Lisbon University Institute

Department of Information Science and Technology

Hierarchical Evolution of Robotic Controllers for Complex Tasks

Miguel António Frade Duarte

A Dissertation presented in partial fulfillment of the Requirements for the Degree of

Master in Computer Science

Supervisor:

Prof. Dr. Anders Lyhne Christensen, assistant professor, ISCTE-IUL

Co-Supervisor:

Prof. Dr. Sancho Moura Oliveira, assistant professor, ISCTE-IUL

June 2012

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

Alan Turing

Resumo

A robótica evolucionária é uma metodologia que permite que robôs aprendam a efetuar uma tarefa através da afinação automática dos seus "cérebros" (controladores). Apesar do processo evolutivo ser das formas de aprendizagem mais radicais e abertas, a sua aplicação a tarefas de maior complexidade comportamental não é fácil. Visto que os controladores são habitualmente evoluídos através de simulação computacional, é incontornável que existam diferenças entre os sensores e atuadores reais e as suas versões simuladas. Estas diferenças impedem que os controladores evoluídos alcancem um desempenho em robôs reais equivalente ao da simulação.

Nesta dissertação propomos uma abordagem para ultrapassar tanto o problema da complexidade comportamental como o problema da transferência para a realidade. Mostramos como um controlador pode ser evoluído para uma tarefa complexa através da evolução hierárquica de comportamentos. Experimentamos também combinar técnicas evolucionárias com comportamentos pré-programados.

Demonstramos a nossa abordagem numa tarefa em que um robô tem que encontrar e salvar um colega. O robô começa numa sala com obstáculos e o colega está localizado num labirinto ligado à sala. Dividimos a tarefa de salvamento em diferentes sub-tarefas, evoluímos controladores para cada sub-tarefa, e combinamos os controladores resultantes através de evoluções adicionais. Testamos os controladores em simulação e comparamos o desempenho num robô real. O controlador alcançou uma taxa de sucesso superior a 90% tanto na simulação como na realidade.

As contribuições principais do nosso estudo são a introdução de uma metodologia inovadora para a evolução de controladores para tarefas complexas, bem como a sua demonstração num robô real.

Palavras-chave: Robótica Evolucionária, Redes Neuronais, Hierarquia, Controladores Robóticos.

Abstract

Evolutionary robotics is a methodology that allows for robots to learn how perform a task by automatically fine-tuning their "brain" (controller). Evolution is one of the most radical and open-ended forms of learning, but it has proven difficult for tasks where complex behavior is necessary (know as the *bootstrapping problem*). Controllers are usually evolved through computer simulation, and differences between real sensors and actuators and their simulated implementations are unavoidable. These differences prevent evolved controllers from *crossing the reality gap*, that is, achieving similar performance in real robotic hardware as they do in simulation.

In this dissertation, we propose an approach to overcome both the bootstrapping problem and the reality gap. We demonstrate how a controller can be evolved for a complex task through hierarchical evolution of behaviors. We further experiment with combining evolutionary techniques and preprogrammed behaviors.

We demonstrate our approach in a task in which a robot has to find and rescue a teammate. The robot starts in a room with obstacles and the teammate is located in a double T-maze connected to the room. We divide the rescue task into different sub-tasks, evolve controllers for each sub-task, and then combine the resulting controllers in a bottom-up fashion through additional evolutionary runs. The controller achieved a task completion rate of more than 90% both in simulation and on real robotic hardware.

The main contributions of our study are the introduction of a novel methodology for evolving controllers for complex tasks, and its demonstration on real robotic hardware.

Keywords: Evolutionary Robotics, Artificial Neural Networks, Hierarchy, Robotic Controllers.

Acknowledgements

I would like to thank my advisor Professor Anders Christensen for teaching me so much about research and evolutionary robotics, for inspiring my newfound passion for this research field, and for his brilliant insights that greatly improved the quality of this dissertation.

I would like to thank my co-advisor Professor Sancho Oliveira for his amazing dedication and constant support, both on the small, everyday problems, as well as on the big decisions.

To my family, especially my parents, António and Isabel, and my sister, Mariana, for their unconditional love, support, and understanding. They have always done the possible and the impossible for me, and for that I will forever be grateful.

To my friends, for their true friendship and encouragement.

Last, but certainly not least, I want to thank my girlfriend and muse, Margarida, for her love, for being there for me when I needed the most, and for being responsible for the best years of my life.

Contents

Resumo **v**

Abstract **vii**

Acknowledgements **ix**

List of Figures **xiii**

Abbreviations **xv**

1 Introduction **1**

1.1 Objectives 5

1.2 Scientific Contribution 6

1.3 Structure of the Dissertation 7

2 State of the Art **9**

2.1 Classic AI 9

2.2 Evolutionary Algorithms 12

2.3 Evolutionary Robotics 13

3 Hierarchical Composition of Controllers **21**

3.1 The Double T-Maze Task 22

3.2 Simulator and Robot 24

3.3 Methodology 26

3.4 Experiments and Results 28

3.4.1 Controller Architecture 29

3.4.1.1 Exit Room Sub-Task 29

3.4.1.2 Solve Double T-Maze Sub-Task 31

3.4.1.3 Return to Room Sub-Task 34

3.4.2 Evolving the Main Controller 36

3.4.3 Transfer to the Real Robot 39

3.5 Discussion 40

4 Combining Preprogrammed Behaviors and Evolved Behavioral Control **43**

4.1 Experimental Setup 45

4.2 Results 46
4.2.1 Transfer to Real Robotic Hardware 48
4.3 Discussion 49

5 Conclusions 51

Appendices 57

A E-puck Sensor Samples 57

B Turn Left Behavior Mazes 59

Bibliography 61

List of Figures

2.1 Shakey the robot 10
2.2 Artificial neural network model 14

3.1 The experimental environment 22
3.2 Double T-maze 23
3.3 The e-puck robot 24
3.4 Representation of the hierarchical controller 27
3.5 The experimental controller 30
3.6 Exit Room behavior primitive fitness graph 31
3.7 Training mazes for navigation behavior primitives 33
3.8 Solve double T-maze behavior arbitrator fitness graph 35
3.9 Return to Room behavior arbitrator fitness graph 36
3.10 Rescue behavior arbitrator fitness graph 38
3.11 Rescue behavior arbitrator output graph 39
3.12 Screenshots of the rescue task experiments 40

4.1 Behavior arbitrator with preprogrammed behaviors 44
4.2 Results from the preprogrammed behaviors experiment 47

A.1 E-puck sensor samples 57

B.1 Turn left behavior primitive mazes 59

Abbreviations

AI	**A**rtificial **I**ntelligence (see page 2)
ANN	**A**rtificial **N**eural **N**etwork (see page 3)
CTRNN	**C**ontinuous-**T**ime **R**ecurrent **N**eural **N**etwork (see page 15)
GOFAIR	**G**ood **O**ld **F**ashion **A**rtificial **I**ntelligence and **R**obotics (see page 9)
EA	**E**volutionary **A**lgorithms (see page 2)
ER	**E**volutionary **R**obotics (see page 2)

Chapter 1

Introduction

The research field of robotics has existed for over half a century. Throughout the decades, robots have been introduced as means of automation in manufacturing, replacing humans in dangerous tasks, as well as leading to a substantial reduction of costs and to mass production. Robots have the potential to become the next revolution by moving out of factories and into the real world. Several attempts have been made to bring the assistance of robots to our daily lives, but so far it has proven difficult to get robots to reliably perform tasks beyond vacuuming floors and cleaning pools. This challenge lies in finding a way to make robots perform tasks in our complex world.

Partially fueled by the futuristic visions of Isaac Asimov and other science fiction authors from the 1940's onwards, it was believed that humanoid robots would be an integral part of our life, and that their intelligence would surpass that of our own. Producing real artificial intelligence has, however, proved a challenging task. Humans sense an incredible amount of information. Our brains can select what data to process, generalize past experiences, and apply existing knowledge to new situations. Robots, on the other hand, are often very limited in terms of sensors and actuators, and although computing power has increased substantially over the years, we still do not know how to translate all that computational power into the kind of intelligence that humans possess. While the vision of intelligent humanoid robots has not (yet) come to reality, we do instead have many different specialized robots that are extremely good at particular tasks.

One of the most popular techniques for the design of robotic controllers consists of manually specifying every characteristic of the robot's behavior, usually in the form of a computer program. The behavior of a robot is the result of its interaction with the environment: it senses its surroundings, feeds that information to a control mechanism, and acts on its environment. It is, thus, a dynamic system, in which every action may have an influence on the subsequent ones. Manual and detailed specification of every characteristic of a robot's behavior may be possible for simple tasks and/or simple environments, but when any of these variables starts to increase in complexity, manual specification of a suitable behavior for all possible situations becomes infeasible.

To overcome the limitations of manual specification of robotic behavior, researchers have studied the application of artificial intelligence (AI) to the decision-making mechanisms of robots. Since the 1950's, many AI techniques have been applied to robotics. The first class of approaches was based on the belief that human intelligence could be reduced to symbol manipulation. As such, the robots had a simplistic internal model of the world that they would use to decide what actions to take. In the 1980's, Rodney Brooks introduced the notion of behavior-based robotics and reactive controllers. In his view, robots should not try to model the world around them, they should simply react to sensory stimulus. In the 1990's, the research field of evolutionary robotics (ER) emerged when researchers started combining robots and evolutionary algorithms (EA). EA are metaheuristic optimization algorithms inspired by biological evolution. By continuously selecting the best individuals and by applying genetic operators such as mutation and recombination, candidate solutions with the highest fitness reproduce and become the basis for the next generation. In this way, the evolutionary process should gradually produce increasingly better solutions.

ER techniques have the potential to automate the design of behavioral control without the need for manual and detailed specification of the desired behavior [10].

The robot repeats a particular task many times while the controller is adjusted automatically. This random variation potentially introduces new individuals that are better at solving the task than their predecessors. Artificial neural networks (ANN) are often used as controllers in ER because of their capacity to tolerate noise [19] such as that introduced by imperfections in sensors and actuators. Since artificial evolution is a process that requires many evaluations until a suitable controller is found, such an approach would be extremely time-consuming to carry out in real robots. As a consequence, computer simulation is widely used in ER. This approach, however, is not without its flaws. Two main issues have prevented ER from being widely used as an engineering tool for automatic design of behavioral control: bootstrapping (especially when complex tasks are considered), and the transfer of behavioral control from simulation to reality.

It has proven difficult for evolution to find controllers that are able to solve complex tasks. Since all the initial controllers fail at solving the task, they receive an equally low fitness. It is, thus, not trivial to define a fitness function that will allow a smooth progression through the fitness landscape towards the desired solution. This is known as the bootstrapping problem.

Numerous studies have demonstrated that it is possible to evolve robotic control systems capable of solving tasks in surprisingly simple and elegant ways [29]. To date, relatively simple tasks have been solved using ER techniques, such as obstacle avoidance, gait learning, phototaxis, and foraging [26]; but as Mouret and Doncieux write: "*. . . [the evolutionary process] hides many unsuccessful attempts to evolve complex behaviors by only rewarding the performance of the global behavior. The bootstrap problem is often viewed as the main cause of this difficulty, and consequently as one of the main challenges of evolutionary robotics: if the objective is so hard that all the individuals in the first generation perform equally poorly, evolution cannot start and no functioning controllers will be found*". The bootstrapping problem is the reason that there have been no reports of successful evolution of control systems for complex tasks.

Another problem lies in the use of evolutionary techniques for real robotic hardware. Simulations are simplified versions of reality and there may be differences in the sensors, in the actuators, and in the physics. This means that controllers may evolve to rely on aspects in the simulated world that are different or may not exist at all in the real world. The controllers may therefore fail to complete the task executed on real robotic hardware.

We present a novel approach to the evolution of behavioral control and study how to overcome bootstrap issues and how to allow for successful transfer of control evolved in simulation to real robotic hardware. We experiment with giving evolution access to previously learned behaviors and to preprogrammed behaviors that have been tested on real robotic hardware. Behavioral control for complex tasks may therefore be learned in an incremental and hierarchical manner where the successful transfer to real hardware will be ensured at each increment.

Several different incremental approaches have been studied as a means to overcome the bootstrapping problem and to enable the evolution of behaviors for complex tasks. In incremental evolution, the initial random population starts in a simple version of the environment to avoid bootstrapping issues. The complexity of the environment is then progressively increased as the population improves (see for instance [12, 6]). Alternatively, the goal task can be decomposed into a number of sub-tasks that are then learned in an incremental manner (see for instance [16, 8, 6]). While a single ANN controller is sometimes trained in each sub-task sequentially (such as in [6, 16]), different modules can also be trained to solve different sub-tasks (see [8] for an example). The approach presented in this dissertation falls in the latter category: we recursively decompose the goal task into sub-tasks and train different ANN-based controllers to solve the sub-tasks. The controllers for the sub-tasks are then combined though an additional evolutionary step into a single controller for the goal task.

We use a task in which a robot must rescue a teammate. Our rescue task requires several behaviors typically associated with ER [26] such as exploration, obstacle avoidance, memory, delayed response, and the capacity to navigate safely through corridors: (i) an e-puck robot must first find its way out of a room with obstacles, (ii) the robot must then solve a double T-maze [4] in which two light flashes in the beginning of the maze instruct the robot on the location of the teammate, and finally (iii) the robot must guide its teammate safely to the room. We evolve behaviors in simulation and evaluate their performance on a real robot. We also experiment with giving evolution access to preprogrammed behaviors. While there are several studies on incremental evolution of behavioral control for autonomous robots, the study presented in this dissertation is novel in four respects: (i) sub-tasks are solved by one or more sub-controllers, that are either preprogrammed behaviors or continuous time recurrent neural networks that are evolved independently, (ii) we introduce the concept of derived fitness functions during composition for sequential tasks, (iii) we give evolution access to preprogrammed behaviors, and (iv) we demonstrate a fully evolved controller solving a complex task on real robotic hardware.

1.1 Objectives

The main objective of the proposed research is to evolve robotic controllers for complex tasks that can be successfully transferred to real robotic hardware. This is done by hierarchically composing evolved controllers: once behavioral control to solve a given task has been evolved (or preprogrammed), that behavior becomes a previously-learned behavior, which can be used by evolution in future increments. Behavioral control for a complex goal task may thus be learned in an incremental manner if it can be divided into sub-tasks that can be learned independently. The transfer from simulation to real robotic hardware can be conducted in an incremental manner as behavior primitives and sub-controllers are evolved. This allows the designer to address issues related to transferability immediately and locally in the controller hierarchy.

In summary, the objectives are:

- to evolve robotic controllers for complex tasks;

- to ensure the correct transfer of the evolved controllers to real robotic hardware.

A key objective of this dissertation is to show that the proposed approach works in real robotic hardware. For real robot experiments, we use the open source e-puck robotic platform [25].

1.2 Scientific Contribution

This dissertation presents the following contributions:

- reviews existing approaches to overcoming the complexity and reality gap problems;
- introduces a novel methodology for evolving and transferring controllers for complex tasks;
- demonstrates the successful application of the methodology both in simulation-based experiments and on real robotic hardware;
- extends and improves on the open source simulation platform and neuroevolution framework JBotEvolver.

The work conducted in this dissertation has resulted in four publications:

- M. Duarte, A. L. Christensen, S. Oliveira (2011), “Towards Artificial Evolution of Complex Behavior Observed in Insect Colonies”. Proceedings of the Portuguese Conference on Artificial Intelligence, 2011, Lisbon. Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, Germany, pages 153-167.
- M. Duarte, S. Oliveira, A. L. Christensen (2012), “Automatic synthesis of controllers for real robots based on preprogramed behaviors”. Proceedings

of the 12th International Conference on Adaptive Behavior, 2012, Odense. Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, Germany, pages 249-258.

- M. Duarte, S. Oliveira, A. L. Christensen (2012), "Structured Composition of Evolved Robotic Controllers." Proceedings of the 5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems, 2012, Montpellier, in press.
- M. Duarte, S. Oliveira, A. L. Christensen (2012), "Hierarchical Evolution of Robotic Controllers for Complex Tasks." Proceedings of the IEEE Conference on Development and Learning, and Epigenetic Robotics, 2012, San Diego. IEEE Press, Piscataway, NJ, in press.

1.3 Structure of the Dissertation

In Chapter 2, we review some of the milestones in robotics research and provide a more in-depth view over the field of evolutionary robotics. In Chapter 3, we present our hierarchical methodology, discuss the details of our experimental setup, and analyze the obtained results. In Chapter 4, we experiment with giving evolution access to preprogrammed behaviors for fine sensory-motor tasks. Finally, in Chapter 5, we discuss the results and different ways in which this methodology can be further studied.

Chapter 2

State of the Art

While robots are common in manufacturing and industrial environments, it has proven challenging to create robotic controllers that allow robots to perform tasks in our complex world. In this chapter, we review the various approaches that have been studied for the problem of creating robotic controllers. In Section 2.1, we start with a broad historical view of the field of robotics and artificial intelligence. In Section 2.2, we focus on the evolutionary techniques used in our experiments. Finally, in Section 2.3, we discuss the main challenges faced by researchers in the field of evolutionary robotics, what solutions have been proposed, and how our approach differs.

2.1 Classic AI

Research in artificial intelligence and robotics took its first steps in the 1940's and 1950's with an emphasis on approaches that are now sometimes referred to as GOFAIR, Good Old Fashion Artificial Intelligence and Robotics. The approaches are based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols, an assumption defined as the "physical symbol systems hypothesis" by Allen Newell and Herbert Simon [27]. One of the most famous works that emerged from this line of thought was Shakey the robot [28] (see Figure 2.1), the first general-purpose mobile robot to be able to reason about its own actions. Shakey was a logical, goal-based agent, and it experienced a limited world. Its world model was composed of a number of rooms connected by corridors, with doors, light switches and objects available for the robot to interact with. Shakey's actions involved traveling from one location to another, pushing movable objects around, opening and closing the doors, climbing up and down from rigid objects, and turning the light switches on and off.

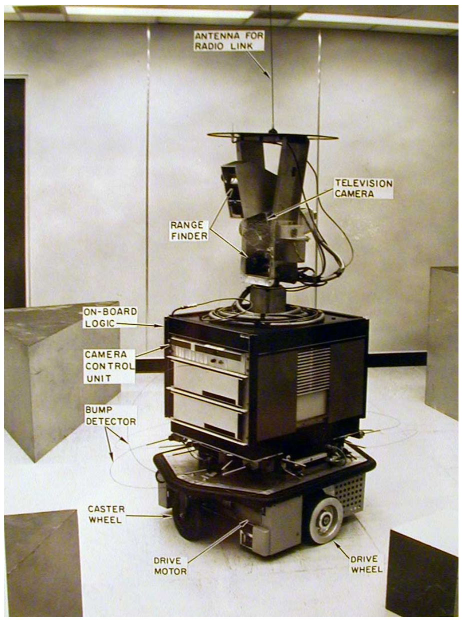


FIGURE 2.1: Shakey the robot (adapted from [13]).

GOFAIR remained the dominant paradigm until the introduction of behavior-based robotics in the 1980's. Many researchers started to doubt that high-level symbol manipulation alone could produce intelligent behavior and moved to behavior-based robotics. The concept of behavior-based robotics, which was introduced by Rodney Brooks, rejects GOFAIR's assumption that a robot should have an internal representation of the world and is based upon the idea of providing the robot with a collection of simple basic behaviors [1]. These behaviors are programmed into the robot's controller and provide a mapping between inputs and outputs. The robot senses the surrounding environment through sensors, such as proximity sensors, cameras and microphones, and takes action by using actuators, such as wheels, grippers or speakers. The interaction between the robot and the environment determines its global behavior. The controller of the robot is responsible for deciding which behavior is more suitable at a particular situation, and can be composed of competitive or cooperative methods. If the controller uses a competitive method, the various behaviors compete with one another in an attempt to find the most suitable action for a particular moment. If the controller uses a cooperative method, several different behaviors can contribute to the action performed by the robot. The decision is based on the sensor readings, the current action of the robot and, in some cases, an internal state. Behavior-based controllers are usually designed by a trial and error process, in which the designer updates existing behaviors or creates new ones in an iterative approach. The designer accomplishes the breakdown of the desired behavior into simple behaviors based on intuition. After making a change, the designer has to permit the robot to act within the environment to observe the robot's performance.

Rodney Brooks, one of the pioneers in the field of behavior-based robotics, created the subsumption architecture in the 1980's [5].The subsumption architecture is characterized by the decomposition of complex behavior into many simple behavior modules, which are in turn organized into layers. The layers are stacked and ordered in terms of behavioral priority and represent the goals of the agent: the behaviors in the bottom layers stand for simpler goals, such as obstacle avoidance, and have a higher priority than the ones on the higher layers, which are increasingly abstract. Layers are built on top of previous ones. They are allowed to examine and inject data into the internal interfaces of previous layers. Such a system allows an experimenter to have a functional controller from early stages that can easily be extended by adding higher layers. Since high-level layers can only interfere with the robot's behavior by actively suppressing the outputs of the lower level layers, the global controller will still produce a sensible result even if the top layers fail to produce results in time. The solution proposed by Brooks has to be carefully hand designed.

By using behavior-based methodologies, roboticists were successful at designing and implementing robotic behaviors for real hardware, but the manual programming of the controllers proved a bottleneck both in terms of behavioral complexity and in terms of development effort needed. In order to circumvent this problem, researchers have experimented with evolutionary computation as a means of automatic controller design.

2.2 Evolutionary Algorithms

Evolution is an optimization process that can often lead, as Darwin wrote, to "*organs of extreme perfection*" [7]. By randomly varying the individuals and applying the principles of natural selection using computer simulation, it is possible to shape the behavior of a population over many generations. EA are a search heuristic that generally rely on a population of contending individuals to solve a particular problem. An individual in a population usually corresponds to a candidate solution to a given problem. The individuals compete for survival and for the right to reproduce, which generates a new population. This concept is borrowed from nature, in which the survival of the fittest is a key component for the evolution of different species. Thus, the idea of iterated variation and selection that is common to evolutionary processes is modeled in an algorithm and used to iteratively improve the quality of solutions.

An EA are typically characterized by five elements: (i) representation for the solution, (ii) variation operators, (iii) fitness criterion, (iv) selection method, and (v) initialization. The representation for the solution is the genetic description of each individual. Like the genome in living organisms, the chosen representation describes the entirety of an individual's hereditary information. In order to diversify the genome pool of the population, a variation operator is applied to a selected subsection of the existing individuals. These operations have also been inspired from biology, with mutation and cross-over of the simulated genomes. It is necessary to choose which individuals should survive and populate the next generation. This is done by using a fitness criteria (or performance index). The designer of the experiment should decide which goals should be accomplished, and a numerical value is attributed to each genome after testing, indicating how well it performed. To populate the next generation, several selection methods can be used. One example is taking the individuals with the highest fitness score of the current generation and applying the variation operators to populate the next generation. The initialization of an EA can be completely at random, or it can incorporate human or other expertise about solutions that may perform better than others.

The use of EA can be traced back to the 1950's and 1960's. The first experiments were conducted on John von Neumann's computer in Princeton and originated what is now called *artificial life*. One of the first programs to use EA was developed in 1953 by Nils Barricelli [2]. It featured an environment composed by cells, forming a grid. Numbers resided in the cells and could migrate to neighboring cells based on a predefined set of rules. If there were a collision between two numbers on the same cell, they would compete for survival. Barricelli found that even with very simple rules for propagating throughout the environment, certain numeric patterns would evolve. Some of these patterns could only persist in the presence of other patterns.

2.3 Evolutionary Robotics

Evolutionary robotics emerged as a field in the beginning of the 1990's [30]. It is distinguished from other fields of research by the use of EA in the synthesis of controllers for robots. Numerous studies followed which demonstrated robots with evolved control systems solving basic tasks in surprisingly simple and elegant ways. However, only relatively simple tasks have been solved using evolutionary robotics such as obstacle avoidance, gait learning, phototaxis, foraging, and other searching tasks [26].

The use of ANNs as controllers is widespread in evolutionary robotics. An ANN is a mathematical or computational model inspired by the structure and functional aspects of biological neural networks. It is composed by an interconnected group of artificial neurons and processes information using a connectionist approach to computation (see Figure 2.2). An ANN is typically defined by three types of parameters: (i) the interconnection pattern between different layers of neurons, (ii) the activation function that converts a neuron's weighted input to its output activation, and (iii) the method for updating the parameters of the network.

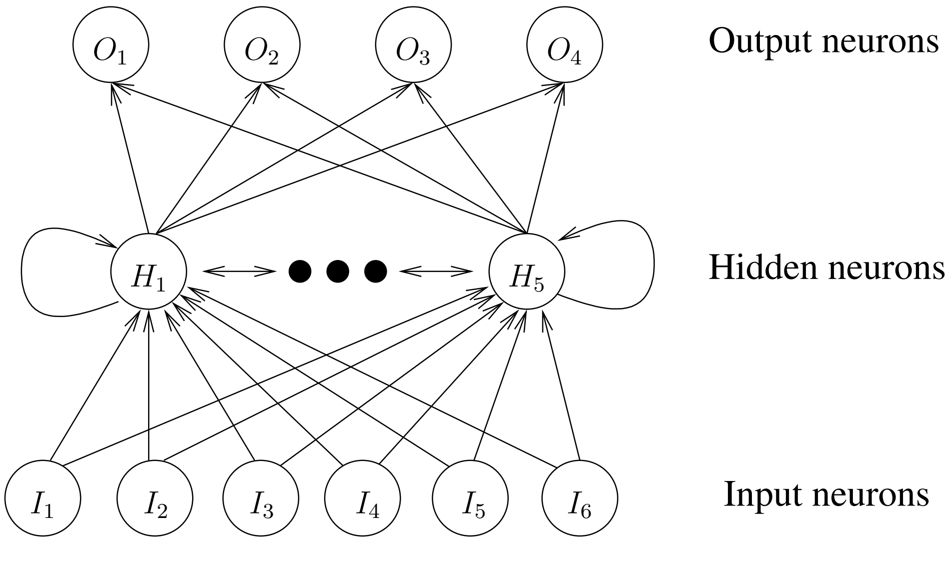


FIGURE 2.2: An example of an artificial neural network. This particular example has a hidden layer with fully connected neurons.

By using an evolutionary approach, the parameters of the neural network (such as the interconnection weights, the number of neurons or their activation threshold function) are changed from one generation to the next. This allows for the self-organization of the controller, in contrast with the traditional approaches of behavior-based robotics in which the designer had to program the robots' behaviors by hand. There are many justifications for the use of artificial neural networks in robotics [29]. Neural networks offer a continuous, smooth search space, which means that gradual changes of the parameters that define it will often correspond to gradual changes of its behavior. They provide various levels of evolutionary granularity, allowing the designer to decide where to apply artificial evolution: on the lowest level specification of the networks, such as the connection strengths, or to higher levels, such as the coordination of predefined modules composed of predefined sub-networks. The straightforward mapping between sensors and motors make ANNs especially interesting for robots. They can accommodate analog input signals and provide either continuous or discrete motor outputs, depending on the transfer function chosen. Robustness to noise is another factor. Since their units are based upon a sum of several weighted signals, oscillations in the individual values of these signals usually do not have a drastic effect on the behavior of the robot, which is a very useful property for physical robots with noisy sensors. It has also been claimed that "*Neural networks can be a biologically plausible metaphor of mechanisms that support adaptive behavior*" [29].

Continuous-time recurrent neural networks (CTRNN) [3] are a type of artificial neural networks that are composed by an input layer, used to feed robot sensor data into the network, a fully interconnected hidden layer, and an output layer, which is connected to the actuators of the robot. The hidden layer is able to provide the network with short-term memory: since the neurons of the hidden layer are interconnected, they can maintain state from one computation cycle to the next, which can be useful for a robot interacting in a dynamic environment. In [33], Tuci et al. demonstrate how a Khepera robot using a CTRNN can make decisions influenced by the passage of time. In their experiments, a light source and a circular band surrounding it composed the environment. The objective of the task was for the robot to get close to the light source while avoiding to cross the band. The robot had to distinguish between two versions of the environment: in the first version, an opening in the circular band allowed the robot to cross over to the light source, but in the second version, the band completely surrounded the objective. Their results showed that the robot was able to differentiate between the setups by "feeling" how much time it had spent circling around the band [33]. Duarte et al. [9] experimented with the evolution of insect-like behavior in robots using a CTRNN as the controller. The task consisted of a foraging scenario, in which several robots had to forage prey. After finding a prey, the robots needed to carry it to a central nest, spending energy in the process. In order to regain energy and avoid death, the robots had to rest in the nest before returning to the foraging task. The authors were able to evolve colonies that displayed multiple complex macroscopic behaviors observed in insect colonies, such as task allocation, synchronization, and communication.

Soon after the research into evolutionary robotics began, two main challenges became clear, namely, (i) that the number of evaluations required meant that simulation had to be used extensively, and (ii) that it often is non-trivial to ensure successful transfer of behavior evolved in simulation to real robots (known as the *reality gap*). In [23], three complementary approaches to cross the reality gap were proposed: "*(a) an accurate model of a particular robot-environment dynamics can be built by sampling the real world through the sensors and the actuators of the robot; (b) the performance gap between the obtained behaviors in simulated and real environments may be significantly reduced by introducing a 'conservative' form of noise; (c) if a decrease in performance is observed when the system is transferred to a real environment, successful and robust results can be obtained by continuing the evolutionary process in the real environment for a few generations.*" The use of samples from real robots and a conservative form of noise has become widespread. In 1997, Jakobi [18] even advocated the use of minimal simulations in which the simulator would only implement the specific features of the real world that the experimenter deemed necessary for a robot to complete its task. In order to use the minimal simulation approach, however, the experimenter needs to have a priori knowledge of all the relevant features that a robot will encounter during task-execution.

In order to address the bootstrapping problem, several approaches of incremental evolution of robotic controllers have been proposed. The approaches fall into three different categories: (i) incremental evolution where controllers are evolved with a fitness function that is gradually increased in complexity; (ii) goal task decomposition in which a single ANN is trained sequentially on different sub-tasks; (iii) goal task decomposition in which hierarchical controllers are composed of different sub-controllers evolved for different sub-tasks along with one or more arbitrators that delegate control.

A methodology belonging to the first category, namely in which controllers are evolved with a fitness function that is gradually increased in complexity, was proposed by Gomez and Miikkulainen [12]. They used a prey-capture task for their study. First, a simple behavior was evolved to solve a simplified version of the global task, in which the prey does not move. Gradually, by repeatedly increasing the prey's speed, they evolved a more general and complex behavior that was able to solve the prey-capture task. The controllers that they obtained through the incremental approach were more efficient and displayed a more general behavior than controllers evolved non-incrementally. They also found that the incremental approach helped to bootstrap evolution.

Harvey et al. [16] proposed an approach that falls in the second category, namely where a single ANN is trained sequentially on different sub-tasks. The authors describe how they evolved a controller to robustly perform simple visually guided tasks. They incrementally evolved the controller starting with a "big target", then a "small target", and finally to a "moving target". The controller was evolved in few generations and it performed well on real robotic hardware. Christensen and Dorigo [6] compared two different incremental evolutionary approaches, to evolve a controller for a swarm of connected robots that had to perform phototaxis while avoiding holes. They found no benefits in using neither an incremental approach where the controllers were trained on different sub-tasks sequentially nor an incremental increase in environmental complexity over a non-incremental approach for their highly integrated task.

There are several examples of studies on incremental evolution that fall in the third category, namely in which the global controller is composed of different sub-controllers that have been trained on different sub-tasks. Moioli et al. used a homeostatic-inspired GasNet to control a robot [24]. They used two different sub-controllers, one for obstacle avoidance and one for phototaxis, that were inhibited or activated by the production and secretion of virtual hormones. The authors evolved a controller that was able to select the appropriate sub-controller depending on internal stimulus and external stimulus.

Lee [22] proposed an approach in which different sub-behaviors were evolved for different sub-tasks and then combined hierarchically through genetic programming. Behavior arbitrators would decide when each sub-behavior was active. The approach was studied in a task where a robot had to search for a box in an arena and then push it towards a light source. By evolving different reactive sub-behaviors such as “circle box”, “push box” and “explore”, the author managed to synthesize a robotic controller that solved the task. The author claims that his controllers were transferable to a real robot, but only some of the sub-controllers were tested on real hardware. Larsen et al. [21] extended Lee’s work by using reactive neural networks for the sub-controllers and for the arbitrators instead of evolved programs. However, the chosen goal task used by both Lee and Larsen is relatively simple and the scalability of their respective approaches to more complex tasks was never tested.

In this dissertation, we propose and study a novel and structured approach to the engineering of control systems in which we hierarchically divide the controller into simpler sub-controllers. Each sub-controller is responsible for solving part of the task. Our approach shares similarities with Lee’s [22] and Larsen et al.’s [21] approaches in that controllers are evolved and composed hierarchically based on task decomposition. However, as we demonstrate in this dissertation, our approach scales to complex tasks because (i) we use non-reactive controllers, and (ii) during the composition of sub-controllers into larger and more complex controllers, the fitness function for the composed task can be derived directly from the decomposition. We also demonstrate transfer of behavioral control from simulation to real robotic hardware without a significant loss of performance, and we discuss the benefits of transferring controllers incrementally.

We furthermore experiment with mixing evolution with preprogrammed controllers, which can be helpful for tasks in which fine sensory-motor behavior is required. In previous studies, there have been reports of ad-hoc use of artificially evolved controllers for partial behaviors in otherwise preprogrammed control systems: in [14], for instance, a neural network was handed the control whenever a robot needed to grasp another robot, but the control was otherwise preprogrammed. The approach studied in this dissertation is different. We allow for a structured integration of learned and preprogrammed behavior in a hierarchical and incremental manner.

Chapter 3

Hierarchical Composition of Controllers

The purpose of our research is to evolve robotic controllers for complex tasks, and to ensure their transferability to real robotic hardware. While it is possible to evolve robotic controllers for simple, well-defined tasks, increasing task complexity proves to be a challenge both in terms of bootstrapping and in terms of ensuring transferability of the evolved controllers to real robotic hardware. Our approach consists of dividing the task into simpler sub-tasks and evolving simpler controllers to solve each sub-task. The controllers are then composed hierarchically through additional evolutionary runs.

For our experiments, we use a task in which a robot must rescue a teammate. Our rescue task requires several behaviors typically associated with ER [26] such as exploration, obstacle avoidance, memory, delayed response, and the capacity to navigate safely through corridors: (i) a robot must first find its way out of a room with obstacles, (ii) the robot must then solve a double T-maze [4], and finally (iii) the robot must guide its teammate safely to the room. We evolve controllers in simulation and evaluate their performance on a real robot.

The environment is composed of a room, in which the robot starts, and a double T-maze (see Figure 3.1). A number of obstacles are located in the room. The room

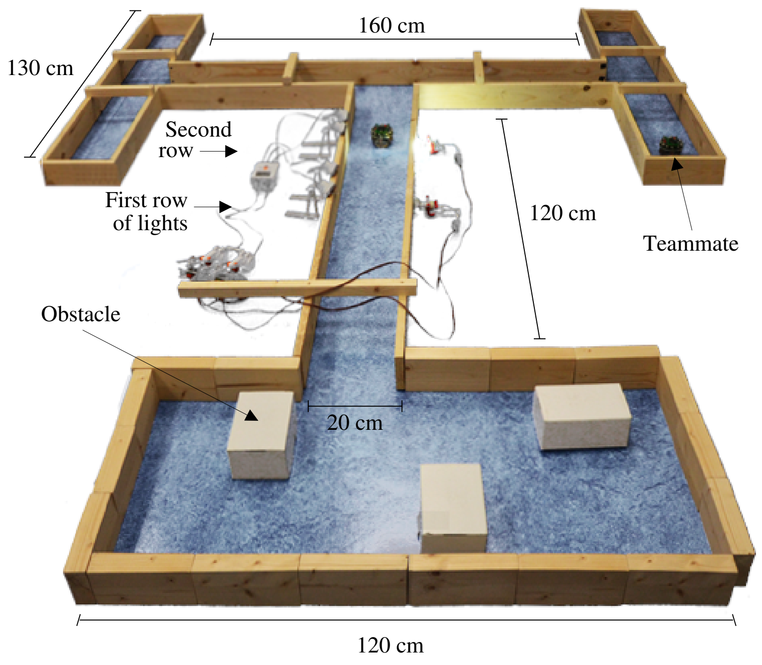


FIGURE 3.1: The environment is composed of a room with obstacles and a double T-maze. The room is rectangular and its size can vary between 100 cm and 120 cm. The double T-maze has a total size of 200 cm × 200 cm. The two rows with the lights are located in the central maze corridor. The activation of these two rows of lights indicates the location of a teammate.

has a single exit that leads to the start of a double T-maze. In order to find its teammate, the robot should exit the room and navigate to the correct branch of the maze. Two rows of flashing lights in the main corridor of the maze give the robot information regarding the location of the teammate. Upon navigating to the correct branch of the maze, the robot must guide its teammate back to the room.

3.1 The Double T-Maze Task

We use a double T-maze task [4] as part of our experiment. An example of a double T-maze can be seen in Fig. 3.2. The T-maze contains three T-junctions. At the start of each experiment, the robot is placed in the "Start zone" and must navigate towards the first junction. On its way, it passes two rows of lights. In each row, one of the lights is activated. The activated light flashes as the robot passes by. The activated light in the first row informs the robot on to which side it must turn in the first T-junction it encounters, while the activated light in the second row informs the robot to which side it must turn in the second T-junction that it encounters. If L1 and R2 are activated, for instance, the robot must make a left turn in the first T-junction and a right turn in the second T-junction so that it reaches exit LR (see Fig. 3.2), and so on.

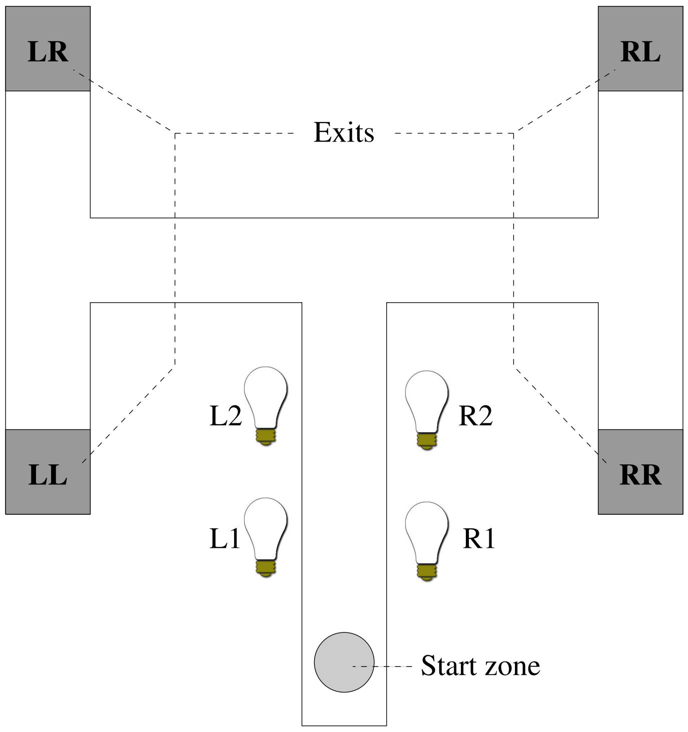


FIGURE 3.2: A double T-maze. A robot is placed in the start zone and must navigate to one of the four exits depending on which lights flash as the robot passes by.

Variations of the T-maze task have been used extensively in studies of learning and motivation in animals, neuroscience, and robotics (see [31, 32, 18] for examples). In robotics, T-mazes have been used to study different neural network models such as diffusing gas networks [17], the online learning capability of continuous time recurrent neural networks [4], and the evolution of transferable controllers [18, 20]. However, in the studies where controllers were tested on real hardware, only a single T-maze was used and the mazes were relatively small with respect to the robot. In our experiment, the robot must first find the exit of a room before it is able to solve the double T-maze.

3.2 Simulator and Robot

We use JBotEvolver for offline evolution of behavioral control. JBotEvolver is an open source, multirobot simulation platform, and neuroevolution framework. The simulator is written in Java and implements 2D differential drive kinematics. Evaluations of controllers can be distributed across multiple computers and different evolutionary runs can be conducted in parallel. The simulator can be downloaded from: `http://sourceforge.net/projects/jbotevolver`.

For our real robot experiments, we used an e-puck [25]. The e-puck is a small circular (diameter of 75 mm) differential drive mobile robotic platform designed for educational use (see Figure 3.3). The e-puck's set of actuators is composed of two wheels, that enable the robot to move at speeds of up to 13 cm/s, a loudspeaker, and a ring of 8 LEDs which can be switched on/off individually. The e-puck is equipped with several sensors: (i) 8 infrared proximity sensors which are able to detect nearby obstacles and changes in light conditions, (ii) 3 microphones (one positioned on each side of the robot, and one towards the front), (iii) a color camera, and (iv) a 3D accelerometer. Additionally, our e-puck robots are equipped with a range & bearing board [15] which allows them to communicate with one another.

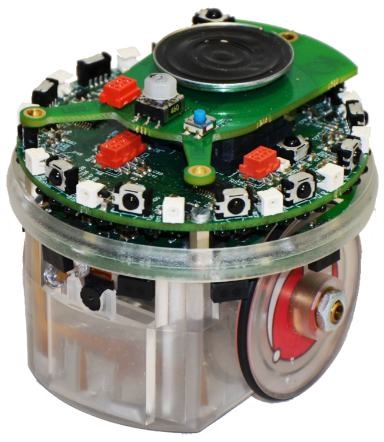


FIGURE 3.3: The e-puck is a differential drive robot with a diameter of 75 mm and is equipped with a variety of sensors and actuators, such as a color camera, infrared proximity sensors, a loudspeaker, 3 microphones, and two wheels. Our e-pucks are also equipped with a range & bearing board that allows for inter-robot communication.

We use four of the e-puck's eight infrared proximity sensors: the two front sensors and the two lateral sensors. We collected samples (as advocated in [23]) from the sensors on a real e-puck robot in order to model them in JBotEvolver. Each sensor was sampled for 10 seconds (at a rate of 10 samples/second) at distances to the maze wall ranging from 0 cm to 12 cm. We collected samples at increments of 0.5 cm for distances between 0 cm and 2 cm, and at increments of 1 cm for distances between 2 cm and 12 cm. The sampled sensor readings can be seen in Appendix A.

To model the infrared sensors in the simulation, we used a ray-casting technique: a certain number of rays are cast from the sensor at different angles, from $-\frac{\alpha}{2}$ to $\frac{\alpha}{2}$, where α is the sensor's opening angle. Based on experimental data from the robot, we used an α value of 90° and a total of 7 rays per sensor. The distances at which each ray detected an obstacle are averaged and a lookup table is used to estimate what the real value of the sensor would be. Afterwards, noise is added to the sensor readings. The amount of noise is based on the standard deviation of the real sensor readings. The sensor readings are converted back to a distance, based on a fixed lookup table, which is composed by the average values of all sampled sensors at each distance. The distance is then normalized to the maximum distance (12 cm) in order to be fed to the network with a value between 0 and 1. We furthermore added a 5% offset noise to the sensor's value.

The e-puck's proximity infrared sensors can also measure the level of ambient light. In this study, we use ambient light readings from the two lateral proximity sensors to detect light flashes in the double T-maze sub-task. When a light flash is detected, the activation of one of the two dedicated neurons is set to 1 depending on the side from which the light flash is detected. The input neuron stays active with a value of 1 for 15 simulation cycles (equivalent to 1.5 seconds) to indicate that a flash has been detected. We also included a boolean "near robot" sensor that lets the robot know if there is any other robot within 15 cm. For this sensor, we use readings from the range & bearing board. In simulation, we added Gaussian noise (5%) to the wheel speeds in each control cycle. The robot's speed was limited to 10 cm/s.

If the control code does not fit within the e-puck's limited memory (8 kB), it is necessary to run the control code off-board. When the control code is executed off-board, the e-puck starts each control cycle by transmitting its sensory readings to a workstation via Bluetooth. The workstation then executes the controller, and sends back the output of the controller (wheel speeds) to the robot. We use off-board execution of control code in the real robot experiments conducted in this chapter, and on-board execution of control code in the real robot experiments conducted in Chapter 4.

For this dissertation, the simulator was extended to include several new features:

- the e-puck's infrared sensors were modeled and added to the library of sensors;
- a hierarchical controller model was implemented to accommodate our methodology;
- preprogrammed behaviors can now be added to a controller as a behavior primitive;
- it is possible to remotely control an e-puck robot via Bluetooth (support for different robots can be easily added);
- several extensions to the simulator's graphical user interface have been developed to facilitate the analysis of results.

3.3 Methodology

The controller has a hierarchical architecture and it is composed of several ANNs (see Figure 3.4). Each network is either a *behavior arbitrator* or a *behavior primitive*. These terms were used in [22] to denote similar controller components. A behavior primitive network is usually at the bottom of the controller hierarchy and directly controls the actuators of the robot, such as the wheels. If it is relatively easy to find an appropriate fitness function for a given task, a behavior primitive (a single ANN) is evolved to solve the task. An appropriate fitness function is one that (i) allows evolution to bootstrap, (ii) evolves a controller that is able to solve the task consistently and efficiently, and (iii) evolves a controllers that transfers well to real robotic hardware. In case an appropriate fitness function cannot be found for a task, the task is recursively divided into sub-tasks until appropriate fitness functions have been found for each sub-task.

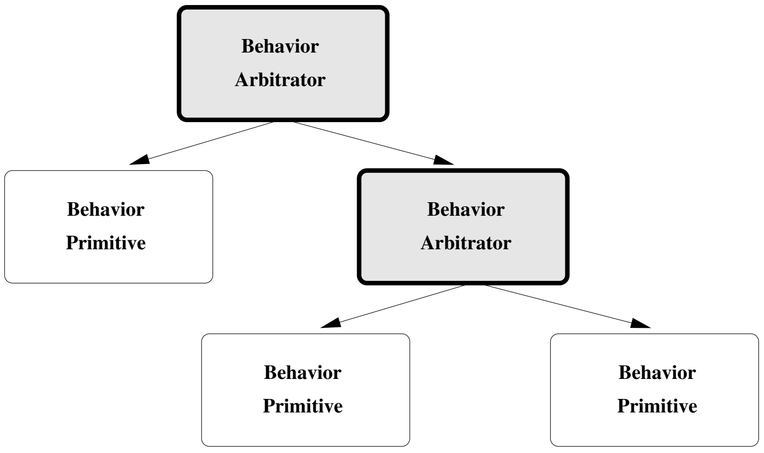


FIGURE 3.4: A representation of the hierarchical controller. A behavior arbitrator network delegates the control of the robot to one or more of its sub-controllers. A behavior primitive network can control the actuators of the robots directly.

Controllers evolved for sub-tasks are combined through the evolution of a behavior arbitrator. A behavior arbitrator receives either all or a subset of the robot's sensory inputs, and it is responsible for delegating control to one or more of its sub-controllers. Each behavior arbitrator can have a different *sub-controller activator*. The sub-controller activator activates one or more sub-controllers based on the outputs of the ANN in the behavior arbitrator. The behavior arbitrators used in this study have one output neuron for each of their immediate sub-controllers. The sub-controller activator we use activates the sub-controller for which the corresponding output neuron of the arbitrator has the highest activation. The state of a sub-controller is reset whenever it gets deactivated. Alternative sub-controller activators could be used, such as activators that allow for multiple sub-controllers to be active at the same time, or activators that do not change the state of their sub-controllers. Parallel activation of different sub-controllers could, for instance, allow a robot to communicate at the same time as it executes motor behaviors.

If the fitness function for the evolution of a behavior arbitrator is difficult to define, it can be derived based on the task decomposition. The derived fitness function is constructed to reward the arbitrator for activating a sub-controller that is suitable for the current sub-task, rather than for solving the global task. The use of derived fitness functions in the composition step circumvents the otherwise increase in fitness function complexity as the tasks considered become increasingly complex.

The topology of each network in the hierarchy (such as the number of input neurons, the number of hidden neurons, and the number output neurons) is completely independent from one another. The basic behavior primitives are evolved first. The behavior primitive are then combined though the evolution of a behavior arbitrator. The resulting controller can then be combined with other controllers through additional evolutionary steps to create a hierarchy of increasingly more complex behavioral control. Each time a new sub-controller (either a behavior primitive or a composed controller) has been evolved, its performance on real robotic hardware can be evaluated. The experimenter can thus address issues related transferability incrementally as the control system is being synthesized.

3.4 Experiments and Results

In our experiments, a robot must rescue a teammate that is located in a particular branch of a maze. The robot must find the teammate and guide it to safety. To test the controller on real robotic hardware, we built a double T-maze [4] with a size of 200 cm $\times$ 200 cm (see Figure 3.1). In the real maze, a Lego Mindstorms NXT brick controlled the flashing lights. The brick was connected to four ultrasonic sensors that detected when the robot passed by. Lights were turned on by the 1st and 3rd ultrasonic sensor and turned off by the 2nd and 4th ultrasonic sensor. The brick controlled the state of the lights using two motors.

3.4.1 Controller Architecture

The rescue task is relatively complex, especially given the limited amount of sensory information available to the robot, and it would be difficult to find an appropriate fitness function that allows evolution to bootstrap. We therefore divided the rescue task into three sub-tasks: (i) exit the room, (ii) solve the double T-maze to find the teammate, and (iii) return to the room, guiding the teammate. Several controllers are evolved to solve each of the sub-tasks. A hierarchical controller is then given access to each of the previously learned controllers and evolved to complete the full rescue task. The structure of the controller for the complete rescue task can be seen in Figure 3.5.

For each evolutionary run, we used a simple generational evolutionary algorithm with a population size of 100 genomes. The fitness score of each genome was averaged over 50 samples with varying initial conditions, such as the robot's starting position and orientation. After the fitness of all genomes had been sampled, the 5 highest scoring individuals were copied to the next generation. 19 copies of each genome were made and for each gene there was a 10% chance that a Gaussian offset with a mean of 0 and a standard deviation of 1 was applied. All the ANNs in the behavior primitives and in the behavior arbitrators were time-continuous recurrent neural networks [3] with one hidden layer of fully connected neurons. A controller's genome encoded both the weights of the network and the decay constants of the neurons.

3.4.1.1 Exit Room Sub-Task

The first part of the rescue task was an exploration and obstacle avoidance task in which the robot must find a narrow exit leading to the maze. The room was rectangular with a size that varied between 100 cm and 120 cm. We placed either 2 or 3 obstacles in the room depending on its size. Each obstacle was rectangular with side lengths ranging randomly from 5 cm to 20 cm. The location of the room exit was also randomized in each trial.

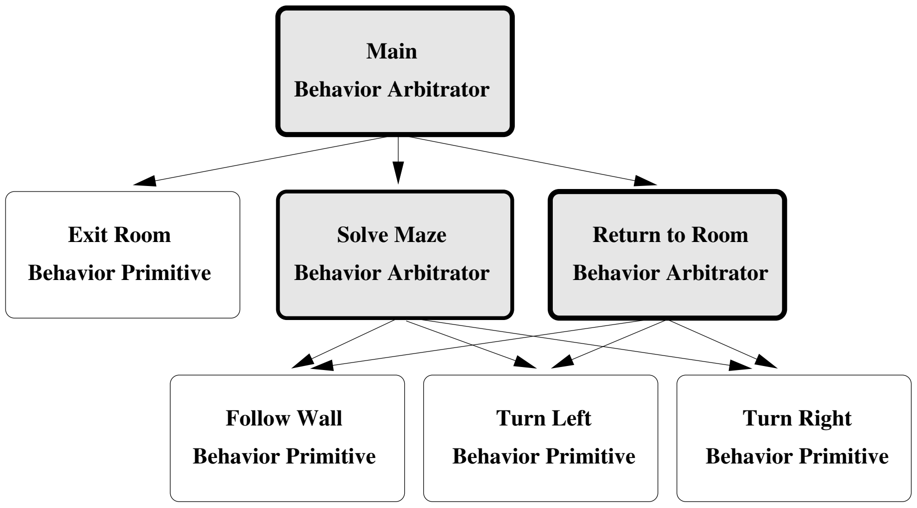


FIGURE 3.5: The controller used in our experiments is composed of 3 behavior arbitrators and 4 behavior primitives.

We found that an ANN with 4 input neurons, 10 hidden neurons, and 2 output neurons (a genome with a total of 182 floating point alleles) could solve the task. Each of the input neurons was connected to an infrared proximity sensor, and the output neurons controlled the speed of the robot's wheels. The robot was randomly oriented and positioned near the center of the room at the beginning of each sample and the it was evaluated differently if it succeeded or failed to find the exit of the room within the allotted time (100 seconds), according to f_1:

$$f_1 = \begin{cases} 5 + \frac{C-c}{C} & \text{if exit was found} \\ \frac{D-d}{D} & \text{if exit was not found} \end{cases} \tag{3.1}$$

where C is the maximum number of cycles (100 seconds $\times$ 10 cycles/second $=$ 1000 cycles), c is the number of cycles spent, D is the distance from the center of the room to its exit, and d is the closest point to the exit that the robot reached. Since the value for finding the exit varies between 5.0 and 6.0 (depending on how long it takes to complete the task), the expected maximum fitness of a controller should be 5.5.

The "exit room" controllers were evolved until the 500th generation and each sample was evaluated for 1000 control cycles, in a total of 10 evolutionary runs.

Afterwards, we conducted a post-evaluation of the best controllers of each evolutionary run in a total of 100 samples each. The controllers achieved an average solve rate of 52%, with a solve rate of 96% in the best evolutionary run. The best performing controller starts by moving away from the center of the room until it senses a wall, which it then follows clockwise until the room exit is found. 3 of the 10 evolutionary runs produced consistent results, finding the exit of the room in over 90% of the samples. The remaining runs did not produce successful behaviors: the robots would spin/circle around, sometimes finding the exit by chance and often crashing into one of the walls or into an obstacle. The fitness graph for the best controller at each generation and the average values of all 10 evolutionary runs for the exit room sub-task can be seen in Figure 3.6.

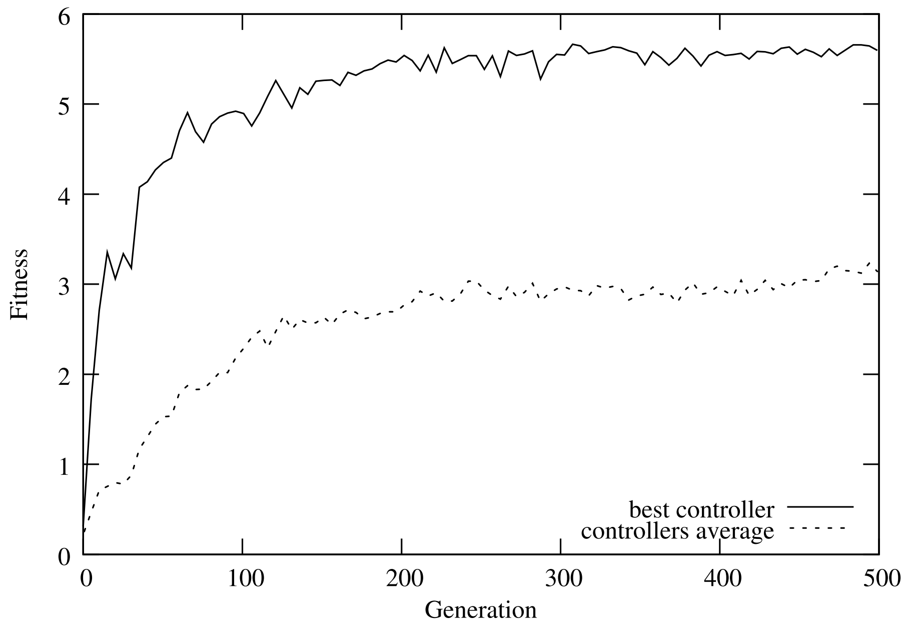


FIGURE 3.6: Fitness graph with the average fitness values of each of the highest scoring controllers of all 10 evolutionary runs, and the fitness values from the best controller at each generation for the exit room sub-task.

3.4.1.2 Solve Double T-Maze Sub-Task

In the second sub-task, the robot had to solve a double T-maze in order to find the teammate that had to be rescued. The robot was evaluated according to f_2:

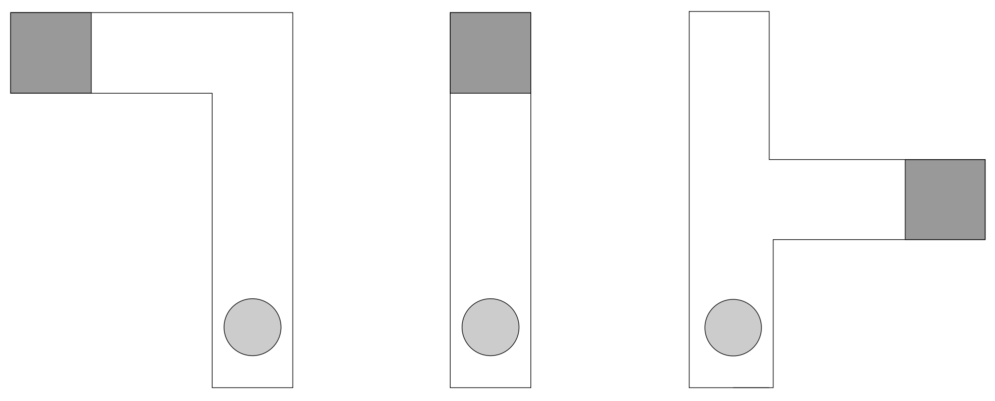
$$f_2 = \begin{cases} 1 + \frac{C-c}{C} & \text{if navigated to destination} \\ \frac{D-d}{3D} & \text{if crashed or chose wrong path} \\ 0 & \text{if time expires} \end{cases} \tag{3.2}$$

where C is the maximum number of cycles, c is the spent number of cycles, D is the total distance from the start of the maze to the the robot's destination, and d is the final distance from the robot to its destination. The maximum allotted time was 1000 cycles (equivalent to 100 seconds).

We experimented with using a single ANN to solve this sub-task. The ANN was composed of 6 input neurons, 10 hidden neurons, and 2 output neurons (a genome with a total of 202 floating point alleles). The input neurons were connected to the 4 proximity sensors and the 2 light sensors. The output neurons directly controlled the speed of the wheels.

We conducted 10 evolutionary runs, each lasting 1000 generations. The controllers were post-evaluated and the fitness of every controller was sampled 100 times for each of the 4 possible light configurations. The evolved controllers had an average solve rate of only 40%. The best controller had a solve rate of 83%, with just 3 other controllers were able to correctly solve the T-maze in more than 50% of the samples.

Since we could not obtain controllers that could solve the task consistently, we followed our methodology and further divided the solve maze sub-task into three different sub-tasks: "follow wall", "turn left" and "turn right", for which appropriate fitness functions could easily be specified. The behavior primitive network for each of these three sub-tasks had 4 input neurons, 3 hidden neurons, and 2 output neurons (a genome with a total of 35 floating point alleles). The input neurons were connected to the infrared proximity sensors and the outputs controlled the speed of the wheels. The three behavior primitives were evolved in corridors of various lengths. The environment for the "turn" controllers was also composed of either left or right turns, depending on the controller (see Figure 3.7). We used 9



(a) Left turn (b) Corridor (c) Right turn

FIGURE 3.7: Examples of some of the mazes used to evolve the behavior primitives “follow wall”, “turn left” and “turn right”. The filled circles represent the start zone and the filled squares represent the destination.

different mazes for the “turn left” and “turn right” controllers (see Appendix B), and 1 maze composed of a single corridor for the “follow wall” controllers.

A total of 10 evolutionary runs were simulated for each of the basic behaviors (“follow wall”, “turn left” and “turn right”). The evolutionary process lasted 100 generations, and the best controller from each evolutionary run were then sampled 100 times in order to evaluate the controller’s solve rate. The best “turn left” controllers from each evolutionary run achieved an average solve rate of 59%, with a solve rate of 100% for the controller that obtained the highest fitness; the “turn right” controllers achieved an average solve rate of 64%, with a solve rate of 100% for the controller that obtained the highest fitness; and the “follow wall” controllers achieved an average solve rate of 98%, with a solve rate of 100% for the controller that obtained the highest fitness. The best controllers for the basic behaviors achieved a performance of 100% in relatively few generations and the majority of the evolutionary runs converged towards the optimal solve rate. The “turn” controllers from some evolutionary runs did not generalize their solution to the 9 different types of mazes that we used. This brought down the average solve rate for these controllers to lower levels (59% in the turn left controllers and 64% in the turn right controllers).

We then evolved a behavior arbitrator with the three best behavior primitives as sub-controllers. The behavior arbitrator network had 6 input neurons, 10 hidden neurons, and 3 output neurons (a genome with a total of 213 floating point alleles). The inputs were connected to the 4 infrared proximity sensors and the 2 light sensors. A sub-controller activator chooses between the correct behavior primitives based on the activation levels of the output neurons. At the beginning of each trial, the robot was placed at the start of the double T-maze and had to navigate to the correct branch based on the activations of the lights that were placed on the first corridor (see Figure 3.1). For instance, if the left light of the first row and the right light of the second row were activated, the robot should turn left at the first junction and right at the second junction. The fitness awarded was based on f_2 and the sample was terminated if the robot collided into a wall or if it navigated to a wrong branch of the maze.

The evolution process lasted until the 1000th generation, in a total of 10 evolutionary runs. After conducting the post-evaluation, the controllers achieved an average solve rate of 93%, with a solve rate of 99.5% for the highest performing controller. The fitness graph for the best controller at each generation and the average values of all 10 evolutionary runs for the solve maze sub-task can be seen in Figure 3.8. The subdivision of the controller resulted in a significantly better solve rate (Mann-Whitney U, $p < 0.01$) when compared with the single ANN approach.

3.4.1.3 Return to Room Sub-Task

The final sub-task consisted of the robot returning to the initial room, guiding its teammate. For this sub-task, we reused the behavior primitives previously evolved for maze navigation (“follow wall”, “turn left” and “turn right”) and we evolved a new behavior arbitrator. The behavior arbitrator network was trained in the double T-maze with the robot starting in one of the four branches of the maze (chosen at random in the beginning of each trial). The behavior arbitrator had 4 input neurons, 10 hidden neurons, and 3 output neurons (a genome with a total of 193 floating point alleles). The input neurons were connected to the

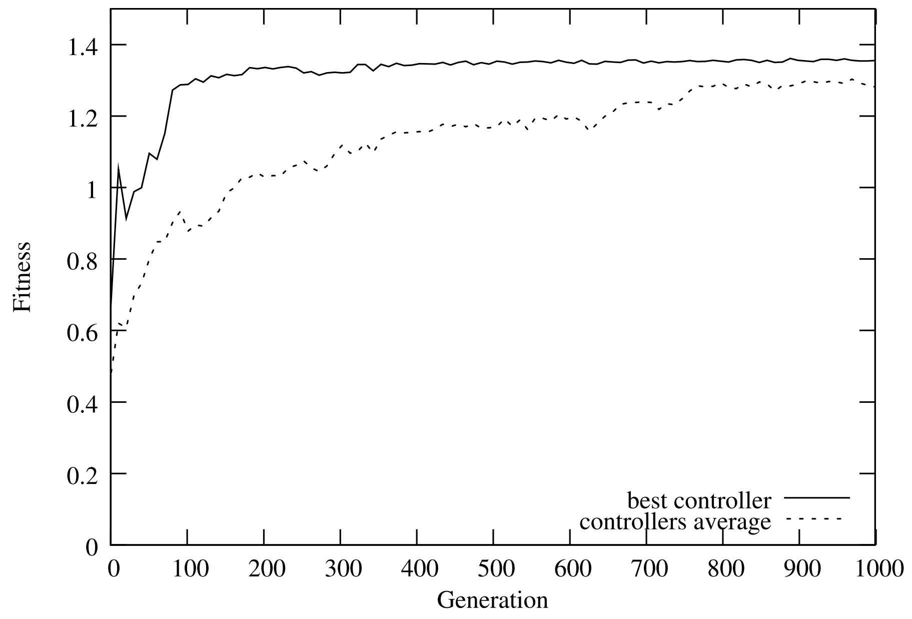


FIGURE 3.8: Fitness graph with the average fitness values of each of the highest scoring controllers of all 10 evolutionary runs, and the fitness values from the best controller at each generation for the solve double T-maze sub-task.

robot's infrared proximity sensors. A sub-controller activator used the activation level of the 3 output neurons to decide which behavior primitive should be active at any given moment.

The teammate being rescued continuously emitted a signal while waiting for the main robot. We used the e-puck range & bearing extension board to determine the distance between the two robots. When the distance between the robots was less than 15 cm, the "near robot" sensor's reading was set to 1. Since this was a task in which the robot had to navigate correctly through the maze, we used the same fitness function, f_2, as in the solve double T-maze sub-task described in the previous section. The only difference was the objective: the robot was evaluated based on its distance to initial room, not the distance to the teammate.

We conducted a total of 10 evolutionary runs until the 500th generation for the "return to room" behavior. The controllers achieved an average solve rate of 90%, with a solve rate of 99% for the highest performing controller. 8 of the 10 controllers converged to the optimal solution within 150 generations. The graph of the best controller's fitness at each generation and the average fitness of all 10 controllers can be seen in Figure 3.9.

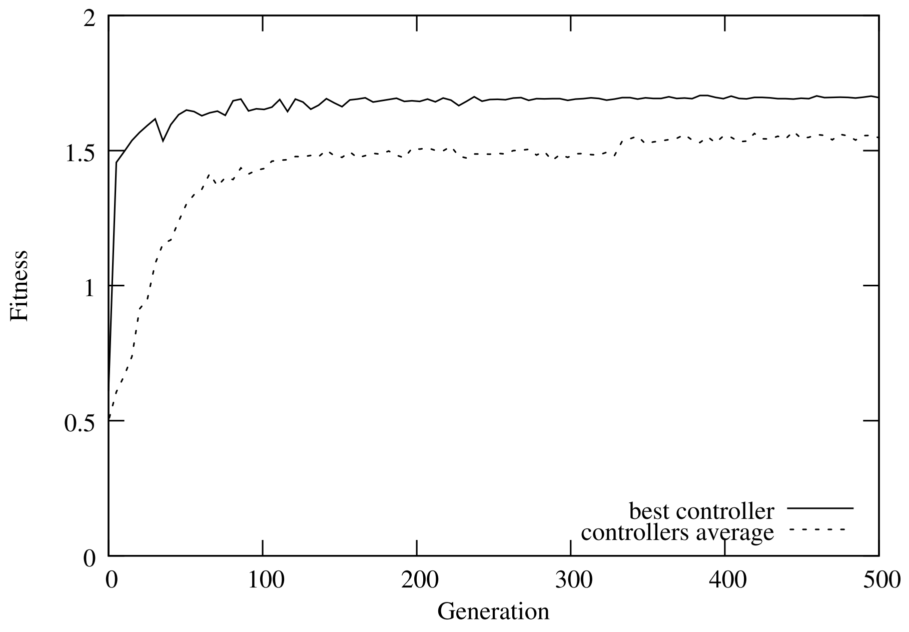


FIGURE 3.9: Fitness graph with the average fitness values of each of the highest scoring controllers of all 10 evolutionary runs, and the fitness values from the best controller at each generation for the return to room sub-task.

3.4.2 Evolving the Main Controller

For the composed task, we evolved a behavior arbitrator with the best controllers for the exit room, the solve maze, and the return to room sub-tasks as sub-controllers. The sub-controller activator decided which of the sub-controllers was active at any given time.

The robot had to first find the entrance to the double T-maze, then navigate the maze in order to find its teammate, and finally return to the room, guide its teammate. The behavior arbitrator for the complete rescue task had 5 input neurons, 10 hidden neurons, and 3 output neurons (a genome with a total of 193 floating point alleles). The inputs were connected to the 4 infrared proximity sensors and to a boolean "near robot" sensor, which indicated if there was a teammate within 15 cm (based on readings from the range & bearing board).

We evolved the controller with a derived fitness function, f_3, that rewards the selection of the right behaviors for the current sub-task. The controller was awarded a fitness value between 0 and 1 for each sub-task (for a maximum of 3 for all sub-tasks), depending on the amount of time that it selected the correct behavior. f_3 is defined as follows:

$$f_3 = \begin{cases} \displaystyle\sum_{s=1}^{N} \frac{t_s}{T_s} + \frac{C-c}{C} & \text{if the task was completed} \\ \\ \displaystyle\sum_{s=1}^{N} \frac{t_s}{T_s} & \text{if the task was not completed} \end{cases} \tag{3.3}$$

where the sum is over all the sub-tasks (in this study, $N = 3$ subtasks), T_s is the number of simulation cycles that the controller has spent in sub-task s, t_s is the number of cycles in which the controller chose the correct sub-controller for sub-task s, C is the maximum number of cycles, and c is the spent number of cycles.

We ran 10 evolutionary runs until the 1000th generation for the rescue task. The fitness of each genome was sampled 20 times and the average fitness was computed. Each sample lasted a maximum of 2000 control cycles (equivalent to 200 seconds). The 10 resulting controllers achieved an average solve rate for the composed task of 85%, after a post-evaluation with 400 samples. 6 controllers achieved the maximum fitness in under 150 generations, with 80 being the lowest number of generations (see Figure 3.10). The 4 remaining controllers achieved the maximum fitness between 250 and 600 generations. The fitness graph for the best controller at each generation and the average values of all 10 evolutionary runs for the rescue task can be seen in Figure 3.10.

We analyzed how the main controller managed to solve each part of the composed task. On the "exit room" task, all 10 controllers averaged a solve rate of 91%. All the controllers successfully learned that they should activate the exit room behavior primitive in the first part of the composed task.

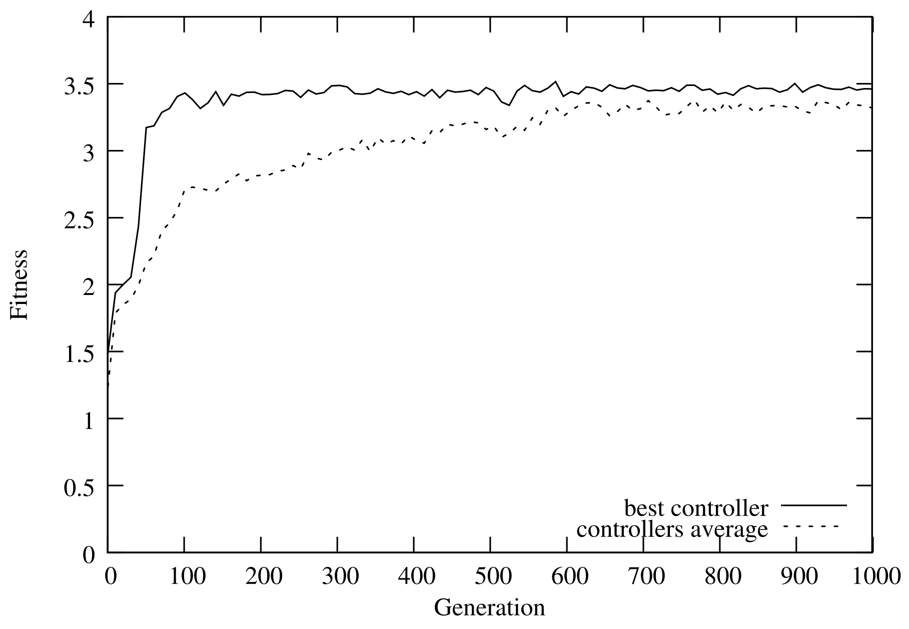


FIGURE 3.10: Fitness graph with the average fitness values of each of the highest scoring controllers of all 10 evolutionary runs, and the fitness values from the best controller at each generation for the rescue task.

After exiting the room, the controller should activate the "solve maze" behavior in order to find the robot's teammate. An important detail is that once the controller selects this behavior, it should not switch to another one until it reaches the end of the maze: switching resets the state of the selected sub-controller, meaning that the "solve maze" behavior arbitrator would forget which light flashes it previously sensed. The average solve rate dropped from 91% to 88%, which means that 3% of all the samples failed at solving the maze sub-task.

Upon finding the teammate, the robot should return to the initial room, completing the rescue task. This should be done by activating the return behavior at the end of the maze. The 10 controllers achieved an average solve rate of 85%, with a solve rate of 93% for the highest performing controller. An example of the decision process of the best performing controller can be seen in Figure 3.11. The analyzed controller distinguished the transition between each sub-task and activated the output neuron of the correct behavior, while suppressing the remaining output neurons.

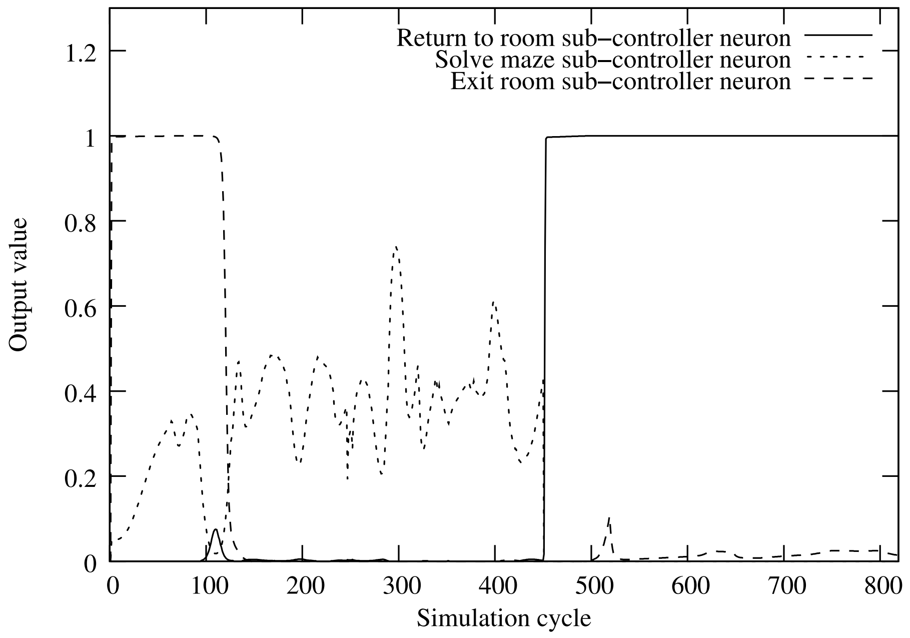
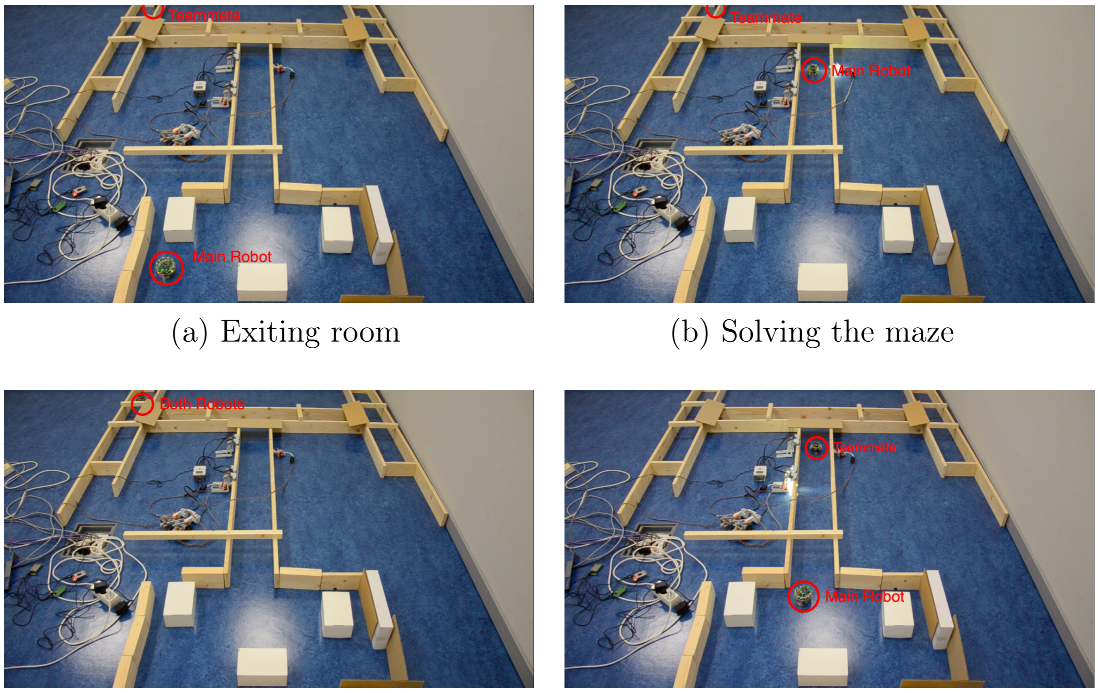


FIGURE 3.11: Values from the output neurons of the rescue behavior arbitrator. The output with the highest activation value determines which sub-controller is active. It is possible to see the clear distinction in the chosen sub-controller by the behavior arbitrator as it moves from one sub-task to the next.

3.4.3 Transfer to the Real Robot

After evaluating all the different evolutionary runs, the best performing controller from the simulation was tested on a real e-puck. The robot had to solve the composed task: find the exit of the initial room, navigate the double T-maze to the correct branch, and return to the room. We used a room with a size of 120 cm $\times$ 100 cm for our real robot experiments. Three identical obstacles with side lengths of 17.5 cm and 11 cm were placed in the room as shown in Figure 3.1. We sampled the controllers 6 times for each light combination, for a total of 24 samples. Since the purpose of these experiments was to test the transferability of the evolved controller, the teammate was not used and the near-robot sensor was remotely triggered if the robot reached the correct maze branch.

The controller solved the composed task on the real robot in 22 out of 24 samples (a solve rate of 92%). It consistently chose the correct sub-network at each point of the task, and only failed in the return to room behavior twice.



(a) Exiting room (b) Solving the maze
(c) Finding the teammate (d) Returning to the room

FIGURE 3.12: A series of screenshots a real robot experiment in which the teammate robot follows the main robot back to the initial room.

We ran additional proof-of-concept experiments in which we include a teammate that was preprogrammed to follow the main robot back to the initial room. Videos of these experiments can be found in `http://miguelduarte.pt/media/msc_thesis.html`. A series of screenshots from one of the videos can be seen in Figure 3.12.

3.5 Discussion

In this experiment, we demonstrated how controllers can be composed in a hierarchical fashion to allow for the evolution of behavioral control for a complex task.

We started by decomposing the goal task into sub-tasks until a controller for each sub-task could easily be evolved. We showed that previously learned sub-controllers can be reused in different parts of the hierarchical controller. After we had obtained controllers for each of the three sub-tasks, exit room, solve maze, and return to room, we combined them in an additional evolutionary step. When we combined the sub-controllers, we used a derived fitness function that rewarded controllers for activating the sub-controller corresponding to the current sub-task rather than for solving the global task.

For the main behavior arbitrator we used a fitness function directly derived from the immediate decomposition, that is, a fitness function that rewards a controller for activating an appropriate sub-controller given the current situational context. During evolution, an arbitrator (an ANN) was rewarded for (i) activating the exit room sub-controller while the robot was in the room, (ii) the solve sub-controllers while the robot was in the maze, and (iii) the return to room behavior after the teammate had been located. In this way, we avoid that the complexity of the fitness function increases with the task complexity as sub-behaviors are combined.

Our approach overcomes a number of fundamental issues in evolutionary robots. Often the experimenter has to go through a tedious trial and error process in order to design a suitable fitness function for the task at hand. In our approach, we recursively divide tasks into sub-tasks until a simple fitness function can easily be specified.

The transfer of behavioral control from simulation to a real robot is usually a hit or miss because a controller for the goal task is completely evolved in simulation before it is tested on real hardware. In our approach, the transfer from simulation to real robotic hardware can be conducted in an incremental manner as behavior primitives and sub-controllers are evolved. This allows the designer to address issues related to transferability immediately and locally in the controller hierarchy.

The applicability of our approach depends on if the task for which a controller is sought can be broken down into reasonably independent sub-tasks. For highly integrated tasks where it is unclear if or how the goal task can be divided into sub-tasks [6], our approach may not be directly applicable. However, in cases where a controller for an indivisible sub-task cannot be evolved, either because a good fitness function cannot be found or because evolved solutions do not transfer well, the evolved control may be combined with preprogrammed behaviors. Such preprogrammed controllers could be fine-tuned to work correctly both on the simulation and on real robotic hardware. They would also be beneficial for tasks where fine sensory-motor behaviors are required, since such behaviors might be difficult to evolve and transfer. We further explore this idea in the next chapter.

Chapter 4

Combining Preprogrammed Behaviors and Evolved Behavioral Control

Evolution of complex, transferrable behavior may not be feasible for some tasks if only evolved controllers are considered. Behavioral control that requires fine sensory-motor coordination has proven challenging to transfer successfully from simulation to real robots using evolutionary techniques. There have been examples in literature where researchers have combined evolved control and preprogrammed control, but it has been done in an ad-hoc manner. In [14], for instance, a neural network was handed the control whenever a robot needed to grasp another robot, but the control was otherwise preprogrammed. Our approach, on the other hand, allows for a structured integration of learned and preprogrammed behavior in a hierarchical and incremental manner.

Combining evolved and preprogrammed control could be used in integrated tasks for which bootstrapping and decomposition are difficult, or in tasks that are difficult to simulate with sufficient accuracy for the evolved controller to transfer. In this chapter, we experiment with the evolution of control systems that can take advantage of preprogrammed behavior primitives. Using the double T-maze task, the chosen primitives are simple preprogrammed behaviors (*follow wall*, *turn left*, and *turn right*). While the double T-maze task does not require fine sensory-motor coordination apart from correctly navigating a maze without touching its walls, this experiment allows us to test how evolution can be combined with preprogrammed behaviors.

The controller is composed of a behavior arbitrator network that can choose between several preprogrammed behavior primitives (see Figure 4.1). Each output neuron of the neural network corresponds to a single behavior primitive. The primitive which has the highest activation value is executed in a winner-takes-all approach. Some primitives can take more than one control cycle to complete, such as turning 90° left or right. The sub-controller activator does not execute any other behavior primitive before the previously selected behavior primitive has completed.

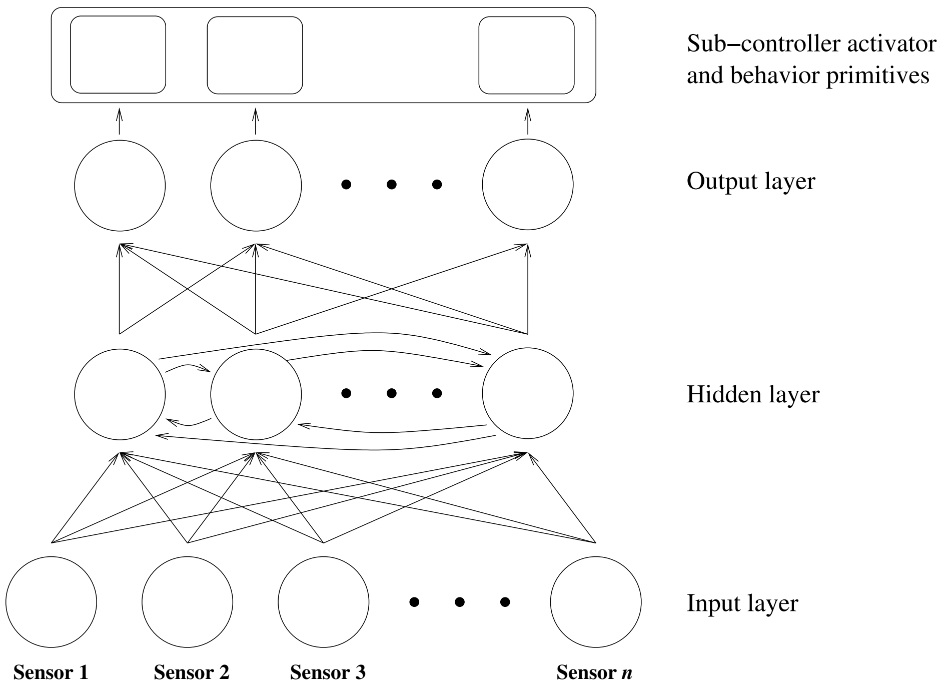


FIGURE 4.1: Example of the behavior arbitrator with preprogrammed behavior primitives: a continuous-time recurrent neural network [3] receives readings from the robot's sensors. The activation of the neurons in the output layer are fed to the *sub-controller activator*, which executes one of the behavior primitives based on the activations.

4.1 Experimental Setup

For our experiments, we used a double T-maze task. The experimental setup is in many ways identical to the one previously described in Section 3.1. The neural controller used in this experiment was a continuous-time recurrent neural network [3]. The input layer of the ANN was composed of 6 neurons: one for each of the four infrared proximity sensors, and one for each of the two light sensors. The readings from the proximity sensors are mapped to distances and then converted to input neuron activations (interval $[0, 1]$). When a light flash was detected, the corresponding input neuron was assigned an activation value of 1.0 for a duration of 15 control cycles.

The hidden layer of the ANN was composed of 10 fully connected neurons. The output layer of the neural network was composed of 3 neurons, one for each of the 3 preprogrammed behavior primitives available to the network: *turn left*, *turn right*, and *follow wall*. The genome of the ANN was encoded with a total of 213 floating point alleles. The sub-controller activator compares the activations of the three output neurons and executes the behavior that corresponded to the neuron with the highest activation. The two *turn* behaviors turn the robot 90°, which takes on average 40 control cycles. During that time, the sub-controller activator ignores the values of the output neurons in order to allow the turn to complete before executing a new behavior. The *follow wall* behavior moves the robot forward along the closest perceived wall. We limited the speed of the robot to 10 cm/s.

We evolved controllers with a simple generational evolutionary algorithm. Each generation was composed of 100 genomes, and each genome corresponded to an ANN with the topology described above. The fitness of a genome was sampled 40 times and the average fitness was computed. Each sample lasted a maximum of 1000 control cycles (equivalent to 100 seconds of simulated time). The starting position of the robot was varied up to 5 cm to the left or to the right, and up to 10 cm forward or backward.

The top 5 genomes were selected to populate the next generation using an elitist approach. An offspring was created using mutation: for each gene there was a 10% chance that a Gaussian offset with a mean of 0 and a standard deviation of 1 was applied. The 95 mutated offspring and the original 5 genomes constituted the next generation. The robot was evaluated according to f:

$$f = \begin{cases} 1 + \frac{C-c}{C} & \text{if navigated to destination} \\ \frac{D-d}{3D} & \text{if crashed or chose wrong path} \\ 0 & \text{if time expires} \end{cases} \tag{4.1}$$

where C is the maximum number of cycles, c is the number of cycles spent, D is the total distance from the start of the maze to the the robot's destination, and d is the final distance from the robot to its destination. The maximum allotted time was 1000 cycles (equivalent to 100 seconds).

We ran an additional set of experiments in a traditional ER setup in which the outputs of the neural network controlled the robot's wheels directly. Aside from the difference in the interpretation of the networks output, the experimental setup (network topology, inputs, simulation conditions, and evolutionary parameters) were the same as those described above.

4.2 Results

In order to experiment how preprogrammed behaviors can be mixed with evolutionary techniques, we synthesized robotic controllers for a double T-maze task. In this experiment, the values of the output neurons of the neural network activated one of the three possible preprogrammed behaviors: follow wall, turn left 90° and turn right 90°.

We conducted 30 evolutionary runs, each lasting 1000 generations. We conducted a post-evaluation of the evolved controllers in which the fitness of every controller was sampled 100 times for each of the 4 possible light configurations.

In experimental setup A, the behavior arbitrator had access to preprogrammed behavior primitives, and in experimental setup B, the controller was composed of a single ANN. The results are summarized in Figure 4.2.

In experimental setup A, the evolved controllers had an average solve rate of 87%. A solve rate of over 95% was observed in 12 of the 30 controllers. Some of the trials evolved controllers with good solutions as early as the 150th generation. The solutions produced in different evolutionary runs were similar. The robots learned how to navigate the T-maze correctly, but one of the controllers was not able to use the information from the light flashes to consistently make the correct decisions at the T-junctions, which caused it to navigate to the wrong maze branch.

In experimental setup B, the evolved controllers had an average solve rate of only 42%. The best controller had a solve rate of 88%, and only 12 controllers were able to correctly solve the T-maze in more than 50% of the samples.

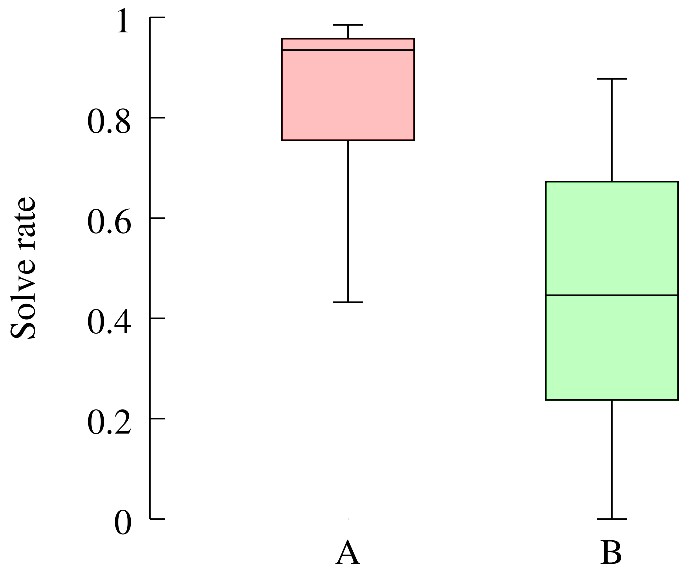


FIGURE 4.2: Summarized results from simulation for setup A and setup B.

When one of the controllers evolved in a traditional ER setup is able to solve a maze, it can sometimes solve the maze faster than controllers that rely on preprogrammed behaviors. The reason for the performance difference is that the preprogrammed *turn* behaviors cause the robot to turn on the spot while a neural network that has direct control over the actuators can turn the robot while it continues to move forward.

4.2.1 Transfer to Real Robotic Hardware

After the evolutionary process had finished, the 5 highest performing controllers synthesized in setup A, and the 5 highest performing controllers synthesized in setup B were tested on a real e-puck. Each controller was tested 16 times, 4 for each light configuration. The results are listed in Table 4.1.

All of the 5 controllers synthesized based on preprogrammed behaviors were able to successfully cross the reality gap and solve the real maze consistently. The controllers synthesized in run A22 and A25 managed to solve all 16 samples. The remaining 3 controllers sometimes navigated to an incorrect maze branch: A4 and A13 failed 1 out of 16 samples, and A9 failed 2 out of 16 samples.

The controllers from setup B did not display as high a performance as those synthesized in setup A. Partly, this was because their in simulation performance was not as high as the one in experimental setup A. 4 of the 5 controllers transferred correctly, achieving even comparable performance in reality, but the controller from trial B19 only solved 11 out of 16 samples in the real robot experiments.

Table 4.1: Summary of the real robot results for the controllers of the five highest performing evolutionary runs of experimental setup A and experimental setup B.

Evolutionary run	**A22**	**A9**	**A25**	**A13**	**A4**	**Average**
Solve rate (Simulation)	99%	98%	98%	97%	97%	98%
Solve rate (Real robot)	100%	88%	100%	94%	94%	95%
Evolutionary run	**B11**	**B13**	**B19**	**B16**	**B9**	**Average**
Solve rate (Simulation)	88%	86%	79%	70%	70%	79%
Solve rate (Real robot)	100%	100%	56%	75%	75%	81%

4.3 Discussion

We demonstrated how controllers can be synthesized by combining artificial evolution with simple preprogrammed behaviors. Our results show that the proposed approach found good solutions in fewer generations and achieved higher final fitness scores (Mann-Whitney U, $p < 0.01$) than in a traditional ER setup in which the neural controller has direct control over the robot's actuators. On real robotic hardware, the performance of the controllers synthesized with our approach was similar to their performance in simulation. The results were also comparable with the real robot experiments where the behavior arbitrator had access to previously evolved behavior primitives (see Chapter 3).

We gave neural controllers three simple preprogrammed behaviors: *follow wall*, *turn left*, and *turn right*. If we had used a different set of preprogrammed behaviors, we would potentially have seen different solutions. The solution space is defined by the set of behaviors to which a neural controller has access and is smaller than the solution space in a traditional ER setup in which the neural controller has direct control over the robot's actuators. The restricted solution space may exclude the optimal solution(s) for a given robot and task. In our experiments, the controllers that had direct access to the actuators were able to cut corners and continued to move forward while turning in a T-junction. The controllers synthesized in our approach were limited to the *turn left* and *turn right* behaviors that cause the robot to turn $90°$ on the spot. Consequently, controllers that had direct access to the robot's actuators were sometimes able to complete the task faster than the controllers that were restricted to a predefined set of preprogrammed behaviors.

While the use of a finite set of predefined behaviors may forestall the synthesis of the theoretically optimal controllers, it opens a number of interesting possibilities. Behaviors can be hand-optimized for a particular robot and for particular sub-tasks. For some sub-tasks, it may be relatively easy to rely on artificial evolution to find a good solution, while for others, such as those that are difficult to simulate or transfer with sufficient accuracy, may be more easily solved by manually programming a behavior.

Chapter 5

Conclusions

In this dissertation, we demonstrated how controllers can be composed in a hierarchical fashion to allow for the evolution of behavioral control for a complex task, and for the successful crossing of the reality gap. We suggest to divide the task into two or more sub-tasks, when a fitness function that allows for the bootstrapping of behavior cannot easily be found. In this way, controllers for complex tasks can be synthesized in a hierarchical fashion, while at the same time, they can benefit from evolutionary robotics techniques, namely (i) automatically synthesis of control, and (ii) evolution's ability to exploit the way in which the world is perceived through the robot's (often limited) sensors.

We evaluated the evolved behavior on a real e-puck performing a rescue task. The real robot managed to solve the task in 22 out of 24 experiments (solve rate of 92%), which is similar to the robot's performance in simulation (solve rate of 93% in 400 experiments). We further experimented with giving evolution access to preprogrammed behaviors. Controllers evolved faster and to a higher degree of performance with the preprogrammed behaviors (average solve rate of 87%) when compared with a controller composed of a single ANN (average solve rate of 42%). When we transferred the best controllers from each experimental setup to a real robot, the controllers with access to preprogrammed behaviors outperformed controllers composed of a single ANN (average solve rate of 95% and 81%, respectively).

The transfer of behavioral control from simulation to a real robot is usually a hit or miss because a controller for the goal task is completely evolved in simulation before it is tested on real hardware. In our approach, the transfer from simulation to real robotic hardware can be conducted in an incremental manner as behavior primitives and sub-controllers are evolved. This allows the designer to address issues related to transferability immediately and locally in the controller hierarchy. By giving evolution access to preprogrammed behaviors, we can still take advantage of the benefits of evolutionary techniques while being able to solve fine sensory-motor tasks. Such tasks would be very difficult to evolve and to transfer to a real robot.

The potential cost of an engineered approach, such as the approach proposed in this dissertation, is that evolution is constrained. Surprisingly simple and elegant solutions that the experimenter did not foresee may therefore never be discovered. This limitation, however, is a widely accepted fact for classical controllers that are programmed by hand. By mixing evolutionary techniques with a hierarchy that is designed by a human, we can synthesize controllers for complex tasks more efficiently, while still taking advantage of the benefits of artificial evolution of controllers.

Some researchers advocate the use of implicit, behavioral, and internal fitness functions [11], because fitness functions with such characteristics, in theory, allow for solutions to emerge through an autonomous self-organization process. In practice, however, such fitness functions, which are supposed to be redeemed from any constraints imposed by a priori knowledge, are often the result of a series of unsuccessful experiments. After each unsuccessful experiment, the fitness function is modified based on the results of the experiment and based on the experiment's guess concerning what may be "wrong". As a result, the fitness function used in the final successful experiment often contains factors and values, and sometime even entire terms that seem arbitrary. We do not dismiss the potential benefits of implicit, behavioral, and internal fitness functions in our approach. Instead, we suggest dividing the task into more sub-tasks, when such a fitness function cannot easily be found.

Our long-term goal is to combine the benefits of manual design of behavioral control with the benefits of automatic synthesis though evolutionary computation to obtain capable, efficient, and robust controllers for real robots. We think our approach could be adapted for the evolution of controllers for multi robot systems, where the interaction between various robots makes the manual design of control systems much more challenging. Since it may be difficult to decide how to decompose a multi robot task into different sub-tasks in the way discussed in this dissertation, the hierarchical approach might be adapted for these tasks. By giving robots access to various social behaviors (such as “follow teammate”) and/or task-oriented behaviors (such as “find red ball”), we could potentially allow for an easier evolution of controllers. The use of preprogrammed behaviors could also facilitate the transfer of the controllers to real robots, as shown in our experiments. In many multi robot systems, communication also plays a fundamental role. Communicative behaviors, however, are not always exclusive and can be carried out in parallel with other behaviors such as those related to locomotion. We will therefore study how to incorporate communication so that it can take place in parallel with motor-control behaviors in the hierarchy.

Appendices

Appendix A

E-puck Sensor Samples

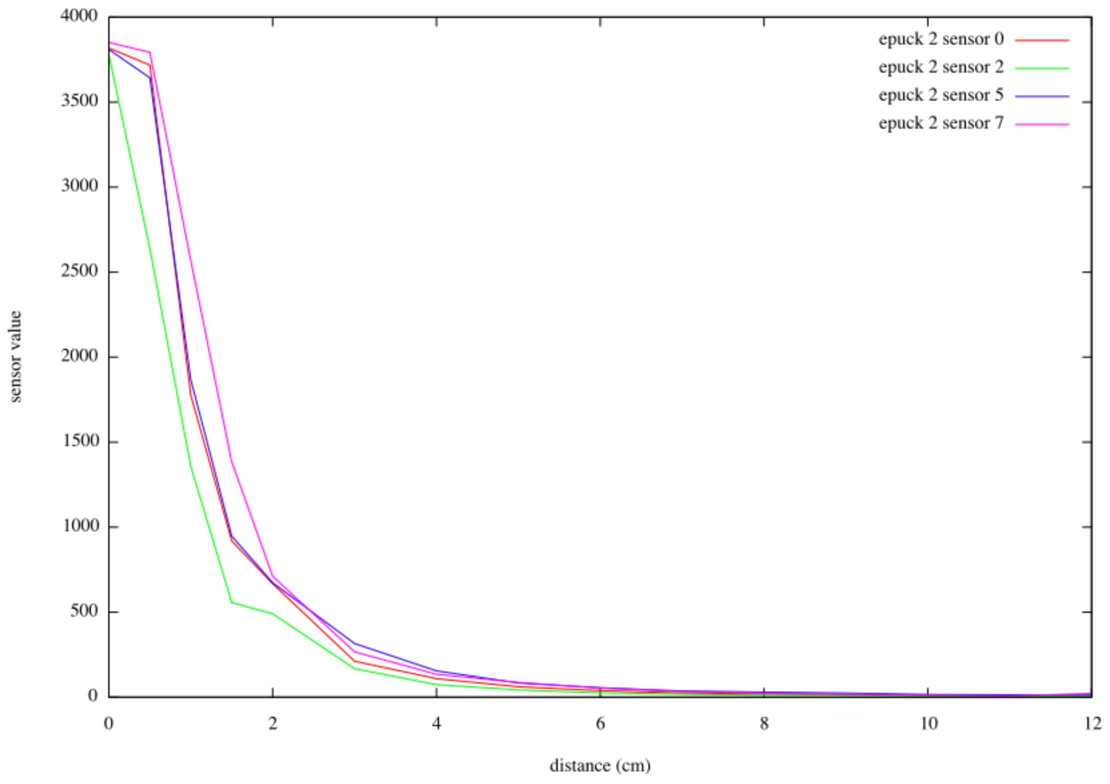


FIGURE A.1: E-puck sensor samples

Appendix B

Turn Left Behavior Mazes

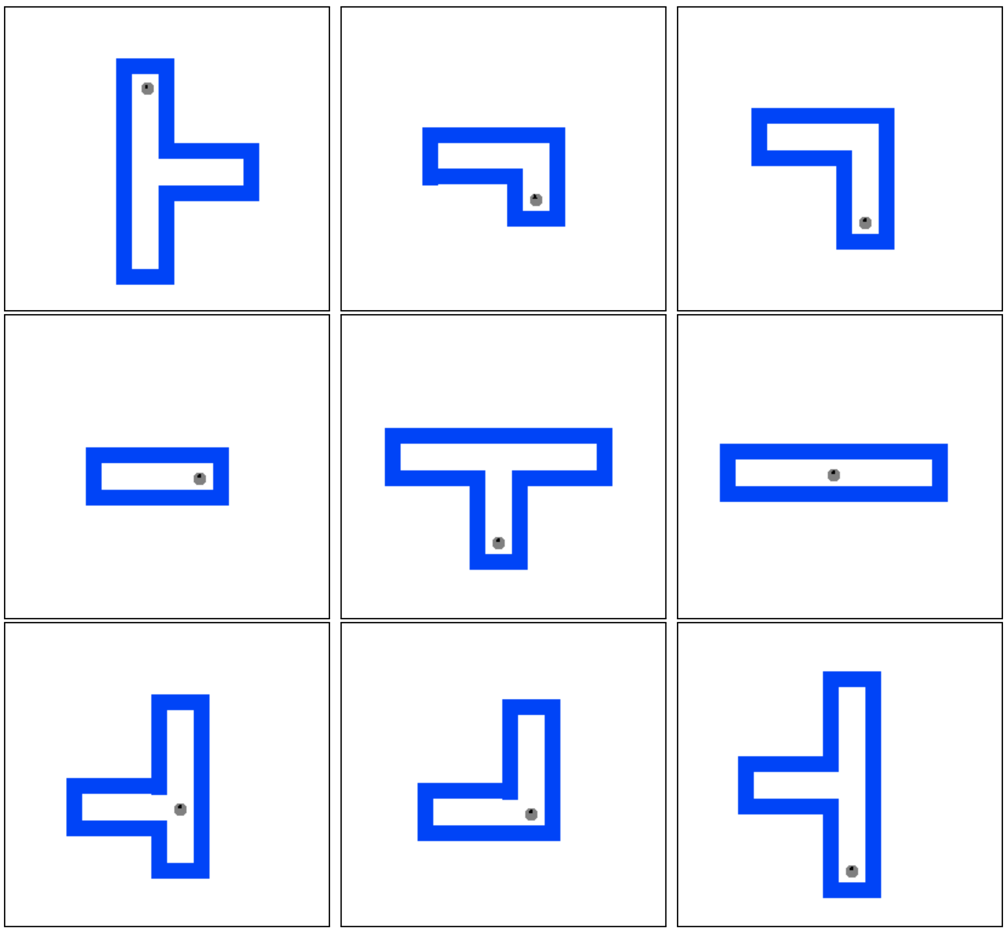


FIGURE B.1: For the "turn left" behavior primitives we used a total of 9 different training mazes. For the "turn right" behavior primitive, the mazes were mirrored.

Bibliography

[1] R. C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, MA, 1998.

[2] N. Barricelli. Esempi numerici di processi di evoluzione, 1954.

[3] R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1:91–122, 1992.

[4] J. Blynel and D. Floreano. Exploring the t-maze: Evolving learning-like robot behaviors using CTRNNs. In *Applications of Evolutionary Computing*, pages 593–604. Springer, Berlin, Germany, 2003.

[5] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, March 1986.

[6] A. L. Christensen and M. Dorigo. Evolving an integrated phototaxis and hole avoidance behavior for a swarm-bot. In *Proceedings of Tenth International Conference on the Simulation and Synthesis of Living Systems (ALIFEX)*, pages 248–254. MIT Press, Cambridge, MA, 2006.

[7] C. Darwin. *On the origin of species.* New York. Appleton and Co., 1859.

[8] R. de Nardi, J. Togelius, O. E. Holland, and S. M. Lucas. Evolution of neural networks for helicopter control: Why modularity matters. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'06)*, pages 1799–1806. IEEE Press, Piscataway, NJ, 2006.

[9] M. Duarte, A. L. Christensen, and S. Oliveira. Towards artificial evolution of complex behaviors observed in insect colonies. In *Proceedings of the 15th*

Portuguese conference on Progress in artificial intelligence, EPIA'11, pages 153–167, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] D. Floreano and L. Keller. Evolution of adaptive behaviour in robots by means of Darwinian selection. *PLoS Biology*, 8:1–8, 2010.

[11] D. Floreano and J. Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4–5):431–443, 2000.

[12] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, (5):317–342, 1997.

[13] M. W. Greenia. *History of Computing: An Encyclopedia of the People and Machines that Made Computer History.* Lexikon Services, Elverta, CA, 2001.

[14] R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in swarmbots. *IEEE Trans. Robot*, pages 1115–1130, 2006.

[15] A. Gutierrez, A. Campo, M. Dorigo, D. Amor, L. Magdalena, and F. Monasterio-Huelin. An open localization and local communication embodied sensor. *Sensors*, 8(11):7545–7563, 2008.

[16] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: artificial evolution, real vision. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 392–401. MIT Press, Cambridge, MA, 1994.

[17] P. Husbands. Evolving robot behaviours with diffusing gas networks. In *Proceedigs of the 1sr European Workshop Evolutionary Robotics, EvoRobot98*, pages 71–86. Springer, Berlin, Germany, 1998.

[18] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6:325–368, 1997.

[19] J. Kam-Chuen, C.L. Giles, and B.G. Horne. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on Neural Networks*, 7:1424–1438, 1996.

[20] S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 2012. In press.

[21] T. Larsen and S.T. Hansen. Evolving composite robot behaviour - a modular architecture. In *Robot Motion and Control, 2005. RoMoCo '05. Proceedings of the Fifth International Workshop on*, pages 271 – 276, 2005.

[22] W.-P. Lee. Evolving complex robot behaviors. *Information Sciences*, 121(1-2):1–25, 1999.

[23] O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2:417–434, 1996.

[24] R. C. Moioli, P. A. Vargas, F. J. Von Zuben, and P. Husbands. Towards the evolution of an artificial homeostatic system. In *IEEE Congress on Evolutionary Computation*, pages 4023–4030. IEEE Press, Hong Kong, China, 2008.

[25] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. Instituto Politecnico de Castelo Branco, Castelo Branco, Portugal, 2009.

[26] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

[27] A. Newell and H. A. Simon. Computer science as empirical inquiry: symbols and search. *Commun. ACM*, 19(3):113–126, March 1976.

[28] N. J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

[29] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, Cambridge, MA, 2000.

[30] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Proceedings of the 4th International Workshop on Artificial Life*, pages 190–197. MIT Press, Cambridge, MA, 1994.

[31] E. C. Tolman and C. H. Honzik. Introduction and removal of reward, and maze performance in rats. *University of California Publications in Psychology*, 4:257–275, 1930.

[32] A. B. L. Torta, M. A. Kramer, C. Thorn, D. J. Gibson, Y. Kubota, A. M. Graybiel, and N. J. Kopell. Dynamic cross-frequency couplings of local field potential oscillations in rat striatum and hippocampus during performance of a t-maze task. *Proceedings of the National Academy of Sciences*, 105(51):20517–20522, 2008.

[33] E. Tuci, V. Trianni, and M. Dorigo. 'feeling' the flow of time through sensorimotor co-ordination. *Connection Science*, 16(4):301–324, 2004.