The University Of Sheffield.

Turing Learning: Advances and Applications

Yue Gu

Supervisors:

Dr Roderich Groß
Prof. Neil Lawrence

A Thesis Submitted for the Degree of Doctor of Philosophy

in the

Department of Automatic Control and Systems Engineering

September 2021

I'm afraid that the following syllogism may be used by some in the future.

Turing believes machines think
Turing lies with men
Therefore machines do not think

—*Alan Turing*

Abstract

Turing Learning is the family of algorithms where models and discriminators are generated in a competitive setting. This thesis concerns the coevolutionary framework of *Turing Learning* and investigates the advances for improved model accuracy and the applications in robotic systems.

Advances proposed in this thesis are as follows: an interactive approach to enable the discriminator to genuinely influence the data sampling process; a hybrid formulation to combine the benefits of the interactive discriminator in improving model accuracy and the advantages of the passive discriminator for reducing training cost; an exclusiveness reward mechanism to promote candidates with the exclusive performance during the coevolutionary process.

Applications presented in this thesis are as follows: an approach for a mobile robotic agent to automatically infer its sensor configuration; an approach for the robot agent to automatically calibrate its sensor reading; a novel approach to infer swarm behaviours from their effects on the environment.

The interactive approach has been validated in the inference of sensor configuration and calibration model, leading to the *self-modelling*/*self-discovery* process of robotic agents. Results suggest an improved model accuracy with the interactive approach in both cases, compared with the passive approach. The hybrid formulation and the exclusiveness reward mechanism have been demonstrated in the inference of the calibration model. Results show that almost half of the training cost can be reduced without a decrease in model accuracy by applying the hybrid formulation. The novel reward mechanism can accelerate the convergence without a decrease in model accuracy. The indirect way of inferring swarm behaviours requires a small amount of training and reveals novel behavioural controllers for individual robots.

Acknowledgements

Thanks to my supervisors, Dr Roderich Groß and Prof. Neil Lawrence, for their support throughout my PhD study. Special thanks to Roderich for his professional suggestions and essential insights, not only on my research but on the whole academia life.

Thanks to my lab mates for their help, collaboration and critical feedback. They made my journey of PhD an unforgettable one.

Thanks to anonymous reviewers for their constructive comments.

Thanks to my friends in Sheffield. They made my stay a cheerful one.

Thanks to my family. To my parents, thank you for your consistent support, and selfless sacrifices, which made me who I am today. To my girlfriend, Xia, thank you for your unwavering love, trust and company, which always encourages me to keep a positive life attitude.

Lovely thanks to my cat. You lighted up the lockdown time during the Covid-19 pandemic.

Contents

Chapter 1

Introduction

In 1950, Alan Turing proposed the Turing Test to answer the question "Can machines think?" [1] and established the birth of *Artificial Intelligence* (AI). The field of AI was formally instituted at the Dartmouth Conference in 1956. Researchers at the conference determined that the ultimate goal of developing AI is to realise human intelligence on machines. Over the past decades, the development of AI has introduced significant benefits to our daily life and improved our living quality.

Most current techniques implemented in AI are rooted in machine learning, of which *supervised learning* is the most broadly used form. Supervised learning aims to find the associated mapping from each input to each output in a given dataset of example input-output pairs. Although supervised learning often leads to better accuracy than humans, the training process requires millions of data samples. Worse, these data samples usually rely on human supervisors to provide labels. In order to learn from fewer data and reduce human supervision, many studies focus on another form of machine learning: *unsupervised learning*. By definition, unsupervised learning aims to identify useful patterns that are not well-defined from a dataset of unlabelled inputs.

Generative modelling has been widely applied to achieve unsupervised learning. Traditional approaches to generative modelling depend on optimising an explicit representation $p_{model}(\mathbf{x}; \boldsymbol{\theta})$ of parameters $\boldsymbol{\theta}$ to estimate an unknown distribution

$p_{data}(\mathbf{x})$ of data samples $\mathbf{x}$. However, for some complex data, such as realistic images, explicit representations are usually complicated and computationally difficult to be optimised. Other approaches avoid this issue by building implicit generative models that can generate samples directly from the distribution p_{model}. The most popular and successful approach in this category is the *Generative Adversarial Networks* (GANs) [2]. GANs are frameworks where generative models are inferred in a competitive setting with discriminators: The discriminator aims to correctly classify samples generated from either the model or the training data distribution. The model aims to misguide the discriminator to categorise its samples as training data. In the perspective of game theory, GANs are implemented as a *game* between two players: the model and the discriminator [3]. Since its inception in 2014, GANs have been subsequently studied in various variants and successfully applied in developing generative models for complex distributions, especially for the generation of images.

Another field where explicit methods are not always available is the identification process of robotic systems. Traditional system identification approaches rely on predetermined metrics to minimise the difference between the output of the model and that of the target system. However, determining a metric for complex systems can be challenging. For example, it is hard to evaluate agents' behaviour in a robotic system with an explicit function as the behaviour is usually generated from complex interactions and can be stochastic due to noise. In order to achieve a metric-free approach, *Turing Learning* [4], inspired by the Turing Test, was developed to optimise models simultaneously with discriminators in a game setting. The idea of *Turing Learning* was first proposed in [5] and has been demonstrated to infer behavioural rules of robot agents in a swarm, where the discriminator's objective is to identify the trajectory data from either the agents under investigation or the agents executing the rules defined by models [6].

Even though *Turing Learning* and GANs were developed separately, the idea behind them is similar. Assuming a Turing perspective in [7], GANs and their variants can be generalised as instances of the *Turing Learning* algorithms. The

framework of *Turing Learning* algorithms does not propose a concrete implementation. Instead, it enables the designer to choose their own implementations, including representations for the model and the discriminator (other than the neural networks) and the optimisation algorithm (other than the gradient-based ones). As a result of this, *Turing Learning* holds the promise of feasibility in a wide range of scenarios.

1.1 Motivation

The theoretical model of the original GANs suggests that there is only one Nash equilibrium where the model distribution p_{model} is identical to the training data distribution p_{data} [2]. However, it is difficult to reach equilibrium in practice. Discussions about the convergence of GANs are still open, and many works have focused on this [8–12]. On the other hand, the *Turing Learning* framework allows not only a single candidate as implemented in regular GANs but a population of candidates for the model and the discriminator as implemented in [5]. In the latter case, the convergence to a Nash equilibrium can be interpreted as the *competitive coevolution* between the model and discriminator populations. Although convergence can not be guaranteed, the coevolution effect could lead to a more stable optimisation for both the model and the discriminator. Recently, more and more studies have also focused on the coevolution effect in GANs [13–16] and presented a more stable training.

One motivation for this thesis comes from the techniques that could promote the convergence of the coevolutionary process in *Turing Learning*. In general, the competitive coevolution leads to an "arms race", where individuals compete against each other and obtain fitness through the competition [17]. Ideally, improved solutions could continually occur for both populations during the coevolution. However, this relies on the arms race running for a significant long time, which is difficult to achieve in practice. A reason for the failure of maintaining a steady process

is the lack of diversity in populations [17]. Without enough new individuals introduced into the population, the evolution could easily get stuck in local optimums. In *Turing Learning*, this issue could be avoided by enabling the discriminator to interact with the model and the system under investigation. The discriminator could hence examine between different local optimums and push the convergence towards the Nash equilibrium. The inspiration comes from the interrogator's role in the Turing Test, where the interrogator influences the data sampling process by posing questions to the other two players. Previous studies on *Turing Learning* have shown the advantage of a pre-determined interaction sequence in the inference of a simple one-dimensional behaviour [5]. However, whether a fixed protocol of interaction is capable enough for the inference of complex behaviours needs to be further investigated.

Another reason for the failure of competitive coevolution is the vanishing fitness gradient when one population is extremely dominant in the competition. Then, both populations stop evolving due to the loss of selection pressure. Then, the subjective fitness in coevolution no longer represents the quality of solutions as it becomes constant in each population [17]. Inspired by the Elo system [18] for rating the strength of players in chess, continual selection pressure on both populations in *Turing Learning* could be achieved by establishing a new fitness mechanism to evaluate the exclusive abilities of candidates.

The other motivation for this thesis comes from the possibility of *Turing Learning* in opening up new application domains in robotics. Robotics is the field where perception is connected with action. AI plays an essential role in enabling such connections to be intelligent [19]. One promise of combining AI and robotics is to enhance the perception, which is not only for planning but also for bringing the sense of self-awareness to robots [20]. This opens up an interesting topic about the *self-modelling/self-discovery* of a robotic agent. Inspired by the resilient machine in [21], the interactive approach of *Turing Learning* could contribute to the self-modelling process of a robotic agent to infer its characteristics, such as the sensor configuration and sensing capabilities.

However, the interactive approach with robotic systems could be expensive. Each generated sample can not be reused as specific to the interaction, which leads to a costly process. In general, the energy expended and time spent increase, usually linearly, with the amount of data to be collected. Hence, how to adapt the interactive approach of *Turing Learning* to applications in robotics would be another promising topic.

Furthermore, *Turing Learning* was initially proposed as an automated reverse engineering method of agent behaviours. Compared with the *forward engineering*, where high-level abstractions and logical designs are transferred into physical implementations, *reverse engineering* is the process of replicating an exciting system without "drawings, documentations or a computer model" [22]. In terms of the study of animal behaviours — *ethology*, the robotic system is commonly used to duplicate and demonstrate the behavioural rules of how living organisms respond to internal and external stimuli [23]. These rules are usually designed by researchers based on observation, which is time-consuming and exhausting. Moreover, it is typically hard for researchers to manipulate the environmental features in an effective way to provide more insights into the biological system under observation. *Turing Learning* offers an automated way to carry out the whole process of observing, designing and demonstrating. For example, it has been applied to infer controllers for the behaviours of a swarm of robots, such as the aggregation and the object clustering in [4]. In both cases, the inference relies on full observability of the robotic agents (i.e. full knowledge of individual agent's velocities). The third possible application with *Turing Learning* is mostly curiosity-driven. If swarm behaviours could be learned by monitoring some tractable features, instead of directly observing the swarm, the inference could be easily applied in the physical world.

1.2 Problem Statement

In general, the robotic agent considered in this thesis consists of a rigid body, a class of onboard sensors, and actuators. Case studies will be conducted with a simulation platform that has been broadly verified and shown to be able to provide an effective transition into the physical lab. The robot is placed into an environment along with other robots and/or objects. It moves around and perceives the surroundings with its sensors. The model is executed on the robot and determines its way of sensing and operating. The discriminator can observe the robot's sensor reading values and potentially control the robot's motion.

In the first case study, we focus on the active self-discovery of robotic systems. Robots have been used in many practical scenarios, for example, industrial robots that are used to perform repetitive tasks in well-structured production lines. Achieving reliable performance in complex environments with uncertainties is still challenging for robotic systems. During the operation, unexpected topological changes may be introduced to the robot's body due to damage or collisions. These changes would affect the ability of the system to plan actions. In particular, sensors can not provide accurate information about the environment if they are not mounted precisely, or their positions are physically changed. To address this problem, we present a case study where a robotic agent uses *Turing Learning* to infer its sensor configuration. The robot moves in a bounded environment with obstacles but has no knowledge about its relative positions to obstacles. The discriminator has full control of the robot's motion and observes sensor readings at the same time. The task requires the discriminator to construct some active patterns of control to examine the inferred configuration.

We also consider the training cost of the coevolutionary process in *Turing Learning*, where a pairing strategy is required to evaluate fitness for individuals. To provide stable training, we use the "*all vs. all*" pairing, where each discriminator candidate is evaluated against each model candidate. However, this pairing approach results in a large amount of training data being generated, which could be challenging when the evaluation is conducted with a physical robot. To address this problem,

we present a *hybrid* formulation of *Turing Learning* which consists of two types of discriminators: the interactive one, to maintain the advantages of interaction, and the passive one, to reuse the recorded training data. We demonstrate the hybrid formulation in our second case study, which is about a fully autonomous robot self-calibrating its distance sensor without any global knowledge about its environment (e.g. the ground-truth distance). Moreover, we present a novel reward mechanism to take into account the *exclusiveness* of each candidate so that the coevolution process can be enhanced.

In the third case study, we investigate the inference of swarm behaviours through *Turing Learning*. Modelling swarm behaviours is usually done by designing controllers that can capture the features of biological swarms. In previous studies, the inference relies on the full observability of the robot's behaviour. For example, in [6], *Turing Learning* has been implemented to infer the *sense-act* controller [24] of individual robots by observing the trajectory of robot agents in a swarm. Although, the observation is not always available, especially for complex behaviours where the interactions within individuals are hard to be observed. In this case study, we investigate if swarm behaviour can be inferred indirectly by monitoring its effects on the environment. Particularly, we study two behaviours: object clustering and shepherding. In all situations, the individual's motion follows a simple *sense-act* rule. To reveal such a rule, the *Turing Learning* algorithm observes (i) a single object being clustered; (ii) a single sheep of the flock. This study is promising as it requires a small amount of information. It may also potentially discover new possibilities for the solution.

1.3 Preview of Contributions

- An approach for a mobile robotic agent to automatically infer its own sensor configuration when starting from a random position in a given environment. The approach is an example of active self-modelling that requires no predefined performance measurement. It does need any global knowledge, such

as positional information. It is validated by simulation. The inferred model parameters accurately matched those defined in the simulator.

- An interactive approach of *Turing Learning* in which the discriminator actively influences the data sampling process, thereby assuming the role of an interrogator (as opposed to a passive observer), akin to the one in the Turing Test. The approach allows a bi-directional flow of information between the discriminator and either the model or the system under investigation. The approach is validated in the above study and has shown its advantage in improved model accuracy compared with the passive approach.

- An approach for the robot agent to automatically calibrate its sensor reading when starting from a random position in a given environment. The approach is an example of active self-calibrating that requires no ground-truth information as traditional approaches do. It is validated in simulation. The inferred model parameters accurately matched those defined in the simulator.

- A hybrid formulation of *Turing Learning* that combines the use of interactive discriminators to retain the advantage of active learning with the use of passive ones to allow recorded training data to be reused. The formulation is validated in the self-calibrating study and has shown an almost 50% reduction of the training cost, compared with the purely interactive approach, with no decrease in model accuracy.

- An exclusiveness reward mechanism for *Turing Learning* to take into account the exclusive performance of each player in the game. The mechanism promotes the individuals that perform well against rarely defeated opponents. The mechanism is validated in the self-calibrating study, and it was shown that it is able to boost the coevolution process with no decrease in model accuracy.

- A novel approach to automatically infer swarm behaviours from their effects on the environment. To the best of our knowledge, this is the first instance where the behaviour of a swarm is inferred indirectly. The approach is

validated in simulations of learning two behaviours: object clustering and shepherding, where the inference is done by observing the position changing of a single object or sheep agent that is randomly picked. Compared with previous studies, the approach requires a reduced amount of information.

- A novel controller for a swarm of robots with binary, line-of-sight sensor to cluster objects, which enables a relatively small number of robots to effectively cluster a large number of objects; Another novel controller for a swarm of robots with binary, line-of-sight sensor to accomplish shepherding, which enables shepherd robots to efficiently herd a group of sheep robots towards a dynamical goal.

1.4 Publications

This thesis presents the author's own work, and parts of the work have been published.

A short version of Chapter 3 was included in the publication:

- R. Groß, **Y. Gu**, W. Li, and M. Gauci, "Generalizing GANs: a turing perspective," in *Advances in neural information processing systems*, Curran Associates Inc. (2017), pp. 6316-6326.

It was orally presented at the 31st Conference on Neural Information Processing Systems (NIPS 2017), held at Long Beach, CA, USA.

A preliminary version of Chapter 4 was included in the publication:

- **Y. Gu**, W. Li, and R. Groß, "Turing Learning with Hybrid Discriminators: Combining the Best of Active and Passive Learning," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO 2020)*, ACM Press, pp. 121-122.

It was orally presented by the author at the respective conference, held online.

Apart from his main work, the author has also contributed to other projects that are not presented in this thesis. These led to the following publication:

- M. J. Doyle, J. V. A. Marques, I. Vandermeulen, C. Parrott, **Y. Gu**, X. Xu, A. Kolling, and R. Groß, "Modular Fluidic Propulsion Robots," in *IEEE Transactions on Robotics*, in press.

1.5 Thesis Outline

This thesis is structured as follows:

- Chapter 2 presents an overview of the background that this thesis is positioned in and previous work that relates to the main topic of this thesis. The background starts with the development of artificial intelligence (AI) and robotics in Section 2.1, including a historical review of AI (Section 2.1.1) and AI in robotics (Section 2.1.2). Then, Section 2.2 briefly discusses the evolutionary computation, including the inspiration in biology (Section 2.2.1), an introduction of different methods (Section 2.2.2) and their applications (Section 2.2.3). The final part of the background presents an overview of swarm intelligence (Section 2.3), including swarms in nature (Section 2.3.1), two swarm optimisation methods (Section 2.3.2) and the development of swarm robotics (Section 2.3.3). The related works include the Generative Adversarial Networks (GANs) (Section 2.4.1) and previous studies on *Turing Learning* (Section 2.4.2).

- Chapter 3 presents the *Turing Learning* framework and demonstrates the method, particularly an interactive approach, to infer the sensor configuration of a robot agent. Section 3.2.1 describes the definition of the framework within the context of coevolution. Section 3.2.2 introduces the simulation platform used in this chapter and thesis. Section 3.3 presents a case study to

demonstrate the framework, including a formal problem formulation (Section 3.3.1). Section 3.3.2 lists the implementation options that are chosen for this scenario. Section 3.3.3 describes the simulation setup designed for carrying out the investigation. The simulation results are presented in Section 3.3.4, including an analysis of the inferred model, a discussion about the coevolutionary dynamics, a comparison between the interactive setup and passive setups and an illustration of the discriminator's control pattern. Section 3.4 summarises the chapter.

- Chapter 4 presents the hybrid formulation of *Turing Learning* and an exclusiveness reward mechanism. They are applied to calibrate the distance sensor of a robot agent. Section 4.2.1 describes the hybrid formulation in detail. Section 4.2.2 formally defines the novel reward mechanism. Section 4.3 presents a case study to demonstrate the method, including an introduction of the simulation platform (Section 4.3.1). Section 4.3.2 lists the implementation options of hybrid formulation that are chosen for this scenario. The simulation results are presented in Section 4.3.3, including an analysis of the non-hybrid *Turing Learning* formulations, a consideration of the practical cost, a discussion about the coevolutionary dynamics among three populations and the impact of the exclusiveness. Section 4.4 summarises the chapter.

- Chapter 5 presents a novel way to infer swarm behaviours from their effects via *Turing Learning*. Section 5.2 describes the methodology, including a detailed definition of the problem (5.2.1), an introduction about the simulation platform (5.2.2) and the implementation of the *Turing Learning* algorithm (5.2.3). Section 5.3 presents two case studies to demonstrate the method. Section 5.3.1 illustrates how the object clustering behaviour could be learned by *Turing Learning* through observing the trajectory of the object, and Section 5.3.2 implements *Turing Learning* to infer shepherding behaviour via monitoring the trajectory of sheep. The inferred controllers in both cases are analysed in terms of their resulting emergent behaviours and scalability with respect to the different numbers of agents. At the same time, their

performance is compared with one of the controllers designed in the literature. In addition, Section 5.3.1 investigates how the training data affects the scalability of the clustering controller learned by *Turing Learning*, and Section 5.3.2 shows the potential of the inferred controller to accomplish the shepherding task with a dynamical goal. Section 5.4 summarises the chapter.

- Chapter 6 concludes the thesis and discusses the potential directions for future work.

Chapter 2

Background and Related Work

2.1 Development of AI and Robotics

2.1.1 A Historical Review of AI

The Birth

In 1950, the British mathematician Alan Turing tried to answer a highly ambiguous question, "Can machines think?", in his *Mind* paper "Computing Machinery and Intelligence" [1]. He proposed the *Imitation Game* to transform the abstract question into a particular situation. The original game is played among three people: a man (A), a woman (B), and an interrogator (C) who can be either man or woman. C stays apart from the other two and puts questions on any subjects in writing; A and B write down their answers and pass them back to C. The objective of the interrogator is to distinguish between the man and the woman based on their answers. Both the man and the woman are to let the interrogator believe they are the woman. Then, Turing raised new questions to replace the original one:

> "What will happen when a machine takes the part of A in this game? Will the interrogator decide wrongly as often as when the game is

> played like this as he does when the game is played between a man and a woman?" [1, p.433]

Ignoring the gender issue, the imitation game played between a machine (A), a human (B) and an interrogator (C) is also widely known as the *Turing Test* (as shown in Figure 2.1). What Turing tries to answer is whether a machine has the ability to imitate a human. His idea has been acknowledged by some as the origin of *Artificial Intelligence* (AI), and the Turing Test has been as its ambition [25].

The field of AI was formally instituted at the Dartmouth summer research project (or Dartmouth Conference) in 1956. McCarthy, Minsky, Rochester and Shannon proposed the main problems that are aimed to solve [26]:

> "to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves"

The success of the Dartmouth Conference gave rise to a new age of discovery and stimulated the longing for new knowledge [20]. Researchers were attracted and engaged in the development of AI, many of whom were eventually considered the founders of the field.

Seasons

The evolution of AI has experienced several "seasons" (the ups and downs) since the birth of the field. We generally summarise them as:

- First spring: Dartmouth Conference remarks the beginning of the first spring of AI for nearly two decades. That time had seen many significant successes in the field, for example, the natural language processing program ELIZA [27] and the General Problem Solver [28].
- First winter: Unlike the general optimism since the birth of AI, the public and media started to question the promising prospects claimed by AI

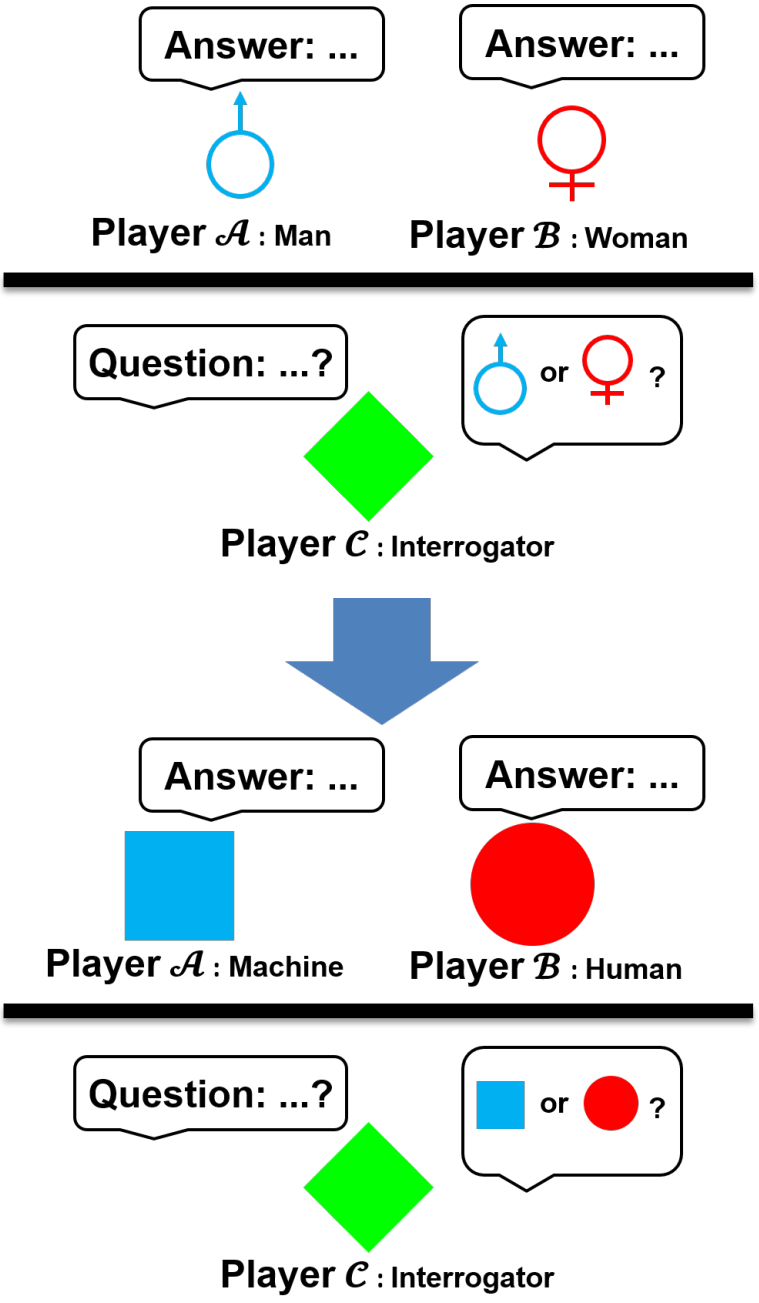


FIGURE 2.1: Transformation from Imitation Game to Turing Test. Turing's idea set the ultimate goal of Artificial Intelligence, that is, imitating humans.

researchers. In 1973, Lighthill stated that machines would never break the level of an "experienced amateur" in chess games and would never be able to solve common-sense problems as humans [29]. Following the sudden ending of support of the British and U.S. government, the research of AI entered its first winter.

- Second spring: In the early 1990s, the second spring came with the revival of several techniques, such as expert systems and neural networks. Expert systems were developed in the mid-1960s by AI researchers. They are represented as collections of rules that formalise human intelligence in specific domains into "If-Then" statements by human experts [30]. However, the top-down approach of expert systems performs poorly in complex forms of intelligence. As the lack of ability of processing external data, expert systems are technically not considered the true AI [31]. The discussion about statistical methods for realising true AI can be traced back to the 1940s when psychologist Donald Hebb proposed an idea of replicating the process of human brain neurons [32], which led to the research of artificial neural networks. Due to the limited ability of computers at that time, the applications and benefits of artificial neural networks came up in the later years.

- Fall: Since 2000, along with the increased computer power and the Big Data revolution, artificial neural networks were back to the stage and AI finally entered its first golden season. Machine learning, especially Deep Learning, has become the cornerstone of AI. Many applications were developed in wide areas, including image processing, speech recognition, natural language processing, computer vision, auto-driving and robotics.

Debates

The birth of AI brought a lot of discussions into the field. First of all, the definition of "artificial intelligence" was and still is highly debated. Inspired by Turing's idea, some AI researchers described AI as the ability to simulate human intelligence. In [33], Charniak gave a definition as: "Artificial Intelligence is the study of mental

faculties through the use of computational models". Moreover, Minsky offered another definition as: " Artificial Intelligence is the science of making machines do things that would require intelligence if done by men" [34]. However, McCarthy argued that AI is not always about simulating the behaviours observed in humans or animals. Instead, "the situation in AI study is the reverse" [35, p.3]. The methods involved in AI are usually not the ones used by humans and requires plenty of computing efforts that humans don't have. He summarised that this is due to the lack of knowledge of the intellectual mechanisms that humans have [35].

As the development of computers was at its early stage, AI researchers attempted to represent human intelligence into a declarative form, like symbols and rules for manipulation, which led to the primary paradigm, *symbolic AI*, from the middle 1950s to the late 1980s. Researchers believed that symbolic AI was the way to achieve the ambitions, although they realised that computers have a vastly different mechanism than the biological one observed in humans.

Symbolic AI has had remarkable successes. For example, expert systems have been applied in assisting humans in decision making. However, not all philosophers were convinced by the promised outlooks of symbolic AI. John Searle argued that it is incoherent for a non-biological machine to be intelligent. He proposed the *Chinese room argument* in 1980 to illustrate the symbol grounding problem of "strong AI", which he referred to as the belief of realising real cognition in machines with symbolic AI [36]. The argument outlined that a human could communicate in Chinese by manipulating Chinese characters, but without truly understanding Chinese. Searle concluded that machines could pass the Turing Test if they manipulate symbols reasonably without the need of understanding the meaning or grounding of these symbols. However, that is opposite to AI's ambitions.

In the last decade of the 20th century, researchers started to realise that philosophers' criticisms on the goal of achieving human intelligence on machines through symbolic AI was justified. Rodney Brooks revealed a fact that "Early hopes diminished as the magnitude and difficulty of that goal was appreciated. Slow progress was made over the next 25 years in demonstrating isolated aspects of

intelligence" [37]. Brooks argued that the reason for the limited success in the early development was the obsession with a complete and explicit representation of the complex human-level intelligence. As he entitled one of the sections in [37], "Abstraction as a dangerous weapon". Brooks thus called for a new view of intelligence as "Intelligence Without Representation" [37] or "Intelligence Without Reasons" [38]. In the later work [39], Brooks suggested a bottom-up manner for the development of AI: comprehending every aspect and mechanism of low-level intelligence, like the insect-level intelligence, before moving up to a higher-level one, like the human-level intelligence.

2.1.2 Combining AI and Robotics

Robotics has its roots in the human history of designing mechanical tools. After World War II, the boost of the nuclear industry and advances in control theory encouraged engineers to design robot arms to replace the telemaipulator in transporting nuclear materials. Robot-like machines started to emerge [40]. These techniques began being introduced in general manufacturing in 1956. In the next two decades, applications like industrial manipulators and automated guided vehicles were under development, during which time the AI experienced its first spring. Although AI and robotics were not combined originally, the early success of symbolic AI inspired researchers to solve problems in the physical world. In the meantime, the progress of producing computationally powerful embedded devices gave AI researchers a chance to demonstrate complicated algorithms on robots.

As introduced previously, symbolic AI was developed on the basis of the physical symbol system hypothesis, which requires a symbolic formalisation of the problem under consideration and the solutions as well. However, the physical world does not provide available symbols to the robot. The main challenge of building robots with intelligence was to find the approaches that allow robots to realise such formalisation of the information they obtain from the environment. Moreover, the symbolic solutions need to be converted into actions that robots can execute. This led to the three primitives of robotics: *sense*, perceiving the environment with

sensors and generating sensory readings for extracting symbols; *plan*, processing symbols and producing tasks and a plan of actions for the robot; *act*, outputting actuator commands for the robot [40].

From 1967-1990, the widely used paradigm of organising intelligence in robots is known as the Hierarchical Paradigm, under which the robot follows a sequential architecture: "$sense \rightarrow plan \rightarrow act$". The planner uses a global world model to transmit all sensing information into actions. The first concrete robot that carried out this paradigm was the Shakey, developed between 1966 and 1972 at the Artificial Intelligence Centre of the Stanford Research Institute (now SRI International) [41]. Shakey is arguably considered the first robot that is capable of AI. However, similar to other projects at that time, Shakey operated only in simplified environments with well-controlled conditions. This is because that the need for generic global world models for Hierarchical Paradigm is hard to be met in complex environments, which is also the shortcoming of symbolic AI. To adapt to more realistic situations, each community is faced with challenging problems in its own field. AI and robotics diverged from each other, while both achieved enormous progress during this period of time.

In the early 90's, Brooks's new view of intelligence also inspired "New Approaches to Robotics" [42]. He proposed the subsumption architecture, which breaks the chain of information processing modules in the traditional approach, and instead, connects the signals from the real world directly to actions and allows different behaviour-generating modules running in parallel (which is also referred to as behaviour-based robotics [43]):

> "There was no central model of the world explicitly represented within the systems. There was no implicit separation of data and computation [...] There were no pointers and no easy way to implement them, as there is in symbolic programs. [...] There was no central locus of control. [...] There was no notion of one process calling on another as a subroutine. [...] The boundary between computation and the world was harder to draw as the systems relied heavily on the dynamics of

> their interactions with the world to produce their results." [42, p.1229-1230]

This idea is related to the Reactive Paradigm of robotics, under which the robot follows multiple concurrent "$sense \rightarrow act$" couplings. Each coupling generates one behaviour, and the final action of the robot is a combination of all behaviours. The subsumption architecture encouraged various research groups to reunion AI and robotics and create many famous robots, for example, Allen [44], Raibert's hopping robots [45] and Genghis [46]. But, sooner, it became clear that an entirely dropping *plan* was too narrow for organising high-level intelligence. On the basis of the Reactive Paradigm, the Hybrid Paradigm let robots plan to break down a task into subtasks before executing Reactive Paradigm. In other words, "$plan$, $sense \rightarrow act$". At the end of 90's, more and more robots were created as an integration of AI and robotics, such as Flakey [47], Rhino [48], Minerva [49] and Deep Space One spacecraft [50, 51].

2.2 Evolutionary Computation

2.2.1 The Inspiration from Biology

Darwinism

Darwin's theory of evolution was first formally introduced in his book "On the Origin of Species" [52] in 1859. It explains the diversity of living organisms and the underlying process from a macroscopic view, that is, *natural selection*. Assuming in an environment with limited resources (e.g. capacity), the instinct of individuals producing offspring eventually leads to the selection. In a natural way, individuals that are better adapted to the environment are more likely to survive and reproduce, which is also referred to as *survival of the fittest*. This phenomenon reveals one of the two essentials in the evolutionary process: competition. The other one is referred to as phenotypic variations among individuals.

Within a population, each member is represented as a combination of phenotypic traits, which consists of the biological and behavioural features determining the way of interaction with the environment and other members, for example, organs. During natural selection, those traits that are favoured by the environment have a higher chance to be retained through offspring. Otherwise, they are dying out as evolution proceeds. Darwin identified that mutations happen during reproduction, which introduces new phenotypic traits (e.g. beneficial organs [53]) as well as new combinations (e.g. hippopotamuses into whales). Eiben summarised Darwin's theory as a model of a population of individuals, where individuals are "units of selection" and "the population is the 'unit of evolution' " [54].

Modern Synthesis

Modern synthesis, or modern evolutionary synthesis, explains the natural evolution from a microscopic perspective, that is, *genetics*. Under genetics, an individual's phenotypic traits are encoded from its genotype. In other words, evolution happens on the genotypic level, where genes are the basic unit. Nature favours individuals that have beneficial genes to generate phenotypic features for survival. As a result, they have a higher chance to propagate this genotype information through offspring. Meanwhile, phenotypic variations are the consequences of mutation and/or recombination of genes during reproduction.

Although the whole genetic process is not fully understood yet, the modern study has provided more insights. Genes are carried by DNA, the double helix of nucleotides, and govern the production of proteins that form all living organisms on Earth. The information from DNA is passed to RNA first and then is translated to proteins. One concept dogma is that this information path is a single-way path, which means that phenotypes cannot affect genotypes. Consequently, the genetic diversity in a population only derives from the variations in genes and natural selection. The features that are learned during an individual's lifetime cannot be retained in its genes. Eiben emphasised this view as:

> "It is important to understand that all variations (mutation and combination) happen at the genotypic level, while selection is based on actual performance in a given environment, that is, at the phenotypic level." [54, p.7]

2.2.2 Introduction to Evolutionary Computation

Since the biological evolution has created multiple optimal solutions in nature, such as the fish body for less resistance in the water, the birds' wings for more lift force to fly and the geckos' feet for strong stickiness on surfaces, many researchers attempted to mimic the mechanisms that drive biological evolution in order to develop computational techniques for optimisation problems. These attempts have motivated a large amount of research leading to many algorithms that have resulted in a wide range of applications.

Motivation

The motivations of developing evolutionary computation can be summarised as follows:

- *Desire for automated problem solvers.* Searching for automated problem solvers has been one of the ambitions in mathematics and computer science, and solutions in nature provide many inspirations [54]. The biological evolution, which created the most powerful problem solver - the human brain, encouraged researchers to apply nature's answer in design [55], leading to the original idea of evolutionary computation.

- *Need for robust algorithms.* The increasing problem complexity urged researchers to develop algorithms that could widely adapt in different domains and provide good solutions, but without much tailoring. The biology diversity has proved the robustness of evolution, which motivated researchers to

copy the mechanisms of natural evolution in developing powerful algorithms for adaptation and optimisation problems [56].

- *Human curiosity.* Similar to the development of other techniques, curiosity drove researchers to simulate the biological evolution in computer programs, which could not only deepen the understanding of the evolutionary process and also benefit in developing better algorithms [54].

Evolutionary algorithm

The earliest attempt to explain the evolutionary system from an algorithmic view could be traced back to the 1930s when Wright illustrated the problem domain as a multi-peaked landscape and visualised the mechanism as dynamically searching candidate solutions from lower peaks to higher peaks [57]. Wright's perceptive exposed the nature of the evolutionary system as an optimisation process. Another perceptive is to understand the evolutionary system as an adaptive system interacting with a dynamical problem domain, which leads to the insight of a process of maintaining system static in changing conditions via feedback-control [58]. For example, Friedman applied mechanisms to evolve robotic control circuits [59].

Throughout history, many evolutionary algorithms (EAs) have been implemented. The majority can be categorised into four related approaches that were developed independently: *genetic algorithms*, *evolution strategies*, *evolutionary programming* and *genetic programming*.

- Genetic algorithms (GAs) were initially formulated by Holland [60] as an approach of evolving adaptive behaviours and have been widely implemented in optimisation problems. Individuals are represented as genetic-like strings which consist of binary, integer or real-valued genes. Accordingly, the reproduction of offspring, including the mutation and recombination, operates on the string level as well. For example, for a binary string, the mutation could be a bitwise flip with a small probability, and the recombination

could be a one-point crossover where two children are created by exchanging the remaining sequence from a random point chosen in parents' genetic sequences. Two GA models are studied in the literature: the *generational model*, where μ parents are selected in each generation and $\lambda = \mu$ offspring are produced to replace the whole population in the next generation; the *steady-state model* [61], where $\lambda < \mu$ offspring are produced to replace λ individuals in next generation.

- Evolutionary strategies (ESs) were invented by Rechenberg and Schwefel [62, 63] and typically deal with continuous parameter optimisation problems. Therefore, individuals are represented as vectors of floating-point variables. For each object variable x_i, the mutation in ES is operated by adding a random noise drawn from a Gaussian distribution $\mathcal{N}(0, \sigma_i)$. One particular feature of ES is the self-adaption that is done by coevolving the step size, leading to an adaptive mutation operator. Hence, the individuals' representations are extended with strategy variables σ_i and doubled in size. Accordingly, the recombination operates on both the object and strategy variables. One commonly used ES model is $(\mu + \lambda)$, where the fittest μ individuals are selected as the parent population and forms the whole population in the next generation together with λ produced offspring.

- Evolutionary programming (EP) was originally proposed to create artificial intelligence [64, 65]. Individuals are evolved as Finite State Machines (FSM), adapting their behaviour in the environment. Thus, the mutation in EP is to generate new FSMs by modifying the structural features, such as states and arcs. In practice, EP is implemented without the recombination operator as it is hard to design an appropriate one but is still able to achieve impressive results [66]. This leads to asexual reproduction in EP.

- Genetic programming (GP) was introduced in the early 1990s by Koza [67] as an approach to evolving computer programs. Unlike the other members in the EA family were developed as optimisation approaches, GP is rooted in machine learning and modelling tasks. Individuals are represented as

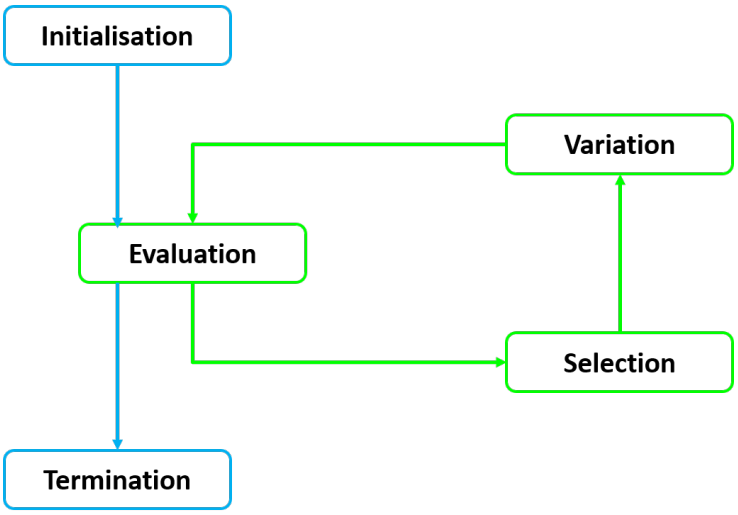


FIGURE 2.2: An illustration of evolutionary algorithms in the flowchart. After initialising the original population, the process proceeds to evaluate all individuals, select better ones and reproduce new candidates through variation operators, and then repeats until the termination criteria are met. Reproduced from [69]

.

parse trees in GP, including arithmetic and logic formulas and the programming procedure. Mutation and recombination operators in other EAs can be adapted in GP to operate in tree structures. Meanwhile, evolution models can be referenced to the two GA models as explained above.

Although different algorithms were developed under different principles, they all contain several important components, including a population base, the representation for individuals, an evaluation method (i.e. fitness function), a parent selection mechanism, recombination and/or mutation operator(s), and a survivor selection mechanism, many of which are stochastic. The general scheme of all EA variants can be presented with a unified view [54, 58, 68]. Given a population, the environment evaluates each candidate's performance with a fitness function and thus causes natural selection. The fittest candidates are selected to reproduce offspring for the next generation through variation operators, which leads to new candidates emerging and replacing some old ones in the population. This process, as illustrated in Figure 2.2, is repeated for generations until a sufficiently good candidate emerges or a pre-set computational limit is met. More importantly, the process is drove by two essentials:

> "Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty; Selection acts as a force pushing quality." [54, p.16]

Coevolutionary algorithm

Darwin's idea of *coadaptation* of two or more species leads to the special form of evolution in nature: coevolution [70], where the fitness of one species adapting to the environment is determined by not only its own genes but also the interactions with other species. Similarly, in coevolutionary algorithms (CoEAs), two or more populations are coevolved simultaneously[1]. The coevolutionary process can be considered several sub-processes for all populations, each of which doesn't interact with others unless for the fitness evaluation. In other words, the individual's fitness is determined by the correlation of its own performance and others' performance in other populations. Depending on the type of such correlation, CoEAs can be categorised into cooperative and competitive ones.

- Cooperative CoEAs are mainly used in the situation where a complex problem is decomposed into tractable sub-problems, and each population represents the candidate solutions for a part of the problem. Hence, populations cooperate with each other to come up with a combined solution. Example applications could be optimising high-dimensional functions [74] and Job Shop Scheduling (JSS) problem [75]. A major issue when applying cooperative CoEAs is to design an effective pairing strategy between populations during the fitness evaluation process. This issue has been broadly studied in [74–77].

- Competitive CoEAs produce an arms race between populations [17]. Individuals from different populations compete against each other and obtain

[1]It is debatable whether a single-population algorithm, where individuals are evaluated through interacting with others, is also called coevolutionary [71]. When we talk about coevolutionary algorithms in this thesis, we refer to the multi-population case. For the single-population coevolutionary, we recommend [72, 73] for further readings.

rewards at each other's loss [54], leading to the evolution of all populations. The most famous example is the iterated prisoner's dilemma (IPD) proposed by Axelrod [72], where each of two players needs to decide either cooperation or defection in each iteration, while how many rewards each can receive depends on the other player's action. Other examples include finding minimal sorting networks [78] and optimising classification systems [79].

In the following, we continue to discuss specifics about competitive CoEAs and competitive coevolution in general.

In CoEAs, there is no determined fitness function. Instead, an individual's fitness depends on the outcomes of interaction with other individuals. This is so-called subjective fitness, compared with the objective one calculated by direct metrics in traditional EAs [71]. For competitive CoEAs, a simple way to measure an individual's subjective fitness is to calculate how many opponents it defeated. To achieve that, competitive CoEAs require a careful pairing strategy to choose opponents and set the competition. Common strategies are: *Round Robin*, under which an individual is paired with all individuals from other populations one by one [72, 80]; *Random Pairing*, literally choosing a random opponent in populations [81]; *Single-Elimination Tournament*, under which individuals are randomly paired, but losers are eliminated from the tournament after one game [82]. Different ways of pairing may influence the quality of competitions and the computational cost of the coevolutionary process. Usually, Random Pairing costs less but performs not well; Round Robin behaves better but costs more.

As discussed above, competitive coevolution relies on the arms race between populations. Good solutions only occur after a certain number of generations. However, maintaining such an arms race in practice is difficult. This issue can be summarised as the synchronisation challenges of the competitive coevolutionary process:

- *The Red Queen Effect.* It is introduced by the pure subjective fitness measurement. It happens when the subjective fitness of two populations continuously improves. However, neither of them constantly makes progress on

the absolute quality, or when they actually improve their absolute quality, but the subjective fitness fails to reflect their progress [83].

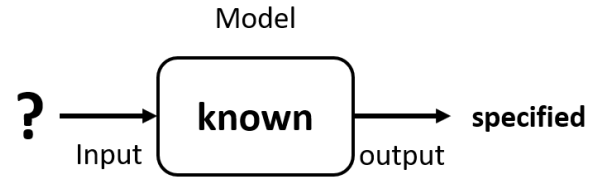
- *Cycling.* It happens due to the instant subjective fitness criteria. As the opponents are changing over generations, individuals with good abilities may be lost and rediscovered in future generations, performing a cycling behaviour. At the same time, the same individuals may have different outcomes against different opponents [83].

- *Disengagement.* It occurs when one population dominates the competition. Subjective fitness no longer represents the quality of solutions as it becomes constant in each population. Thus, selection pressure is lost. Evolution is suspended in all populations [84].

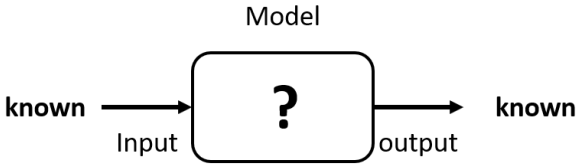
Several methods have been introduced in the literature to improve the competitive coevolutionary process and tackle the above challenges. For example, Rosin and Belew [17] proposed *competitive fitness sharing* to reward individuals with outstanding abilities others don't have, *shared sampling* to maintain the diversity in populations and *hall of frame* to save good solutions from previous generations. Moreover, *reducing virulence* [84] and *reducing sharing* [85] were introduced to address the disengagement issue.

2.2.3 Applications

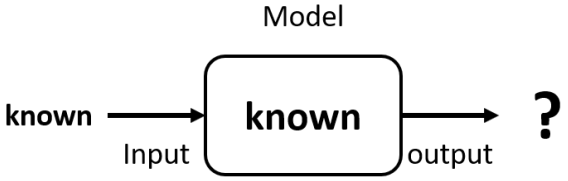
Evolutionary computation has been widely applied in solving real-world problems. From a system analysis perceptive, Eiben [54] positioned various applications into three types of problems: optimisation, modelling or system identification, and simulation. Considering a working system composed of inputs, outputs and the internal model, these three problems can be illustrated in Figure 2.3. In an optimisation problem, the objective is to find inputs that can lead to desired outputs via a known model. In a modelling problem, the objective is to build a model that can connect the given inputs and outputs. In a simulation problem, the objective is to design the outputs given the model and inputs.



(a) Optimisation problems



(b) Modelling or system identification problems



(c) Simulation problems

FIGURE 2.3: From a system analysis perceptive, applications of evolutionary computation fit in three problem domains. Reproduced from [54].

In the following, we focus on three particular domains.

Black-Box Optimisation

Black-Box Optimisation (BBO) problems refer to the situation where the optimiser aims to find the optimal inputs that minimise (or maximise) the outputs of an objective function without knowing its internal structure. According to [86], a mathematical representation of BBO could be:

> In a search space Ω, a candidate solution $\boldsymbol{x}$ is a N-dimensional vector, that is, $\boldsymbol{x} = (x_1, \ldots, x_N)$. For a black-box function $f : \Omega \to \mathbb{R}$, the objective is to find the optimal solution $\boldsymbol{x}^* \in \Omega$ that leads to the global minimum[2] $f(\boldsymbol{x}^*)$. The optimiser is supposed to sample Ω to find a sufficiently good solution x that has a small value of $f(\boldsymbol{x}) - f(\boldsymbol{x}^*)$.

[2]A maximisation problem could be converted to a minimisation problem by applying $-f$

The challenges of solving BBO problems are [87]:

- *high-dimensional*. The objective function's dimension, N, results in an exponential explosion in the search space, which is also identified as the *curse of dimensionality* in [88].
- *non-separable*. The objective parameters in $f(x_1, \ldots, x_N)$ are coupled. Thus the optimisation cannot be simplified as multiple sub-processes that are tractable.
- *highly multi-modal*. The objective function has a landscape with multiple local optima. Hence, the optimiser could be easily trapped in the local optima and missed the global one.
- *multiple objectives*. The objective function is usually formulated with multiple objectives and/or constraints in real-world problems. Hence, the search space is divided into several regions, which increases the difficulty of finding the global optimal.

As the representation of f is unknown, operational research methods are not applicable. Moreover, as the derivative of f is not accessible, gradient-based methods are not feasible. However, given the nature of evolutionary algorithms (EAs), they are best suited in solving BBO problems.

System Identification

In the control community, system identification is the process of building dynamical models from observation data, including input and output signals of the target system. It shares the basic features of the common modelling process in other areas [89].

Generally, system identification concerns two situations: *black-box models*, where no prior knowledge is available, either of the structure or of the parameters; *grey-box models*, where the system structure is known, but no information about the

parameters is known, or only a few parameters are known. EAs have been largely considered in these two situations since the research done by Kristinsson and Dumont [90]. They applied genetic algorithms (GAs) in both continuous-time and discrete-time identification problems and reported better results compared with traditional methods but with a notable high computational cost.

There are two important problems that can be distinguished in system identification: *modelling* and *estimation*. In both areas, EAs has been subsequently implemented:

- Modelling. In system identification, a model is composed of the input, output and the delay term. EAs contribute a simple way to find the desired structures linking these terms. Non-linear AutoRegressive Moving Average eXogenous (NARMAX) [91] have been widely used to represent non-linear systems, for which Fonseca et al. [92] applied an EA to select terms in polynomial models. In further studies, EAs were also used to identify NARMAX models [93, 94]. Moreover, the tree-based framework of genetic programming (GP) naturally contributes the structure identification. Particularly, Marenbatch et al. [95] proposed a general approach to apply GP with block diagrams, including the time-delay, switch, loop and domain-specific components. Gray et al. [96] applied GP to find block-like and equation-like representations of non-linear models. Furthermore, multi-objective GP benefits in building multi-objective NARMAX polynomial models. For example, seven objectives and constraints, involving the number of terms, degree, lag, variance, prediction error and correlation aspects, were optimised at the same time in [97]. However, the complexity in structure may cause the growth of trees in GP, which is also known as the *bloating* problem [98].

- Estimation. After structuring a model of the target system, the next step is to estimate the unknown parameters inside the model. Traditional methods are least-square estimation for linear models or gradient-based recursive techniques [99]. For non-linear models or situations where the gradient is not applicable, or problem with multiple local optima, EAs potentially fit in. For

example, the evolutionary strategy (ES) was implemented to estimate parameters in friction models [100]. Usually, the modelling and estimation are achieved simultaneously, such as in the evolutionary NARMAX approaches discussed above.

Evolutionary Robotics

Evolutionary robotics (ER) is a concept of designing robots with embodied intelligence and aims to improve the robustness and adaptability of robots [101, 102]. An essential feature of ER is that the robot is considered as a whole, distinguished with traditional approaches where many aspects, including the morphology, sensory mechanism, actuation system and controller, are designed separately and simultaneously with high interdependence [103]. The procedure of implementing ER (as shown in Figure 2.4) is [102]: (1) converting the genotype in the initial population into the phenotype feature of a robot, for example, the controller or morphology; (2) placing the robot in an environment and observing its behaviour; (3) determining the fitness based on observations; (4) the fitness is transferred back to evolutionary algorithms for selection, and the population is then updated via reproduction; (5) the above cycle is repeated until termination.

ER originated from engineering and biology, then contributes to both areas:

- In engineering, ER breaks the boundaries of individual fields in robotics, integrates multiple sub-processes in robot design, and thus introduces new possibilities and perspectives. For example, instead of designing the morphology as a prior, Lipson and Pollack [104] combined the morphology and control in the evolutionary process. Bongard [105] considered the impact of morphological changes on the behaviour and found that more robust behaviours emerged as the complexity in morphology increased.

- In biology, ER provides an alternative way to understand biological evolution and allows researchers to study hypotheses efficiently in a realistic environment or simulation. Particularly, AVIDA [106] and AEvol [107] are

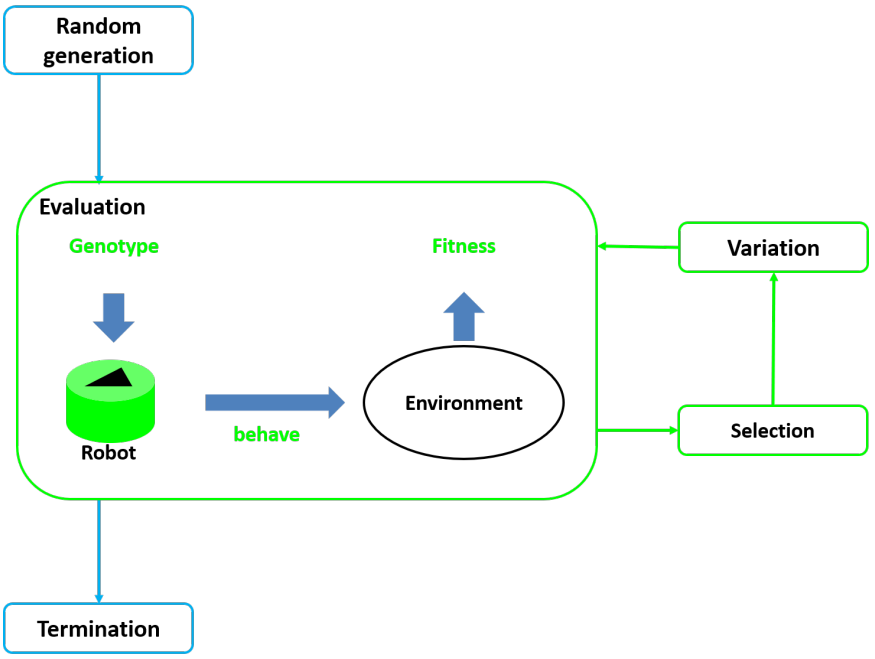


FIGURE 2.4: An illustration of evolutionary robotics in flow-chart. After initialising the original population, the process proceeds to evaluate the genotype in the phenotype level - the robot's behaviour, determine fitness on the basis of observations and operate the reproduction, and then repeats until the termination criteria are met. Adapted from [102]

.

simulation tools to study the evolution of bacterial. Also, ER was applied to study some unique properties, such as how cooperation evolves [108, 109] and when efficient communication strategies emerge [110].

One challenge in ER and in robotics generally is to transfer principles into physical devices. Researchers attempted to apply the evolutionary process directly to real robots, like Aibo robots, who learn to walk in the real-word with ER techniques [111]. However, due to the need for long training time and continual human intervention and safety reasons [112], the design in ER is usually conducted in simulations. The reality gap [113] is thus introduced between the outcomes in a simulated environment and the performance in real-world systems. ER opens up many angles to tackle this issue. For example, mixing the transferability into the objectives of ER led to optimal solutions in both simulation and reality [113]. This also introduces resilience in ER, such as a resilient robot updates its morphology model after a motor or mechanical failure and adapts its controller automatically [21]. Another challenge is online learning, which is about the adaptability of

ER in dynamical or unknown environments. Online ER has been considered both with single robots and multiple robots. The resilient robot introduced in [21] is a try to address online learning on a single robot. Attempts with multiple robots lead to the distributed online ER, also known as embodied evolution [114]. The nature of evolutionary algorithms makes it possible to distribute over a population of real robots [115] or virtual ones [116].

ER provides new insights into the research in the AI community, that is, the emergence of intelligence. Similar to the human brains evolved in biological evolution, ER is motivated to build a robot "brain" in a long evolutionary process, which could run for the robot's lifespan to maintain the adaptability [102]. The cognitive framework, proposed by Bellas and Duro [117], allows a robotic system to adapt to its environment in a life-long evolutionary process. The learning of the robot's brain, in turn, also inspired neuroscientists with models to understand mechanisms of the human brain [118].

2.3 Swarm Intelligence

2.3.1 Natural Swarms

Swarms have been widely seen in nature, for example, a group of ants, a flock of birds or a school of fish. From a top view, a swarm is a group of simple agents, each of which is governed by local rules but achieves some emergent behaviours as a group through cooperation with other agents [119]. There are two core mechanisms in swarm systems:

- *Self-organisation* is defined as "a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system" [120, p.8]. In other words, patterns in a swarm system are formulated via a bottom-up process, without top-down

procedures. One should note that interactions among individuals can happen directly or indirectly by modifying environmental conditions that affect others' behaviours, which leads to the second mechanism: stigmergy.

- *Stigmergy* is defined as "one means of information flow within a decentralised system that involves gathering information from the shared environment" [120, p.60]. In other words, it allows individuals to communicate with each other through changes to the environment which enables the information to be passed beyond space and time. Stigmergy is first observed as significant stimuli in the insect colony [121] and later named as a particular way of communication [122]. Stigmergy can encourage complex patterns by considering the environmental states and the individual distribution both in the spatial and time domain [123].

Swarm Intelligence (SI), firstly proposed by Beni and Wang [124] in 1989, then refers to the collective behaviour of such swarm systems, both in natural and artificial forms. For example, in nature, a swarm of animals is able to solve problems that are difficult or impossible for single individuals, for example, a group of ants foraging for food, a flock of birds or a school of fish avoiding predators during travels. In the artificial field, SI inspired two main applications: swarm optimisation and swarm robotics, as discussed in the following sections.

2.3.2 Swarm Optimisation

We briefly review two main techniques: ant colony optimisation (ACO) and particle swarm optimisation (PSO), which are nature-inspired and developed for solving combinational problems and real-valued problems, respectively. They were both introduced in the 1990s and attracted much attention in their own fields. Many variants appeared later on both sides, and some have been applied in the other's domains. For example, ACO was applied in continuous domains without fundamental modifications in structure [125]; A PSO-based algorithm was implemented to solve the flow shop scheduling problem [126].

Ant Colony Optimisation

Ant Colony Optimisation (ACO) was introduced by Dorigo et al. [127] in 1991 and later named in [128, 129]. It is a search algorithm inspired by the foraging behaviour of ants (e.g. the stigmergy mechanism described above). When ants search for food, they tend to randomly explore the area around their nest and leave pheromones on the path when they carry food back home. The amount of pheromone represents the quality or quantity of the food so that another ant is guided back to the food source.

Similar to the capability of real ants, the algorithm is implemented with a pheromone model of virtual ants to sample the search space with a probability distribution. In ACO, virtual ants are dispersed randomly among nodes. The searching starts with each ant moving between nodes randomly, but each node is not visited twice. After one tour is completed, each ant follows the same route and leaves pheromones on its way back to the origin point. The strength of the pheromones indicates the distance of the tour the ant has completed. For the next tour, each ant prefers to choose the node with the strongest pheromone next to its current node. To avoid the convergence towards the local optimal, the pheromone fades slowly if no ant reinforces it.

ACO was initially proposed to solve hard combinational problems, that is, finding the optimal combination of a finite set of problem components. Classic examples include scheduling and routing problems. In [127], ACO is applied to the travelling salesman problem where a salesman needs to plan the shortest tour of a number of cities. Other applications of ACO are like the assignment optimisation in [130], scheduling solution in [131] and vehicle routing plan in [132]. For more implementation details about ACO, we refer to [133].

Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) was introduced by Kenedy and Eberhart [134] in 1995. It was designed for optimising parameters of non-linear equations in

continuous space, such as the weights of neural networks in its initial application. The inspiration source of PSO is from a group of animals locating a desirable pace as a whole, for example, bird flocks searching for food. PSO is implemented with a swarm of agents, called particles. Each agent's fitness is determined by its location within an N-dimensional search space. Communication between agents is available to exchange information about their fitness during moving in the search space.

In the original PSO, for a particle i, it is defined with three vectors: its location in the N-dimensional search space $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,N})$, its velocity $\boldsymbol{v}_i = (v_{i,1}, \ldots, v_{i,N})$ and the best location it has identified so far $\boldsymbol{x}_i^{best} = (x_{i,1}^{best}, \ldots, x_{i,N}^{best})$. All particles are initialised at a random location based on a uniform distribution and with a random velocity. Particles are updated at each time step t by the following rules [135]:

$$\begin{aligned} \boldsymbol{v}_i^{(t+1)} &= \boldsymbol{v}_i^{(t)} + c_1 \boldsymbol{\delta}_1^{(t)} (\boldsymbol{x}_i^{best} - \boldsymbol{x}_i^{(t)}) + c_2 \boldsymbol{\delta}_2^{(t)} (\boldsymbol{x}_{goal}^{best} - \boldsymbol{x}_i^{(t)}) \\ \boldsymbol{x}_i^{(t+1)} &= \boldsymbol{x}_i^{(t)} + \boldsymbol{v}_i^{(t)} \end{aligned} \tag{2.1}$$

where c_1 and c_2 are constants; $\boldsymbol{\delta}_1^{(t)}$ and $\boldsymbol{\delta}_2^{(t)}$ are constant N-dimensional vectors that are randomly generated at time step t; $\boldsymbol{x}_{goal}^{best}$ is the best location found by neighbours of particle i. Hence, each particle moves towards a combined objective of its previous direction, the direction of the best location it found and the direction of the best location its neighbours found. For more details about PSO, we refer to [136, 137].

2.3.3 Swarm Robotics

Another application motivated by swarm intelligence is swarm robotics:

> "[...] is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collection behaviour emerges from the local interactions among agents and between the agents and the environment." [138]

From the definition of swarm robotics, a swarm robotics system is composed of several main characteristics: *autonomous* robots, *situated* in the environment, *cooperate*, through *local sensing and communication*, with no access to *centralised control* and/or *global knowledge* [139].

Motivations

The motivation for developing swarm robotics comes from the properties of swarm behaviours observed in nature. In general, these behaviours are unique for robustness, scalability and flexibility.

- Robustness. Generally, it is the ability of a system to maintain operation when undesired disturbances occur. In swarm robotics, these disturbances usually come from the failure of some individual agents, for example, the malfunctions of the sensor, motor or communication units, which could happen both at the hardware and software level. The robustness of swarm animals is achieved by the redundancy of individuals and the absence of a leader. Similarly, the robustness of swarm robotics is promoted by a large number of simple and homogeneous individual agents. Each agent behaves locally to tackle small tasks and can be replaced by other agents if it fails. However, as the "No Free Lunch" theorem, this advantage does not come by default and needs to be properly considered [140].

- Scalability. In swarm robotics, it refers to the ability to maintain emergent behaviour despite small changes in the number of agents. The scalability of swarm animals is promoted by local sensing and communication [139]. Similarly, the coordination mechanism is essential to enable a swarm robotic system to perform desirably with different group sizes [138].

- Flexibility. In general, it is the ability of a system to achieve effective performance in different scenarios. In social animals, such as honeybees, multiple tasks can be conducted by a group of individuals. The flexibility of swarm robotics is benefited from the simplicity of each agent's behaviour.

Applications

The research on swarm robotics covers a wide range of problems, including pattern formation (e.g. aggregation, lattice formation, covering of areas and mapping), specific problems (e.g. goal-searching, homing and forging) and complex behaviours (e.g. transporting, mining and flocking) [124]. We briefly discuss two specific problems that are related to the work in this thesis.

Object-clustering is a self-organised aggregation problem. The behaviour is observed in ant colonies sorting the food. The study of this problem with swarm robots was firstly proposed by Deneubourg et al. [141], where a decentralised algorithm, based on a probabilistic finite state machine, was implemented in simulation. Beckers et al. [142] firstly demonstrated with physical robots, each of which was embedded with two infra-red proximity sensors and a gripper. The robot's behaviour followed a deterministic finite state machine. Later, the object-clustering problem has been subsequently studied in the literature. In particular, Gauci et al. [24] proposed a simple memory-less mechanism to cluster objects, demonstrated both with virtual and physical robots. The robot's controller basically maps its binary sensor reading to its wheel velocities.

Shepherding is the process of guiding a group of dynamic agents towards the desired location, for example, dogs herding sheep into the sheepfold or officials guiding the human crowd moving in one direction. The process can be done with one or multiple shepherd agents. Vaughan et al. [143] used a single robot to herd real ducks, which is considered one of the earliest studies of the shepherding problem with robots. Lien et al. [144] suggested multiple robots could solve the problem more efficiently than a single one. In the latter study, different shepherding algorithms have been developed for swarm robotic systems [145, 146]. Particularly, a similar memory-less mechanism like the one in [24] was proposed to herd sheep-behaving-like robots with shepherd robots [147].

Evolutionary Swarm Robotics

Designing controllers that lead to emergent behaviours of a swarm of robots rely on a definition of the behavioural rules that results in a desired pattern of the swarm. This is also referred to as the *design problem* [148]. The challenge to solve this problem is to decompose the global behaviour into local interactions between the individual robots and between robots and the environment where the robots operate. However, it is usually difficult to predict what emergent behaviours that a given set of behavioural rules can lead to [149]. Thus, it is difficult to achieve decomposition.

Evolutionary robotics offers an effective way to deal with the design problem by eliminating arbitrary decomposition [148]. This leads to the field of evolutionary swarm robotics (ESR). ESR takes account of the swarm robotics system as a whole. It evaluates the definition of the behavioural rules based on the emergence of the desired global behaviour. The procedure of ESR can be simplified as:

> "[...] the controller encoded into each genotype is directly evaluated looking at the resulting global behaviour. The evolutionary process is responsible of selecting the 'good' behaviours and discarding the 'bad' ones. Moreover, the controllers are directly tested in the environment [...]" [148, p.50]

Applications of ESR focus on multiple tasks, for example, evolving the aggregation behaviour and the coordinated motion by Dorigo et al. [150] and evolving solitary transport and group transport behaviours by Groß and Dorigo [151]. Apart from the standard evolutionary approaches, some distinctive techniques have contributed to the research of swarm robotics as well.

Novelty search, proposed by Lehman and Stanley [152] in 2011, replaced the objective-based evolution in standard evolutionary computation with a dynamic measurement that evaluates the novelty of candidate solutions among the current

population in order to avoid premature convergence. Gomes et al. [153] demonstrated this approach with swarm robotic tasks and showed that simple and diverse solutions are evolved.

Minimal surprise, proposed by Hamann [154] in 2014, positioned the selection pressure on a world model that predicts the robot's future perceptions instead of on the behavioural model. In the original application, it evolved two neural networks: one for the robot's controller and the other one for predictions. Results showed that by minimising prediction errors, some basic behaviour, like aggregation, dispersion and flocking, were usefully evolved. In the later work, this approach has been applied to evolve complex behaviours, such as the collective construction [155] and the resilient self-assembly after damages [156].

2.4 Problem Formalisation

Machine learning is a discipline that addresses two correlated questions — a scientific question: "What are the fundamental statistical-computational-information-theoretic laws that govern all learning systems, including computers, humans, and organisations?" and an engineering question: "How can one construct computer systems that automatically improve through experience?" [157]. *Experience* refers to the information that a learner obtained in the previous learning. Such information typically consists of electronic data that could take the form of human-labelled training sets or other forms of data collected from the environment via interactions. Since the success of a learner relies on the quality and size of the data used for analysis, the learning approaches are data-driven methods that merge statistics, probability and optimisation into computer science [158]. Broadly speaking, machine learning methods can be defined as a set of computational methods "that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!)" [159, p.1].

Within artificial intelligence, the definition gives one of the main objectives of machine learning is to design algorithms that predict accurately on unseen data, which leads to a broad range of practical applications for wide areas, including computer vision and robot control. Recently, a new class of algorithms has progressed dramatically, where models and discriminators are improved competitively in a game setting: The discriminator's task is to correctly classify the data generated by the model and the one from the training set; The model's task is to produce data that is miss-classified as the training data by the discriminator. The most popular algorithm designed with this setting is Generative Adversarial Networks (GANs) [2] which have been widely applied and studied in the area of image generation. The first attempt to infer models in a competitive setting with discriminators was introduced in [5]. The method was named *Turing Learning* in the subsequent work [4]. *Turing Learning* has been applied to study the behavioural rules of robotic agents since its inception. In the following sections, we start with discussions about GANs and previous works on *Turing Learning* and then formalise the *Turing Learning* algorithm for this thesis.

2.4.1 Generative Adversarial Networks

For most applications of machine learning, it is easier to train a system with desired input-output examples than to design it manually to meet the desired outputs for all possible inputs, which leads to the most commonly used machine learning method — *supervised learning* [157]. By definition, supervised learning requires an output label provided by a human supervisor for each input example. From a probabilistic perspective, the goal of supervised learning is to find a function $f(\boldsymbol{x})$ that maps an input vector $\boldsymbol{x}$ to an output label y, based on N input-output examples in a training set $\mathcal{T} = (\boldsymbol{x}^{(i)}, y^{(i)})_{i=1}^{N}$ [159]. To achieve that, existing approaches often rely on a training set comprising millions of examples. While, another widely studied machine method, *unsupervised learning*, reduces both the human effort and the training size. The goal of unsupervised learning is not clearly

defined. Generally, it aims to find useful patterns from an unlabelled training set $\mathcal{T} = (\boldsymbol{x}^{(i)})_{i=1}^{N}$.

One approach to implementing unsupervised learning is *generative modelling*, where GANs are categorised. Generative modelling aims to learn a model distribution p_{model} of training samples $\boldsymbol{x}$ that matches an unknown distribution $p_{\text{data}}(\boldsymbol{x})$. Specifically, a function $p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})$ optimises its parameters $\boldsymbol{\theta}$ to estimate p_{data} as closely as possible. In traditional statistics, explicit density functions have been widely used to model data distributions. The optimisation task is then to search for parameters that fit best p_{data}. A common approach to achieve that is the maximum likelihood estimation which minimises the Kullback-Leibler divergence between p_{model} and p_{data}. The explicit density modelling works well for simple representations, but for complex ones, such as the deep neural network, it is computationally difficult to solve. Alternatively, approximation techniques are usually applied to deal with this intractable issue. However, either the deterministic approximations, like the one used in variational autoencoder [160], or stochastic approximations, like the Markov chain approximations used in Boltzmann machine [161], would limit the quality of models even with an ideal optimisation method. Some generative models avoid explicit functions, instead they directly sample from the p_{model}. For example, the generative stochastic network [162] relies on Markov chains transition operator running multiple times to generate samples. However, when handling samples with a high dimension, such as realistic images, the Markov chain would become inefficient, and the convergence could be very slow and unclear [163]. GANs were introduced to build deep implicit generative models, where samples could be generated in a single step, without the need for heavy computational effort and Markov chains.

As introduced above, GANs are based on a game setting of models and discriminators. The model, often called the *generator*, is able to draw samples from p_{model}. The discriminator is able to examine samples and predict whether the input sample is from p_{data} or from p_{model} by running the generator. Within this setting, the training process involves the training of both the generator and the discriminator.

The generator is trained to fool the discriminator; simultaneously, the discriminator is trained as a classifier to label input samples. In [3], GANs are interpreted as a competition between counterfeiters and police:

> "[...] the counterfeiters make fake money while the police try to arrest counterfeiters and continue to allow the spending of legitimate money. Competition between counterfeiters and police leads to more and more realistic counterfeit money until eventually the counterfeiters produce perfect fakes and the police cannot tell the difference between real and fake money."

The original promise of GANs is to generate high-quality realistic images. Besides the straightforward generation, GANs have shown their potentials in accomplishing many other tasks in computer vision, for example, image synthesis [164], image-to-image translation [165] and superresolution [166]. For a broad review of applications of GANs, we refer to [167, 168]. Even though GANs and their variants have been widely studied in recent years, the convergence of GANs is still an open question. Major problems are: *mode collapse*, where only a single mode of the distribution can be learned by the model; *partial mode convergence*, where only a small set of modes of the distribution can be captured by the model; *vanishing gradient*, where the gradient for model vanishes if the discriminator dominates the game, leading to suspended optimisation both of the model and discriminator. Meanwhile, the discriminator may choose to *forget* parts of the input space that are not covered by the model. Many researchers have focused on the convergence of GANs: Arjovsky and Bottou [8] conducted a theoretical analysis of the training dynamics; Arora et al. [9] discussed the existence of the equilibrium of and introduced a new framework MIX+GAN to improve the performance; Unterthiner et al. [10] proposed Coulomb GANs and proved that only one Nash equilibrium exists in potential fields; Nagarajan and Kolter [11] discussed the gradient-based optimisation process and showed that with a regularisation gradient decent GANs are locally stable; Mescheder et al. [12] analysed the numerics of training algorithms and presented an improved convergence. On the other hand, Oliehoek et

al. [169] provided a novel perceptive. They closely related GANs to game-theoretic methods and formulated the original GANs as finite zero-sum games: GANG (e.g. Generative Adversarial Network Games), so that the convergence is guaranteed. Meanwhile, researchers also investigated the coevolution effect on GANs [13–16]. In general, they implemented multiple sub-GAN frameworks and studied the co-evolutionary process of multiple generators and discriminators. The properties of coevolution have shown improved performance on GANs.

2.4.2 Turing Learning

Considering the definition of machine learning, a learning problem can be considered the problem where the measure of performance on some tasks is improved through the training experience. Thus, conceptually speaking, machine learning algorithms are searching techniques that target to find a program from a large candidate space to optimise the performance metric [158]. In some cases, the difficulty is not only to design an effective and efficient algorithm but to define a suitable performance metric. The quality of the metric can determine the quality of the obtained program or even direct the searching progress.

In the domain of system identification, one limitation of the current methods is also that they depend on predefined metrics, such as the square errors, to evaluate how close is the model to the system under investigation, which can be challenging for complex systems. *Turing Learning* was first introduced as a metric-free system identification method to avoid this issue. As introduced above, *Turing Learning* shares a similar game setting with GANs, while its inspiration was actually from the Turing test. Considering the setup in Figure 2.1, the objective of the model, referring to player $\mathcal{A}$, is to imitate the behaviour of the system under investigation, referring to player $\mathcal{B}$, and the objective of the discriminator, referring to player $\mathcal{C}$, is to distinguish between the model and system through observations.

Originally, *Turing Learning* was motivated to infer the behaviour of a natural or artificial system, especially a swarm system. The identification of collective behaviours in a swarm system could be particularly difficult as individuals interact with both the environment and each other, and their motion tends to be stochastic [170]. In the early works on *Turing Learning*, the method has been applied to infer the deterministic behaviour of an agent in a simple environment [5], and to infer the behavioural rules of a swarm of robots both in a simulated environment [6] and a physical environment [4]. In [5], the discriminator was allowed to perform a pre-designed sequence of interactions to control the environmental stimuli, and such interaction encouraged a more accurate inference. Moreover, in [4], it has shown that *Turing Learning* outperforms a metric-based system identification method in terms of the inference of swarm behaviours. In the most recent work, *Turing Learning* was implemented to model and generate realistic human movements [171].

Since its inception, the *Turing Learning* method has used coevolutionary algorithms, where a population of models and a population of discriminators are optimised at the same time. While, in principle, other optimisation algorithms can also be implemented, considering the successes of previous works, we continuously investigate the coevolutionary approach of *Turing Learning*.

2.4.3 Turing Learning Formalisation

The core idea of *Turing Learning* and GANs are similar, that is, optimising two components simultaneously in a competitive game setting, although these two methods were developed separately and in different contexts. Assuming a Turing perceptive, Groß et al. [7] considered GANs and their variants as members of the *Turing Learning* family and generalised the defining features of *the Turing Learning* algorithm as: a *training agent*, a *model* agent, a *discriminator* agent, a *process* where the discriminator *observes* or *interact* with the other two agents and an *optimisation* mechanism. One should note that the interaction becomes

one of the central features. These features allow the *Turing Learning* algorithm to be tailored for a wide range of problems.

In the following, we provide a formalisation of the *Turing Learning* algorithm as the standard problem we discuss and investigate in this thesis.

Model

A model $\mathcal{M}$ is represented as a generative function which is able to draw samples from a distribution p_{model}:

$$\mathcal{M}_{\boldsymbol{u}}(h), \boldsymbol{u} \in \mathcal{U}, \tag{2.2}$$

where $\boldsymbol{u}$ is a set of parameters that defines the model's strategy. $\mathcal{U}$ can be any set of data structures, for example, the N-dimensional real-valued search space $\mathbb{R}^N$, the integer search space $\mathbb{Z}^N$, the state space $\mathbb{S}_N$ of finite state machines, the trees $\mathbb{T}_N$ as used in genetic programming and a mixture of spaces. The input h is problem specific and can take any form, for example, a noise vector or time-series data. Given an input, a sample x can be generated by $x = \mathcal{M}(h)$. The main role of a model is to find a function $\mathcal{M}(h)$ so that p_{model} is similar (preferably identical) to a training distribution p_{data}.

Discriminator

A discriminator $\mathcal{D}$ is represented as a function which is able to classify whether a given sample is real (drawn from p_{data}) or fake (drawn from p_{model}):

$$\mathcal{D}_{\boldsymbol{v}}(x), v \in \mathcal{V}, \tag{2.3}$$

where $\boldsymbol{v}$ is a set of parameters that defines the discriminator's strategy. $\mathcal{D}$ is typically a neural network in practice, then $\mathcal{V}$ is a N-dimensional real-valued search space $\mathbb{R}^N$. Given an input sample x, $\mathcal{D}(x)$ returns a scalar that indicates the "realness" of x. The main role of a discriminator is to learn a function $\mathcal{D}(x)$ that outputs a high value when x is real and a low value when x is fake.

Two-Player Game

The game between the model and the discriminator is defined by two objective functions:

$$O_{\mathcal{M}}(\boldsymbol{u}, \boldsymbol{v}), O_{\mathcal{D}}(\boldsymbol{u}, \boldsymbol{v}) \to opt., \quad (\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{U} \times \mathcal{V}. \tag{2.4}$$

The goal of the model is to maximise $O_{\mathcal{M}}$, whereas the goal of the discriminator is to maximise $O_{\mathcal{D}}$. A Nash-equilibrium of the game occurs at a point $o(\boldsymbol{u}^*, \boldsymbol{v}^*)$ given by the two conditions:

$$\begin{aligned} \boldsymbol{u}^* &\in \arg\max_{\boldsymbol{u}} O_{\mathcal{M}}(\boldsymbol{u}, \boldsymbol{v}^*), \\ \boldsymbol{v}^* &\in \arg\max_{\boldsymbol{v}} O_{\mathcal{D}}(\boldsymbol{u}^*, \boldsymbol{v}). \end{aligned} \tag{2.5}$$

Roughly speaking, $O_{\mathcal{M}}$ encourages the model to generate samples that the discriminator classifies as real; $O_{\mathcal{D}}$ encourages the discriminator to correctly label the input sample as real or fake.

Examples

- GANs. In GANs, the model $\mathcal{M}$, often referred to as the generator $\mathcal{G}$, and the discriminator $\mathcal{D}$ are defined as differentiable functions represented by multilayer perceptrons.

  $\mathcal{G}$ takes a noise variable $\boldsymbol{z}$ as input, which is sampled from a prior distribution p_z, then generates a sample $x = \mathcal{G}(\boldsymbol{z})$. $\mathcal{D}$ outputs a single scalar in $[0, 1]$ representing the probability of the input sample to be real.

  The objective functions $O_{\mathcal{G}}$ and $O_{\mathcal{D}}$ are referred to as cost functions $J_{\mathcal{G}}$ and $J_{\mathcal{D}}$. Each player attempts to minimise its own cost. In the original version of GANs, $J_{\mathcal{D}}$ was defined to be the negative log-likelihood that the discriminator assigns real and fake labels:

$$J_{\mathcal{D}} = -\frac{1}{2}\mathbb{E}_{x \sim p_{\text{data}}}[\log \mathcal{D}(x)] - \frac{1}{2}\mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))]. \tag{2.6}$$

The original work of GANs offered two versions of the cost for the generator. One version, $J_{\mathcal{G}} = -J_{\mathcal{D}}$, leading to a minimax game of the objective function $O(\mathcal{D}, \mathcal{G})$:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} O(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{\text{data}}}[\log \mathcal{D}(x))] + \mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))]. \quad (2.7)$$

In practice, $\mathcal{D}$ rejects samples generated from $\mathcal{G}$ with high confidence early in learning when $\mathcal{G}$ is poor. In this case, $\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))$ saturates. The other version, instead of training $\mathcal{G}$ to minimise $\log(1-\mathcal{D}(\mathcal{G}(\boldsymbol{z})))$, trains $\mathcal{G}$ to maximise $\log \mathcal{D}(\mathcal{G}(\boldsymbol{z}))$. This objective helps to avoid the gradient saturation.

- Coevolutionary Framework of *Turing Learning*. In the coevolutionary framework of *Turing Learning*, both the model and the discriminator are represented as populations of candidate solutions, and the coevolution operates between the two populations.

  For the model population of size $P_{\mathcal{M}}$, an individual m_i with index i comprises a set of parameters $\boldsymbol{u}_i$ and defines a distribution p^i_{model}. Similarly, for the discriminator population of size $P_{\mathcal{D}}$, an individual d_j with index j comprises a set of parameters $\boldsymbol{v}_j$.

  The *Two-Player Game* then includes the games between each candidate model and each candidate discriminator. In the context of evolution, the objective functions $O_{\mathcal{M}}$ and $O_{\mathcal{D}}$ are usually referred to as the fitness functions $F_{\mathcal{M}}$ and $F_{\mathcal{D}}$. In the following, we use f_{m_i} and f_{d_j} to represent the fitness values for model m_i and discriminator d_j respectively.

  After a game finishes, the discriminator's output is simply used to determine the fitness value. Each candidate's final fitness is calculated as the normalised sum of the values it obtains from all games:

$$f_{m_i} = \frac{1}{P_{\mathcal{D}}} \sum_{j=1}^{P_{\mathcal{D}}} d_j(x \mid x \sim p^i_{\text{model}}), \quad (2.8a)$$

$$f_{d_j} = \frac{1}{2P_{\mathcal{M}}} \sum_{i=1}^{P_{\mathcal{M}}} d_j(x \mid x \sim p^i_{\text{model}}) + \frac{1}{2P_{\mathcal{M}}} \sum_{n=1}^{P_{\mathcal{M}}} d_j(x_n \mid x_n \sim p_{\text{data}}), \quad (2.8b)$$

where x_n is the n^{th} sample drawn from p_{data}. For convenience, $d_j(x) \in \{0, 1\}$, where 0 denotes that d_j classifies x as fake; 1, otherwise, as real. In other words, the discriminator is implemented as a regular binary classifier.

Throughout the evolutionary process, each candidate tries to learn a strategy in order to maximise its fitness and survive in its population.

Chapter 3

Inferring Sensor Configuration through Self-Discovery

3.1 Introduction

Turing Learning has been applied to infer behaviours, including those of swarms of agents [6], those of swarms of real robots [4] and those of humans [171], through observation. In order to fully reveal the behavioural features, high observability of the system is thus essential. However, when the observability is limited, the inference may not be reliable.

In [5], the discriminator used a pre-determined sequence of interactions to infer a simulated behavioural rule. Therein, it was shown that the algorithm performed better with interaction than with passive observation. However, such an interactive approach would not be able to learn all the features of complex systems. In this chapter, we extend the discriminator agent in *Turing Learning* with the ability to genuinely interact with the system under investigation, which makes it more like the interrogator in the Turing Test [172]. We assume that, by allowing the discriminator to influence the data sampling process, the algorithm could explore the system as it tends to and benefit the inference. Similar methods have been applied by ethologists to understand the relationships between animal behaviour

and its environment. In [173], biologists adjusted the environmental conditions during the experiments to study how the dung beetle behaves to adapt to the changes.

In this chapter, we focus on the active self-discovery of robotic systems. It is a process designed for the system continuously modelling its own morphology. For example, a four-legged robot was able to infer its topology and then use this model to perform desired motions [21]. Similarly, we investigate how a robot can use *Turing Learning*, particularly with the interactive discriminator, to infer its own sensor configuration through the process of self-discovery. To validate the advantage of the interaction, we compare the accuracy of inferred configuration from the interactive approach with that of the inferred configuration from the passive approach.

This chapter is organised as follows. Section 3.2 describes the methodology, including a detailed definition of the *Turing Learning* framework with a perspective of coevolutionary (3.2.1) and an introduction about the simulation platform used in this chapter and thesis(3.2.2). Section 3.3 presents a case study to demonstrate the framework, including a formal formulation of the problem (Section 3.3.1). Section 3.3.2 lists the implementation options of the algorithm that are chosen for this scenario. Section 3.3.3 describes the simulation setup designed for carrying out the investigation. The simulation results are presented in Section 3.3.4, including the analysis of the inferred model, dynamics of the coevolutionary, a comparison between the interactive setup and passive setups and an illustration of the discriminator's control pattern. Section 3.4 summarises the chapter.

3.2 Methodology

In this section, we start with an introduction to the *Turing Learning* framework, including the key components. We then describe the simulation platform used in this chapter. The same platform will be used to carry out investigations in the next two chapters, but with different settings according to the problems considered. As

for the implementation details of *Turing Learning*, we will specify them for each problem.

3.2.1 Coevolutionary Framework of Turing Learning

As introduced in Section 2.4.3, machine learning algorithms where models and discriminators are generated in a competitive setting can be considered as instances of *Turing Learning*. In this thesis, we take account of a coevolutionary framework of *Turing Learning*. Thus, there are a population of candidate solutions for both the model and discriminator. In the following, we present the framework with respect to the defining features of *Turing Learning*.

- *Training Agent*, $\mathcal{T}$. The training agent is the system under investigation. It could take any form (e.g. robots and human beings) that one expects to learn or imitate. The data samples obtained from it are referred to as *genuine data samples* (i.e. the training data). In this thesis, we focus on the study of robotic systems, where the training agent could be a single robot (Chapter 3 and 4)or a swarm of robots (Chapter 5). Therefore, the genuine data samples come from the robot's perception or motion, which are usually time-series data.

- *Model Agent*, $\mathcal{M}$. The model agent is assumed to have the potential to produce the equivalent behaviour as the training agent. In other words, the *counterfeit data samples* (i.e. the model data) generated by the model agent are expected to be indistinguishable from the genuine ones. In the context of robotic systems, the model agent is executed on a replica robot that has the same sensing and actuation ability as that of the system under investigation. For example, the model agent could determine how sensors are distributed on the robot's body (Chapter 3), how the robot perceives the environment (Chapter 4) or behaves (Chapter 5). It could take any form (e.g. vectors, neural networks and computer programs) as long as the representation is expressive enough to produce data with the same distribution as the training

data. In this thesis, we use parametric vectors to represent our models (i.e. sensor configuration vector in Chapter 3, sensor calibration vector in Chapter 4 and the controller vector in Chapter 5).

- *Discriminator Agent*, $\mathcal{D}$. The discriminator agent is assumed to have the ability to distinguish between genuine data samples and counterfeit data samples. It could take any form. In this thesis, given the sequential data from robotic systems, we represent the discriminator as a recurrent Elman neural network [174] with the structure shown in Figure 3.1[1]. The network has a context layer that can store the hidden neuron values and feedback these values to the input layer. This feature equips the network with an internal memory to process arbitrary sequential inputs. With the recurrent neural network, the discriminator is able to observe input samples for a certain period of time and make judgements. The activation function for all hidden and output neurons is a logistic sigmoid function that has the range $(0, 1)$:

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

  The discriminator is trained as a regular binary classifier. After the observation for a set period of time, the final value of one of the output neurons (O) is used to label the input data samples as either genuine ($O \geq 0.5$) or counterfeit ($O < 0.5$) samples. The network's memory is then reset for the next period of observation. Other output neurons depend on the design. For example, if the interaction is allowed, other neurons can be used to pass the information from the discriminator to the model and system (Chapter 3 and 4).

- *Evaluation Process*. The evaluation process is a process where $\mathcal{D}$ observes the data samples from either $\mathcal{T}$ or $\mathcal{M}$ for a certain period of time and then make judgements. While observing, $\mathcal{D}$ could also influence the sampling process by interacting with $\mathcal{T}$ and $\mathcal{M}$.

[1] In principle, other networks could also be used to process the time-series data, such as the long short-term memory (LSTM). Given the success of the Elman network in previous works of *Turing Learning*, we continuously use it in this thesis.

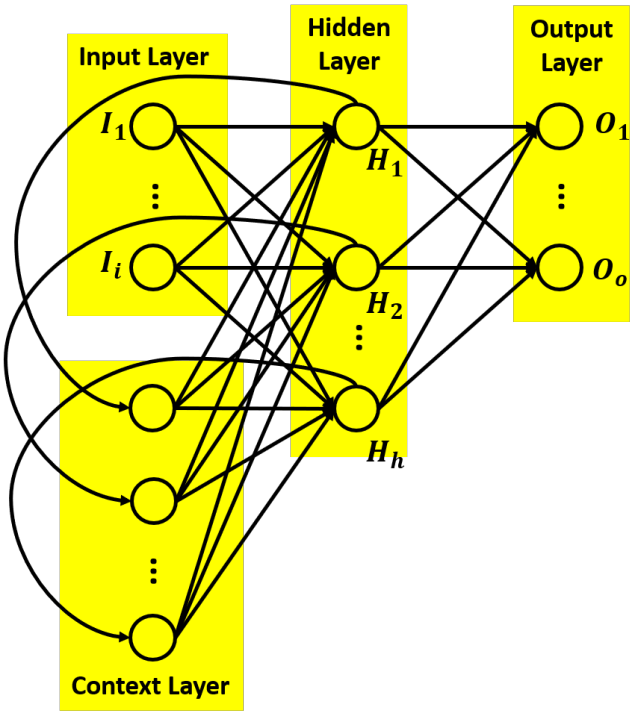


FIGURE 3.1: An Elman Neural Network with i inputs, h hidden neurons and o outputs. It can be considered as a feedforward network with an additional context layer. Considering a biased network, it has $(i+1)\cdot h + h^2 + (h+1)\cdot o$ parameters in total.

- *Optimisation Process.* $\mathcal{D}$ and $\mathcal{M}$ are optimised at the same time. According to the classification made by $\mathcal{D}$ after a period of observation, $\mathcal{M}$ is rewarded when $\mathcal{D}$ mistakenly labels its data samples as genuine. Otherwise, $\mathcal{D}$ is rewarded when it labels input data samples correctly, that is, labelling data samples from $\mathcal{T}$ as genuine and labelling data samples from $\mathcal{M}$ as counterfeit. As introduced in Section 2.4.3, we implement coevolutionary algorithms as the optimisation methods, for example, the $(\mu + \lambda)$ evolution strategy with self-adaptive mutation strengths in Chapter 3 and 4 and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) in Chapter 5. The optimisation process is composed of two sub-processes: one for the population of $\mathcal{D}$ and the other one for the population of $\mathcal{M}$. These two sub optimisation processes run separately and don't interact with each other except when fitness values are calculated. Hence, the synchronisation challenges as introduced in Section 2.2.2 might be raised for the coevolutionary of $\mathcal{D}$ and $\mathcal{M}$. In our case, the synchronisation comes from the evaluation process:

Assuming the population sizes for $\mathcal{D}$ and $\mathcal{M}$ are $P_{\mathcal{D}}$ and $P_{\mathcal{M}}$ respectively, each of the discriminator candidates is evaluated once with each of the model candidate, and additional $P_{\mathcal{M}}$ times with $\mathcal{T}$ as well. After all evaluations finish, the fitness value for each candidate can be calculated according to Equation 2.8. The evolution of each population then proceeds based on the fitness values of all candidates. The optimisation process continues until the termination criteria is met (e.g. a certain number of generations).

3.2.2 Simulation Platform

We constantly use the open-source Enki library [175] in this thesis, which has been verified broadly to model kinematics and two-dimensional dynamics of rigid objects. As for the robot agent, we use the built-in model of the e-puck robot [176]. Figure 3.2 shows a physical e-puck robot. In Enki, it is represented as a cylinder of diameter 7.4 cm, height 4.7 cm and weight 152 g. It has two symmetrically aligned differential wheels with the inter-wheel distance of 5.1 cm. Each wheel's ground velocity can be set within [-12.8, 12.8] cm/s. Random noise is applied in Enki for each wheel velocity by multiplying the velocity value with a random number chosen uniformly in the range (0.95, 1.05). The robot has a few sensors embedded on its cylindrical body, for example, infrared proximity sensors. We also extended the robot model with some additional sensors, such as a laser-based distance sensor described in Chapter 4 and a binary line-of-sight sensor described in Chapter 5. In all simulations presented in this thesis, the control cycle is 0.1 s, and physics is updated 10 times per control cycle.

According to [177], given a pair of wheel velocities, the differential-wheel robot, for example, the e-puck robot, moves in a circular trajectory as illustrated in Figure 3.3. Assuming the robot moves from position a to position b with its left and right wheel ground contact velocities of v_l and v_r respectively, its actual trajectory follows a circular arc centred at O with a radius R and an angular speed

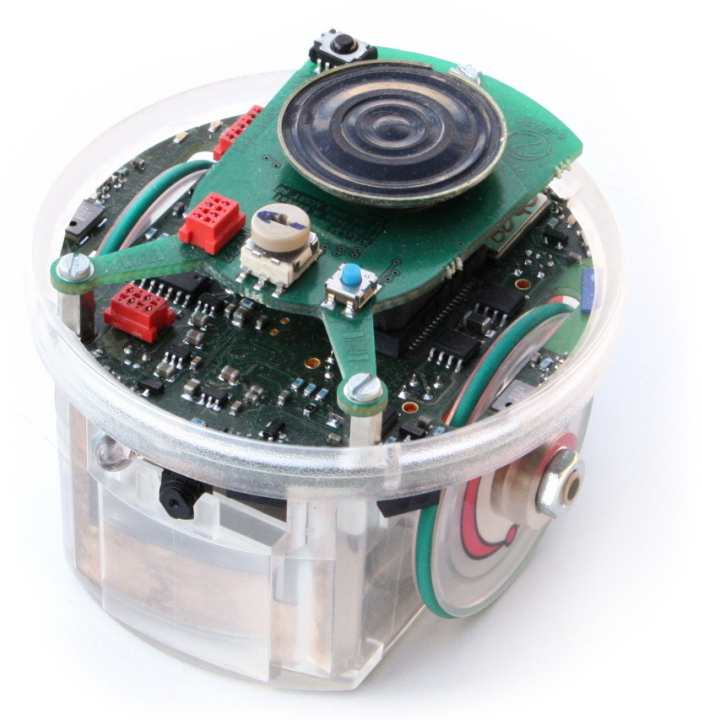


FIGURE 3.2: An e-puck robot. It has two differential wheels and a circular body with diameter 7.0 cm and height 4.5 cm, approximately. The robot is embedded with multiple sensory capabilities, for example, eight infrared proximity sensors and a VGA colour camera. Image used under the Creative Commons License.

ω:

$$R = \frac{l}{2}\left(\frac{v_r + v_l}{v_r - v_l}\right),$$
$$\omega = \frac{1}{l}\left(v_r - v_l\right),$$

where l is the inter-wheel distance.

3.3 Case Study

In this section, we present a case study of a robot inferring its own sensor configuration and show that the interactive discriminator could improve the model accuracy. We start with a formal definition of the problem considered in this

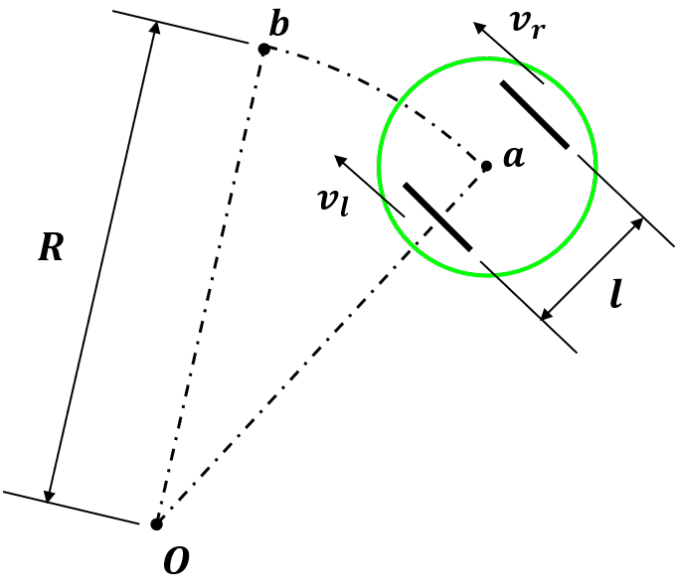


FIGURE 3.3: Kinematics of differential-wheel robot. From position a to b, the robot (green) follows a circular trajectory. R indicates the radius of the curve which is centred at O. v_l and v_r are the left and right wheel velocities along the ground, respectively. l is the inter-wheel distance.

chapter, then explain the implementation details of the framework described in the above sections.

3.3.1 Problem Formulation

Given the defining features of *Turing Learning*, if the training agent is a physical system while the model agent is represented in simulation, the issue of the *reality gap* needs to be addressed. It is a well-known problem in robotics: Often, behaviours that work well in simulation do not translate effectively into real-world implementations [113]. This is because simulations are generally unable to capture the full range of features of the real world, and therefore make simplifying assumptions. Yet, simulations can be important and are widely used to speed up the planning and optimisation process, even on-board a physical robot. Solutions to reduce the gap could be improving modelling accuracy of the robot-environment interactions [178], or of reality properties that are relative to the desired behaviours [179]. We assume that our method has the potential to benefit

this process. To validate that, we investigate how a robot can use *Turing Learning* to improve the accuracy of a simulation model of itself through a process of self-discovery. In a practical scenario, the inference could take place on-board a physical platform. For convenience, we use the simulation platform introduced above.

Specifically, we consider an e-puck robot inferring its sensor configuration. The robot perceives the surrounding environment with eight infrared proximity sensors that are distributed on its body, as shown in Figure 3.4. The sensors provide noisy reading values $(s_1, s_2, \ldots, s_8)$ in the range of $[0, 3000]$, but do not provide the information about the distance. The reading value indicates how close the robot is to the perceived objects (the larger the readings, the closer the objects). We assume the robot does not know where its sensors are placed. Similar situations are common in practice as sensors may not be precisely mounted, or the sensors' locations may change due to collisions with other objects during operation or might be reconfigured by the robot itself. The sensor configuration can be described with orientations relative to the robot's front direction and displacements from the robot's centre, which is denoted as follows:

$$\boldsymbol{q} = (\theta_1, \theta_2, \ldots, \theta_8, d_1, d_2, \ldots, d_8)\,, \tag{3.2}$$

where $\theta_i \in [-\pi, \pi]$ represents the orientation of sensor i, and $d_i \in (0, R]$ (R is the robot's radius) represents the displacement of sensor i. The inference task is thus specified to finding the $\boldsymbol{q}$.

3.3.2 *Turing Learning* Implementation

Given the defining features, the *Turing Learning* framework can be applied in a wide range of problem domains. In the following, we present the implementation options of the framework in a general view and introduce the options designed specifically for this case study. In different scenarios, these options allow users to choose the appropriate and familiar ones for the given problem.

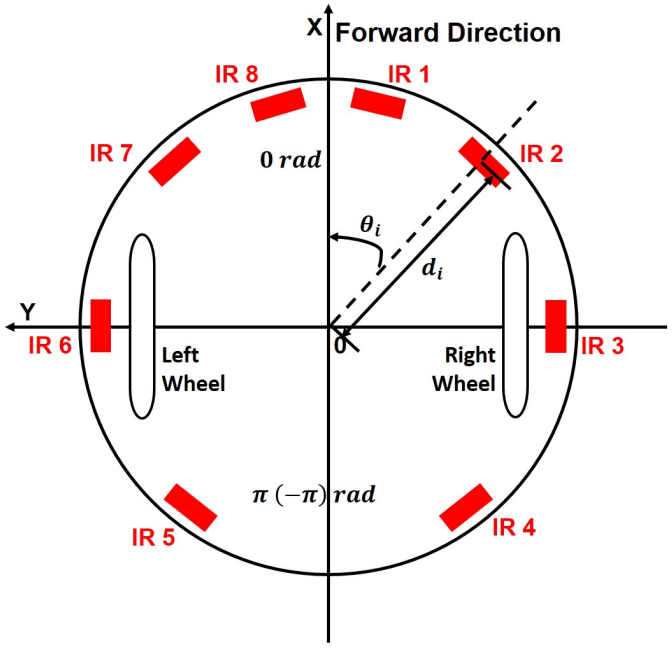


FIGURE 3.4: The e-puck robot discovers its environment via eight infrared proximity sensors (IR 1-8). We assume the robot is unaware of the configuration of these sensors and has to infer it. Each sensor's location is determined by the orientation (θ) and displacement d in the X-Y frame. The configuration is thus composed of 16 parameters to determine the configuration of 8 proximity sensors. The one to be inferred is as the one on a physical robot.

- *Training data.* The training data comes from the eight proximity sensors of an e-puck robot using the sensor configuration $\boldsymbol{q}$ as defined in Figure 3.4. According to the Enki library, $\boldsymbol{q} = (\boldsymbol{\theta}, \boldsymbol{d})$ is given as:

$$\begin{aligned} \boldsymbol{\theta} &= (-0.31, -0.79, -1.57, -2.48, 2.48, 1.57, 0.79, 0.31)\,\text{rad}, \\ \boldsymbol{d} &= (3.5, 3.5, 3.3, 3.4, 3.4, 3.3, 3.5, 3.5)\,\text{cm}. \end{aligned} \tag{3.3}$$

- *Model presentation.* The model is represented as the estimated sensor configuration $\hat{\boldsymbol{q}}$:

$$\hat{\boldsymbol{q}} = \left(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_8, \hat{d}_1, \hat{d}_2, \ldots, \hat{d}_8\right), \tag{3.4}$$

where $\hat{\theta}_i \in [-\pi, \pi]$ represents the inferred orientation of sensor i, and $\hat{d}_i \in (0, R]$ (R is the robot's radius) represents the inferred displacement of sensor i. Hence, there are 16 parameters to be estimated. The model data is obtained from an e-puck robot using configuration $\hat{\boldsymbol{q}}$. We assume all parameters vary continuously so that the eight proximity sensors can be distributed

at arbitrary locations on the robot's body. The remaining aspects of the robot are exactly the same as the ones used to obtain training data.

- *Discriminator presentation.* As discussed in Section 3.2.1, the discriminator is represented as an Elman neural network. In detail, the network has 5 hidden neurons and 8 inputs and 3 outputs. This gives a total of 88 parameters to be inferred. At each time step t, the normalised reading values from the robot's proximity sensors $(s_1, s_2, \ldots, s_8)$ are fed into the input layer. The network is then updated, and one of the outputs is used to label the input data. Moreover, the interaction comes from the other two outputs, which are used to determine the robot's wheel velocities (v_{left} and v_{right}) at time $t + 1$, where $v_{left}, v_{right} \in [-1, 1]$ represent the normalised ground contact velocities of the left and right wheel respectively (negative values correspond to the wheel rotating backwards). In each trial, the discriminator observes and controls the robot for 10 s. As the robot's sensors and actuators are updated 10 times per second, this results in 100 time steps.

- *Optimisation algorithms.* We use a standard $(\mu + \lambda)$ evolution strategy with self-adaptive mutation strengths for both the model and discriminator populations. Compared with (μ, λ) evolution strategy, $(\mu + \lambda)$ evolution strategy has the advantage to pass good candidates in the current generation to the next generation, which shares a similar idea of "hall of frame" [17] to tackle the *cycling* challenges in the coevolutionary process. We set $\mu = \lambda = 50$. Thus, for each population, it consists of 100 candidate solutions. All candidates are initialised at zero. The implementation details of this algorithm are as described in [180].

- *Coupling mechanism between the model and discriminator optimiser.* Given the synchronisation of the two sub-optimisation processes of the model and discriminator populations, each of the 100 discriminator candidates is evaluated once for each of the 100 model candidates, and 100 times with the training agent as well. Each time when the candidate gains a reward, it receives one point. Hence, in Equation 2.8a, $d_j(x \mid x \sim p^i_{\text{model}}) = 0$, if model data

samples are labelled as counterfeit; otherwise, $d_j(x \mid x \sim p^i_{\text{model}}) = 1$. Each model's fitness value sits in $\{0, 1, \ldots, 100\}$. Meanwhile, in Equation 2.8b, $d_j(x \mid x \sim p^i_{\text{model}}), d_j(x_n \mid x_n \sim p_{\text{data}})re = 0$ if the discriminator mistakenly labels input data samples; otherwise, $d_j(x \mid x \sim p^i_{\text{model}}), d_j(x_n \mid x_n \sim p_{\text{data}}) = 1$. This gives each discriminator's fitness value sits in $\{0, 1, \ldots, 200\}$. Given the total points received, a normalised fitness value is calculated for each candidate.

- *Termination criterion.* It is challenging to choose a suitable criterion to terminate the coevolutionary process as the performance of the model population and that of the discriminator population are correlated with each other. In this case study, we use a fixed time limit. The optimisation stopped after 1000 generations.

3.3.3 Simulation Setup

The robot operates in a bounded square environment with sides 50 cm as shown in Figure 3.5. The environment also contains 9 moveable objects. The object is simulated as a cylinder of diameter 3.7 cm, height 10 cm and mass 152 g with a ground friction coefficient of 0.58. Before each trial starts, the objects are arranged in a 3×3 grid. The distance between objects is just wide enough for the robot to pass through. The robot is placed randomly into the environment with a random orientation[2].

We have two simulations running in parallel: the *training simulation*, where an e-puck robot with the sensor configuration defined as the physical one operates in the environment and generates training (genuine) data samples; the *model simulation*, where the same robot but with the sensor configuration defined by a model candidate operates in the same environment and generates model (counterfeit) data samples. Each discriminator candidate is involved in both simulations. This procedure is illustrated in Algorithm 1.

[2]As the robot does not know its sensor configuration or its relative location to the objects, this scenario can be considered as a *chicken-and-egg* problem.

Algorithm 1 Turing Learning

procedure INFERRING SENSOR CONFIGURATION
    initialisation of $P_{\mathcal{M}}$ model candidates and $P_{\mathcal{D}}$ discriminator candidates
    while termination criterion not met **do**
        for all models $i \in \{1, \ldots, P_{\mathcal{M}}\}$ **do**
            obtain model parameters
            for all discriminators $j \in \{1, \ldots, P_{\mathcal{D}}\}$ **do**
                observe *Training Simulation*
                reward discriminator j for correct classifications
                observe *Model Simulation*
                reward model i for misleading discriminator j
                reward discriminator j for correct classifications
            end for
        end for
        update populations based on corresponding rewards
    end while
end procedure

procedure TRAINING SIMULATION
    create e-puck robot with sensor configuration $\boldsymbol{q}$
    for all time steps **do**
        obtain sensor reading values
        obtain and store outputs of discriminator
        execute corresponding actions
    end for
    reset simulation world
end procedure

procedure MODEL SIMULATION
    create e-puck robot with sensor configuration $\hat{\boldsymbol{q}}$
    for all time steps **do**
        obtain sensor reading values
        obtain and store outputs of discriminator
        execute corresponding actions
    end for
    reset simulation world
end procedure

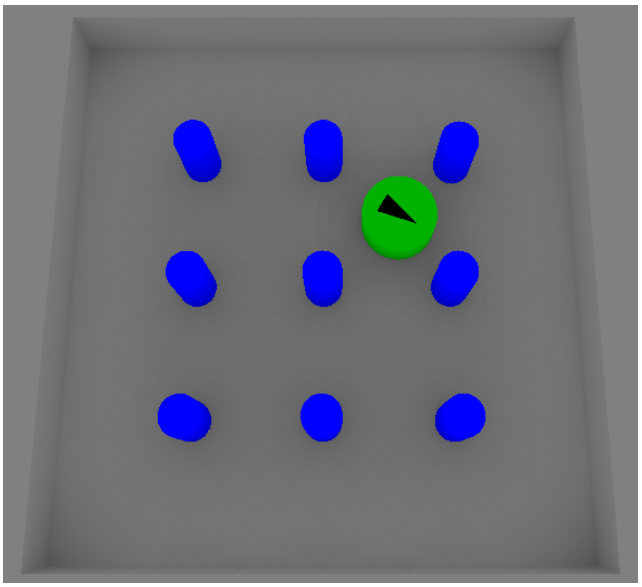


FIGURE 3.5: The e-puck robot (green) operates in a bounded square environment with nine moveable objects (blue). At the beginning of each trial, the objects are placed in a grid, and the robot is placed at random into the environment.

3.3.4 Simulation Results

A set of 20 evolutions was conducted. Each evolution lasted for 1000 generations. In each generation, the candidate with the highest fitness among its population is considered the best candidate. Therefore, for each evolution, the best candidate in the last generation is chosen as the best solution found in that evolution. We present our results as follows[3].

Inference Analysis

Figure 3.6 shows the evolutionary dynamics of the model parameters. Both of the orientations and displacements of all eight proximity sensors are inferred well after 500 generations. Interestingly, the orientation parameters corresponding to the two sensors placed at the back of the robot's body (i.e. θ_4 and θ_5) converged

[3]The choice of hyper-parameters, including the number of hidden neurons of the Elman network, the parameters regarding the $(\mu + \lambda)$ evolution strategy, the number of generations and the simulation settings, was not studied in this case study. They were chosen from some preliminary experiments and by experience as well. Noises were implemented as the ones in the Enki simulator. Different noise levels were not studied in this case study.

to the true values not closely as others did. This could be due to how the sensors are distributed on the robot. Given the sensor configuration in Figure 3.4, most of the sensors are placed at the front half of the circular body. In order to achieve an accurate classification, the discriminator may control the robot in a certain motion and pay more attention to the reading values from the sensors in the front. At the same time, the model parameters that are relevant to these sensors achieved better accuracy than the parameters of other sensors.

Coevolutionary Dynamics

We also studied the coevolutionary process of the *Turing Learning* framework. The fitness dynamics of the model and discriminator reveals how these two populations are evolved influence each other, as shown in Figure 3.7. The fitness of the discriminator population begins at 0.5, as all candidates are initialised identically, and they treat the training agent and models equally. Meanwhile, the fitness of the model population starts from 1.0, as all discriminators classify the candidates as the training agent. After that, there is an obvious increase in the average fitness of discriminators accompanied by a sharp decrease of the average fitness of models, which means that all models performed poorly in the early evolution. However, when the models are improved, the average fitness of discriminators increases slowly and then starts decreasing. Although there is a minor drop in the average fitness of models, it keeps increasing until it reaches a steady state. After around 700^{th} generation, the fitness of discriminators and of models remain stable until the process is terminated, which means the two populations finally find a way to balance each other in the course of coevolution. During the whole process, the population of the model did not manage to beat the population of the discriminator in terms of fitness values.

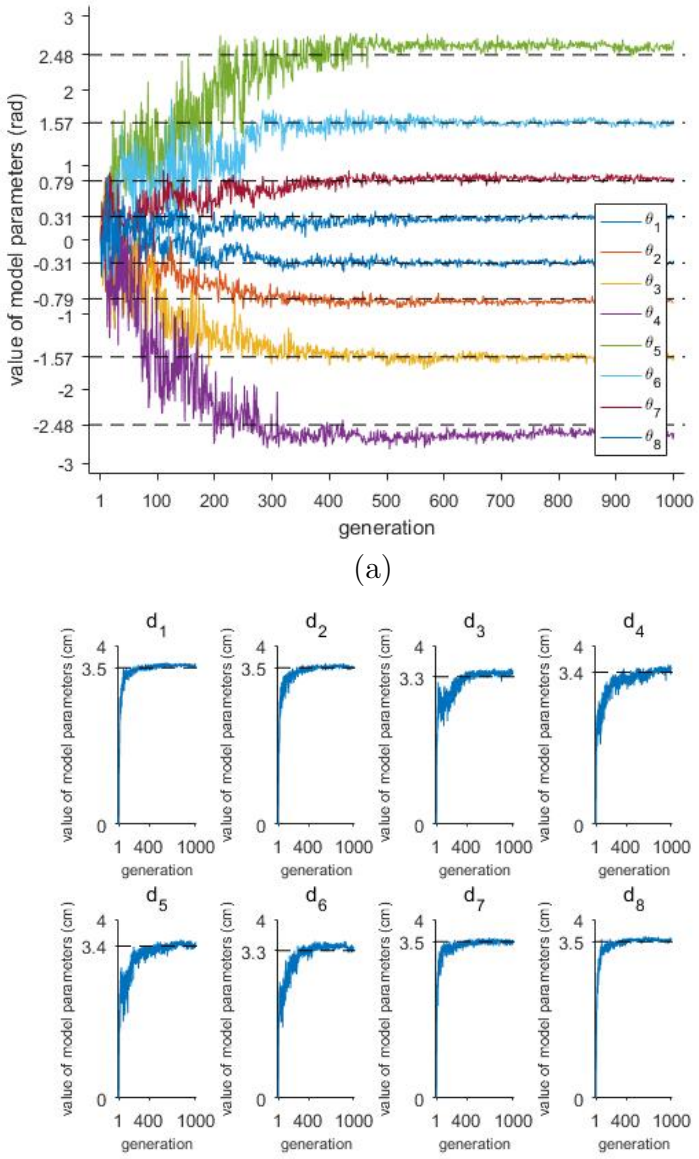


FIGURE 3.6: Evolutionary dynamics of the inferred sensor configuration: (a) orientations; (b) displacements. Curves represent the average values across 20 evolutions of the models with the highest fitness in each generation. Black dashed lines represent the ground truth.

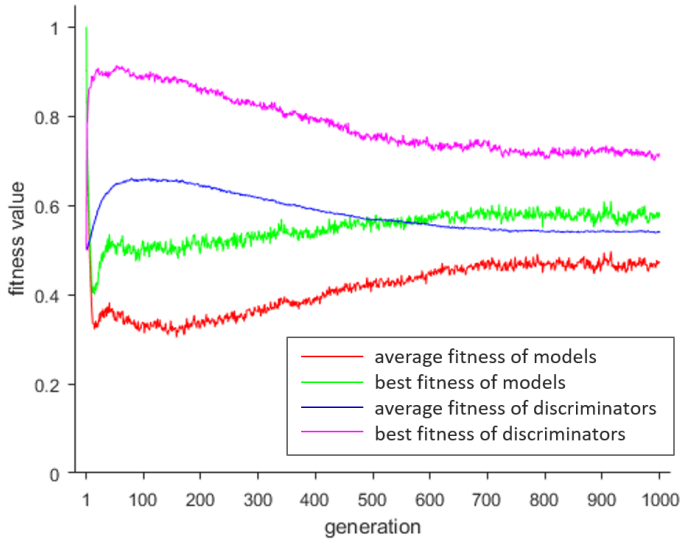


FIGURE 3.7: Fitness of the model and discriminator populations in *Turing Learning* framework. The evolution of the two populations remains balanced after around 700 generations. Curves represent the average values across 20 evolution runs.

Interaction Study

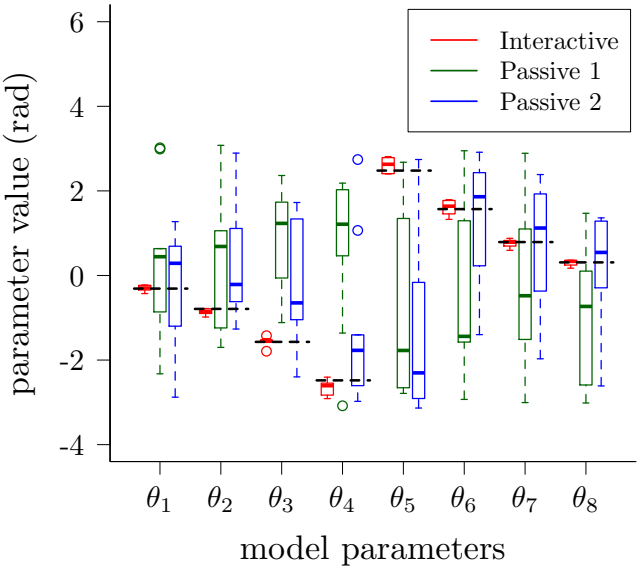
We refer to the above setup as the "Interactive" setup: the discriminator is able to control the movements of the robot while observing its sensor readings. To validate the advantage of the interactive approach, we implemented two passive setups where the discriminator observes the robot's sensor readings in a passive way; that is, besides the classification output, the other two outputs are not used to determine the movements of the robot. Instead, the robot moves in a random trajectory where its wheel velocities are chosen uniformly randomly at the beginning and change with a probability of 0.1 at every time step. In other words, the robot is expected to move in a random pattern every 10 time steps. In "Passive 1" setup, the discriminator network also has 8 inputs to receive sensor readings $(s_1, s_2, \ldots, s_8)$ from the 8 proximity sensors on the robot. In "Passive 2" setup, the discriminator network has additional two inputs to observe the wheel velocities v_{left} and v_{right}. The remaining aspects of the passive setups are implemented exactly as the "Interactive" setup. For each passive setup, a set of 20 evolutions were conducted. Figure 3.8 presents the distribution of the inferred model parameters with the highest fitness value in the 1000^{th} generation. The passive setups

failed on the inference of the orientation parameters, while the "Interactive" setup achieved that with good accuracy. The displacement parameters were inferred in all setups, but none of them managed to estimate these parameters accurately. One reason could be the noise applied to the sensors. As the dimension of the robot is relatively small to the sensing range of its proximity sensors, any small disturbance or noise could affect the detection accuracy in the distance, which will reflect on the shift of displacements.

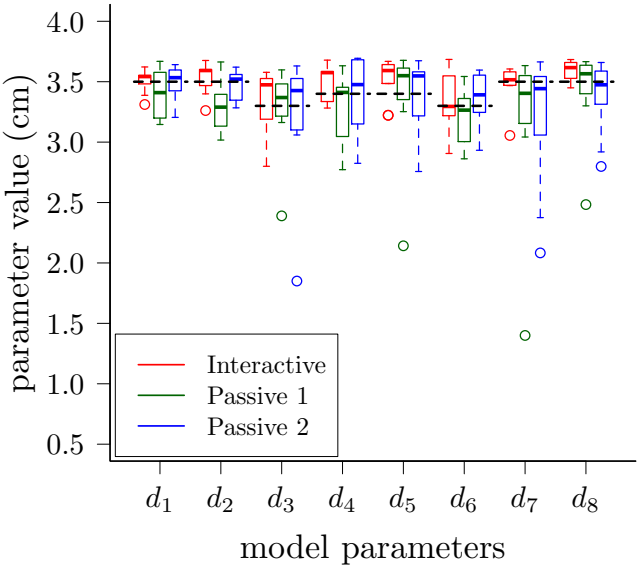
Behavioural Study

In order to understand how the robot moves under the control of a discriminator, we present a typical example of the relationship between the sensor readings and the corresponding wheel velocities in Figure 3.9. Considering the differential wheeled robot kinematics in Section 3.2.2, the robot rotates in the clockwise direction at the beginning (v_{left} is positive and v_{right} is negative). It perceives the environment with its sensors ($s_7, s_6, \ldots, s_2$ are activated successively). The robot moves forward when sensor s_1 and s_8 detects one object (v_{left} and v_{right} turn positive while s_1 and s_8 are activated after around 20 time steps). The discriminator leads the robot to push the object and keep reading s_1 and s_8, which confirms these two sensors are facing forward. The process is repeated once the robot has no object in its front. Similar behaviour of the robot can be found with other discriminators. As most sensors are facing forward or sides, the robot could easily miss the object at its back, even though the discriminator is able to drive the robot backwards. This also explains why the parameters corresponding to sensor s_4 and s_5 were not inferred as well as other parameters.

To validate if the *sensor-to-motor* correlation is essential for the discrimination task, we recorded the trajectory of the robot (wheel velocities at each time step) controlled by each best discriminator of the 20 evolution runs. We conducted 50 trials with the robot using the genuine sensor configuration and another 50 trials with the robot using the best model configuration of the corresponding evolution. In these 2000 "closed-loop" experiments, the discriminator made correct



(a)



(b)

FIGURE 3.8: Distribution of model parameters in interactive and passive setups: (a) orientations; (b) displacements. In the "Interactive" setup, the discriminator drives the robot moving in the environment and observes its sensor readings. In passive setups, the discriminator observes sensor reading and/or movements while the robot moving in a random manner (for more details, see text). Each box indicates the models with the highest fitness in the last generation of 20 runs. Black dashed lines represent the ground truth.

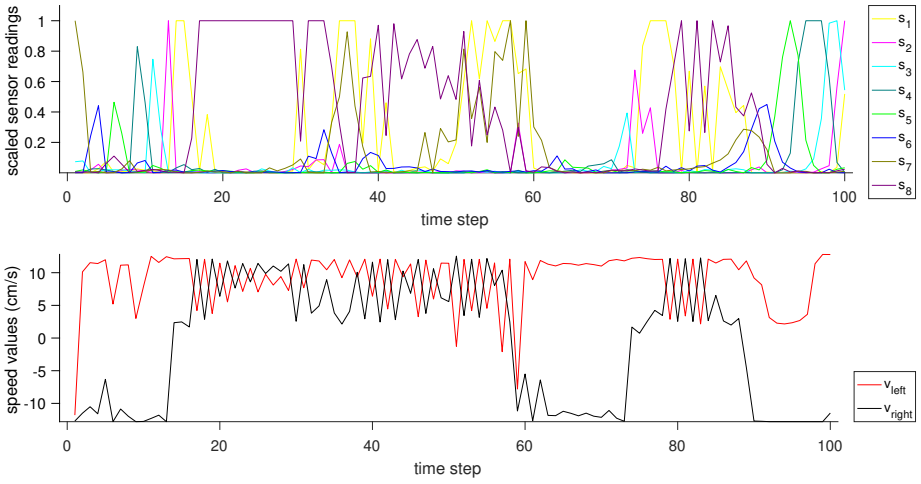


FIGURE 3.9: Example of how one discriminator controlled the movements of the robot. The discriminator processes the robot's sensor reading values ($s_1, s_2, \ldots, s_8$, shown at the top) and determines the velocities of the robot's left and right wheel (v_{left} and v_{right}, shown at the bottom). See text for more details.

judgements in 69.45% of the cases. We then repeated the 2000 trials, now ignoring the discriminator's control outputs but rather using the movements recorded earlier. In these 2000 "open-loop" experiments, the discriminator made correct judgements in 58.60% of the cases. There is a significant drop, though it is still better than random guessing (50%).

3.4 Summary

This chapter presents the coevolutionary *Turing Learning* framework where the discriminator agent and the mode agent are represented as populations of multiple candidate solutions. We demonstrated this framework with a case study of a robot agent inferring its own sensor configuration via a process of self-discovery. To implement that, the discriminator is able to control the movements of the robot while observing its sensor reading values. By comparing the "Interactive" approach with the passive way, we have shown that allowing the discriminator to influence the data sampling process could improve the model accuracy. The inference task was not simple, as the robot started randomly in the environment

and had no knowledge about either its sensor configuration or the relative locations of obstacles. At the same time, the robot's actuators and sensors were affected by noise. The discriminator thus needed to come up with an effective control sequence that drives the robot to stay close to an obstacle so that it could test the sensor configuration.

The case study has been done in simulation, but we believe it could serve as a template for modelling physical devices. If the training data comes from a physical system while the model data is simulated, there is a concern about the reality gap issue. Our framework has shown that its potential to reduce the gap and improve the accuracy of the simulator. In the future, we intend to implement this study on a physical e-puck robot, as well as build models of more complex systems.

A current limitation of this work is that the control process needs to be repeated for every classification the discriminator makes, which slows down the learning process of the algorithm. It could also be a problem for experiments with physical robots as it can lead to large energy consumption.

Chapter 4

Combining the Best of Active and Passive Learning

4.1 Introduction

A disadvantage of current *Turing Learning* algorithms is that they tend to rely on the availability of vasts amounts of training data [2]. This is a particular problem for applications in robotics. For example, in [6], the training data comprised the recorded trajectories of individual robots of a swarm. In general, this is a costly process, as the energy expended and time spent increase, usually linearly, with the amount of training data to be collected. For certain problems, methods have been proposed to augment the training data [181] or incorporate regularisation, though the approaches tend to be sensitive to the choice of hyper-parameters [182].

In prior work, the discriminators of *Turing Learning* algorithms have been either all active [5, 7], or all passive [2, 4, 6, 183]. A passive discriminator would merely observe a data sample and make a judgement. An active discriminator would, while observing, control the conditions under which the data sample is produced. The discriminator would thus act like an interrogator, akin to the setup in the Turing test [172]. This may be considered a step towards optimal experimental design [184, 185]. In the context of a mobile robot inferring its sensors' positions, it

was shown that *Turing Learning* with active discriminators outperformed *Turing Learning* with passive ones in terms of model accuracy [7]. However, the active learning approach is costly, as, for each judgement, a bespoke data sample has to be created.

In this chapter, we present a hybrid formulation of *Turing Learning*, in which the model population competes against two discriminator populations, one composed of active discriminators, the other composed of passive discriminators. We evaluate the system using a simulated scenario, where a fully autonomous robot, which has no knowledge of where it is located within its environment, infers a model for calibrating its laser-based distance sensor. It is shown that the hybrid formulation requires fewer data samples from the system under investigation than a purely active formulation. At the same time, it outperforms the purely passive formulation in model accuracy. Using the hybrid formulation of *Turing Learning* could enable robots to self-calibrate their sensors in situations where it is not possible or too costly to involve humans for manual calibration, for example, in highly confined or hazardous environments such as pipe networks, nuclear reactors, mines, deep sea or space.

This chapter is organised as follows. Section 4.2 describes the methodology, including the hybrid *Turing Learning* formulation (4.2.1) and the exclusiveness reward mechanism (4.2.2). Section 4.3 presents a case study to demonstrate the method, including an introduction of the simulation platform (Section 4.3.1). Section 4.3.2 lists the implementation options of hybrid formulation that are chosen for this scenario. The simulation results are presented in Section 4.3.3, including the analysis of the non-hybrid *Turing Learning* formulations, consideration of the practical cost, dynamics of the coevolutionary between three populations, the impact of the exclusiveness. Section 4.4 summarises the chapter.

4.2 Methodology

In this section, we will present a hybrid formulation of *Turing Learning* and a novel reward mechanism that takes into account the exclusiveness of the model's and discriminator's behaviour.

4.2.1 Hybrid *Turing Learning* Formulation

The *Turing Learning* formulation that is discussed here was proposed in [7] as a generalisation of a family of algorithms where models and discriminators are competitively optimised. The standard framework has been discussed in Section 3.2.1. It includes three agents: A training agent $\mathcal{T}$, a model $\mathcal{M}$ and a discriminator $\mathcal{D}$. $\mathcal{D}$ is able to observe or interact with $\mathcal{T}$ and $\mathcal{M}$. In this chapter, we define the discriminator as a hybrid agent $\mathcal{D}$ which contains two types of discriminators, an *interactive* discriminator $\mathcal{D}_i$, which acts as an interrogator and thus may influence the sampling process, and a *passive* discriminator $\mathcal{D}_p$, which acts as a passive observer. Hence, $\mathcal{D} = (\mathcal{D}_i, \mathcal{D}_p)$. Note that although $\mathcal{D}_i$ and $\mathcal{D}_p$ are referred to as single agents here, they are in general populations of agents.

The hybrid formulation of *Turing Learning* is illustrated in Figure 4.1. It allows information to flow in multiple ways among the involved agents. Agent $\mathcal{D}_p$ obtains purely observational information, as it lacks the ability to influence, in real-time, the agent generating the data ($\mathcal{M}$ or $\mathcal{T}$). By contrast, agent $\mathcal{D}_i$ can influence, in real-time, the agent generating the data, making closed-loop control a possibility.

The hybrid formulation of *Turing Learning* requires a new coupling mechanism that determines how the rewards for the agents are to be obtained. As in the standard formulation of *Turing Learning*, the model and discriminator agents are competitively optimised, and individual discriminators receive rewards that reflect the quality of their judgements (e.g., the percentage of correct judgements). However, a particular model can now be tested against discriminators of both

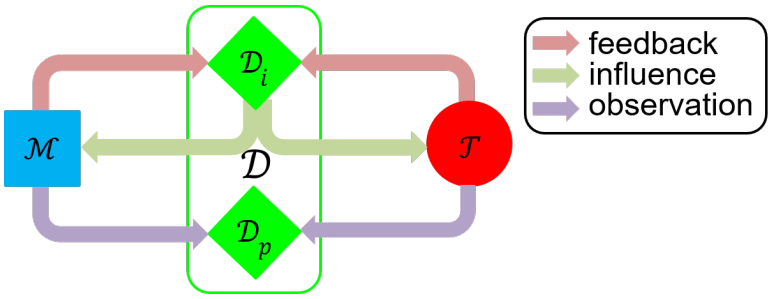


FIGURE 4.1: Hybrid formulation of *Turing Learning*. A model $\mathcal{M}$ can compete with two types of discriminators, an *interactive* discriminator, $\mathcal{D}_i$, and a *passive* discriminator, $\mathcal{D}_p$. The interactive discriminator can influence, in real-time, the agent (i.e., the model, $\mathcal{M}$, or training agent, $\mathcal{T}$) while observing it. The passive discriminator only observes the agent. The hybrid approach promises to combine the advantages of active learning (i.e., higher model accuracy) and passive learning (i.e., fewer data samples, as they can be reused for different discriminators).

types. This opens up many possibilities. For example, where the model is tested against both types of discriminators, a mechanism to combine individual rewards becomes necessary, such as a weighted sum reflecting the percentage of times the model misled discriminators of either type. Another approach would be to use a protocol that identifies the type of discriminator to be used in the evaluation. For example, as the training progresses, model candidates could be exposed to discriminators of either type, though not necessarily simultaneously.

4.2.2 Exclusiveness Reward Mechanism

In previous studies on *Turing Learning* in [4] and the case study in the previous chapter, an agent is rewarded one point each time it wins a "game". That is, a model receives a point for misleading a discriminator and zero otherwise. A discriminator receives a point for making a correct judgement (where the data is either genuine or not) and zero point otherwise. Each game is hence equally important. Such a pure subjective measurement could lead to the *Red Queen Effect* during the coevolutionary process. To avoid this, we borrowed the idea of the Elo rating system [186], invented by Arpad Elo, which is widely used to estimate the strength of players in chess. The points that a player could obtain depend on the relative difference between the opponents. For example, a low-rated

player could win more points by defeating a high-rated player than the other way around.

We propose a novel reward mechanism, which is not directly based on the opponents' strength but takes into account the *exclusiveness* of the game's outcome. To illustrate this, consider a setting with two populations: one of the models and one of the discriminators. If an agent loses in a game, its corresponding reward is 0 points. Otherwise, the corresponding reward points, $r \in [0, 1]$, are

$$r = \alpha + (1 - \alpha) \cdot \beta, \tag{4.1}$$

where $\alpha \in [0, 1]$ represents a reference reward, which the agent receives irrespective of the outcome of other games, and $\beta \in [0, 1]$ represents an additional reward, based on how exclusive the outcome of the game is compared with all other games involving the same opponent.

For example, let us assume there are n models and k discriminators and that every model plays against every discriminator once. The results of these games are summarised in Table 4.1, where "0" and "1" indicate a discriminator agent that labels data samples of a model agent as counterfeit or genuine, respectively. For a particular discriminator d_i, model m_i is the only one who misleads d_i, consequently its exclusive points $\beta_{m_i} = \frac{\text{number of failed models}}{n-1} = 1$, resulting in the maximum reward of $r = 1$ points for that game. For d_j, there are other models, such as m_j, who also misleads d_j, therefore $\beta_{m_i} = \frac{\text{number of failed models}}{n-1} < 1$, resulting in a lower reward of $r < 1$ points for that game. The rewards of discriminators follow the same calculation, though also taking into account the games against the training agents.

It is hoped that the refined mechanism for calculating rewards provides a more accurate reflection of an agent's importance relative to the other agents within its own population, thereby boosting the optimisation process. This is highly related to the motivation of *novelty search* [152]. Both approaches aim to encourage the candidates that have outstanding performance in the current generation by giving them extra rewards, which leads to the increasing diversity of the current

TABLE 4.1: Illustration of the exclusive rewards mechanism. Each of n model candidates $(m_1, \ldots, m_n)$ plays one game against each of k discriminator candidates $(d_1, \ldots, d_k)$, with "0" indicating the discriminator won, and "1" indicating the model won. The reward of any agent winning a game is determined by how exclusive the outcome was compared to all games involving the same opponent. In the case presented here, m_i obtains a higher reward for winning against d_i than for winning against d_j.

	m_1	$\ldots$	m_i	$\ldots$	m_j	$\ldots$	m_n
d_1	1	$\ldots$	0	$\ldots$	1	$\ldots$	1
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
d_i	0	$\ldots$	1	$\ldots$	0	$\ldots$	0
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
d_j	0	$\ldots$	1	$\ldots$	1	$\ldots$	0
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
d_k	0	$\ldots$	0	$\ldots$	0	$\ldots$	1

population. The reference reward α determines the extent to which exclusiveness is taken into account. The process used in [4–6] corresponds to the nonexclusive case with $\alpha = 1$.

4.3 Case Study

With the reality gap problem introduced in Section 3.3.1, as simulators are unable to capture all features of the real world, in many cases, a controller that gave rise to a certain behaviour on a simulated robot gives rise to a different behaviour when transferred onto a physical robot. The latter is a critical problem, especially in evolutionary robotics, where simulations play a significant role in speeding up the optimisation process [187]. Through a process of trial and error, a human could carefully tune a simulator to faithfully reproduce a system's behaviour within certain constraints. However, in general, this process is costly and would have to be repeated manually every time the robot or environment had changed.

In the following, we present a case study to investigate how an autonomous mobile robot can use the hybrid formulation of *Turing Learning* to build an accurate model of a simple rangefinder sensor it is equipped with. The study is conducted

in simulation. Hence, the exact ground-truth parameters can be used to evaluate the final derived model parameters, enabling us to quantify the performance of the overall approach. As in Chapter 3, we distinguish between two types of simulations:

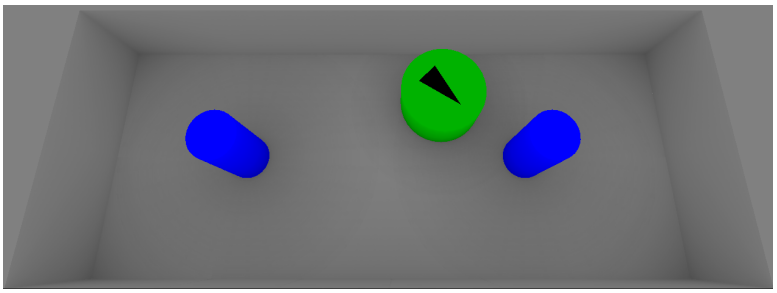
- *Training data* simulations obtain data directly from the system under investigation. They produce genuine data samples. If employing *Turing Learning* on a real robot, the training data would be collected from the sensor while the robot is moving. As this is costly, the fewer training data are needed, the better.
- *Model data* simulations obtain data using a candidate model. They produce counterfeit data samples. Even if employing *Turing Learning* on a real robot, this data could be produced using a computer simulation. Hence, model data are considered less costly to produce than training data.

For our purposes, we continue to use the Enki platform, for which it was possible to port simplistic behaviours from simulation to reality without modification [188]. In the following sections, we will describe the robot platform used in this case study and present the implementation details of the hybrid *Turing Learning* formulation.

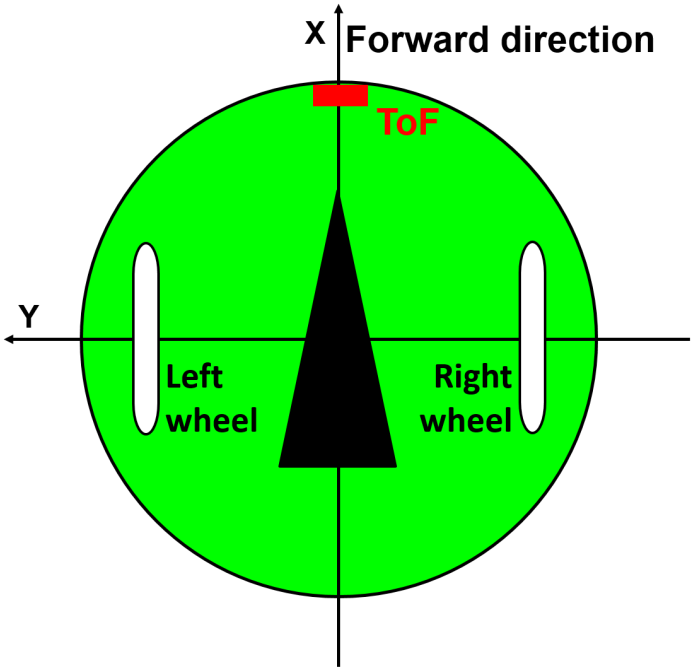
4.3.1 Robot Simulation Platform

The environment is a horizontal plane with a rectangular boundary of dimensions 50 cm and 20 cm. It contains a robot and two unmovable obstacles, as shown in Figure 4.2(a).

The obstacle is modelled as a cylinder of diameter 3.7 cm, height 10 cm. We consider the second version of the e-puck robot, e-puck2, as our robot. It shares the same model as e-puck in Enki. Recall that the robot has a cylindrical body of radius 3.7 cm, height 4.7 cm and weight 152 g and two wheels, which are arranged as a differential drive, at a distance of 5.1 cm from each other. The velocity of each wheel, v_{left} and v_{right}, can be set within [-12.8, 12.8] cm/s relative to the ground.



(a)



(b)

FIGURE 4.2: Case study involving a mobile robot that is tasked to infer a model of its distance sensor. (a) Simulation environment with the robot (green cylinder; arc indicates forward direction) and two static obstacles (blue cylinders). The robot starts from a random position and orientation and obtains sensor reading values while moving around. It is not aware of its position and has no access to the ground-truth distance corresponding to the sensor reading value. (b) Illustration of the robot, an e-puck2. The robot has two wheels and perceives the environment with a time-of-flight (ToF) distance sensor heading forwards.

As introduced in 3.2.2, random noise is applied every control cycle by multiplying each velocity value with a random number chosen uniformly in the range (0.95, 1.05). The control cycle is 0.1 s.

The real robot has a rangefinder, more specifically, a laser-based time-of-flight (ToF) distance sensor located on the edge of its cylindrical body, as illustrated in Figure 4.2(b). The sensor returns an estimate of the distance (in cm) to the

closest object in the robot's front. The estimate is given as an integer value. In our simulation study, the sensor provides an integer value too, which is calculated as

$$d^* = \text{round}(k^* \cdot d \cdot \delta + b^*), \tag{4.2}$$

where $d \in \mathbb{Z}$ is the true distance (in cm) from the sensor to the first object ahead of the robot, k^* and b^*, respectively, are the slope and offset parameters of a linear function, and δ is a multiplicative noise term, which is uniformly chosen from the range $(0.95, 1.05)$.

4.3.2 Hybrid *Turing Learning* Implementation

The *Turing Learning* implementation for this case study is as follows:

- *Training data.* To generate a data sample, the robot is placed at a uniformly random position and orientation within the environment. It is observed for a fixed duration, T. Every control cycle, one sensor reading, d^*, is obtained from (4.2) with the ground-truth slope and offset parameters, $k^* = 1.167$ and $b^* = -1.789$, respectively. Although a linear approximation is used, the distribution of the training data involves some nonlinearity, which is introduced by the noise and limited resolution of the simulated sensor, as shown in Figure 4.3. The movements that the robot performs depend on the type of discriminator (see *Discriminator representation*).

- *Model representation.* To generate a data sample, the robot is placed at a uniformly random position and orientation within the environment. It is observed for a fixed duration, T. Every control cycle, one sensor reading, d^*, is obtained from (4.2) with the model parameters for the slope and offset, $\hat{k}$ and $\hat{b}$, respectively, as well as $\delta = 1$. Both parameters are initialised at zero and can freely evolve in $\mathbb{R}$ during the optimisation (subject to the limited precision of computers). The movements that the robot performs depend on the type of discriminator (see *Discriminator representation*).

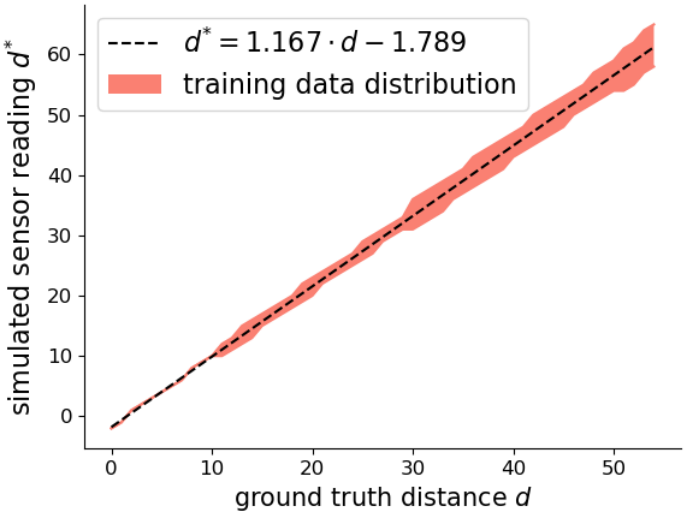


FIGURE 4.3: Training data distribution for ground-truth distances from 1 cm to 54 cm (the maximum point-to-point distance in the environment, ignoring obstacles). The dashed line represents the linear transformation of the ground-truth distance, ignoring the limited sensor resolution and noise. Note that *Turing Learning* has no access to the ground-truth data for the system under investigation.

- *Discriminator representation.* The hybrid discriminator consists of two types: the interactive one and the passive one. The structure of both is represented as an Elman neural network with one input, five hidden neurons and one classification output. Hidden and output neurons use the logistic function $sig(\cdot) = \frac{1}{1+e^{-(\cdot)}}$ as the activation function, which results in a number within [0, 1]. For classification outputs of 0.5 or higher, the data sample is categorised as "genuine"; otherwise, it is categorised as "counterfeit".

  The interactive discriminator has two additional outputs which determine the robot's wheel velocities (v_{left} and v_{right}). This is done by using the aforementioned logistic function, followed by linear scaling. Hence, the network of the passive discriminator has 41 parameters, whereas the network of the active discriminator has 53 parameters. All parameters are initialised at zero and can freely evolve in $\mathbb{R}$ during the optimisation (subject to the limited precision of computers).

  For the passive discriminators, a data sample of duration $T = 5\,s$ is produced. During this time, the robot follows a pre-designed passive manner. For the

active discriminators, a data sample of duration $T = 5\,s$ is produced. The outputs of the discriminator are used to control the movements of the robot in real-time.

- *Optimisation algorithms.* A coevolutionary algorithm is used to solve the optimisation problem. The algorithm is comprised of three populations: one of the models, one of the interactive discriminators and one of the passive discriminators. Each population is evolved by the $(\mu + \lambda)$ evolution strategy with self-adaptive mutation strengths. We set $\mu = \lambda = 50$ leading to 100 candidates in each population ($P_{\mathcal{M}} = P_{\mathcal{D}} = 100$). These three populations are optimised independently except when fitness values are calculated. More details of the evolutionary algorithm can be found in [6, 180].

- *Coupling mechanism.* The model population is evaluated in every generation of the optimisation process. In the first n_p generations, the models are evaluated against the passive discriminators. In the following n_i generations, the models are evaluated against the interactive discriminators.[1] The process is then repeated. By default, $n_p = n_i = 1$.

  We first consider a generation where the passive discriminators are evaluated. To reduce the amount of training data, which in reality would be costly to obtain, only a single *training data* simulation is performed. The resulting data sample is used for every passive discriminator. Moreover, $\mu + \lambda = 100$ *model data* simulations are performed, that is, one per model. The resulting data samples are tested on every passive discriminator. The fitness of the passive discriminator is a weighted percentage of its correct judgements (equal weight is assigned to the sets of genuine and counterfeit data samples).

  In generations where the active discriminators are evaluated, one *training data* simulation is performed per active discriminator, that is, $\mu + \lambda = 100$ *training data* simulations in total. Moreover, $\mu + \lambda = 100$ *model data* simulations are performed, that is, one per model. The resulting data samples are

[1] We hypothesise that it is simpler to evolve passive discriminators than active ones. By starting the evaluation with the passive discriminators, the models are more likely to obtain meaningful rewards early on, which then guide their development.

tested on every active discriminator. The fitness of the active discriminator is a weighted percentage of its correct judgements (equal weight is assigned to the sets of genuine and counterfeit data samples).

The final reward values are adjusted using (4.1), as described in Section 3.2.1. By default, $\alpha = 1$. Thus, the exclusiveness reward is discarded. The above process can be illustrated in Algorithm 2.

Algorithm 2 Hybrid Turing Learning

procedure CALIBRATING DISTANCE SENSOR
  initialisation of $P_{\mathcal{M}}$ candidate models and $P_{\mathcal{D}}$ candidate discriminators
  while termination criterion not met **do**
    create e-puck2 robot with calibrated sensor
    perform defined passive manner and record data samples for one trial
    reset simulation world
    repeat
      for all models $i \in \{1, \ldots, P_{\mathcal{M}}\}$ **do**
        obtain model parameters
        for all passive discriminators $j \in \{1, \ldots, P_{\mathcal{D}}\}$ **do**
          observe recorded *Training Simulation*
          reward passive discriminator j for correct classifications
          observe *Model Simulation*
          reward model i for misleading passive discriminator j
          reward passive discriminator j for correct classifications
        end for
      end for
      update populations based on corresponding rewards
    until n_p generations

    repeat
      for all models $i \in \{1, \ldots, P_{\mathcal{M}}\}$ **do**
        obtain model parameters
        for all interactive discriminators $j \in \{1, \ldots, P_{\mathcal{D}}\}$ **do**
          observe *Training Simulation*
          reward interactive discriminator j for correct classifications
          observe *Model Simulation*
          reward model i for misleading interactive discriminator j
          reward interactive discriminator j for correct classifications
        end for
      end for
      update populations based on corresponding rewards
    until n_i generations
  end while
end procedure

```
36: procedure Training Simulation
37:     create e-puck2 robot with calibrated sensor
38:     for all time steps do
39:         obtain sensor reading values
40:         obtain and store outputs of discriminator
41:         if interaction is available then
42:             execute corresponding actions
43:         else
44:             perform defined passive manner
45:         end if
46:     end for
47:     reset simulation world
48: end procedure
49:
50: procedure Model Simulation
51:     create e-puck2 robot with uncalibrated sensor
52:     for all time steps do
53:         obtain calibrated sensor readings with model parameters
54:         obtain and store outputs of discriminator
55:         if interaction is available then
56:             execute corresponding actions
57:         else
58:             perform defined passive manner
59:         end if
60:     end for
61:     reset simulation world
62: end procedure
```

- *Termination criterion.* The optimisation process terminates after a fixed number of generations. We consider up to 50 generations, though the exact limit depends on the particular study.

4.3.3 Simulation Results

This section presents the simulation results of the case study[2]. Throughout this section, we use the Mann-Whitney test with a significance level of 0.05 and present the resulting p values for each text.

[2]Similar to the case study in Chapter 3, the choice of hyperparameters, including the number of hidden neurons of the Elman network, the parameters regarding the $(\mu+\lambda)$ evolution strategy and the simulation settings, was not studied in this case study. They were chosen from some preliminary experiments and by experience as well. Noises were implemented as the ones in the Enki simulator. Different noise levels were not studied in this case study.

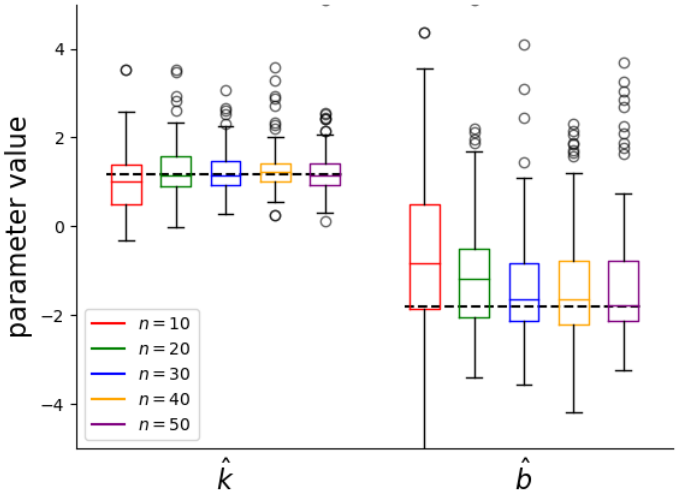


FIGURE 4.4: Parameters ($\hat{k}$ and $\hat{b}$) that interactive formulations of *Turing Learning* inferred. Shown are the models of the highest subjective fitness of the final generation. Each run lasts n generations. Each box comprises 100 runs. Black dashed lines indicate the ground truth.

Analysis of Interactive Formulation

Before evaluating the hybrid formulation, $\mathcal{D}_i\&\mathcal{D}_p$, we characterise the performance of the two non-hybrid formulations. In the passive formulation, $\mathcal{D}_p$, only passive discriminators are considered. In the interactive formulation, $\mathcal{D}_i$, only interactive discriminators are considered.

For the active simulations, 100 *Turing Learning* runs were performed for $n = 10, 20, \ldots, 50$ generations. Figure 4.4 presents the results. Shown are the inferred parameters of the model with the highest subjective fitness in the final generation of each run. The interactive formulation achieves high performance on both parameters for $n = 30$. The slope parameter required only 20 generations to be inferred ($p = 0.442, 0.292, 0.427$, paired with $n = 30, 40, 50$ respectively). After that, increasing the number of generations didn't bring significant improvement to the model accuracy, which means the interactive formulation is able to solve the inference task with a small training data set.

We also investigated how frequently the interaction is required during the simulation. We enabled the control of $\mathcal{D}_i$ for every δ time steps. For each of

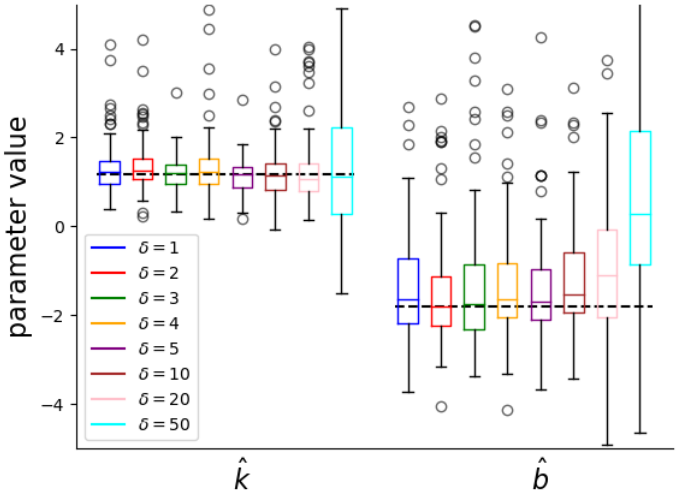


FIGURE 4.5: Parameters ($\hat{k}$ and $\hat{b}$) that interactive formulations of *Turing Learning* inferred when the interaction enabled every δ time steps of 50 steps in total. Shown are the models of the highest subjective fitness of the final generation. Each run lasts 30 generations. Each box comprises 100 runs. Black dashed lines indicate the ground truth.

$\delta = 1, 2, 3, 4, 5, 10, 20, 50, 100$ *Turing Learning* runs were performed. Each run lasted for $n = 30$ generations. When $\delta = 1$, the formulation is implemented exactly as the above; when $\delta = 50$, no interaction is implemented and the robot stands still through the whole simulation. Figure 4.5 illustrates the distribution of the inferred model parameters with the highest fitness in the final generation. Interestingly, the inference was not affected much if the interaction is implemented less frequently, given that $p = 0.06, 0.02$ regarding $\hat{k}$ and $p = 0.08, 0.02$ regarding $\hat{b}$ in the paired tests between $\delta = 1$ and $\delta = 10, 20$ respectively.

We statistically analysed the control behaviour of the inferred discriminator with the highest fitness in the final generation. Table 4.2 presents the percentage of time when the robot was controlled to move "linearly" (i.e. move forward or backward with $v_{left} = v_{right}$) throughout 100 simulations for each δ. The active discriminator adapts to the increase of δ and tends to control the robot moving in a straight line most of the time, which provides some guidance to design the behaviour in the passive formulation.

[3]Note that it is not 100% when $\delta = 50$. This is due to the control outputs of the active discriminator at the last time step, which was not implemented as the simulation terminated.

TABLE 4.2: Percentage of time when the robot moved forward or backward in interactive formulation with the interaction implemented every δ time steps of 50 steps in total. Each number represents the statistical result across 100 simulation[3].

$\boldsymbol{\delta}$	1	2	3	4	5	10	20	50
percentage(%)	43	43	53	55	52	66	65	98

To validate the advantage of interactive setup, we implemented three passive setups where the control outputs of the active discriminator are ignored. Instead, the robot follows determined movements. As shown in Table 4.2, when the interactive discriminator interacts with the robot less frequently, it acts more like a passive discriminator and prefers to control the robot moving linearly. Therefore, we considered two linear movements and a random movement of the robot in the passive setups. In "Passive 1" setup, the robot moves in a random trajectory where its wheel velocities are chosen uniformly randomly at the beginning and change with a probability of 0.1 at every time step. In other words, the robot is expected to move in a random pattern every 10 time steps. In "Passive 2" setup, the robot moves forward with a velocity of 10 cm/s. This corresponds to 1 cm per iteration, though due to noise on the actuation and potential collisions with objects, the distance covered is likely to vary between iterations. In "Passive 3" setup, the robot moves backwards with a velocity of 10 cm/s. For each setup, we performed 100 runs, and each run lasted 30 generations. Figure 4.6 shows the inferred model parameters in each setup. The interactive approach is essential to achieve a good inference on the slope parameter; "Passive 3" setup failed to infer both parameters. Surprisingly, by implementing passive manners, the inference of the offset was slightly improved, especially in "Passive 2" setup (given $p = 0.19$).

In order to understand *sensor-to-motor* correlation in the interactive setup, Figure 4.7 presents an example of the discriminator controlling the robot's wheel velocities based on the corresponding sensor readings. Considering the differential wheeled robot kinematics in Section 3.2.2, the robot approximately rotated in the anti-clockwise direction when its sensor perceived the wall or an obstacle far away; otherwise, it relatively moved forward if the wall or an obstacle was close. Similar behaviours can also be found with other discriminators.

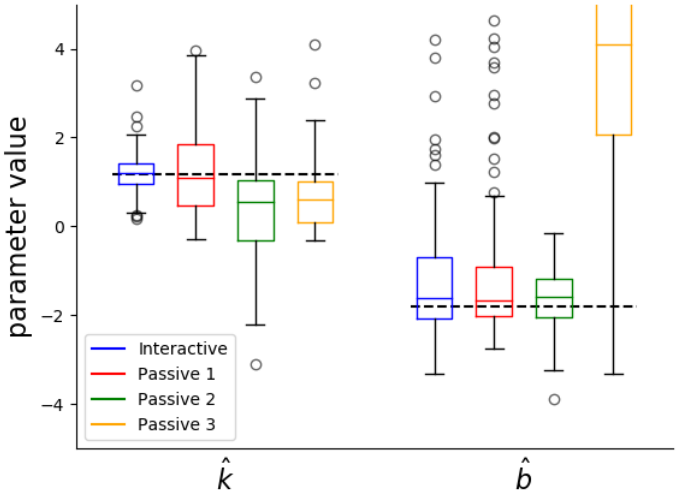


FIGURE 4.6: Parameters ($\hat{k}$ and $\hat{b}$) inferred with interactive and passive setups. Shown are the models of the highest subjective fitness of the final generation. In the "Interactive" setup, the active discriminator drives the robot moving in the environment and observes its sensor readings. In passive setups, the discriminator observes sensor reading while the robot moving in determined manners (for more details, see text). Each run lasts 30 generations. Each box comprises 100 runs. Black dashed lines indicate the ground truth.

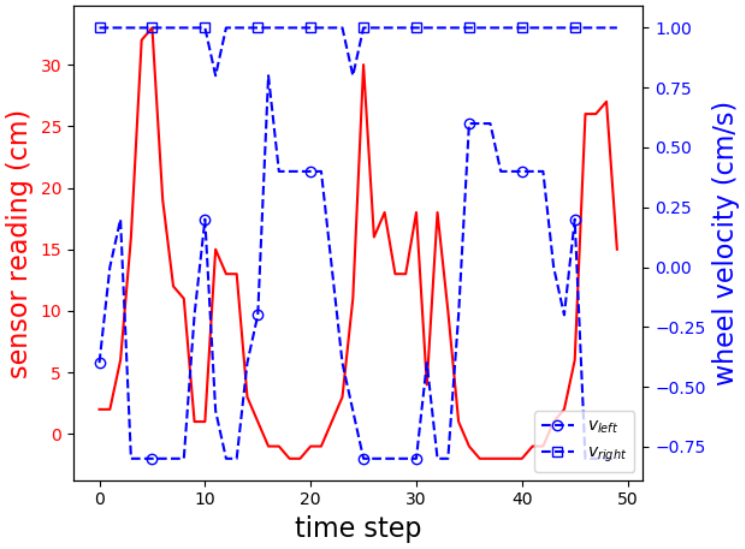


FIGURE 4.7: Example of how one discriminator controlled the robot in the interactive setup. The discriminator processes the robot's sensor reading values and determines the velocities of the robot's left and right wheel (v_{left} and v_{right}). See text for more details.

Analysis of Passive Formulation

Recall that the robot is expected to move in a straight line more often if the discriminator controls it less frequently. We design the robot's manner in passive formulation exactly as the one in the "Passive 2" setup above, that is, the robot moves forward with a velocity of 10 cm/s for 5 s, resulting in 1 cm per time step (ignoring obstacles and noise). Similarly, 100 *Turing Learning* runs were performed for $n = 10, 20, \ldots, 50$ generations. Figure 4.4 presents the results. Shown are the inferred parameters of the model with the highest fitness in the final generation of each evolution run. The passive formulation performed poorly regarding the slope parameter, in particular, when taking into account that it is applied as a factor rather than an addend. However, it achieved a fairly good level of performance regarding the offset parameter, even in 20-generation runs. Consider a model that uses a slope parameter of zero. As a result, all sensor reading values it produces match its offset parameter. It is then not too difficult for the model offset parameter to get further optimised. This makes it hard for a passive discriminator to detect that the data samples are not genuine. The robot could have simply moved against an obstacle for the duration of the trial, which would cause the ground-truth distance to be 0, making the sensor reading match the genuine offset parameter. This is where the *disengagement* could have occurred. Both the model population and discriminator population stopped being optimised as the evolution was suspended. One should note that increasing the number of generations didn't improve the model accuracy. Instead, the inference of the offset parameters tended to be less accurate, which could indicate a possible overfitting problem of the passive formulation when the training data set increases.

Analysis of Practical Cost

The practical costs of employing each formulation on a real robot are also to be considered. They are likely dominated by the number of hours the robot is busy collecting the *training data*, whereas the costs of the *model data* simulations are negligible. For all three formulations of *Turing Learning*, including the hybrid

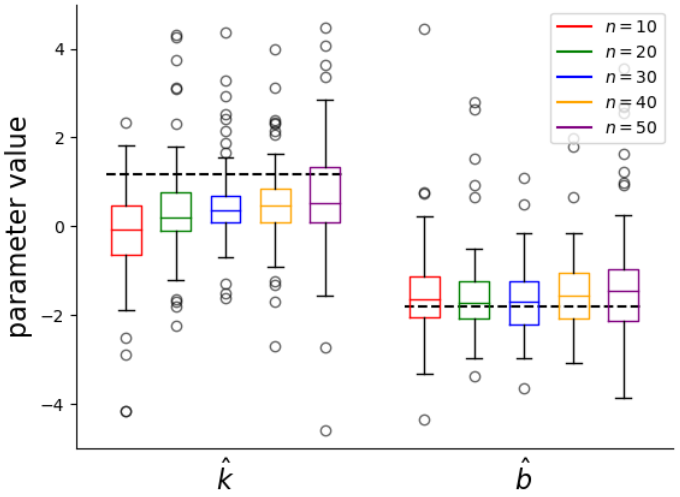


FIGURE 4.8: Parameters ($\hat{k}$ and $\hat{b}$) that passive formulations of *Turing Learning* inferred. Shown are the models of the highest subjective fitness of the final generation. Each run lasts n generations. Each box comprises 100 runs. Black dashed lines indicate the ground truth.

TABLE 4.3: Hours of *training data* required by the interactive ($\mathcal{D}_i$), hybrid ($\mathcal{D}_i$&$\mathcal{D}_p$), and passive ($\mathcal{D}_p$) formulations of *Turing Learning*. When inferring the sensor model for a real robot, this number is a good indication of the true costs. Note that in the present study, the *training data* is obtained from simulations, lasting 5 s for both the interactive and passive discriminators.

setup	$\mathcal{D}_i$	$\mathcal{D}_i$&$\mathcal{D}_p$	$\mathcal{D}_p$
cost	13.89·n	6.95·n	0.0014·n

one, the practical costs of performing a single run of n generations are shown in Table 4.3. As can be seen, the interactive formulation is by far the most costly formulation. By using the hybrid formulation, almost half of the costs can be saved. The costs of the passive setup are remarkably low.

Analysis of Hybrid Formulation

We now consider the situation that only a limited budget is available, preventing us from conducting more than 10 generations of the costly interactive formulation. We evaluate a hybrid formulation of 20 generations. In half of the generations, the passive discriminators are trained, whereas, in the other half, the interactive discriminators are trained. Figure 4.9 shows the results. We have also included the

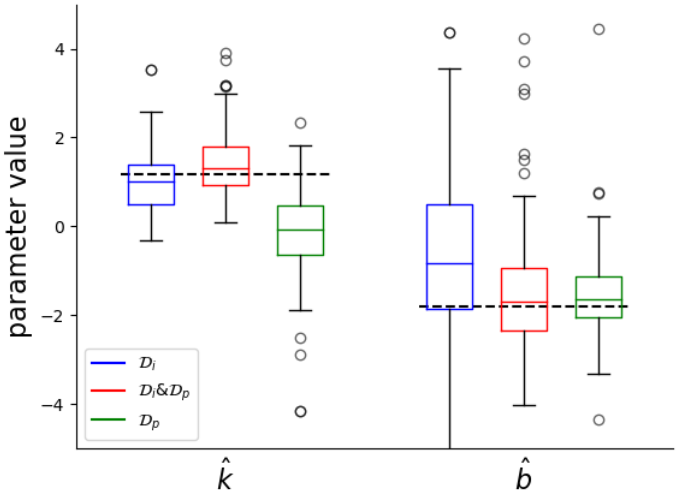


FIGURE 4.9: Comparison between the hybrid formulation using 20 generations ($\mathcal{D}_i$&$\mathcal{D}_p$) and its two components in isolation: the interactive formulation using 10 generations ($\mathcal{D}_i$) and the passive formulation using 10 generations ($\mathcal{D}_p$). Each box represents the distribution of estimates $\hat{k}$ and $\hat{b}$ of the model with the highest fitness in the final generation and contains 100 runs.

results of performing only 10 generations of either non-hybrid formulation. This makes it possible to quantify what each part of the hybrid formation adds beyond and what the other part could achieve on its own. It can be seen that the non-hybrid formulations are unable to accurately infer both parameters. The hybrid formulation outperforms the interactive formulation with 10 generations regarding $\hat{k}$ ($p < .0001$) and $\hat{b}$ ($p = 0.0002$). It outperforms the passive formulation with 10 generations regarding k ($p < 0.0001$). When compared with an interactive formulation using 50 generations [see Figure 4.4], the hybrid formulation achieves a similar performance regarding $\hat{k}$ ($p = 0.046$) and $\hat{b}$ ($p = 0.231$) while saving almost 80% of the cost.

In general, the passive discriminators help infer the offset parameter well, for the reasons discussed earlier. However, they do not help to infer the slope parameter. The performance stagnates as the number of generations increases, posing the risk of over-fitting. The interactive discriminator helps infer the slope parameter, as it can control the robot to not get stuck but rather observe a wide range of sensor readings. But it is too costly to be used exclusively. By using the

hybrid discriminator, the advantages of either type of discriminator are combined. The hybrid formulation gives additional degrees of freedom to adjust the learning strategy to the budget at hand.

Coevolutionary Dynamics and the Impact of Exclusiveness

Figure 4.10 shows the dynamics of fitness values of the three populations in the hybrid formulation of *Turing Learning*. The average fitness of the passive discriminators starts from 0.5, then increases slowly until the end of 20 generations. The fitness of the best candidate increases rapidly and reaches a high value (about 0.9) within 5 generations. This means that some candidates can easily distinguish the model data samples by observing the effect of $\hat{b}$ when operating the passive manner. The fitness value of the interactive discriminator shares a similar curve with the passive discriminator, except that best fitness has a lower value. This is due to the fact that without a direct insight into the effect of b, the classification task becomes more difficult. The fitness value of models shows that models are able to adapt to the improvement of two types of discriminators. After 5 generations, the curve has obvious fluctuations, and the fitness drops every generation when the interactive discriminators get involved. But in general, the fitness of models increases till the end of evolution. This phenomenon shows that when the models adapt to the passive discriminators and have a good estimation on b, the interactive discriminators can filtrate the model population by the effect of estimation of k. The models, in turn, are optimising themselves to improve their inference.

It is known that *Turing Learning* (and similar co-evolutionary) algorithms can suffer from unstable behaviour. For example, *generative adversarial networks* [2] can experience *model collapse* when the discriminator dominates the "game", resulting in vanishing gradients for the generative models. Possible solutions are to weaken [189, 190] or regularise the discriminator [191]. In our case, the discriminators tend to have higher fitness than the models. This can lead to the situation that they dominate the competition.

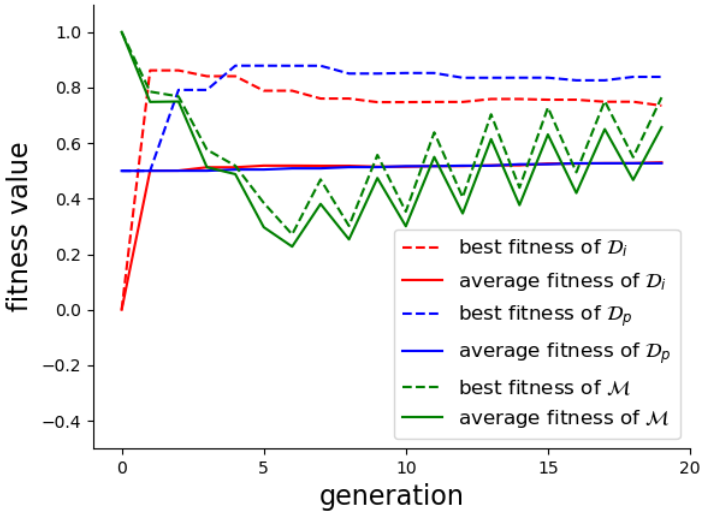


FIGURE 4.10: Normalised fitness of discriminators and models in the hybrid formulation of *Turing Learning* over 20 generations. Each curve represents the average value across 100 runs.

In the following, we examine whether the performance of the hybrid *Turing Learning* formulation can be improved by introducing the novel reward mechanism that takes the exclusiveness of the games' outcomes into account. We evaluate a setting with only 10 generations, 5 for each type of discriminator, where the novel reward mechanism is only applied to the reward calculation of the interactive discriminators. As can be seen in Figure 4.11, without exclusiveness taken into account, neither the slope parameter nor the offset parameter can be accurately inferred unless the number of generations is doubled. By applying 20% exclusiveness on $\mathcal{D}_i$ ($\alpha_i = 0.8$) and 50% exclusiveness on $\mathcal{D}_p$ ($\alpha_p = 0.5$), the slope parameter can be more accurately inferred (p-values 0.002) compared with the nonexclusive case. The exclusiveness mechanism enables the hybrid formulation to solve the inference task with reasonable good accuracy even with only $n = 10$ generations. Adding exclusiveness to the evaluation process increases the difficulty of corresponding candidates to obtaining rewards, especially where the diversity of solutions is limited. This leads to high selection pressure, promoting solutions that possess unique skills, helping them to defeat opponents in ways that may otherwise be overlooked.

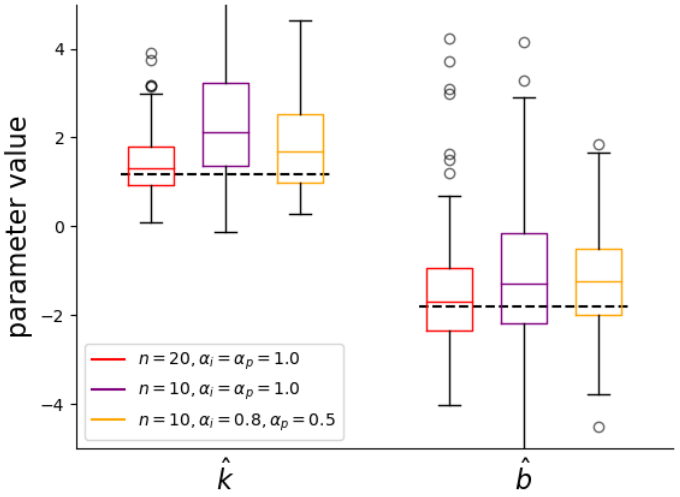


FIGURE 4.11: Benefits of a reward mechanism that takes exclusiveness into account. If $\alpha = 1$, all games that are won are equally important. If $\alpha < 1$, games against opponents that are rarely defeated count more than other games. Based on n generations, 100 simulation runs per box.

4.4 Summary

This chapter presented a hybrid formulation of *Turing Learning* that allows models and both interactive and passive discriminators to be optimised at the same time. To evaluate the hybrid formulation, we presented a case study where a simulated robot calibrates its own distance sensor without having access to ground-truth data. The hybrid formulation combines the advantages of either approach. It was shown to outperform the passive approach in terms of model accuracy while outperforming the interactive approach in terms of the amount of training data that is needed. Minimising the latter is critical when the tasks involve real robots that shall learn autonomously.

The setup is particularly challenging, as the robot does not know its position within the environment. In fact, it started from a purely random position and orientation. It could be already touching a wall or be far away from it. By learning a closed-loop controller, as a byproduct of the discrimination task, the hybrid formulation succeeded in identifying the parameters of the sensing model with good accuracy.

We also proposed a novel reward mechanism that takes into account how frequently the opponent was defeated by the competing solutions. The case study presented in this paper showed that using this mechanism for the reward calculation of interactive discriminators improved the model accuracy in situations where only limited training data was available.

The result demonstrates that the purely interactive formulation outperforms other approaches if the available budget is not constrained. The hybrid formulation maintains a good performance where the budget is of concern.

In the future, we intend to run the hybrid formulation of *Turing Learning* on a computer connected to a real e-puck2 robot to investigate whether the robot can infer its own sensing model without any intervention by humans.

Chapter 5

Inferring Swarm Behaviours from Their Effects

5.1 Introduction

In previous chapters, we implemented the *Turing Learning* framework in order to the sensor configuration and the sensor calibration model of a single robot agent. In both cases, the robot operated in an environment with obstacles. The inference was accomplished on the basis of the robot's sensor readings only and didn't require any knowledge about the environment, e.g. locations of obstacles, or any other information about the robot itself, e.g. the relative location between the robot and obstacles. Following a similar idea, in this chapter, we investigate how *Turing Learning* could benefit the inference of swarm behaviours when the observability is limited.

As introduced in Section 2.3, swarm behaviours are emergent behaviours of simple individuals following local behavioural rules [120]. In the field of swarm robotics, robots are used to mimic the swarm behaviours that are observed in nature, such as foraging of ant colonies, which, in turn, helps to provide a deeper understanding of such behaviours. Usually, behaviours of swarm robotic systems are generated by designing controllers for individual robots to capture the features that are found

in biological swarms. Considering the simplicity of the robot agent, the controller usually has a relatively simple structure and does not require centralised mechanisms. The memory-less controller proposed in [188] pushes the simplistic nature to the extreme. It is built with a *sense-act* architecture, implemented as a lookup table between the binary sensing and the corresponding motion. The *Turing Learning* algorithm has been demonstrated to infer such controllers for aggregation and object clustering in previous studies [4]. In both cases, the inference relies on the full observability of the robot agent (i.e. full knowledge of the robot's velocities). However, when the observation is not fully available, the inferred controller is hard to capture the features of the desired behaviour.

In this chapter, we consider an extreme situation where the robots in a swarm are not available to observe at all. Instead, we investigate if the local behavioural rule can still be inferred by examining its effects on the surrounding environment. The motivation for this case study is mainly curiosity-driven, but it still has practical meanings. Considering the design problem in Section 2.3.3, the advantage of evolutionary swarm robotics is that it searches for the optimal solutions by evaluating the emergence of the desired global behaviour. For some swarm behaviours, their emergence reflects on the effects on the embedded environment. By observing these effects, the evolutionary process tends to find the optimal individual controllers from the global level.

Specifically, we focus on two swarm behaviours: object clustering and shepherding. In general, the clustering behaviour of robots results in objects being transferred into one cluster and, in shepherding, robots would herd the "sheep" towards a desired goal position. In both cases, *Turing Learning* is only allowed to observe the changes of objects or sheep but has no access to the robot or other information about the environment, including the goal of shepherding.

This chapter is organised as follows. Section 5.2 describes the methodology, including a formal definition of the problem (5.2.1), an introduction about the simulation platform (5.2.2), and implementation details of the *Turing Learning* algorithm (5.2.3). Section 5.3 presents two case studies to demonstrate our method.

Section 5.3.1 illustrates how the object clustering behaviour could be learned by *Turing Learning* through observing the trajectory of a single object, and Section 5.3.2 implements *Turing Learning* to infer shepherding behaviour through observing the trajectory of a single sheep. The inferred controllers in both cases are analysed in terms of emergent performance and scalability. At the same time, the performance is compared with that of the controller designed by the method introduced in the literature. In addition, Section 5.3.1 investigates how the training data affects the scalability of the clustering controller inferred through our method, and Section 5.3.2 shows the ability of the inferred shepherding controller to herd the sheep group tracking a dynamical goal. Section 5.4 summarises the chapter.

5.2 Methodology

In this section, we formally define the problem considered in this chapter and introduce the simulation platform used in this scenario, including the sensing and control capabilities of the robot agents. We then explain the implementation options of the *Turing Learning* algorithm for solving this problem. The general algorithm framework applied in this chapter is similar to the ones in previous chapters. We will highlight the differences and briefly describe what has been introduced before.

5.2.1 Problem Formulation

Different swarm behaviours in nature have been studied throughout the literature, which naturally inspired researchers to imitate these behaviours in swarm robotic systems. In most cases, the principles of such systems are designed through the decomposition of desired behaviours into local rules for individual robots, during which the understanding of nature is the fundamental factor. The *Turing Learning* algorithm has been applied to learn the behavioural rules of robotic agents for

performing aggregation and object clustering [4]. In both cases, individual robots follow a simple reactive controller, which is inferred through the observation of their trajectory data.

In our formulation, we consider an environment where neither the human designer nor the learning algorithm has access to observe the desired behaviours at the global level or the local level. In other words, individual robots in a swarm are "invisible" all the time during the designing process[1]. Instead, we apply the *Turing Learning* algorithm to find the local rules of individuals through observing the effects on the environment. This formulation has practical meaning. In some cases, the behaviour of individuals in a swarm can be complicated and difficult to observe directly, while the effects can be accessible to notice and monitor. We demonstrate this formulation in an unbounded simulation environment studying two swarm behaviours, object clustering and shepherding, which cause effects on the surroundings. When clustering objects, a swarm of robots would transfer the objects between different positions. On the other hand, when herding sheep (or sheep-like robots), a swarm of shepherd robots would cage the sheep in a group and march them towards the desired goal.

5.2.2 Simulation Platform

Similar to Chapter 3 and Chapter 4, we use the open-source Enki library and the built-in e-puck model as our simulation platform and implement some specific capabilities for the robot. Recall that in Enki, the e-puck robot is modelled as a cylinder of radius 3.7 cm, height 4.7 cm and weight 152 g, with an inter-wheel distance of 5.1 cm. Each wheel's velocity, v_{left} and v_{right}, can be set within [-12.8, 12.8] cm/s relative to the ground. As introduced in 3.2.2, random noise is applied every control cycle by multiplying each velocity value with a random number chosen uniformly in the range (0.95, 1.05). The control cycle is 0.1 s. The physics is updated 10 times per control cycle.

[1] As there is no direct knowledge about the behaviour under investigation, the scenario can be considered as a *black-box* system identification problem.

5.2.2.1 Sensor

Each robot is equipped with a line-of-sight sensor I at its front. The sensor is able to return the type of items it detects (e.g. other robots, objects, the empty space or the boundary of the environment) but does not provide any more information, for example, the distance to a perceived item or the number of items in front of it. In general, the possible sensor state can be represented as $I \in \{0, 1, \ldots, n-1\}$. Each state indicates one particular item. In our simulations, the sensor is simulated as a ray cast from the robot's front and returns the type of the very first item it intersects, or nothing if there isn't any. The sensor has an unlimited range.

5.2.2.2 Controller

The robot is embedded with a memoryless controller with an identical structure as the one introduced in [24, 188]. The controller uses a simple *sense-act* logic, i.e. at each time step t, each robot's motion is solely determined by the current sensor state I^t. Given the discrete sensor state above, the controller can thus be considered as a mapping from each state to a pair of wheel velocities. We use $v_{l,I}, v_{r,I} \in [-1, 1]$ to represent the normalised angular velocities of the left and right wheel, respectively, corresponding to the sensor state I, where negative velocities indicate the wheel rotating backwards. Therefore, the controller can be represented as a $2n$-valued vector:

$$\boldsymbol{v} = (v_{l,0}, v_{r,0}, v_{l,1}, v_{r,1}, \ldots, v_{l,n-1}, v_{r,n-1}), \quad \boldsymbol{v} \in [-1, 1]^n. \tag{5.1}$$

5.2.3 *Turing Learning* Implementation

In Chapter 3, we have introduced that *Turing Learning* is a family of algorithms where models and discriminators are optimised in a competitive game setting. Recall that the framework can be applied to a wide range of problems by choosing different implementation options. In the following, we will present the options that are designed for the problem considered in this chapter.

- *Training data.* The training data comes from the observation of the environmental change caused by a swarm of robots, that is, change of the object's absolute position in the study of object clustering or change of the sheep's absolute position in the study of shepherding. Training data is obtained when robots operate a determined controller, which is referred to as the *training controller* and represented as the vector $\boldsymbol{v}$. The controller is designed through the method in the literature and can provably generate the desired behaviour on the global level (i.e. by the swarm) [24, 147]. On the other hand, the model data has the same form as the training data but is obtained while robots operate a *model controller*, which is defined by a candidate model.

- *Model presentation.* The model is considered as a substitution of the training controller vector $\boldsymbol{v}$ (given in Equation 5.1). We use $\boldsymbol{v}'$ to represent the model controller vector, which has the same form as $\boldsymbol{v}$:

$$\boldsymbol{v}' = (v'_{l,0}, v'_{r,0}, v'_{l,1}, v'_{r,1}, \ldots, v'_{l,n-1}, v'_{r,n-1}), \quad \boldsymbol{v}' \in [-1, 1]^n. \tag{5.2}$$

  Similarly, each pair of $(v'_{l,I}, v'_{r,I})$ represents the normalised angular velocities of the left and right wheel, respectively, with respect to the sensor state I. The inference task is thus to find a parameter setting so that robots embedded with controller $\boldsymbol{v}'$ are able to generate the desired behaviour. In other words, a total of $2n$ parameters are to be inferred. Note that $\boldsymbol{v}'$ is not necessarily required to be identical with $\boldsymbol{v}$, as it is possible that different local rules wound end with similar emergent behaviour on the global level. We will then illustrate and evaluate the behaviour with the learned model controller and compare the performance of the model controller to that of the training controller.

- *Discriminator presentation.* The discriminator is implemented as an Elman neural network with a similar structure as the passive one introduced in Chapter 4. In this scenario, the discriminator observes the environmental changes and determines whether the individual robots are operating the

training controller or a model controller. The network has 5 hidden neurons and one classification output. For each classification it makes, the network receives a data sample from either the training data or the model data. The remaining features are implemented exactly as in the previous chapter. We assume all parameters of the network can vary in $\mathbb{R}$. For each trial, the observation lasts for a certain period of time (i.e. t seconds). Given that the robot's control cycle is updated 10 times per second, this results in $10t$ time steps.

- *Optimisation algorithm.* In order to compare the performance of the controller learned with *Turing Learning* and that of the controller found in [24, 147], We use the same optimisation method, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [192], for both the model and discriminator population. CMA-ES is a stochastic method to optimise real-valued vectors associated with self-adapted variance of each variable and all covariances between variables[2]. The same constraint as the one used in [24] is applied to the model optimiser by processing each parameter of the candidate solution obtained from CMA-ES with a sigmoid-based function (as shown in Equation 5.3) to ensure parameters in vector $\boldsymbol{v}'$ vary in $[-1, 1]$.

$$sig(x) = \frac{1 - e^{-(x)}}{1 + e^{-(x)}} \quad \forall x \in \mathbb{R} \tag{5.3}$$

The discriminator optimiser is applied with the standard form of CMA-ES. Furthermore, there are three external parameters required for CMA-ES: the population size p, the initial candidate vector $\boldsymbol{m}^{(\mathbf{0})}$ and the initial step size $\boldsymbol{\delta}^{(\mathbf{0})}$. For both of the model and discriminator optimiser, we set $\boldsymbol{m}^{(\mathbf{0})}$ to be a zero vector and $\boldsymbol{\delta}^{(\mathbf{0})}$ to 0.72. As shown in Monte Carlo simulations, these external parameters give an initial model population approximately uniformly distributed in $[-1, 1]^n$ with the above constraint [24]. The population size is given by the suggested setting in [192, 193]:

$$p \approx \lfloor 4 + 3 \ln d \rfloor, \tag{5.4}$$

[2]For more details about CMA-ES, we refer to [192, 193].

where d is the number of variables in decision vector. For example, an n-state sensor leads to a model population size $p_{\mathcal{M}} \approx \lfloor 4 + 3 \ln 2n \rfloor$.

- *Coupling mechanism between the model and discriminator optimiser.* The coupling mechanism is implemented exactly as the one in the case studies of Chapter 3 and Chapter 4. Each discriminator candidate is evaluated once with each model candidate and $p_{\mathcal{M}}$ times with the training controller. It is rewarded one point for each time it correctly distinguishes between the training data and model data. Each model candidate is rewarded one point every time it misleads a discriminator candidate to judge its data as training data. Given the total points they gain, the normalised fitness values for each discriminator and model candidate are calculated by Equation 2.8b and Equation 2.8a separately.

- *Termination criterion.* Similar to previous studies, the optimisation process is terminated after a fixed number of generations.

5.3 Case Studies

In the following, we validate the method described above on learning two swarm behaviours: the object clustering and the shepherding, which are simulated as the studies in [24] and [147] respectively[3].

5.3.1 Object Clustering

5.3.1.1 Simulation Setup

Similar to the study in [24], we consider an unbounded environment with 2 robots and 5 objects. The object is simulated as a cylinder of radius 5 cm, height 10 cm

[3]Similar to the case studies in Chapter 3 and Chapter 4, the choice of hyperparameters, including the number of hidden neurons of the Elman network, was not studied in this case study. They were chosen from some preliminary experiments and by experience as well. Noises were implemented as the ones in the Enki simulator. Different noise levels were not studied in this case study.

and weight 35 g, with a ground friction coefficient of 0.58. In this case, the robot's sensor has 3 states (i.e. $n = 3$ in Equation 5.1 and 5.2, which gives the model population size $p_{\mathcal{M}} = 10$): $I = 0$ when it detects nothing, $I = 1$ when it detects an object and $I = 2$ when it detects another robot. The robots and objects are initialised uniformly in a virtual square of sides 111.8 cm. That gives the average area per object is 2500 cm^2. In addition, each robot's initial orientation is chosen uniformly randomly in $[-\pi, \pi]$.

Similar to previous studies in Chapter 3 and Chapter 4, we have two simulations running in parallel: the *training simulation*, where the training data samples are collected, and the *model simulation*, where the model data samples are collected. Both of these two simulations are conducted with the same setup as described above.

- *Training Simulation.* The individual robot operates the training controller $\boldsymbol{v}$, as shown in Figure 5.1. The controller was found through the method introduced in [24], where its performance was evaluated by measurement $U(T)$[4]:

$$U(T) = \sum_{t=1}^{T} tu(t), \tag{5.5}$$

where T is the total number of the time steps of the simulation and

$$u(t) = \frac{1}{4r_O^2} \sum_{i=1}^{m} ||\bar{\boldsymbol{x}}(t) - \boldsymbol{x}_i(t)||^2, \tag{5.6}$$

where r_O and m are the radius and the number of the objects respectively, $\bar{\boldsymbol{x}}(t)$ is the centroid position of object group at time t and $\boldsymbol{x}_i(t)$ is the position of the object i at time t. Thus, $u(t)$ takes account of how widely the objects are dispersed. $U(T)$ benefits the solution that collects objects fast by introducing the time index t. In other words, lower values of $U(T)$ indicate better and faster solutions of the controller. We re-implemented the study

[4]In order to replicate the previous study, $U(T)$ is adapted directly from [24] and is used to evaluate the performance of the controller learned with *Turing Learning* as well.

TABLE 5.1: The training controller $\boldsymbol{v}$ operated by individual robots during the training simulation process in the study of object clustering.

$v_{l,0}$	$v_{l,1}$	$v_{l,2}$
0.9832	-0.7797	0.7866
$v_{r,0}$	$v_{r,1}$	$v_{r,2}$
0.5591	0.7524	0.0501

in [24] and found the best controller as shown in Table 5.1[5]. With such a controller, a swarm of robots performs the object clustering behaviour for 100 s (i.e. 1000 time steps), during which we randomly choose a single object and record its absolute position $(x(t), y(t))$ at each time step as training data samples. That gives 2 input neurons of the discriminator network, leading to a total of 46 parameters to be inferred. The discriminator population size is determined as $p_{\mathcal{D}} = 20$. Figure 5.2 shows the snapshots of a training simulation lasting for 100 s. The cluster of objects emerged after 65 s.

- *Model Simulation.* Within the same simulation environment as in the training simulation, robots and objects are initialised with random configurations[6] and individual robots operate the model controller $\boldsymbol{v}'$ defined by a model candidate (shown in Figure 5.3). Same as the training simulation, model simulation lasts for 100 s, and the model data samples consist of the absolute positions of a single object which is randomly picked.

[5]We noticed that it is not possible to fully replicate the study in [24] due to the version of Enki library, the computing platform on which simulations are conducted and some other uncontrollable factors. We then re-implemented the study with the sources we have at hand. Although the controller we found is different from the one reported in [24], it is still able to achieve the desired behaviour with similar performance. We assume that such difference would not affect the learning outcome of *Turing Learning* as the robot's motion is not considered in our formulation. To maintain the consistency of our study, we use the above controller to generate training data samples.

[6]We noticed that if the model simulation and the training simulation have the same initial configuration of objects and robots, the discriminator might simply compare the position of the object at each time step. To avoid the discriminator taking advantage of the difference between single data samples, the initial configuration is set differently for the model simulation and the training simulation.

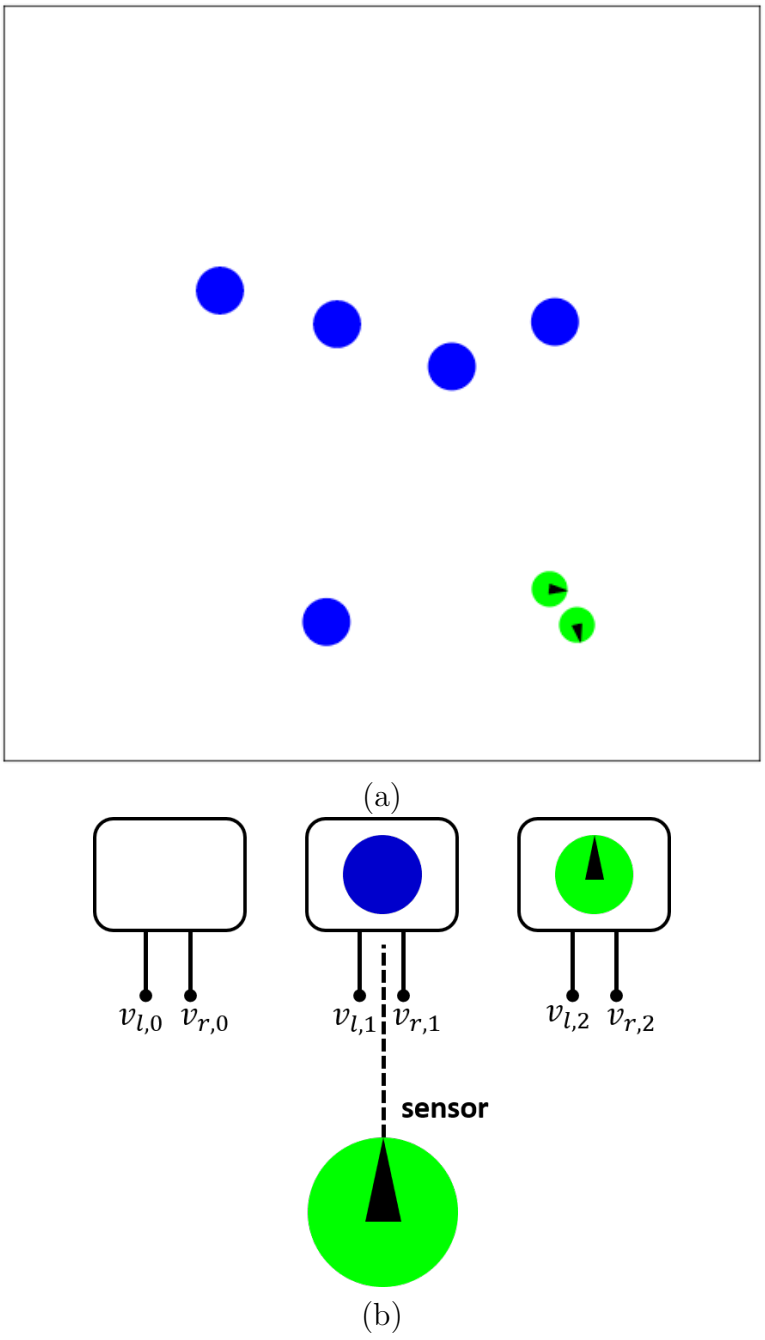


FIGURE 5.1: (a) The snapshot presents the *training simulation* environment of the object clustering study, where 5 blue objects and 2 robots (shown as disks with arrows) are randomly placed. Green colour indicates that the robots are executing the training controller. (b) The schematic illustrates the robot's training controller, which maps what the sensor detects in its front onto a pair of wheel velocities.

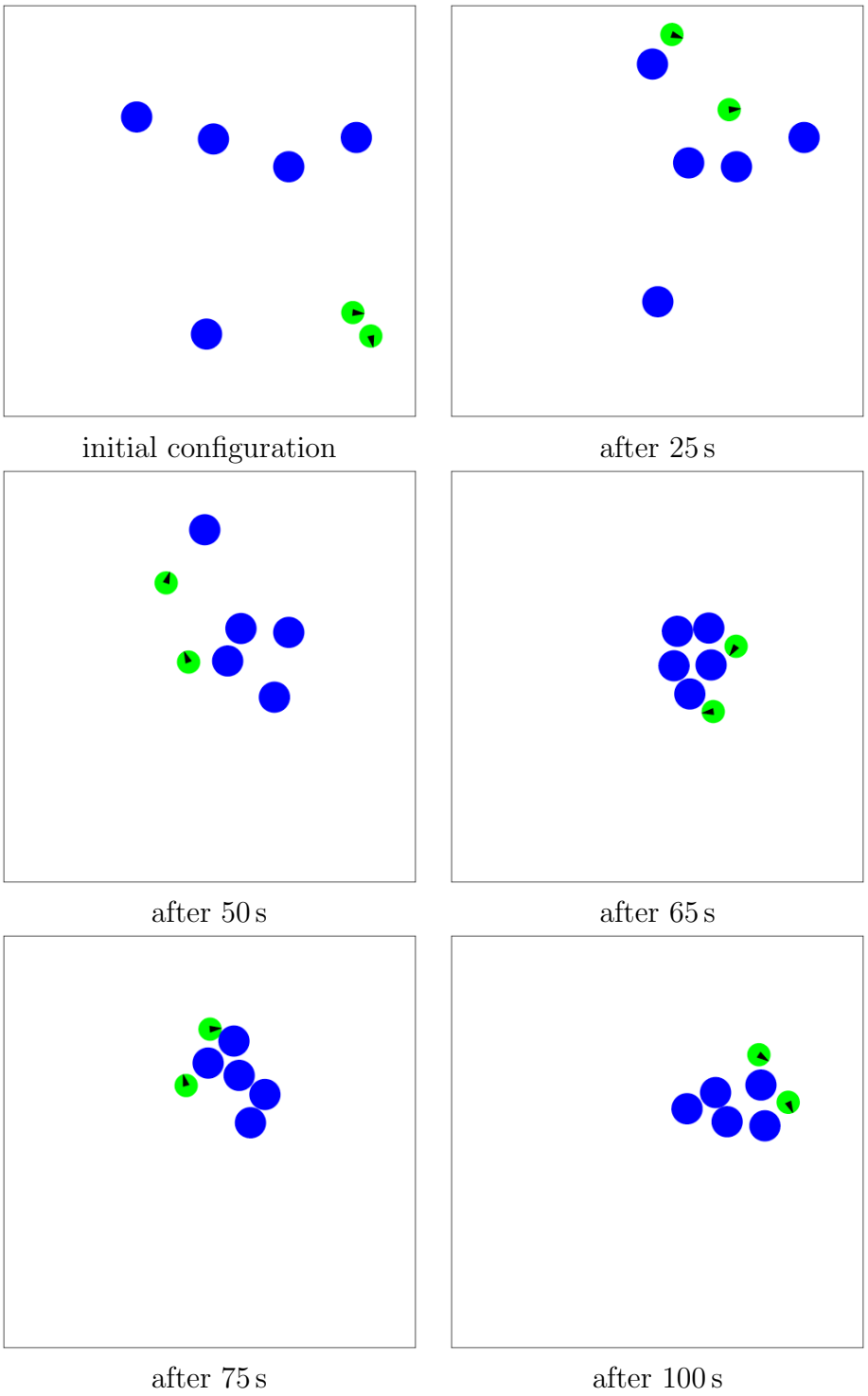


FIGURE 5.2: Emergent behaviour of object clustering with robots (green disks) operating the training controller. The absolute positions of a single object (blue disk) are recorded as training data samples and evaluated by the discriminator in *Turing Learning*.

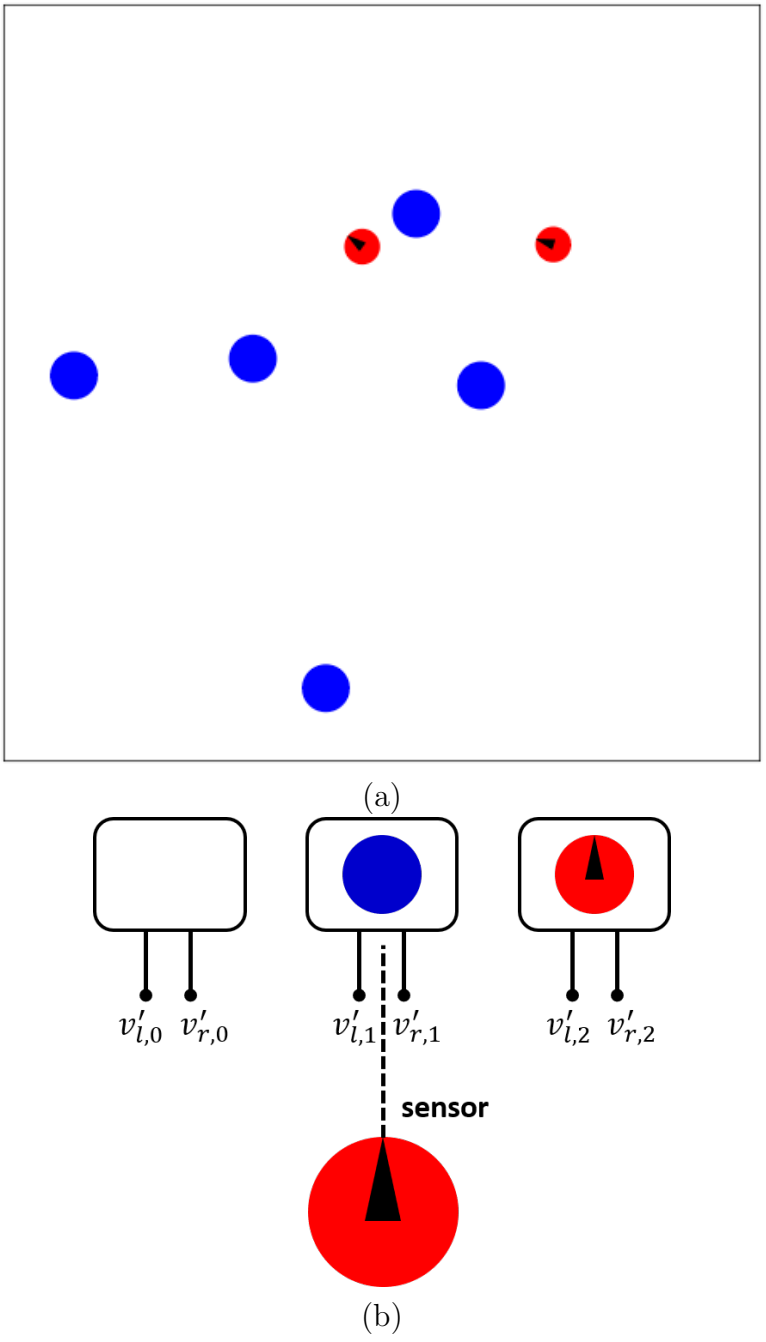


FIGURE 5.3: (a) The snapshot presents the *model simulation* environment of the object clustering study, where 5 blue objects and 2 robots (shown as disks with arrows) are placed randomly. Red colour indicates that the robots are operating a model controller. (b) The schematic illustrates the robot's controller, which maps what the sensor detects in its front onto a pair of wheel velocities.

5.3.1.2 Simulation Results

A set of 30 evolutions were conducted, and each evolution was run for 1000 generations. In each generation, $p_{\mathcal{M}} = 10$ model candidates were evaluated in the model simulation with different initial configurations and generated 10 model data sets. Additionally, the training simulation was implemented 10 times with different initial configurations to generate 10 training data sets. Each of these data sets was unlabelled and then evaluated by each of the $p_{\mathcal{D}} = 20$ discriminator candidates. At the end of each evaluation, fitness values were assigned according to Equation 2.8b and Equation 2.8a. The model candidate with the highest fitness value in the last generation was considered as the best controller found in that evolution, which gives 30 best controllers in total. We processed a post-evaluation process for each of these controllers, and each controller is reevaluated 100 times with different initial configurations. The controller with the lowest average value of $U(T)$ was chosen as the best controller throughout the 30 evolutions, as given in Table 5.2. Figure 5.4(a) shows its evolutionary dynamics during the learning with *Turing Learning*. Parameters $v_{l,0}^{'}$, $v_{r,1}^{'}$ and $v_{r,2}^{'}$ were inferred parameters within 100 generations, followed by the rest converging after 600 generations. Note that the first three convergent values are close to the velocity limit, and then the others are searched in the middle. That means the algorithm came up with a strategy to fix one parameter of each pair $(v_{l,I}^{'}, v_{r,I}^{'})$ and optimised the other one in later generations. To measure the quality of the controller, we compared its performance with the training controller $\boldsymbol{v}$ in terms of $U(T)$ during the post-evaluation process. Figure 5.4(b) shows that there is no significant difference between these two (paired Wilcoxon signed-ranked test[7] with $p = 0.49$).

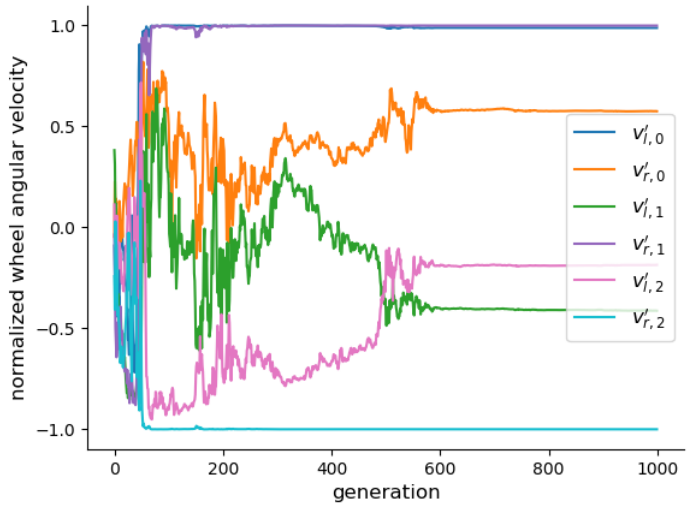
Behavioural Analysis

Considering the differential wheeled robot kinematics in Section 3.2.2, each sensor state I is mapped onto a circular movement of the robot: when $I = 0$ (nothing

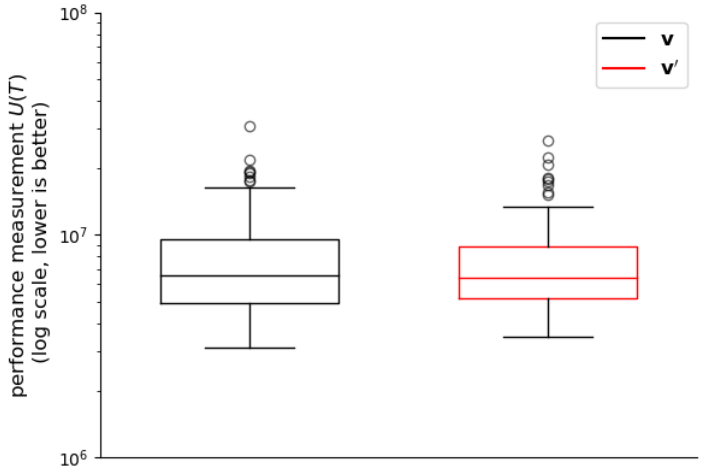
[7]The Wilcoxon signed-ranked test is used to compare data that is not assumed normally distributed. In this case study and the following case study, the test is used to compare the performance of different controllers.

TABLE 5.2: The best model controller $\boldsymbol{v}'$ for object clustering learned through *Turing Learning*.

$v'_{l,0}$	$v'_{l,1}$	$v'_{l,2}$
0.9873	-0.4130	-0.1857
$v'_{r,0}$	$v'_{r,1}$	$v'_{r,2}$
0.5745	0.9984	-0.9995



(a)



(b)

FIGURE 5.4: (a) Evolutionary dynamics of the parameters of the best model controller learned through *Turing Learning* for object clustering. (b) Comparison, in terms of the performance in object clustering task, between the training controller $\boldsymbol{v}$ found by the method in the previous study and the best model controller $\boldsymbol{v}'$ learned through *Turing Learning* in our formulation. Each box contains 100 trails in simulation.

detected), the robot travels in a clockwise manner; when $I = 1$ (an object detected), the robot takes a sharp turning in anticlockwise; when $I = 2$ (another robot detected), the robot avoids the other robot by reversing in clockwise. A sequence of snapshots in a simulation trial with 50 objects and 5 robots shows the emergent behaviour with the best model controller in Figure 5.5. Initially, all robots are inside of the object group, but they eventually manage to move to the periphery after 30 s. After this, the robots move in a circular formation around objects and push them towards the centroid so that the periphery shrinks until all objects end within one cluster after 800 s.

The emergent behaviour described above is similar to the one generated by the training controller in Table 5.1 and the controller reported in [24]. We also found that this behaviour is robust and can be observed with different initial configurations. However, the robots could miss the objects that are placed very far from others at the beginning. We found that 91 out of 100 simulations ended up with one cluster of objects.

Scalability Study

We investigated the performance of the best controller found in our formulation relative to the number of robots. We considered an environment with 50 objects and conducted 100 simulations with $n = 2, 5, 10, 15, 20, 30, 40, 50$ robots. Each simulation lasted for 1000 s (i.e. 10000 time steps in total). For each number of robots, we implemented the training controller $\boldsymbol{v}$ given in Table 5.1 and the best model controller $\boldsymbol{v}'$ given in Table 5.2 separately, and recorded the trajectories of objects. Figure 5.6(a) shows the compactness of objects at the final time step, which is defined as the second moment of objects $u(t)$ in Equation 5.6 with $t = T$. The dotted line indicates the lowest value of compactness for 50 objects theoretically, as given in [194]. Controller $\boldsymbol{v}'$ showed its benefits with a small number of robots (paired Wilcoxon signed-ranked test with $p < 0.001$ when $n = 5, 10$), but it failed to collect all objects in one cluster when $n \geq 15$. In contrast, controller $\boldsymbol{v}$ has better scalability with a large number of robots. The measurement $U(T)$

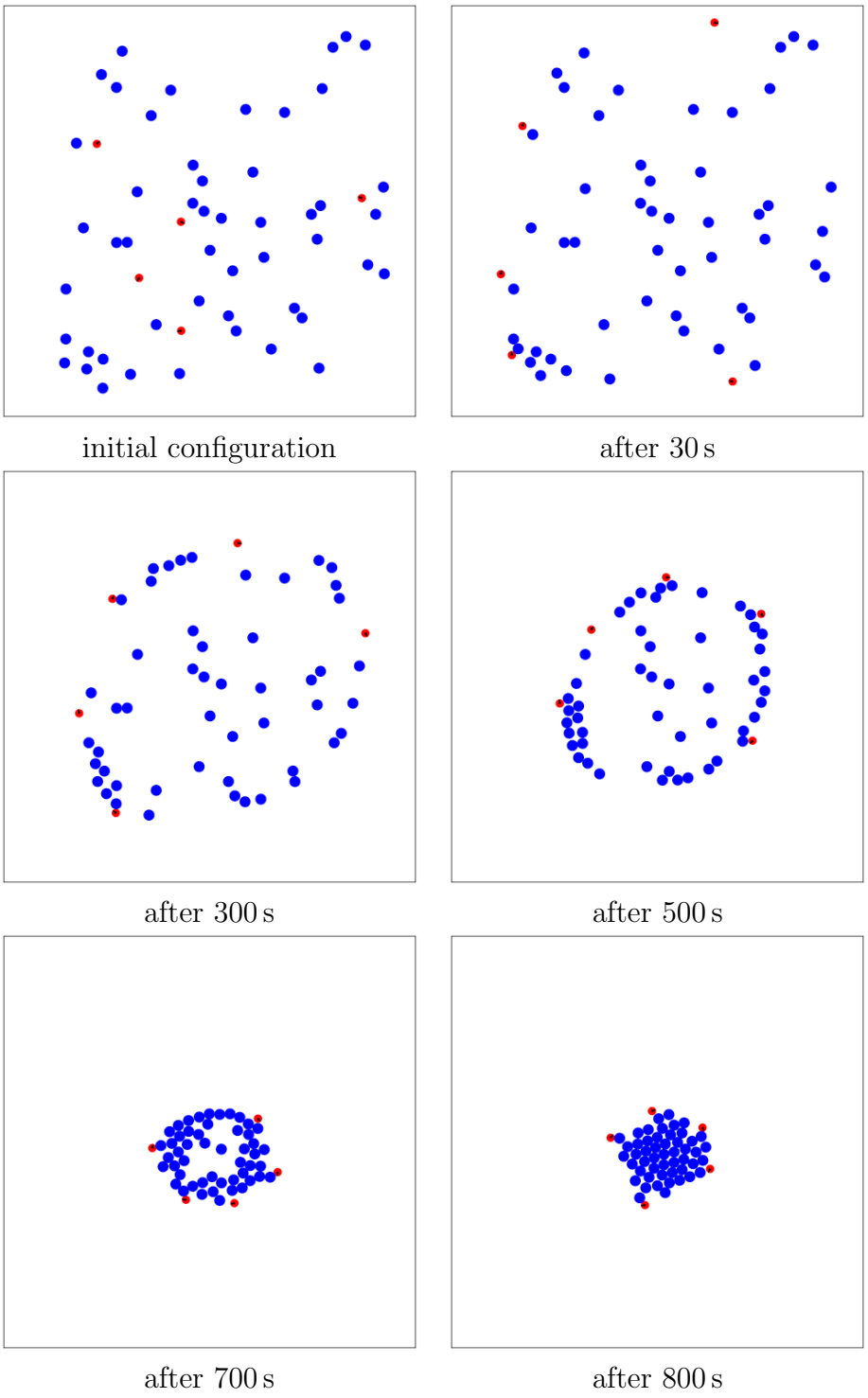


FIGURE 5.5: Emergent behaviour of object clustering with the best model controller. The robots (red disks) travel to the periphery and move in a circular formation. They push the objects (blue disks) inwards when they have contacts. All objects end within one cluster.

given in Equation 5.5 revealed a similar phenomenon as shown in Figure 5.6(b), where the dotted line indicates the value of $U(T)$ if no robot moves throughout the simulation (observed in Monte Carlo simulations [24]). Controller $\boldsymbol{v}'$ achieved its lowest value at $n = 10$ (paired Wilcoxon signed-ranked test with $p < 0.001$), which means that it is able to accomplish the task faster with 10 robots. But its performance degrades beyond this point. The performance of $\boldsymbol{v}$ keeps improving as the number of robots increases.

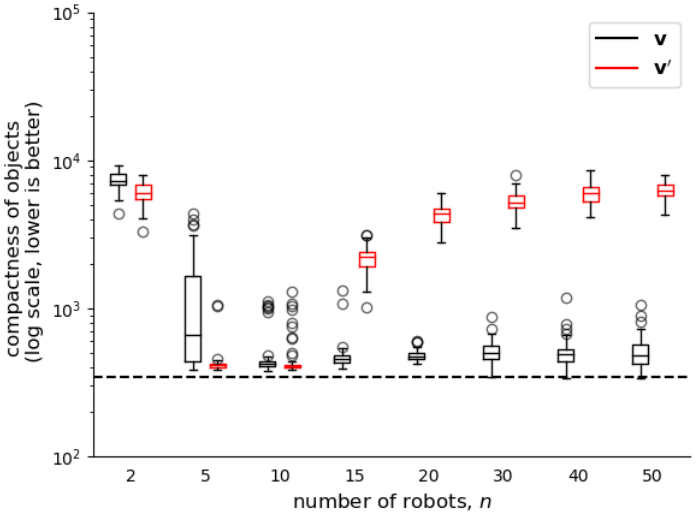
We observed that with controller $\boldsymbol{v}'$, as the number of robots increases, it becomes easier for robots to get stuck in the circular formation. This is due to the robot's vision being occluded by other robots for most of the time. The parameter setting in Table 5.2 didn't manage to deal with this situation. The reason why the solution was not further optimised in our formulation could be the *Turing Learning* learned from a small scale setup (5 objects and 2 robots) where the situation above is less likely to occur. Without a well-defined metric to measure the behaviour globally (e.g. for example, $U(T)$ used in [24]), the learning outcome is easily limited to the training data that has been given. To alleviate the problem, we re-ran the learning process but with a large scale setup of 50 objects and 50 robots and validated the best model controller with different numbers of robots. Figure 5.7 shows that the performance with a large number of robots has been dramatically improved and a small number of robots could accomplish the task as well with the same controller[8].

5.3.2 Shepherding

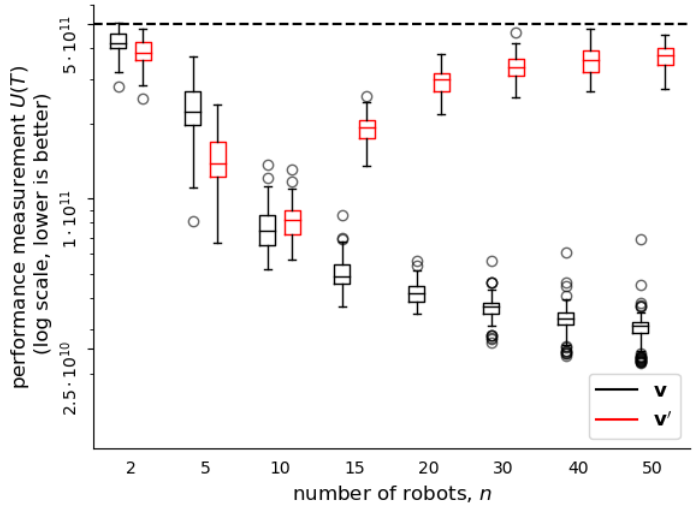
5.3.2.1 Simulation Setup

Same as the study in [147], we consider an environment with 10 shepherds, 20 sheep and one goal. The shepherd and the sheep are both simulated as e-puck robots. The shepherd robot is embedded with the sensor and controller described

[8]Its parameters are given as: $\boldsymbol{v}' = (v'_{l,0}, v'_{r,0}, v'_{l,1}, v'_{r,1}, v'_{l,2}, v'_{r,2}) = (0.743818, 1.0, 0.301877, -0.995316, 0.524595, 0.982407)$.

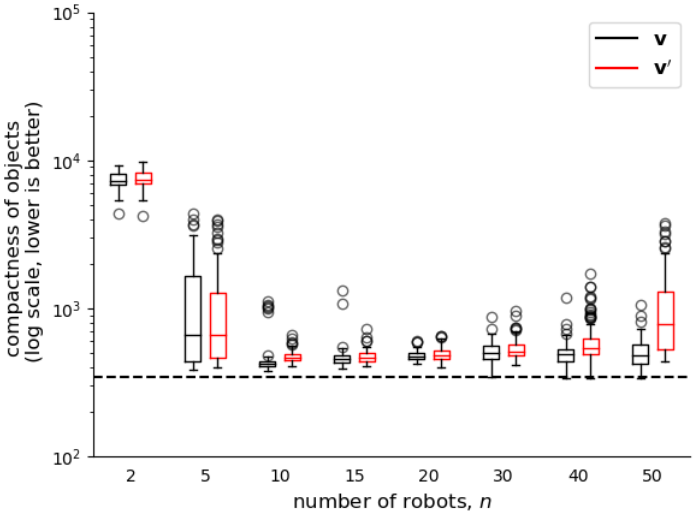


(a)

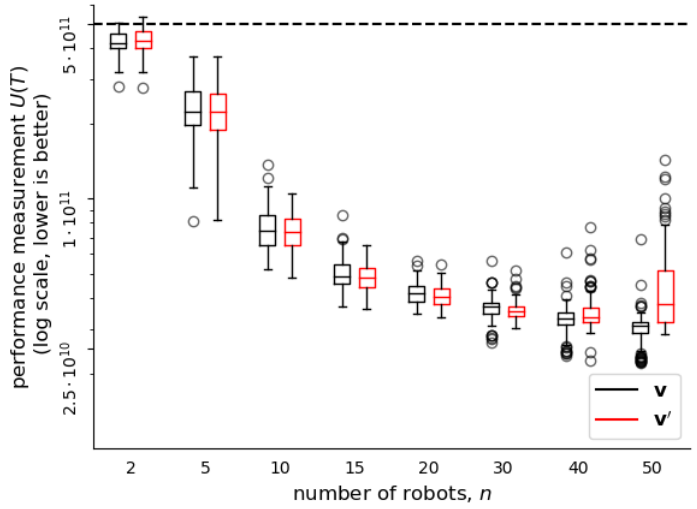


(b)

FIGURE 5.6: Both in (a) and (b), simulations were evaluated with different numbers of robots, n, operating the training controller $\boldsymbol{v}$ found by the method in literature and the best model controller $\boldsymbol{v}'$ learned through *Turing Learning* in our formulation. Each box contains 100 simulations. (a) Compactness of 50 objects at the final time step. The dotted black line represents the theoretical lowest value. (b) The performance quality of clustering 50 objects. The dotted black line represents the situation when robots do not move.



(a)



(b)

FIGURE 5.7: Both in (a) and (b), simulations were evaluated with different numbers of robots, n, operating the training controller $\boldsymbol{v}$ found by the method in the previous study and the best model controller $\boldsymbol{v}'$ learned through *Turing Learning* from a large scale setup of 50 objects and 50 robots. Each box contains 100 simulations. (a) Compactness of 50 objects at the final time step. The dotted black line represents the theoretical lowest value. (b) The performance quality of clustering 50 objects. The dotted black line represents the situation when robots do not move.

in Section 5.2.2. In this case, the shepherd's sensor has 4 states (i.e. $n = 4$ in Equation 5.1 and 5.2, which gives the model population size $p_{\mathcal{M}} = 20$): $I = 0$ when it detects nothing, $I = 1$ when it detects a sheep, $I = 2$ when it detects another shepherd and $I = 3$ when it detects the goal. According to [147, 195], the sheep robot's behaviour is determined and designed as a magnitude-dependent motion. Each sheep moves off all shepherds and all sheep as well, but with a lesser extent. The repulsion force that drives the sheep robot can be formalised as:

$$\boldsymbol{F}_i = \sum_{k \in S \setminus \{i\}} \frac{c_k}{\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2} \hat{\boldsymbol{r}}_{k_i}, \tag{5.7}$$

where S is the set of all agents, $\boldsymbol{x}_i$ is the position vector of sheep i, $\boldsymbol{x}_k$ is the position of agent k (either a shepherd or any other sheep), $\hat{\boldsymbol{r}}_{k_i}$ is the unit vector from agent k to sheep i, and $c_k = 450$, if agent k is a shepherd, or $c_k = 100$, if agent k is a sheep. The actual motion of each sheep is a combined result of the repulsion force and a default tendency to move forward, as shown in the following:

$$\begin{pmatrix} v_l \\ v_r \end{pmatrix} = \begin{pmatrix} K_1 & K_2 \\ K_1 & -K_2 \end{pmatrix} \begin{pmatrix} f_x \\ f_y \end{pmatrix} + \begin{pmatrix} u \\ u \end{pmatrix}, \tag{5.8}$$

where f_x and f_y are the components that the repulsion force applies on the sheep's local coordinate frame, $K_1 = 2.0$ and $K_2 = 1.3$ are the linear and angular gain. The default forward velocity u is 2.0 cm/s. The maximum velocity for a sheep is truncated at 6.4 cm/s.

Each sheep is assumed to be able to see all other sheep and shepherds but is not aware of the goal, which is modelled as a steady cylinder of radius 22.2 cm. The sheep and shepherds are initialised uniformly in a virtual circle of radius 200 cm and 400 cm away from the goal. Each sheep and shepherd are distributed with random orientations in $[-\pi, \pi]$. Additionally, we define a circular area of radius 100 cm centred with the goal as our goal region. Any sheep is considered to successfully reach the goal if it stays inside of the region.

Similarly, we have two simulations running in parallel: the *training simulation* and the *model simulation*, both with the same setup described above.

- *Training Simulation.* Individual shepherd operates the training controller $\boldsymbol{v}$, as shown in Figure 5.8. The controller was optimised by the method used in [147], where the objective function is given as $F(T)$[9]:

$$F(T) = \sum_{t=1}^{T} tf(t), \tag{5.9}$$

where T is the total number of the time steps of the simulation and

$$f(t) = \frac{1}{4nr_S^2} \sum_{i=1}^{m} ||\bar{\boldsymbol{x}}(t) - \boldsymbol{x}_i(t)||^2 ||\bar{\boldsymbol{x}}(t) - \boldsymbol{g}||^2, \tag{5.10}$$

where r_S and m are the number and radius of the sheep robot, $\bar{\boldsymbol{x}}(t)$ is the centroid of sheep swarm at time t and $\boldsymbol{x}_i(t)$ is the position of object i at time t. The position of the goal is represented as $\boldsymbol{g}$, which is not relative to the time. Thus, $f(t)$ evaluates how disperse the sheep swarm is and how far to the goal. Minimising $F(T)$ will optimise the controller to accomplish the task as fast as possible. We re-implemented the study in [147] and obtained the best controller given in Table 5.3[10]. In each simulation, shepherds execute the controller for 1500 s (i.e. 15000 time steps). At the same time, we let the discriminator randomly pick one sheep robot and monitors its absolute position $(x(t), y(t))$ changing throughout the simulation, which gives the exact same network structure as the one used in clustering study. The discriminator population size is $p_{\mathcal{D}} = 20$ as well. Figure 5.9 shows the snapshots of a training simulation lasting for 1500 s. The shepherds tend to cage all sheep and then herd them towards the goal, which is similar to the

[9] In order to replicate the previous study, $F(T)$ is adapted directly from [147] and is used to evaluate the performance of the controller learned with *Turing Learning* as well.

[10] Similar to the object clustering study, the controller we found here is different to the one in [147]. We assume these differences would not affect the learning outcome of our formulation as the shepherd's motion is not considered by *Turing Learning*. To maintain the consistency of our study, we use the above controller to generate training data samples.

TABLE 5.3: The training controller $\boldsymbol{v}$ operated during the training simulation of shepherding.

$v_{l,0}$	$v_{l,1}$	$v_{l,2}$	$v_{l,3}$
0.7895	0.9989	-0.0970	0.9779
$v_{r,0}$	$v_{r,1}$	$v_{r,2}$	$v_{r,3}$
1.0000	-0.4205	0.0501	0.9998

behaviour presented in [147]. The sheep flock reaches the goal region after 400 s.

- *Model Simulation.* As shown in Figure 5.10, the shepherds and sheep are initialised randomly in the same simulation environment as in the training simulation[11] and the model controller $\boldsymbol{v}'$ defined by a model candidate is operated by shepherds each time. The simulation lasts for 1500 s for each trial, during which the absolute positions of a randomly chosen sheep are collected as model data samples.

5.3.2.2 Simulation Results

A set of 30 evolutions were performed, and each evolution lasted for 80 generations. For each generation, 20 model data sets were generated from the model simulation with shepherds running each of the $p_{\mathcal{M}} = 20$ model candidates. In addition, 20 training data sets were collected from 20 trials of the training simulation. Each of these data sets was provided for each of the $p_{\mathcal{D}} = 20$ discriminator candidates to determine which label the data belongs to, after which fitness values were calculated based on Equation 2.8b and Equation 2.8a. At the end of each evolution, the best controller was selected as the model candidate with the highest fitness value. Then, each of the 30 best candidates was re-evaluated 100 times with different initial configurations, among which the average value of $F(T)$ in Equation 5.9 was computed. Finally, the one with the lowest $F(T)$ was considered as the best controller throughout 30 evolutions. Its parameters are given in Table 5.4, and the evolutionary dynamics of each parameter are shown in Figure 5.11(a). Unlike

[11] Similar to the object clustering study, the model simulation is initialised differently with the training simulation.

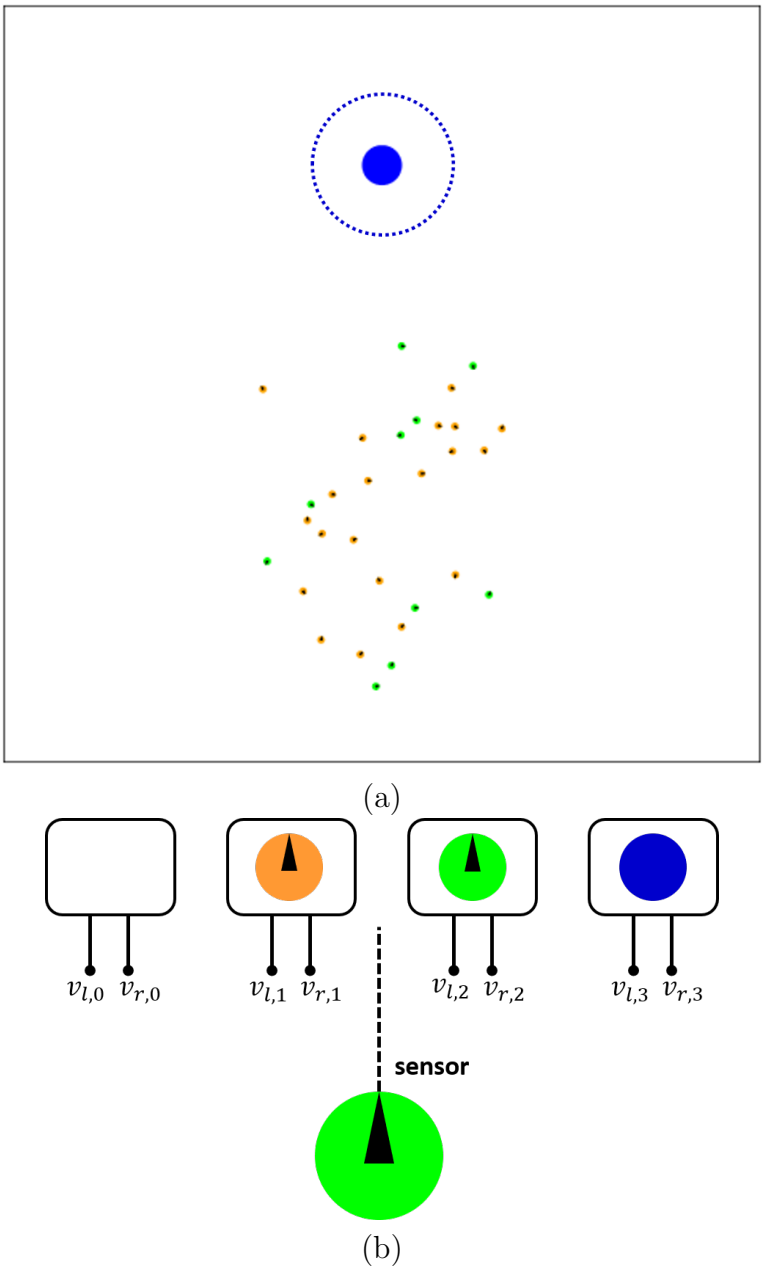


FIGURE 5.8: (a) A snapshot presents the *training simulation* environment of the shepherding study, where 20 sheep robots (orange disks) and 10 shepherd robots are randomly placed in a circular area of radius of 200 cm. The goal (blue disk) is located 400 cm away with a region of radius 100 cm (dotted blue circle). Green colour indicates that the shepherds are executing the training controller. (b) A schematic illustrates the shepherd's training controller, which maps what the sensor detects in its front onto a pair of wheel velocities.

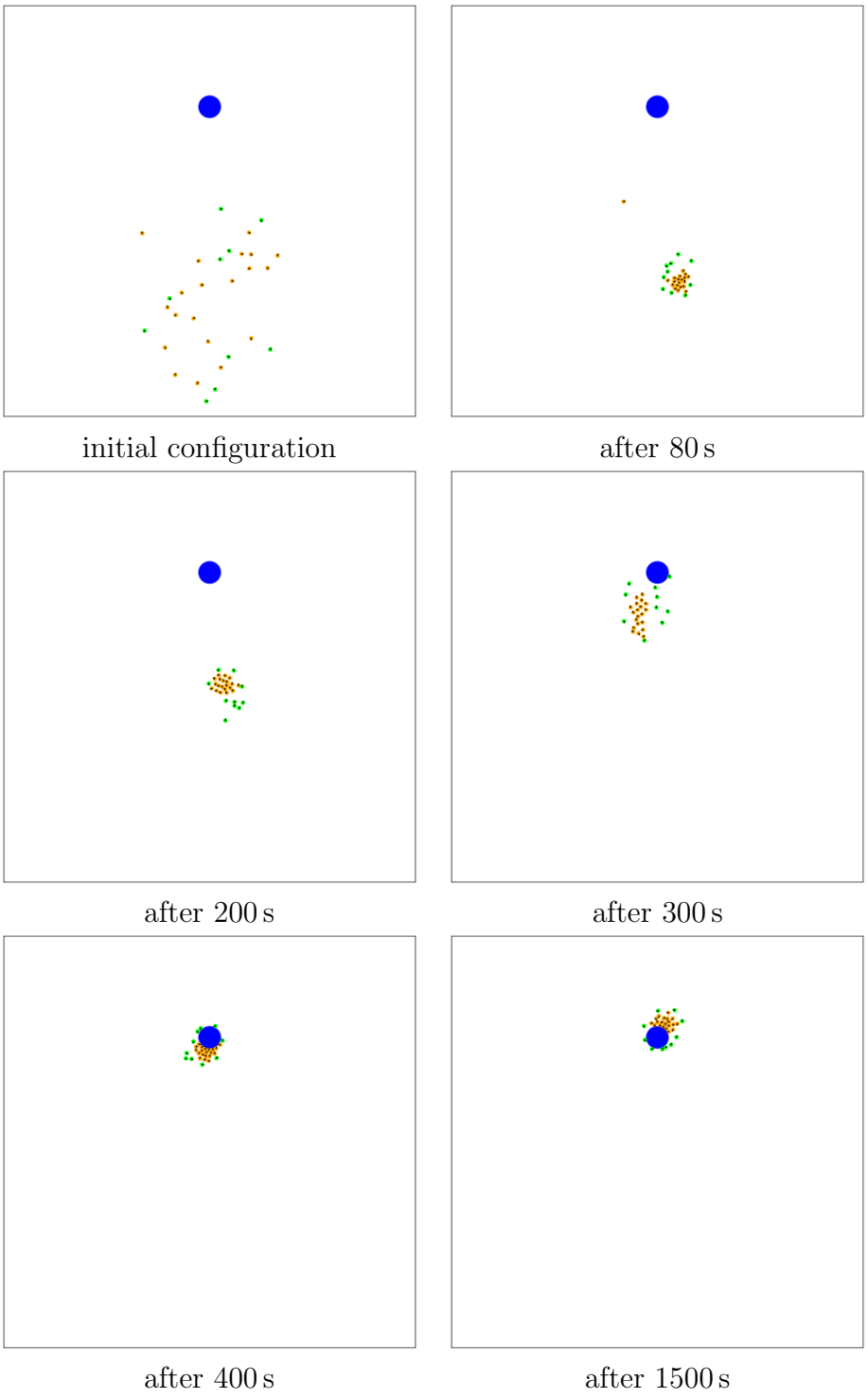


FIGURE 5.9: Emergent behaviour of shepherds (green disks) herding sheep (orange disks) towards the goal (blue disk) with the training controller. The absolute positions of the sheep are recorded as training data samples and evaluated by the discriminator in *Turing Learning*.

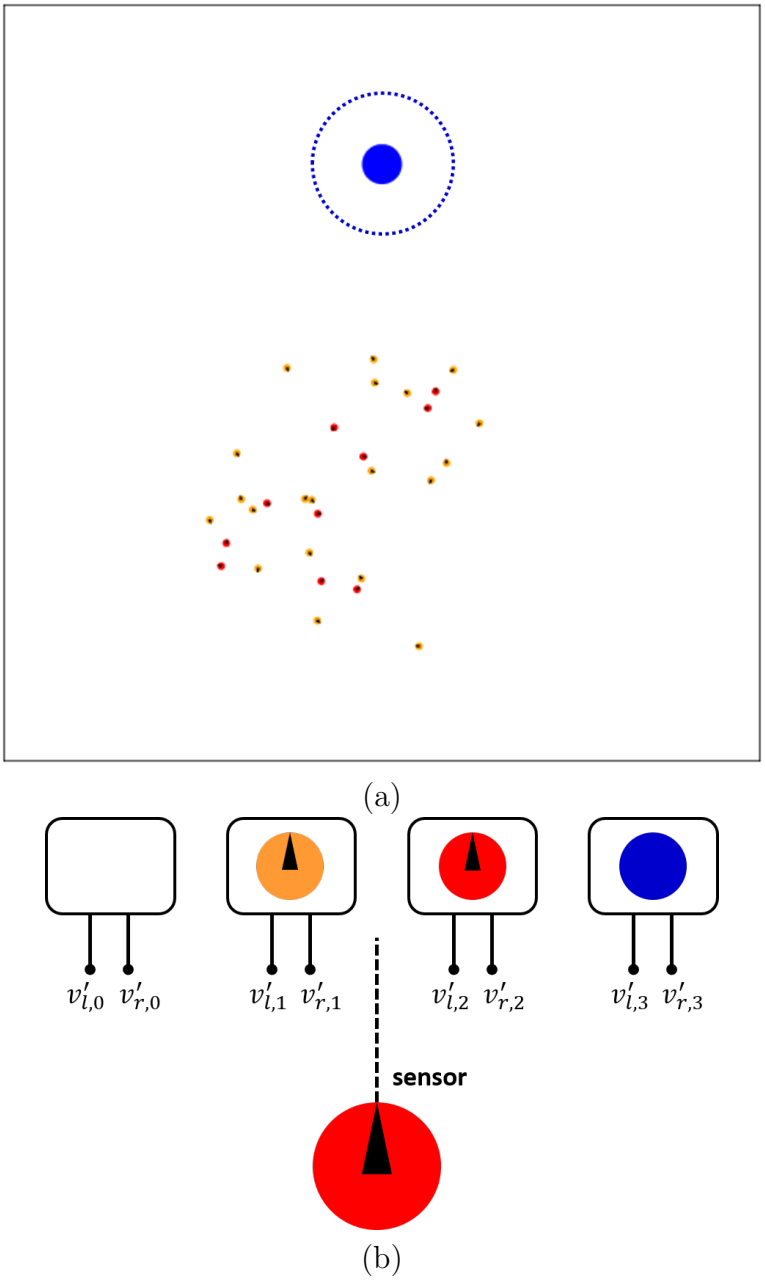


FIGURE 5.10: (a) A snapshot presents the *model simulation* environment of the shepherding study, where there are 20 sheep robots (orange disks), 10 shepherd robots and a steady goal (blue disk). They are initialised with the same rules as in the training simulation. Red colour indicates that the shepherds are executing the model controller. (b) A schematic illustrates the shepherd's model controller, which maps what the sensor detects in its front onto a pair of wheel velocities.

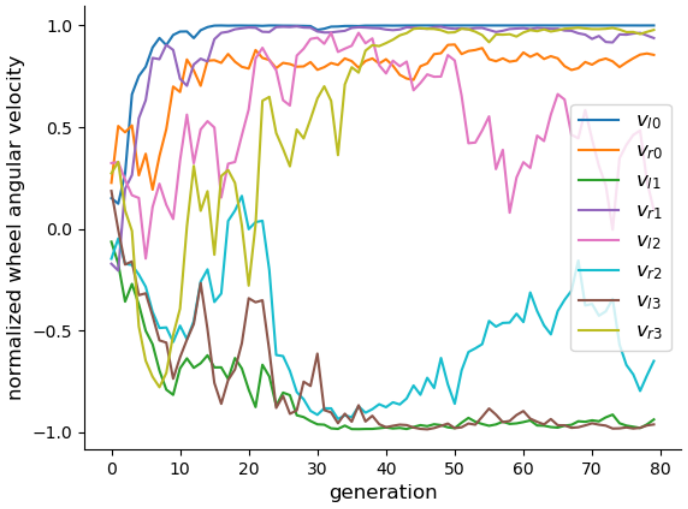
TABLE 5.4: The best model controller $\boldsymbol{v}'$ of shepherding learned through *Turing Learning*.

$v'_{l,0}$	$v'_{l,1}$	$v'_{l,2}$	$v'_{l,3}$
0.9998	-0.9370	0.0909	-0.9615
$v'_{r,0}$	$v'_{r,1}$	$v'_{r,2}$	$v'_{r,3}$
0.8551	0.9377	-0.6495	0.9779

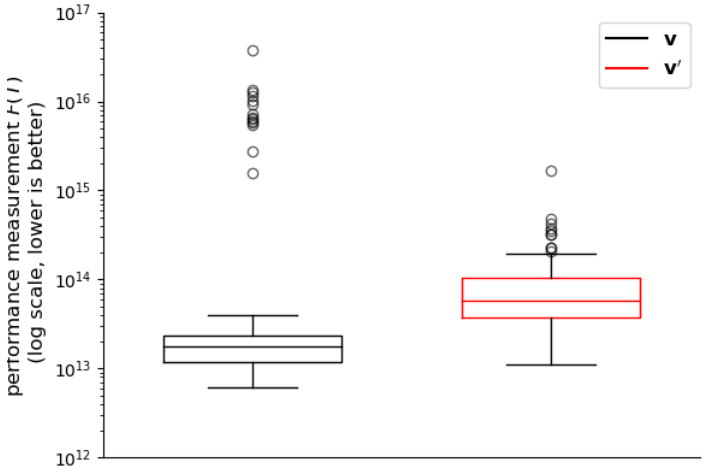
the object clustering study, the model came up with a strategy to learn the pair of wheel velocities with respect to each sensor state one by one. The $(v'_{l,0}, v'_{r,0})$ was firstly inferred after 10 generations, followed by $(v'_{l,1}, v'_{r,1})$ after 30 generations and $(v'_{l,3}, v'_{r,3})$ after 40 generations. All of these 3 pairs of parameters converged close to the maximums. $(v'_{l,2}, v'_{r,2})$ drifted in the middle. Figure 5.11(b) compares the quality of the best model controller $\boldsymbol{v}'$ with the training controller $\boldsymbol{v}$ in terms of $F(T)$ during the post-evaluation process. As can be seen, $\boldsymbol{v}$ achieved lower values, which means that it is able to accomplish the shepherding task faster than $\boldsymbol{v}'$. This is due to the different emergent behaviours generated by these two controllers.

Behavioural Analysis

Considering the differential wheeled robot kinematics in Section 3.2.2, each sensor state I is mapped onto a circular motion of the shepherds: when $I = 0$ (nothing detected), the shepherd behaves in a clockwise manner; when $I = 1$ (a sheep detected), the shepherd rotates in anticlockwise; when $I = 2$ (another shepherd detected), the shepherd reverses in clockwise to avoid the other shepherd; when $I = 3$ (the goal detected), the shepherd takes a similar move as $I = 1$ and rotates in anticlockwise. A sequence of snapshots of a simulation trial with 100 sheep and 40 shepherds shows the emergent behaviour with the best model controller in Figure 5.12. Initially, most shepherds are inside of the sheep flock, but they eventually manage to move to the periphery after 100 s. After this, the shepherds move to form a circular formation which includes all of the sheep and the goal inside, then force the sheep to move towards the centroid so that the periphery shrinks and all sheep stay in one cluster with the goal (after 600 s). Interestingly, if there are shepherds that are not required to keep all sheep together, they would



(a)



(b)

FIGURE 5.11: (a) Evolutionary dynamics of the parameters of the best model controller learned through *Turing Learning* for shepherding task. (b) The performance quality of the training controller $\boldsymbol{v}$ found by the method in the previous study and the best model controller $\boldsymbol{v}'$ learned through *Turing Learning* in our formulation. Each box contains 100 trails in simulation.

wander outside of the group. We also found that this behaviour is robust and can be observed with different initial configurations.

The emergent behaviour described above is significantly different to the one generated by the training controller in Table 5.3. At the beginning of the simulation, the shepherds prefer not to cage the sheep. Instead, they treat the goal as another "sheep" that does not move and orbit around them. This also explains why controller $\boldsymbol{v}'$ has higher values of $F(T)$ than $\boldsymbol{v}$. We also noticed that with such a feature, the controller could potentially deal with the situation where the goal's position is dynamically changing. To illustrate that, we tested both the training controller and the best model controller in an environment of twenty sheep, ten shepherds and one goal that moves relatively slowly in a circular. Figure 5.13 and Figure 5.14 present snapshots of the simulation with controller $\boldsymbol{v}$ and $\boldsymbol{v}'$ respectively. As can be seen, the caging behaviour is not very efficient to track the dynamical goal. However, by circling the goal and sheep all the time, controller $\boldsymbol{v}'$ is able to herd the sheep flock with respect to the changing of the goal's position.

The reason for this different behaviour could be that the *Turing Learning* formulation has no knowledge about the objective function $F(T)$. By observing the trajectory of a single sheep robot, the discriminator figures out that the sheep tends to move towards the goal in general. However, it could not infer how the sheep flock behaves due to the lack of global information. To validate that, We decomposed the objective function and extracted two key opponents: one measures the average distance from the individual sheep to their centroid; the other measures the distance from the centroid to the goal. We re-implemented the learning process of our formulation but fed the discriminator network with this global information. Unsurprisingly, a set of 30 evolutions ended with two different controllers. One controller (among 20 out of 30 evolutions) generates a similar behaviour as shown above, and the other one from the rest of evolutions performs similarly to the training controller found with the objective function $F(T)$[12].

[12]Such controller is given as: $\boldsymbol{v}' = (v'_{l,0}, v'_{r,0}, v'_{l,1}, v'_{r,1}, v'_{l,2}, v'_{r,2}, v'_{l,3}, v'_{r,3}) = (0.935648, 0.755371, -0.990944, 0.997309, 0.0677969, -0.623122, 0.534335, 0.830297)$.

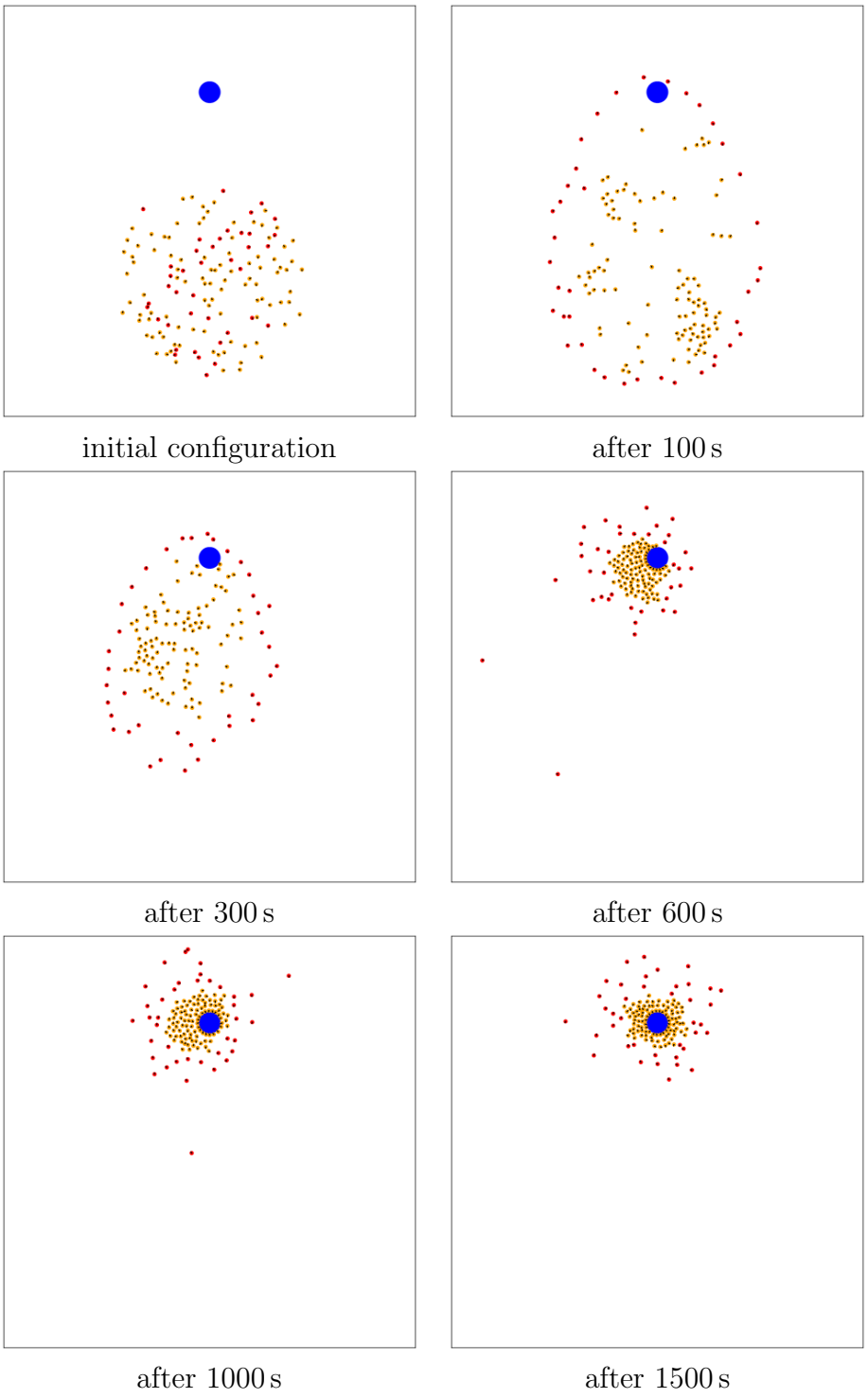


FIGURE 5.12: Emergent behaviour of shepherding with the best model controller. The shepherds (red disks) travel to the periphery and move in a circular formation that encompasses the goal (blue disk). They herd the sheep (orange disks) moving inwards so that all sheep successfully reach the goal.

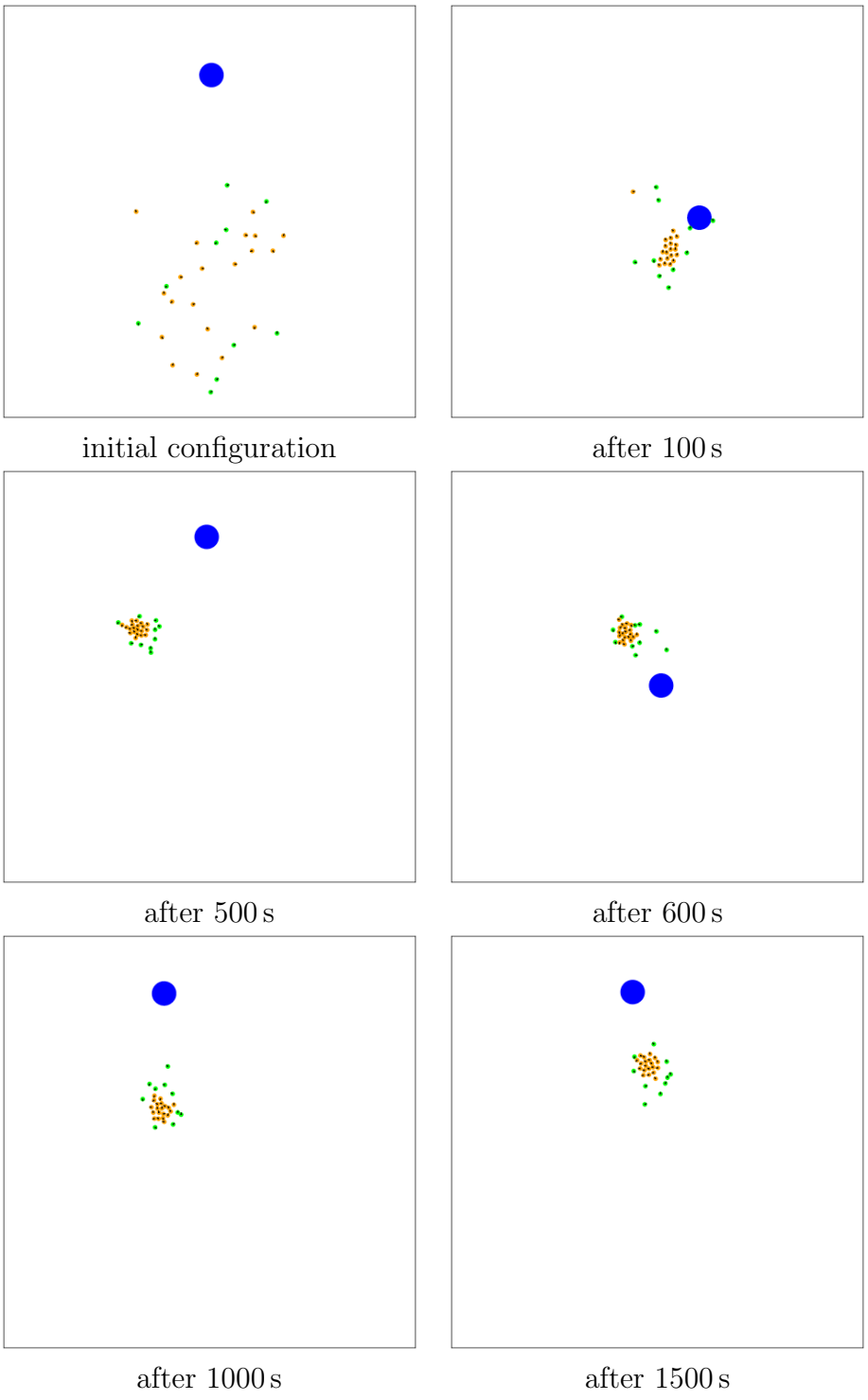


FIGURE 5.13: Emergent behaviour of shepherding with the training controller and a dynamical goal (blue disk). The shepherds (green disks) prefer to cage the sheep (orange disks) but only reach the goal if it is close to them.

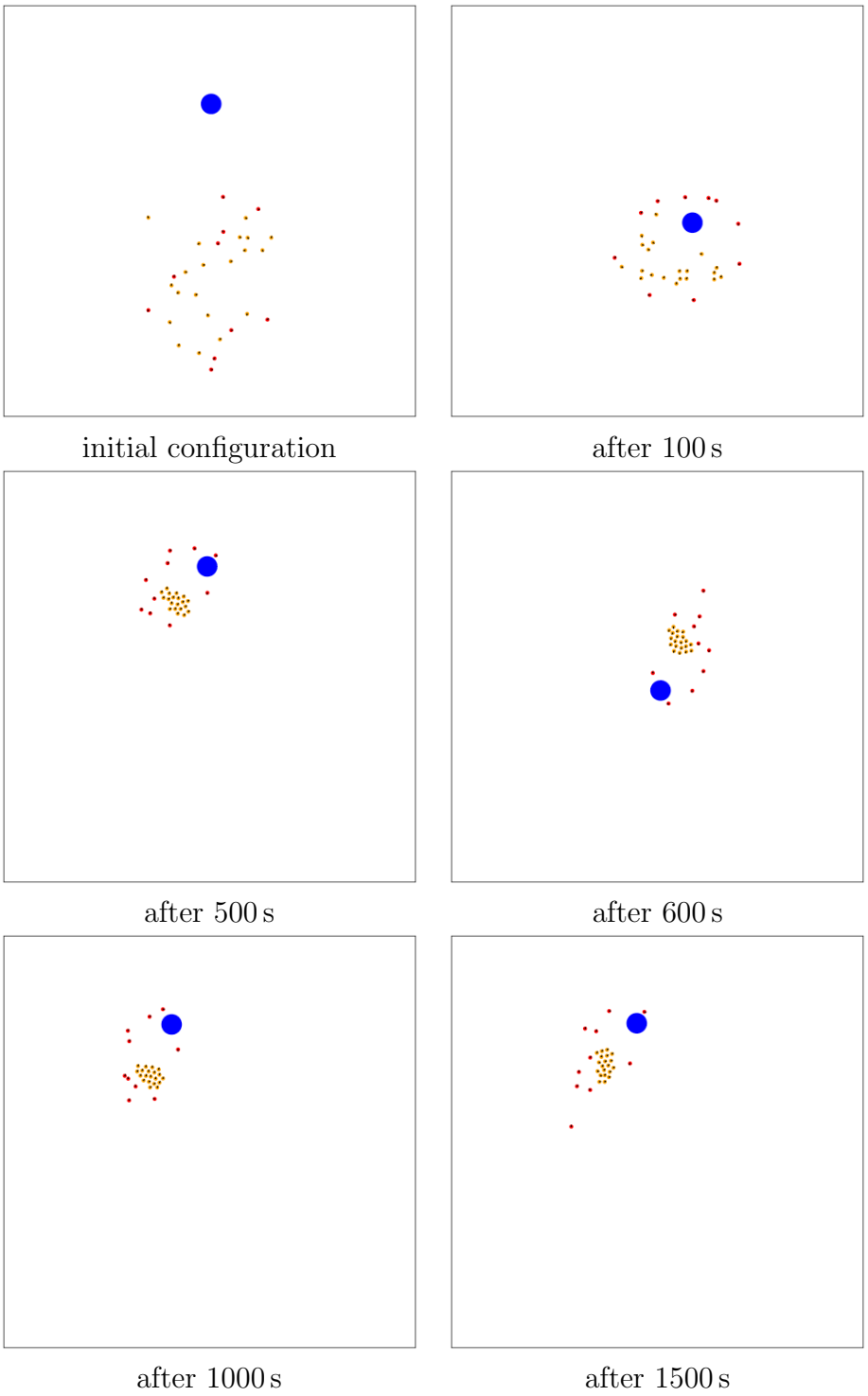


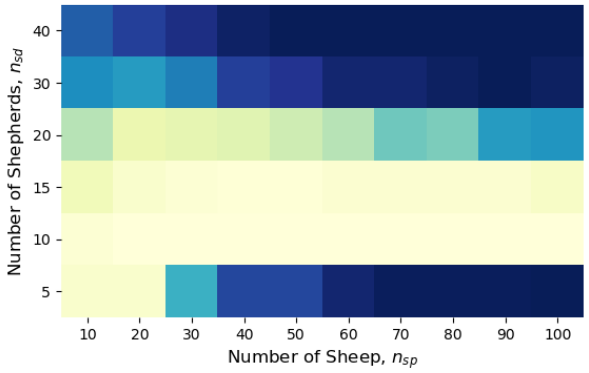
FIGURE 5.14: Emergent behaviour of shepherding with the best model controller found in our formulation and a dynamical goal (blue disk). The shepherds (green disks) manage to herd the sheep flock (orange disks) following the goal.

Scalability Study

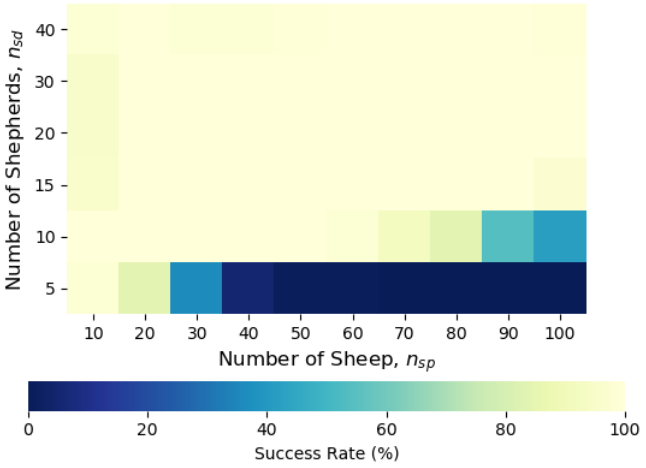
We examined the performance of the best model controller found in our formulation relative to the number of shepherds ($n_{sd} \in 5, 10, 15, 20, 30, 40$) and sheep ($n_{sp} \in 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$). For each combination of n_{sd} and n_{sp}, we performed 100 simulations with the training controller $\boldsymbol{v}$ given in Table 5.3 and the model controller $\boldsymbol{v}'$ given in Table 5.4 separately. Each simulation lasted for 1500 s (i.e. 15000 time step in total). To measure the performance, we used the *success rate*, which is exactly the same as the one defined in [147] and calculated the percentage of sheep inside of the goal region at the end of the simulation. Results are shown in Figure 5.15. The performance of controller $\boldsymbol{v}$ degrades as the number of shepherds is scaled up to 20, which is due to the same reason reported in [147]. As the density of robots increases in the environment, the shepherd's vision could be easily occluded by others. The performance of controller $\boldsymbol{v}'$ scales up well to a large number of shepherds. With enough shepherds, it can always accomplish the task with almost all sheep reside within the goal region. This is very similar to the extended controller in [147], which has two additional sensor states to ensure the goal is constantly detected by the shepherd.

5.4 Summary

This chapter has presented the simulation results of the *Turing Learning* algorithm inferring swarm behaviours indirectly from their effects. In the case study of object clustering, our method successfully found a controller for robots to collect all objects in one cluster. The algorithm requires minimal information. The inference has been done by only observing the trajectory of a single object. The emergent behaviour is visually close to that of the reference swarm system, but the controller has different parameter settings. The scalability analysis showed that, with such a controller, a small number of robots is enough to cluster a relatively large number of objects. However, the controller is not well scaled up to a large number of robots. By providing the training data from a large scale simulation, the



(a)



(b)

FIGURE 5.15: Success rates of shepherding (averaged over 100 trials in simulation) for different numbers of sheep and shepherds. (a) The training controller found by the method reported in the previous study (b) The best model controller learned through *Turing Learning* in our formulation.

scalability could be dramatically improved. The process of obtaining training data samples could be inefficient, especially when considering coevolutionary robotics, given a large number of robots evolved in the simulation. The learning process could be tailored according to the availability of sources. In the case study of shepherding, our method successfully found a controller for shepherd robots to herd all sheep towards the goal position. Again, minimal information is required. The inference has been done by monitoring the trajectory of a single sheep robot only.

From a behavioural analysis, the result controller generated a different emergent behaviour, where the shepherd robot considered the goal as another "sheep" and forced all sheep to move together, that is, close to the goal. This is due to the lack of global information on the sheep flock. Such controller has shown better scalability with respect to different numbers of shepherds and sheep compared with the pre-designed one. With enough shepherds, it is able to accomplish the task with a relatively large number of sheep. An additional study showed that the learned controller has the potential to be implemented in a situation with a dynamical goal position. In this chapter, our method has no knowledge about the robot agent which operates the pre-designed controller. Thus, it is not necessary to compare the parameter values. The learned controller has been evaluated with manually designed metrics, and its emergent behaviour has been visually examined.

Chapter 6

Conclusion

6.1 Summary

This thesis presented the advances and applications of the coevolutionary *Turing Learning* algorithm in which the population of model candidates and the population of discriminator candidates are optimised simultaneously in a competitive game setting through coevolutionary approaches. Advances are regarding the techniques that can improve the convergence of coevolution towards better model accuracy and methods that can achieve a low computational cost without a decrease in model accuracy. Applications are regarding the inference of a robotic agent's characteristics and the inference of the behavioural controllers for swarm behaviours. We summarise our findings as follows.

- We extended the discriminator in *Turing Learning* with the ability to influence the sampling process through genuine interactions with the model and the system under investigation and showed that the interactive approach leads to a better model accuracy compared with the passive approach where the model and the system are passively observed by the discriminator. We demonstrated the approach with two case studies: one for a robotic agent inferring its sensor configuration and the other for a robotic agent calibrating

its distance sensor. The later case study leads to the self-modelling/self-discovery process of intelligent robots. In both case studies, the interactive discriminator can freely control the robot's movements while observing the robot's sensor readings in a simulated environment with obstacles. In the case study of inferring the configuration, the discriminator came up with a control strategy to examine the robot's sensor configuration by tracking obstacles it perceives from any direction. We validated that the sensor-to-motor correlation is essential for the discrimination task, which means a fixed sequence of interactions would not be sufficient. In the case study of calibration, we investigated the effect of the frequency of the discriminator's interaction and found that the less frequent the interaction is, the more often the linear movements (i.e. moving backwards or forward) occur. The inference tasks in these two case studies are not simple, as the robot starts from a random location and has no access to the global information (i.e. relative position and distance to obstacles), and randomness of its motion and sensing are introduced by noise.

- We presented a hybrid formulation of *Turing Learning* where the discriminator is a combination of the interactive one and the passive one. We demonstrated this hybrid formulation in the calibration case study. We validated that the hybrid discriminator can retain the advantage of the interactive discriminator and achieve an almost 50% reduction of the training cost by reusing recorded data for the passive discriminator. We also demonstrated a novel reward mechanism along with the hybrid formulation in the calibration study. The mechanism evaluates a candidate's exclusiveness with respect to other candidates in the population and promotes the candidate that performs well against rarely defeated opponents. Meanwhile, designers can adjust the amount of exclusiveness to be evaluated for the desired performance. Results showed that with a certain amount of exclusiveness, the coevolution process of *Turing Learning* could be enhanced, at the same time, without a decrease in model accuracy.

- We proposed a novel approach to infer the local behavioural controllers operated by individuals in a swarm of robots indirectly from the effects on the environment. We presented two case studies: inference of object clustering behaviour and shepherding behaviour. Instead of observing the movements of individual robots as implemented in previous studies in [4], the discriminator of *Turing Learning* observes the positions of a randomly picked object to be clustered, or the positions of a randomly picked sheep in the flock. We compared the performance of the controller inferred with our approach to that of the controller learned in previous studies in [24] and [147]. Results showed that the indirect observation is sufficient to infer the behavioural controllers for swarm behaviours with good quality or even better performance in the scalability. This indirect approach requires a small amount of data and encourages diverse solutions. In particular, the inferred clustering controller can successfully cluster a large number of objects with a relatively small number of robots, and the inferred shepherding controller can effectively herd the sheep flock towards a dynamical goal.

6.2 Future Work

Encouraged by the promising finding of this thesis, some future work could focus on:

- *Design protocols.* The coevolutionary framework *Turing Learning* is not easy to implement as there are many hyperparameters to be determined, for example, the number of the population size, the number of generations and settings regarding the simulation. In this thesis, these are chosen from preliminary experiments and by experience as well. However, how different parameters affect the learning outcomes needs a systematic study. Moreover, an appropriate topology of the discriminator network is required to understand the possible data patterns. Otherwise, the model is not able to evolve towards the desired one. We continuously use the Elman recurrent network,

which has shown good performance in the case studies presented in previous works and this thesis. However, how different network structures affect learning outcomes is not well studied. Thus, a proper protocol to design the framework would be helpful to implement *Turing Learning* in future studies.

- *Physical demonstrations.* The case studies presented in this thesis were conducted in simulation. There are two types of simulations running in parallel: the *training simulation* and the *model simulation*. In a physical scenario, the *model simulation* could run on a PC or on the physical robot's onboard processor, and the *training simulation* would be replaced by the data sampling process in a real-world environment. This leads to the self-learning capability of a physical robot. Challenges to conducting physical experiments include: (1) the synchronisation between the physical robot and the software running on PC; (2) tracking of the physical robot to generate input samples for the discriminator. Meanwhile, it is interesting to investigate whether a joint model could be inferred by a combined discrimination task. For example, the inference of the sensor configuration and the sensor calibration could be conducted simultaneously. We list two possible ways to achieve the joint inference: (1) running two sub-*Turing Learning* algorithms in parallel, which requires a suitable synchronisation mechanism; (2) inferring a joint model that is expressive enough to cover both inference tasks.

- *Complex representations.* As described above, one option is to evolve complex models in *Turing Learning*. In this thesis, models were represented as vectors of real-valued parameters, which is not sufficient to describe complex systems, such as biological systems. One might notice that there is a large difference in terms of the magnitudes of model parameters and discriminator parameters. It is not clear whether this difference is essential for a good performance of the current framework. Preliminary experiments have shown that evolving neural networks as the model with *Turing Learning* is difficult. It is promising to improve the current framework so that complex models can be inferred. Another option is the complexity of the inference task in general. Some future work could focus on the inference of real-world

data, including behaviour data, such as trajectory data of the fish school, and non-behavioural data, such as realistic images, through *Turing Learning*, especially the coevolutionary framework of *Turing Learning*.

- *Interactive approach to inferring swarm behaviours indirectly.* This future work is mostly curiosity-driven. The interactive approach presented in Chapter 3 and Chapter 4 could potentially be adapted to the case studies in Chapter 5. One option could be implementing the interactive discriminator as a manipulator, which can manipulate the objects or the sheep during the learning process.

Bibliography

[1] A. M. Turing. I.—Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 10 1950.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, MA,USA, 2014. MIT Press.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[4] Wei Li, Melvin Gauci, and Roderich Groß. Turing learning: a metric-free approach to inferring behavior and its application to swarms. *Swarm Intelligence*, 10(3):211–243, 2016.

[5] Wei Li, Melvin Gauci, and Roderich Groß. A coevolutionary approach to learn animal behavior through controlled interaction. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 223–230, 2013.

[6] Wei Li, Melvin Gauci, and Roderich Groß. Coevolutionary learning of swarm behaviors without metrics. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 201–208, 2014.

[7] Roderich Groß, Yue Gu, Wei Li, and Melvin Gauci. Generalizing gans: a turing perspective. *Advances in neural information processing systems*, 30:6316–6326, 2017.

[8] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[9] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017.

[10] Thomas Unterthiner, Bernhard Nessler, Calvin Seward, Günter Klambauer, Martin Heusel, Hubert Ramsauer, and Sepp Hochreiter. Coulomb gans: Provably optimal nash equilibria via potential fields. *arXiv preprint arXiv:1708.08819*, 2017.

[11] Vaishnavh Nagarajan and J Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in neural information processing systems*, pages 5585–5595, 2017.

[12] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *Advances in neural information processing systems*, pages 1825–1835, 2017.

[13] Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado. Coegan: evaluating the coevolution effect in generative adversarial networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 374–382, 2019.

[14] Victor Costa, Nuno Lourenço, and Penousal Machado. Coevolution of generative adversarial networks. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 473–487. Springer, 2019.

[15] Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado. Neuroevolution of generative adversarial networks. In *Deep Neural Evolution*, pages 293–322. Springer, 2020.

[16] Shiming Chen, Wenjie Wang, Beihao Xia, Xinge You, Zehong Cao, and Weiping Ding. Cde-gan: Cooperative dual evolution based generative adversarial network. *arXiv preprint arXiv:2008.09388*, 2020.

[17] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary computation*, 5(1):1–29, 1997.

[18] Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[19] Michael Brady. Artificial intelligence and robotics. *Artificial intelligence*, 26(1):79–121, 1985.

[20] Javier Andreu Perez, Fani Deligianni, Daniele Ravi, and Guang-Zhong Yang. Artificial intelligence and robotics. *arXiv preprint arXiv:1803.10813*, 2018.

[21] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[22] Vinesh Raja and Kiran J Fernandes. *Reverse engineering: an industrial perspective*. Springer Science & Business Media, 2007.

[23] Johan J Bolhuis and Luc-Alain Ed Giraldeau. *The behavior of animals: Mechanisms, function, and evolution.* Blackwell Publishing, 2005.

[24] Melvin Gauci, Jianing Chen, Wei Li, Tony J Dodd, and Roderich Groß. Clustering objects with robots that do not compute. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 421–428, 2014.

[25] Ayse Pinar Saygin, Ilyas Cicekli, and Varol Akman. Turing test: 50 years later. *Minds and machines*, 10(4):463–518, 2000.

[26] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[27] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[28] A Newell and JC Shaw. A variety op intelligent learning in a general problem solver. *RAND Report P-1742, dated July*, 6, 1959.

[29] James Lighthill. Artificial intelligence: A general survey. In *Artificial Intelligence: a paper symposium*, pages 1–21. Science Research Council London, 1973.

[30] Shu-Hsien Liao. Expert system methodologies and applications—a decade review from 1995 to 2004. *Expert systems with applications*, 28(1):93–103, 2005.

[31] Andreas Kaplan and Michael Haenlein. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, 62(1):15–25, 2019.

[32] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[33] Eugene Charniak. *Introduction to artificial intelligence*. Pearson Education India, 1985.

[34] Blay Whitby. *Reflections on artificial intelligence*. Intellect Books, 1996.

[35] John McCarthy. What is artificial intelligence. *Computer Science Department, Stanford University*, page 2, 2007.

[36] John R Searle et al. Minds, brains, and programs. *The Turing Test: Verbal Behaviour as the Hallmark of Intelligence*, pages 201–224, 1980.

[37] Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159, 1991.

[38] Rodney A Brooks et al. Intelligence without reason. *Artificial intelligence: critical concepts*, 3:107–63, 1991.

[39] Rodney A Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15, 1990.

[40] Robin R Murphy. *Introduction to AI robotics*. MIT press, 2019.

[41] Nils J Nilsson et al. Shakey the robot. 1984.

[42] Rodney A Brooks. New approaches to robotics. *Science*, 253(5025):1227–1232, 1991.

[43] Ronald C Arkin, Ronald C Arkin, et al. *Behavior-based robotics*. MIT press, 1998.

[44] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[45] Marc H Raibert. *Legged robots that balance*. MIT press, 1986.

[46] Rodney A Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2):253–262, 1989.

[47] Alessandro Saffiotti, Kurt Konolige, and Enrique H Ruspini. A multivalued logic approach to integrating planning and control. *Artificial intelligence*, 76(1-2):481–526, 1995.

[48] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Aaai/iaai*, pages 11–18, 1998.

[49] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, et al. Minerva: A second-generation museum tour-guide robot. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 3. IEEE, 1999.

[50] Nicola Muscettola, P Pandurang Nayak, Barney Pell, and Brian C Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial intelligence*, 103(1-2):5–47, 1998.

[51] Kanna Rajan, Douglas Bernard, Gregory Dorais, Edward Gamble, Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, P Pandurang Nayak, Nicolas F Rouquette, et al. Remote agent: An autonomous control system for the new millennium. In *ECAI*, volume 14, pages 726–730, 2000.

[52] Charles Darwin. *The origin of species*. PF Collier & son New York, 1909.

[53] Michael M Desai and Daniel S Fisher. Beneficial mutation–selection balance and the effect of linkage on positive selection. *Genetics*, 176(3):1759–1798, 2007.

[54] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[55] David B Fogel. Evolutionary computation. the fossil record. selected readings on the history of evolutionary computation. *Classifier Systems*, 1998.

[56] Thomas Back and H-P Schwefel. Evolutionary computation: An overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29. IEEE, 1996.

[57] Sewall Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. 1932.

[58] Kenneth De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 185–199, 2016.

[59] George J Friedman. Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, UCLA, Engineering, 1956.

[60] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[61] Darrell Whitley. Genitor: A different genetic algorithm. In *Proceedings of the Rocky Mountain conference on artificial intelligence, 1988*, pages 118–130, 1988.

[62] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, 1978.

[63] Hans-Paul Paul Schwefel. *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., 1993.

[64] David B Fogel. *Evolutionary computation*. IEEE Press, 1995.

[65] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

[66] David B Fogel and J Wirt Atmar. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63(2):111–114, 1990.

[67] John R Koza. Non-linear genetic algorithms for solving problems by finding a fit composition of functions, August 4 1992. US Patent 5,136,686.

[68] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. Handbook of evolutionary computation. *Release*, 97(1):B1, 1997.

[69] Agoston E Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482, 2015.

[70] John N Thompson. Concepts of coevolution. *Trends in Ecology & Evolution*, 4(6):179–183, 1989.

[71] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary principles., 2012.

[72] Robert Axelrod. The evolution of strategies in the iterated prisoner's dilemma. *Genetic algorithms and simulated annealing*, pages 32–41, 1987.

[73] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

[74] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994.

[75] Phil Husbands. Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In *AISB Workshop on Evolutionary Computing*, pages 150–165. Springer, 1994.

[76] Jan Paredis. The symbiotic evolution of solutions and their representations. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*. Morgan Kaufmann Publishers, 1995.

[77] Lawrence Bull, Terence C Fogarty, C Langton, and K Shimohara. Horizontal gene transfer in endosymbiosis. In *Proceedings of the 5th International Workshop on Artificial Life: Synthesis and Simulation of Living Systems (ALIFE-96)*, pages 77–84. MIT Press, Cambridge, MA, 1997.

[78] W Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

[79] Jan Paredis. Coevolutionary computation. *Artificial life*, 2(4):355–375, 1995.

[80] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[81] Liviu Panait and Sean Luke. A comparative study of two competitive fitness functions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 503–511. Citeseer, 2002.

[82] Sean Luke et al. Genetic programming produced competitive soccer softbot teams for robocup97. *Genetic Programming*, 1998:214–222, 1998.

[83] Josh C Bongard and Hod Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

[84] John Cartlidge and Seth Bullock. Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary computation*, 12(2):193–222, 2004.

[85] Hugues Juille and Jordan B Pollack. Coevolving the" ideal" trainer: Application to the discovery of cellular automata rules. In *University of Wisconsin*. Citeseer, 1998.

[86] Shiu Yin Yuen, Yang Lou, and Xin Zhang. Selecting evolutionary algorithms for black box design optimization problems. *Soft Computing*, 23(15):6511–6531, 2019.

[87] Melvin Gauci. *Swarm robotic systems with minimal information processing*. PhD thesis, University of Sheffield, 2014.

[88] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[89] Lennart Ljung. System identification. *Wiley encyclopedia of electrical and electronics engineering*, pages 1–19, 1999.

[90] Kristinn Kristinsson and Guy Albert Dumont. System identification and control using genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1033–1046, 1992.

[91] IJ Leontaritis and Stephen A Billings. Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International journal of control*, 41(2):303–328, 1985.

[92] Carlos M Fonseca. Non-linear model term selection with genetic algorithms. In *Proceedings of IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, volume 2, pages 271–278, 1993.

[93] GC Luh and CY Wu. Non-linear system identification using genetic algorithms. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 213(2):105–118, 1999.

[94] Jinyao Yan and John R Deller Jr. Narmax model identification using a set-theoretic evolutionary approach. *Signal Processing*, 123:30–41, 2016.

[95] Peter Marenbach and Kurt D Bettenhausen. Data-driven structured modelling of a biotechnological fed-batch fermentation by means of genetic programming. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 211(5):325–332, 1997.

[96] Gary J Gray, David J Murray-Smith, Yun Li, Ken C Sharman, and Thomas Weinbrenner. Nonlinear model structure identification using genetic programming. *Control Engineering Practice*, 6(11):1341–1352, 1998.

[97] Katya Rodriguez-Vazquez, Carlos M Fonseca, and Peter J Fleming. Multiobjective genetic programming: A nonlinear system identification application. In *Late breaking papers at the 1997 genetic programming conference*, pages 207–212. Citeseer, 1997.

[98] William B Langdon and Riccardo Poli. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer, 1998.

[99] Dag Ljungquist and Jens G Balchen. Recursive prediction error methods for online estimation in nonlinear state-space models. In *Proceedings of 32nd IEEE Conference on Decision and Control*, pages 714–719. IEEE, 1993.

[100] SH Choi, CO Lee, and HS Cho. Friction compensation control of an electropneumatic servovalve by using an evolutionary algorithm. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 214(3):173–184, 2000.

[101] Josh C Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[102] Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E Gusz Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015.

[103] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. springer, 2016.

[104] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[105] Josh Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239, 2011.

[106] David M Bryson and Charles Ofria. Understanding evolutionary potential in virtual cpu instruction set architectures. *PLoS One*, 8(12):e83242, 2013.

[107] Bérénice Batut, David P Parsons, Stephan Fischer, Guillaume Beslon, and Carole Knibbe. In silico experimental evolution: a tool to test evolutionary scenarios. In *BMC bioinformatics*, volume 14, pages 1–11. Springer, 2013.

[108] Jean-Marc Montanier and Nicolas Bredeche. Surviving the tragedy of commons: Emergence of altruism in a population of evolving autonomous agents. In *European Conference on Artificial Life*, 2011.

[109] Markus Waibel, Dario Floreano, and Laurent Keller. A quantitative test of hamilton's rule for the evolution of altruism. *PLoS Biol*, 9(5):e1000615, 2011.

[110] Steffen Wischmann, Dario Floreano, and Laurent Keller. Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *Proceedings of the National Academy of Sciences*, 109(3):864–868, 2012.

[111] Gregory S Hornby, Seiichi Takamura, Takashi Yamamoto, and Masahiro Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE transactions on Robotics*, 21(3):402–410, 2005.

[112] Dario Floreano, Phil Husbands, and Stefano Nolfi. Evolutionary robotics. Technical report, Springer Verlag, 2008.

[113] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[114] Sevan G Ficici, Richard A Watson, and Jordan B Pollack. Embodied evolution: A response to challenges in evolutionary robotics. In *Proceedings of the eighth European workshop on learning robots*, pages 14–22. Citeseer, 1999.

[115] Nicolas Bredeche, Jean-Marc Montanier, Wenguo Liu, and Alan FT Winfield. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129, 2012.

[116] Nicolas Bredeche. Embodied evolutionary robotics with large number of robots. In *Artificial Life Conference Proceedings 14*, pages 272–273. MIT Press, 2014.

[117] Francisco Bellas, Richard J Duro, Andrés Faiña, and Daniel Souto. Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. *IEEE Transactions on autonomous mental development*, 2(4):340–354, 2010.

[118] Chrisantha Thomas Fernando, Eors Szathmary, and Phil Husbands. Selectionist and evolutionary approaches to brain function: a critical appraisal. *Frontiers in computational neuroscience*, 6:24, 2012.

[119] Jens Krause, Graeme D Ruxton, and Stefan Krause. Swarm intelligence in animals and humans. *Trends in ecology & evolution*, 25(1):28–34, 2010.

[120] Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Guy Theraula, and Eric Bonabeau. *Self-organization in biological systems*. Princeton university press, 2003.

[121] Pierre-Paul Grassé. *Les Insectes et leur univers*. Palais de la Découverte, 1949.

[122] Plerre-P Grassé. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux*, 6(1):41–80, 1959.

[123] Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial life*, 5(2):173–202, 1999.

[124] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?*, pages 703–712. Springer, 1993.

[125] Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

[126] Bo Liu, Ling Wang, and Yi-Hui Jin. An effective pso-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):18–27, 2007.

[127] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Positive feedback as a search strategy. 1991.

[128] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[129] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[130] Krzysztof Socha, Michael Sampels, and Max Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Workshops on Applications of Evolutionary Computation*, pages 334–345. Springer, 2003.

[131] Christian Blum and Michael Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.

[132] Marc Reimann, Karl Doerner, and Richard F Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591, 2004.

[133] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[134] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[135] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *2007 IEEE swarm intelligence symposium*, pages 120–127. IEEE, 2007.

[136] Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part i: background and development. *Natural Computing*, 6(4):467–484, 2007.

[137] Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 7(1):109–124, 2008.

[138] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

[139] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[140] Jan Dyre Bjerknes and Alan FT Winfield. On fault tolerance and scalability of swarm robotic systems. In *Distributed autonomous robotic systems*, pages 431–444. Springer, 2013.

[141] Jean-Louis Deneubourg, Simon Goss, Nigel Franks, Ana Sendova-Franks, Claire Detrain, and Laeticia Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots. In *From animals to animats: proceedings*

of the first international conference on simulation of adaptive behavior, pages 356–365, 1991.

[142] Ralph Beckers, Owen E Holland, and Jean-Louis Deneubourg. Fom local actions to global tasks: Stigmergy and collective robotics. In *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, Volume 1, Volume 2 Prerational Intelligence: Interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems, Volume 3*, pages 1008–1022. Springer, 2000.

[143] Richard Vaughan, Neil Sumpter, Jane Henderson, Andy Frost, and Stephen Cameron. Experiments in automatic flock control. *Robotics and autonomous systems*, 31(1-2):109–117, 2000.

[144] Jyh-Ming Lien, Samuel Rodriguez, Jean-Phillipe Malric, and Nancy M Amato. Shepherding behaviors with multiple shepherds. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3402–3407. IEEE, 2005.

[145] Daniel Strömbom, Richard P Mann, Alan M Wilson, Stephen Hailes, A Jennifer Morton, David JT Sumpter, and Andrew J King. Solving the shepherding problem: heuristics for herding autonomous, interacting agents. *Journal of the royal society interface*, 11(100):20140719, 2014.

[146] Alyssa Pierson and Mac Schwager. Bio-inspired non-cooperative multi-robot herding. In *ICRA*, pages 1843–1849. Citeseer, 2015.

[147] Anil Özdemir, Melvin Gauci, and Roderich Groß. Shepherding with robots that do not compute. In *Artificial Life Conference Proceedings 14*, pages 332–339. MIT Press, 2017.

[148] Vito Trianni. *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots*, volume 108. Springer, 2008.

[149] Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[150] Marco Dorigo, Vito Trianni, Erol Şahin, Roderich Groß, Thomas H Labella, Gianluca Baldassarre, Stefano Nolfi, Jean-Louis Deneubourg, Francesco Mondada, Dario Floreano, et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2):223–245, 2004.

[151] Roderich Groß and Marco Dorigo. Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305, 2008.

[152] Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*, pages 37–56. Springer, 2011.

[153] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2):115–144, 2013.

[154] Heiko Hamann. Evolution of collective behaviors by minimizing surprise. In *Artificial Life Conference Proceedings 14*, pages 344–351. MIT Press, 2014.

[155] Tanja Katharina Kaiser and Heiko Hamann. Self-organized construction by minimal surprise. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 213–218. IEEE, 2019.

[156] Tanja Katharina Kaiser and Heiko Hamann. Engineered self-organization for resilient robot self-assembly with minimal surprise. *Robotics and Autonomous Systems*, 122:103293, 2019.

[157] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[158] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[159] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[160] Diederik P Kingma. Fast gradient-based inference with continuous latent variable models in auxiliary form. *arXiv preprint arXiv:1306.0733*, 2013.

[161] Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. *Boltzmann machines: Constraint satisfaction networks that learn.* Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

[162] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234, 2014.

[163] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[164] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

[165] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[166] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[167] Kunfeng Wang, Chao Gou, Yanjie Duan, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Generative adversarial networks: introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 4(4):588–598, 2017.

[168] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[169] Frans A Oliehoek, Rahul Savani, Jose Gallego-Posada, Elise Van der Pol, Edwin D De Jong, and Roderich Groß. Gangs: Generative adversarial network games. *arXiv preprint arXiv:1712.00679*, 2017.

[170] Dirk Helbing and Anders Johansson. Pedestrian, crowd, and evacuation dynamics. *arXiv preprint arXiv:1309.1609*, 2013.

[171] Alessandro Zonta, Selmar K Smit, Evert Haasdijk, and Agoston E Eiben. Modelling human movements with turing learning. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2254–2261, Bangalore, India, 2018. IEEE.

[172] Alan M Turing. Computing machinery and intelligence. *Creative Computing*, 6(1):44–53, 1980.

[173] Emily Baird, Marcus J Byrne, Jochen Smolka, Eric J Warrant, and Marie Dacke. The dung beetle dance: an orientation behaviour? *PLoS One*, 7(1):e30211, 2012.

[174] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[175] S. Magnenat, M. Waibel, and A. Beyeler. Enki: The fast 2D robot simulator, 2011. `https://github.com/enki-community/enki`.

[176] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[177] Thomas Hellström. *Kinematics equations for differential drive and articulated steering*. Department of Computing Science, Umeå University, 2011.

[178] Orazio Miglino, Henrik Hautop Lund, and Stefano Nolfi. Evolving mobile robots in simulated and real environments. *Artificial life*, 2(4):417–434, 1995.

[179] Nick Jakobi. *Minimal simulations for evolutionary robotics*. PhD thesis, University of Sussex, 1998.

[180] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[181] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9459–9468, Seoul, Korea, 2019. IEEE.

[182] Akshay Mehrotra and Ambedkar Dukkipati. Generative adversarial residual pairwise networks for one shot learning. *arXiv preprint arXiv:1703.08033*, 1(1):1–8, 2017.

[183] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(1):1–1, 2019.

[184] Josh C. Bongard and Hod Lipson. Nonlinear system identification using coevolution of models and tests. *Transactions on Evolutionary Computation*, 9(4):361–384, August 2005.

[185] Yu Sun and John M. Hollerbach. Active robot calibration algorithm. In *2008 IEEE International Conference on Robotics and Automation*, pages 1276–1281, CA, USA, May 2008. IEEE.

[186] Arpad E. Elo and Sam Sloan. *The rating of chessplayers, past and present*. Ishi Press, Japan, 2008.

[187] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2012.

[188] Melvin Gauci, Jianing Chen, Wei Li, Tony J Dodd, and Roderich Groß. Self-organized aggregation without computation. *The International Journal of Robotics Research*, 33(8):1145–1161, 2014.

[189] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, USA, 2016. MIT.

[190] Dan Zhang and Anna Khoreva. Progressive augmentation of gans. In *Advances in Neural Information Processing Systems*, pages 6246–6256, USA, 2019. MIT.

[191] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, MA,USA, 2017. MIT Press.

[192] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[193] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[194] Ronald L. Graham and Neil JA Sloane. Penny-packing and two-dimensional codes. *Discrete & Computational Geometry*, 5(1):1–11, 1990.

[195] Eliseo Ferrante, Ali Emre Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, and Marco Dorigo. Self-organized flocking with a mobile robot swarm: a novel motion control method. *Adaptive Behavior*, 20(6):460–477, 2012.