched those of the original system.

2) A metric-free approach to automatically produce a classifier (system) that can be used for detecting abnormal behaviors (e.g., faulty agents in a swarm). This was validated by both simulated and physical robot swarms.

3) A physical system for performing metric-free system identification in an autonomous manner. The system was validated through successfully inferring the behavioral rules of a physical robot swarm. Both the inferred model parameters and emergent global behaviors closely matched those of the original system.

4) A metric-free approach to automatically infer deterministic behavior of a single agent. The machine actively constructs complex patterns of inputs (stimuli) to extract the agent's hidden information.

5) A metric-free approach to automatically infer stochastic behavior of a single agent through controlled interaction. The machine dynamically changes the environmental stimulus to interact with the agent on the fly.

## 1.4 Publications

This thesis presents the author's own work. Some parts of the thesis have been published. A preliminary version of Chapter 3 was orally presented in a conference by the author:

- **W. Li**, M. Gauci and R. Groß, "Coevolutionary learning of swarm behaviors without metrics," *Proceedings of 2014 Genetic and Evolutionary Computation Conference (GECCO 2014)*. ACM Press, Vancouver, Canada, 2014, pp. 201–208.

A preliminary version of Chapter 5 was orally presented in a conference by the author of this thesis:

- **W. Li**, M. Gauci and R. Groß, "A coevolutionary approach to learn animal behavior through controlled interaction," *Proceedings of 2013 Genetic and Evolutionary Computation Conference (GECCO 2013)*. ACM Press, Amsterdam, The Netherlands, 2013, pp. 223–230.

Preliminary versions of Chapters 3 and 4 have been included in a paper submitted to the following journal:

- **W. Li**, M. Gauci, J. Chen and R. Groß, "Reverse Engineering Swarm Behaviors Through Turing Learning," *IEEE Transactions on Evolutionary Computation*, under review.

Apart from the work presented in this thesis, the author has also contributed to other projects. This led to the following publications:

- M. Gauci, J. Chen, **W. Li**, T. J. Dodd, and R. Groß, "Self-organized aggregation without computation," *The International Journal of Robotics Research*, vol. 33, no. 8, pp. 1145–1161, 2014.

- J. Chen, M. Gauci, **W. Li**, A. Kolling and R. Groß, "Occlusion-based cooperative transport with a swarm of miniature mobile robots." *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 307–321, 2015.

- M. Gauci, J. Chen, **W. Li**, T. J. Dodd, and R. Groß, "Clustering objects with robots that do not compute," in *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*. IFAAMAS Press, Paris, France, 2014, pp. 421–428.

During his PhD studies, the author has also been a Marie Curie Research Fellow with the Department of Mechanical Engineering, University of Western Ontario, Canada, where he contributed to a project on Mechanical Cognitivization. This led to the following publication:

- G. Avigad, **W. Li**, A. Weiss, "Mechanical Cognitivization: A kinematic system proof of concept" *Adaptive Behavior*, vol. 23, no. 3, pp. 155–170, 2015.

## 1.5 Thesis Outline

This thesis is structured as follows:

- Chapter 2 describes the background of the thesis as well as the related work presented in the literature.

- Chapter 3 introduces a metric-free system identification method—*Turing Learning*. It is applied to learn two swarm behaviors (self-organized aggregation [21] and self-organized object clustering [22]) through observation. Section 3.1 describes the implementation of *Turing Learning* and the two swarm behaviors. Section 3.2 introduces the simulation platform and setups for performing coevolution runs. Section 3.3 presents the results obtained from the two case studies. In particular, Section 3.3.1 analyzes the evolution of models, objectively measuring their quality in terms of local and global behaviors. Section 3.3.2 investigates the coevolutionary dynamics. Section 3.3.3 investigates the evolution of classifiers, showing how to construct a robust classifier system to potentially detect abnormal behaviors in the swarm. Section 3.3.4 studies the effect of observing only a subset of agents in the swarm. Section 3.3.5 presents a study where an aspect of the agents' morphology (their field of view) and brain (controller) are inferred simultaneously. Section 3.3.6 shows the results of using *Turing Learning* to learn other swarm behaviors.. Section 3.3.7 presents a study showing the method's sensitivity to noise. Section 3.4 summaries the findings in this chapter.

- Chapter 4 presents a real-world validation of *Turing Learning* to infer the behavior of a swarm of physical robots. Section 4.1 introduces the physical platform, which includes the robot arena, the robot platform and the sensors implementation. Section 4.2 details the tracking system, including motion capture and video

processing. Section 4.3 describes the programs executed by each component (machine, agent and replica) during the coevolutionary learning process. Section 4.4 describes the experimental setup. Section 4.5 discusses the results obtained. This includes the analysis of the evolved models and the analysis of the evolved classifiers. Section 4.6 analyzes the sensitivity of *Turing Learning* for individual failure during the experimental process. Section 4.7 summaries the results obtained and discusses the findings in this chapter.

- Chapter 5 investigates how to infer the deterministic and stochastic behaviors of an agent through *Turing Learning* with interaction. The machine not only observes the behavior of the agent but also interacts with it through changing the stimulus that influences the agent's behavior. This chapter is organized as follows. Section 5.2 describes the methodology, illustrating how *Turing Learning* is extended to have interactive capability. The deterministic and stochastic behaviors under investigation are presented as two case studies (Sections 5.3 and 5.4). Section 5.3.1 describes the deterministic behavior. Section 5.3.2 presents the simulation setup. Section 5.3.3 presents the results of inferring the deterministic behavior. Section 5.4.1 describes the stochastic behavior (using a state machine) for the general case. Section 5.4.2 presents the simulation setup for inferring the stochastic behavior. Sections 5.4.3 and 5.4.4 present the obtained results for the case of 2 states and 3 states, respectively. Section 5.5 summaries the chapter.

- Chapter 6 concludes the thesis and discusses the future work.

# 2 Background and Related Work

This chapter presents the literature review. In Section 2.1, we introduce the background. This includes a brief introduction of the development of artificial intelligence (AI) and robotics, and the research that has been done in recent years in the areas 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 introduces how AI/robotics and animal behavior study benefit from each other, which includes how animal behavior can be used as inspiration for AI and robotics and the 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 intelligent behaviors such as pattern recognition and decision making. However, intelligence is not a property that is limited to biological creatures. It should be equally applicable to computers or machines. The term AI emerged in a conference in 1956 at Dartmouth College, where several pioneers of this filed including Marvin Minsky, John McCarthy, etc., discussed the development of digital computer and the future of AI. The definition of AI is still a disputed topic. Some researchers argued that AI is to simulate the intelligent behaviors which 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 live creatures when encountering the same problems [**?** ]. Others gave the following definition: "Artificial Intelligence is the study of mental

faculties through the use of computational models" [**?** ]. According to Fogel [**?** ], an intelligent system should know how to make decision 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 [**?** ], Minsky even 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 [**?** ] 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 system—a computer program that mimics a human expert's decision-making capability [**?** ], significantly promoted the development of AI. An expert system can solve complicated problems through reasoning about the knowledge (which is mainly represented as *if-else* rules) it has. One of the most representative examples is IBM's chess program (Deep Blue). It defeated the champion of the world chess (Gary Kasparov) in 1997 [**?** ], which provides evidence that a computer program can even outperform a human expert in term of decision-making ability. An expert system consists of two components: knowledge base and inference engine. Knowledge base contains facts and rules that are known to the system. Inference engine uses the knowledge to make decision and derive new rules, which are then stored in the system to update the knowledge base. Expert systems have many commercial applications such as medical diagnosis [**?** ], prediction [**?** ] and monitoring [**?** ], etc. 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 [**?** ] 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", etc. Fuzzy logic is used common 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 market 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 in such as air conditioner, digital camera and and hand

writing recognition, etc. Another representation of AI is neural network, which mimics the processing ability of nervous systems of biological creatures (especially human brain). Through a combination of weights and excitation functions (e.g., sigmoid function), 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 the robots "smarter". For example, a robot with AI can move autonomously and make decision while interacting with the environment it is operating on. Through combining AI and robotics, machines can be created to behave more like humans or animals. Instead of only executing fixed programs as in a car assembling line, robots can learn and adapt to the changing environment.

There are two common architectures adapted for the control of a robot: deliberative architecture and subsumption architecture [**?** ]. In deliberative architecture, the robot operates on a top-down fashion and its action mainly depends on planning. A typical cycle is: $sensor \rightarrow plan \rightarrow act$. In the sensor stage, the robot gets the information from the world based on sensors such camera, infrared sensors. 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 to decide which action to take to fulfill a goal (e.g., maximize its reward). This architecture has led to many successful applications. The pioneer work is *Shakey the robot* which is capable of reasoning about its own actions [**?** ]. In this work, the environment the robot is operating on 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 [**?** ]. Another architecture adapted widely nowadays is subsumption architecture [**?** ], in which the robot makes decision based on $sensor \rightarrow act$ 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 by each layer. This could enhance the robustness of the robotic control system. Some famous robots using subsumption architecture are Allen [**?** ], Herbert [**?** ] and Genghis [**?** ].

### 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 [**?** ] was created in 1993, such systems are increasingly used in drug discovery, agriculture and energy labs, etc. Nowadays, it is the demand that machines could automate the whole process of scientific research. The question of whether it is possible to automatically conduct scientific research is very interesting in theory, and it also involves a lot of practical work which needs to be solved (e.g., communication and coordination). In order to speed up experimental process, researchers should take advantage of intelligent and automation systems to help, for example, collect and analyze thousands of data, because these things are very time-consuming and boring if only carried out manually. The ideal situation is to make a machine conduct scientific research automatically without or with little human intervention, and it can do experiments day and night in a constant manner without any tiredness and complain.

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 [**?** ] are one of the early efforts. HTS systems could do many things such as preparation, observation and analysis of assay, greatly enhancing the speed of data collection and analysis process in a short time. Recently, King, et al. have built a Robot Scientist—Adam, which can automatically generate functional genomics hypotheses about the yeast *Saccharomyces cerevisiae* and carry out experiments to test and refine the hypotheses based on the techniques developed in AI and robotics [**?** 7]. This Robot Scientist is able to do plenty of experiments and observations a day. The experiments are performed automatically by machines, which makes it possible to test all aspects of experimental process. 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 [**?** ].

In system identification area, Gauld et al. have developed a digital automated identification system (DAISY) to identify biological species automatically with high accuracy using advanced image processing technique [**?** ]. This technology has gone though great

improvement in recent years, raising the possibility of automation, or at least semi-automation, in the process of routine taxonomic identification. In [**?** ], MacLeod et al. reported that an imaging system that is originally designed for identifying marine zooplankton was used by the US government for monitoring horizon oil spill in the deep water. 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].

## 2.2 Evolutionary Computation

Evolutionary computation is a technique which draws inspiration from biological evolution. It is a stochastic search method and uses trial-and-error to guide the search process. We use evolutionary computation technique to automate generation of models during the system identification process. Section 2.2.1 briefly introduces the biological evolution. Section 2.2.2 details the principles, strengths, weaknesses of evolutionary computation, including evolutionary algorithms and coevolutionary algorithms. Section 2.2.3 presents three main applications of evolutionary computation and the 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 [**?** ], species fight for survival. The species that can fit the environment survive, and others that can not would die. This phenomenon is regarded as *survival of the fittest* or *natural selection*. There are heritage (e.g., through sexual reproduction) and random mutation among species' genes. The genes that help the species survive would have a higher chance of be preserved and passed on to the next generation, while the genes that are harmful or not useful would be abandoned.

Table 2.1: The relationship between different species

| | 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) |

Natural selection tends to reserve and accumulate small beneficial genetic mutations. 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 threat from 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 more likely to die out. Natural selection helps the species to compete and adapt better in the environment. At the same time, it also accelerates the extinction of the species that can not fit the environment. The dinosaur used to be a dominant species in the ancient world due to its big body and flying ability. 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 led to accelerate the extinction of dinosaurs.

Coevolution is special form of evolution, which involves in the simultaneous evolution of two or more 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 on its chromosome, but also the interaction with the individuals from other species. Although the correlation between different species is complicated, for a specific species, there are three possibilities: beneficial, injured and neutral. Therefore, through permutation and combination, the relationship between two species can be summarized in Table 2.1. It includes 6 concrete relationship: reciprocity, neutral, symbiosis, amensalism, predation and antibiosis [**?** ].

Where, symbol '+', '-' and '0' represent beneficial, injured and neutral. For example, "+, -" indicates A benefits and B gets injured in the relationship.

### 2.2.2 Introduction of Evolutionary Computation

#### 2.2.2.1 Principles

Based on the principle of biological evolution, a bi-inspired algorithm — genetic algorithm (GA) was proposed by Halland in 1960s [**?** ]. GA simulates the heritage, mating, and mutation of the natural evolution. In GA, the solution for a given problem is represented as chromosome, which contains several genes. Each gene could be a binary number, integer, or floating-point number. Heritage is also called reproduction in GA. Mating corresponds to crossover/mate, in which different individuals exchange genes. 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 natural evolution. It creates the diversity and creativity among the population. GA is driven by a fitness function, which defines the goal or solution to be achieved. The evolutionary process is to optimize (e.g., maximize) the fitness of the individuals in the population. There are other types of evolutionary algorithms that based on the basic idea of natural evolution. Fogel, Owens and Walsh invented *evolutionary programming (EP)* [**?** ]. Rechenberg and Schwefel introduced *evolution strategies (ES)* [**?** ], which are mainly dealing with real-value continuous optimization problem. In the early 1990s, another new evolutionary algorithm called *genetic programming (GP)* was presented by Koza [**?** ]. Fig. 2.1 shows a brief classification of evolutionary algorithms.

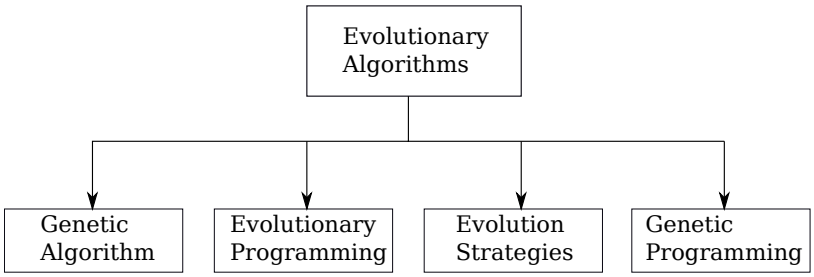



Figure 2.1: Classification of evolutionary algorithms

The implementation of evolutionary algorithms follow a general flow during the operation process. They can be divided into 5 steps: initialization, evaluation, mutation, selection and termination. At the beginning, a random population of individuals is initialized. These could be several random strings (chromosomes or individuals), each of which contains several genes (e.g., floating point number). These strings are encoded of the

solutions to be achieved. Different chromosomes are then evaluated using the predefined fitness function. The fitness of individuals in the population is only decided by their own chromosomes. That means in different generations, the fitness of individuals who own the identical chromosome is constant. Selection happens after the evaluation of each individual, and the ones with higher fitness normally have a higher chance of being selected to the next generation and have offsprings. The offsprings would go through mutation, which helps to maintain diversity of the whole population. Fig. 2.2 show a diagram of how the evolutionary algorithms proceed.

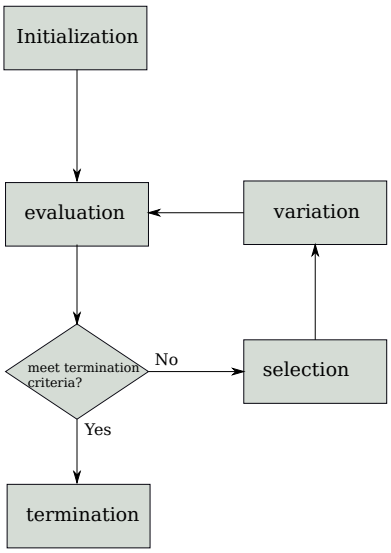



Figure 2.2: This diagram shows the flow of evolutionary algorithms.

As we mentioned before, the creatures in nature are not independent, and they have different kinds of relationship shown in Table 2.1. The evolution of individuals among different species are coupled. Based on the idea of biological coevolution, the coevolutionary algorithms, which coevolve simultaneously two or more populations, are widely used in various research areas [**?** ]. Figure 2.3 shows a schematic diagram of coevolutionary algorithms between two different populations. 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.

The essential difference between evolutionary algorithms and coevolutionary algorithms

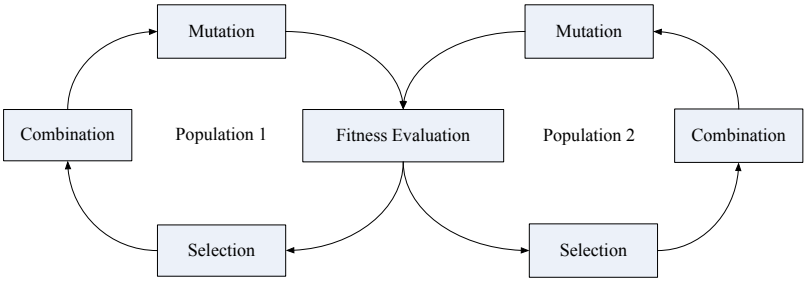



Figure 2.3: A schematic diagram of coevolutionary algorithms between two different populations.

is the way of fitness evaluation. While the fitness of an individual in evolutionary algorithm depends only one its chromosome, in coevolutionary algorithms the individual's fitness depends on its performance when evaluated by the individuals from the other populations. According to the way of evaluation, the coevolutionary algorithms are generally divided into two categories—competitive coevolutionary algorithm (Comp-CEA) and cooperative coevolutionary algorithm (Coop-CEA). 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 [**?** ], competitive coevolution can produce a phenomena of "arm races" by increasing the complexity of each population in the coevolution. The evolution of one population may drive another population to evolve new strategies, which makes 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, thare are several cooperative species evolving simultaneously, and each sub-species represent a part of the whole solution, which is the combination of the eventual solution in each sub-species according to a certain sequence. 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 [**?** ]. 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 in different generations may vary because of changing opponents. Conversely, the fitness in evolutionary algorithms is called absolute or objective fitness.

Comp-CEA can also be applied in single population or multiple populations. The single-

population Comp-CEA is realized by competition between individuals within the same population. 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, its simple subjective fitness is the number of evaluators (in the current population) that this learner defeated [**?** ]. It is described in equation (2.1) as follows:

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

$CF_i$ is the fitness of learner $i$.

Another fitness calculation approach is called competitive fitness sharing [**?** ] as described in the following:

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

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

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

In competitive fitness sharing [**?** ], the learner that could defeat the more competitive evaluator get 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 important genetic materials which are worth passing to the next generation.

There are many ways to choose the evaluators (temporary opponents). Random Paring [**?** ] means finding a random temporary opponent for each leaner. In single elimination tournament [**?** ], all the individuals randomly match, and the losers are taken out and winners are selected into next round of random match. Round Robin [**?** ] means all the evaluators are the temporary opponents of each leaner. There are also other ways such as K-random opponent [**?** ] and fitness sampling [**?** ]. In fitness sampling, the selected

temporary opponents should have relatively higher fitness value in the last generation. The evaluation time for random paring is the shortest, but the performance is the worst; the calculation time for round robin is the longest, but the performance is the best. In our thesis, we chose to use the simple subjective fitness calculation and Round Robin.

#### 2.2.2.2 Strengths

The primary advantage of evolutionary computation is it is conceptually simple. Evolutionary computation draws inspiration from biological evolution, and it can be implemented using simple operators as mentioned before. Moreover, evolutionary computation is task-independent, which means it can be applied to virtually any task that can be formulated as a function optimization problem [**?** ]. The advantage of function optimization as well as black-box optimization will be detailed in Section 2.2.3.1. Evolutionary computation (especially evolutionary algorithms) can be run in parallel (e.g., using GPU computing) to accelerate the evolutionary process. Each individual can be evaluated independently according to the predefined fitness function. Only the selection and recombination process need the serial computing. Evolutionary computation is robust to changes of circumstance. Once the circumstance has changed, the evolve solutions can be used as a start for further development without the need to restarting the whole process [**?** ]. Apart from the general advantages of evolutionary computation, when compared with evolutionary algorithms, 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" [**?** ], 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 would be challenging to define a fitness to determine which program is better. An effective way of evaluating a chess program is making it play with other programs and then calculating its subjective fitness [**?** **?** ].

#### 2.2.2.3 Weaknesses

The main weakness of evolutionary computation is that it requires a sufficient number of generations to obtain good solutions. In simulation, this would not be a problem; however when the evaluation needs to be conducted on the real system, it would take a long time and may also cost a lot. If the fitness function can not be defined properly, the obtained solutions may be biased. Although coevolutionary algorithms have some advantages over evolutionary algorithms, there are three main pathologies:

- *Red Queen Effect* During the coevolutionary process, two populations keep competing with each other. For a particular population, when *Red Queen Effect* happens, the variation tendency of subjective fitness and objective (absolute) fitness is opposite [**?** ]. For example, the increase of its subjective fitness may correspond to the decrease of its objective fitness. In other words, the objective fitness does increase, but the landscape of subjective fitness does not reflect such situation.

- *Cycling* In coevolutionary algorithms, the aim of each individual is to defeat its temporary opponents. The optimal solution which is obtained in the previous generations would probably be lost. After some generations, the optimal solution may be found but lost again. The coevolution is trapped into this endless cycling, failing to find the optimal solution [**?** ]. The general way of overcoming the problem of "cycling" is "hall of fame"[**?** ]. "hall of fame" obtains the excellent 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.

- *Disagreement* During the coevolutionary process, when one species is entirely better than the other, disengagement will occur. In this case, the selection criteria won't make any sense and selection gradient will disappear, since the subjective fitness of each population is constant. The diversity of populations will converge into zero, and it is impossible to form "arm race" [**?** ]. The common solution for solving the disengagement problem is "resource sharing" and "reducing virulence"[**?** ]. "Resource sharing" keeps the diversity of populations, and "reducing virulence" selects the evaluators to keep the gradient of learners' fitness.

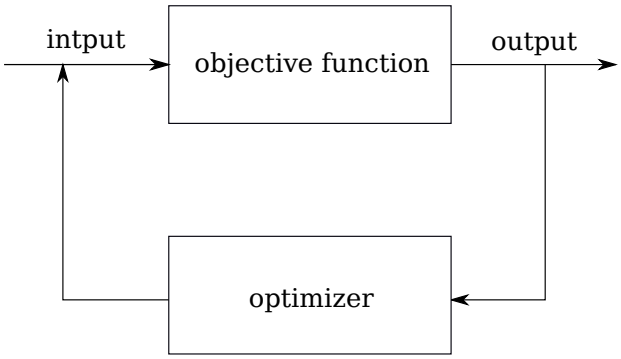



Figure 2.4: This diagram shows the process of black box optimization. The task is to find a candidate solution (input) that can optimize (e.g. maximize or minimize) the objective function.

### 2.2.3 Applications of Evolutionary Computation

Evolutionary algorithms are widely used for solving various engineering tasks ranging from optimization, control, pattern recognition, robotics and system identification/modelling. In the following sections, we will focus on three main fields in which evolutionary computation technique plays a role.

#### 2.2.3.1 Black Box Optimization

One of the promising application of evolutionary computation is black box optimization. Black box optimization refers to such problems that the optimization algorithm aims to optimize an objective function without assuming the hidden structure of that function (e.g., linear or differentiable). In particular, the aim is to found a set of inputs that can maximize or minimize the output of the objective function. For example, in a traveling salesman problem, in which the task is to find the best combination of route of cities (input) that can minimize the length of tour of visiting all cities. Fig. 2.4 shows a diagram of the black box optimization.

Due to the complexity of the real problem in nature, the stochastic search algorithm such as evolutionary algorithms provide us an efficient and relatively 'perfect' solutions. Although evolutionary algorithms could not guarantee the best solution would be found every time, they are still superior to many traditional search algorithms such as the greedy local search algorithm [**?** ]. There are many real-world examples that using

evolutionary computation techniques to solve the optimization problem. In the area of nanophotonic light trapping, an urgent 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 optimal parameters. In [**?** ], a highly efficient light-trapping structure was designed using a genetic algorithm. It was shown that this new structure can increase the trapping efficiency three times comparing with the classic limit. 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 [**?** ], and one of the antennas was used in the mission. The antennas is a critical device for the spacecraft to communicate with the ground, as faulty communication may cause a lot of data lost or the crash of the spacecraft. The antennas designed using evolutionary algorithms are significantly better than those designed by human experts.

In the following, we list several cases (but not all) that evolutionary computation could be applied to solve the 'tough' optimization problem.

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

- *Multi-Model* Multi-model means a system (function) has more than one optimums (e.g., Rastrigin Function). The one/s with the best fitness value is/are considered as global optimum/s, and the other are considered as local optimums. These local optimums around the global optimum/s are very misleading for the gradient-based search algorithms (e.g., *hill climbing* algorithm), as the solutions may easily get trapped at the local optimums. Evolutionary algorithms are shown to be very efficient to find the global optimum/s [**?** ].

- *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 the separable function, it would be much easier to optimize, as we can treat each variable separately. However, for non-separable system, the variables are normally coupled. Also, the function may not be differential, which makes many mathematical

optimization algorithms (e.g., quasi-Newton BFGS algorithm [**?** ] or conjugate gradient algorithm [**?** ]) infeasible.

- *Multi-objective* When encountering real-world problem, 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 more importantly cost. As some of the objectives (such as performance and cost) are conflicting, the designer needs to find a Pareto optimal solution [**?** ], in which none of the value of the objective functions can not be increased without considering decreasing the value of other objectives. Evolutionary multi-objective optimization is an efficient technique for generating such Pareto optimal solutions for multi-objective optimization problems. For a review, see [**?** ].

#### 2.2.3.2 Evolutionary Robotics

Another application of evolutionary computation is evolutionary robotics, in which the controller and/or morphology of the robots are automatically generated. The user considers the robot as a whole and only needs to specify the criterion which is 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. The normal procedure of evolutionary robotics is as follows. First, an initial population of random controllers are generated. Each controller are represented by a chromosome. If the controller is a neural network, the chromosome is a vector of real numbers. Each controller was imported into the robot(s), and the robot's performance when performing various tasks is measured and evaluated using the pre-defined fitness function. After some genetic operators (e.g., crossover, mutation), the 'fitter' robot controller has a higher chance of being selected and generating offspring in the next generation. This process iterates until a good solution is found.

In contract to evolutionary robotics, another method of designing robot controller and/or morphology is behavior-based approach, where the designer divides the whole system into several simple parts intuitively. These separate parts are then integrated all in once through a coordination mechanism—competitive or cooperative coordination [**?** ]. In competitive coordination, only one part has an effect on the output of the robot, while in cooperative coordination, several parts contribute to the output of the robot

with different weights. The behavior-based method was shown to be very robust in many tasks such as gait control in locomotion and object transport. The challenge of designing robot controller and/or morphology using behavior-based method is it requires a lot of experience from the designer. Moreover, when the system is highly coupled, separating the whole design into different parts may not be a good strategy. However, evolutionary robotics provides the designer an alternative way of controlling the robot as a whole, rather than focuses on the details of each separate component. The synthesized control system is a result of self-organized process.

There are many control structures that can be adapted in evolutionary robotics. Artificial neural network is a popular choice for the robot's controller due to its simple representation and strong power of control. The tree-based structure like LISP is also very robust, and it is widely used in evolutionary programming [**?** ]. The *building blocks* method was proposed by Brooks [**?** ]. However, these blocks are described using high-level languages, which are not suitable for evolution in the low level. Comparing with other control structures, artificial neural network have many advantages [**?** **?** ]. According to the types of behavior (e.g., simple perception or non-linear dynamic), we can choose different neural networks (forward neural networks and recurrent neural networks, etc.). Recent development reveals that we can even evolve the topology and weights of the neural networks [**?** ].

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, a range of work has been presented, ranging from simple behaviors, such as phototaxis behavior, self-charging behavior to complex behaviors such as navigation and locomotion of robots with multiple degrees of freedom. In biology, evolutionary robotics is used as a tool to understand the general principles of evolution. It provides much more efficient and faster way to validate or even create hypothesis based on evolution in simulation or real robots, compared with the slow evolutionary process in nature. AVIDA [**?** ] and AEvol [**?** ] 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 altruistic plays an important role in cooperation among species [**?** **?** ], the conditions of emergence of communication during the evolution [**?** ], and how morphology and control are coupled [**?** ], coevolution of predator and prey [**?** **?** ].

Two methodologies are adapted in evolutionary robotics. The first one is evolving the robot(s)'s solutions in simulation and then transferring the best solution into the real robot(s); the other is evolving the solution directly on the physical robot(s). A drawback of the first method is *reality gap*. That is, the simulation used for generating the solutions may not can not match the robot(s)' real operating environment, causing the decline of the robot(s)'s performance. Many works are done to reduce the *reality gap* [**?** **?** **?** ]. In [**?** **?** ], the author presented the transferability approach to improve the control quality of the robot. The controllers were generated in simulation, and the best controller (selected with multi-objective criteria) was transferred into the real robot, and the data collected were used for refining the simulator. This increased the quality of controller generated in simulation, and also reduced the number of experiments to be conducted on the real robot.

In [**?** ], the evolution was even performed directly on the real (Khepera) robot to evolve the controller to perform navigation task. The major advantage of this is the *reality gap* has been completed avoided. However, it may take a significant amount of time to evolve a desirable solution. In [**?** ], it took two weeks to evaluate only 100 generations. The battery is still a problem as it can only last a limit time. Although the authors in [**?** ] used a wire to connect the power of the robot with a charging station, it is not desirable for outdoor experiments and multiple robots. Recently, a distributed online onboard evolutionary method (artificial embodied evolution) has received much attention [**?** **?** **?** ]. 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 particularly suitable for the situation where the environment that the robots are operating on is not predictable or changing after deploying the robots. The robots need to evolve 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* [**?** ]. 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 technique makes this method appealing [**?** ].

#### 2.2.3.3 System Identification

System identification which is about building the model of a hidden system through conducting a set of experiments is widely used in both academic and industrial areas [**?** ]. The experiments are conducted by a series of inputs into the system and the outputs corresponding to the inputs are collected. The process of system identification is to find a model that fits the inputs and outputs. System identification process is composed of observed data, model structure, a criterion to evaluate different models according to the observed data, validation of the obtained model based on different data set, and revision of model if necessary.

There are two system identification approaches: the offline approach and online approach. In the offline approach, the observation 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 easy to collect or the parameters and operating environment of the system do not change too much in a short time. However, in some cases, the parameters of the system are always changing due to different operating environment. That means the obtained model based on the previous observation data can not be applied to the new situation any more. In this case, the model should be updated using the new observed experimental data and online system identification method. The two approaches of system identification are shown in Fig. 2.5. Note that for some systems, there are only outputs.

System identification can be divided into two main procedures: modeling and estimation. Modeling defines the order or general structure of the hidden system [**?** ]. Estimation identifies the parameters associated with the given structure. There are many estimation algorithms to determine the parameters for a given structure. Typical estimation algorithms are recursive prediction error methods [**?** ] which are based on gradient search. Gradient search method such as greedy algorithm can easily get trapped in local optima, especially when there are numerous local optimal points near the global optima. An alternative search method is using evolutionary algorithms. As evolutionary algorithms use population-based search method, this makes it more likely to get rid of local optima. There are some system identification methods that combine the modeling and estimation process, e.g., NARMAX method [11]. Neural network is a also good representation of the system under investigation, as its topology (structure) and weights (parameters) can be optimized simultaneously. The disadvantage of neural network is it is difficult to interpret the obtained model especially when there are multiple layers.

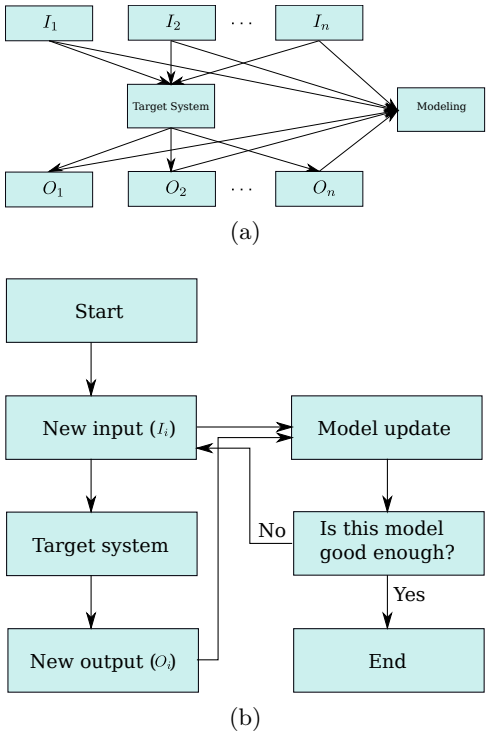



Figure 2.5: The diagram showing the two approaches ((a): offline; (b): online) for system identification. $(I_i, O_i)$, where $i \in [1, 2, \cdots, n]$, represents a pair of input data and output data.

Genetic programming provides an alternative way for finding the structure and parameters of the system. The model represented by genetic programming is described in a tree-based structure. As the structure in genetic programming is evolving, the obtained model may have different structures in different runs [**?** ]. Bloating [29] 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 [**?** ], [**?** **?** **?** 15**?** **?** ]. A range of work have been performed on simulated agents. In [**?** ], 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 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 [**?** ], 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 [**?** ]. In [**?** ], an algorithm which 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 [**?** ] and control quality of a given task). Models were then refined through comparing their prediction to each selected test trajectory. In [**?** ], the damage detection process is conducted by a coevolutionary algorithm to extract the maximum information from the system. In [**?** ], a coevolutionary algorithm is 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. In these works, predefined metrics are critical for evaluating the performance of models.

Many studies also investigated the implementation of evolution directly in physical environments, on either a single robot [**?** **?** **?** ] or multiple robots [**?** ]. In [**?** ], 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 on-board. 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 [**?** ], a distributed coevolutionary approach was presented to co-evolve 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 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 studying animal behavior, as the model/knowledge learned can be used to build a more intelligent machine. At the same time, building machine that mimics the 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 the study of animal behavior, which is the theme of this thesis.

### 2.3.1 Animal Behavior in Nature

Comparing with the behaviors exhibited by complex animals such as mammals, insect behavior arises a large interest both by biologists and roboticists. One reason is due to the simple neural system of the insects, which provides a good inspiration for engineering purpose. This simplicity also makes it easy 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 [**?** ]. 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 [**?** ]. 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 [**?** **?** ]) 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 [**?** ]. 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 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 [**?** ]. 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 flocking behavior of birds are of particular interest to humans. This is not only because of their beautiful shape formed in a group, but also how the birds coordinate with each other to maintain that shape. A simple mathematical model was proposed by [**?** ] to describe the individual behavior of each bird in a flocking. 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 very closely. There are many other swarm behaviors which are also studied extensively such as the aggregation of cockroaches [**?** ], foraging in ants [**?** ], flashing synchronization in fireflies [**?** ], mound building in termites [**?** ].

### 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 [**?** ] and particle swarm optimization algorithm [**?** ]. Swarm robotics which uses the swarm behavioral rules of social insects as an design principles 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) is an optimization technique that gets inspiration from foraging behavior of ants. When ants go out to search for food, they will leave pheromone in the path. The other ants will be attracted by the pheromone, the strength of which represents the quality of the food source. Researchers have found that this indirection communication, which is know as *stigmergy* [**?** ], leads them to found the shortest path along their nest and location of food source. The initial application of ACO is to find the optimal path in the combinational (discrete) problem. Note that nowadays the application of ACO algorithms ranges from network optimization (e.g., routing and load balance [**?** ]) to continuous optimization [**?** ].

The principle of ACO algorithms can be divided into the following two steps:

- Use the pheromone model to generate condidate solutions. From the aspect of mathematics, the pheromone model is a parameterized probability distribution in the search space.
- The candidate solutions are used as a bias for the future sampling to get better solutions.

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

**Particle Swarm Optimization**

Particle Swarm Optimization (PSO) is another an optimization algorithm which gets inspiration from flocking of birds or schooling of fish. It was first proposed by Kennedy and Eberhart [**?** ]. 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 (it has found so far) and its current position, and deviation between the best position by its neighbors and its current position. This will result 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:

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

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

where $\overrightarrow{R_1}$ 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 sections, we talk about how swarm behaviors can be used as inspiration for algorithms design. In this section, we introduce how to use swarm intelligence techniques to multiple robots research, which is referred to as swarm robotics. Many social insects' (e.g, ants, termites, wraps and bees) behavior can be used as inspirations for swarm robotics.

Swarm robotics investigates how multiple robots each with limited ability communicate, coordinate and self-organize to accomplish complex tasks. To finish the same task, using a single expensive robot with complex control structure may be feasible but it may have low efficiency and prone to failure. The advantages of swarm robotics are as follows:

- *Robustness* The robustness of a swarm robotic system can be explained in the following reasons: 1) if some robots failed, the other robots would replace the functions of the failed robots; 2) the control is distributed; 3) as the individual is simple, it is less likely to be damaged; 4) the perception from multiple robots would increase the system's robustness. Note that in some swarm robotic systems, there may be exceptions. That is, some individuals' failure would influence the whole self-organized process [**?** ].

- *Scalability* In swarm robotics, the number of robots do not have significant difference on the global performance/behavior in the system, unless the number is sufficiently small. That is, increasing/decreasing certain number of robots, the system is still under control and the coordination is maintained. When investigating the performance of a swarm robotic system, scalability study is usually considered [**?** **?** ].

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

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

- *interaction via environment* In this interaction method, the robots communicate with each through changing the environment. There is no explicit communication between each robot (i.e., they do not exchange message). In nature, the pheromone ants leave when foraging is an environmental stimulus for locating the 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 each robot. This requires the robots can distinguish between robots and objects in the environment. In nature, when ants need to collectively pull the food to the nest, they need to perceive each other (to avoid collision) and the food (object).

- *interaction via explicit communication* In this method, a network is required to communicate with a swarm of robots in real time. This could be done by broadcast (e.g., WiFi [**?** ]) or a distributed sensing network [**?** ]. How to build a reliable network when the number of robots is significantly large is still a hot 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 [? 21? ? ], dispersion [? ? ], pattern formation [? ? ], collective movement [? ] to cooperative transport [? ? ? ? ], etc. Aggregation can be considered as the fundamental behavior of other more complex tasks. In [? ], a group of robots mimic the cockroaches' aggregation behaviors, in which the robots gather into or leave the nest with a probability proportional to the size of the nest. The advantage of using stochastic algorithm is that they do not need to form a connected network in the initial configuration. 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 do not need to perform algorithmic computation. This work was scaled well using 1000 robots in simulation. In [? ], 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 [? ], 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 [? ]. In cooperative transport, Chen et al. [? ] proposed a strategy in which the robots only push the object when the robots' vision of the goal is occluded by the object. This strategy was proved to push any convex object in a plenary environment.

### 2.3.3 Contribution of AI/Robotics to Ethology

In the previous sections, we reviewed how animal behavior study can be used as inspiration for designing algorithms and robotic systems. However, robotics can also benefit animal behavior study. In this section, we review two approaches in AI/robotics to contribute to the study of animal behavior: *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 [? ]. The first step is observation, and after that they formulate some scientific questions on the observed behavior, and

generate hypothesis to answer these questions. In order to verify the hypothesis, they actively 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 [**?** 6]. For example, taxis behavior has often been implemented on mobile robotic systems for investigating steering and navigation [**?** ]. In [**?** ], Webb used a robot to model the phonotaxis behavior of crickets [**?** ]. 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 trial has been used as a model for robots to find odour source based on artificial neural networks [**?** ] and even Braitenberg vehicles [**?** ]. 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. [**?** ] built executable models of the behaviors and 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 [**?** ], 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 the many specific kinds of contributions that AI can make to biological study. They also give some suggestions on how AI and robotics researchers collaborate with biologists to combine the advantage of each other and 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 and conduct further experiments.

#### 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 programme robots (i.e., replicas) that can be inserted into the group of social animals [**?** **?** **?** **?** **?** ]. 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 [**?** ]. In this case, one "animal" in a group is completely controlled and they can observe the behaviors of the mixed society [**?** ]. The behavior of the inserted robot can be controlled and the model can also be embedded into the robot for verification [**?** ]. 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 easier to verify a hypothesis through controlled interaction in social behaviors.

In [**?** ], 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 [**?** ], autonomous robots which executed the 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 [**?** **?** ], Vaughan et al. have built a mobile robot that can interact with the ducks in a circular arena and drive them to the safe place. Halloy et al. In [**?** ], Gribovskiy et al. designed a robot which is capable of interacting with chicks to study how the behavior of chicks can be influenced by the others in a group. In [**?** ], a robot-fish that can interact intelligently with live zebrafish to study their preference and locomotion behavior was designed. Although robots which are well designed can be mixed in social animals, building such kind of robot is a time-consuming process. It is also expensive to some extent and requires the collaboration of researchers from different disciplines. In these 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 proposed system identification method, which autonomously infers the collective behavior.

# 3 Reverse Engineering Swarm Behaviors Through Turing Learning

As mentioned in Chapter 1, swarm behaviors are emergent behaviors that arise from the interactions of a number of simple individuals in a group [17]. To understand swarm behaviors, researchers in artificial intelligence and swarm robotics use simulated agents or physical robots to mimic the swarm behaviors such as foraging, aggregation and flocking observed in nature. This approach is what we refer to as understanding from synthesis, and it has been shown to provide a deep insight to validate the models of swarm behaviors or generate new hypothesis [**?** ]. The design of the controllers for the agents and robots comply with some common principles. For example, the controllers are distributed, that is, there is no central control for the swarm. The structure of the controllers is relatively simple.

In this chapter, we demonstrate how the proposed system identification (coevolutionary) method allows a machine to infer the behavioral rules of a group of homogeneous agents in an autonomous manner. A replica, which resembles the agents under investigation in terms of behavioral capabilities, is mixed into the group. The coevolutionary algorithm consists of two competitive populations: one of *models*, to be executed on the replica, and the other of *classifiers*. The classifiers observe the motion of an individual[1] in the swarm for a fixed time interval. They are not, however, provided with the individual's sensory information. Based on the individual's motion data, a classifier outputs a Boolean value indicating whether the individual is believed to be an agent or replica. The classifier gets a reward if and only if it makes the correct judgment. The fitness of the classifiers thus depends solely on their ability to discriminate between the agents and the replica. Conversely, the fitness of the models depends solely on their ability to 'trick' the classifiers into categorizing them as an agent. Consequently, our method does

[1]Note that 'individual/robot' in the context of swarm behaviors refers to a single unit. It could refer to an agent under investigation or the replica executing a model. However, in the context of evolutionary algorithms (see Section 3.1.1.3), 'individual' refers to a chromosome.

not rely on predefined metrics for measuring the similarity of behavior between models and agents; rather, the metrics (classifiers) are produced automatically in the learning process. Our method is inspired by the Turing test [**?** **?** ], which machines can pass if behaving indistinguishably from humans. Similarly, the models, which evolve, can pass the tests by the coevolving classifiers if behaving indistinguishably from the agents. We hence call our method *Turing Learning*.

To validate the *Turing Learning* method, we present two case studies on canonical problems in swarm robotics: self-organized aggregation [21] and object clustering [22]. We show that observing individual motion is sufficient to guide the learning of these collective behaviors. In this chapter, we only present the simulation results (a real world validation using physical robots based on one of the case studies will be presented in Chapter 4).

This chapter is organized as follows. Section 3.1 describes the implementation of *Turing Learning* and the two swarm behaviors under investigation. Section 3.2 introduces the simulation platform and setups for performing coevolution runs. Section 3.3 presents the results obtained from the two case studies. In particular, Section 3.3.1 analyzes the evolution of models, objectively measuring their quality in terms of local and global behaviors. Section 3.3.2 investigates the coevolutionary dynamics. Section 3.3.3 investigates the evolution of classifiers, showing how to construct a robust classifier system to potentially detect abnormal behaviors in the swarm. Section 3.3.4 studies the effect of observing only a subset of agents in the swarm. Section 3.3.5 presents a study where an aspect of the agents' morphology (their field of view) and brain (controller) are inferred simultaneously. Section 3.3.6 shows the results of using *Turing Learning* to learn other swarm behaviors.. Section 3.3.7 presents a study showing the method's sensitivity to noise. Section 3.4 summaries the findings in this chapter.

## 3.1 Methodology

In this section, we present the *Turing Learning* method and two case studies: self-organized aggregation [21] and object clustering [22]. In both case studies, individuals execute simple behavioral rules that lead to meaningful emergent behaviors on a global level.

### 3.1.1 Turing Learning

*Turing Learning* uses a coevolutionary algorithm that comprises two populations: one of models, and one of classifiers. These populations coevolve competitively. In the following, we describe the models, classifiers, optimization algorithm, fitness calculation method and termination criterion.

#### 3.1.1.1 Models

We assume that one or more replicas, which have actuation and sensing abilities that are equivalent to those of the agents under investigation, are available. In this thesis, the replica(s) will be mixed into a group of homogeneous agents. They should therefore be perceived by the agents as con-specifics [**?** ].

Models are executed on replicas. The models can be represented explicitly (e.g., parameters) or implicitly (e.g., artificial neural networks).

#### 3.1.1.2 Classifiers

The classifier can be any algorithm that takes a sequence of data about the observed individual as input, and outputs a decision (i.e., whether the observed individual is believed to be an agent or replica).

We represent the classifier as a recurrent Elman neural network [27], which is shown in Figure 3.1. The network has $i$ inputs, $h$ hidden neurons and one output neuron. Each neuron of the hidden and output layers has a bias. The network has a total of $(i+1)h+h^2+(h+1)$ parameters, which all assume values in $\mathbb{R}$. The activation function used in the hidden and the output neurons is the logistic sigmoid function, which has the range $(0,1)$ and is defined as:

$$\operatorname{sig} x = \frac{1}{1+e^{-x}} \quad \forall x \in \mathbb{R}. \tag{3.1}$$

The classifier makes judgments about all individuals—both agents and replica. It observes each individual over a set period of time (i.e., a trial). The final value of its output neuron, $O$, is used to make the judgment: the classifier judges the individual to be a replica if $O < 0.5$, and an agent otherwise. The classifier's memory (value of hidden neurons) is reset after each judgment.

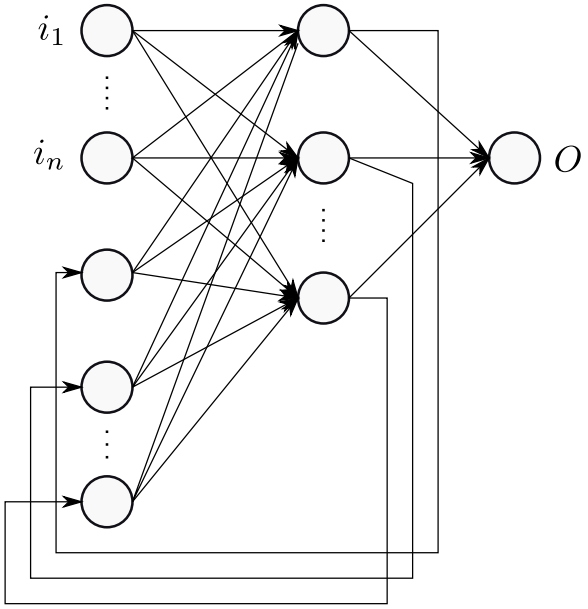



Figure 3.1: The plot shows the structure of a classifier, which is a recurrent Elman neural network, with $i$ inputs, $h$ hidden neurons, and one output neuron ($O$). $O$ is used for making a judgment. Two bias neurons (which are not shown) with a constant input of 1.0 are connected to each neuron of the hidden and output layers. See text for details.

#### 3.1.1.3 Optimization Algorithm

The optimization of models and classifiers is realized using evolutionary algorithms. In this thesis, we use a $(\mu+\lambda)$ evolution strategy with self-adaptive mutation strengths [28, 29] to optimize either population. The optimization algorithm can be thought of consisting of two sub-algorithms: one for the model population, and another for the classifier population. The sub-algorithms do not interact with each other except for the fitness calculation step (described in Section 3.1.1.4). In each sub-algorithm, the value of $\mu$ and $\lambda$ is equal, which is half of the population size. The implementation of the evolutionary algorithm is detailed as follows.

In this algorithm, an individual is a 2-tuple, $\mathbf{a} = (\mathbf{x}, \boldsymbol{\sigma})$, where $\mathbf{x} \in \mathbb{R}^n$ represents objective parameters, and $\boldsymbol{\sigma} \in (0, \infty)^n$ represents mutation strengths. The $i$-th mutation strength in $\boldsymbol{\sigma}$ corresponds to the $i$-th element in $\mathbf{x}$.

Each generation $g$ comprises a population of $\mu$ individuals:

$$\mathcal{P}^{(g)} = \left\{ \mathbf{a}_1^{(g)}, \mathbf{a}_2^{(g)}, \ldots, \mathbf{a}_\mu^{(g)} \right\}.$$

In the population of the first generation, $\mathcal{P}^{(0)}$, all the objective parameters are initialized to 0.0 and all the mutation strengths are initialized to 1.0. Thereafter, in every generation $g$, the $\mu$ parent individuals are first used to create $\lambda$ offspring individuals by recombination. For the generation of each recombined individual $\mathbf{a}_k'^{(g)}$, $k \in \{1, 2, \ldots, \lambda\}$, two individuals are chosen randomly, with replacement, from the parent population: $\mathbf{a}_\chi^{(g)}$ and $\mathbf{a}_\psi^{(g)}$, where $\chi, \psi \in \{1, 2, \ldots, \mu\}$. The intermediate population, $\mathcal{P}'^{(g)}$, is produced using discrete and intermediate recombination, which generates the objective parameters and the mutation strengths of the recombined individual, respectively:

$$x_{k,i}'^{(g)} = x_{\chi,i}^{(g)} \quad \text{OR} \quad x_{\psi,i}^{(g)}, \tag{3.2}$$

$$\sigma_{k,i}'^{(g)} = \left( \sigma_{\chi,i}^{(g)} + \sigma_{\psi,i}^{(g)} \right) / 2, \tag{3.3}$$

where $i \in \{1, 2, \ldots, n\}$ is indexing the elements within the vectors and, in Equation 3.2, the selection is performed randomly and with equal probability.

Each of the $\lambda$ recombined individuals is then mutated in order to obtain the final offspring population, $\mathcal{P}''^{(g)}$. This is done according to:

$$\sigma_{k,i}''^{(g)} = \sigma_{k,i}'^{(g)} \exp\left( \tau' \mathcal{N}_k(0,1) + \tau \mathcal{N}_{k,i}(0,1) \right), \tag{3.4}$$

$$x_{k,i}''^{(g)} = x_{k,i}'^{(g)} + \sigma_{k,i}''^{(g)} \mathcal{N}_{k,i}(0,1), \tag{3.5}$$

for all $\{k, i\}$, where $k \in \{1, 2, \ldots, \lambda\}$ is indexing the individuals within the population and $i \in \{1, 2, \ldots, n\}$ is indexing the elements within the vectors. Equation 3.4 generates the perturbed mutation strength from the original one according to a log-normal distribution. Equation 3.5 mutates the objective parameter according to a normal distribution having the perturbed mutation strength as its deviation. In Equation 3.4, $\mathcal{N}_k(0,1)$ and $\mathcal{N}_{k,i}(0,1)$ are both random numbers generated from a standard normal distribution; however, the former is generated once for each individual (i.e. for each value of $k$), while the latter is generated separately for each element within each individual (i.e. for each combination of $k$ and $i$). The parameters $\tau'$ and $\tau$ determine the learning rates of the mutation strengths, and are set as $\tau' = 1/2\sqrt{2n}$, $\tau = 1/2\sqrt{2\sqrt{n}}$ (similar to [**?** ]), where $n$ corresponds to the population size.

Once the offspring population has been generated, the $\mu$ individuals with the highest fitness from the combined population, $\mathcal{P}^{(g)} \cup \mathcal{P}''^{(g)}$ (which contains $\mu+\lambda$ individuals), are selected as the parents to form the population of the next generation, $\mathcal{P}^{(g+1)}$. Individuals with an equal fitness have an equal chance of being selected.

#### 3.1.1.4 Fitness Calculation

Let the population sizes for the models and classifiers in the coevolution be $M$ and $C$, respectively. Let the number of replicas and agents in a trial be $n_r$ and $n_a$, respectively. $n_t$ trials are conducted for a model in each generation; throughout this thesis, we assume $n_t = 1$.

In our thesis, whenever multiple replicas are used, each of them executes a different model. The fitness of each model in a trial is determined by each of the $C$ classifiers in the competing population. For every classifier that wrongly judges the model as an agent, the model's fitness increases by 1. After all evaluations, the model's fitness takes a value in $\{0, 1, 2, \ldots, C\}$. This value is then normalized to $[0, 1]$.

The fitness of each classifier in a trial is determined by its judgments for the $n_r$ replicas (each executing a different model) and $n_a$ agents. For each correct judgment of the replica, the classifier's fitness increases by $\frac{1}{2n_r}$; for each correct judgment of the agent, the classifier's fitness increases by $\frac{1}{2n_a}$. Therefore, the fitness of each classifier in a trial is in range $[0, 1]$. $\left\lceil \frac{M}{n_r} \right\rceil$ trials[2] are conducted in each generation, and the fitness value of each classifier is then normalized to $[0, 1]$.

#### 3.1.1.5 Termination Criterion

The coevolutionary algorithm stops after running for a fixed number of generations.

### 3.1.2 Case Studies

#### 3.1.2.1 Problem Formulation

The agents used in our case studies are embodied and move in a two-dimensional, continuous space. The agents' embodiment is based on the e-puck [**?** ], which is a miniature,

[2] We suggest choosing $n_r$ to be a factor of $M$.

differential-wheeled robot. Figure 3.2 shows an e-puck robot used in the physical experiments in Chapter 4.

Each agent is equipped with a line-of-sight sensor that it can use to detect the item (e.g., the background, other agents or objects [21], [22]) in front of it.

The swarm behaviors investigated in this thesis use a reactive control architecture, as found in many biological systems[3]. The motion of each agent solely depends on the state of its line-of-sight sensor ($I$). Each possible sensor state, $I \in \{0, 1, \cdots, n-1\}$, is mapped onto a pair of predefined velocities for the left and right wheels, $(v_{\ell I}, v_{rI})$. $v_{\ell I}, v_{rI} \in [-1, 1]$ represent the normalized left and right wheel velocities, respectively, where 1 $(-1)$ corresponds to the wheel rotating forwards (backwards) with maximum velocity. Given $n$ sensor states, any reactive behavior can thus be represented using $2n$ system parameters. In the remainder of this thesis, we describe the corresponding controllers by writing the $2n$ parameters as a tuple in the following order:

$$\mathbf{p} = (v_{\ell 0}, v_{r0}, v_{\ell 1}, v_{r1}, \cdots, v_{\ell(n-1)}, v_{r(n-1)}). \tag{3.6}$$

We assume that the replica has the same differential drive and line-of-sight sensor[4] as the agents. The system identification task is thus to infer the control parameters in Equation (3.6). This explicit representation makes it possible for us to objectively measure the quality of the obtained models in the post-evaluation analysis (as discussed in the results section). To make the evolution more challenging, the search space for the algorithm in simulation is unbounded. That is, the model parameters are unconstrained, and the replica can move with arbitrary speed.

The classifier does not have any prior knowledge about the individual under investigation. It is fed with the sequence of motion data of the individual. It has two input neurons $(i = 2)$, five hidden neurons $(h = 5)$ and one output neuron. The input neurons represent the linear speed $(v)$ and angular speed $(\omega)$ of the individual. They are obtained by tracking the positions and orientations of individuals. In simulation, the tracking is noise-free (situations with noise being present are considered in Section 3.3.7 and the physical experiments in Chapter 4 where noise is inherently present). We define the

[3] For example, researchers have found that the complex auditory orientation behavior of female crickets is derived from simple reactive motor responses to specific sound pulses [**?** ].

[4] In Section 3.3.5, we show that this assumption can be relaxed by also evolving some aspect of the agent's morphology.

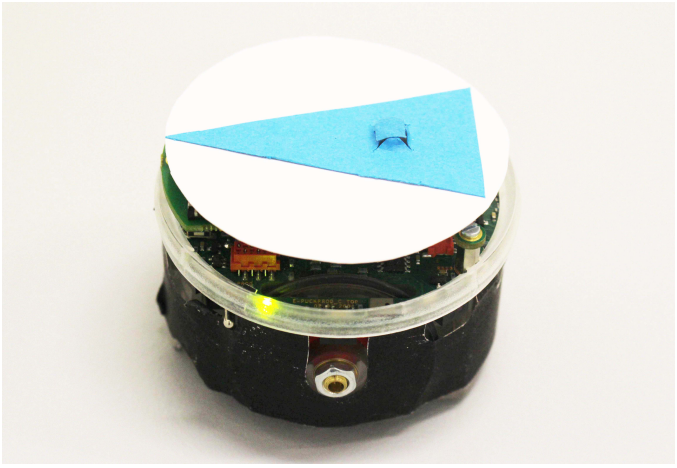


Figure 3.2: An e-puck robot fitted with a black 'skirt' and a top marker for motion tracking.

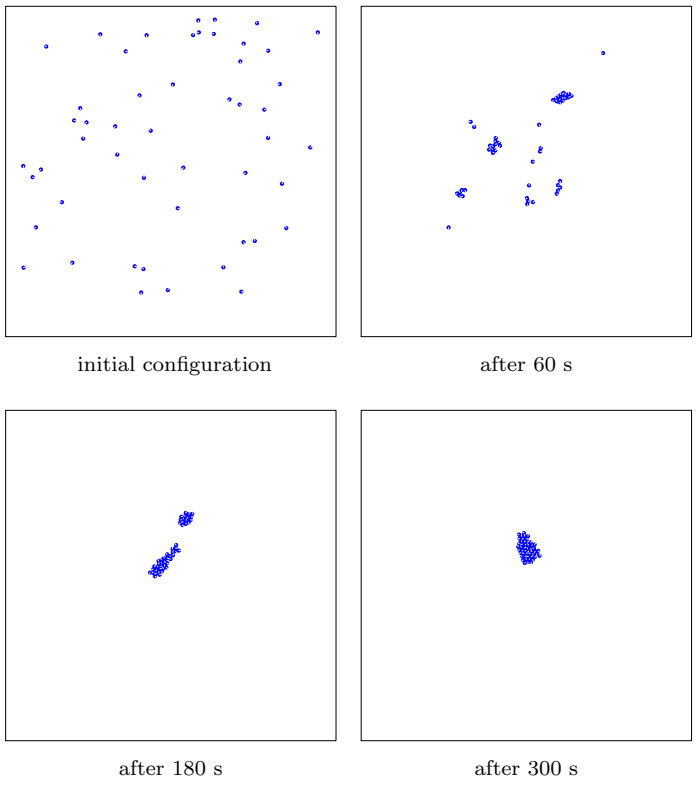
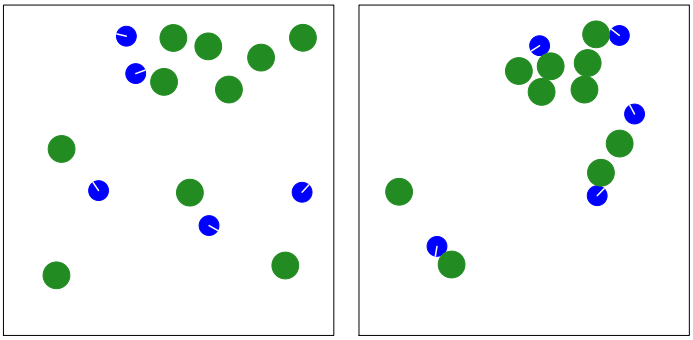
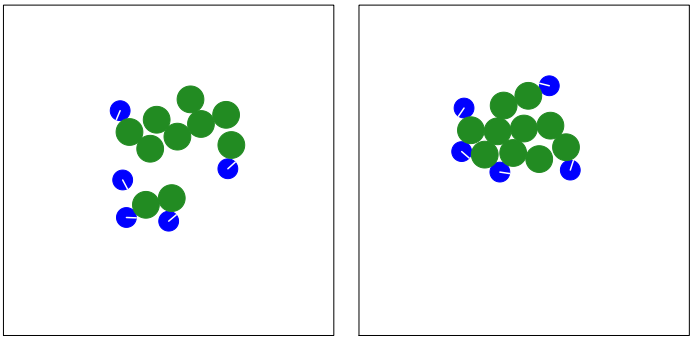



Figure 3.3: Snapshots of the aggregation behavior of 50 agents in simulation.



initial configuration after 20 s



after 40 s after 60 s

Figure 3.4: Snapshots of the object clustering behavior in simulation. There are 5 agents (blue) and 10 objects (green).

linear speed to be positive when the angle between the individual's orientation and its direction of motion is smaller than $\pi/2$ rad, and negative otherwise.

In the following, we detail the behavioral rules of the two swarm behaviors.

#### 3.1.2.2 Aggregation

In this behavior, the sensor is binary, that is, $n = 2$. It gives a reading of $I = 1$ if there is an agent in the line of sight, and $I = 0$ otherwise. The environment is free of obstacles. The objective for the agents is to aggregate into a single compact cluster as fast as possible. Further details, including a validation with 40 physical e-puck robots, are reported in [21].

The 'optimal' controller for aggregation was found by performing a grid search over the entire space of possible controllers (with finite resolution) [21]. The 'optimal' controller's

parameters are:

$$\mathbf{p} = (-0.7, -1.0, 1.0, -1.0)\,. \tag{3.7}$$

When $I = 0$, an agent moves backwards along a clockwise circular trajectory ($v_{\ell 0} = -0.7$ and $v_{r0} = -1.0$). When $I = 1$, an agent rotates clockwise on the spot with maximum angular velocity ($v_{\ell 1} = 1.0$ and $v_{r1} = -1.0$). Note that rather counter-intuitively, an agent never moves forward, regardless of $I$. With this controller, an agent provably aggregates with another agent or a quasi-static cluster of agents [21]. Figure 3.3 shows snapshots from a simulation trial with 50 agents.

#### 3.1.2.3 Object Clustering

This behavior uses $n = 3$ sensor states: $I = 0$ if the sensor is pointing at the background (e.g., the wall of the environment, if the latter is bounded), $I = 1$ if the sensor is pointing at an object, and $I = 2$ if it is pointing at another agent. The objective of the agents is to arrange the objects into a single compact cluster as fast as possible. Details of this behavior, including a validation using 5 physical e-puck robots and 20 cylindrical objects, are presented in [22].

The controller's parameters, found using an evolutionary algorithm [22], are:

$$\mathbf{p} = (0.5, 1.0, 1.0, 0.5, 0.1, 0.5)\,. \tag{3.8}$$

When $I = 0$ and $I = 2$, the agent moves forward along an anti-clockwise circular trajectory, but with different linear and angular speeds. When $I = 1$, it moves forward along a clockwise circular trajectory. Figure 3.4 shows snapshots from a simulation trial with 5 agents and 10 objects.

## 3.2 Simulation Platform and Setups

In this section, we present the agent platform for simulating the two swarm behaviors under investigation and the simulation setups for performing coevolution runs.

### 3.2.1 Simulation Platform

We use the open-source Enki library [**?** ], which models the kinematics and dynamics of rigid objects, and handles collisions. Enki has a built-in 2-D model of the e-puck. The robot is represented as a disk of diameter 7.0 cm and mass 150 g. The inter-wheel distance is 5.1 cm. The speed of each wheel can be set independently. Enki induces noise on each wheel speed by multiplying the set value by a number in the range $(0.95, 1.05)$ chosen randomly with uniform distribution. The maximum speed of the e-puck is 12.8 cm/s, forward or backward. The line-of-sight sensor is simulated by casting a ray from the e-puck's front and checking the first item with which it intersects (if any). The range of this sensor is unlimited in simulation.

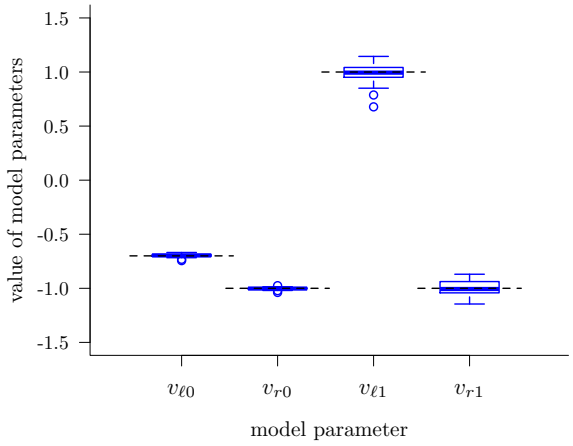
In the object clustering case study, we model objects as disks of diameter 10 cm with mass 35 g and a coefficient of static friction with the ground of 0.58, which makes it movable by a single e-puck.

The robot's control cycle is updated every 0.1 s, and the physics is updated every 0.01 s.

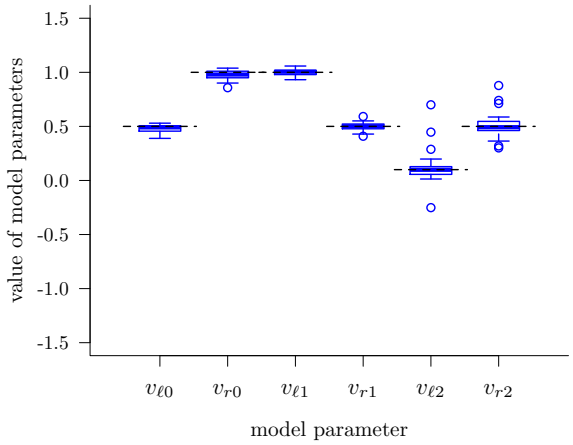
### 3.2.2 Simulation Setups

In all simulations, we used an unbounded environment. For the aggregation case study, we used groups of 11 individuals—10 agents and 1 replica that executes a model. The initial positions of individuals were generated randomly in a square region of sides 331.66 cm, following a uniform distribution (average area per individual = 10000 $\text{cm}^2$). For the object clustering case study, we used groups of 5 individuals—4 agents and 1 replica that executes a model—and 10 cylindrical objects. The initial positions of individuals and objects were generated randomly in a square region of sides 100 cm, following a uniform distribution (average area per object = 1000 $\text{cm}^2$). In both case studies, individual starting orientations were chosen randomly in $[-\pi, \pi]$ with uniform distribution.

We performed 30 coevolution runs for each case study. Each run lasted 1000 generations. The model and classifier populations each consisted of 100 solutions ($\mu = 50$, $\lambda = 50$). In each trial, classifiers observed individuals for 10 s at 0.1 s intervals (100 data points).




(a) Aggregation




(b) Object Clustering

Figure 3.5: Parameters of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation in the coevolutions for (a) the aggregation behavior and (b) the object clustering behavior. Each box corresponds to 30 coevolution runs in simulation. The dotted black lines correspond to the values of the parameters that the system is expected to learn (i.e., those of the agent).

## 3.3 Simulation Results

### 3.3.1 Analysis of Evolved Models

In order to objectively measure the quality of the models obtained through *Turing Learning*, we define two metrics. Given an evolved controller (model) $\mathbf{x}$ and the original agent

controller $\mathbf{p}$, where $\mathbf{x}, \mathbf{p} \in [-1, 1]^{2n}$, we define the absolute error (AE) in a particular parameter $i \in \{1, 2, \ldots, 2n\}$ as:

$$\mathrm{AE}_i = |x_i - p_i|. \tag{3.9}$$

We define the mean absolute error (MAE) over all parameters as:

$$\mathrm{MAE} = \frac{1}{2n} \sum_{i=1}^{2n} \mathrm{AE}_i. \tag{3.10}$$

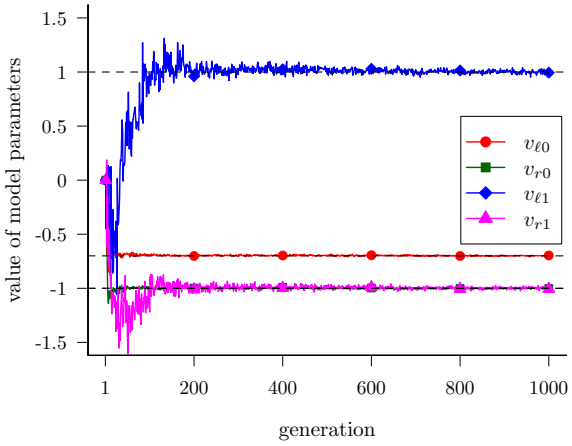
Fig. 3.5 shows a box plot[5] with the parameters of the evolved models with the highest subjective fitness[6] in the final generation. It can be seen that *Turing Learning* identified the parameters for both behaviors with good accuracy (dotted black lines represent the ground truth, that is, the parameters of the observed swarming agents). In the case of aggregation, the means (standard deviations) of the AEs in the parameters were (from left to right in Fig. 3.5(a)): 0.01 (0.01), 0.01 (0.01), 0.07 (0.07) and 0.06 (0.04). In the case of object clustering, these values were: 0.03 (0.03), 0.04 (0.03), 0.02 (0.02), 0.03 (0.03), 0.08 (0.13) and 0.08 (0.09).

We also investigated the evolutionary dynamics. Fig. 3.6 shows how the model parameters converged over generations. In the aggregation case study, the parameters corresponding to $I = 0$ were learned first. After around 50 generations, both $v_{\ell 0}$ and $v_{r0}$ closely approximated their true values ($-0.7$ and $-1.0$), shown in Fig. 3.6(a). For $I = 1$, it took about 200 generations for both $v_{\ell 1}$ and $v_{r1}$ to converge. A likely reason for this effect is that an agent spends a larger proportion of its time seeing nothing ($I = 0$) than other agents ($I = 1$)—simulations revealed these percentages to be 91.2% and 8.8%, respectively (mean values across 100 trials).

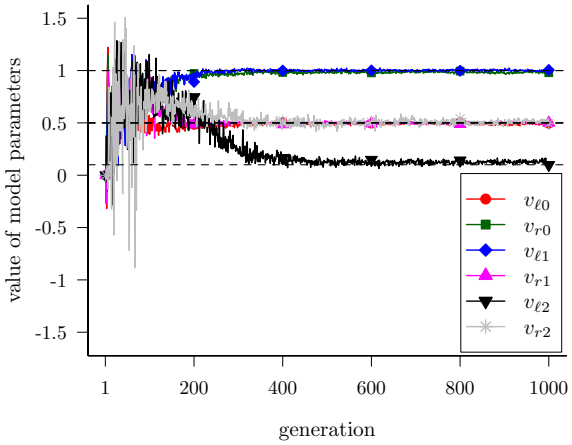
In the object clustering case study, the parameters corresponding to $I = 0$ and $I = 1$ were learned faster than the parameters corresponding to $I = 2$, as shown in Fig. 3.6(b). After about 200 generations, $v_{\ell 0}$, $v_{r0}$, $v_{\ell 1}$ and $v_{r1}$ started to converge; however it took about 400 generations for $v_{\ell 2}$ and $v_{r2}$ to approximate their true values. Note that an

[5] The box plots presented here are all as follows. The line inside the box represents the median of the data. The edges of the box represent the lower and the upper quartiles of the data, whereas the whiskers represent the lowest and the highest data points that are within 1.5 times the range from the lower and the upper quartiles, respectively. Circles represent outliers.

[6] The fitness of the models depends solely on the judgments of the classifiers from the competing population, and is hence referred to as *subjective*.




(a) Aggregation

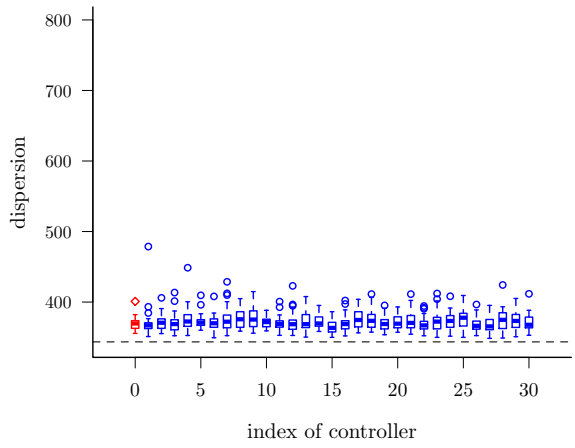



(b) Object Clustering

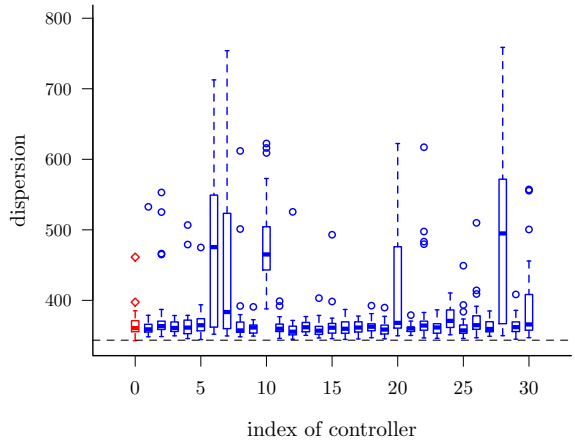
Figure 3.6: Evolutionary process of the evolved model parameters for (a) the aggregation behavior and (b) the object clustering behavior. Curves represent median values across 30 coevolution runs. Dotted black lines indicate true values.

agent spends the highest proportion of its time seeing nothing ($I = 0$), followed by objects ($I = 1$) and other agents ($I = 2$)—simulations revealed these proportions to be 53.2%, 34.2% and 12.6%, respectively (mean values across 100 trials).

Although the evolved models approximate the agents well in terms of parameters, it has often been observed in swarm systems that small changes in individual agent behaviors can lead to vastly different emergent behaviors, especially with large numbers of agents [**?** ]. For this reason, we evaluated the quality of the emergent behaviors that the models give rise to. In the case of aggregation, a good measure of the emergent behavior is




(a) Aggregation




(b) Object Clustering

Figure 3.7: (a) Dispersion (after 400 s) of 50 agents executing the original aggregation controller (red box) or one of the 30 evolved models (blue boxes) of the 1000$^{\text{th}}$ generation. (b) Dispersion (after 400 s) of 50 objects in a swarm of 25 agents executing the original object clustering controller (red box) or one of the 30 evolved models (blue boxes). In both (a) and (b), boxes show distributions over 30 trials. The dotted black lines indicate the minimum dispersion that 50 agents/objects can possible achieve [**?** ]. See Section 3.3.1 for details.

the dispersion of the swarm after some elapsed time as defined in [21][7]. For each of the 30 models with the highest subjective fitness in the final generation, we performed 30 trials with 50 agents each executing the model. For comparison, we also performed 30 trials using the original controller (see Equation (3.7)). The set of initial configurations

[7]The measure of dispersion is based on the robots'/objects' distances from their centroid. For a formal definition, see Equation (5) of [21], Equation (2) of [22] and [**?** ].

was the same for all models and the original controller. Fig 3.7(a) shows the dispersion for the original controller and models after 400 s. All models led to aggregation. We performed a statistical test[8] on the final dispersion of the agents between the original controller and each model. There was no statistically significant difference in 26 out of 30 cases (30 out of 30 cases with Bonferroni correction).

In the case of object clustering, we use the dispersion of the objects after some elapsed time as a measure of the emergent behavior. With the original controller (see Equation (3.8)) and each of the models, we performed 30 trials with 25 agents and 50 objects. The results are shown in Fig. 3.7(b). In a statistical test on the final dispersion of the objects between the original controller and each model, there was no statistically significant difference in 24 out of 30 cases (26 out of 30 cases with Bonferroni correction).

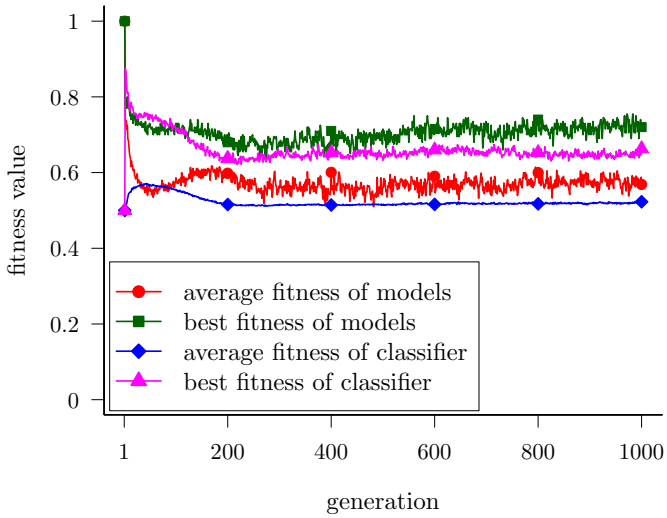
We also investigated the evolutionary process of the model parameters. Figure 3.6 shows the convergence of the model parameters over generations. In the aggregation behavior, the parameters corresponding to $I = 0$ were learned first. After about 50 generations, both $v_{\ell 0}$ and $v_{r0}$ closely approximated their true values ($-0.7$ and $-1.0$) shown in Figure 3.5(a). For $I = 1$, it took about 200 generations for both $v_{l1}$ and $v_{r1}$ to converge. A likely reason for this effect is that an agent spends a larger percentage of its time seeing nothing ($I = 0$) than other agents ($I = 1$)—simulations revealed these percentages to be 8.8% and 91.2%, respectively (mean values over 100 trials).

In the object clustering behavior, the parameters corresponding to $I = 0$ and $I = 1$ were learned faster than the other two parameters corresponding to $I = 2$, as shown in the Figure 3.6(b). After about 200 generations, $v_{\ell 0}$, $v_{r0}$, $v_{l1}$ and $v_{r1}$ started to converge; however it took about 400 generations for $v_{l2}$ and $v_{r2}$ to approximate their true values. This is likely because an agent spends the most percentage of its time seeing nothing ($I = 0$), followed by objects ($I = 1$) and other agents ($I = 2$)—simulations revealed these percentages to be 53.2%, 34.2% and 12.6% , respectively (mean values over 100 trials).

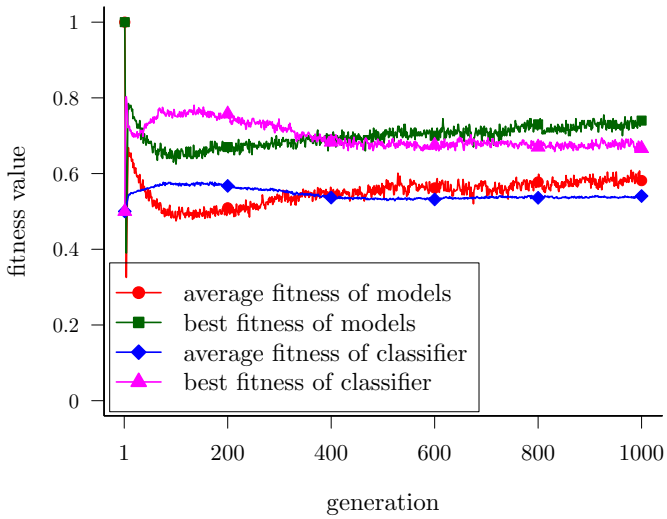
### 3.3.2 Coevolutionary Dynamics

In order to analyze how the classifiers and the models interact with each other during the course of the coevolution, we investigate the dynamics of the subjective fitness of

[8] Throughout this paper, the statistical test used is a two-sided Mann-Whitney test with a 5% significance level.




(a) Aggregation




(b) Object Clustering

Figure 3.8: This plot shows the subjective fitness of the classifiers and the models for (a) the aggregation behavior and (b) the object clustering behavior. The curves show the median value across 30 coevolution runs.

the classifiers and the models as shown in Figure 3.8.

In the aggregation behavior, at the beginning, the fitness of the classifiers is 0.5 as they output 1 for all the agents and models, which means the classifiers make uninformed decisions[9]. Therefore, the fitness of the models starts from 1.0, as all the classifiers

[9]Note that in the implementation of the coevolutionary algorithm, the parameters of the models and

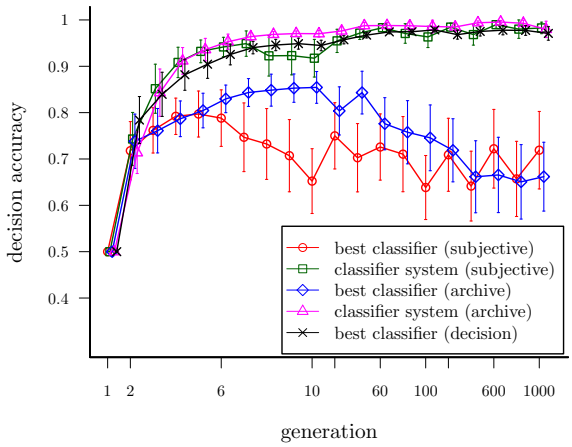



Figure 3.9: The average decision accuracy of the best classifiers and classifier systems over generations (nonlinear scale) in 30 coevolution runs. The error bars show standard deviations. See text for details.

judge them the agent. Then, the average fitness of the classifiers quickly increases, corresponding to the decline of the average fitness of the models. As the models learn to adapt, the average fitness of the classifiers only increases slightly until about the $50^{\text{th}}$ generation. After that, the average fitness of the models starts to increase. However, the best fitness of the classifiers is still higher than that of the models. The best fitness of the models surpasses that of the classifiers after about the $120^{\text{th}}$ generation; at this point, the fitness of the 'best' model (selected by the classifiers) is around 0.7. This means that the 'best' model is able to mislead 70% of the classifiers into judging it as the agent. From the $200^{\text{th}}$ generation onwards, the fitness of the classifiers and models remains "balanced" until the last generation.

The coevolutionary dynamics of the object clustering behavior is similar. Compared with the dynamics of the aggregation behavior, it takes more generations for the fitness of the models to surpass that of the classifiers. This could be explained by the higher number of parameters and the higher complexity of the behavior to be evolved in the object clustering behavior.

classifiers are initialized to 0.0, which means all the classifiers are identical at the $1^{\text{st}}$ generation.

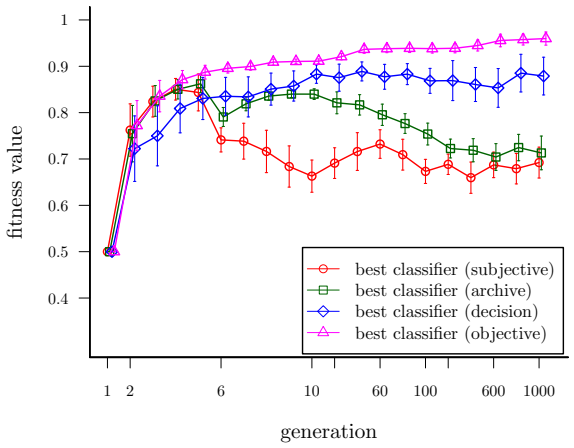
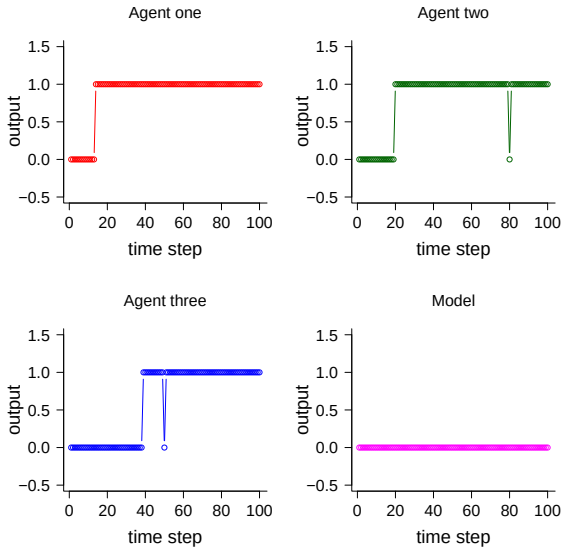



Figure 3.10: This plot shows the average fitness of the classifiers over generations (non-linear scale) in 30 coevolution runs. The fitness is calculated based on 14641 testing models. See text for details.

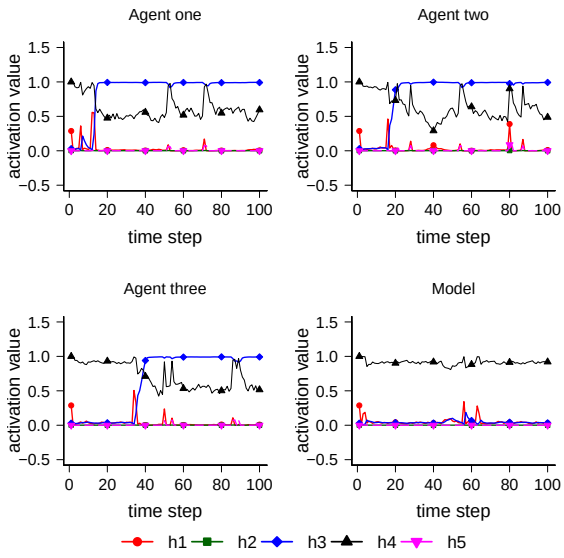
### 3.3.3 Analysis of Evolved Classifiers

The primary outcome of the *Turing Learning* method (and of any system identification method) is the model, which has been discussed in the previous section. However, the evolved classifiers can also be considered as a useful byproduct. For instance they could be used to detect abnormal agents in a swarm. We will now analyze the performance of the evolved classifiers. For the remainder of this paper, we consider only the aggregation case study.

To assess the performance of the classifiers, we measure the percentage of correct judgments over agents and a wide range of models. The models are uniformly distributed across the entire parameter space of the agents: $[-1, 1]^4$. To keep the analysis of classifiers within a reasonable computation time, we discretize this space using 11 settings per parameter, to obtain: $\mathcal{X} = \{-1.0, -0.8, ..., 0.8, 1.0\}^4$. This discretized space is a grid consisting of $|\mathcal{X}| = 11^4 = 14641$ points (i.e., models). The classifier's performance is computed as follows. The model is executed by a replica mixed into a group of 10 agents (as in the coevolution runs). 10 trials are performed using a set of initial configurations common to all classifiers. The motion data is fed to each classifier, which makes 10 judgments per individual. If the classifier consistently judges the individual as a model (i.e. not an agent) in 10 out of 10 trials, it outputs a "model" decision. Otherwise, it outputs "agent". This conservative approach was used to minimize the risk of false positive detection of abnormal behavior.




(a) decision-making process




(b) activation of hidden neurons

Figure 3.11: This plot shows the (a) decision-making process and (b) the corresponding activation value of the 5 hidden neurons of a classifier for three random-chosen agents and the replica that executes a very good model in a trial. Hidden neurons are labeled as h1, h2, h3, h4, and h5.

#### 3.3.3.1 Using a Single Classifier

The average decision accuracy of the classifier with the highest subjective fitness in 30 coevolution runs is shown in Fig. 3.9 (*best classifier (subjective)*). The accuracy combines the percentage of correct judgments about models (50% weight) with the percentage of correct judgments about agents (50% weight). The accuracy of the classifier increases in the first 5 generations, then drops and fluctuates within range 62%–80%.

An alternative strategy is to select the classifier that achieves the highest fitness when evaluated on the whole historical tracking data (not just those of the current generation). The decision accuracy of this classifier is also shown in Fig. 3.9 (*best classifier (archive)*). The trend is similar to that of *best classifier (subjective)*. The accuracy increases in the first 10 generations, and then starts *decaying*, dropping to around 65% by the $1000^{\text{th}}$ generation. However, in the earlier generations, the accuracy of the *best classifier (archive)* is higher than that of the *best classifier (subjective)*. For a comparison, we also plot the highest decision accuracy that a single classifier achieves for each generation (*best classifier (decision)*). Interestingly, the accuracy of the *best classifier (decision)*, which is shown in Fig. 3.9 (black curve), increases almost monotonically, reaching a level above 95%. Note that to select the *best classifier (decision)*, one needs to perform additional trials (146410 in this case).

At first sight, it is counter-intuitive that selecting the best classifier according to the historical data still leads to low decision accuracy. This phenomenon, however, can be explained when considering the model population. We have shown in the previous section (see especially Fig. 3.6(a)) that the models converge rapidly at the beginning of the coevolutions. As a result, when classifiers are evaluated in later generations, the trials are likely to include models very similar to each other. Classifiers that become overspecialized to this small set of models (the ones dominating the later generations) have a higher chance of being selected in the post-evaluation. These classifiers may however have a low performance when evaluated across the entire model space.

To investigate whether the decision accuracy is a good measurement of the quality of the classifiers, we plot a figure showing the fitness of the best classifiers corresponding to Figure 3.9. The fitness is calculated based on the 14641 models. In addition, we also plot the classifier with the highest fitness for comparison, which is termed as *best classifier (objective)*. The results are shown in Figure 3.10. As we can see, the trend of fitness curves show good correspondence to that of the decision accuracy in Figure 3.9,

although there is still a small gap between the *best classifier (decision)* and *best classifier (objective)*. This means measuring the quality of classifiers according to decision accuracy is a reasonable choice.

In order to understand how the classifiers make judgment, we analyze the internal processing through monitoring the activation of hidden neurons. Figure 3.11 shows the decision-making process and the corresponding activation value of the 5 hidden neurons of the classifier with the highest subjective fitness in the last generation of a coevolution run for 3 randomly-chosen agents and the replica in a trial. The model executed on the replica has a parameter set of $(-0.7, -1.0, 1.0, -0.9)$, which is very near to that of the agent in Equation (3.7). As we can see in Figure 3.11(a), for the agents, the classifier outputs 0 at the beginning, and then starts to output 1 after certain time steps, which means it needs some time to make the correct judgment. Note that for some time steps after it starts to output 1, it still outputs 0, but this happens only occasionally. For the model, it always outputs 0. This phenomenon can be explained by the activation value of the hidden neurons of the classifier in the following paragraph.

The classifier's activation value of the hidden neurons (labeled as h1, h2, h3, h4, and h5) shown in Figure 3.11(b) reveals its internal processing for the motion of different individuals[10]. It seems that the decision-making ("sudden jump" in Figure 3.11(a)) of the classifier is most likely related with the variation of h3 and h4. Take the agents for example, at the beginning, h3 has a very low value (which is almost 0) and then after certain time steps it starts to increase. The variation tendency of h4 is opposite. Every time the classifier's judgment starts to jump from 0 to 1, the value of h3 starts to surpass that of h4. h1, h2 and h5 are almost not activated during the whole period, although sometimes the activation of h2 influences the judgment of the classifier. For instance, there are two peaks at about $50^{\text{th}}$ and $80^{\text{th}}$ time step in h2 for agent two and three, which corresponds to a wrong judgment of the classifier (that is, it outputs 0) at the same point. For the model, all the hidden neurons are almost not activated.

#### 3.3.3.2 Using a Classifier System

The results of the previous section have shown that using a single classifier is not a good solution; although there may be a good classifier in each generation, it may take significant effort (trials with agents) to find it.

[10] Note that the activation of hidden neurons may not be the same for all the classifiers due to the symmetry of neural networks, but the dynamics of each hidden neuron is similar.

To address this problem, we propose the use of a classifier system, that is, a number of classifiers working in tandem to judge a given candidate. We choose the best 10 classifiers[11]. This is either the set of 10 classifiers with the highest subjective fitness in the current generation or the set of classifiers that achieve the highest fitness when evaluated on the whole historical data. If one or more classifiers make a decision about the candidate as a model (i.e. not an agent), the system outputs a "model" decision. Otherwise, it outputs "agent".

The results of using a classifier system are shown in Fig. 3.9 (green and magenta, respectively). The two systems exhibit significantly improved decision accuracy across all generations. After 1000 generations, each system has a high accuracy of above 95%, on average.

When post-evaluating the performance of classifier systems, we kept the setup the same as the one used in the coevolution runs (i.e., 10 agents and 1 replica in each trial). In practice, the ratio of agents and replicas in a trial could be different. Therefore, we analyzed the scalability of the classifier systems selected in the last generation over 30 coevolution runs with different number of agents in the group. Note that there is always one replica in the group. We have verified that with various number of replicas in the group, similar results are obtained. We tested 14641 different models. Figure 3.12 shows the decision accuracy of the *classifier system (archive)* when changing the number of agents in the group. As we can see, the decision accuracy of the classifier system is not affected by the variation of the number of agents, which shows the robustness of the classifier system.

As we mentioned before, we chose 10 classifiers to form a classifier system to have a reasonable decision accuracy. Here we investigate how the classifier systems perform with various number of classifiers chosen to form a system. The decision accuracy of the *classifier system (archive)* is shown in Figure 3.13[12]. It seems that as long as the number of classifiers chosen to form a system is within a certain range, the system could perform well. For example, when the number of selected classifiers is between 10 and 25,

[11] Note that the number of classifiers chosen to form the system here is not necessarily optimal. There is no guarantee that the individually best few classifiers will form the best *system* when working in tandem. In principle, one could exhaustively search every possible combination of a given number of classifiers from the population. However, this is often infeasible—with our settings of choosing 10 classifiers out of 100, $1.73 \times 10^{13}$ possibilities exist. We therefore propose the heuristic of choosing the individually best classifiers to form the system, and empirically show that this nevertheless yields good results.

[12] The result of *classifier system (subjective)* is similar.

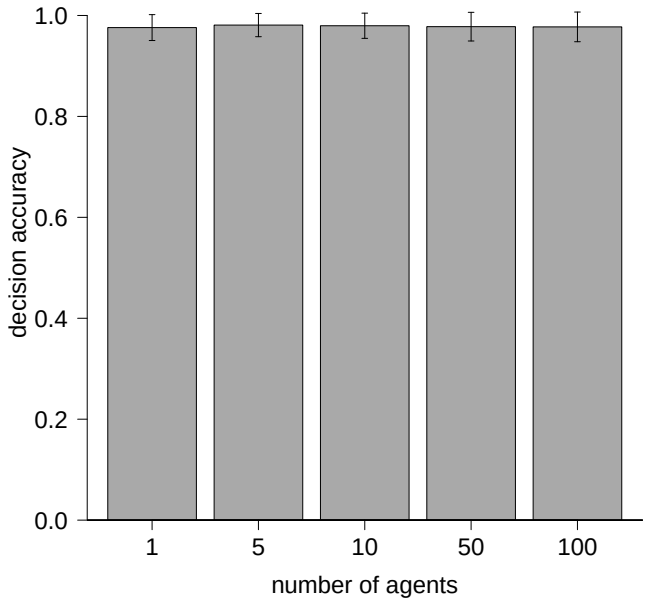



Figure 3.12: This plot shows the decision accuracy of the *classifier system (archive)* in the $1000^{\text{th}}$ generation of 30 coevolution runs for different numbers of agents in the group. In each case, 14641 different models were tested. A single replica was present in each trial.

the classifier system can obtain a high decision accuracy. However, when the number is high (e.g., 75), the decision accuracy declines dramatically.

### 3.3.4 Observing Only a Subset of Agents

So far, we have used motion data about all agents in the swarm when evaluating classifiers. However, this may not always be feasible in practice. For instance, given a video recording of a large and/or dense swarm, extracting motion data about all agents may be infeasible or lead to highly inaccurate results. A more practical solution might be to only track a subset of agents (e.g., by equipping them with markers).

We now compare the case where all agents are observed with the other extreme, where only one agent is observed. We study how these two cases compare as the swarm size increases. We conducted 30 coevolution runs with each number of agents $n \in \{1, 5, 10, 50, 100\}$. There was always one replica in the group. When observing only one

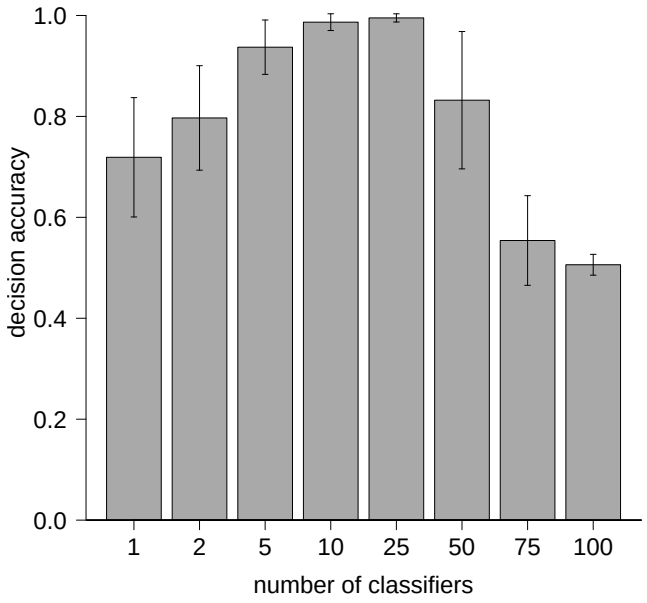



Figure 3.13: This plot shows the decision accuracy of the *classifier system (archive)* in the $1000^{\text{th}}$ generation of 30 coevolution runs with various number of classifiers chosen to form a system.

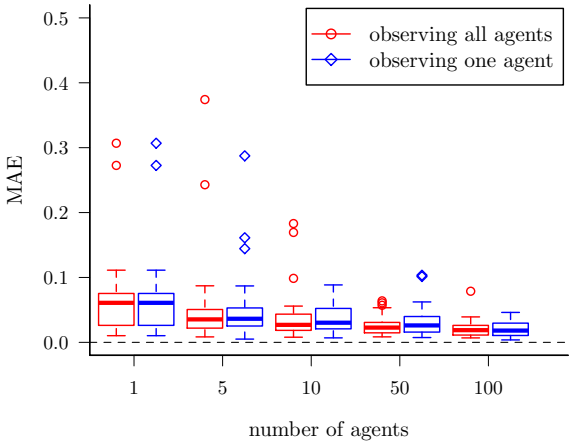



Figure 3.14: MAE (defined in Equation (3.10)) of the evolved models with the highest subjective fitness after 1000 generations, when using *Turing Learning* with varying numbers of agents (excluding the replica). Red and blue boxes show, respectively, the cases where all agents are observed, and one agent is observed. Boxes show distributions over 30 coevolution runs.

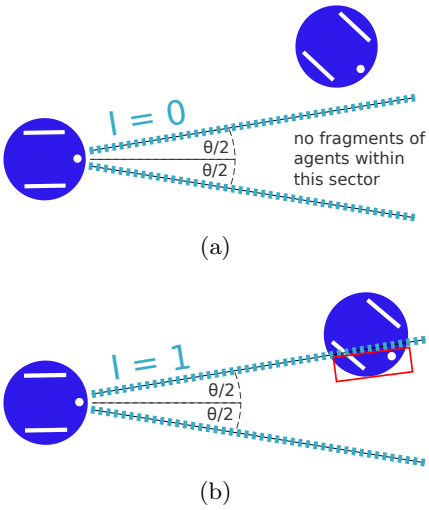



Figure 3.15: A diagram showing the angle of view of the agent's sensor investigated in Section 3.3.5.

agent, this was chosen randomly in each trial. Note that the total number of trials in each coevolution run for the case of observing all agents and one agent is identical. We measured the total square error of the model with the highest subjective fitness in the last ($1000^{\text{th}}$) generation of each run. The results are shown in Fig. 3.14.

There is no statistically significant difference for any $n$. On the other hand, as the swarm size increases, performance improves. For example, there is a statistically significant difference between $n = 10$ and $n = 100$, both when observing all agents and one. These results suggest that the key factor in the coevolutionary process is not the number of observed agents, so much as the richness of information that comes from increasing inter-agent interactions, and is reflected in each agent's motion. This means that in practice, observing a single agent is sufficient to infer accurate models, as long as the swarm size is sufficiently large.

A similar phenomenon was observed in a different scenario, where the goal was to distinguish between different 'modes' of a swarm (i.e., global behaviors) through observing only a few individuals [**?** ].

### 3.3.5 Evolving Control and Morphology

In the previous sections, we assumed that we fully knew the agents' morphology (i.e., structure), and only their behavior (controller) was to be identified. We now present a

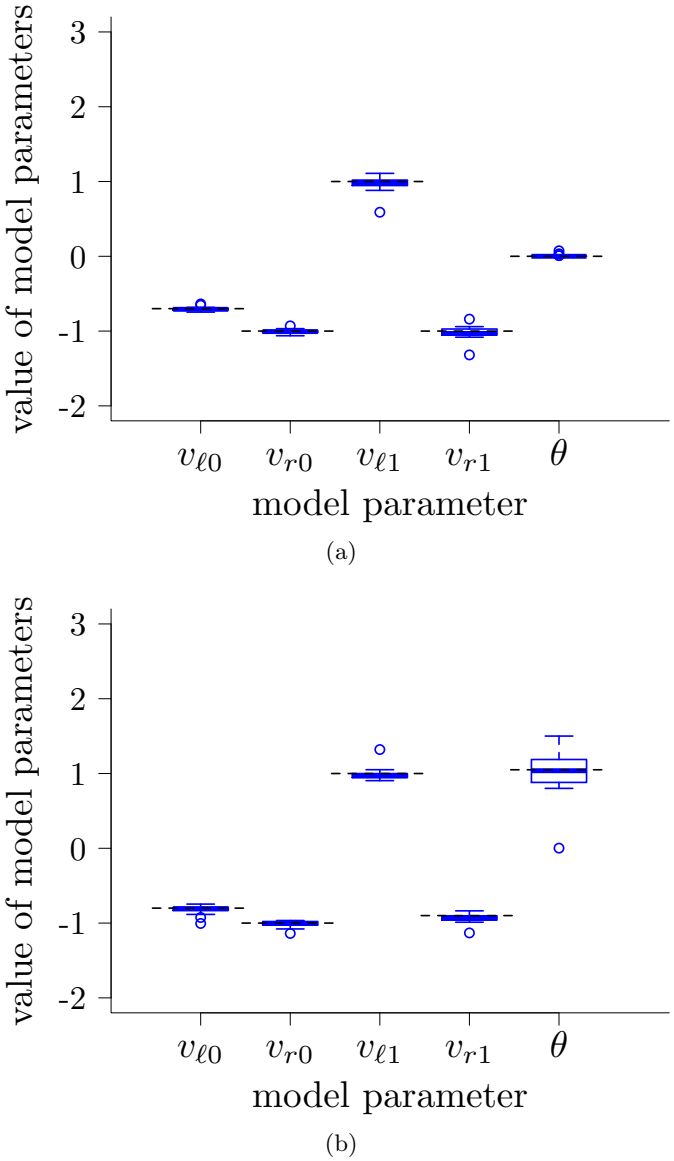



Figure 3.16: Parameters (controller parameters and angle of view in rad) of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation corresponding to the case of the agents' angle of view equal to (a) 0 rad and (b) $\pi/3$ rad. Boxes show distributions over 30 coevolution runs. Dotted black lines indicate true values.

variation where one aspect of the morphology is also unknown. The replica, in addition to the four controller parameters, takes a parameter $\theta \in [0, 2\pi]$ rad, which determines the horizontal field of view of its sensor, as shown in Fig. 3.15 (however, the sensor is still binary). Note that in the previous sections the agents' line-of-sight sensor can be

considered as a sensor with angle of view of 0 rad.

The models now have five parameters. As before, we let the coevolution run in an unbounded search space (i.e., now, $\mathbb{R}^5$). However, as $\theta$ is necessarily bounded, before a model was executed on a replica, the parameter corresponding to $\theta$ was mapped to the range $[0, 2\pi]$ using a logistic sigmoid function (Equation (3.1)). The controller parameters were directly passed to the replica. In this setup, the classifiers observed the individuals for 100 s in each trial (preliminary results indicated that this setup requires a longer observation time).

Fig. 3.16(a) shows the parameters of the subjectively best models in the last ($1000^{\text{th}}$) generations of 30 coevolution runs. The means (standard deviations) of the AEs in each model parameter were: 0.02 (0.01), 0.02 (0.02), 0.05 (0.07), 0.06 (0.06) and 0.01 (0.01). All parameters including $\theta$ were still learned with high accuracy.

The case where the true value of $\theta$ is 0 rad is an edge case, because given an arbitrarily small $\epsilon > 0$, the logistic sigmoid function maps an infinite domain of values onto $(0, \epsilon]$. This makes it easier for the coevolution to learn this parameter. For this reason, we also considered another scenario where the agents' angle of view is $\pi/3$ rad rather than 0 rad. The controller parameters for achieving aggregation in this case are different from those in Equation (3.7). They were found by re-running a grid search with the modified sensor. Fig. 3.16(b) shows the results from 30 coevolutions with this setup. The means (standard deviations) of the AEs in each parameter were: 0.04 (0.04), 0.03 (0.03), 0.05 (0.06), 0.05 (0.05) and 0.20 (0.19). The controller parameters were still learned with good accuracy. The accuracy in the angle of view is noticeably lower, but still reasonable.

### 3.3.6 Evolving Other Behaviors

The aggregation controller that agents used in our case study was originally synthesized by searching over the space $[-1, 1]^4$, using a metric to assess the swarm's global performance [21]. Other points in this space lead to different global behaviors that can be 'meaningful' to a human observer (e.g. circle formation [**?** ]).

We now investigate whether our coevolutionary method can learn arbitrary controllers in this space, irrespective of the global behaviors they lead to. We generated 1000 controllers randomly in $[-1, 1]^4$, with uniform distribution. For each controller we performed one coevolution run, and selected the subjectively best model in the last ($1000^{\text{th}}$) generation.

Fig. 3.18(a) shows a histogram of the MAE of the evolved models. The distribution has a single mode close to zero, and decays rapidly for increasing values. Over 89% of the 1000 cases have an error below 0.05. This suggests that the accuracy of *Turing Learning* is not highly sensitive to the particular behavior under investigation (i.e., most behaviors are learned equally well). Fig. 3.18(b) shows the AEs of each model parameter. The means (standard deviations) of the AEs in each parameter were: 0.01 (0.05), 0.02 (0.07), 0.07 (0.6) and 0.05 (0.20). We performed a statistical test on the AEs between the model parameters corresponding to $I = 0$ ($v_{\ell 0}$ and $v_{r0}$) and $I = 1$ ($v_{\ell 1}$ and $v_{r1}$). The AEs of the evolved $v_{\ell 0}$ and $v_{r0}$ were significantly lower than those of the evolved $v_{\ell 1}$ and $v_{r1}$. This was likely due to the reason reported in Section 3.3.1; that is, an agent spends more time seeing nothing ($I = 0$) than other agents ($I = 1$) in each trial.

### 3.3.7 Noise Study

We conducted a study to investigate how the performance of *Turing Learning* is affected by noise on the measurements of the individuals' position and orientation. As the e-puck robot has a maximum linear speed of 12.8 cm/s, the maximum distance that it can travel in one control cycle (0.1 s) is 1.28 cm. For this reason, we define a disturbance of 1.28 cm to an individual's position as a measurement error of 100%. Similarly, we define a 100% measurement error on the individual's orientation as the maximum change in orientation that an e-puck can undergo in one control cycle. This corresponds to 0.5 rad.

We conducted 5 sets of 30 coevolutions for the noise values $M = \{0, 25, 50, 75, 100\}\%$. In a coevolution with a noise value $M$, every measurement of an individual's position is perturbed in a random direction and by a random distance chosen uniformly in $\left[0, 1.28\frac{M}{100}\right]$ cm. Similarly, every orientation measurement is perturbed by a random angle chosen uniformly in $\left[-0.5\frac{M}{100}, 0.5\frac{M}{100}\right]$.

Figure 3.19 shows the total square error of the evolved parameters in the final generations of the coevolutions. This plot reveals that the system still performs relatively well when a considerable amount of noise affects the individuals' motion tracking system.

## 3.4 Summary

This chapter has presented the simulation results of using the *Turing Learning* method to autonomously learn swarm behaviors through observation. To our knowledge, *Turing*

*Learning* is the first system identification method that does not rely on any predefined metric to quantitatively gauge the difference between agents and learned models. This eliminates the need to choose a suitable metric and the bias that such metric may have on the obtained solutions.

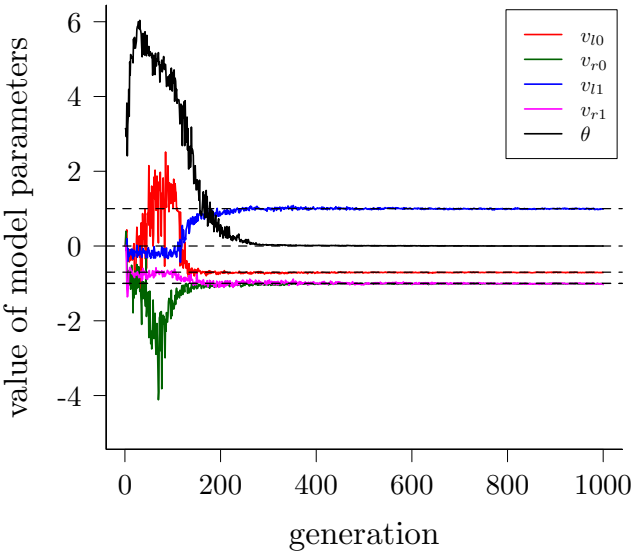
Through competitive coevolution of models and classifiers, the system successfully learned two swarm behaviors (self-organized aggregation and object clustering). Both the model parameters, which were automatically inferred, and emergent global behaviors closely matched those of the original swarm system. We also constructed a robust classifier system that, given an individual's motion data, can tell whether the individual is an original agent or not. Such classifier system could be effective in detecting abnormal behavior, for example, when faults occur in some members of the swarm. Note that *Turing Learning* produces these classifiers automatically without the need to define a priori what constitutes abnormal behavior.

A scalability study showed that the interactions in a swarm can be characterized by the effects on a subset of agents. In other words, when learning swarm behaviors especially with large number of agents, instead of considering the motion of all the agents in the group, we could focus on a subset of agents. This becomes critical when the available data about agents in the swarm is limited. Our approach was proven to work even if using only the motion data of a single agent and replica, as the data from this agent implicitly contained enough information about the interactions in the swarm.

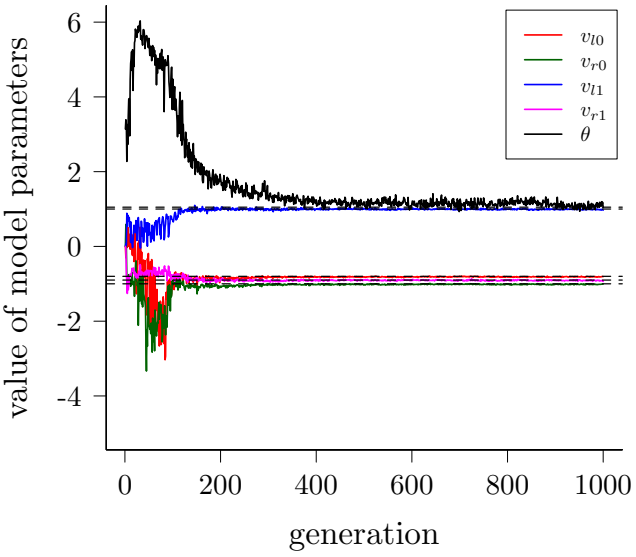
In this thesis, the model was explicitly represented by a set of parameters. The evolved parameters could thus be compared against the ground truth, enabling us to objectively gauge the quality of evolved models in two case studies as well as for 1000 randomly sampled behaviors. In principle, we could also evolve the structure of the agent's control system. The results of learning the agent's angle of view showed that our method may even learn the morphology of the swarming agents.

In collective behavior, abnormal agent(s) may have a great impact on the swarm [**?** ]. For the same reason, the insertion of a replica that exhibits different behavior or is not recognized as con-specific may disrupt the global behavior and hence the models obtained may be biased. An appropriate strategy would be to isolate the influence of the replica. In particular, to evaluate a model one could perform two trials, one with only replicas each executing the model and the other with only agents. The data of the replicas and agents from each trial could then be fed into the classifiers for making judgments. Some

preliminary results suggest that there is no significant difference between either approach for the case studies considered in this paper.

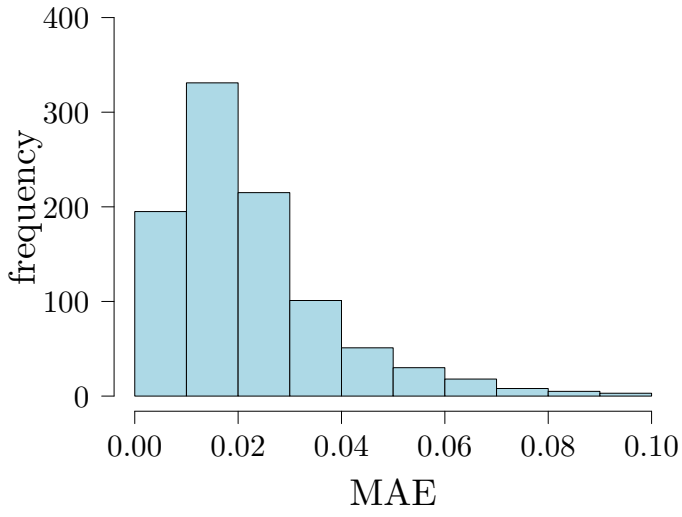



(a)

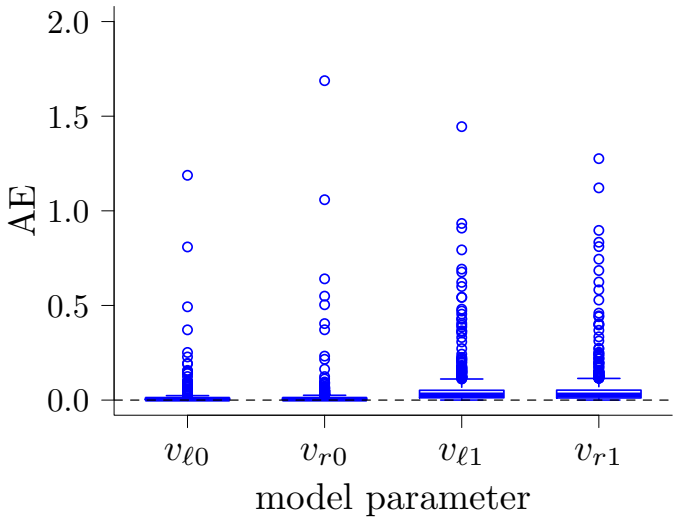



(b)

Figure 3.17: Evolutionary process of the evolved models (controller parameters and angle of view in rad) in the $1000^{\text{th}}$ generation corresponding to the case of the agents' angle of view equal to (a) 0 rad and (b) $\pi/3$ rad. Boxes show distributions over 30 coevolution runs. Dotted black lines indicate true values..




(a)




(b)

Figure 3.18: This plot shows (a) a histogram of the MAE (defined in Equation (3.10)) of the evolved models and (b) the AEs (defined in Equation (3.9)) of each model parameter in the $1000^{\text{th}}$ generation over 1000 random behaviors. For each behavior, we performed one coevolution run. In (a), 43 points that have MAE larger than 0.1 are not shown.

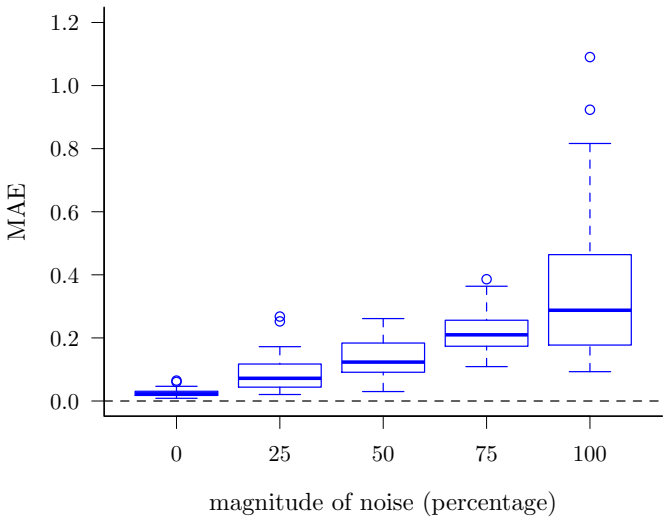



Figure 3.19: This plot shows the total square error of the evolved parameters when increasing the percentage of noise on the measurements of the individuals' position and orientation in the aggregation behavior. Each box corresponds to 30 coevolutionary runs. See text for details.

# 4 A Real-World Validation of Turing Learning

In this chapter, we present a real-world validation of *Turing Learning*. In particular, we have built an autonomous system for performing system identification with little human intervention. We demonstrate how the system can be used to identify the behavior of a swarm of real agents. The agents and replica are represented by physical robots. We use the same type of robot (e-puck) as in simulation. The agents execute the aggregation behavior described in Section 3.1.2.2. The replicas execute the evolved models. We use two replicas to speed up the coevolutionary process, as will be explained in Section **??**. Compared with simulation which could not model very accurately the dynamics (such as collision) in reality, evolution on physical systems breaks the gap between simulation and reality and brings us new insight on how to implement evolution in real world.

This chapter is organized as follows. Section 4.1 introduces the physical platform, which includes the robot arena, the robot platform and the sensors implementation. Section 4.2 details the tracking system, including motion capture and video processing. Section 4.3 describes the programs executed by each component (machine, agent and replica) during the coevolutionary learning process. Section 4.4 describes the experimental setup. Section 4.5 discusses the results obtained. This includes the analysis of the evolved models and the analysis of the evolved classifiers. Section 4.6 analyzes the sensitivity of *Turing Learning* for individual failure during the experimental process. Section 4.7 summaries the results obtained and discusses the findings in this chapter.

## 4.1 Physical Platform

The physical setup, shown in Figure 4.1, consists of an arena with robots (representing agents or replicas), a personal computer (PC) and an overhead camera. The PC runs the

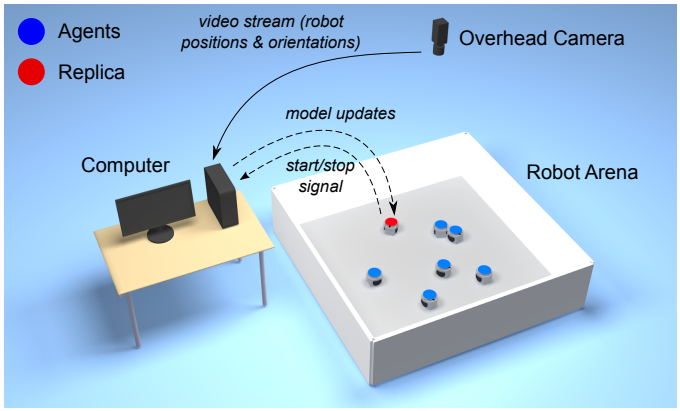



Figure 4.1: Illustration of the general setup for inferring the behavior of physical agents—e-puck robots (not to scale). The computer runs the coevolutionary algorithm, which produces models and classifiers. The models are uploaded and executed on the replica. The classifiers run on the computer. They are provided with the agents' and replica's motion data, extracted from the video stream of the overhead camera.

coevolutionary algorithm[1]. It communicates with the replicas, providing them models to be executed, but does not exert any control over the other agents. The overhead camera supplies the PC with a video stream of the swarm. The PC performs video processing to obtain motion data about individual robots. We will now describe the physical platform in more detail.

### 4.1.1 Robot Arena

The robot arena was rectangular with sides 200 cm × 225 cm, and bounded by walls 50 cm high. The floor had a light gray color, and the walls were painted white.

### 4.1.2 Robot Platform and Sensor Implementations

#### 4.1.2.1 Robot Platform

In Section 3.2.1, we presented the e-puck's shape and dimensions as a basis for the agents' embodiment in simulation. We now present further details about the e-puck relevant to

[1] The evolution of the model population could in principle be conducted on the on-board micro-controller of the e-puck, but running it on the PC reduces experimental time [**?** ] and eases post-evaluation analysis.

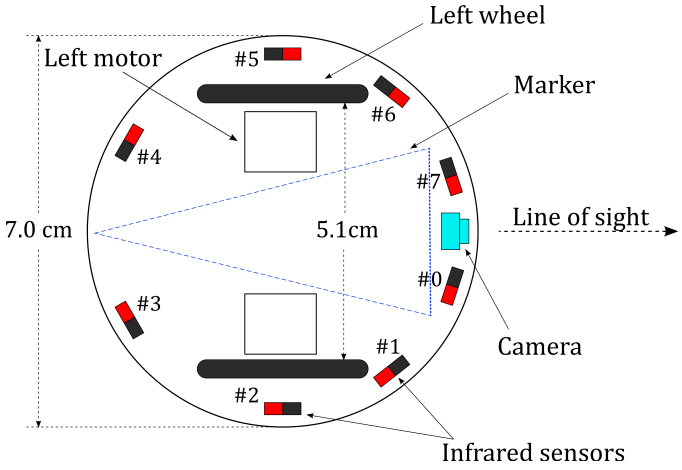



Figure 4.2: Schematic top-view of an e-puck, indicating the locations of its motors, wheels, camera and infrared sensors. Note that the marker is pointing towards the robot's back.

our physical implementation. A schematic top view of the e-puck, showing the sensors and actuators, is shown in Figure 4.2.

The e-puck has a on-board micro-controller, which is the Microchip dsPIC30F6014A with 8KB of RAM and 144 KB of flash memory. The two wheels are driven by the stepper motors. The e-puck has a directional (color) camera located in its front. The resolution of the camera is up to $640 \times 480$, corresponding to the horizontal and vertical angle view of $56°$ and $48°$. There are eight infrared proximity sensors around the body of the robot. The proximity sensors are only used for collision/wall avoidance in the physical coevolutions where the environment is bounded.

#### 4.1.2.2 Sensor Implementations

We implemented the line-of-sight sensor using the e-puck's directional camera. For this purpose, we wrapped the robots in black 'skirts' (see Figure 3.2) to make them distinguishable against the light-colored arena. However, we use the camera in monochrome mode, and sub-sample the image to $40 \times 15$ pixels, due to the e-puck's limited memory (which cannot even store a single full-resolution image). While in principle the sensor could be implemented using one pixel, we used a column of pixels from a sub-sampled image to compensate for misalignment in the camera's vertical orientation. The gray values from these pixels were used to distinguish robots ($I = 1$) against the arena ($I = 0$). If

any pixel of that column exceeds a certain threshold in its gray scale, the sensor outputs 1; otherwise, it outputs 0. For more details about this sensor realization, see [21].

We also used the e-puck's infrared sensors, in two cases. Firstly, before each trial, the robots dispersed themselves within the arena (behavior *R2* in Section 4.3.2). In this case, they used the infrared sensors to avoid both robots and walls, making the dispersion process more efficient. Secondly, we observed that using only the line-of-sight sensor can lead to robots becoming stuck against the walls of the arena, hindering the coevolutionary process. We therefore used the infrared sensors for wall avoidance, but in such a way as to not affect inter-robot interactions[2]. In the following, we details the implementation of these two programs (*disperse* program and *wall avoidance* program).

1) The implementation of *disperse* program after a trial:

After finishing a trial, we disperse the robots for a while (which is equivalent to initial configuration for a new trial) in order to automate the coevolutionary learning process. This program consists of two behaviors  obstacle avoidance and disperse. The obstacle avoidance behavior is to prevent the robots colliding with other robots and the walls. In particular, before executing the disperse behavior, each robot detects whether some other objects (robots/walls) exist around it using its infrared proximity sensors. If it detects something, it moves away from the objects through adjusting the linear and angular speed accordingly using a single-layer neural network controller. The obstacle avoidance behavior lasts for 3 seconds. In the disperse behavior, each robot is moving forward with a fixed linear speed while avoiding collisions with other robots and the walls. This behavior lasts for 5 seconds.

2) The implementation of *wall avoidance* program during a trial:

During a trial, in order to reduce the chances of robots getting stuck against the walls (note that the aggregation behavior was designed in an unbounded environment), we imposed a wall avoidance effect to the original behavior, but in such a way as to not affect inter-robot interactions. In particular, when the robot detected the white walls using the infrared sensors or saw another robot (I=1) using the camera, it executed the same behavior. For example, for the agent, it would also turn on the spot when detecting the walls, which makes it easier to avoid the walls. However, the behavior of the replica depends on the model it is executing. Different from the Disperse program, the program of wall avoidance was only triggered when the value of any of the robot's infrared sensor

[2]To do so, the e-pucks determine whether a perceived object is a wall or another robot.

was above a pre-set high threshold. This ensures that the value of the robot's infrared sensors when other robots (covered with a black 'skirt') were nearby was always below the threshold. Therefore, the wall avoidance program did not affect the aggregation of robots.

## 4.2 Motion Capture and Video Processing

### 4.2.1 Motion Capture

To facilitate motion data extraction, we fitted robots with markers on their tops, consisting of a colored isosceles triangle on a circular white background. The triangle's color allowed for distinction between robots; we used blue triangles for all agents, and orange and purple triangles for the two replicas. The triangle's shape eased extraction of robots' orientations. Note that the orientation of the triangle is pointing to the backward of the e-puck robot so that it can be easily attached.

The robots' motion was captured using a GigE color camera (Basler Technologies), mounted around 300 cm above the arena floor. The camera's frame rate was set to 10 fps. The video stream was fed to the PC, which performed video processing to extract motion data about individual robots (position and orientation). The size of the arena in the image is 700 pixels × 800 pixels. An image of the arena captured using the overhead camera is shown in Figure 4.3.

### 4.2.2 Video Processing

The video processing software was written using OpenCV—an open-source computer vision library [**?** ]. The details of the video processing algorithm are described as follows.

The image captured using the overhead camera was encoded using RGB. We changed the encoding into HSV in order to make the tracking algorithm less sensitive to lighting variation. This was realized using a built-in function inside the OpenCV library. After that, the image was converted into grayscale and thresholded. As the background of the arena was light-colored, there was no need to do background extraction. After that,

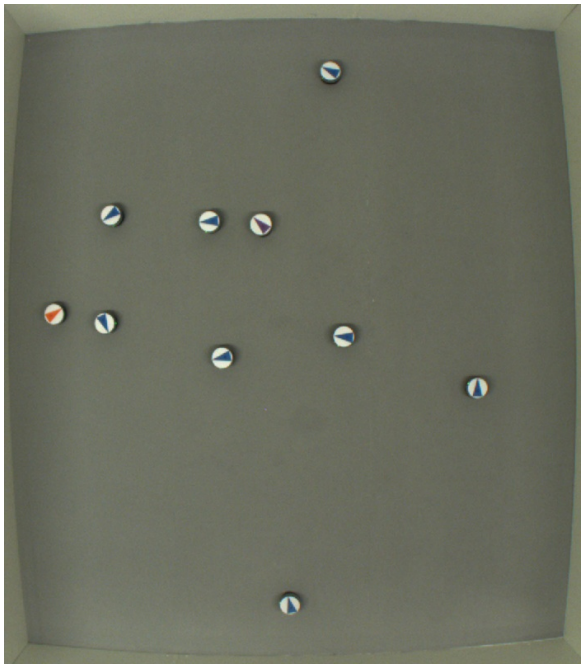


Figure 4.3: An image of the robot arena captured by the overhead camera. The robots inside the arena are covered with a colored triangle for facilitating the motion tracking.

the color images was converted into binary images. A morphological operation (erosion followed by dilation) is applied on the binary images to filter some noise. Blobs in the binary image with a size above certain threshold (36 pixels) are used for robot detection. These selected blobs indicate the robots in the arena. Figure 4.4 shows the selected blobs in an image.

The robots were tracked using the nearest neighbor algorithm. For each blob in the image, we found a minimum triangle that encloses the marker, and obtained the coordinates of the three vertices of this triangle. The vertex that has the longest distance to the other two vertices indicates the direction of the robot. Therefore the orientation of the robot was estimated using the vector pointing from the midpoint of the other two vertices to this vertex. We use moments [**?** ] to calculate the position of the center of the robot. The x and y coordinate of the position is the $1^{\text{th}}$ order spatial moments around x-axis and y-axis divided by the $0^{\text{th}}$ order central moments of the blob, respectively. Figure 4.5 shows a diagram of the video processing algorithm.

Agents Replica one Replica two

Figure 4.4: The binary images showing the blobs of the robots (agents or replicas) in the arena after video processing. The blobs are selected according to certain compactness and size.

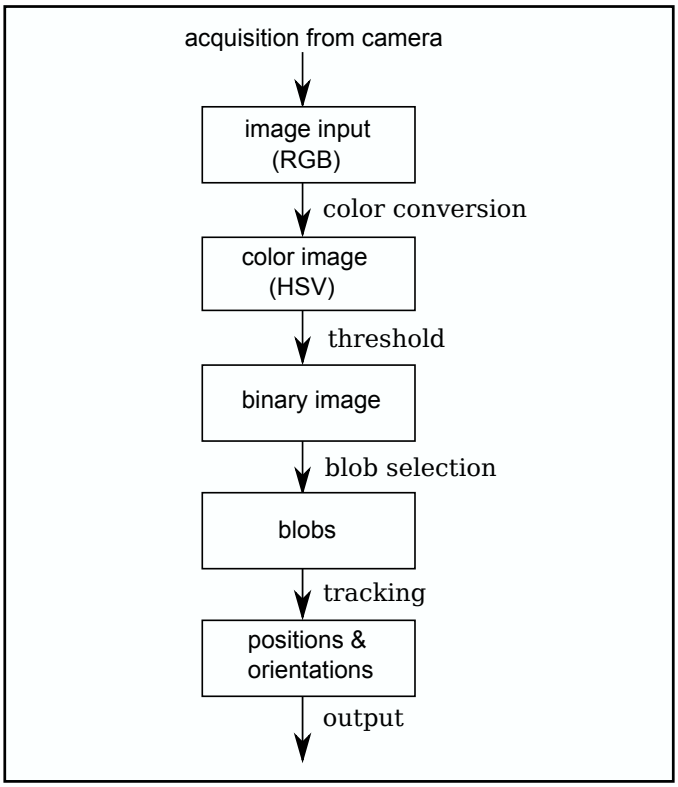



Figure 4.5: A diagram showing the flow of image processing algorithm used in the tracking system.

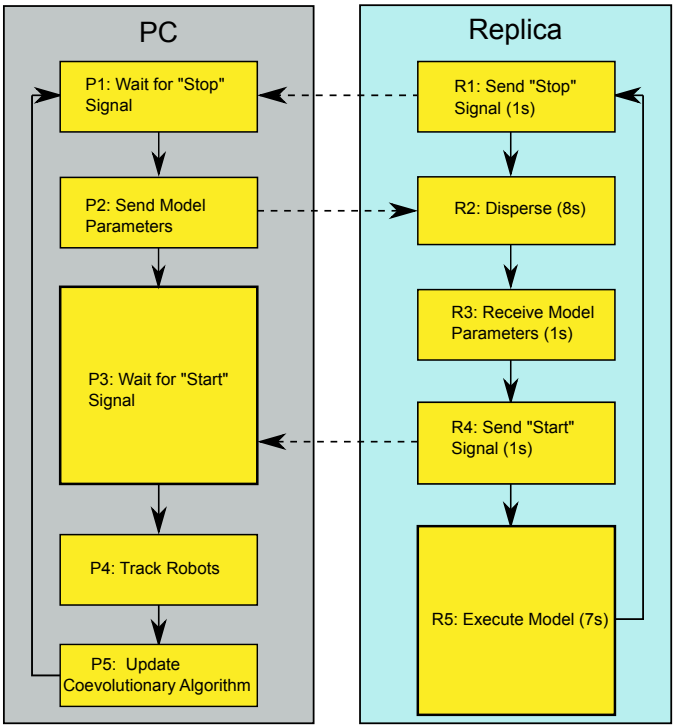



Figure 4.6: Schematic of the programs run by the PC and a replica in the physical experiments. Dotted arrows represent communication between the two units. See Section 4.3 for details.

## 4.3 Coevolution with Physical Robots

To automate the coevolutionary process, the robots (agents and replicas) were implicitly synchronized. This was realized by making each robot execute a fixed behavioral loop of constant duration. The PC also executed a fixed behavioral loop, but the timing was determined by the signals received from the replicas. Therefore, the PC was synchronized with the swarm. The PC communicated with the replicas via Bluetooth. At the start of a coevolution run, or after a human intervention (see Section 4.4), robots were initially synchronized using an infrared signal from a remote control.

Figure 4.6 shows a flow diagram of the programs run by the PC and the replicas, respectively. Dotted arrows indicate communication between the units. The agents executed a similar behavioral loop to the replicas. In the following, we detail the states of the programs executed by the PC, replicas, and agents.

### 4.3.1 PC Program

- *P1: Wait for "Stop" Signal.* The program is paused until "Stop" signals are received from both replicas. These signals indicate that a trial has finished.
- *P2: Send Model Parameters.* The PC sends new model parameters to the buffer of each replica.
- *P3: Wait for "Start" Signal.* The program is paused until "Start" signals are received from both replicas, indicating that a trial is starting.
- *P4: Track Robots.* The PC waits 1 s and then tracks the robots using the overhead camera for 5 s. The tracking data contains the positions and orientations of the agents and replicas.
- *P5: Update Coevolutionary Algorithm.* The PC uses the motion data from the trial observed in *P4* to update the fitness of the corresponding two models and all classifiers. Once all models in the current generation have been evaluated, the PC also generates new model and classifier populations. The fitness calculation and optimization algorithm are described in Section 3.1.1.4. The PC then goes back to *P1*.

### 4.3.2 Replica Program

- *R1*: *Send "Stop" Signal.* After a trial stops, the replica informs the PC by sending a "Stop" signal. The replica waits 1 s before proceeding with *R2*, so that all robots remain synchronized. Waiting 1 s in other states serves the same purpose.
- *R2*: *Disperse.* The replica disperses in the environment, while avoiding collisions with other robots and the walls. This behavior lasts 8 s.
- *R3*: *Receive Model Parameters.* The replica reads new model parameters from its buffer (sent earlier by the PC). It waits 1 s before proceeding with *R4*.
- *R4*: *Send "Start" Signal.* The replica sends a start signal to the PC to inform it that a trial is about to start. The replica waits 1 s before proceeding with *R5*.
- *R5*: *Execute Model.* The replica moves within the swarm according to its model. This behavior lasts 7 s (the tracking data corresponds to the middle 5 s, see *P4*). The replica then goes back to *R1*.

### 4.3.3 Agent Program

- The agents follow the same behavioral loops as the replicas. However, in the states analogous to *R1*, *R3*, and *R4*, they simply wait 1 s rather than communicate with the PC. In the state corresponding to *R2*, they also execute the *Disperse* behavior. In the state corresponding to *R5*, they execute the original aggregation controller, rather than a model.

## 4.4 Experimental Setup

As in simulation, we used a population size of 100 for classifiers ($\mu = 50$, $\lambda = 50$). However, the model population size was reduced from 100 to 20 ($\mu = 10$, $\lambda = 10$), to shorten the experimental time. We used 10 robots: 8 representing agents executing the original aggregation controller (Equation (3.7)), and 2 representing replicas that executed models. This meant that in each trial, 2 models from the population could be evaluated simultaneously; consequently, each generation consisted of $20/2 = 10$ trials.

The coevolutionary algorithm was implemented without any modification to the code used in simulation (except for model population size and observation time in each trial). We still let the model parameters evolve unboundedly (i.e., in $\mathbb{R}^4$). However, as the speed of the physical robots is naturally bounded, we applied the hyperbolic tangent function ($\tanh x$) on each model parameter, before sending a model to a replica. This bounded the parameters to $(-1, 1)^4$, with $-1$ and 1 representing the maximum backward and forward wheel speeds, respectively.

The coevolution runs proceeded autonomously. In the following cases, however, human intervention was made:

- The robots had been running continuously for 25 generations. All batteries were replaced.
- Hardware failure occurred on a robot, for example because of a lost battery connection or because the robot became stuck on the floor. Appropriate action was taken for the affected robot to restore its functionality.
- A replica lost its Bluetooth connection with the PC. The connection with both replicas was restarted.

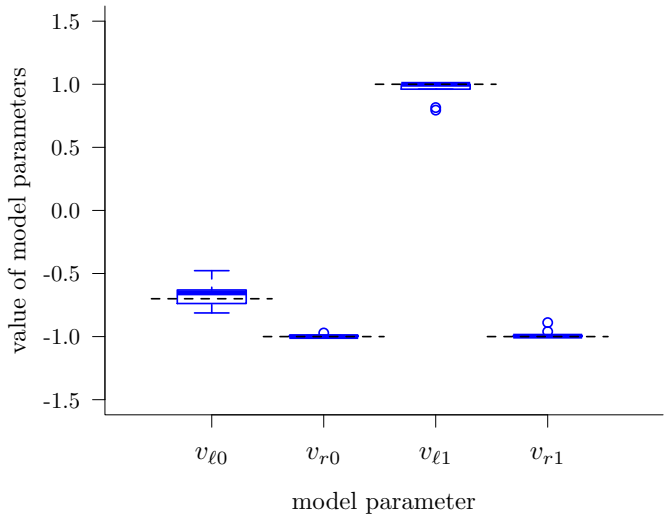



Figure 4.7: Parameters of the evolved models in the $100^{\text{th}}$ generation of 10 physical coevolution runs. Dotted black lines indicate true values.

- A robot indicated a low battery status through its LED after running for only a short time. That robot's battery was changed.

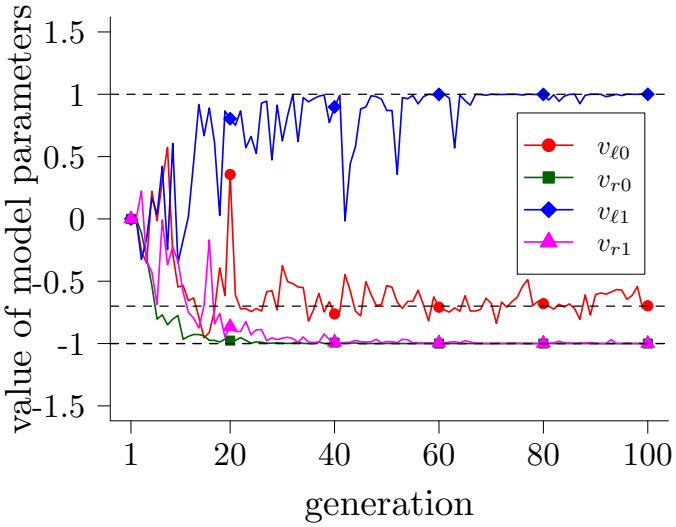
After an intervention, the ongoing generation was restarted, to limit the impact on the coevolutionary process.

## 4.5 Results

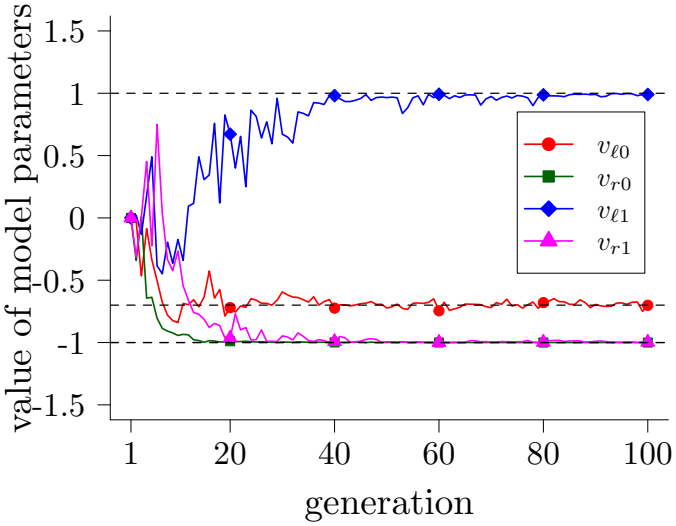
We conducted 10 coevolution runs using the physical system. Each run lasted 100 generations, corresponding to 5 hours (excluding human intervention time). Video recordings of the coevolution runs can be found in the online supplementary materials [**?** ].

### 4.5.1 Analysis of Evolved Models

We will first investigate the quality of the models obtained. To select the 'best' model from each coevolution run, we post-evaluated all models of the final generation 5 times using all classifiers of that generation. The parameters of these models are shown in Figure 4.7. The means (standard deviations) of the AEs in each parameter were: 0.08 (0.06), 0.01 (0.01), 0.05 (0.08), and 0.02 (0.04).



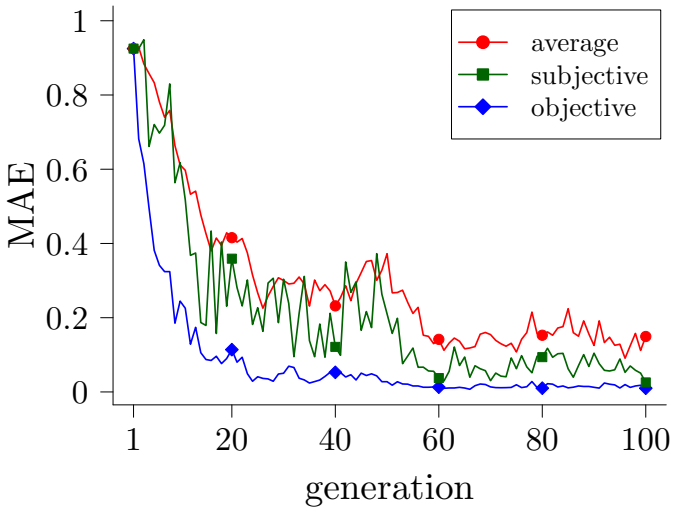

(a) Physical Coevolutions



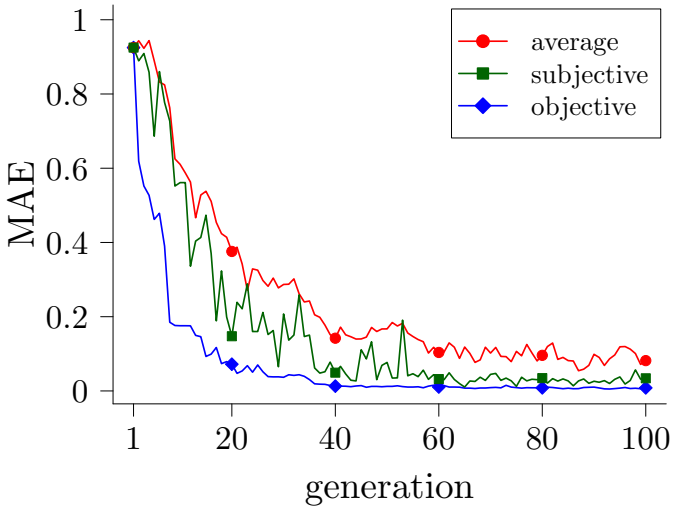

(b) Simulated Coevolutions

Figure 4.8: Evolutionary progress of each model parameter in 10 (a) physical and (b) simulated coevolution runs. Curves represent median values across 10 runs. Dotted black lines indicate true values.

To investigate the effects of real-world conditions on the coevolutionary process, we performed 10 simulated coevolution runs with the same setup as in the physical runs. Figure 4.8 shows the evolutionary dynamics of the parameters of the evolved models (with the highest subjective fitness) in the physical and simulated coevolution runs. The dynamics show good correspondence. However, the convergence in the physical




(a) Physical Coevolutions




(b) Simulated Coevolutions

Figure 4.9: Evolutionary progress of the MAE (defined in Equation (3.10)) of models in 10 (a) physical and (b) simulated coevolution runs. Curves represent median values across 10 runs. The red curve represents the average error of all models in a generation. The green and blue curves show, respectively, the errors of the models with the highest subjective and the highest objective fitness in a generation.

coevolutions is somewhat less smooth than that in the simulated ones (e.g., see spikes in $v_{\ell 0}$ and $v_{\ell 1}$). In each generation of every coevolution run (physical and simulated), we computed the MAE of each model. We compared the error of the model with the

highest subjective fitness with the average and lowest errors. The results are shown in Figure 4.9. For both the physical and simulated coevolution runs, the subjectively best model (green) has an error in between the lowest error (blue) and the average error (red) in the majority of generations.

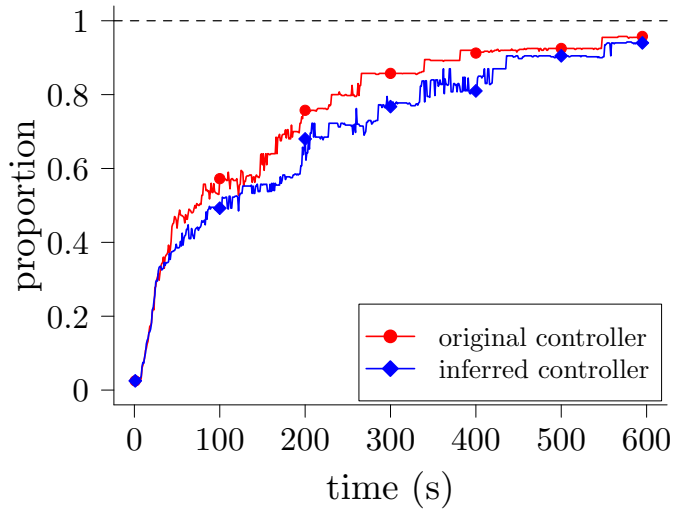
As we argued before (Section 3.3.1), in swarm systems, good agreement between local behaviors (e.g., controller parameters) may not guarantee similar global behaviors. For this reason, we performed 20 trials using 40 physical e-pucks, lasting 10 minutes each: 10 trials with the original controller (Equation (3.7)), and 10 trials with a controller obtained from the physical coevolution runs. This latter controller was constructed by taking the median values of the parameters over the 10 runs, which are:

$$\mathbf{p} = (-0.65, -0.99, 0.99, -0.99)\,.$$

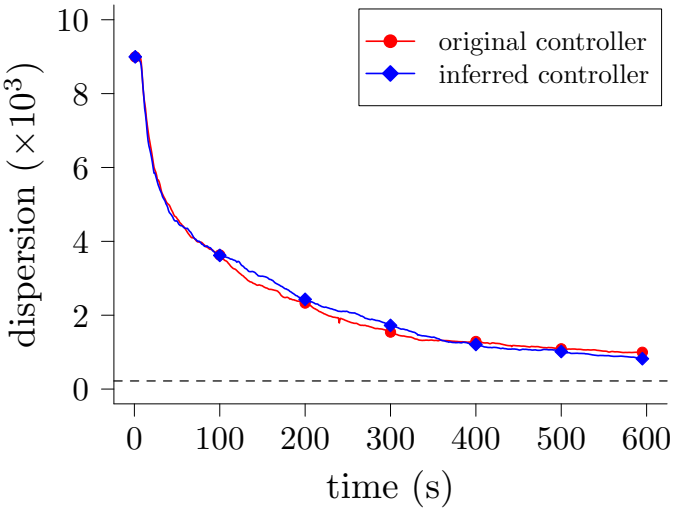
The set of initial configurations of the robots was common to both controllers. As it was not necessary to extract the orientation of the robots, a red circular marker was attached to each robot so that its position can be extracted with higher accuracy in the offline analysis [21]. Figure 4.10(a) shows the proportion of robots in the largest cluster[3] over time with the original and inferred controllers. Figure 4.10(b) shows the dispersion (as defined in Section 3.3.1) of the robots over time with the two controllers. The aggregation dynamics of the original and inferred behaviors show good correspondence. Figure 4.12 shows a sequence of snapshots from a trial with 40 e-pucks executing the evolved controller.

We also performed a scalability study in simulation to compare the real and evolved controllers. We calculated the time taken to form a single cluster with different numbers of robots: $n \in \{10, 20, \ldots, 100\}$. For each number of robots, we performed 30 trials with each controller. The results are shown in Figure 4.11. In all cases, the real controller slightly outperforms the evolved one. However, with either controller, the robots aggregate in a relatively short time ($< 600\,\mathrm{s}$). There is a statically significant difference in aggregation times with the two controllers for $n = 10, 30, 50, 70, 80, 90$—but this suggests no particular pattern with respect to $n$. We also performed a linear least squares regression on the aggregation times with the two controllers. The fits were: $T = 53.2 + 2.2n$ for the real controller, and $T = 64 + 2.4n$ for the evolved controller ($n > 1$). This indicates that the evolved controller does not scale much worse than the real one.

[3] A cluster of robots is defined as a maximal connected subgraph of the graph defined by the robots' positions, where two robots are considered to be adjacent if another robot cannot fit between them [21].




(a) Largest Cluster Dynamics




(b) Dispersion Dynamics

Figure 4.10: Average aggregation dynamics in 10 physical trials with 40 e-puck robots executing the original controller (red) and the evolved controller (blue). In (a), the vertical axis shows the proportion of robots in the largest cluster; in (b), it shows the robots' dispersion (see Section 3.3.1). Dotted lines in (a) and (b), respectively, represent the maximum proportion and minimum dispersion that 40 robots can achieve.

A video accompanying this paper shows the evolutionary process of the models (in a particular coevolution run) both in simulation and on the physical system. Additionally, videos of all 20 post-evaluation trials with 40 e-pucks, are available in [**?** ].

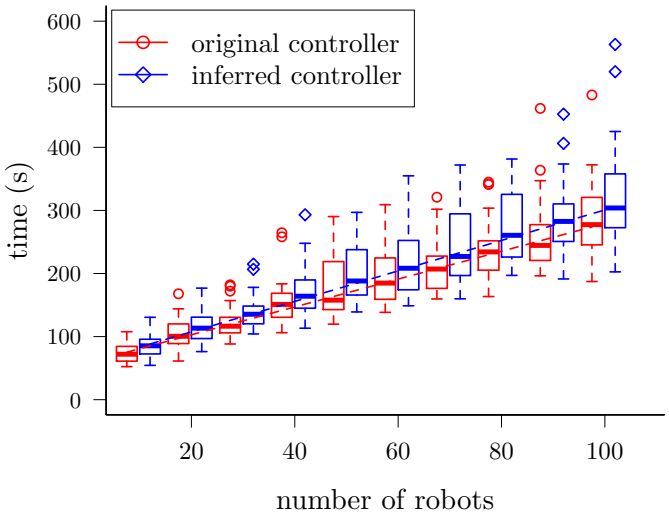



Figure 4.11: Time taken for different numbers of robots to aggregate into a single cluster. Each box corresponds to 30 simulation trials with the real controller (red), and the controller obtained from the physical coevolution runs (blue).

### 4.5.2 Analysis of Evolved Classifiers

When post-evaluating the classifiers evolved in the physical coevolution runs, we limited the number of candidate models to 100, in order to reduce the experimental time. Each candidate model was randomly chosen with uniform distribution from $[-1, 1]^4$. Figure 4.13 shows the average decision accuracy of the best classifiers and classifier systems over the 10 coevolution runs. Similar to the results in simulation, the classifier systems obtained in the physical coevolution runs still have a high decision accuracy. The *classifier system (archive)* has a higher accuracy than that of the *classifier system (subjective)* and *best classifier (objective)*. However, in contrast to simulation, the decision accuracy of the *best classifier (subjective)* and *best classifier (archive)* does not drop within 100 generations. This could be due to the smaller model population size in the physical coevolution runs, which may have prevented the classifiers from getting over-specialized in a short time.

To investigate whether the decision accuracy is still a good measurement of the quality of the classifiers in the physical coevolution runs, we plot a figure showing the fitness of the best classifiers corresponding to Figure 4.13. The fitness is calculated based on the 100 randomly generated models. The results are shown in Figure 4.14. Similar to the results obtained in simulation, the trend of fitness curves show good correspondence

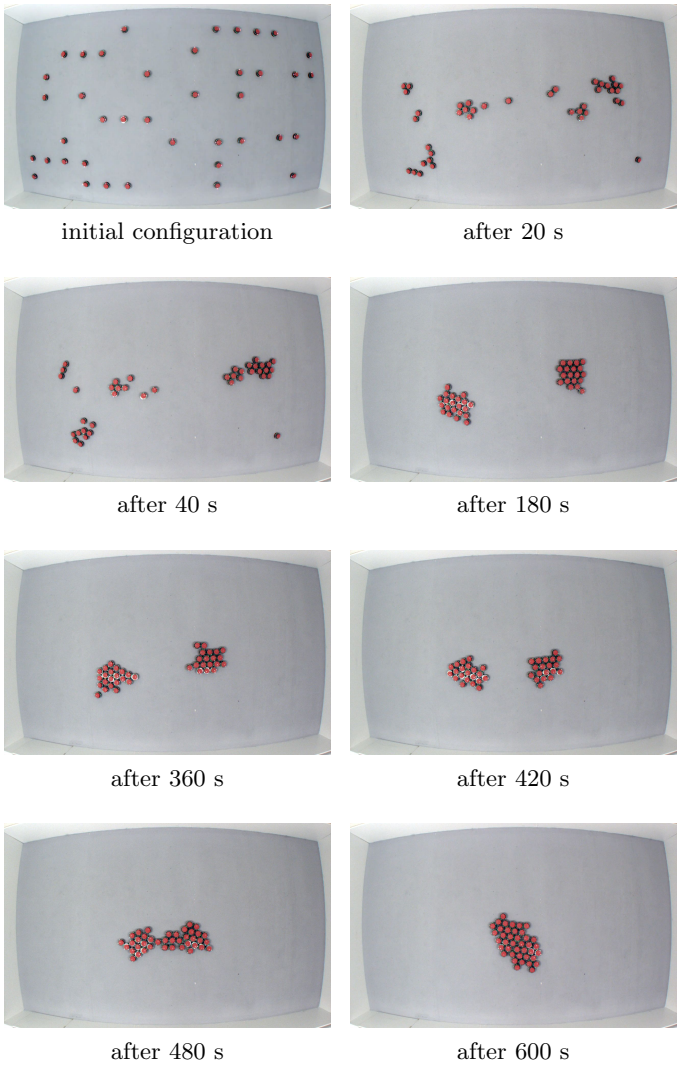



Figure 4.12: Example of a collective behavior obtained by *Turing Learning*. A group of 40 e-puck robots, each executing the inferred model, aggregates in a single spot.

to that of the decision accuracy in Figure 4.13. The gap between the *best classifier (decision)* and *best classifier (objective)* in the physical experiments is bigger than that in simulation. One reason for this may be the limitations in motion data capture and extraction. Note that given the relatively small diameter of the e-puck in our arena, inferring its orientation is particularly challenging.

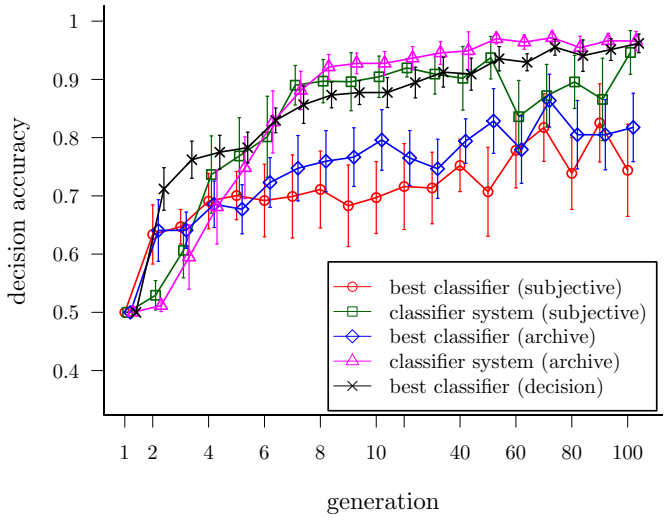



Figure 4.13: The average decision accuracy of the best classifiers and classifier systems over generations (nonlinear scale) in 10 physical coevolution runs. The error bars show standard deviations. See text for details.

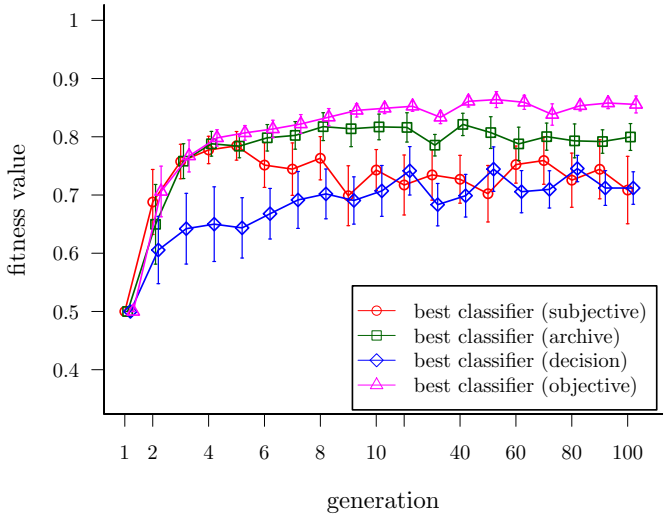



Figure 4.14: This plot shows the average fitness of the classifiers over generations (nonlinear scale) in 10 physical coevolution runs. The fitness is calculated based on 100 randomly generated models. See text for details.

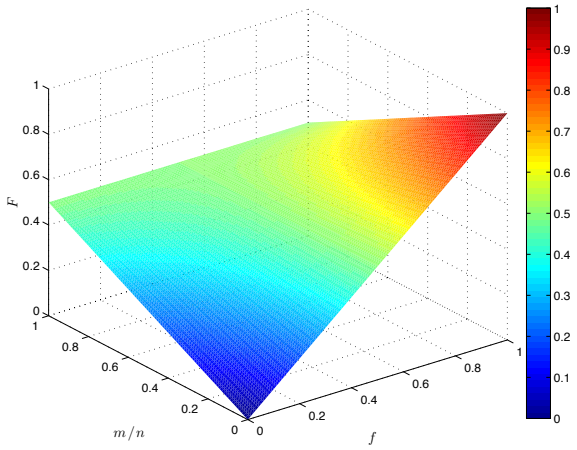



Figure 4.15: This plot shows the relationship between the old fitness, $f$, and the new fitness, $F$, of the classifiers/models, when the failure ratio of agents in the swarm, $\frac{m}{n}$, changes. When $m = 0$ or $\frac{m}{n} \to 0$, $F = f$. When $0 < \frac{m}{n} < 1$, $F$ is shifted, but the fitness order is not changed.

## 4.6 Analysis of Sensitivity for Individual Failure

In the physical experiments, we have observed that some robots may get stuck or stop working because of the reasons mentioned in Section 4.4. This could also happen in a real swarm system, for example, some individuals may stop moving or display other abnormal behaviors due to various factors such as collision. These abnormal behaviors (which are considered failure here) may not be recognized during the process of experiments. Therefore, we have analyzed whether abnormal behaviors (failure) of some individuals may bias the evolution and hence affect performance of our proposed coevolutionary approach. Assuming the failure is equally likely to be present in the replica and agent in a trial. We have proved that: ***Turing Learning* is not biased by failure of a few individuals in the swarm.** The mathematical proof is provided as follows:

*Proof.* Let $n$ be the total size of the group, consisting of $n-1$ agents and 1 replica. $m$ represents the number of individuals that fail ($m < n$) in a trial. $p$ and $q$ are the probabilities of a classifier to make the correct judgment for models and agents respectively. Therefore, the expected fitness of the classifier without failure of any individuals in one generation, $f_c$, is equal to $\frac{p+q}{2}$. $f_m$ denotes the fitness of a model in that generation.

There are two failure cases: 1) failure with the replica; 2) failure without the replica. We assume that if an individual fails during a trial, the classifier makes random judgment, that is, it has equal probability, which is 50%, of judging the individual as an agent or a model. If failure exists in a trial, the probability that failure with the replica occurs, $p_1$, is as follows:

$$p_1 = \binom{n-1}{m-1} / \binom{n}{m} = \frac{m}{n}. \tag{4.1}$$

The probability that failure without the replica occurs in a trial, $p_2$, is: $1 - p_1 = \frac{n-m}{n}$. The proof is divided into two parts: one for the classifiers and the other is for models.

For the classifier, the new fitness in the first failure case, $f_{c1}$, can be calculated as follows:

$$f_{c1} = p_1 \cdot (\frac{1}{2} + \frac{1}{2} \cdot \frac{m-1}{n-1} + \frac{q \cdot (n-m)}{n-1})/2. \tag{4.2}$$

In the second failure case, the new fitness of the classifier, $f_{c2}$, is given by:

$$f_{c2} = p_2 \cdot (p + \frac{1}{2} \cdot \frac{m}{n-1} + \frac{q \cdot (n-m-1)}{n-1})/2. \tag{4.3}$$

Combining Eqs. (4.2) and (4.3) and simplifying the resulting expression, the new fitness, $F_c$, of the classifier when $m$ individuals fail in a trial is:

$$F_c = f_{c1} + f_{c2} = f_c \cdot (1 - \frac{m}{n}) + \frac{m}{2n}. \tag{4.4}$$

In Equation(4.4), $F_c$ is monotonic increasing in terms of $f_c$. When $f_c$, is equal to 0.5, $F_c$ maintains the same value. When $f_c$ is greater than (or smaller than) 0.5, $F_c$ is decreased (or increased) but still greater (or smaller) than 0.5. Therefore, the fitness order of all the classifiers after failure of $m$ individuals in a trial is not changed, which means it does not affect the selection of the classifiers in the evolutionary process.

For the model, it has a new fitness of $f_{m1} = \frac{m}{n} \cdot \frac{1}{2}$ and $f_{m2} = \frac{(n-m) \cdot f_m}{n}$ in the first and second failure case, respectively. Therefore, the new fitness of the model, $F_m$, when $m$ individuals fail in a trial is:

$$F_m = f_{m1} + f_{m2} = f_m \cdot (1 - \frac{m}{n}) + \frac{m}{2n}. \tag{4.5}$$

Comparing Eqs. (4.4) and (4.5), we can see that the fitness order of all the models is also unchanged when failure occurs in a trial. □

## 4.7 Summary

In this chapter, the *Turing Learning* method was validated using a physical autonomous coevolutionary system. We applied it to identify the aggregation behavior in swarms of e-puck robots. The behavior was learned successfully, and the results obtained in the physical experiments showed good correspondence to those obtained in simulation. This shows the robustness of our method with respect to noise and uncertainties in real world. *Turing Learning* could be beneficial in the context of science automation [**?** **?** ], in particular to reveal the mechanisms underpinning collective animal behavior.

The model obtained in the physical coevolution runs was validated using 40 e-puck robots. The global behavior of the obtained model is similar to that of the original controller. The selected classifier system over the coevolution runs still obtained a high performance, which means our classifier system could be potentially applied to detect or monitor the behaviors of the agents in a swarm.

In order to speed up the coevolutionary process, different from simulation, we used two replicas instead of one. Therefore, two models can be executed simultaneously. In principle, we could further speed up the process through using more replicas in parallel. However, considering the reliability of communication between the replicas and PC as well as the influence of replicas on the whole swarm behaviors, a limited number of replicas is recommended.

We have proved that *Turing Learning* is robust to failure of individuals in the mixed group. That is, even if some individuals fail during the trials, it won't bias the models obtained. This is another advantage of our *Turing Learning* method, as other metric-based system identification approaches may be biased due to the abnormal behavior of even a single agent.

# 5 Inferring Individual Behaviors Through Interactive Turing Learning

## 5.1 Introduction

In the previous chapters, we demonstrated how *Turing Learning (TL)* can be used for inferring swarm behaviors through observation. In fact, this is based on an implicit assumption that the behavioral repertoire of agents in the swarm could be fully revealed through observation. From the perspective of system identification, the target system has high observability. However, when the target system has low observability [23], the agent's behavioral repertoire may not be fully revealed only through observation (e.g., randomly generating the sequences of inputs). The machine needs to interact with the agent to explore its hidden information. Based on this idea, in this chapter we aim to infer agent behaviors that have low observability. In particular, we extend *Turing Learning* in Chapter 3 with interactive capability.

Observation and interaction are widely adapted by ethologists when investigating the behavior of animals [24, 25, 26]. When investigating animals' behavior in their natural habitats, passive observation is preferable as it is difficult to change the environmental stimuli. In this case, inferring the causal relations between the animal's behavior and its environmental stimuli may become challenging, since the stimuli are not under the observer's control. However, when the experiments are carried out in the indoor laboratories, it is possible to change/control the stimuli to interact with the animals under investigation in a meaningful way. In [26], in order to investigate the cause of the dung beetle dance, biologists designed various experiments to interact with it and learn how it adapts to the environmental changes (e.g., appearance of disturbance or obstacles).

In this chapter, we investigate whether a machine could automatically infer the agent behaviors (with low observability) through controlled interactions using the extended

*Turing Learning* method. To validate this, two case studies are presented: deterministic agent behavior (Section 5.3.1) and stochastic behavior (Section 5.4.1). In these two case studies, the machine is able to control the agent's environmental conditions, which in this work corresponds to the intensity of the ambient light. At the same time, it is capable of simulating the actions of the agent. The learning result is a model of the agent that captures its behavior in relation to the environmental stimulus.

This chapter is organized as follows. Section 5.2 describes the methodology, illustrating how *Turing Learning* is extended to have interactive capability. The deterministic and stochastic behaviors under investigation are presented as two case studies (Sections 5.3 and 5.4). Section 5.3.1 describes the deterministic behavior. Section 5.3.2 presents the simulation setup. Section 5.3.3 presents the results of inferring the deterministic behavior, including analysis of the evolved models, the coevolutionary fitness dynamics, analysis of the evolved classifiers, a study showing the method's sensitivity to noise, and a comparison of *Turing Learning* with two metric-based methods—a single-population evolutionary approach and an approach based on coevolution of inputs and models. Section 5.4.1 describes the stochastic behavior (using a state machine) for the general case. Section 5.4.2 presents the simulation setup for inferring the stochastic behavior. Section 5.4.3 and 5.4.4 present the obtained results for the case of 2 states and 3 states, respectively. Section 5.5 summaries the chapter.

## 5.2 Methodology

In this chapter, we extend *Turing Learning* described in Chapter 3 with interactive capability. The basic idea is the same, that is, the method is comprised of two populations: one of models, and one of classifiers, which coevolve with each other competitively. The fitness of the classifiers depends solely on their ability to distinguish the behavior of the models from the behavior of the agent. The fitness of the models depends solely on their ability to mislead the classifiers into making the wrong judgment, that is, classifying them as the agent. In this following, we will describe the implementation that is related to the work in this chapter.

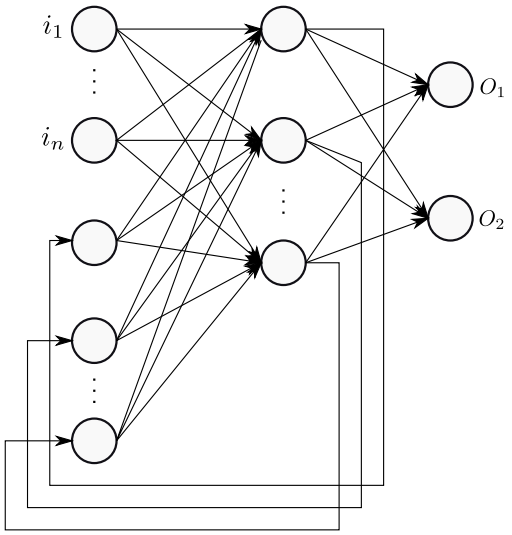



Figure 5.1: This diagram shows the structure of the classifiers used in this chapter. It is a recurrent Elman neural network [27] with $i$ input neurons, $h$ hidden neurons, and two output neurons ($O_1$ and $O_2$). $O_1$, which controls the stimulus, is fed back into the input; $O_2$ is used for making a judgment. Two bias neurons with a constant input of 1.0 are connected to each neuron of the hidden and output layers. See text for details.

### 5.2.1 Models

The models are represented by a set of parameters that govern the rules of the agents. The details of these parameters will be described in Section 5.3.1 and Section 5.4.1. As we have argued in the previous chapters, explicit representation (i.e., evolving only the parameters) and knowing the ground truth (i.e., real parameters) makes it feasible for us to objectively gauge the quality of the models obtained.

### 5.2.2 Classifiers

Figure 5.1 shows the structure of the classifiers. The structure is similar to the one used in Chapter 3 (see Figure 3.1). However, the classifier (neural network) has an additional output that is used to control an environmental stimulus. Moreover, the environmental stimulus is also fed into the classifier as an additional input. In the following, we will explain how the classifier works.

Suppose the agent responds to the level of light intensity in the environment. We assume

that the classifier can observe the agent's speed. The classifier (network) has two inputs. One is the light intensity in the environment at time step $t$, $I^{(t)} \in [0, 1]$, and the other is the speed $v^{(t)}$ of the agent. In order to make this setup more feasible to implement, it is assumed that the system cannot directly measure the speed of the individual, but rather its position. The speed of the individual for the classifier's input is then calculated by subtracting the previous estimated position from the current estimated position, and dividing the resulting number by the time interval between two measurements.

In order to make a judgment between a model and the agent, the classifier observes the behavior (speed) over a period of time. In addition, the classifier is also in control of the light intensity in the individual's environment. At time $t = 0$, the value of the light intensity is chosen randomly with a uniform distribution in the range $[0, 1]$. The neural network is then updated, using $I^{(0)}$ and $v^{(0)}$. The value of the light intensity for the next time step is obtained from the classifier's output neuron $O_1$, and the process repeats. After having iterated through all the time steps (a single trial), the final value of output neuron $O_2$ is used to make a judgment: the network decides on a model if $O_2 < 0.5$, and on the agent if $O_2 \geq 0.5$. The memory (value of hidden neurons) of the classifiers is reset at the end of every trial.

### 5.2.3 Optimization Algorithm

The algorithm used here is based on a $(\mu + \lambda)$ evolution strategy with self-adaptive mutation strengths [28, 29]. It is the same as the one used in Chapter 3. For the details of the implementation, see Section 3.1.1.3.

### 5.2.4 Fitness Calculation

Suppose the population sizes for the model and classifier are $M$ and $C$, respectively. The fitness of each model is obtained by evaluating it with each of the classifiers in the competing population ($C$ in total). For every classifier that wrongly judges the model as being the agent, the model's fitness increases by $\frac{1}{C}$. The final fitness is in $[0, 1]$.

The fitness of each classifier is obtained by using it to evaluate (i) each model in the competing population ($M$ in total) once, and (ii) the agent $L$ times with different initial light intensities. For each correct judgment of the model and the agent, the classifier's fitness increases by $\frac{1}{2 \cdot M}$ and $\frac{1}{2 \cdot L}$, respectively. The final fitness is in $[0, 1]$.

Table 5.1: This table shows the change of the agent's speed (shown in Figure 5.2), for an example sequence of light levels.

| **level** | $M$ | $H$ | $L$ | $H$ | $L$ | $L$ | $L$ | $H$ | $H$ | $L$ | $L$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **speed** | $k(I-0.5)$ | $c_1$ | $c_2$ | $c_1$ | $c_2$ | $\alpha_1^1 c_2$ | $\alpha_1^2 c_2$ | $c_1$ | $\alpha_1^1 c_1$ | $c_2$ | $\alpha_2^1 c_2$ |

| $L$ | $H$ | $H$ | $L$ | $L$ | $H$ | $H$ | $H$ | $M$ | $H$ | $L$ | $L$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\alpha_2^2 c_2$ | $c_1$ | $\alpha_2^1 c_1$ | $c_2$ | $\alpha_3^1 c_2$ | $c_1$ | $\alpha_3^1 c_1$ | $\alpha_3^2 c_1$ | $k(I-0.5)$ | $c_1$ | $c_2$ | $\alpha_1^1 c_2$ |

## 5.3 Case Study One

To validate our method, we present two case studies: one with deterministic behaviors and one with stochastic behaviors. The behaviors to be identified in this chapter were chosen to serve as a proof of concept study. While it may loosely correspond to how some animals react to the stimuli in their environment, it is not intended to mimic any specific animal. In these behaviors, non-trivial interaction with the agent is critical for leading the agent to reveal all of its behavioral repertoire.

### 5.3.1 Deterministic Behavior

We simulate a one-dimensional environment in continuous space. The simulation advances in discrete time steps $t \in \{0, 1, 2, \dots\}$. The (ambient) light intensity in the environment, $I$, can be varied continuously between 0 and 1. The agent distinguishes between three levels of light intensity, low ($0 \leq I < I_L$), medium ($I_L \leq I \leq I_H$), and high ($I_H < I \leq 1$). Hereafter, these levels will be referred to as $L$, $M$, and $H$.

If the light intensity is at level $M$ at time $t$, the speed of the agent, $s^{(t)} \in \mathbb{R}$, varies linearly with $I^{(t)}$ as:

$$s^{(t)} = k\left(I^{(t)} - 0.5\right), \tag{5.1}$$

where $k$ is a constant.

We define two constants: $c_1 = k\,(I_H - 0.5)$ and $c_2 = k\,(I_L - 0.5)$. The deterministic behavior under investigation is shown in Figure 5.2. The agent's behaviors for levels $L$ and $H$ depend on the previous levels of light intensity (i.e. the agent has memory). The

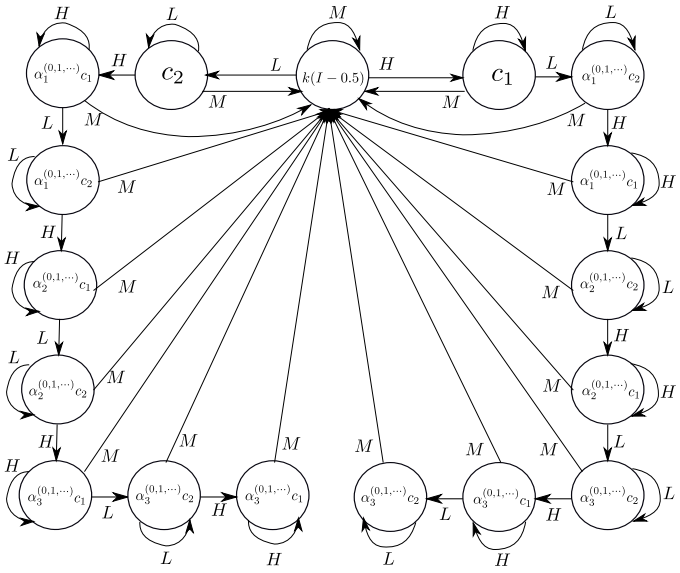



Figure 5.2: The deterministic behavior under investigation. It shows how the agent responses to the level of light intensity ($L$, $M$ and $H$) in its environment. Each state represents the agent's speed. See text for details.

two behaviors are symmetrical to each other. Here, we will describe the behavior for level $L$; the behavior for level $H$ is obtained by exchanging $L$ with $H$ and $I_L$ with $I_H$ in the following description.

When the light intensity is at level $L$, the agent's default speed is $k\,(I_L - 0.5)$ and remains at that value as long as the light intensity remains at level $L$. If the light intensity is at level $H$ (for any number of time steps), and then immediately changes to level $L$, and remains at that level for at least one more time step, then the agent's speed decays exponentially with a rate of $\alpha_1$: that is, in the first time step that the light intensity is at level $L$, the agent's speed is $\alpha_1^0 k\,(I_L - 0.5)$; it then changes to $\alpha_1^1 k\,(I_L - 0.5)$, $\alpha_1^2 k\,(I_L - 0.5)$, and so on as long as the light intensity remains at level $L$. The agent now registers that $\alpha_1$ has been activated. If another $H \to L \to L$ sequence is observed, the agent's speed now decays exponentially with a rate of $\alpha_2$. If further $H \to L \to L$ sequences are observed, the exponential decay rate becomes and remains at $\alpha_3$. Note that at any time, if the agent observes a light intensity at level $M$, its speed is proportional to the light intensity, as shown in Eq. 5.1, and it forgets all its past observations of the light intensity (i.e., the memory of the agent is reset).

The behavior of the agent can thus be represented by five cases; one where the agent's response to the light intensity is *proportional* (which occurs whenever the light intensity

is at level $M$); one where the agent's response is *constant* (i.e. the agent's speed reaches the lower and upper saturation values ($c_1$ or $c_2$) as shown in Figure 5.2); and three where the agent's response *decays exponentially* with the decay rates $\alpha_1$, $\alpha_2$ and $\alpha_3$, respectively.

Table 5.1 shows an example sequence of light levels, along with the corresponding speed of the agent (i.e., the speed shown in Figure 5.2).

Here, $I_L$ and $I_H$ are set to 0.1 and 0.9 respectively. $k$ is set to 1.25; hence, the lower and the upper saturation values of the speed are $k\,(I_L - 0.5) = -0.5$ and $k\,(I_H - 0.5) = 0.5$. The exponential decay rates are set to: $\alpha_1 = 0.8$, $\alpha_2 = 0.4$, $\alpha_3 = 0.2$. Thus, in each case, the agent's speed decays exponentially towards zero. Note that these values ($k$, $\alpha_1$, $\alpha_2$ and $\alpha_3$) are chosen arbitrarily and the coevolutionary algorithm is not sensitive to them.

The system identification task is to learn these four parameters ($k$, $\alpha_1$, $\alpha_2$ and $\alpha_3$) of the agent. It is worth while to mention again that to learn $\alpha_3$, the machine needs to at least output the sequence: $H \to L \to L \to H \to L \to L \to H \to L \to L$ or $L \to H \to H \to L \to H \to H \to L \to H \to H$. Since $I_L$ and $I_H$ are set to a relatively small and large value in $[0, 1]$ respectively, it is very unlikely to randomly generate such sequences.

### 5.3.2 Simulation Setup

We use three setups for the *Turing Learning* method. The setup, in which the classifier is in control of the light intensity in the agent's environment, is hereafter referred to as the "Interactive" setup. In order to validate the advantages of the interactive approach, we compared it against the situation where the classifier only observes the agent in a passive manner; that is, it does not control the light intensity in the environment. We considered two such setups: in the first setup (hereafter, "Passive 1") the light intensity is randomly chosen from the uniform distribution in $[0, 1]$, in every time step. In the second setup (hereafter, "Passive 2"), the light intensity is randomly chosen only after certain number of time steps (in this setup the number is chosen to be 10). All other aspects of these two setups are identical to the "Interactive" setup.

The population sizes of the models and classifiers are chosen to be 100, respectively. We performed 100 coevolution runs for each setup. Each coevolution run lasts 1000

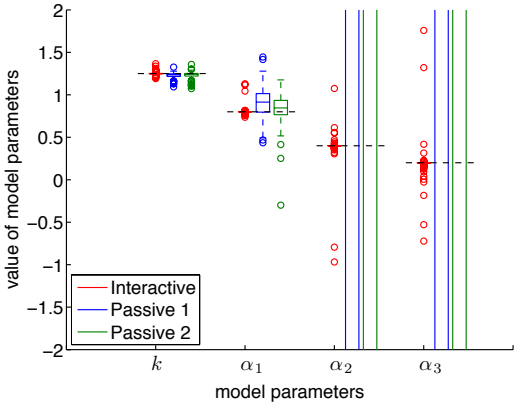



Figure 5.3: This plot shows the distributions of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation in the coevolutions. Each box corresponds to 100 coevolution runs. The dotted lines correspond to the values of the four parameters that the system is expected to learn (i.e. those of the agent). From top to bottom, these are 1.25, 0.8, 0.4, 0.2, respectively. Note that in order to zoom in on the relevant range, some boxes and outliers are omitted from the plot.

generation. In one generation, each classifier conducts 100 trials on the agent. In each trial, the classifier observes the agent for 10 s at 0.1 s intervals, that is, a total of 100 data points.

### 5.3.3 Results

#### 5.3.3.1 Analysis of Evolved Models

Figure 5.3 shows a box plot with the distributions of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation over 100 coevolution runs of the three setups. The passive coevolutions are able to evolve the parameters $k$ and $\alpha_1$ with a reasonable accuracy; however, they are not able to evolve $\alpha_2$ and $\alpha_3$. In the Passive 1 coevolution, the relative errors of the medians of the four evolved parameters $(k, \alpha_1, \alpha_2, \alpha_3)$ with respect to those of the agent are 1.2%, 14.3%, $7.8 \times 10^4$%, and $2.3 \times 10^5$%, respectively. The Passive 2 coevolution leads to similarly large relative errors in the evolved values of $\alpha_2$ and $\alpha_3$. This phenomenon can be explained as follows. If the light intensity changes randomly (either every time step, or every ten time steps), it is unlikely that the $H \to L \to L$ and/or $L \to H \to H$ sequences will occur enough times, without a

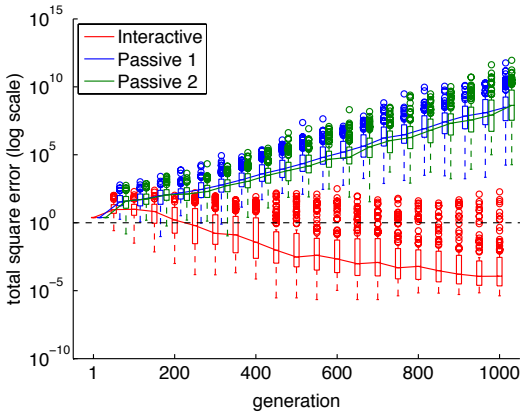



Figure 5.4: This plot shows the total square errors of the evolved model parameters compared to those of the agent over generations. The models with the highest subjective fitness in each generation are selected. Each box corresponds to 100 coevolution runs, and the solid lines correspond to the median error.

level of $M$ in between, such that the classifiers can observe the effects of $\alpha_2$ and $\alpha_3$. Therefore, the classifiers do not evolve the ability to distinguish the behavior of models from the behavior of the agent with respect to these two parameters, and in turn, these parameters do not converge to their true value in the model population.

In contrast to the passive coevolutions, the Interactive coevolution is able to evolve all the four parameters with a good accuracy. The relative median errors are 0.024%, 0%, 0.025% and 0.15% for $k$, $\alpha_1$, $\alpha_2$ and $\alpha_3$ respectively. This implies that by the $1000^{\text{th}}$ generation, the classifiers have learned how to control the pattern of the light intensity in such a way that they can distinguish models from the agent based on the effect of any of the four parameters. Therefore, in order to compete for being selected in the population, the models are evolved to behave like the agent in every aspect.

#### 5.3.3.2 Coevolutionary Dynamics

Figure 5.4 shows the dynamics of the coevolutionary algorithms. The horizontal axis shows the generation, whereas the vertical axis shows the total square error of the model parameters, that is, the sum of the square errors in the four parameters (of the model with the highest subjective fitness in each generation) with respect to their true values. In the case of the Interactive coevolution, the median error starts to reduce after around the $100^{\text{th}}$ generation, and keeps decreasing until the last generation where it reaches a

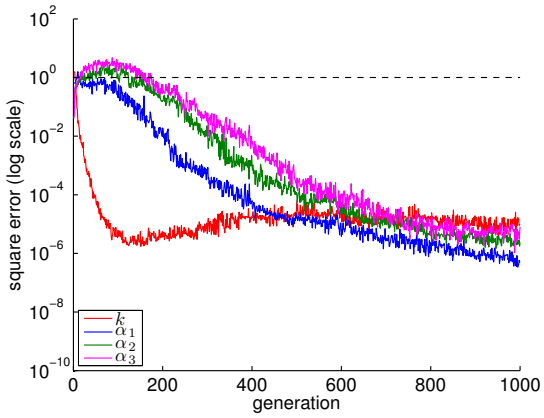
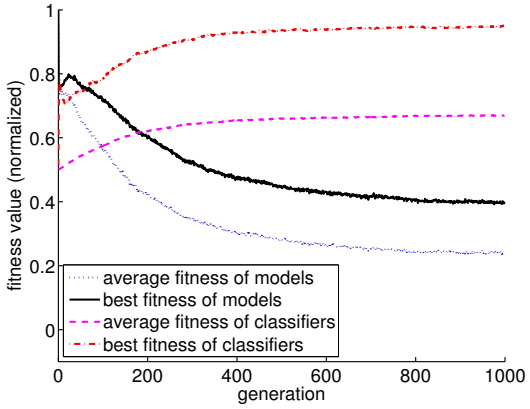



Figure 5.5: This plot shows how the square error in the individual model parameters changes over the generations in the Interactive coevolution. The curves correspond to median values from 100 coevolution runs.

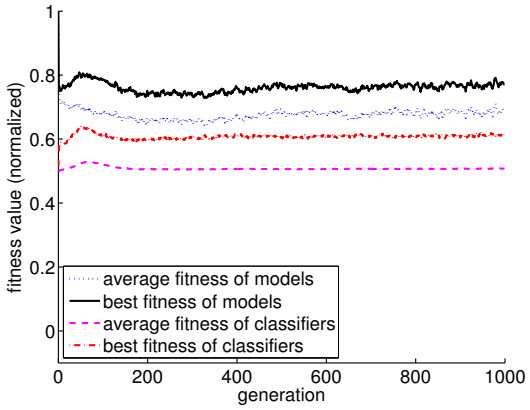
value of $10^{-4}$. In contrast, in the case of the passive coevolutions, not only does the median error not decrease, but it increases to a value of $10^8$ by the $1000^{\text{th}}$ generation.

We now analyze how the four individual parameters evolve during the course of the Interactive coevolution, which is the only fully-successful setup. The plot shown in Figure 5.5 reveals how the learning proceeds in the coevolution. Parameter $k$ is the first to be learnt, followed by $\alpha_1$, while parameters $\alpha_2$ and $\alpha_3$ take a longer time to approximate the true values. This means that the classifiers first learn to distinguish models from the agent on the basis of $k$ and $\alpha_1$. This ability of the classifiers drives the model population to evolve $k$ and $\alpha_1$, in order to mislead the classifiers. Eventually, the classifiers also learn to exploit the effects of $\alpha_2$ and $\alpha_3$ in order to make the right judgment; thereby driving the model population to evolve these two parameters accurately. After about the $600^{\text{th}}$ generation, the learning of the four parameters proceeds with approximately identical rates.

In order to analyze why the Interactive coevolution is successful while the passive ones are not, we can look at the dynamics of the subjective fitnesses of the classifiers and the models (as defined in Section 5.2.4) during the course of the coevolution. As both of the passive coevolutions fail to converge, we present the analysis of fitness dynamics only for Passive 1 coevolution (the other case was found to have similar dynamics). Figure 5.6 shows the fitness dynamics of the Interactive and the Passive 1 coevolutions. In the case of the Interactive coevolution (see Figure 5.6(a)), the average fitness of the classifiers starts off at 0.5, which means that the classifiers make judgments that are no better




Interactive coevolution




Passive 1 coevolution

Figure 5.6: This plot shows the subjective fitness (normalized) of the classifiers and the models in (a) the Interactive coevolution, and (b) the Passive 1 coevolution. The curves show the average fitness across 100 coevolution runs.

than random judgments. However, the classifiers quickly improve in fitness, which in turn causes the fitness of the models to decrease. This increases the selective pressure on the models. After about 600 generations both the fitness of classifiers and models reach a steady state, which according to Figure 5.5 corresponds to the region where the four parameters evolve with virtually identical rates. In the case of the Passive 1 coevolution (see Figure 5.6(b)), the average fitness of the classifiers also starts off at 0.5. In the first few generations, this increases slightly, because the classifiers learn how to distinguish models from the agent on the basis of parameters $k$ and $\alpha_1$. However, the models quickly adapt to this new ability of the classifiers. Now, as the classifiers are unlikely to have the

opportunity to observe the effects of $\alpha_2$ and $\alpha_3$, their average fitness returns to 0.5. This leads to a disengagement phenomenon, in which there is no more meaningful selection in the model population, therefore leading the parameters $\alpha_2$ and $\alpha_3$ to drift, the effect of which can be seen in Figure 5.4.

#### 5.3.3.3 Analysis of Evolved Classifiers

In this section, we analyze the evolved classifiers in the Interactive coevolution. The model described in Section 5.2.1 is defined by four parameters: $\{k, \alpha_1, \alpha_2, \alpha_3\}$. In order to evaluate the quality of the evolved classifiers we performed a grid search over the space of the amount of disturbance (i.e. noise) injected into each of the four parameters. We used 11 noise magnitude settings per parameter, $M \in \{0, 0.1, \ldots, 1\}$, with the noise being added to the parameters as follows:

$$p' = p(1 + \mathcal{U}(-M, M)), \tag{5.2}$$

where $p \in \{k, \alpha_1, \alpha_2, \alpha_3\}$ represents any of the four parameters, and $\mathcal{U}(-M, M)$ denotes a uniform distribution on the interval $(-M, M)$. Note that $M = 0$ corresponds to no noise, whereas $M = 1$ means that the noisy value of the parameter can be between 0 and twice its actual value.

For the sake of simplicity, we only analyzed the classifiers with the highest subjective fitness in the last generation of the 100 coevolutions. For each of the 100 classifiers, and for each combination of noise magnitudes, we conducted 100 trials. In other words, we conducted $100 \cdot 11^4 \cdot 100 = 146,410,000$ trials in total. In each trial, we modulated the agent's parameters according to Eq. 5.2. Each trial was run for T = 100 time steps (the same setting used within the coevolutions). For each combination of magnitudes employed, the overall performance $U$ was computed as the sum of the final judgments of the classifier in each trial, divided by the total number of trials, $N$:

$$U = \sum_{i=1}^{N} J_i / N, \tag{5.3}$$

where, $J_i \in \{0, 1\}$ is the final judgment of the classifier in trial $i$, and $N$ is the total number of trials conducted. Note that $J_i = 0$ and $J_i = 1$ imply that the classifier has judged the behavior as being that of a model and the agent, respectively.

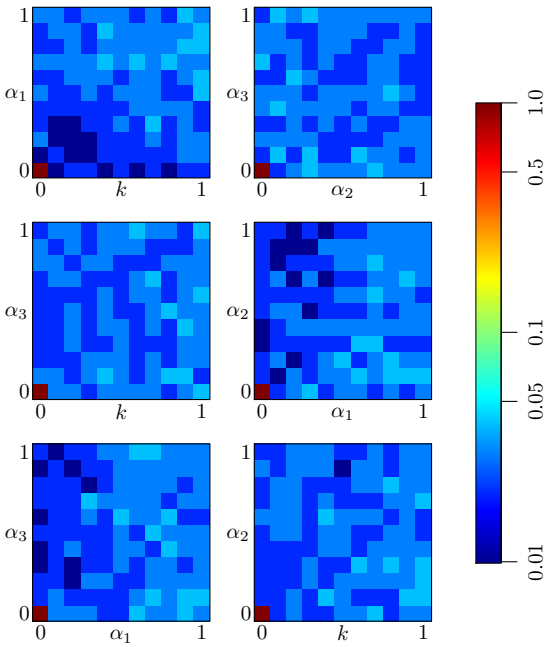



Figure 5.7: This figure shows the landscape of $U^*$ (log scale) for the overall best classifier over the six sub-spaces with two parameters as degrees of freedom (from 100 coevolution runs). Each axis in each plot ranges between 0 and 1, corresponding to the minimum and maximum magnitude of noise added into each parameter respectively. See text for details.

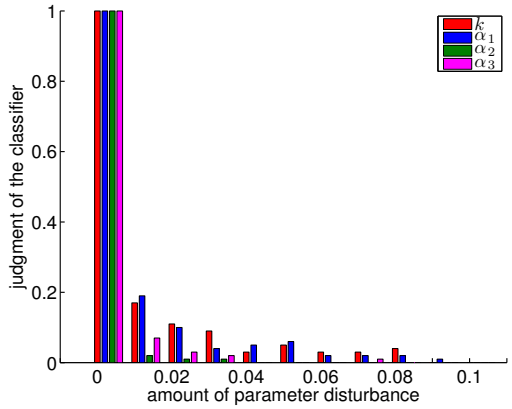



Figure 5.8: This plot shows the average judgment [$U$, see Eq. 5.3] of the best evolved classifier over 100 trials when the magnitude of noise added into each parameter of the agent is within 0.1.

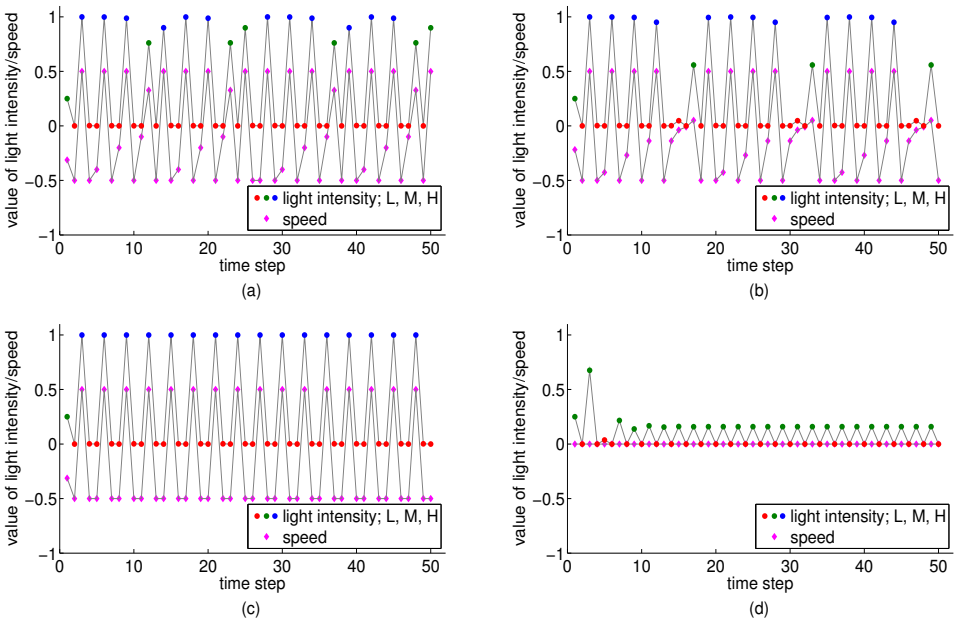



Figure 5.9: This plot shows the light intensity sequences as output by the overall best classifier (circular points), along with the corresponding speeds of the agents (diamond-shaped points) in four trials conducted on different agents: (a) the agent; (b) a model similar to the agent, generated randomly according to Eq. 5.2 with $M = 0.5$; (c) a model whose speed is linear to the light intensity; and (d) a static model, whose speed is always 0 (i.e. $k = 0$). The colors of the circular points (red, green, blue) correspond to light intensities at levels $L$, $M$ and $H$, respectively.

In order to find the overall best classifier from the 100 classifiers analyzed, we defined the following metric:

$$W = [1 - U(0,0,0,0)] + \frac{1}{\Omega} \sum_{\substack{i,j,k,\ell \in \{0,0.1,\ldots,1\} \\ i^2+j^2+k^2+\ell^2 \neq 0}} \sum \sum \sum \sum (i+j+k+\ell)\, U(i,j,k,\ell)\,, \tag{5.4}$$

where $\Omega = 29282$ is the maximum value that the quadruple sum can achieve (i.e. if all the $U$'s are equal to 1). The first term in Eq. 5.4 penalizes the classifier if $U < 1$ when there is no noise on the parameters; they are thus undisturbed and identical to the ones

of the agent. The second term penalizes the classifier if $U > 0$ for any non-zero noise combination, with increasing penalties being applied to higher noise magnitudes. The normalization of the second term by $\Omega$ serves to make the two terms contribute equally to $W$, which can take values in $[0, 2]$. Note that the minimum value of $W = 0$ can only be achieved by the perfect classifier, that is, one that outputs 1 if $i = j = k = \ell = 0$ and 0 otherwise. In our case, the best classifier achieved a value of $W = 3.7 \cdot 10^{-3}$.

The performance landscape of the models is 5-dimensional (4 model parameters plus performance measure), and cannot be visualized directly. Therefore, we considered each combination of two model parameters ($\binom{4}{2} = 6$ combinations) as a sub-space, and for each point in this sub-space, we calculated the performance measure as the maximum value over the sub-space spanned by the remaining two parameters. For instance, on the sub-space $(k, \alpha_1)$, the performance measure $U^*(k, \alpha_1)$ was calculated as:

$$U^*(k, \alpha_1) = \max_{\alpha_2, \alpha_3} U(k, \alpha_1, \alpha_2, \alpha_3). \tag{5.5}$$

Note that for all the points except $(0, 0, 0, 0)$, Eq. 5.5 corresponds to the worst-case scenario for the classifiers, because the value of $U$ for the ideal classifier at these points is 0. For the point $(0, 0, 0, 0)$, the value of $U$ for the ideal classifier is 1.

Figure 5.7 shows the landscape of $U^*$ (log scale) for the overall best classifier over the six sub-spaces. When interacting with the agent (i.e. $i = j = k = \ell = 0$), the output of the classifier is always 1. This corresponds to the point $U^*(0, 0)$ in the six sub-spaces of Figure 5.7 (here only the maximum is shown). When interacting with the models (i.e. when the parameters of the agent are perturbed), for any combination of noise magnitudes, the average output of the classifier is below 0.05 for the six sub-spaces.

In order to further analyze how sensitive the overall best classifier is when the parameters of the agent are only slightly perturbed, we set the maximum magnitude of noise added to each parameter to 0.1, and used a resolution of 0.01. In this evaluation, we only injected noise into one parameter at a time. For each noise magnitude, the classifier was evaluated in 100 trials (with different initial light intensities, and randomly-generated noise), and the average performance measure was computed according to Eq. 5.3.

Figure 5.8 shows the average judgment [$U$, see Eq. 5.3] of the best classifier, which was evaluated in 100 trials. With increasing noise magnitudes, the average judgment of the classifier approaches zero, which means that the classifier can identify the agents as models more consistently. The classifier has different sensitivities to the four parameters;

the effects of noise on $\alpha_2$ and $\alpha_3$ on its judgment are higher than those of $k$ and $\alpha_1$. For instance, in the case of $\alpha_2$, even with $M = 0.01$, the classifier correctly judges the behavior as being that of a model in 98% of the trials. As we have seen, $k$ and $\alpha_1$ are the first two parameters to be identified in the coevolution. $\alpha_2$ and $\alpha_3$ are addressed in the later generations of the coevolutionary process, and it seems that the classifier is more sensitive to these two parameters.

We now investigate how the overall best classifier interacts with the agents. We conducted four trials with four different agents: (a) the agent; (b) a model similar to the agent, generated randomly according to Eq. 5.2 with $M = 0.5$; (c) a model whose speed is linear to the light intensity without exponential decay (i.e. $k = 1.25$, $\alpha_1 = \alpha_2 = \alpha_3 = 1$); and (d) a static model, whose speed is always 0 (i.e. $k = 0$). In each trial, the initial light intensity was set to 0.25. The trials lasted for 100 time steps.

Figure 5.9 shows the sequences of light intensity output by the classifier, along with the speed of the agents in the four trials, for the first 50 time steps (we observed that the last 50 time steps constitute a repetition of the first 50). As we can see, the classifier outputs different sequences of light intensity in order to interact with the different agents. The greater the difference between a model and the agent, the more varied is the sequence of light intensities that the classifier outputs when interacting with it as compared to the one it outputs when interacting with the agent. For the agent, the classifier repeatedly outputs a $H \rightarrow L \rightarrow L$ sequence, in order to fully reveal the behavior (this is analyzed in detail in the following paragraph). For the model that is similar to the agent (see Figure 5.9(b)), the sequence of light intensities is similar to the one produced for the agent, but it is not identical. Interestingly, for the model without exponential decay (see Figure 5.9(c)), the classifier produces the $H \rightarrow L \rightarrow L$ sequence even more often than it does for the agent, but it does not set the light intensity to level M. For the static model (see Figure 5.9(d)), the classifier never outputs a $H \rightarrow L \rightarrow L$ sequence; instead, it outputs a sequence that alternates between $M$ and $L$.

We now focus on analyzing the behavior of the agent in Figure 5.9(a). Initially, the classifier outputs the sequence $L \rightarrow H \rightarrow L \rightarrow L \rightarrow H \rightarrow L \rightarrow L \rightarrow H \rightarrow L \rightarrow L \rightarrow M$, which means that by the $12^{\text{th}}$ time step, it has already observed the effects of all four parameters. Interestingly, the classifier then (essentially) repeats this sequence and thus observes the effect of all the parameters for another seven times (four repetitions occur in the last 50 time steps, which are not shown in Figure 5.9(a)). We conjecture that the repetitions make the classifier more robust in determining whether the behavior under

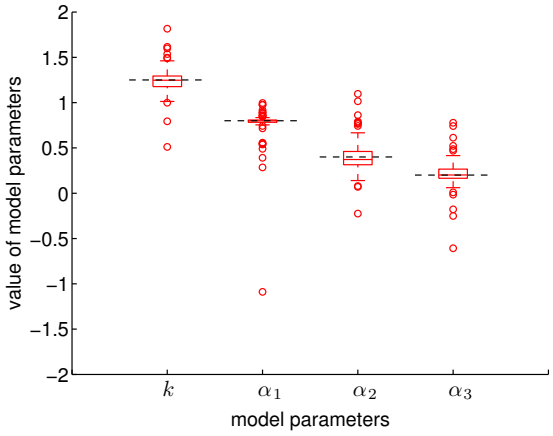



Figure 5.10: This plot shows the distributions of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation of the Interactive coevolution with noise (for a comparison to the case without noise, see Figure 5.3). The dotted lines correspond to the values of the four parameters that the system is expected to learn (i.e. those of the agent). From top to bottom, these are 1.25, 0.8, 0.4, 0.2, respectively.

observation is that of a model or the agent. These results suggest that the classifier succeeds in controlling the level of ambient light in a way that helps distinguish between the agent and the models, revealing all aspects of the underlying behavior. Note that the classifiers are not provided with any prior information about the agent, including its structure. Thus, all knowledge is gained through an evolutionary process that is driven by the accuracy of their judgments.

#### 5.3.3.4 Noise Study

We now consider the situation where, during the coevolutionary process, noise is injected into the agent's behavior, and the agent's positions as measured by the system. This makes the overall setup more realistic as the locomotion of agents and any tracking system will be affected by noise and measurement error. Since the passive coevolutions fail even in the noiseless case, we consider only the Interactive coevolution for the sake of simplicity. We performed 100 coevolutionary runs with the following settings. The light intensity perceived by the agent at time $t$ is obtained by multiplying the actual intensity by a random number generated uniformly in $(0.95, 1.05)$, and capping the perceived intensity to 1 if it exceeds this value. Noise is also applied to the speed of the

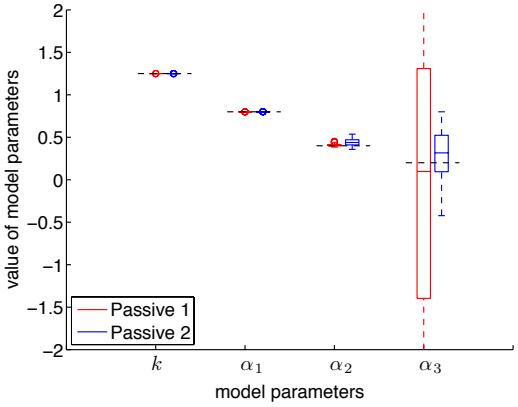



Figure 5.11: This plots shows the distributions of the evolved models with the highest fitness in the $100000^{\text{th}}$ generation in the simple evolutions with a single population. Each box corresponds to 100 evolution runs. The dotted lines correspond to the values of the four parameters that the system is expected to learn (i.e. those of the agent). From top to bottom, these are 1.25, 0.8, 0.4, 0.2, respectively. Note that in order to zoom in on the relevant range, some boxes and outliers are omitted from the plot.

agent by multiplying the original speed with a random number generated uniformly in $(0.95, 1.05)$. Noise on the estimated position on the agent is applied by adding a random number generated from a normal distribution: $\mathcal{N}(0, 0.005)$.

Figure 5.10 shows a box plot with the distributions of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation of the Interactive coevolution with noise. The effect of the noise is to widen the distribution of the evolved parameters across the 100 coevolutionary runs; however, the median values of the evolved parameters are still very close to the true values. Interestingly, the Interactive coevolution does not seem to learn $\alpha_2$ and $\alpha_3$ significantly worse than it does learn $k$ and $\alpha_1$.

#### 5.3.3.5 Using a Single-Population Evolutionary Algorithm

In order to compare *Turing Learning* against a more traditional approach, we used a simple evolution where a single population of models evolves. We call this method SPEA. As there are now no classifiers, an interactive approach is not possible, and thus we conducted 100 evolutionary runs for the Passive 1 and Passive 2 methods of changing the light intensity in the agent's environment. The structure of the evolution

Table 5.2: This table shows a comparison of all approaches. The numbers show the relative errors of the evolved parameters (median values over 100 runs) with respect to the parameters of the agent (in absolute percentage).

| | $k$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ |
|---|---|---|---|---|
| Interactive (TL) | 0.024 | 0 | 0.025 | 0.15 |
| Passive 1 (TL) | 1.2 | 14.3 | $7.8 \times 10^4$ | $2.3 \times 10^5$ |
| Passive 2 (TL) | 0.7 | 5.74 | $1.3 \times 10^4$ | $3.3 \times 10^5$ |
| Passive 1 (SPEA) | 0 | 0 | 1.2 | 48.7 |
| Passive 2 (SPEA) | 0 | 0 | 9.8 | 48.5 |
| CoEAIM | $4.0 \times 10^{-5}$ | $4.9 \times 10^{-5}$ | $1.1 \times 10^{-3}$ | $2.5 \times 10^{-4}$ |

is identical to the sub-algorithms used in the coevolution, except for the fitness evaluation step. Now, in each generation, 100 experiments are performed on the agent using 100 randomly generated intensity patterns. The 100 intensity patterns are used to evaluate a model 100 times. The average square error between the model's and the agent's speed sequences is used as the model's fitness. Each evolutionary run lasts $100,000$ generations. In other words, the number and duration of experiments on the agent is kept the same as that in the coevolutionary approach, as outlined in Section **??**.

Figure 5.11 reveals that the evolution is able to identify parameters $k$, $\alpha_1$, $\alpha_2$, but not $\alpha_3$. Note that, apart from the single-population evolution not being able to consistently identify the parameter $\alpha_3$, they also rely on a pre-defined metric for their operation; in this case, computed as the square error between the model's and the agent's speed sequences.

#### 5.3.3.6 Coevolution of Inputs and Models

In this section, we compare *Turing Learning* with another coevolutionary system identification method, in which inputs and models competitively coevolve. In particular, we use the classifiers to generate sequences of inputs (in this case, light intensity). The models are optimized through minimizing the square error of speed between the agent and models, given the same sequence of inputs generated by the classifiers. The classifiers compete with the models through generating inputs that cause the maximum disagreement between the model's prediction and the agent's output, which is similar to the concept in [14]. Note that in this case the classifiers are only used for generating the inputs and they do not make judgments. We call this method CoEAIM. Figure 5.12

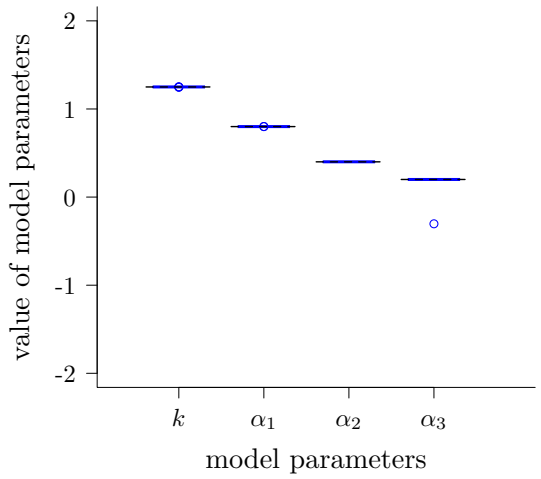



Figure 5.12: This plots shows the distributions of the evolved models with the highest fitness in the $100000^{\text{th}}$ generation in the simple evolutions with a single population. Each box corresponds to 100 coevolution runs. The dotted lines correspond to the values of the four parameters that the system is expected to learn (i.e. those of the agent). From top to bottom, these are 1.25, 0.8, 0.4, 0.2, respectively.

shows the results of CoEAIM. As we can see, the model parameters are also identified with a high accuracy. In other words, there is no 'true' interaction between the agent and classifiers during the experiments in *Turing Learning* when investigating the deterministic behavior. The classifiers only need to learn how to generate a fixed sequence of inputs to extract all the information from the agent. For a comparison of all approaches, see Table 5.2.

In order to further validate and highlight the benefit of *Turing Learning*, we investigate the stochastic behaviors in the following section.

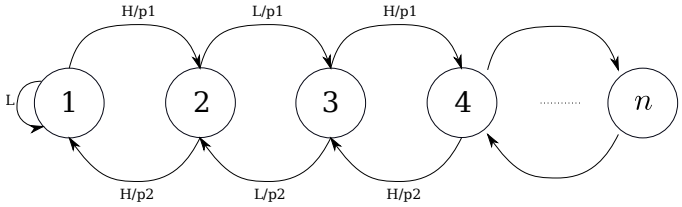



Figure 5.13: A state machine which represents the stochastic behavior of the agent under investigation. The initial state is 1. $p_1$, $p_2$ are probabilities. The ambient light intensity that the agent responds to has two levels: $H$ and $L$. See text for details.

## 5.4 Case Study Two

### 5.4.1 Stochastic Behavior

Stochastic behaviors are widely observed in the animal kingdom. Given the same stimuli, the animal may behave differently. In order to make the animal under investigation reveal all its behavioral repertoire, ethologists sometimes need to interact with the animals in real time and change the stimuli dynamically according to the animal's response. Based on this motivation, in this section we apply *Turing Learning* to infer stochastic behaviors and investigate whether a machine could interact with the agent through dynamically changing the stimulus rather than generating a fixed sequence of inputs.

We will describe the stochastic behavior for the general case. It is represented as a state machine shown in Figure 5.13. Suppose the agent has $n$ states. The agent's behavior depends on the level of the light intensity in the environment and its current state (i.e., the agent has short-term memory). For the initial state (1), if the light intensity is in low level ($L$), the agent keeps staying in state 1; if the light intensity is in high level ($H$), the agent has the probability of $p_1$ to move forward to state 2. For the middle state with even (odd) number, if the level of light intensity is $L$ ($H$), the agent has the probability of $p_1$ to move forward to another state with higher number; otherwise if the light intensity is $H$ ($L$), it has the probability of $p_2$ to move back to a previous state with lower number. When the agent is in the final state, $n$, its behavior is similar to that in the middle state. The only difference is the agent never moves forward. Figure 5.14(a) and Figure 5.14(b) show the agent behavior with 2 and 3 states, respectively. These are the two behaviors to be investigated in the thesis.

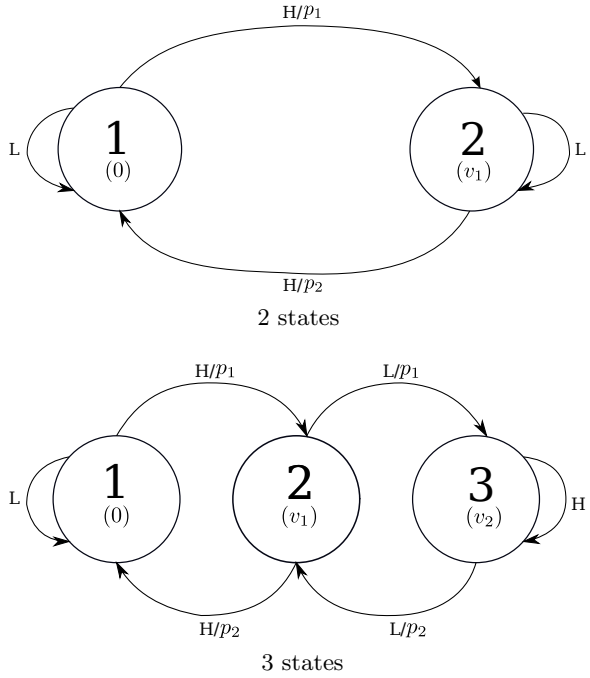



Figure 5.14: The stochastic behaviors with (a) 2 states and (b) 3 states under invesitgation. $p_1$ and $p_2$ are probabilities. For all the other cases (e.g., $H/(1-p_1)$), the agent keeps staying in the same state.

In the agent's behavior, $p_1$ is selected to a low value and $p_2$ is set to a high value. As a consequence of this choice, the agent has a low chance of moving forward to a state with a higher number and thus the higher state has lower observability. The classifiers need to learn how to interact with the agent to infer all its behavioral repertoire.

For a proof of concept study, we assume that the agent moves in one-dimensional space and moves at a consistent speed in each state. Suppose the agent stays static and its initial state is known. The system identification task is to identify the parameters of the agent's other states (i.e., $v_1$ and $v_2$ shown in Figure 5.14) and the two probabilities, $p_1$ and $p_2$. The speed of the agent in each state is chosen arbitrarily. $p_1$ and $p_2$ are chosen to have a value of 0.1 and 1.0. This makes the agent's state with higher number has lower observability.

In order to make the agent stay in the state with higher number as long as possible, the classifiers need to capture the point when the agent starts to move forward from a state with a lower number to a state with a higher number and switches the level of light intensity immediately. This makes it easier for *Turing Learning* to learn the behavior as

the state can be observed for a sufficient time. If the classifiers fail to do that, the agent would immediately move to the previous state with lower number.

For the 2-state machine shown in Figure 5.14(a), $v_1$ is selected to be 0.5; for the 3-state machine shown in Figure 5.14(b), $v_1$ and $v_2$ are selected to be 0.5 and 1.0, respectively. Therefore, the parameters to be identified for the 2-state and 3-state agent behavior are:

$$\mathbf{q}_1 = (v, p_1, p_2) = (0.5, 0.1, 1.0). \tag{5.6}$$

$$\mathbf{q}_2 = (v_1, v_2, p_1, p_2) = (0.5, 1.0, 0.1, 1.0). \tag{5.7}$$

### 5.4.2 Simulation Setup

For *Turing Learning*, we still used three setups: "Interactive", "Passive 1" and "Passive 2", as discussed in Section 5.3.2. We also compared *Turing Learning* with the two metric-based methods (SPEA and CoEAIM) described in Section 5.3.3.5 and Section 5.3.3.6, respectively. Note that for each setup, we added a certain amount of noise into the measurement of speed. This is realized by multiplying the agent's real speed with a random value in $[0.95, 1.05]$ in each time step.

### 5.4.3 Results: Two States

In this section, we compare and analyze the results obtained using the methods discussed in the previous section (5.4.2) for inferring the 2-state agent behavior shown in Figure 5.14(a).

#### 5.4.3.1 Analysis of Evolved Models

Figure 5.15 shows a box plot with the parameters of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation for (a) *Turing Learning*, (b) SPEA, and (c) CoEAIM. Using *Turing Learning*, the system identified all parameters of the agent with good accuracy. For the other two metric-based methods, all the three parameters are

Table 5.3: This table shows a comparison of all approaches for learning the 2-state stochastic behavior shown in Figure 5.14(a). The values show means of AEs (defined in Equation 3.9) of each evolved model parameter with respect to that of the agent.

| | $v_1$ | $p_1$ | $p_2$ |
|---|---|---|---|
| Interactive (TL) | 0.003 | 0.03 | $1.6 \times 10^{-5}$ |
| Passive 1 (TL) | 0.01 | 0.03 | $1.0 \times 10^{-5}$ |
| Passive 2 (TL) | 0.02 | 0.02 | $1.0 \times 10^{-5}$ |
| Passive 1 (SPEA) | 0.47 | 0.9 | 1.0 |
| Passive 2 (SPEA) | 0.47 | 0.9 | 1.0 |
| CoEAIM | 0.47 | 0.9 | 1.0 |

not learned well. Instead, the three evolved parameters converge into three different values: $v_1 \to 0.0$, $p_1 \to 1.0$, $p_2 \to 0.0$. The failure of SPEA and CoEAIM can be explained as follows. As the behavior is stochastic, given the same sequence of inputs the agent would probably exhibit different behaviors. Therefore, quantitatively measuring the difference between the models and agent (e.g., using square error) would not lead the model parameters to converge into their true values. For all methods, the means (standard deviations) of the AEs of each parameter of the evolved model with the highest fitness in the final generation over 30 coevolution runs are shown in Table 5.3. Clearly, *Turing Learning* infers the stochastic behavior significantly better than the two metric-based methods. There is no significant difference among "Interactive", "passive 1" and "passive 2" setups of *Turing Learning* in terms of AEs of the evolved parameters.

In order to show the advantage of "Interactive" setup of *Turing Learning*, we investigate the convergence of model parameters during the evolutionary process. Figure 5.16 shows the convergence of the model parameters over generations for the three setups of *Turing Learning*. As we can see, the evolved model parameters in the "Interactive" setup converge much faster than those in the two passive setups. For the "Interactive" setup, after about 100 generations, all the three parameters converge into their true values. For the "Passive 1", all the parameters converge after about 200 generations, while for the "Passive 2" it takes longer time. In terms of $v_1$, there is much smaller disturbance in the "Interactive" setup than that in the other two setups.

#### 5.4.3.2 Analysis of Evolved Classifiers

In order to investigate why the "Interactive" setup of *Turing Learning* learns the agent behavior much faster than the two passive setups. We post-evaluated how the classifiers interact with the agent during a trial. Figure 5.17 shows how the classifier with the highest subjective fitness in different generations (of a particular coevolution run) interact with the agent in a trial. Similar phenomenon could be observed in other coevolution runs. As shown in the top left of Figure 5.17, the classifier outputs $H$ level and then waits until the agent 'jumps' from state 1 to state 2 (and this process is stochastic), and it immediately switches the light intensity from $H$ to $L$ level in order to make the agent stay in state 2 as long as possible. Therefore, the agent's behavior in state 2 (hidden information) can be observed longer and inferred efficiently. Note that the classifier has learned the good strategy and exhibited such 'intelligent' behavior at the very beginning of the coevolution run (before 50 generations in this case). After 500 generations, the classifier changed the strategy a little bit. Instead of always outputting $H$ level, it keeps switching between $H$ and $L$ level until it observes the agent 'jumps' from state 1 to state 2, and after that it immediately switches the light intensity from $H$ to $L$ level. This is unlikely to happen when generating random sequence of input.

### 5.4.4 Results: Three States

In order to further demonstrate the advantage of "Interactive" setup of *Turing Learning*, this section discusses the results of inferring the 3-state stochastic agent behavior shown in Figure 5.14(b).

#### 5.4.4.1 Analysis of Evolved Models

Figure 5.18 shows the distribution of the evolved models in the $1000^{\text{th}}$ generation for all setups. The "Interactive" setup of *Turing Learning* is the only one that infers all the parameters of the agent with good accuracy. For the two setups using pre-defined metrics (SPEA and CoEAIM), all the parameters are not learned well. Table 5.4 compares the accuracy of each parameter of the evolved model with the highest fitness in the final generation over 30 coevolution runs using all approaches.

Figure 5.19 shows the evolutionary process of the models for the three setups of *Turing Learning*. As we can see, all the model parameters in the"Interactive" setup converged

Table 5.4: This table shows a comparison of all approaches for inferring the 3-state agent behavior shown in Figure 5.14(b). The values show means of AEs (defined in Equation 3.9) of each evolved model parameter with respect to that of the agent.

| | $v$ | $v_2$ | $p_1$ | $p_2$ |
|---|---|---|---|---|
| Interactive (TL) | 0.005 | 0.03 | 0.02 | $2.0 \times 10^{-6}$ |
| Passive 1 (TL) | 0.02 | 0.23 | 0.05 | 0.03 |
| Passive 2 (TL) | 0.02 | 0.47 | 0.08 | 0.13 |
| Passive 1 (SPEA) | 0.47 | 0.97 | 0.9 | 1.0 |
| Passive 2 (SPEA) | 0.47 | 0.98 | 0.9 | 1.0 |
| CoEAIM | 0.46 | 0.96 | 0.67 | 0.89 |

to their true value smoothly within about 200 generations. For the two passive setups, $v_1$, $p_1$ and $p_2$ are learned well, but they still take longer time to converge to their true values than those of the "Interactive" setup. There is dramatic disturbance during the evolutionary process of $v_2$ for the passive setups, and this parameter is not learned well.

#### 5.4.4.2 Analysis of Evolved Classifiers

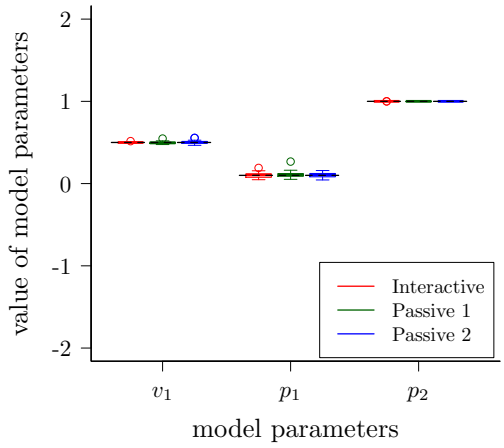
Figure 5.20 shows an example of how the classifier with the highest subjective fitness in different generations (of a particular coevolution run) evolved to interact with the agent. In other coevolution runs, we still observed similar phenomenon. As shown in Figure 5.20, the strategy learned by the classifiers shown in 3 of the 4 sub-figures (corresponding to $100^{\text{th}}$, $200^{\text{th}}$, $1000^{\text{th}}$ generation) is as follows. The classifier outputs $H$ first, and once the agent moves forward from state 1 to state 2, the classifier switches the light intensity into level $L$ and keeps it in that level. As long as the agent moves forward from state 2 to state 3, the classifier switches the light intensity from $L$ to $H$ and keeps the light intensity in that level. Note that the 'best' classifier sometimes lost its ability to interact with the agent in the $500^{\text{th}}$ generation shown in bottom left of Figure 5.20. However, this does not influence the learning process, as long as there are still some other classifiers in the population obtain this interactive ability.

## 5.5 Summary

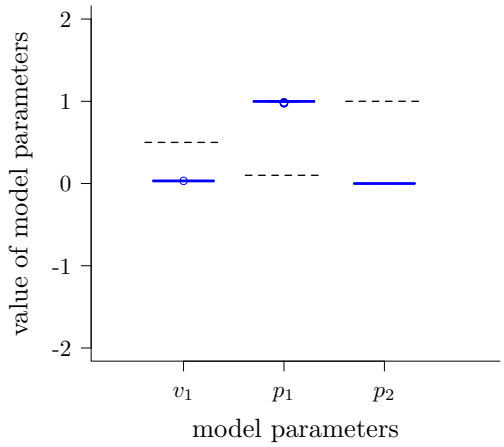
In this chapter, we extend *Turing Learning* with interactive ability to autonomously learn the behavior of an agent. We have shown that, by allowing classifiers to control the stimulus in the agent's environment, the system is able to correctly identify the parameters of relatively complex behaviors. The advantage of *Turing Learning* with interaction is validated using two case studies: stochastic and deterministic behaviors of an agent. In both case studies, the results show that learning through interaction can infer the agent behaviors better or faster than only through passive observation.

When inferring the deterministic behavior, the classifiers have learned to generate a complex sequence of inputs to extract all the hidden information from the agent, which facilitates the learning process. However, generating such sequences in random is unlikely. This makes the SPEA (in which the inputs are generated randomly) fail to infer all the parameters of the agent. However, by coevolving inputs and models (CoEAIM), the system still obtained very good model accuracy in terms of all parameters.

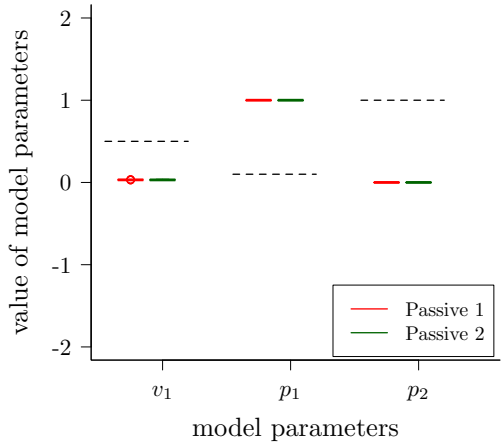
When inferring the stochastic behavior, the advantage of *Turing Learning* with interaction is highlighted. It performs significantly better than the two metric-based methods (SPEA and CoEAIM) in inferring stochastic behaviors. In the "Interactive" setup of *Turing Learning*, the classifiers learned to dynamically interact with the agent, which leading to infer all the agent's parameters in a very efficient way. The results suggest that a machine could also exhibit such intelligent behavior as observed in human behavior, and this could pave the way to further development of machine intelligence.




Turing Learning

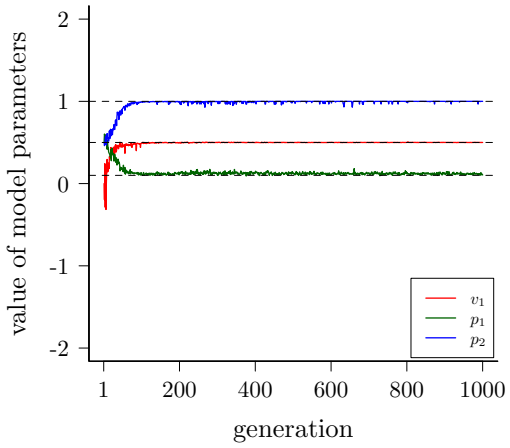



SPEA

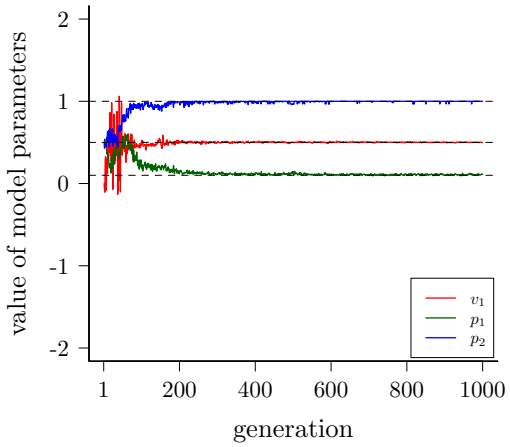



CoEAIM

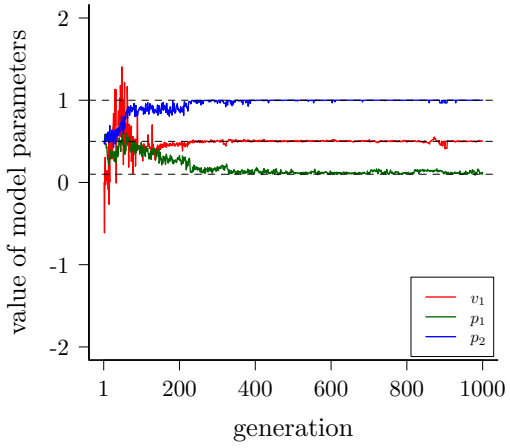
Figure 5.15: This plot shows the distributions of the evolved models with the highest subjective fitness in the 1000$^{\text{th}}$ generation in the coevolutions. Each box corresponds to 30 coevolution runs. The dotted lines correspond to the values of the three parameters that the system is expected to learn (i.e. those of the agent). See text for details.




Interactive




Passive 1




Passive 2



Figure 5.16: Evolutionary process of the evolved model parameters for (a) “Interactive”, (b) “Passive 1” and (c) “Passive 2” setups of the metric-free method when learning the 2-state agent behavior. Curves represent mean values across 30 coevolution runs. Dotted black lines indicate true values.

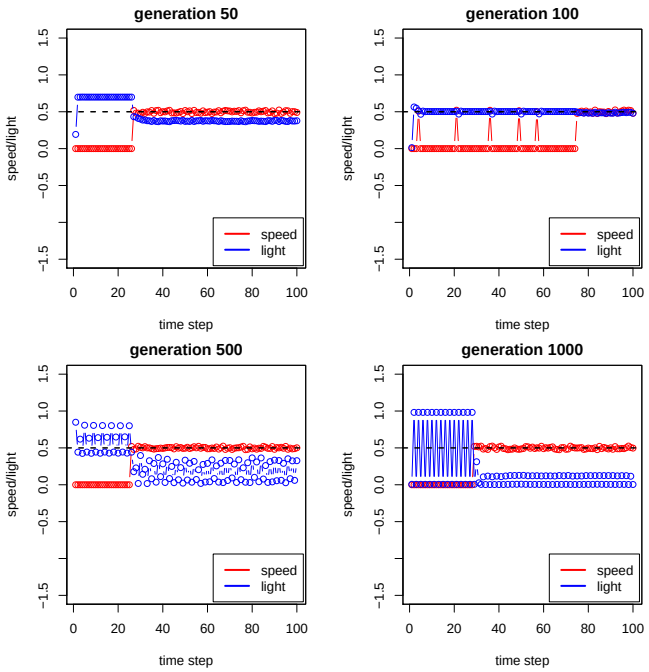
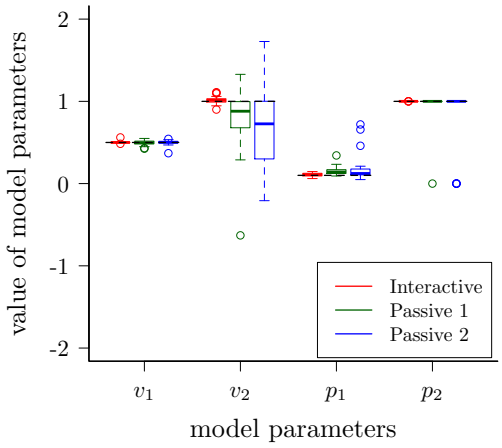
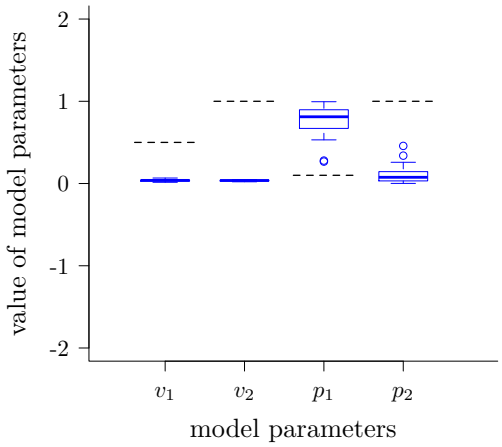


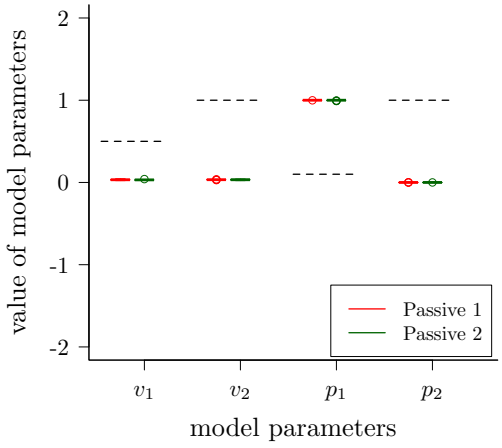

Figure 5.17: This plot shows an example of how the classifier with the highest subjective fitness in different generations (of a particular coevolution run) dynamically change the light intensity to interact with the 2-state agent during a trial.




Turing Learning

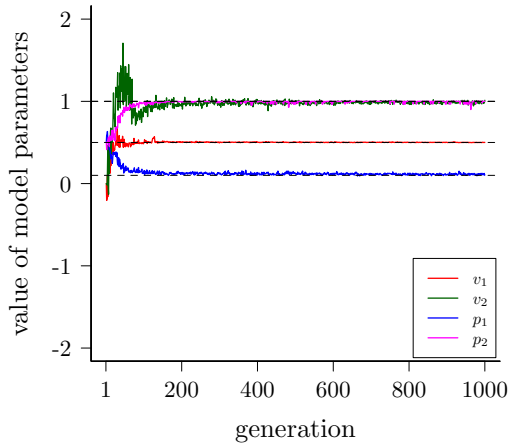



SPEA

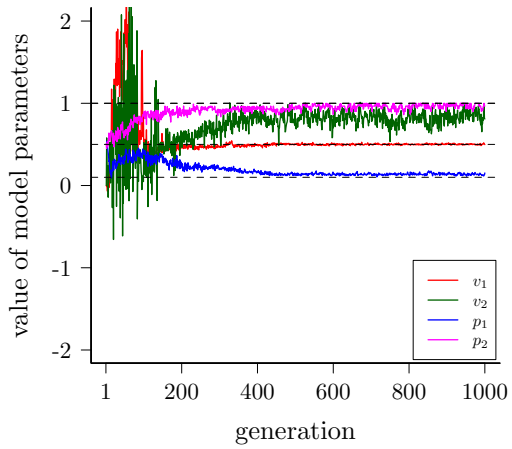



CoEAIM

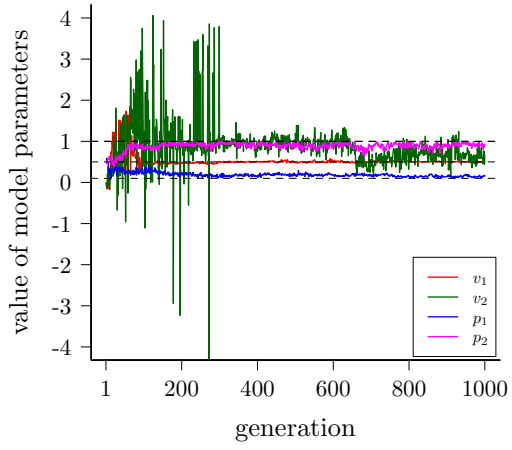
Figure 5.18: This plot shows the distributions of the evolved models with the highest subjective fitness in the $1000^{\text{th}}$ generation in the coevolutions. Each box corresponds to 30 coevolution runs. The dotted lines correspond to the values of the four parameters that the system is expected to learn (i.e. those of the agent). See text for details.




Interactive




Passive 1




Passive 2

Figure 5.19: Evolutionary process of the evolved model parameters for (a) "Interactive", (b) "Passive 1" and (c) "Passive 2" setups of the metric-free method when learning the 3-state agent behavior. Curves represent mean values across 30 coevolution runs. Dotted black lines indicate true values.

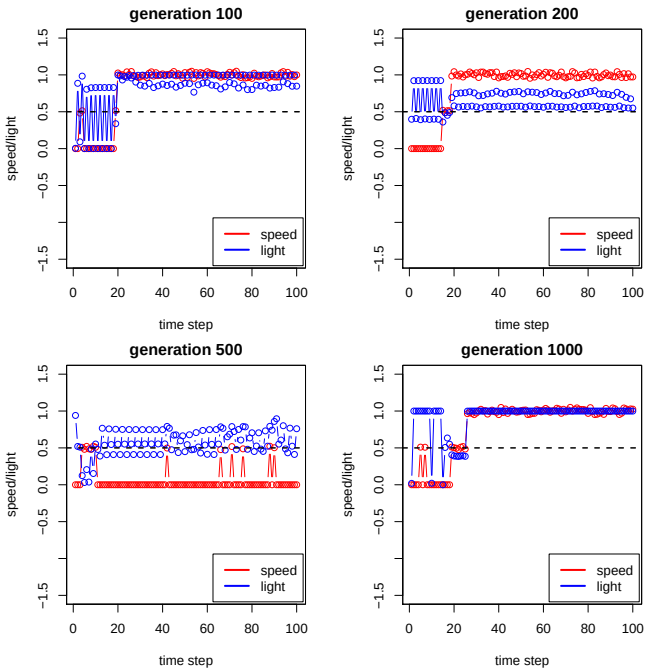



Figure 5.20: This plot shows an example of how the classifier with the highest subjective fitness in different generations (of a particular coevolution run) dynamically change the light intensity to interact with the 3-state agent during a trial.

# 6 Conclusion

## 6.1 Summary of Findings

This thesis presents a novel system identification/modeling method—*Turing Learning* for inferring agent behavior. Through competitive coevolution between classifiers and models, the method does not rely on predefined metrics for measuring the difference between the agents and models. Instead, the classifiers substitute the metrics, which are incorporated into the evolutionary process. During the evolutionary process, the classifiers only need to output a Boolean value indicating whether the observed individual is an agent or model.

The merits of *Turing Learning* were demonstrated through successfully inferring various agent behaviors ranging from swarm behaviors to deterministic/stochastic behaviors of a single agent. When inferring an unknown swarm behavior, it would be challenging to quantitatively measure the difference (e.g., motion) between the models and agents using predefined metrics, due to the interaction within agents and between agents and environment. The motion of each agent in the swarm appears to be stochastic. We have shown that through evolving classifiers that only need to distinguish between the observed agents and models, it is sufficient to infer the behavioral rules of the swarming agents. In other words, in order to 'trick' the classifiers to judge them as agents, the models need to evolve to mimic the behaviors of agents. The evolved models show good match to the original agents in terms of local behaviors (parameter space) and global behaviors. The evolved classifiers perform collectively well and could be potentially used for detecting abnormal behaviors (e.g., faulty agents) in the swarm. We have also shown that swarm behaviors can be directly inferred from the motion of a single agent in the group, as long as the group size is sufficiently large. This may have significant implications for the study of animal collectives, as in practice it may be difficult to track a large number of animals in a group.

When inferring the deterministic/stochastic behaviors of a single agent, we extend the ability of the classifiers in *Turing Learning*. That is, the classifiers can interact with the agent through controlling the environmental stimulus that the agent responds to. The interactive learning approach proves to be superior to passive learning in terms of model converging rate and model accuracy, especially when the agent behavior under investigation has low observability. In the case study about inferring deterministic behaviors of a single agent, we have shown that it is possible to infer the behaviors through coevolving a fixed sequence of inputs and models. Therefore, in this case, the interaction between the classifiers and agent is not very 'intelligent' as they only need to output a fixed sequence no matter what the agent's observed behavior is.

In a later case study, in which *Turing Learning* was applied to infer the stochastic behaviors of a single agent. We have shown that through actively interacting with the agent during the experimental process, the classifiers can 'intelligently' change the stimulus based on the agent's observed behavior in order to extract the hidden information from the agent. In other words, the classifiers could dynamically capture the motion of the agent to adjust the environmental stimulus. This intelligent behavior is also observed in humans when scientists try to investigate the animals' behavior in response to stimuli [**?** ]. As the agent's behavior under investigation is stochastic, it is impossible to evolve a fixed sequence of inputs to extract all the agent's behavioral information. Given the same input, the agent would probably behave differently and this make it hard to optimize the models using predefined metrics. We compared the results obtained by *Turing Learning* and two other metric-based system identification methods, and show that *Turing Learning* learn the agent behavior significantly better than the other two methods. This highlight the benefits of *Turing Learning* in inferring agent behavior.

## 6.2 Future work

In spite of the encouraging results obtained when applying *Turing Learning* to infer agent behaviors, we do not claim that this method can be directly used for modeling animal behaviors. There are still questions need to be discussed before conducting experiments on animals. In the following, we provide some points that could be considered as our future work.

- In the thesis, the models are represented by a set of parameters that govern behavioral rules of the agents. A we argued before, this makes it feasible for us to

objectively gauge the quality of the models through comparing the ground truth. In the future, we will try to evolve the structure of the models as well (e.g., using genetic programming or artificial neural network).

- The swarm behaviors investigated in this thesis is deterministic in terms of the individual's behavioral rules. In the future, we could apply *Turing Learning* to learn such swarm behaviors that are stochastic. In fact, we have shown that if the original controller of the aggregation behavior investigated in Chapter 3 is stochastic, it is still possible to achieve the same global behavior [21]. For the stochastic aggregation controller, in each state, the agent has a probability ($p$) of switching to another state (i.e., its binary sensor is flipped with certain probability). In this case, *Turing Learning* could be used for inferring both the controller and the probability.

- We could apply our approach to learn more complex behaviors (for example, when the agents have more states or rules). When the behaviors become more complex, instead of analyzing only the motion of individual agents, more information (such as number of the agent's neighbors or its internal states) may need to be provided to the classifiers.

- In principle, the *Turing Learning* method is applicable to infer human behavior. It could evolve models that aim to pass the Turing Test [**?** ], at least with regards to some specific subset of human behavior. In this case, the classifiers could then act as Reverse Turing Tests, which could be applied in situations where a machine needs to distinguish human agents from artificial ones as done, for example, by the "Completely Automated Public Turing test to tell Computers and Humans Apart" system (CAPTCHA) [**?** ], which is widely used for internet security purposes. Another exciting application of *Turing Learning* is reversing engineering the signature of human beings. Given a collection of signature from a human being, we could coevolve models which are computer programs that automatically generate signature and classifiers (e.g., neural network) that distinguish between the signature from human or that generated by models. Once the models are obtained, we can use the models to generate electronic signature similar to that of the human being.

# Bibliography


[1] J. P. Grotzinger, "Habitability, taphonomy, and the search for organic carbon on mars," *Science*, vol. 343, no. 6169, pp. 386–387, 2014. [Online]. Available: http://www.sciencemag.org/content/343/6169/386.short

[2] R. P. Hertzberg and A. J. Pope, "High-throughput screening: new technology for the 21st century," *Current Opinion in Chemical Biology*, vol. 4, no. 4, pp. 445 – 451, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1367593100001101

[3] J. Bolhuis and L. Giraldeau, *The behavior of animals: mechanisms, function, and evolution*. USA: Wiley-Blackwell, 2004.

[4] W. J. Sutherland, "The importance of behavioural studies in conservation biology," *Animal Behaviour*, vol. 56, no. 4, pp. 801–809, 1998.

[5] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. Cambridge, MA: MIT Press, 2008.

[6] J.-A. Meyer and A. Guillot, "Biologically inspired robots," in *Springer Handbook of Robot.*, ser. Springer Handbooks, B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg, Germany: Springer, 2008, pp. 1395–1422.

[7] R. King, J. Rowland, S. G. Oliver, and M. Young, "The automation of science," *Science*, vol. 324, no. 5923, pp. 85–89, 2009. [Online]. Available: http://www.sciencemag.org/content/324/5923/85.abstract

[8] J. Evans and A. Rzhetsky, "Machine science," *Science*, vol. 329, no. 5990, pp. 399–400, 2010. [Online]. Available: http://www.sciencemag.org/content/329/5990/399.short

[9] D. Waltz and B. G. Buchanan, "Automating science," *Sci.*, vol. 324, no. 5923, pp. 43–44, 2009.

[10] L. Ljung, "Perspectives on system identification," *Annu. Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[11] S. A. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains.* Hoboken, NJ, USA: Wiley-Blackwell, 2013.

[12] S. M. Henson and J. L. Hayward, "The mathematics of animal behavior: An interdisciplinary dialogue," *Notices of the AMS*, vol. 57, no. 10, pp. 1248–1258, 2010.

[13] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *PNAS*, vol. 99, no. 10, pp. 7280–7287, 2002.

[14] J. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Trans. Evol. Computation*, vol. 9, no. 4, pp. 361–384, 2005.

[15] ——, "Automated reverse engineering of nonlinear dynamical systems," *PNAS*, vol. 104, no. 24, pp. 9943–9948, 2007.

[16] G. D. Ruxton and G. Beauchamp, "The application of genetic algorithms in behavioural ecology, illustrated with a model of anti-predator vigilance," *Journal of Theoretical Biology*, vol. 250, no. 3, pp. 435–448, 2008.

[17] S. Camazine, J.-L. Deneubourg, N. R. Franks, *et al.*, *Self-Organization in Biological Systems.* Princeton, NJ: Princeton University Press, 2001.

[18] D. Helbing and A. Johansson, "Pedestrian, crowd and evacuation dynamics," in *Extreme Environmental Events*, R. A. Meyers, Ed. Springer, 2011, pp. 697–716.

[19] J. Harvey, K. Merrick, and H. A. Abbass, "Application of chaos measures to a simplified boids flocking model," *Swarm Intell.*, vol. 9, no. 1, pp. 23–41, 2015.

[20] W. S, B. S, F. R, *et al.*, "Modeling collective animal behavior with a cognitive perspective: a methodological framework," *PLoS ONE*, vol. 7, no. 6, 2012, e38588.

[21] M. Gauci, J. Chen, W. Li, T. J. Dodd, and R. Groß, "Self-organized aggregation without computation," *The Int. J. of Robot. Research*, vol. 33, no. 8, pp. 1145–1161, 2014.

[22] ——, "Clustering objects with robots that do not compute," in *Proc. 2014 Int. Conf. Autonomous Agents and Multi-Agent Syst.*, IFAAMAS Press, Paris, France, 2014, pp. 421–428.

[23] J. Bongard and H. Lipson, "Active coevolutionary learning of deterministic finite automata," *The Journal of Machine Learning Research*, vol. 6, pp. 1651–1678, 2005.

[24] E. Martin, *Macmillan Dictionary of Life Sciences (2nd ed.)*. London: Macmillan Press, 1983.

[25] M. Dacke, M. J. Byrne, C. H. Scholtz, and E. J. Warrant, "Lunar orientation in a beetle," *Proc. of the Roy. Soc. of London. Series B: Biological Sci.*, vol. 271, no. 1537, pp. 361–365, 2004.

[26] E. Baird, M. J. Byrne, J. Smolka, E. J. Warrant, and M. Dacke, "The dung beetle dance: an orientation behaviour?" *PLoS ONE*, vol. 7, no. 1, p. e30211, 2012.

[27] J. L. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, no. 2, pp. 179–211, 1990.

[28] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies - a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[29] A. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Berlin, Heidelberg: Springer-Verlag, 2003.