Evolution-In-Materio: Solving Computational Problems Using Materials

Maktuba Mohid

PhD

UNIVERSITY OF YORK

Electronics

June 2015

Abstract

The motivation behind the research is to show that evolutionary algorithms can exploit properties of materials to solve various computational problems without requiring a detailed understanding of such properties. This approach is referred to as evolution-in-materio. In this research, it has been shown that using a purpose-built hardware platform called Mecobo, it is possible to evolve voltages and signals applied to physical materials to solve a number of computational problems. Here it has been demonstrated for the first time that the evolution-in-materio method can be applied to function optimisation, machine learning classification, frequency classification, even parity and bin packing problems. This evolution-in-materio method has also been applied here to discriminate tones and control robots. The physical material used in each of these experiments is a mixture of single-walled carbon nanotubes and a polymer. This is the first time that such material has been used to solve computational problems. The results of all of these experiments indicate that evolution-in-materio has promise and further investigations would be fruitful. Other than the solutions regarding these computational problems, this thesis has also devised and investigated suitable input-output mappings and input signals that allow various computational problems to be solved using the Mecobo platform and the experimental material.

Contents

List of Tables

List of Figures

Acknowledgements

Throughout my Ph.D., there have been a number of people who played a role and to whom I want to be thankful. My supervisor, Dr. Julian F. Miller, spared so much time and energy for guiding me. Thank you, Julian, for your guidance, support and inspiration.

I would like to thank to my family. First of all, I want to thank to my husband, without whom I could not even start this Ph.D. My father always insisted me to do the Ph.D. and my mother always supported me all through my life to carry out my education. Thank you, mom and dad. I want to be thankful to my younger sister to spend her valuable times with my son, which helped me to study. I want to thank to the little and the youngest member of my family, who is none other than my son, Taurat, for letting me concentrate on my study whenever it was a time to play with him.

I would like to thank to all my office colleagues for their supports and helps: Andrew Turner and Selahattin Kosunalp, for helping me by the valuable discussion about many issues regarding this thesis, Jose Sanchez De Lucio, for restarting the hardware platform manually every time whenever I was away from the university so that my experiments would be continued without making any delay. I would like to thank the staffs of the university: Camilla Dense, Helen Fagan, James Hilder, Angelica Jonsson, Neil Couper and Dr. Martin Trefzer for their supports and helps.

I would like to thank my undergraduate supervisor, Dr. Mohammad Kaykobad, who encouraged and insisted me to apply for a Ph.D. in UK.

This research is part of an EU-project named NASCENCE (nanoscale engineering for novel computation using evolution) and has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317662. The last, but not the least, I would like to thank for the funding and to my colleagues of this project: Simon Harding, Odd Rune Lykkebø, Stefano Nichele and Mark K. Massey for their helps and supports regarding the experimental materials and the hardware platform.

For Taurat

Declaration of Authorship

I declare that this thesis titled, 'Evolution-In-Materio: Solving Computational Problems Using Materials' and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at the University of York.
- This work has not previously been presented for an award at this, or any other, University.
- Where I have consulted published work, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Some of the material incorporated in this thesis has been presented previously in the following publications, as detailed in the introductory text (Mohid *et al.*, 2014d,c,b,a; Mohid and Miller, 2015a; Mohid *et al.*, 2015; Mohid and Miller, 2015c,b).

Chapter 1

Introduction

Contents

'In conventional design the vast majority of interactions that could possibly contribute to the problem are deliberately excluded' [Conrad (1988)]

Conrad's statement is both intriguing and paradoxical. Why are these interactions excluded? The answer is that designers are unaware of such interactions. However, artificial evolution is a technique that can be used to utilise these. Indeed, one of its potential advantages is that it can exploit physical effects that are either unknown or too complex to understand. In the 1990s, Thompson discovered that he was able to utilise the physical properties of a silicon chip called a field programmable gate array (FPGA) to solve computational problems by demonstrating unconstrained evolution on the chip [Thompson (1998)]. From his experiment in unconstrained evolution, the concept of evolution-in-materio emerged. Evolution-in-materio is the manipulation of a physical system by computer controlled evolution to solve computational problems [Miller and Downing (2002); Miller *et al.* (2014); Massey *et al.* (2015a)]. In evolution-in-materio, a material operates as a computational device. A number of input signals are sent to the material and the recorded response is interpreted as a computation based on a pre-specified scheme (e.g. input-output mapping, fitness calculation method). There are two categories of input signals: the data input, which is defined by the computational task; and the configuration inputs that are evolved to bring the material to the desired computation-inducing state. Evolutionary algorithms are used to search over the space of possible configuration inputs. Evolutionary algorithms are fitting as they are computer algorithms that utilise the forces of natural evolution and self-adaptation [Greenwood and Tyrrell (2006)].

For evolution-in-materio, three components are necessary:

- A physical material;
- A hardware platform that generates input signals, sends the signals to the material and reads the output signals from the material;
- A computer that controls the evolution using an evolutionary algorithm and the hardware platform.

Evolution-in-materio has been applied before to a few problems using a liquid crystal display (LCD): building logic gates, tone discrimination and controlling a simulated robot [Harding and Miller (2004a, 2005); Harding (2006); Harding and Miller (2007)]. In this thesis the range of computational problems, to which evolution-in-materio has been applied, has been considerably extended. Such problems are: machine learning classification, frequency classifier, even parity, function optimisation, bin packing problems and control of both simulated and real robots. In addition, the experimental apparatus and computational material are different to past work. Electronic signals are applied to electrode arrays containing mixtures of single-walled carbon nanotubes and a polymer. Single-walled carbon nanotubes are the conducting or semi-conducting materials in the mixture and the role of the polymer is to introduce insulating regions within the nanotube network, to create non-linear current versus voltage characteristics. Another benefit of the polymer is to help with the dispersion of the nanotubes in solution. The polymer used in the mixture is polymethyl methacrylate (PMMA) or poly butyl methacrylate (PBMA). A variety of mixtures of these materials have been used in the experiments of this thesis.

This research is part of an EU-funded research project named NASCENCE (nanoscale engineering for novel computation using evolution) [Broersma *et al.* (2012)]. The objective of this project is to understand, model and exploit the properties of nanosystems using evolution. The long-term goal of this project is to develop both theoretical and technological foundations of a new type of information processing technology using natural evolution and the advancement of nanotechnology. Four universities and one research institute are involved in this project. These are: the University of York, Durham University[1], the Norwegian University of Science and Technology[2], the University of Twente[3], the Dalle Molle Institute for Artificial Intelligence (IDSIA)[4]. The equipment involved in this research (the physical material, the hardware platform and the computer that controls the evolution-in-materio) has also been produced by this project.

1.1 Hypothesis

It is possible to solve function optimisation, machine learning classification, bin packing, tone discriminator, frequency classification, even parity and robot controlling problems by using computer controlled evolution of signals applied to electrode arrays containing a mixture of single-walled carbon nanotubes and a polymer.

1.2 Thesis Layout

This thesis is organised into nine chapters.

Chapters 2 and 3 deal with the motivations that lie behind this research and former published research that provides the foundations of this thesis. Chapter 2 defines and describes evolutionary algorithms along with different components of evolutionary algorithms. Cartesian genetic programming is a well-known software-based evolutionary approach. The experimental results of some of the problems have been compared with the results of Cartesian genetic programming to evaluate the effectiveness of the evolution-in-materio method for solving such problems.

Chapter 3 discusses relevant aspects of published work on evolvable hardware, evolvable motherboard and physical computation. Evolution-in-materio, with which the topic of this thesis is mainly related, is described in this chapter with a conceptual overview

[1]https://www.dur.ac.uk
[2]http://www.ntnu.edu
[3]http://www.utwente.nl/en
[4]http://www.idsia.ch

and discussion of past related work. Evolution-in-materio research carried out by the members of the NASCENCE project is also described in Chapter 3, providing a clear overview of the work of this European project and how it relates to the thesis.

Chapter 4 describes the equipment used in this research. The equipment consists of the interface hardware named Mecobo, the interface software and the experimental material.

Chapters 5, 6, 7 and 8 discuss and analyse the experiments performed in this research. Chapters 5 and 7 describe those experiments which require few outputs. Of these, Chapter 5 describes classification-based experiments which were used to find solutions to machine learning classification, tone discriminator, frequency classifier and even parity problems. Chapter 7 describes the experiments that use evolution-in-materio to control robots. Both simulated and real robots have been used in these experiments.

Chapter 6 describes two experiments which require many outputs. The problems are complex multi-modal optimisation functions and the NP-hard bin packing problems. Both of these experiments used a technique to handle many outputs, which is called the *split genotype technique*. The technique and the problems which are suitable for its application are also discussed.

Chapter 8 mainly focuses on the analysis regarding the choices of the computational problems, the experiments and the outcomes obtained in the experiments to solve the computational problems of this research. This chapter also deals with some other investigations, such as the stability test on the experimental material, investigations on the input-output mappings and on the experimental settings controlled by the hardware platform to set up input and output signals. Nothing is smooth in this world, so the experiments of this research also faced a number of problems which are described in Chapter 8. Recognising these issues helps to draw a guideline for future work.

Finally, Chapter 9 concludes the thesis with a detailed discussion about future work in the area of evolution-in-materio.

1.3 Contributions

The novel and original contributions of this thesis are:

For the first time, it has been possible to

- show configurations of single-walled carbon nanotubes and polymers can be evolved to solve computational problems;
- devise and investigate suitable input-output mappings and input signals that allow various computational problems to be able to be solved using electrode arrays;
- show that it is possible to use evolution-in-materio to solve well-established and difficult computational problems (function optimisation, machine learning classification, frequency classification, bin packing, even parity and robot controlling problems).

1.4 Publications

Some of the work incorporated in this thesis has been presented in previous publications, through conference and workshop proceedings and journal papers. Six papers were submitted for publication in conference and workshops. Of these, four papers have already been published and two have been reviewed and accepted. Two papers have been accepted for publication in two journals. The frequency classification experiments of Chapter 5 and the bin packing experiments of Chapter 6 were published in [Mohid *et al.* (2014a)] and [Mohid *et al.* (2014b)] respectively. The work of function optimisation experiments of Chapter 6 was demonstrated and published in [Mohid *et al.* (2014c)] and has been accepted for publication in journal [Mohid *et al.* (2015)]. The work on tone discriminator described in Chapter 5 has also been accepted by the same journal, i.e. in [Mohid *et al.* (2015)]. The machine learning classification experiments of Chapter 5 were published in [Mohid *et al.* (2014d)] and have been reviewed and accepted for publication in journal [Mohid and Miller (2015b)]. Some pieces of work performed with a simulated robot of Chapter 7 have also been accepted in the same journal, i.e. in [Mohid and Miller (2015b)] for publication. The rest of the work performed with the simulated robot and the experiments of the real robot of Chapter 7 have been reviewed and accepted for publication in [Mohid and Miller (2015a)]. The work of even parity experiments of Chapter 5 has been reviewed and accepted for publication in [Mohid and Miller (2015c)].

Part I

Motivation and Foundations

Chapter 2

Evolutionary Algorithms

Contents

In a biological context, 'evolution' is the mechanism which allows organisms to adapt to their environment by changing their characteristics over time [Darwin (1859)]. In engineering, the word 'evolution' refers to a process whereby products or processes change over time. This is not to be confused with evolution in a Darwinian sense.

In artificial intelligence, evolution is performed using an evolutionary algorithm. An evolutionary algorithm is a form of non-deterministic search, which is inspired by biological evolution and involves processes such as selection, reproduction, recombination and mutation. It is used in engineering, biology, economics, art, genetics, physics, robotics, chemistry and many other fields.

Evolutionary algorithms were first described by biologists in 1950, who were trying to simulate evolution [Barricelli (1954); Fraser (1957); Friedman (1956, 1959); Campbell (1956a,b, 1960)]. The uses of evolutionary algorithms did not become more widespread and evolutionary algorithms were not used as a technique to solve problems until the 1960s and 1970s. Four types of evolutionary algorithm became more prominent. These are evolutionary programming [Fogel (1998)], evolutionary strategies [Rechenberg (1971); Schwefel (1974)], genetic algorithms [Holland (1992)] and genetic programming [Koza (1992)]. Genetic programming is a search technique which is inspired by Darwinian evolution and is able to produce programs to solve problems. In the year 1997, a form of genetic programming called Cartesian genetic programming was added, which is based on an encoding of graphs [Miller *et al.* (1997)].

2.1 Components of Evolutionary Algorithms

There are a number of components, operators, or procedures that must be specified to define an evolutionary algorithm. These components are described in following sections [Michalewicz (1996); Mitchell (1998); Greenwood and Tyrrell (2006); Eiben and Smith (2015)]:

2.1.1 Representation

The representation is a data structure which encodes the information required to solve a specific problem. In evolutionary algorithms, a gene is an encoded problem parameter. The genotype is the complete set of all genetically encoded information required in order to solve a computational problem. The data structure used for the genetic representation can be represented as a binary bit string, a collection of integers, real numbers, graphs and a mixture of binary, integer and real numbers. Consider an example in circuit design. If the gates used are chosen from the set {NOT, OR, AND, NAND, NOR, XOR}, any circuit configuration using two gates can be encoded with a 7-bit binary string: the first three bits select the first gate (000 for NOT, 001 for OR and so on); next three bits select the second gate; one bit indicates if the two gates are in series (logic 0) or parallel (logic 1). Then 0010110 means OR and NAND gates are connected serially together.

Genetic algorithms use strings of binary values or integers or real numbers for representing genotypes [Holland (1992); Davis (1991); Eshelman and Schaffer (1992)].

Usually, a genetic programming uses tree structures for representation. However, in linear genetic programming, the programs are constrained to linear sequences of instructions (operations and inputs) [Poli *et al.* (2008)]. Cartesian genetic programming uses graph-based representation [Miller (2011)]. An example of genetic programming tree is shown in Figure 2.1.

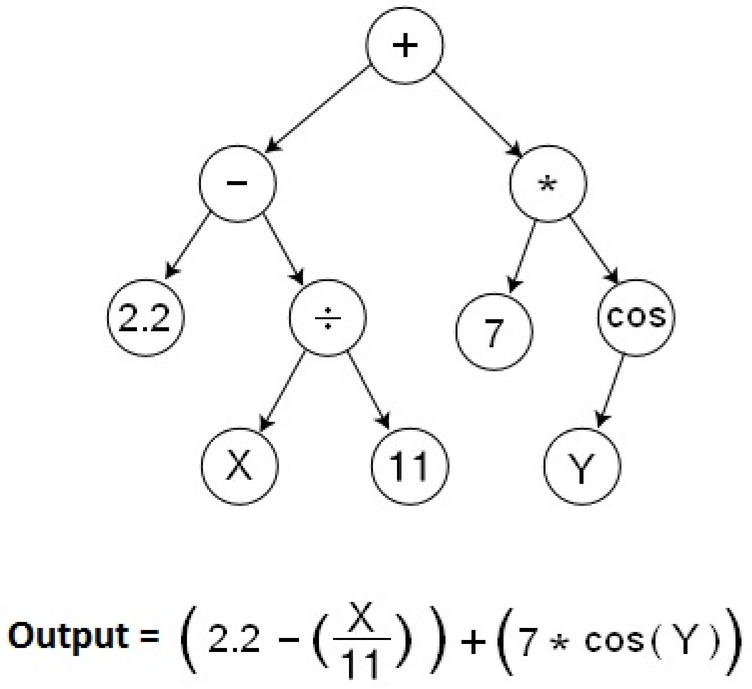


FIGURE 2.1: An example of genetic programming tree.

2.1.2 Population

The population is a set of solutions that evolve in the evolutionary run. Population size is the number of genotypes in each generation. Usually, the initial population is generated randomly. However, sometimes the initial population is generated by hand to have a good starting position at the beginning of an evolutionary run.

2.1.3 Variation Operators

Variation operators are used to create new individuals from old ones. Two types of variation operators are used: recombination and mutation.

2.1.3.1 Recombination

Recombination is the process where new genotypes are created by combining two or more genotypes of the previous generation. The genotypes of the new generation are called children and genotypes of the previous generation are called parents. The point, at which two parents are divided and combined to generate children, is called the crossover point. When two parents are divided at one point, it is called one-point crossover. When parents are divided in more than one point (n number of points), it is called n-point crossover. An example of crossover is shown in Figure 2.2. The percent crossover determines the number of children to be produced in each generation.

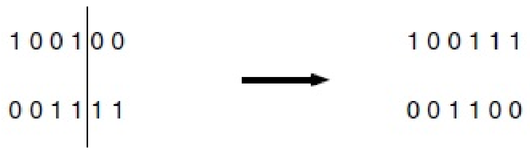


FIGURE 2.2: An example of one-point crossover [Greenwood and Tyrrell (2006)]. Two parents are shown in the left and two children are shown in the right. The crossover point is indicated by the line in the parents. The bits after the crossover point are swapped in the children.

In the case of genetic algorithms, the crossover operator is the main operator [Holland (1992)]. It is considered that if partial solutions occur in parents, at the time of crossover, it can unite the partial solutions in children to create a solution.

Crossover in genetic programming tree selects two function nodes from two genetic programming trees randomly and then exchanges two sub-trees under those two function nodes between each other [Poli *et al.* (2008)]. An example of crossover of genetic programming tree is shown in Figure 2.3.

2.1.3.2 Mutation

Mutation is the process where a child is produced by altering one or more genes of a parent. Mutation can be done in many ways: swapping two genes, mutation by inversion. An example of mutation is shown in Figure 2.4.

Different types of user-chosen parameter are used for mutation. Mutation rate is used to define how many times the part of the chromosome will be mutated. In the case of

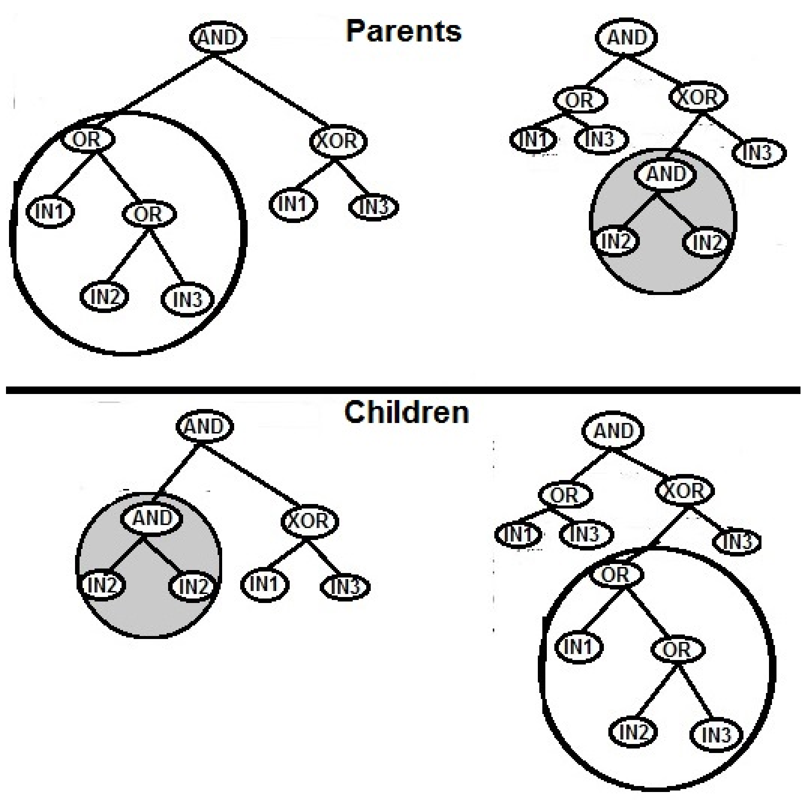


FIGURE 2.3: An example of crossover of genetic programming tree.

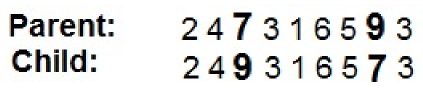

Parent: 2 4 **7** 3 1 6 5 **9** 3
Child: 2 4 **9** 3 1 6 5 **7** 3

FIGURE 2.4: An example of mutation by swapping two genes.

mutation by probability, a small probability is used to choose each gene and a number is generated randomly for each gene. Depending on that random number and the probability, a gene is chosen and mutated.

The mutation is the secondary operator in traditional genetic algorithms [Holland (1992)]. In genetic algorithms, the mutation is applied after crossover. Mutation is used to prevent early convergence on sub-optima.

In evolutionary programming, it is the only variation operator, which is solely responsible for the generation of new individuals.

Mutation in genetic programming happens after recombination [Poli *et al.* (2008)]. In mutation, randomly nodes are selected, and then the sub-trees under those function nodes are substituted by randomly generated sub-trees by mutation. An example of mutation of genetic programming tree is shown in Figure 2.5.

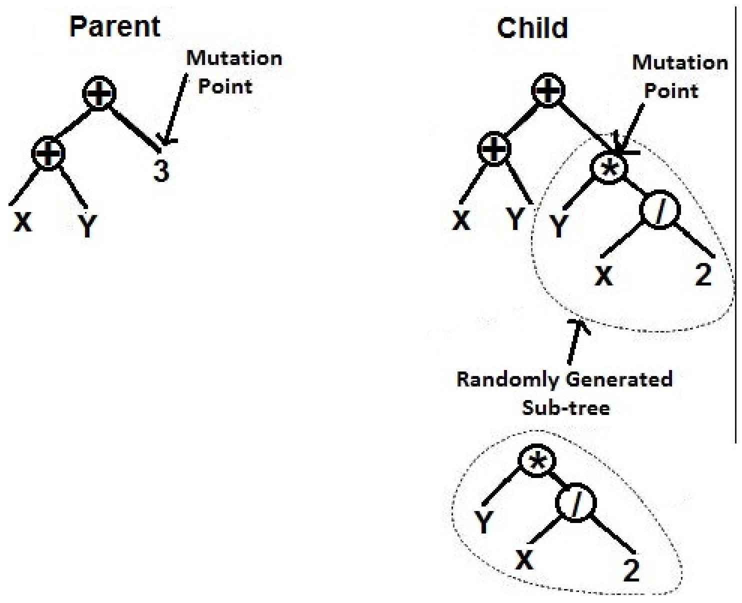


FIGURE 2.5: An example of mutation of genetic programming tree.

2.1.4 Evaluation

After mutation and crossover, an evaluation process is used to evaluate newly created children to promote to the next generation. In this process, normally a numeric score is assigned to each genotype using a fitness function. The score is called the fitness score. The children are promoted to the next generation depending on the fitness scores of their genotypes.

2.1.5 Selection

After the evaluation, a selection process is used to select genotypes of the new generation. There are many selection methods:

- Fitness proportional selection: In fitness proportional selection, individuals are selected according to their absolute fitness scores. The probability of selecting any individual uses its absolute fitness score. If the population size is N and the fitness of i^{th} individual is f_i, the probability of i^{th} individual is $Prob(i)$ which can be calculated using Equation 2.1.

$$Prob(i) = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

- Fitness ranking selection: In fitness ranking selection, individuals are selected according to their rankings, which are done using relative fitness scores. The probability of selecting any individual uses ranking instead of its fitness score.
- Truncation selection: In this selection method, μ parents are used to get λ children in each generation. Then $(\mu + \lambda)$ individuals are sorted according to their fitness scores. From these, the best μ individuals are used as parents in next generation.
- Tournament selection: In tournament selection, random uniform samples of q (q>1) individuals are taken from the population, and then the best-fit individual is selected for crossover.

In $(\mu + \lambda)$-evolutionary algorithm, the population size is $(\mu + \lambda)$, where λ children are generated from μ parents, and the μ individuals having the best fitness values are selected to be the parents in the new population [Rechenberg (1971)].

In (μ , λ)-evolutionary algorithm, λ children are generated from the μ parents, where only the λ children form the new population. The μ individuals having the best fitness values from the λ individuals are selected to be the parents in the new population.

2.1.6 Termination Criteria

The evolutionary run terminates when it meets the termination criteria. Three types of criteria can be used to terminate evolutionary runs:

- A fixed number of generations or evaluations can be used.
- The solution has converged, i.e. no improvement is found in the last fixed number of generations.
- The output is not optimum but good enough to be considered.

The general scheme of an evolutionary algorithm as a flowchart is shown in Figure 2.6.

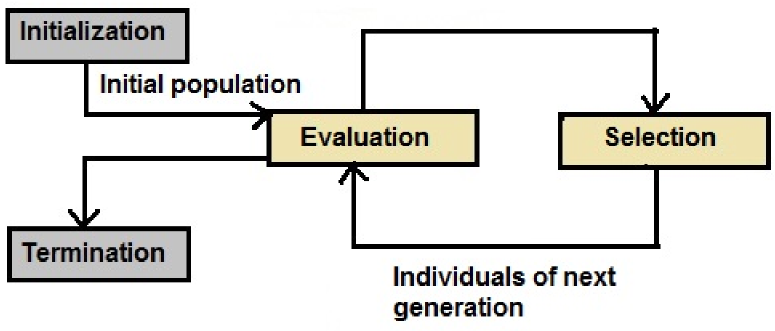


FIGURE 2.6: The general scheme of an evolutionary algorithm as a flowchart.

2.2 Cartesian Genetic Programming

Cartesian genetic programming grew out from the work of evolving digital circuit, which was developed by Miller et al. in the year 1997 [Miller *et al.* (1997)]. The term 'Cartesian genetic programming' first appeared in the year 1999 [Miller (1999)], and then it was proposed as a general form of genetic programming in the year 2000 [Miller and Thompson (2000)].

Cartesian genetic programming uses directed acyclic graphs [Miller (2011)]. The graphs of Cartesian genetic programming are represented by two-dimensional grids of computational nodes, that is why the algorithm is called Cartesian genetic programming. An example of the Cartesian genetic programming graph is shown in Figure 2.7. The genes of the genotype of Cartesian genetic programming are integers. These genes represent where a node gets its data, the operation of the node to be performed on the data and from where the output will be obtained. This means, the genes of the genotype of Cartesian genetic programming can be divided into three categories: function gene, connection gene and output gene. An example of Cartesian genetic programming genotype is shown in Figure 2.7.

- Function gene: Each node of the graph of Cartesian genetic programming represents a function. The types of computational node functions are listed in a look-up table and are decided by the user. The gene of a node, which is the address of computational node function in function look-up table, is called function gene. The function genes are shown in Figure 2.7.

- Connection gene: The node gets its inputs in a feed-forward manner from either the program inputs or from the outputs of nodes of previous columns. The genes of the node, which specify from where the node gets its inputs, are called connection genes. The connection genes represent addresses in a data structure (usually an array). These are integers that take values between 0 and the address of the node located at the bottom of the previous column of nodes. The maximum number of inputs, that any function of the function look-up table has, is called the arity. The number of connection genes of a node is chosen to be its arity. The connection genes are shown in Figure 2.7.

- Output gene: The genes, which specify from where the outputs are taken, are called output genes. The output genes are shown in Figure 2.7.

The user has to set three parameters while running Cartesian genetic programming, these are: number of rows (n_r), number of columns (n_c) and levels-back (l). The number of rows is the number of rows the graph should have, the number of columns is the number of columns the graph should have. If levels-back, l=n_c, a function node can take its inputs from the outputs of any functional nodes of immediate left n_c number of columns or from primary inputs.

At the time of decoding of genotypes, sometimes some nodes are ignored. The ignored nodes and their genes are called 'non-coding'. This happens when the node outputs are not used in the calculation of output data. The nodes, whose outputs are used in the calculation of output data, are called 'active' nodes.

Decoding of Cartesian genetic programming proceeds from the right-hand end, at the output genes. That means, at first output genes are identified, then active functional nodes are identified, from where output genes are obtained; then connection genes are identified, with which previously identified functional nodes are connected. After these connection genes, other active functional nodes are identified, with which these connection genes are connected. In this way, it proceeds by traversing the graph backward. In the decoding process, non-coding nodes are not processed. The decoding process can be implemented in different ways. One is the recursive decoding process, another way is to determine active nodes in recursive way, record them for future use and then only process them.

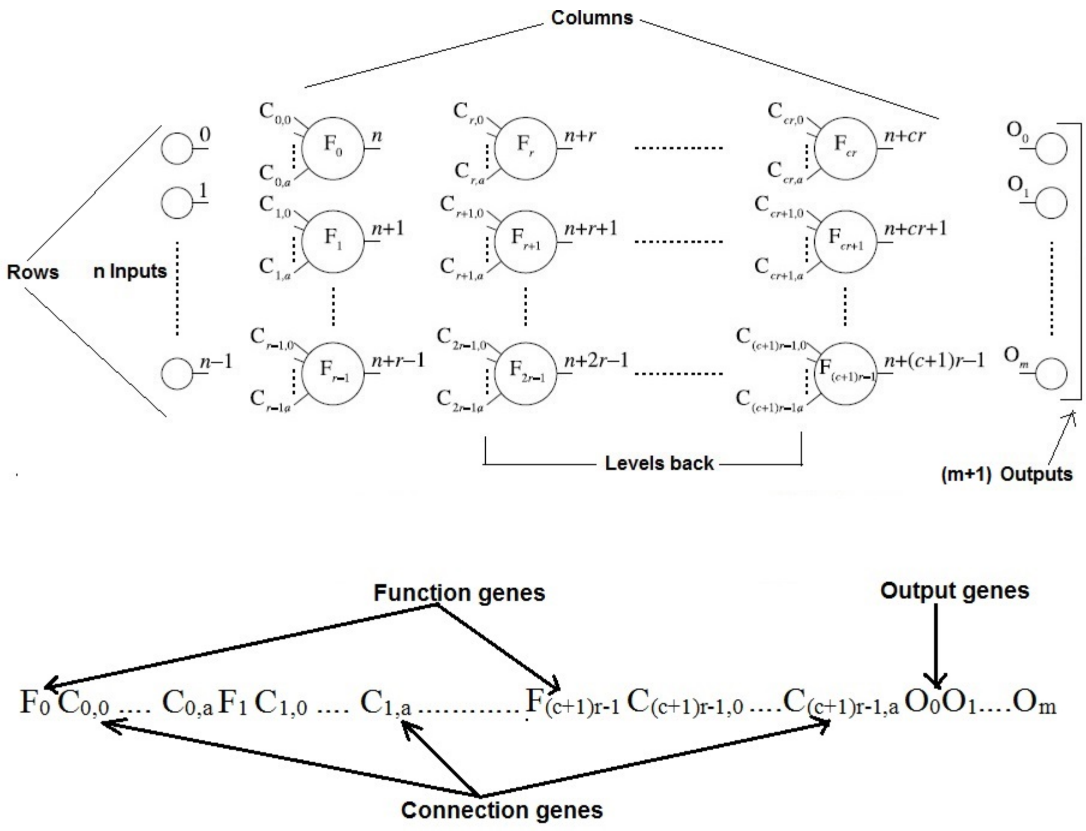

Columns
Rows
n Inputs
Levels back
(m+1) Outputs
Function genes
Output genes
Connection genes
F0 C0,0 .... C0,a F1 C1,0 .... C1,a .... F(c+1)r-1 C(c+1)r-1,0 .... C(c+1)r-1,a O0O1....Om

Figure 2.7: General form of Cartesian genetic programming [Miller (2011)]. It is a grid of nodes whose functions are chosen from a set of primitive functions. The number of program inputs is n_i. The grid has n_c number of columns and n_r number of rows. The number of program outputs is n_o. Each node is assumed to take as many inputs as the maximum function arity, a. The data input or node output is labeled sequentially (starting from index 0), which represents a unique data address that specifies from where the input data or output value can be accessed. Here function genes are $F_0F_1 \dots F_{(c+1)r-1}$, connection genes are $C_{0,0} \dots C_{0,a} \dots C_{(c+1)r-1,a}$ and output genes are $O_0O_1 \dots O_m$.

An example of a Cartesian genetic programming genotype with the corresponding phenotype for a two-bit multiplier is shown in Figure 2.8. An example of decoding procedure of a Cartesian genetic programming genotype (shown in Figure 2.8) for a two-bit multiplier is shown in Figure 2.9.

Usually, recombination is not used in Cartesian genetic programming. Crossover showed a detrimental effect on the performance of Cartesian genetic programming. Point mutation is used in Cartesian genetic programming. In point mutation, the value at a randomly chosen gene location is altered to another valid value, and the valid value is chosen randomly. If the gene is a function gene, the valid value is the address of any function of the function set. If it is a connection gene, the valid value is the address of output of any previous node in the genotype or the address of any program input. In

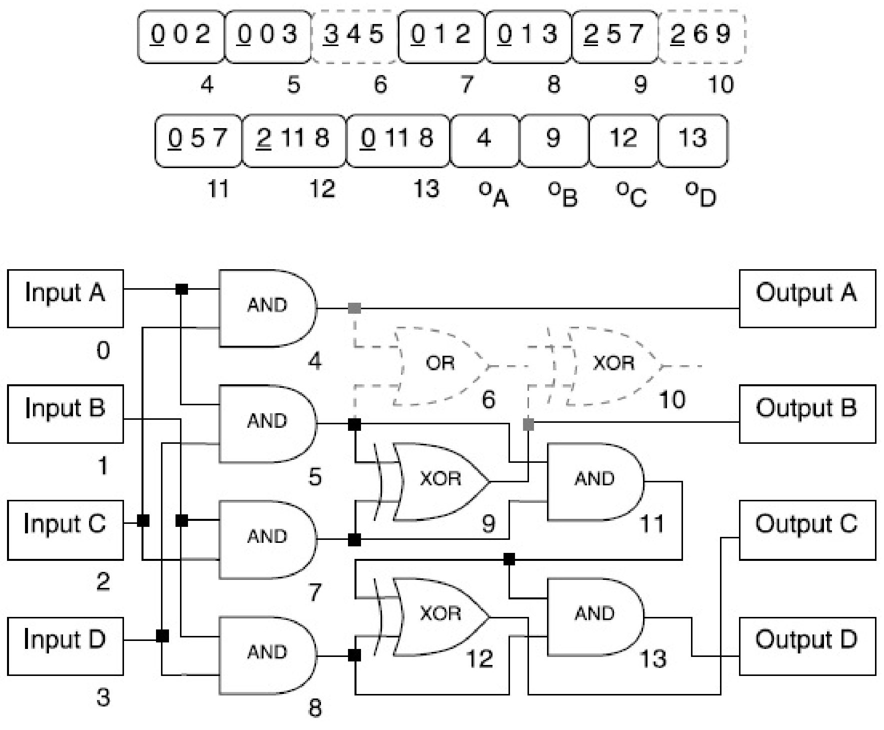


Figure 2.8: An example of a Cartesian genetic programming genotype and corresponding phenotype for a two-bit multiplier circuit [Miller (2011)]. Both in the genotype and phenotype, the addresses are shown underneath the program inputs and nodes. The underlined genes in the genotype encode the functions of the nodes. According to the function look-up table, AND is 0, AND with one input inverted is 1, XOR is 2 and OR is 3. The inactive areas of the genotype and phenotype are shown in grey dashes (here node 6 and 10 are two inactive nodes).

the case of an output gene, the valid value is the address of output of any node in the genotype or the address of any program input. The number of genes of the genotype to be mutated is determined by a user-chosen parameter. The parameter is normally a percentage of total number of genes in a genotype, which is referred to as mutation rate. Sometimes different mutation rates are used for function genes, connection genes and output genes.

In Cartesian genetic programming, (1+4)-evolutionary algorithm is widely used. A child is always chosen if it is equally as fit or has better fitness than the parent. An example of (1+4)-evolutionary algorithm is shown in Figure 2.10.

Cartesian genetic programming has been applied in solving many problems from many areas, such as function optimisation, classification, electronic circuit design, financial prediction, medical diagnostics, evolutionary art and music, image processing, symbolic

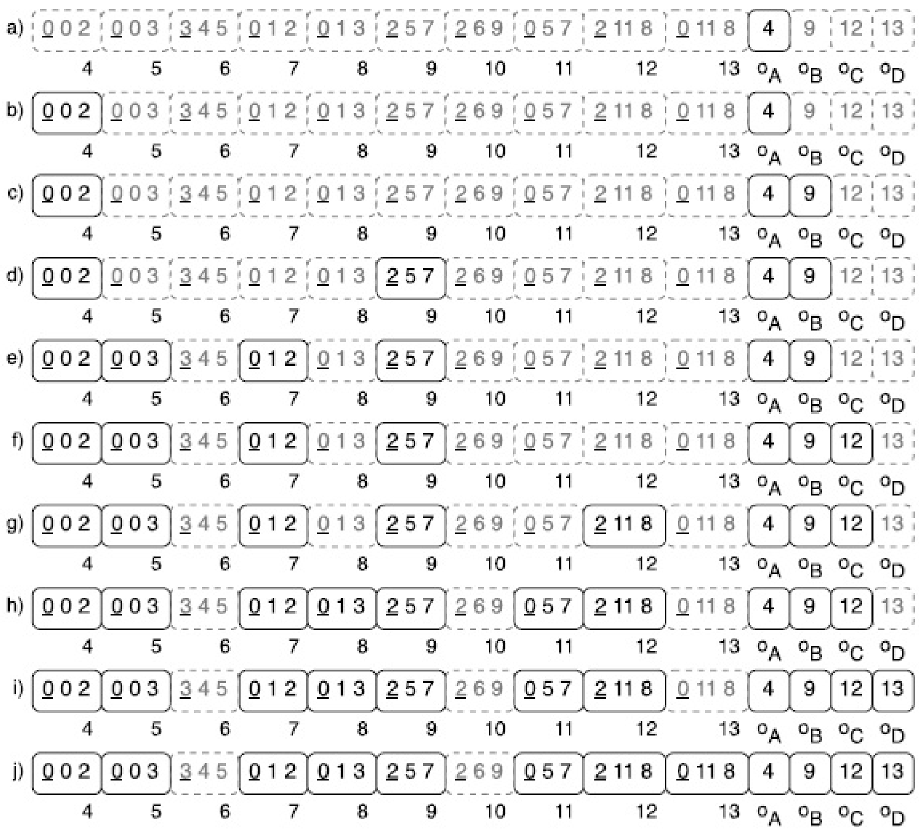


FIGURE 2.9: An example of the decoding procedure of a Cartesian genetic programming genotype for a two-bit multiplier problem [Miller (2011)]. (a) Output A (O_A) connects to the output of node 4. (b) If node 4 is observed, node 4 connects to the program inputs 0 and 2, therefore the output A is decoded. (c) If output B (O_B) is observed, output B connects to the output of node 9. (d) If node 9 is observed, node 9 connects to the outputs of nodes 5 and 7. (e) If nodes 5 and 7 are observed, nodes 5 and 7 connect to the program inputs 0, 3, 1 and 2, therefore output B is decoded. The same procedure continues until outputs C (O_C) and D (O_D) are decoded (steps (f)-(h) for output C and steps (i)-(j) for output D). When all the four outputs are decoded, the genotype is fully decoded.

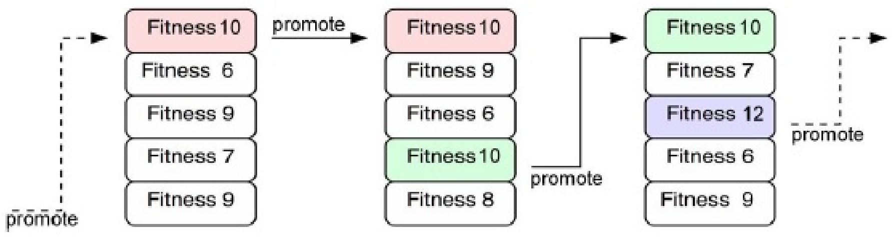


FIGURE 2.10: An example of (1+4)-evolutionary algorithm.

regression, digital adder, travelling salesman problem and so on. Cartesian genetic programming was applied before on some of the computational problems that have been solved in this research, such as machine learning classification, function optimisation, bin packing, even parity and robot controlling problems. Here in this thesis, the results obtained by applying Cartesian genetic programming on some of these problems (machine learning classification, even parity and function optimisation problems) were compared with the results of the experimental material to evaluate the effectiveness of the evolution-in-materio for solving these problems. Dr. Julian F. Miller is the supervisor of this thesis, who is the developer of Cartesian genetic programming. Thus, the Cartesian genetic programming software is available for comparing its results with the results of the experimental material using the same techniques (input-output mappings, fitness calculation methods) and parameters (number of generations, number of runs) in both of these cases (experimental material and Cartesian genetic programming). Cartesian genetic programming is a standard evolutionary search algorithm, which has been proved to be efficient in solving many problems already.

2.3 Summary

An introduction to evolutionary algorithms and the descriptions of their components have been given in this chapter. Cartesian genetic programming has been described in detail here. In the case of three experiments, the results of evolution-in-materio were compared with the results of Cartesian genetic programming. The advantage of using Cartesian genetic programming for comparison is the availability of Cartesian genetic programming software. These experiments along with the results are discussed in Chapters 5 - 8 of this thesis.

Chapter 3

Artificial Evolution of Physical Systems

Contents

Hardware evolution or evolvable hardware is a method which uses evolutionary techniques to design and synthesise hardware. The evolution in evolvable hardware is performed intrinsically or extrinsically. Extrinsic evolution uses simulators or circuit models to evaluate circuit configurations and in the case of intrinsic evolution, every chromosome is downloaded in the hardware and physical testing measures fitness [Greenwood and Tyrrell (2006)]. An evolvable motherboard is a reconfigurable circuit used for intrinsic evolution, where the evolved circuit can be inspected. Some previous work on evolvable hardware and evolvable motherboard is described here, this inspired the evolution-in-materio experiments of this thesis. The evolution-in-materio experiments of this thesis have also used an evolvable motherboard named Mecobo which is described in next chapter.

This thesis has solved many computational problems using physical materials, thus much of the work relates to the broader field of physical computation. A survey of related work performed on physical computation by many researchers is described in this chapter.

A conceptual overview of evolution-in-materio is given in this chapter. In addition, a survey and discussion of past work in evolution-in-materio are also given here so that the work of this thesis can be placed in the context of related published work concerned with evolution-in-materio.

3.1 Evolvable Hardware

In 1990, research was performed by applying an evolutionary algorithm on a computer chip to change hardware functionality and connections of the circuit dynamically [Haddow and Tyrrell (2011)]. The combination of the evolutionary algorithm with programmable electronics, such as FPGAs and field programmable analogue arrays, started a new field in the evolutionary community. It is called 'evolvable hardware'.

Thompson performed many experiments on evolvable hardware using a Xilinx 6216 FPGA (shown in Figure 3.1) [Thompson (1998)].

A Xilinx 6216 FPGA has a two-dimensional array of 64 X 64 reconfigurable logical cells. Each cell contains a function unit which can be configured to perform multiplexer functions of three inputs or Boolean functions of two inputs. Each cell can be connected with four neighbouring cells: north, east, west, south (NEWS). The three inputs (all inputs are not necessary) of a function unit can be sourced by any of the four NEWS neighbours. The output of a cell in each of the NEWS directions can be driven by the signal arriving at any one of the other three NEWS directions or by the output 'F' (output 'F' is shown in Figure 3.1) of its corresponding function unit. A computer controls the FPGA using a software, which determines a circuit design using a string of bits. The string of bits can be changed using the software. The circuit designed by one string is different from the circuit of another string. Though the software determines the circuit design using a string of bits, the circuit is physically instantiated on the chip. The FPGA configuration was evolved using a genetic algorithm, but the fitness was evaluated using the real circuit generated genetically.

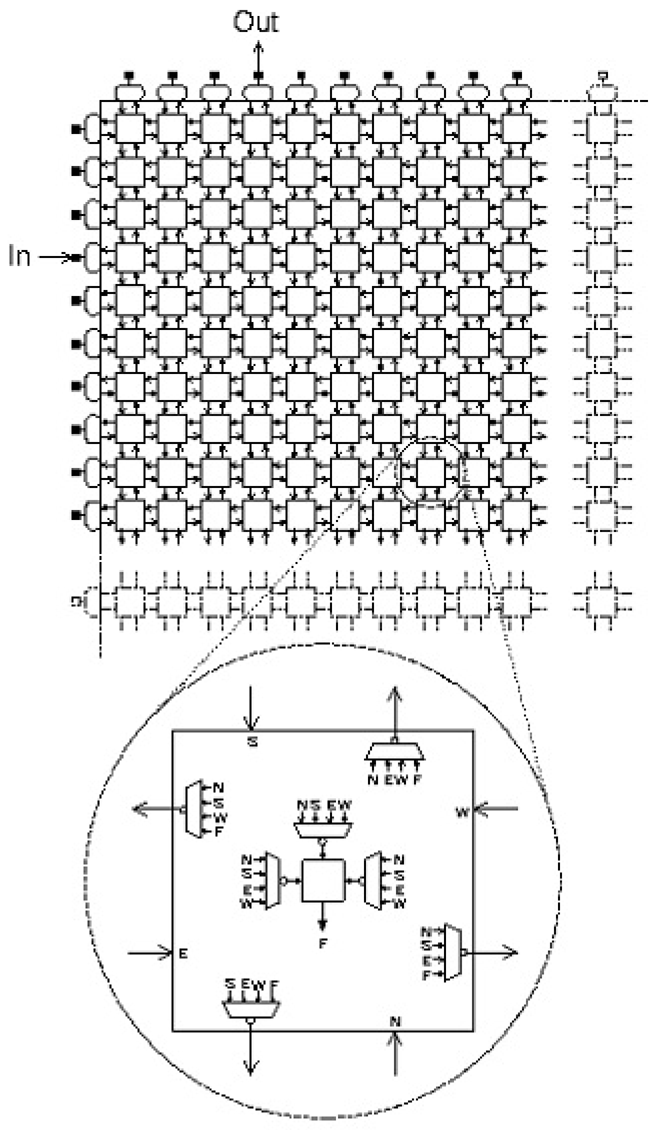


FIGURE 3.1: A simplified view of Xilinx 6216 FPGA [Thompson (1998)].

In one experiment, he evolved oscillators. He used a genetic algorithm with population size 50. He used a 10 X 10 array of cells in chip and 1800 bits of string to configure. He ran up to 40 generations with target frequencies 10 Hz, 1 KHz, 100 KHz, 1 MHz. It was found that the evolved individual obtained the desired behaviour over a period of two seconds.

He also evolved a robot with wall avoidance behaviour using the FPGA. His robot had two sensors (one pointing to the left and another pointing to the right), two motors and a RAM chip that implemented a dynamic state machine. The contents of RAM chip were generated by evolution. The input address lines of RAM chip were connected with sonar outputs, and the outputs of the RAM chip were connected directly to the motors.

Two clocks were used to transfer the output of the state machine to the inputs of the state machine and to control the motor speeds. The evolution defined the operating frequencies of those clocks. A genetic algorithm was used in the experiment. The population size was 30. The fitness score was calculated using Equation 3.1. If the distances from centre of the room to the x and y directions are $c_x(t)$ and $c_y(t)$ respectively at time t, then after an evaluation for T seconds, the fitness score can be defined as

$$fitness = \frac{1}{T}\int_0^T (e^{-k_x c_x(t)^2} + e^{-k_y c_y(t)^2} - s(t)) \tag{3.1}$$

Here s(t) = 1 when robot is stationary, otherwise 0. k_x and k_y were chosen in a way so that their respective Gaussian terms would have values in a range [0.1, 1.0]. The values were 1.0 when the robot was in the centre of the room and 0.1 when the robot hit a wall in their respective directions. Each individual was evaluated for four trials of 30 seconds each. Each time the starting position and orientation were different. The worst of the four scores was taken as the fitness. In the final few generations, the evaluations were extended to 90 seconds.

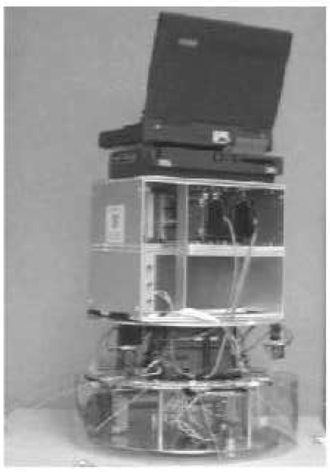


FIGURE 3.2: Thompson's robot [Thompson (1998)].

At first, the experiment was not performed in the real world. The sensor readings were given by the computer. Some noise was included to simulate a realistic environment. The movement of the robot was calculated using motor speeds. Later on, the robot was moved into the real world after evolving up to 35 generations. The sonar readings were measured from real sensors. The robot used in this experiment is shown in Figure 3.2. The behaviour of the robot is shown in Figure 3.3.

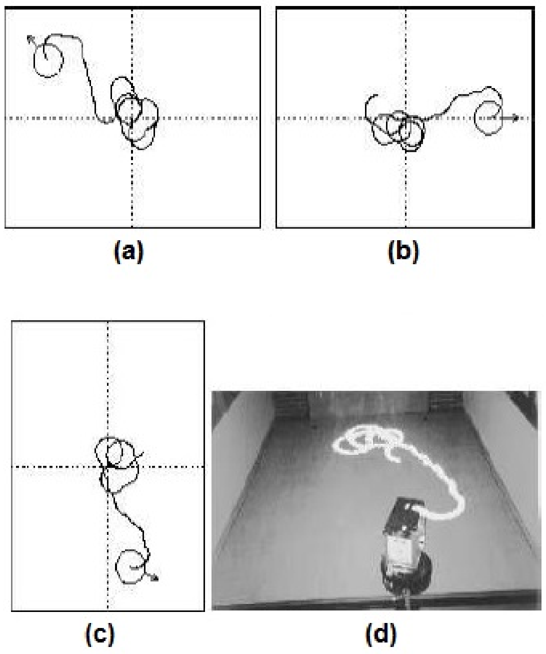


FIGURE 3.3: (a)-(c) Wall avoidance behaviour of robot in virtual reality [Thompson (1998)]. (d) Wall avoidance behaviour of robot in the real world.

In another somewhat famous experiment, he evolved a tone discriminator using the FPGA, where he tried to discriminate two frequencies: 1 KHz and 10 KHz. He tried to obtain an output of +5V for one input frequency and 0V for another. He used a genetic algorithm, and the population size was 50. He used a 10 X 10 array of cells, which required 1800 bits to configure. He used inputs with five 500 milliseconds bursts of the 1 KHz square wave and five of the 10 KHz square wave. Those ten test tones were shuffled randomly and changed every time. There was no gap between those test tones. The fitness function used in this experiment is shown in Equation 3.2, where the output at the end of test tone t is i_t, S_1 is the set of five 1 KHz test tones and S_{10} is the set of five 10 KHz test tones. After generation 3500, almost perfect behaviour was observed. However, there were some infrequent spikes in the output. At generation 4100, those were eliminated. The genetic algorithm was let to run for another 1000 generations. Finally, at generation 5000, a desired output was observed by eye on the oscilloscope.

$$fitness = \frac{1}{5}\left|\left(k_1 \sum_{t\in S_1} i_t\right) - \left(k_2 \sum_{t\in S_{10}} i_t\right)\right| \tag{3.2}$$

where k_1, $k_2 \approx$ 3.3e-5.

Later on, the artificial evolution of this tone discriminator experiment was analysed. The same solution did not work in different temperatures and also in different chips. It was discovered that the physical properties of the chip had been utilised [Thompson and Layzell (1999)]. The concept of evolution-in-materio grew out from this observation result of this tone discriminator experiment.

Tyrrell et al. evolved intrinsically an FPGA-based controller for a mobile robot [Tyrrell *et al.* (2004)]. The controller was made up of look up tables which mapped sensor data to an actuator to give commands to the robot's motors. The evolution was performed using a genetic algorithm. A continuous evolutionary process was used, which was not stopped when any member was chosen for implementation. This continuous evolutionary process was helpful for coping with environment changes and faults. The fitness value might drop at the time of injecting a fault but later on recovered to an acceptable level. They used a robot with 8 sensors and 2 wheels.

In the experiment, they used different types of mutations: fixed mutation, adaptive mutation. In adaptive mutation, the mutation rate was decided using the fitness value. Individuals having higher fitness values would have lower mutation rates, and individuals having lower fitness values would have higher mutation rates. The evolutionary algorithm used a population which used a collection of parents and clones. At first, parents were cloned μ times, then those clones were mutated and evaluated. The parents and mutated clones were sorted using fitness values in descending order. Then from that collection, the best-fit individuals were selected for parents for next generation. The fitness was calculated using the elapsed time and the distance of the robot had travelled before it hit an obstacle. A time limit was imposed. An individual would be killed when it reached the time limit or got stuck anywhere without improving distance. Two nearby sensor values were used to decide how close the robot was to the obstacle, and then a time was provided to the robot to escape from the dead zone where the last individual was killed. Only one individual was allowed to run at a time

It was found from their experimental results that the adaptive mutation rate performed better than the fixed mutation rate and the high mutation rate gave unstable results. They performed two further different types of experiments. In one type, a fault (fault in one of the sensors) was introduced before the evolution, and in the other, the fault was introduced in the middle of the evolution. It was found from the results of their fault tolerance robot controlling experiment that the evolutionary algorithm is suitable for evolving robot controllers when a sensor fault is injected before it is evolved. It was also found that data of some sensors are not important at all, however the data of some sensors are very important to control a robot. This work has inspired some of the robot control experiments performed in this thesis in which a number of tasks were performed by a simulated robot, where one of the tasks was to cope with an injected fault.

The drawback of FPGA-based evolution is that evolved circuits cannot be inspected. However, sometimes it is necessary to access and inspect the circuit as much as possible. If the evolved circuit only functions by exploiting the non-obvious parts of the FPGA, when it is rebuilt, it might not work or might behave differently than was observed at the time of evolution, which inspired Layzell to design an evolvable motherboard [Layzell (1998, 2001)]. An evolvable motherboard is a circuit which can be used to investigate intrinsic evolution. The evolvable motherboard makes a circuit able to be rewired under computer control.

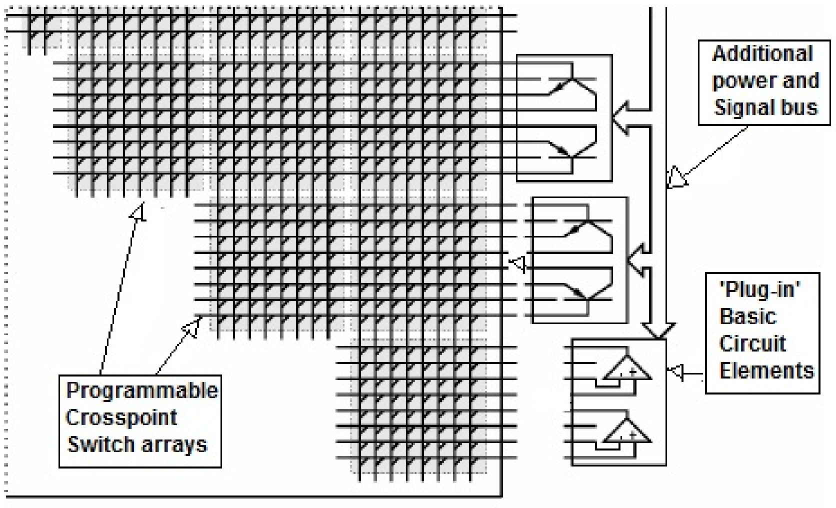

Additional power and Signal bus
'Plug-in' Basic Circuit Elements
Programmable Crosspoint Switch arrays

FIGURE 3.4: Layzell's evolvable motherboard [Layzell (1998)].

In his evolvable motherboard, there are some vertical and horizontal wires. There is an array of crosspoint switches in the motherboard. If one switch is closed, one vertical wire is connected with a horizontal wire. The switches are controlled by a computer via external components such as transistors and capacitors. Every external connection can be connected with every external connection by arranging the switches as a triangular fashion. His evolvable motherboard is shown in Figure 3.4.

His first experiment was to evolve a NOT gate (inverter). One oscilloscope was connected to observe the result. After 25,000 generations, the NOT gate was obtained. However, when the oscilloscope was removed, the NOT gate stopped working. His next experiment was to evolve an oscillator. It was noticed that the output frequency was changed when transistors of the functioning solutions were substituted with nominally identical ones. This means that the evolution depended on the specific electrical characteristics of the components. It was also observed that some of the evolved solutions exploited the features of the environment. Some of the oscillators did not work when a soldering iron was disconnected from the mains, which was located several metres away from the evolvable motherboard. Some other solutions of this oscillator experiment evolved radio receivers which picked up oscillations at the correct frequency. Evolution used the copper tracks and other components of the circuit to form aerials. This is how the signals were connected with the output. The influences of these features of the environment on the solutions of the experiments were unwanted and also possibly would not have been considered and used by any human. However, these observations and outcomes of these oscillator experiments proved that the physical properties of a system could be exploited via evolution, which inspired the work of evolution-in-materio.

Harding designed a liquid crystal evolvable motherboard for evolving liquid crystal, which is discussed in Section 3.3.1.

3.2 Physical Computation

Evolution-in-materio is a new concept or idea, but research related to it has been being performed since 1940. Although it is not directly related to evolution-in-materio, there are many similarities between this research and evolution-in-materio. This research is described in this section before moving to the discussion regarding evolution-in-materio research performed so far. It should be noted that the evolution-in-materio research performed in this thesis is described in Chapters 5-8, and this chapter only contains the evolution-in-materio research which was performed by other researchers.

In the late 1940s, Ashby demonstrated his homeostat machine which was a mixed (digital and analogue) signal machine [Ashby (1960)]. There were a number of interconnected electromechanical units in that machine, which communicated with each other and showed collective adaptive behaviour.

In the late 1950, Pask constructed electrochemical devices, where he used various aqueous solutions of ferrous sulphate as the chemical material through which he passed current [Cariani (1993)]. He wanted to make an analogue control system. His control system could construct its own sensors.

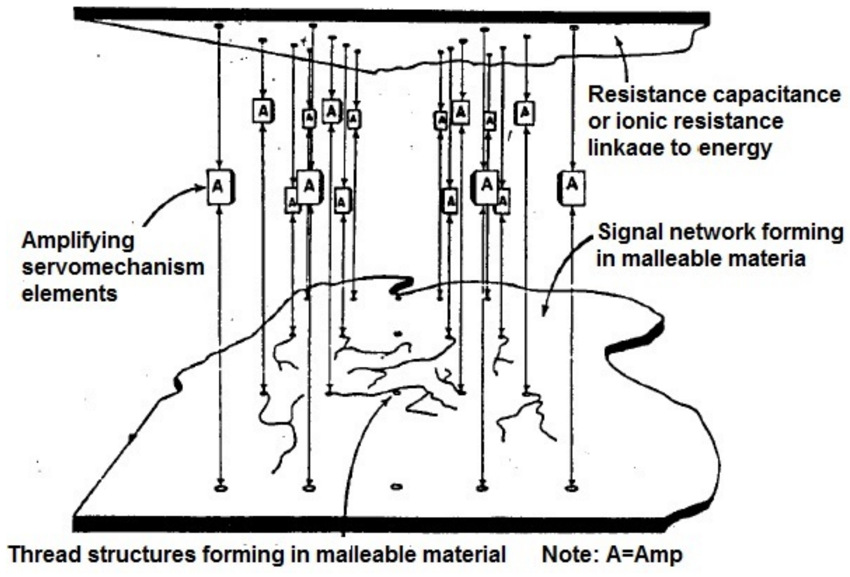

Resistance capacitance or ionic resistance linkage to energy
Amplifying servomechanism elements
Signal network forming in malleable materia
Thread structures forming in malleable material
Note: A=Amp
A

FIGURE 3.5: Pask's experiment [Cariani (1993)].

He used an array of platinum electrodes through which he passed current to ferrous sulphate medium. He found some dendritic metallic threads grew in the chemical medium. The growth of threads was controlled by choosing electrodes and the current flowing through them.

Later on, he made an 'ear', which worked as a tone discriminator, i.e. it could discriminate two frequencies, one was the order of 50 cycles/second and another one was the order of 100 cycles/second. The training procedure took half a day and then finally the ear could recognise and discriminate sounds. Actually, it was a gap in the thread structure, in which there were fibrils, which could resonate at the exciting frequency. His experiment is shown in Figure 3.5.

Walter made three-wheeled tortoise robots [Walter (1953)]. Those robots could turn their front wheels. A light sensor and some vacuum tubes (or two neurons) were used in the controller. That means, a physical system interacted with the environment via those robots.

Beer tried to make an adaptive biological computer, which was based on ecosystem, i.e. a tank holding fishes [Beer (1962)]. The fishes were fed iron particles and interacted using electromagnetism and light.

Stewart pointed out some properties of evolution-in-materio, such as the system cannot be inspected and opened, a 'bulk' production process might be needed [Stewart (1969)]. He performed a similar experiment to Pask's experiment, which was named as linear field trainable experiment. He proposed a machine similar to modern programmable chips. His device was a high pressure (100-2000 psi) container containing nitric acid in which gold or iron particles were placed. It was designed to work as a pattern recognition system. However, the target of the experiment was to produce a number of different behaviours such as threshold functions.

An analogue computer is a system which makes a model of the solution of a problem by using continuously changeable aspects of physical phenomena, such as mechanical, electrical quantities. Mills constructed a version of analogue computer named Kirchhoff-Lukasiewicz machine, which uses physical material for computation. It is made of logical function units that are connected to conductive sheets (usually polymer sheets) [Mills (1995b,c,a); Mills *et al.* (2006, 1990)]. Different computational applications could be addressed using this system, such as models of biological systems, robot controlling, radiosity-based image rendering, and control of a cyclotron beam. This device is much faster in solving partial differential equations than the conventional computer, but the speed depends on the physical materials used and the interfacing to them. The architecture of this system consists of a conductive sheet, a number of arrays of fuzzy logic function units, inputs, outputs, current sources and current sinks. These are connected by a reconfigurable array of wires. Inputs are obtained from digital-to-analog converters (DAC), potentiometers or sensors and outputs are measured directly, or via analog-to-digital converters (ADC). Different versions of this system have been developed. Of these, three different versions have been developed using unconventional non-silicon designs. In networked version of this analogue computer, a socket was used, which permits different components to be connected to the analogue computer. A VLSI chip and Jell-O® brand gelatin were used. Organic semiconductors, cultured neural tissue, and other materials could be used instead. A prototype 3D analog computer was built with a mixture of unflavored Jell-O® brand gelatin and sodium chloride. The mixture had a 3 X 3 X 3 grid of electrodes on non-conductive plastic rods molded into it. A number of different experiments were performed using this system. The 3D analogue computer is shown in Figure 3.6. The USB-networked analogue computer was designed in the year 2004, where the configurations of the system were evolved using genetic algorithms. Though a number of efficient configurations were obtained just after a few trials, the obtained results were limited to connections manually placed in advance.

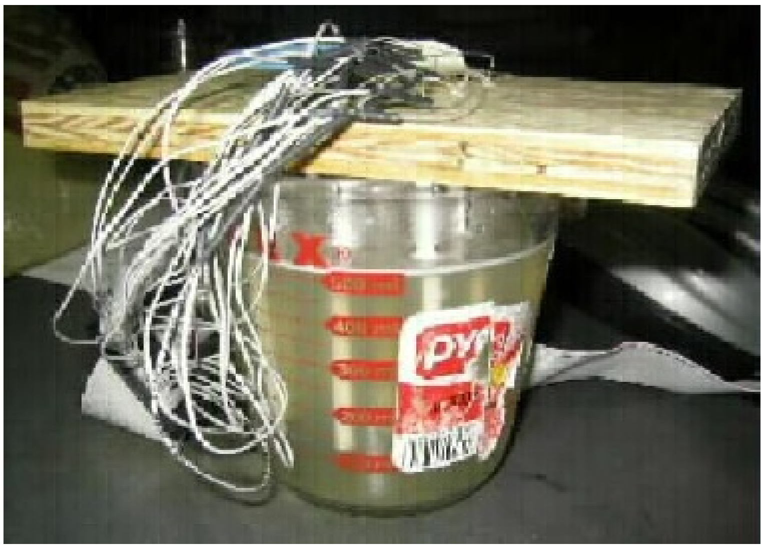


FIGURE 3.6: The prototype 3D analogue computer [Mills *et al.* (2006)].

Reaction-diffusion computer is another example of an analogue computer [Adamatzky (2009)] which uses physical material for computation. Adamatzky explored this system. This reaction-diffusion system defines mathematical models using the changes of spatially distributed concentrations of chemicals with the influences of reaction and diffusion of the local chemical. This computer can solve many problems, such as large scale Voronoi diagram construction, which is an NP-complete problem. This system can also be controlled by electric fields.

Though all these experiments (stated above) of in-materio and physical computation did not use any evolution in solving problems, still these experiments have similarities with evolution-in-materio. Evolution-in-materio exploits physical systems to solve problems and these experiments also solved problems by exploiting physical systems. In work on physical computation in the past there was a system that was manipulated by a person. It is reasonably straightforward to adapt such systems for evolutionary control. Thus they could potentially be accessible to evolution-in-materio.

3.3 Evolution-In-Materio

Natural evolution can be interpreted as an algorithm which exploits the physical properties of materials. Evolution-in-materio has the aim to mimic this by manipulating physical systems via computer controlled evolution. That means, evolution-in-materio is a method which uses artificial evolution to exploit properties of materials in order to solve computational problems without having a detailed understanding of such properties. When properties of a material (such as liquid crystal, carbon nanotubes) are unknown and very difficult to exploit by human to solve problems, evolution-in-materio becomes a suitable way to exploit such properties.

Evolution-in-materio uses a hybrid system involving both a digital computer and a physical material [Miller and Downing (2002); Miller *et al.* (2014)]. In the physical domain, there is a material to which physical signals can be applied or measured. These signals are input signals, output signals and configuration inputs. A computer controls the application of physical inputs to the material, the reading of physical signals from the material and the application to the material of other physical inputs known as configuration inputs. The input data is transformed into physical inputs and applied to the material. A genotype of numerical data is held on the computer and is transformed into configuration inputs. The genotypes are subject to an evolutionary algorithm. Physical signals (output signals) are read from the material and converted to output data in the computer. A fitness value is calculated from the output data and supplied as a fitness of a genotype to the evolutionary algorithm. The conceptual overview of evolution-in-materio is shown in Figure 3.7.

The interesting feature of evolution-in-materio is that the used evolutionary algorithm may increase evolvability by exploiting the unknown physical variables in a material. Natural evolution operates in the physical world and exploits the physical properties of materials (mainly proteins). Banzhaf et al. discussed the importance of physicality and embodiment [Banzhaf *et al.* (2006)]. In spite of this, very few attempts have been taken to date to include materials in the evolutionary process. The few previous attempts are described in Sections 3.3.1 and 3.3.2.

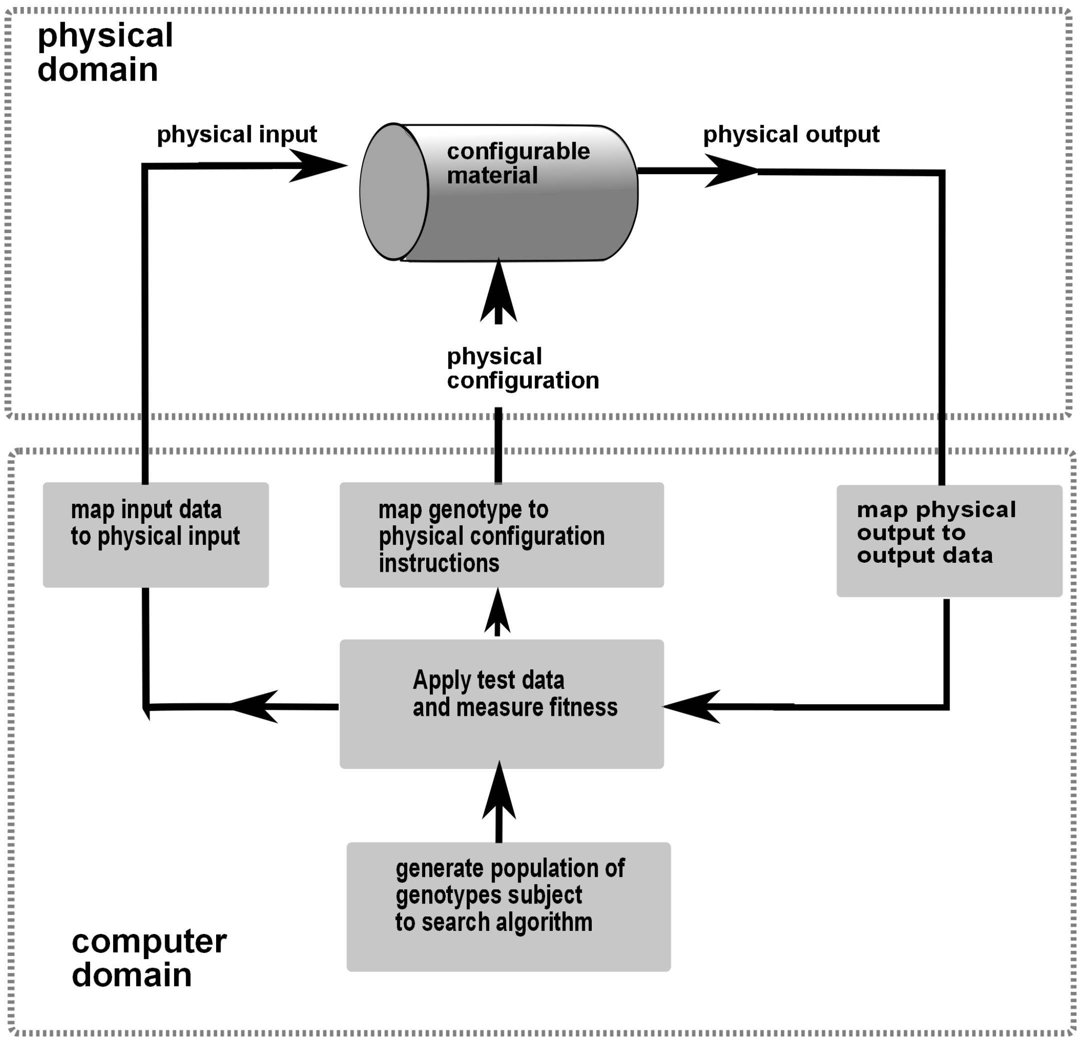

physical domain
physical input
configurable material
physical output
physical configuration
map input data to physical input
map genotype to physical configuration instructions
map physical output to output data
Apply test data and measure fitness
generate population of genotypes subject to search algorithm
computer domain

FIGURE 3.7: Concept of evolution-in-materio [Miller *et al.* (2014)].

3.3.1 Past Research in Evolution-In-Materio

In the year 2002, Miller and Downing suggested a configurable analogue processor as an evolutionary exploitable device [Miller and Downing (2002)]. It is a physical device, whose configuration data is supplied by a computer in many formats such as analogue signals (voltages, magnetic fields, mechanical displacements). The incident signal is modified using configuration data. The incident signals and modified signals can be voltages, radiations, vibration, sound, airflow. The modified signal is tested and depending on the response, fitness is calculated and depending on the fitness score, configuration data is decided. The diagram of a configurable analogue processor is shown in Figure 3.8.

They also suggested a device called field programmable matter array in which applied voltages may cause physical changes to a material. These changes interact with other

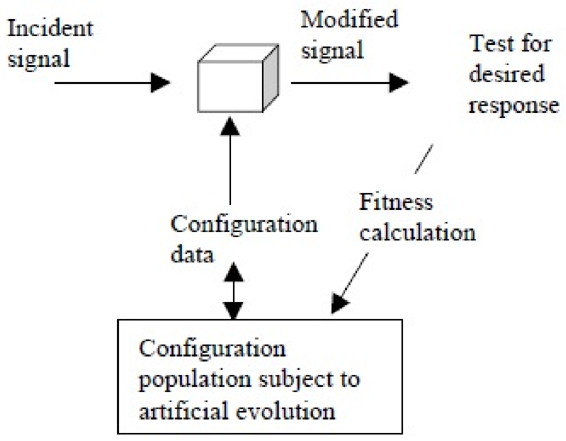

Incident signal
Modified signal
Test for desired response
Fitness calculation
Configuration data
Configuration population subject to artificial evolution

FIGURE 3.8: The configurable analogue processor [Miller and Downing (2002)].

voltage-induced configurations in a physical substrate in unexpected ways. The field programmable matter array is shown in Figure 3.9. Thompson's FPGA or field programmable matter array is a form of configurable analogue processor. There are many difficulties regarding configurable analogue processor, such as sensitivity to the properties of the material (each configurable analogue processor will require separate training), evolutionary algorithms may exploit any physical properties of the part of the training setup.

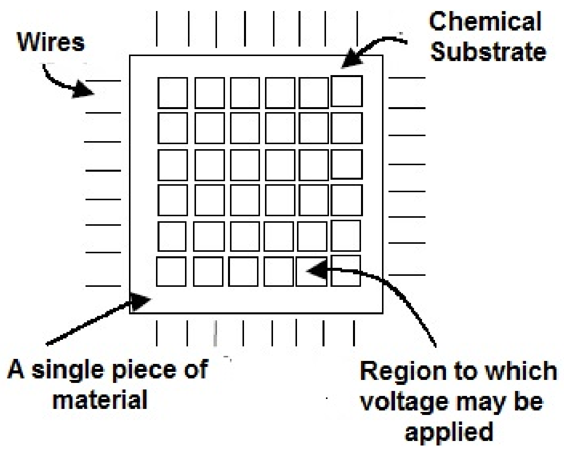

Wires
Chemical Substrate
A single piece of material
Region to which voltage may be applied

FIGURE 3.9: The field programmable matter array [Miller and Downing (2002)].

Not all materials may be suitable for evolution-in-materio. Miller et al. suggested some guidelines for choosing materials [Miller and Downing (2002); Miller *et al.* (2014)]. The material needs to be reconfigurable, i.e. it can be evolved over many configurations to obtain desired response. It is important for a physical material to be able to be 'reset' in some way before applying new input signals to it, otherwise it might preserve some memory from the past behaviour and might give fitness scores that are dependent on the memory. It is preferable that the material should be physically configured using small voltages and can be manipulable at a molecular level. They suggested liquid crystal, especially LCD as a good candidate.

Mahdavi and Bentley investigated smart materials for their robotic hardware in a project called MOBIUS [Mahdavi and Bentley (2003)]. Nitinol wire is an alloy which is made up of nickel and titanium. The wires were used as the muscle hardware for a robotic snake. An evolutionary algorithm was applied to evolve the activations for those wires to control the snake-like movement of the robot. When one wire broke in the middle of the experiment, it could adapt to the failure. Nitinol alloys are super elastic, which have shape memory. It can have two forms at different temperatures. If temperature changes, it causes phase transformation. Both the evolutionary algorithm and the properties of Nitinol alloy are suitable for snake-like robots. This combination is also suitable for making devices that need to change their morphology with environment changes.

Oltean suggested a switchable glass (smart windows) as a suitable material [Oltean (2006)]. This switchable glass controls transmission of light through windows by applying different voltages. He suggested liquid crystal, electrochromic devices and suspended particle devices. Rods suspended in a fluid can be an example of suspended particle devices. In the case of electrochromic devices, the translucency of the glass might be controlled by applying different voltages. In the case of suspended particle devices, the rods align to an electric field and variations of voltages control the amount of light transmitted.

Harding and Miller performed many experiments with LCD using evolution-in-materio [Harding and Miller (2004b,a, 2005); Harding (2006); Harding and Miller (2007)]. They used liquid crystal evolvable motherboard to exploit the properties of the liquid crystal. The motherboard is shown in Figure 3.10

The liquid crystal evolvable motherboard is a circuit which uses four cross-switch matrix devices to dynamically configure circuits connecting to the LCD. In this motherboard, there is an LCD in the middle of the board. Eight external connections and four 8 X 16 analogue switch arrays are used in this motherboard, where the switches are used to wire one of the 8 external connections to 64 connections (32 per side) on the LCD, i.e. each external connection can be connected with any of the connections of the

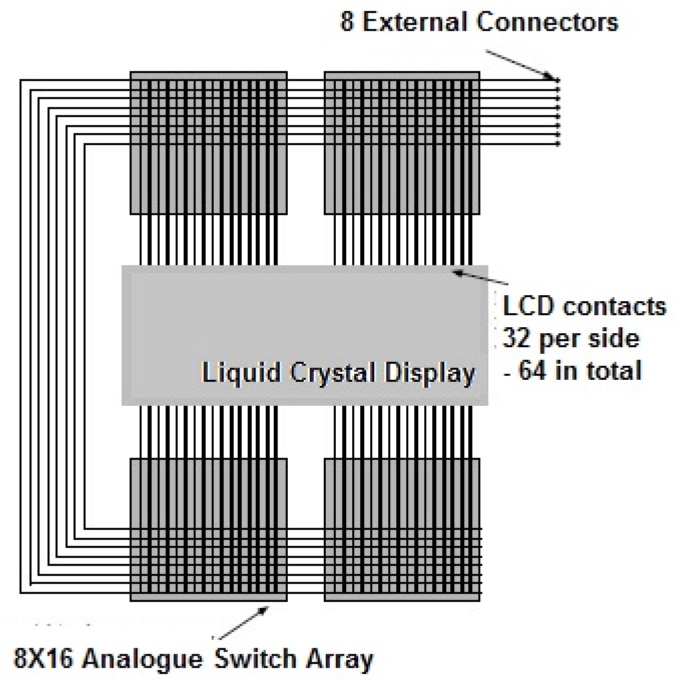


FIGURE 3.10: The liquid crystal evolvable motherboard [Harding and Miller (2004b)].

LCD. The connections are used to provide incident signals, configuration inputs, an electric ground and signals output from the LCD. The external connectors connect the motherboard to the computer. The incident signals and configuration inputs are passed from the computer to LCD and the output signals obtained from LCD are passed to the computer. The board is programmed from the computer by controlling each of the programming lines on the switch arrays using digital values. It takes $\approx$ 1 second to program all four switch arrays.

Their first experiment was to obtain the non-linear behaviour of liquid crystal. They applied a sequence of input voltages and measured the output voltages. Sometimes they found a non-linear step change in the observed response. Their second experiment was to evolve a transistor. They tried to find a point at which the sudden output change occurred (i.e. switching behaviour). They tried to obtain the output changes when the input voltage was 2V. Evolution showed that although no step changes occurred exactly at 2V, step changes were possible near to 2V.

Their third experiment was to evolve a tone discriminator using LCD, where they tried to identify 100 Hz and 5 KHz square waves, having high output for one and low for other. Each signal oscillated between 0V and 5V, equal timing for low and high states.

The tones were used in 250 milliseconds bursts and there was no gap between two tones. The target was output < 0.1V for low frequency (100 Hz) and output > 0.1V for high frequency (5 KHz). The fitness was calculated using Equation 3.3.

$$fitness = \frac{\sum_{i=1}^{L} x(i)}{L} \tag{3.3}$$

Here,

$$x(i) = \begin{cases} \\ 1, & S[i] \leq t \text{ and } O[i] = HIGH \\ 1, & S[i] \geq t \text{ and } O[i] = LOW \\ 0, & Otherwise \end{cases} \tag{3.4}$$

where S is the vector containing the samples obtained from the LCD and L is the length of S. The i^{th} element of S is $S[i]$. O is a vector containing the frequencies (to be discriminated by the LCD) at a given time. The frequency can be either LOW or HIGH. The i^{th} element of O is $O[i]$. t is the threshold (< 0.1V).

The success rate for this evolutionary experiment was 10% of all evolutionary runs. The output was not stable, but in most cases, output was high for high frequency and low for low frequency. Many other pairs of frequencies were also tried between 100 Hz and 4500 Hz randomly for the tone discriminator experiments. Five evolutionary runs were carried out for each pair of frequencies. The results of liquid crystal tone discriminator experiments are shown in Figure 3.11.

Their next and fourth experiment was to control a robot using liquid crystal. They used 2 input connections for 2 sonars, 2 outputs for 2 motors, 1 ground connection and 3 for configuration inputs which were evolved using a genetic algorithm. They used a simulated robot for this experiment. The readings of sonars were converted to signals and sent to the liquid crystal evolvable motherboard. Control signals were passed to motors using this motherboard. The liquid crystal processed signals and controlled the robot. Each distance sensor value was mapped with square wave frequency proportional to the distance between the sensor and the obstacle with a range between 1 Hz (for the near object) to 5000 Hz (for the far object). No artificial noise was used, but the process of making square waves via computer added some noises and timing problems. Fifty milliseconds delay might be expected between a distance measure and the frequency

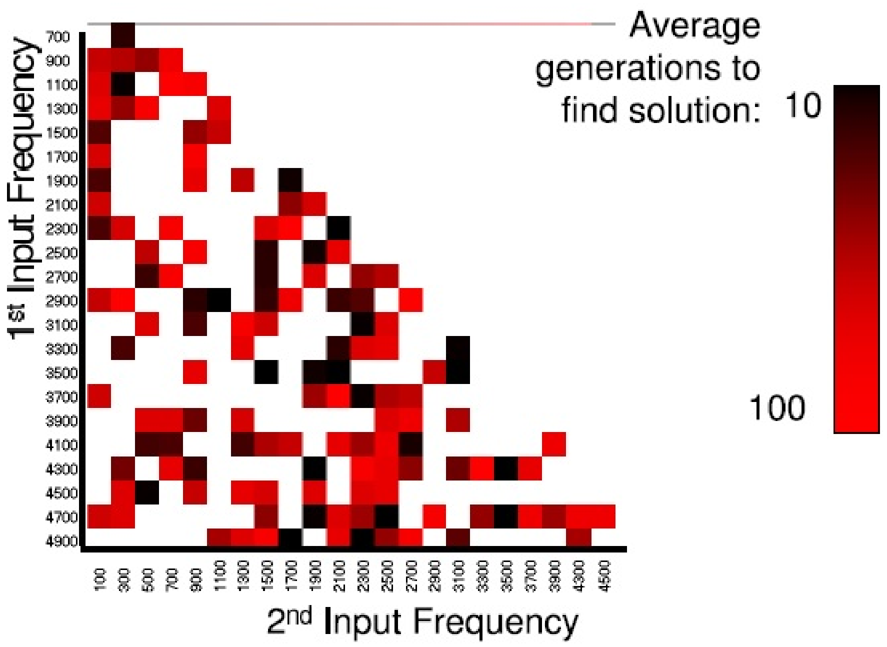


FIGURE 3.11: The results of liquid crystal tone discriminator experiments [Harding (2006)].

change. Two sonars were connected, 30 degrees separated from each other. Two motors were connected in two sides of the robot. The motors were driven by the output voltage values obtained from the liquid crystal. A motor was switched on when the corresponding output voltage value was high (above 3.0V) and the motor was in low speed when the corresponding output was low. When two outputs were high, the robot moved forward and when both were off, the robot was stationary. If only one motor was enabled, the robot turned. The liquid crystal robot controller is shown in Figure 3.12. They built a fitness map to obtain fitness scores for their robot controlling experiment, where each area of the map was assigned an absolute measure of the difficulty in reaching that area. The value of each area of the fitness map was calculated by modelling chemical diffusion within the environment.

Two maps were used in this experiment. For the first map, a good solution was obtained in generation 62 on average and the lowest number of generations required to obtain a good solution was 22. The success rate was 36%. The evolved robot was then tested in the second map and 35% of evolved solutions could explore the second map. Two maps of the robot controlling experiment are shown in Figure 3.13.

They also evolved liquid crystal to obtain digital circuits. They tried to obtain NOT,

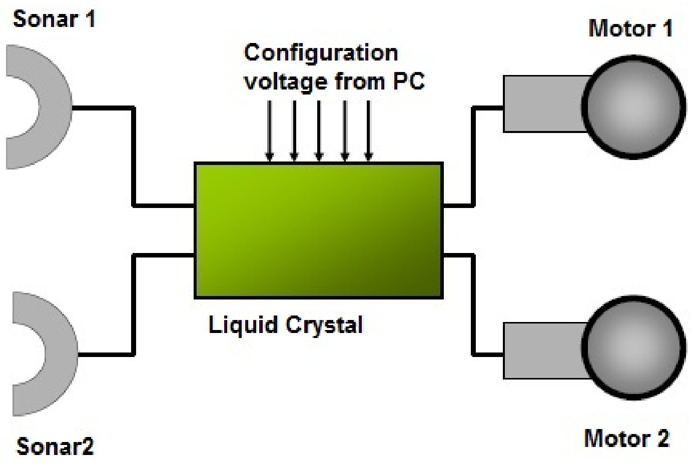


FIGURE 3.12: Liquid crystal robot controller [Harding and Miller (2005)]

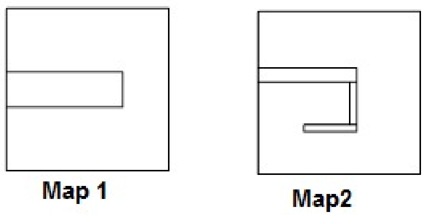


FIGURE 3.13: Maps used in robot controlling experiment by evolution-in-materio using an LCD [Harding and Miller (2005)]

AND NAND, OR, NOR and XOR gates. They used two input voltages: +1V for Boolean 1 and 0V for Boolean 0. They used 0.1V as output threshold, i.e. if the output was greater than +0.1V, they defined it as Boolean 1, otherwise Boolean 0. As they used two voltages, each gate had four combinations of inputs in the truth table. They evaluated fitness by using each row of the table three times. So, in total 12 fitness evaluations were performed for each gate. Any fitness >=10 was considered as a good result, which showed that each row of the table was true at least once. A perfect one had score 12. XOR was the hardest of all. For all gates, solutions were found within 40 generations on average.

Harding performed some stability tests on LCD to find out how stable and reliable the solutions were [Harding (2006)]. He used the same configurations as the robot controlling experiment, where the absolute fitness values were used for the measurement of the tests. In the first experiment, he evaluated the same individual 10 times and observed the fitness scores when they acquired high values. In the second experiment, he loaded other intermediate individuals (modified the configurations of liquid crystal) in between two evaluations of the same individual. In both of these cases, the fitness values were found to be degraded, which showed the poor performance of stability tests of liquid crystal.

Other than the logic gate experiments of Harding and Miller, Toth et al. also used evolutionary algorithms to implement collision-based two-input logic gates, such as AND, NAND by controlling chemical wave fragments in light sensitive Belousov-Zhabotinsky (BZ) reaction [Toth *et al.* (2008, 2011)]. Bull et al. also controlled chemical-wave fragments in light-sensitive BZ reactions using an accuracy-based learning classifier system [Bull *et al.* (2008)]. They also showed that a learning classifier system is able to control the electrical stimulation of cultured neuronal networks so that they can display elementary learning. They used multi-electrode arrays with pyramidal electrodes (40 X 40 X 70 μm, spaced on 200 μm) to record electrical activity of the hen embryo brain spheroids. The image of the electrode array of this experiment is shown in Figure 3.14. The recording from the spheroids was performed with a 60-channel data acquisition system and the output sampling frequency of each channel was 25 KHz. The results of this experiment showed that the learned stimulation protocols could identify seemingly fundamental properties of in vitro neuronal networks.

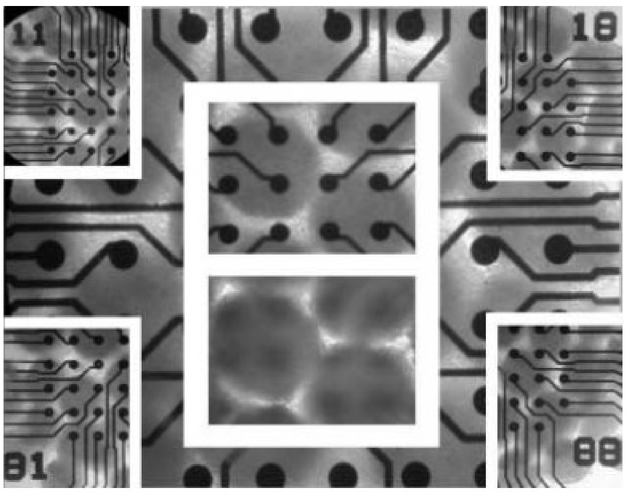


FIGURE 3.14: Composite picture of phase contrast microscopy images taken at various optical magnifications and focal planes of aggregate cell cultures on multi-electrode array dish [Bull *et al.* (2008)].

Roselló-Merino et al. also implemented two-input logic gates (AND, OR, NAND, NOR, XOR) using various radio-frequency pulse parameters such as pulse amplitude, frequency, duration, phase [Roselló-Merino *et al.* (2010)]. In one methodology, pulsed gradient spin echo sequences were used. In this case, the first input was decided using the characteristic as to whether the starting pulse was supplied with the resonance (characteristic frequency) or not. If resonance was on, it was considered as 0, otherwise 1. The second input was the delay of reading the radio frequency response, which was recognised as data acquisition delay. If data acquisition started immediately when the first input was 0, second input was considered as 0. If some time was taken for data acquisition when the first input was 0, second input was 1. If the first input was 1 and data acquisition was 0, second input was 0, otherwise second input was 1. In another method, the integral of spectral intensity was used to determine output. If the total integral was 0, output was considered as 0, otherwise 1.

Bechmann et al. used real-valued numbers (values between 0 and 1) for inputs and outputs for implementing 'continuous logic operation' using continuous spin dynamics [Bechmann *et al.* (2010)]. They defined some simple continuous gates such as sin/ sin, sin/ sinc. Cartesian genetic programming was used to evolve circuits that were made of these simple continuous gates.

Other than the logic operations, Theis et al. performed experiments on how a genetic algorithm could improve the functionality of an amphiphilic chemical system [Theis *et al.* (2006)]. After preparing a number of amphiphilic chemicals, those were mixed with sucrose solution. For each genotype, three identical mixtures were prepared. The turbidity was measured for each mixture. Amphiphiles form vesicles in an appropriate solvent. The fitness score of each genotype was calculated from the difference between the mean turbidity and the standard deviation over the three mixtures. In the experiment, the population size was 30 and the number of generations was 5. After the experiment, it was found that it generated 180 recipes. In this experiment, it was possible to learn about the interactions of the chemicals by analysing which recipes obtained the highest fitness scores.

The past work of evolution-in-materio has established the theoretical framework and methodological focus of the experiments of this thesis. Most of the problems attempted in this thesis were not attempted by any researcher using evolution-in-materio method. Different physical materials have been attempted so far, but this is the first time that mixtures of single-walled carbon nanotubes and polymers have been used in this thesis to solve problems. The previous work of evolution-in-materio of other researchers helps to identify the problems and physical materials, which were not attempted before. The tone discriminator problem which was solved by Harding and Miller in their evolution-in-materio experiment using LCD has been solved in this thesis using mixtures of single-walled carbon nanotubes and polymers. The results of tone discriminator experiments of this thesis were then compared with their results. The comparison is shown in Section 5.3. Some work has been performed using mixtures of single-walled carbon nanotubes and polymers by other researchers of this NASCENCE project. This is described in following section to give a clear overview of the work of this project.

3.3.2 Evolution-In-Materio Research in NASCENCE

Much research has been undertaken in the NASCENCE project regarding the development of evolvable hardware, the investigation on the behaviour of physical materials, the solutions of different types of computational problems using evolution-in-materio.

Lykkebø et al. developed the hardware platform named Mecobo and the driver software (the description of Mecobo hardware and driver software are given in Chapter 4), which have been used in this research [Lykkebø*et al.* (2014)]. They solved logic gates (AND, OR, NAND, NOR, XOR) using Mecobo (version 3.0) and a mixture of single-walled carbon nanotubes and PMMA, where they used 2 electrodes as inputs, 1 electrode as output and 9 electrodes for configuration inputs. They used two approaches to solving logic gates. In both of the approaches, they used static digital voltages for inputs, where the amplitude of the input signal was used for input mapping (input voltage 3.5V was Boolean 1 and input voltage 0V was Boolean 0). They determined the Boolean output value with a threshold by observing the digital values of the buffer of the output electrode. In the case of first approach, they found, by exhaustive search, all types of logic gates (AND, OR, NAND, NOR, XOR) and used digital static signals for configuration inputs. In the case of second approach, they evolved an XOR gate using digital square wave signals for configuration inputs having a frequency in a range [40 Hz, 25 MHz]. In this case, they used a genetic algorithm with 25 individuals, two crossover points and tournament selection having 5 individuals as elite, where almost perfect XOR gate was found after 150 generations.

Massey et al. and Volpati et al. investigated the behaviours of physical materials, where they used single-walled carbon nanotubes and liquid crystal as physical materials

[Massey *et al.* (2015b,a); Volpati *et al.* (2015)]. Kotsialos et al. solved simple threshold logic gates and more complicated circuits using single-walled carbon nanotubes and PMMA with static analogue voltages [Kotsialos *et al.* (2014)]. The training of solving those logic gates and circuits was formulated as an optimisation problem with binary and continuous constraints and was solved by two derivative-free algorithms such as the Nelder-Mead and the differential evolution (DE) algorithms. They evolved AND, OR and XOR gates using a mixture of 0.23% single-walled carbon nanotubes with PMMA, where they used four configuration inputs. They used single threshold for determining outputs of AND and OR gates (the threshold of OR gate was smaller than the threshold of AND gate) and two thresholds for the output of XOR gate. Other than the AND, OR and XOR gates, they evolved half adder, full adder, (AB, A+B) and (AB+BC) circuits using two mixtures of physical materials and these were 0.23% single-walled carbon nanotubes with PMMA and 0.53% single-walled carbon nanotubes with PMMA. In the case of (AB, A+B) circuit, they used three configuration inputs, two inputs (A and B) and two outputs containing results of AB and (A+B). In this circuit, they used a single threshold for both outputs (i.e the same threshold for both outputs). In the case of half adder, they used two inputs, two outputs and three configuration inputs. The half adder, which was made of AND gate and XOR gate, used one output threshold for AND gate and two output thresholds for XOR gate. In the case of (AB+BC) circuit, they used three configuration inputs, three inputs (A, B and C) and one single output deriving from equation (AB+BC). In this circuit, they used only one threshold for output. In the case of full adder, they used two configuration inputs, three inputs and two outputs. One output (the carry output) was determined by only one threshold, where the other one (the output containing the sum) was determined by three thresholds. The hardware platform that they used in all of their experiments was an mbed microcontroller (NXP LPC1768 system) together with some additional electronics (ADC, DAC, operational amplifier).

Other than the logic gates and circuits, Clegg et al. solved travelling salesman problems with 9, 10 and 11 cities using static analogue voltages by evolution-in-materio [Clegg *et al.* (2014b)]. They used 0.1% single-walled carbon nanotubes with PMMA for solving these problems. They performed many experiments with different numbers of configuration inputs to obtain the best result. They used (1+4)-evolutionary algorithm. The evolutionary run was carried out up to 1500 generations. They found that 3 configuration inputs for 9 city and 4 configuration inputs for 10 city travelling salesman problem obtained the lowest value for the average number of generations for successful runs, where 11 city travelling salesman problem could not be tried with more than 4 configuration inputs due to the limitations of the equipment. They used a data acquisition card as the hardware platform for the evolution-in-materio. A 4x4 electrode array (containing the physical material) was connected with a 16x16 analogue crosspoint switch that controlled connections from the physical material to data acquisition card. This allowed the configuration inputs to be placed anywhere in the array. They also tried to solve machine learning classification problems using the same hardware platform [Clegg *et al.* (2014a)]. They used two datasets for the classification problem and these were Banknote and Iris [Lichman (2013)]. They used two different physical materials for their classification experiments and these were 0.1% single-walled carbon nanotubes with PBMA and 0.53% single-walled carbon nanotubes with PMMA. They performed many experiments with the datasets. In all experiments, they used (1+4)-evolutionary algorithm. The evolutionary run was carried out for up to 150-300 generations. In the case of Banknote dataset, they used 4 electrodes as inputs, 2 as outputs and 4 as configuration inputs. In the case of Iris dataset, they used 3 outputs, 4 inputs and 4 configuration inputs with a range $\pm$2V. In the case of Banknote dataset, they used Matthews correlation coefficient (MCC) [Baldi *et al.* (2000)] for fitness calculation, but they did not use three way confusion matrix or MCC for fitness calculation in the case of Iris dataset as Iris dataset was evenly distributed over three output classes. It has been found from the results of the experiments using 0.53% single-walled carbon nanotubes with PMMA that the average training accuracy was $\approx$ 69% (averaged over 10 runs) in the case of Iris dataset and $\approx$ 73% (averaged over 6 runs) in the case of Banknote dataset. In the case of classification experiments using Banknote dataset and 0.1% single-walled carbon nanotubes with PBMA, the average training accuracy was $\approx$ 95% (averaged over 5 runs).

3.4 Summary

In past work in evolvable hardware and evolution-in-materio, the most common search algorithm used has been a form of evolutionary algorithm. Also, researchers have tended to use a variety of evolutionary algorithms. Consequently, so that the work of the thesis can be compared with past work, evolutionary algorithms have also been employed. In this thesis, a simple form of evolutionary algorithm (1+4-evolutionary algorithm) has been used. This was chosen partly because due to the slowness of fitness calculation, it is necessary to use an evolutionary algorithm that is recommended when the number of fitness evaluations is very restricted [Bäck *et al.* (1997)].

Past work in evolution-in-materio has investigated solving a limited set of computational problems such as robot control, tone discrimination, oscillators and logic gates. However, in engineering and computer science, there are many computational problems that are very well studied and whose complexity classes are well understood. In order to assess properly the potential of evolution-in-materio, one needs to examine how to solve such problems. Consequently, the thesis has in addition to tone discrimination and robot control investigated solving some of these well understood problems such as function optimisation, machine learning classification, bin packing, and even parity.

The experiments along with the experimental system of this thesis are described in next chapters.

Part II

The Experiments

Chapter 4

The Experimental System

Contents

The experimental system used in this research has been developed within the NASCENCE project [Broersma *et al.* (2012)]. The evolutionary algorithm and the interface software run on a host computer which communicates with the interface hardware over a USB connection. The genotype data from an evolutionary algorithm is mapped to commands that are sent to the interface. The hardware translates the received commands to electrical stimuli to send to the material. The response from the material is sampled and returned to the evolutionary algorithm by the interface software.

The full experimental system is shown as a stack in Figure 4.1, where the material is at the lowest level and the computational problem targeted by the evolutionary search algorithm is at the topmost level. The computational problem can be seen only as a specification of input data and the targeted output response at the topmost level. The next level down, the evolutionary algorithm maps the problem description to a goal function (fitness) and defines the representation of candidate solutions, i.e. genetic information. The choice of evolutionary algorithm is flexible. To solve a computational problem, one can use a number of evolutionary algorithm types (e.g. genetic algorithm, evolutionary strategy). The interface software, interface hardware and the experimental material are three basic components in the experiments in this thesis. The research in this thesis is concerned with devising experiments using evolutionary algorithms to solve a number of computational problems. This involves devising input and output mappings, methodologies of solving problems using the three basic components, some investigations and analysis of results. These are discussed in detail in following chapters.

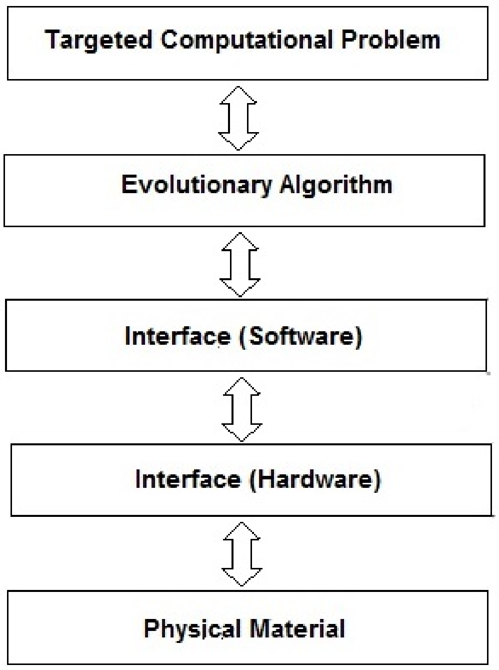


FIGURE 4.1: A view of full experimental system as a stack.

4.1 Mecobo: an Evolution-In-Materio Hardware Platform

The interface hardware, Mecobo [Lykkebø*et al.* (2014)], has been designed and built by Odd Rune Lykkebø and Gunnar Tufte in the Norwegian University of Science and Technology. Mecobo is designed to interface to a large variety of materials. The hardware allows the possibility to map input, output and configuration terminals, signal properties and output monitoring capabilities in arbitrary ways. The platform's software components, i.e. evolutionary algorithm and software stack, are as important as the hardware. Mecobo includes a flexible software platform including hardware drivers. Support of multiple programming languages and a possibility to connect to the hardware over the internet make Mecobo a highly flexible platform for evolution-in-materio experimentation.

It is important to appreciate that in evolution-in-materio, the computational substrate is a piece of material for which the appropriate physical variables to be manipulated by evolution may be poorly understood (Figure 3.7). This means that the selection of signal types, i.e. inputs, outputs and configuration inputs, assignment to I/O ports could not easily be defined in advance. Thus, interactions with the materials should be as unconstrained as possible. This means that any I/O port should be allowed by the hardware to accept any signal type. In addition, the signal properties, such as voltage or current levels, AC, DC, pulse or frequency, should be allowed to be chosen during evolution. The Mecobo platform enables the selection of the I/O port (via which signals are applied to or read from the material), signal types, the characteristics of signals possible during evolution by interface software.

Two versions of Mecobo hardware have been used in the experiments of this thesis: Mecobo 3.0 and Mecobo 3.5.

Mecobo 3.0 hardware allows only two types of inputs to the material, which are digital. The input is either a static voltage (0V or 3.5V) or a square wave signal. In the case of Mecobo 3.5, a static analogue voltage is possible other than the digital square wave signal as an input. It has analogue input in a range [-5V, 5V]. The amplitude of the input signal determines the voltage level. The interface software of Mecobo 3.5 supports amplitude values of input signals in a range [0, 255], where voltage level -5V corresponds to value 0 and voltage level 5V corresponds to value 255.

Different characteristics or input parameters associated with the inputs of Mecobo 3.0 and Mecobo 3.5 can be chosen. These input parameters are described in Table 4.1.

The start time and end time of each input signal determine how long an input is applied. Mecobo 3.0 only samples using digital voltage thresholds, hence the material output is interpreted as high or low (i.e. 1 or 0 respectively). Mecobo 3.5 supports analogue output in a range [-5V, 5V]. Mecobo 3.5 interprets linearly the output value read from the material in a range [-4096, 4096], where output voltage -5V corresponds to -4096 and value 5V corresponds to 4096.

The output is recorded in a buffer. Three output parameters, i.e. a user-defined output sampling frequency, the start time and end time of reading output electrode determine the buffer size of output samples. If the output sampling frequency is F_{out}, start time is

TABLE 4.1: Adjustable Mecobo input parameters.

Parameter name	Description	Note
Amplitude	0V or 3.5V corresponding to 0 or 1 (Mecobo 3.0), Range [-5V, 5V] corresponding to [0, 255] (Mecobo 3.5)	Wave signal amplitude must be 1
Frequency	Frequency of square wave signal	Irrelevant if static voltage input
Mark-space ratio	Percentage of the period for which square wave signal is 1 (examples are shown in Figure 4.2)	Irrelevant if static voltage input
Phase	Phase of square wave signal	Irrelevant if static voltage input
Start time	Start time of applying input signal to an electrode	Measured in milliseconds
End time	End time of applying input signal to an electrode	Measured in milliseconds

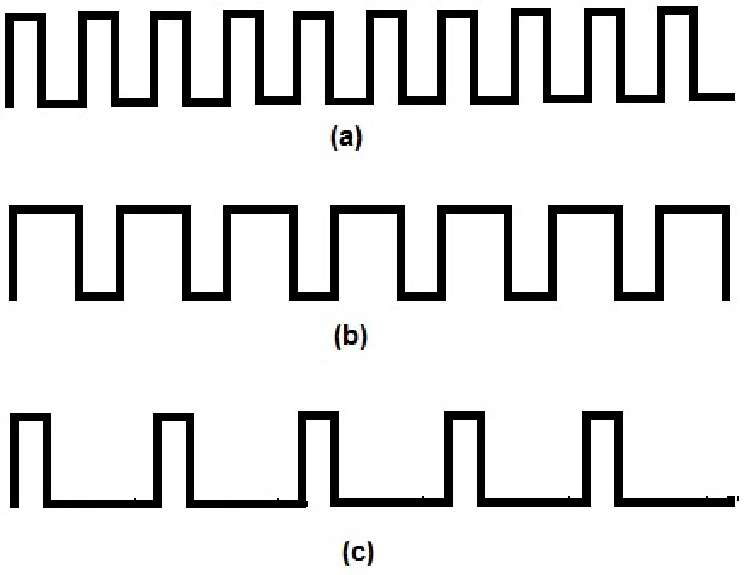


FIGURE 4.2: Examples of mark-space ratio (a) mark-space ratio is 50%, (b) mark-space ratio is 70%, (c) mark-space ratio is 25%.

$Time_{start}$ (start time of reading electrode) and end time is $Time_{end}$ (end time of reading electrode), then the buffer size, Buf_{size} is given by:

$$Buf_{size} = F_{out}(Time_{end} - Time_{start})/1000 \tag{4.1}$$

Here $Time_{start}$ and $Time_{end}$ are measured in milliseconds.

However, in practice due to pin latency, the real buffer size is generally smaller. Pin latency means the wasted time while commands are being sent to a pin. The output parameters are described in Table 4.2.

TABLE 4.2: Adjustable Mecobo output parameters.

Parameter name	**Description**	**Note**
Output sampling frequency	The frequency of reading output in a buffer	
Start time	Start time of reading output from an electrode	Measured in milliseconds
End time	End time of reading output from an electrode	Measured in milliseconds

It should be noted that in all experiments of this thesis, inputs are applied for a number of milliseconds and the outputs are accumulated in a buffer for the same number of milliseconds. This has been referred to as input-output timing.

Figure 4.3 shows an overview of the interface hardware. In this figure, an example set up is shown in the dotted box. The example genotype defines pin 2 as the output terminal, pin 1 as the data input and pins 3 - 12 as configuration inputs. The architecture is controlled by a scheduler controlling the following modules. Digital I/O can sample responses and output digital signals. The DAC module can produce analogue output signals. The DAC can be configured to output any arbitrary time-dependent waveform or static voltages. The ADC module performs the sampling of analogue waveforms from the material. The pulse width modulation (PWM) module produces digital square wave signals. The system's scheduler can set up the system to sample and apply signals statically or produce time scheduled configurations of response or stimuli. The scheduler accepts a sequence of commands from the user software. Each of these sequence items consists of parameters which describe the state of the pin at a specific point in time. In Figure 4.3, pin 2 is set as a 'recording' pin from time 0, pin 1 is set to output a PWM version having mark-space ratio value 33, pins 3, 4 and 11 are set to output the analogue voltages.

The recorder stores samples, time-dependent bit strings, digital discrete values, sampled analogue discrete values or time-dependent analogue waveforms. The recorder can include any combination of these signals. The choices of configuration terminals and data I/O can be put under evolutionary control. Pin routing is placed between the sampling buffer and the signal generator modules, which make it possible to configure any terminal of a material to be input, output or configuration input.

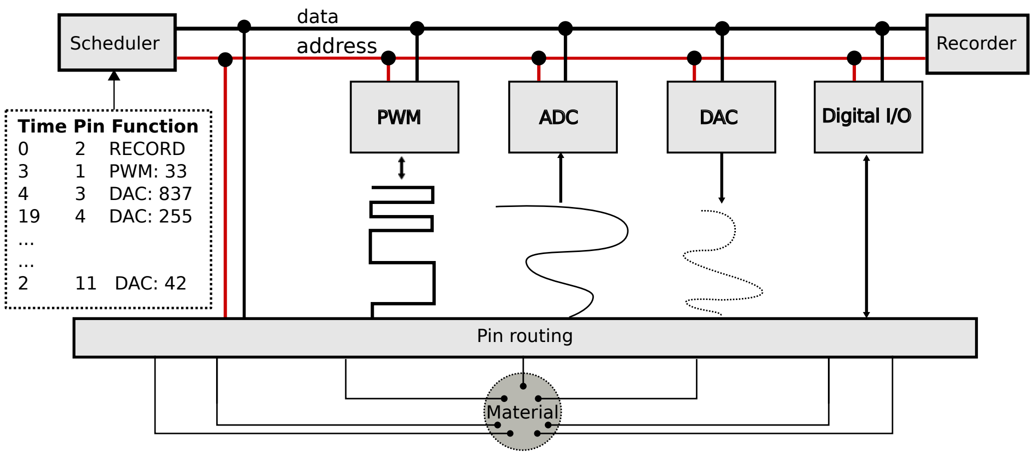


FIGURE 4.3: Overview of the complete system of interface hardware [Lykkebø*et al.* (2014)].

The material signal interface shown in Figure 4.3 is very flexible. It not only allows the possibility to evolve the I/O terminal placement but also a large variety of configuration inputs are available to support materials with different sensitivity, from static signals to time-dependent digital functions. The response from materials can be sampled as digital pulse trains, static digital signals, or analogue signals. The scheduler can schedule time slots for different stimuli when the physical material needs time to settle before a reliable computation can be observed.

The interface hardware is constructed around three main components[1]:

- The microcontroller: A microcontroller is responsible for the communication over USB and runs the system software of the interface. It receives commands and data to set up experiments and returns data to a host computer. The microcontroller issues commands and receives data from the FPGA. The scheduling unit is implemented in the microcontroller. The software interface sends commands to the scheduling unit.
- The FPGA: An FPGA stands as the physical and logical bridge communicating with materials. The FPGA holds the logic that interprets the commands and

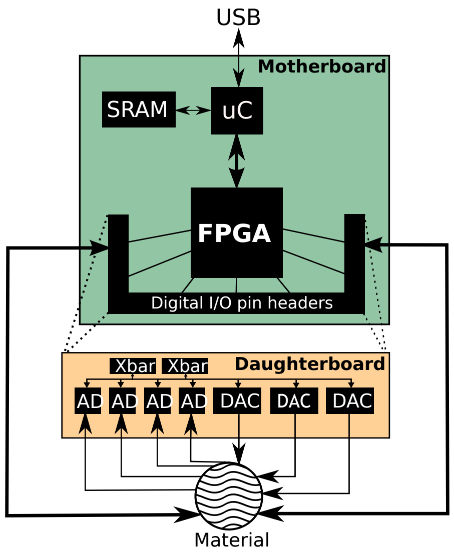
[1]The contents of the description of different components of interface hardware are provided from NASCENCE project reports.

set-up data from the evolutionary algorithm to stimulus used to configure the materials. It samples the responses from the materials.

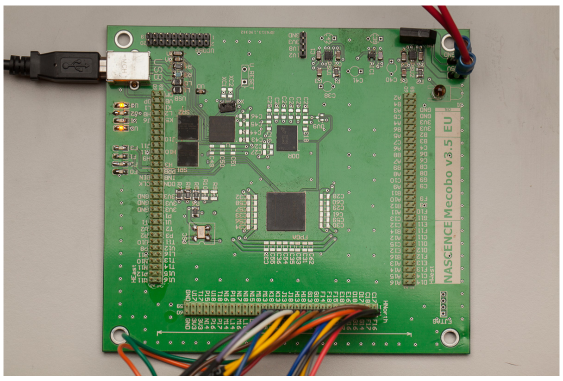
- Connectors to materials and daughter boards: The interface hardware is connected directly to material samples or interfaced to daughter boards via the connectors.

  It should be noted that all analogue components are placed on a daughter board, such as analogue-digital converters and crossbar switches. This split enables the redesign of the analogue part of the system and makes it possible to modify the analogue specification without changing the digital part of the motherboard.

Figure 4.4(a) shows an illustration of the block diagram of the interface hardware. Figure 4.4(b) shows the motherboard with the Xilinx LX45 FPGA, Silicon Labs ARM-based EFM32GG990 microcontroller connected to a 12 terminal material sample. The motherboard is able to control 80 digital I/O signals. These signals can be sent to or received from the material or can be used to control resources of the daughterboard.



(a) Mecobo block diagram.



(b) Picture of Mecobo.

FIGURE 4.4: Hardware interface implementation overview [Lykkebø*et al.* (2014)].

To provide several platforms with equal operating conditions, a physical implementation including a standard box, power supply and connectors are used with Mecobo. The physical box with a connected host computer is shown in Figure 4.5.

A detailed description of different components of Mecobo hardware, i.e. the motherboard, the FPGA, the mixed signal daughterboard and the cross switch PCB is given in Appendix A.

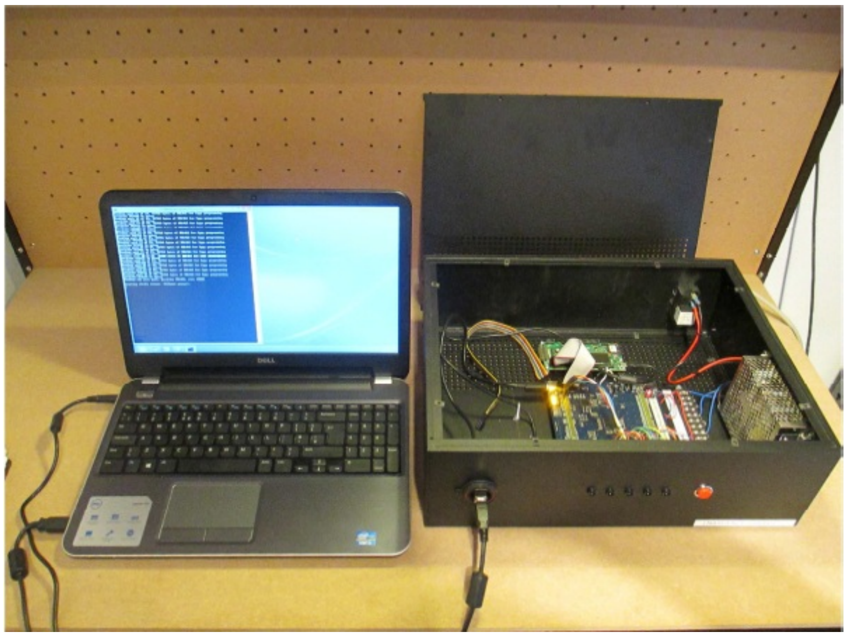


Figure 4.5: The interface stand-alone box shown with host computer

4.2 The interface Software

The interface software [Lykkebø*et al.* (2014)] has been developed by Simon Harding of the University of York (UK). He developed a system inspired by the track-based model of music or video editing applications. An example of this is shown in Figure 4.6. Each track corresponds to an output pin of the FPGA, and on each output pin an action (or a set of actions) is scheduled. Once the output pins are configured onto the FPGA, the sequence is 'played' back. A 'recording' is the data read from an input pin of the FPGA, which can be scheduled as well. The data can be read from more than one input pin of the FPGA. It should be noted that the output from the FPGA is the input to the material and the input to the FPGA is the output from the material.

An application programming interface (API) allows the users to communicate with the hardware. The API makes the functionality of the evolvable motherboard consistent and easier to use. Additional APIs help to collect and process the data.

The client application is the software which performs the evolution using evolutionary algorithms. Control software connects the client application with the evolvable motherboard. The control software is the software that is used to communicate at a low level to the evolvable motherboard. It translates the track-based model into the FPGA's internal model. The communication between the client application and the control software

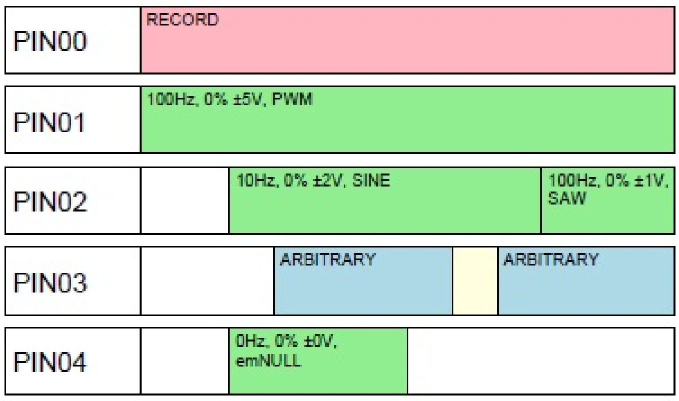


FIGURE 4.6: An example of the track-based model [Lykkebø*et al.* (2014)]. Here each row defines either an input or an output of the FPGA. The horizontal axis is time.

is through shared memory if both are in the same PC and through TCP if these are in different PCs. This means, there is no necessity to operate all the components on the same computer. The platform can be operated over the internet.

A detailed description of the interface software is given in Appendix B.

4.3 The Physical Computational Material

The experimental material consists of single-walled carbon nanotubes mixed with a polymer (PMMA or PBMA) and dissolved in anisole (methoxybenzene) [Massey *et al.* (2015b,a); Volpati *et al.* (2015); Kotsialos *et al.* (2014)]. The sample is baked causing the anisole to evaporate. This results in a material which is a mixture of single-walled carbon nanotubes and a polymer. Single-walled carbon nanotubes and PMMA/PBMA mixtures form electrically complex films having non-linear current versus voltage characteristics due to the conduction of single-walled carbon nanotubes and the dielectric properties of the PMMA/PBMA. Mark K. Massey and Michael C. Petty of Durham University (UK) prepared the materials used as substrates and the electrode masks for the experiments.

Single-walled carbon nanotubes are nanometer-diameter cylinders consisting of a single graphene sheet wrapped up to form a tube [McEuen *et al.* (2002)]. Theory and experiments showed that these tubes can be either metals or semiconductors. Metallic tubes have conductivities and current densities that meet the characteristics of the best metals, and semiconducting tubes have mobilities and transconductances that meet the characteristics of the best semiconductors. Individual single-walled carbon nanotube has remarkable room-temperature properties and mobilities more than ten times larger than silicon [Kang *et al.* (2007)]. It has current-carrying capacities as high as 10^9 A cm^{-2} and ideal sub-threshold characteristics in single-tube transistors. These behaviours of individual single-walled carbon nanotube could be significant for many applications in electronics, opto-electronics, sensing and many other areas. Kang et al. showed the dense, perfectly aligned arrays of long, perfectly linear single-walled carbon nanotube to be an effective thin-film semiconductor, which is suitable for integration into transistors, logic gates and other classes of electronic devices. These behaviours of single-walled carbon nanotubes have motivated the application of evolution-in-materio to solve a number of computational problems using single-walled carbon nanotubes in this research. Single-walled carbon nanotube network's electrical conductivity is the physical property, which is exploited and controlled here to bring the material to a state that performs some computational tasks effectively [Massey *et al.* (2015b,a); Volpati *et al.* (2015); Kotsialos *et al.* (2014)]. Different concentrations of single-walled carbon nanotube mixture (different weight ratios of single-walled carbon nanotubes in polymer) were used in the experiments. In all concentrations, the current increases monotonically with voltage. The material contains a mixture of metallic and semiconducting varieties. So, at higher concentrations, there are likely to be more metallic percolating pathways, thus yielding the significantly higher current. That means, the conductance varies with the concentrations of single-walled carbon nanotubes in the polymer. Polymers help with the dispersion of the nanotubes in solution and thus help to prepare different mixtures having different concentrations of single-walled carbon nanotubes. Polymers act as an insulating layer meaning that the material is a random mixture of conducting and insulating regions.

The electrode arrays were fabricated in gold, on slide glass substrates. These were prepared using conventional etch-back lithographic techniques. The arrays were designed with small electrode separations (22 μm) so that high strength electric fields (5 X 10^{-5} V/m) could be applied even with the modest voltages (10.8 V). A detailed view of the area in contact with the material is shown in Figure 4.7(a). A scanning electron microscope image of the electrodes is shown in Figure 4.7(b). An optical micrograph of the single-walled carbon nanotubes material deposited on the electrodes is shown in Figure 4.7(c).

Two different arrangements of electrode array in slides have been used in the experiments of this thesis. In one arrangement, a single electrode array is placed on the slide. This

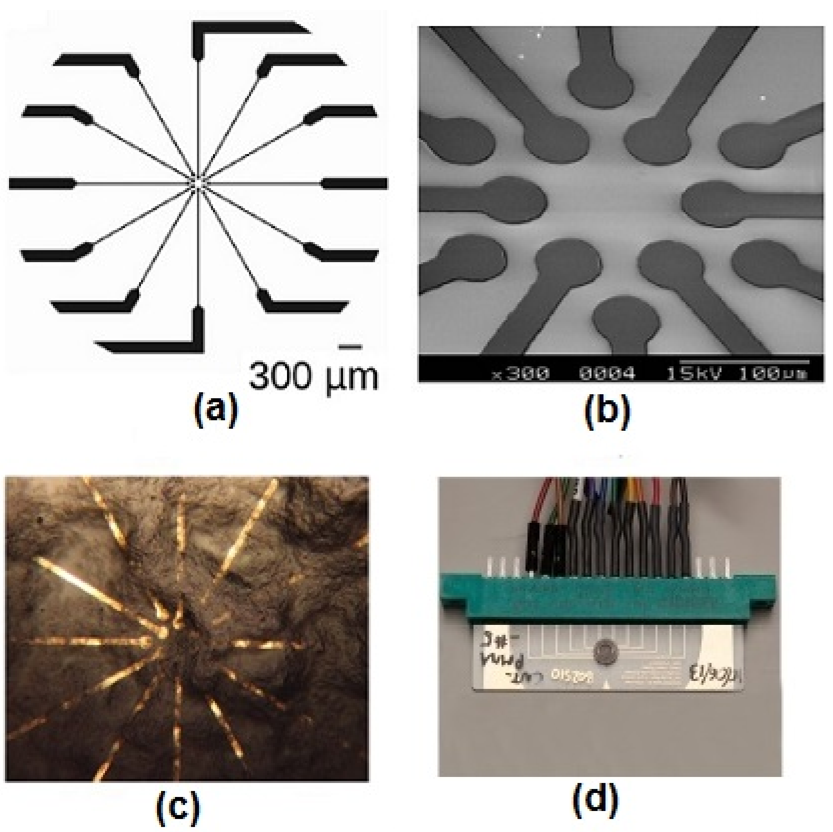


Figure 4.7: (a) A detailed view of the area in contact with the material [Kotsialos *et al.* (2014)]. (b) A scanning electron microscope image of the electrodes [Kotsialos *et al.* (2014)]. (c) An optical micrograph of the single-walled carbon nanotubes material deposited on the electrodes [Kotsialos *et al.* (2014)]. (d) An electrode array connected with wires [Lykkebø*et al.* (2014)].

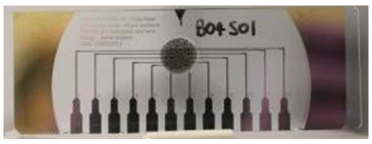


Figure 4.8: Slide with one electrode array and one sample. The electrode array has 12 electrodes.

is prepared by placing one drop of the experimental material in the middle of the slide. Twelve gold electrodes arranged on one side are connected directly with the drop. This electrode arrangement is shown in Figure 4.8. In another arrangement, two electrode arrays are placed in each slide. One drop of experimental material is placed in the middle of each electrode array. Sixteen gold electrodes (eight electrodes on each side) are connected directly with each sample on the electrode array. This electrode arrangement is shown in Figure 4.9. The electrode array is wired directly with the Mecobo board via a suitable connector. An electrode array connected with wires is shown in Figure 4.7 (d).

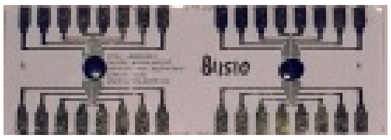


FIGURE 4.9: Slide with two electrode arrays and two samples. Each electrode array has 16 electrodes.

The materials that have been used in the experiments of this thesis are listed in Table 4.3.

TABLE 4.3: Description of materials used in experiments.

Material sample number	Arrangement of electrodes	Mixture of material (weight% fraction of single-walled carbon nanotubes in PMMA or PBMA)
Sample 1	8 electrodes in each side, in total 16 electrodes	1.0% single-walled carbon nanotubes in PBMA
Sample 2	12 electrodes in 1 side	1.0% single-walled carbon nanotubes in PBMA
Sample 3	12 electrodes in 1 side	1.0% single-walled carbon nanotubes in PMMA
Sample 4	12 electrodes in 1 side	0.71% single-walled carbon nanotubes in PMMA
Sample 5	12 electrodes in 1 side	0.50% single-walled carbon nanotubes in PMMA
Sample 6	12 electrodes in 1 side	0.10% single-walled carbon nanotubes in PMMA
Sample 7	12 electrodes in 1 side	0.05% single-walled carbon nanotubes in PMMA
Sample 8	12 electrodes in 1 side	0.02% single-walled carbon nanotubes in PMMA
Sample 9	12 electrodes in 1 side	0.01% single-walled carbon nanotubes in PMMA
Sample 10	12 electrodes in 1 side	Only PMMA

The preparation of different samples (Table 4.3) of material is given as follows:

- Making of sample 1:
  - $\approx$20 μL of material are dispensed onto the electrode array;
  - This is dried at $\approx 85^o$ C for $\approx$30 min to leave a 'thick film';
  - To prevent stress built up in the material, this is allowed to cool to room temperature on the hotplate for $\approx$1.5 h.
- Making of samples 2, 3, 5, 8, 9 and 10:
  - 20 μL of material are dispensed onto the electrode array;
  - This is dried at 85^o C for 30 min to leave a 'thick film';
  - The hotplate is turned off and the substrates are allowed to cool slowly over a period of $\approx$2 h to room temperature.
- Making of sample 4:

  - An M3-sized nylon washer is glued on the electrode array to contain the material whilst drying;
  - 20 μL of material are dispensed into the washer;
  - This is dried at $\approx 100^o$ C for $\approx$1 h to leave a 'thick film'.

- Making of samples 6 and 7:
  - 20 μL of material are dispensed onto the electrode array;
  - This is dried at $\approx 100^o$ C for $\approx$30 min to leave a 'thick film'.

4.4 Summary

The interface hardware, interface software and the experimental material have been described in this chapter in detail. These three are basic and necessary components of the experimental system and are essential for carrying out the experiments. The experiments along with the evolutionary algorithms, which have solved different computational problems, are described in following chapters.

Chapter 5

Solving Few Output Computational Problems Using Materials

Contents

This chapter and Chapters 6 and 7 describe the experiments of this research in detail. The experiments can be divided into two categories:

- Computational problems requiring few outputs;
- Computational problems with many outputs;

This chapter and Chapter 7 describe the experiments requiring few outputs, where the number of inputs, outputs and configuration inputs of a problem is not more than the total number of electrodes of an electrode array. However, this chapter mainly focuses on classification-based problems and Chapter 7 deals with robot behaviour.

Machine learning classification, tone discriminator, frequency classifier and Boolean even parity are problems that can be solved with a limited number of electrodes. Many machine learning classification datasets have too many attributes and classes to be solved using a single electrode array with 12 or 16 electrodes (it was mentioned in Chapter 4 that the electrode arrays used have either 12 or 16 electrodes). Two datasets have been chosen here for the classification experiments, which have few attributes and classes. Also, 3 and 4-even parity problems are investigated here. It is feasible to attempt to evolve solutions to these problems because they have few inputs. Even parity problems with more inputs have more test cases, thus require longer time to run. Also, even parity problems with more inputs need electrode arrays with more electrodes.

The tone discriminator problem consists of trying to decide which of two different frequency square waves has the higher frequency. It was first described in the work of Thompson [Thompson (1998)] and was later studied by Harding and Miller [Harding and Miller (2004a); Harding (2006)]. In this chapter, a related problem is also described, that of classifying whether a square wave has a frequency above or below a particular threshold. This is referred to as the frequency classification problem.

5.1 Statistical Significance Tests

Statistical significance tests were performed for comparing the experimental results in this thesis using the non-parametric two-sided Mann-Whitney U-test and the two-sample Kolmogorov-Smirnov (KS) test [Hollander and Wolfe (1973)]. The effect size [Vargha and Delaney (2000)] statistic has also been computed. In this thesis, a U-or KS test p-value < 0.05 indicates that the difference between two datasets is statistically significant. If $p \geq 0.05$, the differences are statistically not significant. The effect size, A value shows the importance of this difference considering the spread of the data; with values $A < 0.56$ showing small importance, $0.56 <= A < 0.64$ medium importance and $A >= 0.64$ large importance. It is a simple way of quantifying the difference between two groups. If a comparison between results is shown to be statistically significant with a medium or large effect size, then it can be said reasonably that any difference is not due to under sampling.

5.2 Classifying Data

Classification is an important class of problems in machine learning. The objective is to correctly classify data instances.

An important issue about machine learning classification is how the data is used. The dataset can be divided into a training set and a test set. The training set is a subset of a dataset, which is used to 'tune' the settings of the system, i.e. train the system [Michie *et al.* (1994)]. The test set is used to test or assess the final solution produced by the system which is trained on the training set. It is important to assess the generality of the solution by testing using the test set. If the solution obtained using the training set performs well on the test set, it implies that a general solution has been obtained.

In this research, the evolution-in-materio approach has been evaluated on two classification problems: Contact lens and Iris [Lichman (2013)]. Many researchers worked with these two datasets. First Fisher worked with Iris dataset [Fisher (1936)] and Cendrowska worked with Contact lens dataset [Cendrowska (1987)].

Both of these datasets have four attributes which are classified into one of the three classes. The Contact lens dataset consists of 24 instances with integer attributes. The integers are categorical in nature taking the values 1, 2 or 3. The first 16 instances were used as training data and the last 8 as test data. The Iris dataset contains 150 instances with real-valued attributes. The first fifty instances are class 1, the second fifty class two and the third set of 50 are class 3. The dataset was divided into two groups (training and test set) of 75 instances each. Each set contained exactly 25 instances of each class.

5.2.1 Methodology

Thirteen different sets (sets A-M) of experiments were performed. The experimental settings of all sets of classification experiments are described in Table 5.1 and the motives for performing the classification experiments are described in Table 5.2.

All of these experiments were performed with electrode arrays having 12 electrodes. However, in the case of material sample 1, one electrode array was used, where only 12 electrodes were used from the 16 electrodes of that electrode array and these were the middle 6 electrodes from each side of one sample. For both of these datasets, four electrodes were used as inputs (i.e. are instance-related), 3 electrodes were used as outputs (i.e. defining the class) and 5 electrodes were used as configuration inputs. Each output electrode was used for each output class. Each chromosome defined which electrodes were outputs, inputs (received square waves) or received the configuration inputs (square waves or static voltages). Examples of electrode arrangements used in classification experiments are shown in Figure 5.1.

In these experiments, once the evolutionary algorithm completed, the configurations of electrodes having the best fitness were tested with unseen test data to determine their ability to predict correctly such data.

Table 5.1: The experimental settings of all sets of classification experiments. The 'No. of gen.' and 'No. of run' columns show the number of generations and number of runs of the experiments respectively. The 'Mat. sam.' and 'Mecobo vers.' columns show the material sample number (according to Table 4.3) and version of Mecobo hardware respectively. The 'In. sig.', 'Out. sig.' and 'Conf. volt.' columns show the types of input signals (AS=analogue static voltages, DW=digital square waves), output signals (A=analogue, D=digital) and configuration inputs (AS=analogue static voltages, MD=mixtures of digital square waves and digital static voltages, M=mixtures of digital square waves and analogue static voltages) respectively. The 'In. map.' and 'Out. map.' columns show the input mapping (A=amplitude mapping, F=frequency mapping) and output mapping (SB=average of sample values in the output buffer, TG=average transition gap) respectively. The last column shows the input-output timing (measured in milliseconds) used in the experiments. It should be noted that all sets of experiments used 12 electrodes of the electrode array and a 25 KHz output sampling frequency.

Set	Data set	No. of gen.	No. of run	Mat. sam.	Mecobo vers.	In. sig.	Out. sig.	Conf. volt.	In. map.	Out. map.	Time
A	Iris	50	10	1	3.5	AS	A	M	A	SB	32
B	Iris	50	20	1	3.5	AS	A	AS	A	SB	32
C	Iris	50	20	1	3.0	DW	D	MD	F	TG	32
D	Iris	50	10	1	3.0	DW	D	MD	F	TG	128
E	Iris	500	10	1	3.0	DW	D	MD	F	TG	128
F	Iris	500	10	2	3.0	DW	D	MD	F	TG	128
G	Iris	500	10	3	3.0	DW	D	MD	F	TG	128
H	Iris	500	20	4	3.0	DW	D	MD	F	TG	128
I	Iris	500	10	5	3.0	DW	D	MD	F	TG	128
J	Iris	500	10	6	3.0	DW	D	MD	F	TG	128
K	Iris	500	10	7	3.0	DW	D	MD	F	TG	128
L	Iris	500	10	8	3.0	DW	D	MD	F	TG	128
M	Cont. lens	500	30	4	3.0	DW	D	MD	F	TG	128

In the case of Iris dataset, the number of evolutionary runs was either 10 or 20. In the case of Contact lens dataset, the number of runs was 30. The smaller number of runs (10 or 20) for the Iris was due to the large time required for each experiment as Iris has more instances than the Contact lens. It should be noted that in the case of Iris dataset, the number of runs was 20 only in those experiments that dealt with the comparisons with Cartesian genetic programming and the two Mecobo platforms. In the case of other experiments, the number of runs was 10 as many sets of experiments were needed to perform all the many different types of investigations.

Table 5.2: The motives for performing the classification experiments. The first column shows the sets of experiments. The second column shows the motive.

Experiments	Motive
Sets A, B	Comparison of the performance of using all static analogue voltages against the performance of using mixtures of static analogue voltages and digital square waves using Mecobo 3.5.
Sets B, C	Comparison of the performance of using all analogue inputs, outputs and configuration inputs against the performance of using all digital inputs, outputs and configuration inputs (comparison of the performances of two Mecobo platforms).
Sets C, D	Comparison of results using different input-output timings (32 milliseconds and 128 milliseconds).
Sets E, F	Comparison of results using different organisations of electrodes (the same mixture of material, but organisations of electrodes are different)
Sets F, G	Comparison of results using different polymers (the same percentage of single-walled carbon nanotubes, but in different polymers).
Sets G-L	Comparisons of results using different percentages of single-walled carbon nanotubes in PMMA.
Set B	Comparison of experimental results against the results of Cartesian genetic programming.
Set H	Comparison of experimental results against the results of Cartesian genetic programming.
Set M	Comparison of experimental results against the results of Cartesian genetic programming.

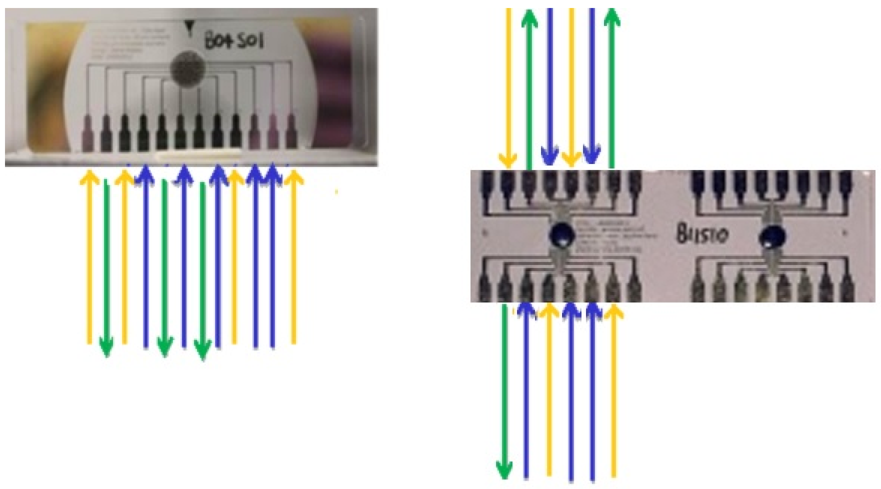


Figure 5.1: Examples of electrode arrangements used in classification experiments using two different material samples having different organisations of electrodes. Green arrows are used to indicate reading outputs from the output electrodes, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 5 electrodes.

5.2.2 Genotype Representation

Each chromosome used $n_e = 12$ electrodes at a time. The values that genes could take are shown in Table 5.3. i takes values 0, 1, …11. The description of the genotype for the experiments is shown in Table 5.4.

TABLE 5.3: Description of the genes for classification experiments

Gene symbol	**Signal applied to, or read from the i^{th} electrode**	**Allowed values**
p_i	Which electrode is used	0, 1, 2 …11
s_i	Type (Irrelevant for set: B)	0 (static), 1(square wave)
a_i	Amplitude	0 , 1 (for sets: C-M) 1, 2 …254 (for sets: A, B)
f_i	Frequency (Irrelevant for set: B)	500 ,501 …10K
ph_i	Phase (Irrelevant for set: B)	1, 2 …10
c_i	Mark-space ratio (Irrelevant for set: B)	0, 1, 2 …100

TABLE 5.4: Description of the genotype for classification experiments. The 'Exp.' column shows the set(s) of experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each genotype. The 'Genotype representation' column shows the representation of a genotype. The 'Genes related to inputs' column shows the gene values of a genotype, which are related to inputs. The 'Genes related to outputs' column shows the gene values of a genotype, which are related to outputs.

Exp.	No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Genotype representation	Genes related to inputs	Genes related to outputs
Sets A, C-M	6	$p_i, s_i, a_i, f_i, ph_i, c_i$	12X6 =72	$p_0s_0a_0f_0ph_0c_0 \dots p_{11}s_{11}a_{11}f_{11}ph_{11}c_{11}$	First 24 genes: $p_0s_0a_0f_0ph_0c_0 \dots p_3s_3a_3f_3ph_3c_3$	Last 18 genes: $p_9s_9a_9f_9ph_9c_9 \dots p_{11}s_{11}a_{11}f_{11}ph_{11}c_{11}$
Set B	2	p_i, a_i	12X2 =24	$p_0a_0 \dots p_{11}a_{11}$	First 8 genes: $p_0a_0 \dots p_3a_3$	Last 6 genes: $p_9a_9p_{10}a_{10}p_{11}a_{11}$

In these experiments, mutated children were created from a parent genotype by mutating a single gene (i.e. one gene of 72 in the case of sets A, C-M and one gene in 24 in the case of set B). In the input and output genes, only the first p_i (here the value of i is 0-3 and 9-11) has any effect, others do not have any effect. The gene p_i decides which electrode will be used for the input or output of the device. Thus, mutations in this gene can choose a different electrode to be used as an input or output.

5.2.3 Input Mapping

In experiments C-M, each of the inputs to the electrode array was a square wave signal of a particular frequency. The frequency was determined by a linear mapping of attribute data. In experiments A and B, each of the inputs to the electrode array was a static voltage of a particular amplitude. The amplitude was determined by a linear mapping of attribute data. The input mappings of these experiments are described as follows:

Denote the i^{th} attribute in a dataset by I_i, where i takes values $\{1, 2, 3, 4\}$. Denote the maximum value and minimum value taken by this attribute in the whole dataset by $I_{i_{max}}$ and $I_{i_{min}}$ respectively. Then the linear input mappings of the classification experiments are shown in Table 5.5.

5.2.4 Output Mapping

The class that an instance belongs to was determined by examining the output buffers which contain samples taken from the output electrodes. Mecobo 3.0 can only recognise binary values, so the output buffers contain bitstrings. In experiments C-M, the *transitions* from 0 to 1 in the output buffers were used to calculate the class that an instance belongs to. For each output buffer, the positions of transitions were recorded and the gaps between consecutive transitions were measured and an average calculated. A transition-based mapping was used as it is frequency related. Since instance data affects frequencies of applied signals, it seemed natural to use the method of reading output buffer bitstrings, which is itself frequency related. An example of average gap calculation for an output electrode is shown in Figure 5.2

The output class was determined by the output buffer with the largest average transition gap. If two or more buffers had the same average gap, then the class was determined by the first such buffer encountered (in precedence order: 1, 2, 3). For instance, if the second output buffer had the highest average gap, the output class would be predicted to be class 2.

Mecobo 3.5 supports analogue outputs. So, in the case of experiments A and B, the average values of output buffers were used to calculate the class that an instance belongs to. The output class was determined by the output buffer with the largest average value.

Table 5.5: Input mappings for classification experiments. The 'Exp.' column shows the set(s) of experiments. The 'Equation' column shows the equation used for input mapping in the experiments. The 'Variable' column describes the variables that are used in the input mapping equation. The 'Other parameters of input signals' column describes those parameters (gene values) of input signals, which are not used in the input mapping equation.

Exp.	Equation	Variables	Other parameters of input signals
Sets C-M	$$F_i = a_i I_i + b_i \quad (5.1)$$ Here, $$a_i = \frac{(F_{max} - F_{min})}{(I_{i_{max}} - I_{i_{min}})} \quad (5.2)$$ $$b_i = \frac{(F_{min} I_{i_{max}} - F_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})} \quad (5.3)$$	I_i is mapped to a square wave frequency F_i, where the maximum allowed frequency is F_{max} and the minimum allowed frequency is F_{min}. Here F_{max}=10KHz and F_{min}=500Hz	Mark-space ratio =50%, amplitude=1 and phase=1
Sets A, B	$$A_i = a_i I_i + b_i \quad (5.4)$$ Here, $$a_i = \frac{(A_{max} - A_{min})}{(I_{i_{max}} - I_{i_{min}})} \quad (5.5)$$ $$b_i = \frac{(A_{min} I_{i_{max}} - A_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})} \quad (5.6)$$	I_i is mapped to a static analogue voltage with amplitude A_i, where the maximum allowed amplitude is A_{max} and the minimum amplitude is A_{min}. Here A_{max}=254 and A_{min}=1	

If two or more buffers had the same average value, then the class was determined by the first such buffer encountered (in precedence order: 1, 2, 3). This mapping was used as it seems more closely related to amplitude.

5.2.5 Fitness Score

The fitness calculation required counts to be made of the number of true positives TP, true negatives TN, false positives, FP and false negatives, FN. For an instance, having a class c, according to the dataset and a predicted class p, the TP, TN, FP, and FN can be calculated using Equation 5.7. The explanation of this is as follows:

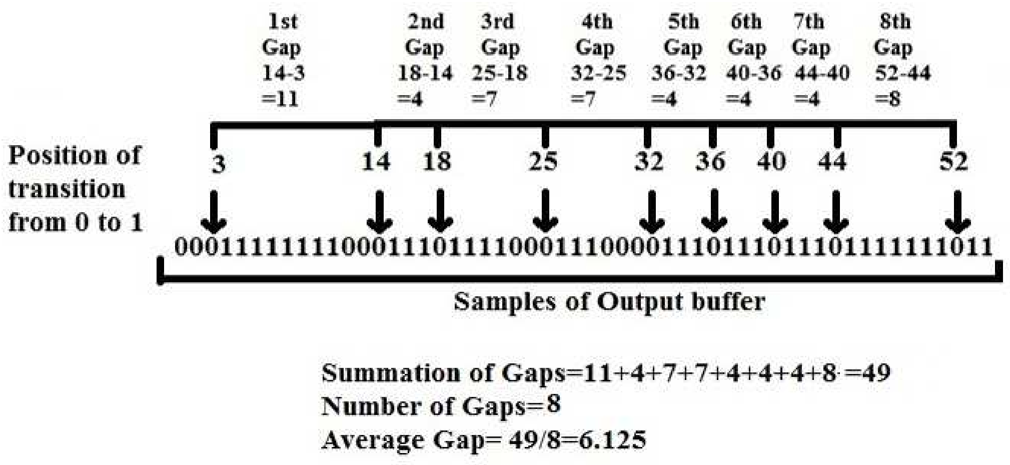


FIGURE 5.2: An example of average transition gap calculation for an output electrode

If the predicted p is correct, then it is a true positive, so TP should be incremented by one. It is also a true negative for the other two classes, hence TN should be incremented by two. If the predicted class is incorrect, then it is a false positive for the class predicted, so FP should be incremented. It is also a false negative for the actual class of the instance, so FN should be incremented. Finally, the remaining class is a true negative, so TN should be incremented.

$$\begin{aligned}&\textbf{if } p = c \textbf{ then } TP = TP + 1;\ TN = TN + 2\\&\textbf{if } p \neq c \textbf{ then } FP = FP + 1;\ FN = FN + 1;\ TN = TN + 1\end{aligned} \tag{5.7}$$

Once all instances have been classified, the fitness of a genotype can be calculated using Equation 5.8 [Akbarzadeh *et al.* (2008)].

$$fitness = \frac{TP.TN}{(TP + FP)(TN + FN)} \tag{5.8}$$

Thus, if all instances are correctly predicted, the fitness is 1 since in this case, $FP = 0$ and $FN = 0$. In the case that all instances are incorrectly predicted, $TP = 0$ and

$TN = 0$, so fitness is zero.

5.2.6 The Experimental Details

In the case of all sets of experiments, a $(1+4)$-evolutionary algorithm was used. It took more than 12 hours (one evolutionary run) to run 500 generations on the Iris training set with input-output timing 128 milliseconds.

To evaluate the effectiveness of the evolution-in-materio method for solving classification problems, the results of experiments B, H and M were compared with Cartesian genetic programming using the same $(1+4)$-evolutionary algorithm over the same number of generations and the same number of runs. In the case of all experiments of the experimental material and Cartesian genetic programming, a child replaced the parent if its fitness was greater than or equal to the parent. The fitness function of Cartesian genetic programming was the same as the fitness function used by the evolution-in-materio. Also, the number of outputs, n_O of Cartesian genetic programming was chosen to be equal to the number of classes in the dataset and the class of a data instance was defined as the class indicated by the maximum numerical output.

The function set chosen was defined over the real-valued interval [0.0, 1.0] and consisted of the following primitive functions. The functions were assumed to have three inputs, z_0, z_1, z_2 (but some are ignored):

$(z_0 + z_1)/2$; $(z_0 - z_1)/2$; $z_0 z_1$;
if $|z_1| < 10^{-10}$ **then** 1 **else if** $|z_1| > |z_0|$ **then** z_0/z_1 **else** z_1/z_0;
if $z_0 > z_1$ **then** $z_2/2$ **else** $1 - z_2/2$.

Three mutation parameters were used:

- A percentage for mutating connections, μ_c;
- A percentage for mutating functions, μ_f;
- Mutation of outputs, μ_o was done probabilistically.

In all experiments, $\mu_c = 3\%$, $\mu_f = 3\%$, and $\mu_o = 0.5$. The output mutation probability was set as 0.5 because there are only as many outputs as there are classes. A linear Cartesian genetic programming geometry was chosen by setting the number of rows, $n_r = 1$ and the number of columns, $n_c = 100$ with nodes being allowed to connect to any previous node. It should be noted that Cartesian genetic programming was applied to classification problems before. Völk et al. applied Cartesian genetic programming to classify mammograms [Völk *et al.* (2009)]. Harding et al. presented a version of Cartesian genetic programming that could handle multiple data types and then applied it to find solutions to multiple classification tasks [Harding *et al.* (2012)].

5.2.6.1 The Results

The experimental results (sets H and M) of material and the results of Cartesian genetic programming are shown in Table 5.6.

Table 5.6: Comparative results of experimental material (experiments of sets H and M) with Cartesian genetic programming on machine learning classification problems using two datasets: Iris and Contact lens. The experiments H and M were performed using material sample 4 and Mecobo 3.0 (the input-output timing was 128 milliseconds). In the case of Iris dataset, the number of runs was 20. In the case of Contact lens dataset, the number of runs was 30. In all of the cases, the number of generations was 500. The first column shows the set of experiments. The second column shows the dataset on which the experiments were performed. The third, fourth and fifth columns show average training accuracy, average test accuracy and best accuracy of experimental material respectively. The sixth, seventh and eighth columns show average training accuracy, average test accuracy and best accuracy of Cartesian genetic programming respectively. Accuracy is the percentage of the training or test set correctly predicted. 'U-t (tr)', 'KS-t (tr)' and 'Ef. sz. (tr)' (L = large, M = medium, S = small) columns show results of statistical significance tests using training dataset. These statistical significance tests have been performed using the total number of correct instances on the training set over all runs. 'U-t (ts)', 'KS-t (ts)' and 'Ef. sz. (ts)' (L = large, M = medium, S = small) columns show results of statistical significance tests using testing dataset. These statistical significance tests have been performed using the total number of correct instances on the test set over all runs. '✓' in 'U-t (tr)', 'KS-t (tr)', 'U-t (ts)' and 'KS-t (ts)' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Set	Data set	Avg. train. acc. of exp. mat.	Avg. test acc. of exp. mat.	Best acc. of exp. mat.	Avg. train. acc. of Car. gen. prog.	Avg. test acc. of Car. gen. prog.	Best acc. of Car. gen. prog.	U-t (tr)	KS-t (tr)	Ef. sz. (tr)	U-t (ts)	KS-t (ts)	Ef. sz. (ts)
H	Iris	84.7%	77.1%	96.7%	97.7%	93.6%	98.0%	✓	✓	L	✓	✓	L
M	Con. Lens	91.7%	66.7%	95.8%	93.8%	68.3%	95.8%	X	X	S	X	X	S

It has been found from the experiments that in the case of the Contact lens dataset, the training and test accuracies of the experimental material are within 2.5% of the corresponding Cartesian genetic programming accuracies, and the best accuracy of the experimental material is equal to the best accuracy of Cartesian genetic programming. In the case of the Iris dataset, the training and test accuracies of the experimental material are within 18% of the corresponding Cartesian genetic programming accuracies, but the best accuracy is within 1.5% of the Cartesian genetic programming accuracy.

The results of the experiments H, M and the results of Cartesian genetic programming have been compared using the U-test and KS-test [Hollander and Wolfe (1973)]. The effect size statistic [Vargha and Delaney (2000)] has also been computed. The results of the statistical significance tests are also shown in Table 5.6.

Experiments A and B were performed using Mecobo 3.5 and the Iris dataset with material sample 1. Mecobo 3.5 is located in Norway and the experiments were performed via the internet by connecting with Norway Mecobo board. Usually, it takes some extra time to communicate and then takes more time to perform the experiment. That is why the input-output timing was decreased to 32 milliseconds and also the number of generations was 50. Mecobo 3.5 supports static analogue input, digital square wave input and analogue output. No more than 8 static analogue inputs (inputs and configuration inputs) can be sent via Mecobo 3.5. It should be noted that Mecobo 3.5 stops working if more than 8 static analogue inputs are used. If more than 8 static analogue inputs are required, this is done by sending 8 static analogue inputs and making the remaining inputs undefined by the program. If any electrode is connected with the material and left undefined by the program, Mecobo 3.5 connects static voltage of -2.3V using that electrode by default. This means that if more than 8 static analogue inputs are needed to be sent to the material, the remaining inputs are set to static -2.3V by default by Mecobo 3.5. In experiment B, 4 static analogue inputs and 5 static analogue configuration inputs were used, i.e. in total 9 static analogue inputs were needed to be sent to the material via Mecobo 3.5, the remaining 1 configuration input was set to static -2.3V by Mecobo 3.5 irrespective of the voltage level set by the genotype for that configuration input. In experiment A, no more than 4 configuration inputs were allowed to have static analogue voltages, i.e. at least 1 configuration input must be a digital square wave signal. The results of experiments A and B were compared over ten runs as experiment A was performed for up to 10 runs. It should be noted that the results of first ten runs from the twenty runs of experiment B were used in this comparison. The detailed comparison results and the statistical significance test results are shown in Table 5.7.

After analysis of the results of experiments A and B, it was found that in the case of classification experiments using Iris dataset, the input signals (inputs and configuration inputs) having only static analogue voltages showed better results than the results obtained using mixtures of static analogue voltages and digital square waves for inputs and configuration inputs.

Table 5.7: Comparative results for different input signal combinations (static analogue voltages against mixtures of static analogue voltages and digital square waves) of Mecobo 3.5. Both of these experiments were performed using Iris dataset with material sample 1. In both of these experiments, the number of generations was 50 and the input-output timing was 32 milliseconds. The comparison was performed over ten evolutionary runs. It should be noted that the results of first ten runs from the twenty runs of experiment B were used in this comparison. Accuracy is the percentage of the training or test set correctly predicted. 'U-test', 'KS-test' and 'Effect size' (L = large, M = medium, S = small) columns show results of statistical significance tests. The first results of these columns show the test results using the training set and the second results of these columns show the test results using the test set. These statistical significance tests have been performed using the total number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Accuracy	Static analogue voltages (set B)	Mixtures of static analogue voltages and digital square waves (set A)	U-test	KS-test	Effect size
Training	93.3%	82.8%	✓	✓	L
Test	87.9%	73.2%	✓	✓	L

The results of experiments B and C show the comparison of the performance of Mecobo 3.5 against the performance of Mecobo 3.0, i.e. the comparison of the performance of using analogue inputs, outputs and configuration inputs against the performance of using digital inputs, outputs and configuration inputs. The detailed comparison results and the statistical significance test results are shown in Table 5.8. After analysis of the results of experiments B and C, it was found that in the case of classification experiments using Iris dataset, the performance of using analogue inputs, outputs and configuration inputs was better than the performance of using digital inputs, outputs and configuration inputs, i.e. the performance of Mecobo 3.5 was better than the performance of Mecobo 3.0.

Table 5.8: Comparative results for experiments using Mecobo 3.5 and Mecobo 3.0. Both of these were performed using Iris dataset with material sample 1. In both of the experiments, the number of generations was 50, the number of runs was 20 and the input-output timing was 32 milliseconds. Accuracy is the percentage of the training or test set correctly predicted. 'U-test', 'KS-test' and 'Effect size' (L = large, M = medium, S = small) columns show results of statistical significance tests. The first results of these columns show the test results using the training set and the second results of these columns show the test results using the test set. These statistical significance tests have been performed using the total number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Accuracy	Mecobo 3.5 (analogue inputs, outputs and configuration inputs) (set B)	Mecobo 3.0 (digital inputs, outputs and configuration inputs) (set C)	U-test	KS-test	Effect size
Training	91.3%	66.9%	✓	✓	L
Test	86.6%	60.7%	✓	✓	L

It was investigated whether different input-output timings show any significant difference in results or not. Experiments C and D were used for this comparison. The detailed comparison results and the statistical significance test results are shown in Table 5.9. It has been found from the results that the difference is statistically not significant according to U-test and KS-test in the case of both training and test datasets.

Table 5.9: Comparative results for experiments C and D having different input-output timings (32 milliseconds and 128 milliseconds respectively). Both of these experiments were performed using Iris dataset with material sample 1. In both of these experiments, the number of generations was 50. The comparison was performed over ten evolutionary runs. It should be noted that the results of first ten runs from the twenty runs of experiment C were used in this comparison. Accuracy is the percentage of the training or test set correctly predicted. 'U-test', 'KS-test' and 'Effect size' (L = large, M = medium, S = small) columns show results of statistical significance tests. The first results of these columns show the test results using the training set and the second results of these columns show the test results using the test set. Statistical significance tests have been performed using the total number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Accuracy	Input-output timing 32 milliseconds (set C)	Input-output timing 128 milliseconds (set D)	U-test	KS-test	Effect size
Training	65.9%	68.4%	X	X	M
Test	61.2%	63.6%	X	X	M

Experiments E and F used the same material mixture (material sample 1 and material sample 2 respectively), but different organisations of electrodes (Table 4.3). The results of these two experiments were compared to investigate whether different organisations of electrodes matter or not. The detailed comparison results and the statistical significance test results are shown in Table 5.10. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test in the case of both training and test datasets.

TABLE 5.10: Comparative results for experiments E and F using different material samples (sample 1 and sample 2 respectively) having different organisations of electrodes. Both of these experiments were performed using Iris dataset by Mecobo 3.0. In both of these experiments, the number of generations was 500 (the input-output timing was 128 milliseconds) and the number of runs was 10. Accuracy is the percentage of the training or test set correctly predicted. 'U-test', 'KS-test' and 'Effect size' (L = large, M = medium, S = small) columns show results of statistical significance tests. The first results of these columns show the test results using the training set and the second results of these columns show the test results using the test set. These statistical significance tests have been performed using the total number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Accuracy	Material sample 1 (set E)	Material sample 2 (set F)	U-test	KS-test	Effect size
Training	84.8%	84.4%	X	X	S
Test	72.1%	78.3%	X	X	S

The same percentage (1.0%) of single-walled carbon nanotubes was used in the different polymers (PBMA and PMMA) in material sample 2 and material sample 3 (Table 4.3). Material sample 2 (contains PBMA) and material sample 3 (contains PMMA) were used in experiments F and G respectively. The results of these two sets were compared to investigate whether the choice of polymer in material mixture plays any role in computation or not. The detailed comparison results and the statistical significance test results are shown in Table 5.11. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test in the case of both training and test datasets.

Different percentages of single-walled carbon nanotubes in PMMA were used in material samples 3-8 (Table 4.3) and investigated in experiments G-L respectively. The comparison results of these experiments (sets G-L) are shown in Table 5.12. It should be noted that results of the first ten runs from the twenty runs of experiment H were used in these comparisons.

Table 5.11: Comparative results for experiments F and G using different material samples (sample 2 and sample 3 respectively) having the same percentage (1.0%) of single-walled carbon nanotubes in the different polymers (PBMA and PMMA respectively). Both of these experiments were performed using Iris dataset by Mecobo 3.0. In both of these experiments, the number of generations was 500 (the input-output timing was 128 milliseconds) and the number of runs was 10. Accuracy is the percentage of the training or test set correctly predicted. 'U-test', 'KS-test' and 'Effect size' (L = large, M = medium, S = small) columns show results of statistical significance tests. The first results of these columns show the test results using the training set and the second results of these columns show the test results using the test set. These statistical significance tests have been performed using the total number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Accuracy	Single-walled carbon nanotubes in PBMA (set F)	Single-walled carbon nanotubes in PMMA (set G)	U-test	KS-test	Effect size
Training	84.4%	82.1%	X	X	M
Test	78.3%	72.3%	X	X	M

Table 5.12: Comparative results for different percentages of single-walled carbon nanotubes in PMMA. All of these experiments were performed using Iris dataset by Mecobo 3.0. In all of these experiments, the number of generations was 10 and the input-output timing was 128 milliseconds. The comparisons were performed over ten evolutionary runs. The first column shows the set of experiments. The second column shows the material sample number (Table 4.3). The third column shows the weight percent fraction of single-walled carbon nanotubes in PMMA. The fourth and fifth columns show the average training accuracy and average test accuracy of the experimental material. Accuracy is the percentage of the training or test set correctly predicted.

Set	Material sample no.	Single-walled carbon nanotubes in PMMA	Average training accuracy	Average test accuracy
G	3	1.0%	82.1%	72.3%
H	4	0.71%	80.7%	71.1%
I	5	0.50%	81.5%	68.7%
J	6	0.10%	81.7%	71.6%
K	7	0.05%	85.1%	72.3%
L	8	0.02%	80.9%	69.5%

It has been found from the results of Table 5.12 that material sample 7 (0.05% single-walled carbon nanotubes in PMMA) showed the best result. Statistical significance tests have also been performed using the results of each of the pairs of the material samples 3-8. It has been found that the difference of the results of each of the pairs of material samples 3-8 (sets G-L respectively) is statistically not significant according to U-test and KS-test in the case of both training and test datasets.

If all the results of all sets of experiments are considered, it has been found that the results of experiment B were the best since they acquired the highest accuracies in the case of both training and test data. However, in the case of experiment B, the number of generations was 50. As the results of experiment B were the best, the results were compared with the results of the well-known Cartesian genetic programming, where the number of generations of Cartesian genetic programming was also 50. The parameters of Cartesian genetic programming were the same as the parameters (Cartesian genetic programming parameters have been described at the beginning of the Section 5.2.6) used in Cartesian genetic programming experiments comparing results with experiment H. It has been found that the average results (accuracies) of experiment B were better than the average results (accuracies) of Cartesian genetic programming in the case of both training and test data. The best accuracy of the experimental material was also better than that of Cartesian genetic programming. The experiment B used Mecobo 3.5, which used analogue signals for inputs, outputs and configuration inputs. The detailed comparisons and the statistical significance test results are shown in Table 5.13

Table 5.13: Comparative results of experimental material (experiments of set B) with Cartesian genetic programming on machine learning classification problem using Iris dataset. The experiment B was performed using material sample 1 and Mecobo 3.5 (the input-output timing was 32 milliseconds). In both of these cases, the number of generations was 50 and the number of runs was 20. The first column shows the set of experiments and second column shows the dataset on which the experiments were performed. The third, fourth and fifth columns show the average training accuracy, average test accuracy and best accuracy of experimental material respectively. The sixth, seventh and eighth columns show the average training accuracy, average test accuracy and best accuracy of Cartesian genetic programming respectively. Accuracy is the percentage of the training or test set correctly predicted. 'U-t (tr)', 'KS-t (tr)' and 'Ef. sz. (tr)' (L = large, M = medium, S = small) columns show results of statistical significance tests using the training dataset. These statistical significance tests have been performed using the total number of correct instances on the training set over all runs. 'U-t (ts)', 'KS-t (ts)' and 'Ef. sz. (ts)' (L = large, M = medium, S = small) columns show results of statistical significance tests using the test dataset. These statistical significance tests have been performed using the total number of correct instances on the test set over all runs. '✓' in 'U-ts (tr)', 'KS-ts (tr)', 'U-t (ts)' and 'KS-t (ts)' columns indicates that the difference between the two results is statistically significant and 'X' indicates that the difference is statistically not significant.

Set	Data set	Avg. train. acc. of exp. mat.	Avg. test acc. of exp. mat.	Best acc. of exp. mat.	Avg. train. acc. of Car. gen. prog.	Avg. test acc. of Car. gen. prog.	Best acc. of Car. gen. prog.	U-t (tr)	KS-t (tr)	Ef. sz. (tr)	U-t (ts)	KS-t (ts)	Ef. sz. (ts)
B	Iris	91.3%	86.6%	97.3%	87.2%	84.4%	96.7%	✓	✓	L	✓	✓	L

Kester et al. also used a mixture of single-walled carbon nanotubes and PMMA to classify data instances of Iris dataset by evolution-in-materio [Clegg *et al.* (2014a)].

Their research is also under the project NASCENCE. They used a mixture of 0.53% single-walled carbon nanotubes in PMMA. The number of generations was 150 and the number of runs was 10. They applied the input signals and read the response for 0.1 seconds. The output sampling frequency was 2 KHz. This gave an output buffer having 200 samples. They used the mean of the values of final 150 samples as the output value. They used four inputs, three outputs and four configuration inputs in their experiment. They used the largest output value to determine the output class, just like the output class was determined in the classification experiments of this research (Section 5.2.4). They used static analogue voltages for inputs and configuration inputs. Each of the inputs to the electrode array was a static voltage of a particular voltage level in a range [-5V, 5V]. The voltage level of each of the input signals was determined by a linear mapping of attribute data. The voltage level of the configuration inputs was in a range [-2V, 2V]. They used (1+4)-evolutionary algorithm. Two types of mutation were performed to generate children: mutation on electrode number and mutation on voltage level. They divided the training and test datasets randomly into 3 groups of 25 instances of each class. The order of these was randomised before testing on the material sample. They used the number of correct instances as the fitness score. They found that the average (averaged over 10 runs) training accuracy was $\approx$ 69% and the best test accuracy was 95.04%.

A summary of outcomes from the classification experiments of this thesis is given as follows:

- In the case of classification experiments using Contact lens dataset, it has been found that the training and test accuracies of the experimental material (material sample 4) are within 2.5% of the corresponding Cartesian genetic programming accuracies. The number of generations was 500. Mecobo 3.0 was used in these experiments.
- In the case of classification experiments using Iris dataset, it has been found that the training and test accuracies of the experimental material (material sample 4) are within 18% of the corresponding Cartesian genetic programming accuracies. The number of generations was 500. Digital inputs, outputs and configuration inputs were used in these experiments using Mecobo 3.0.

- In the case of classification experiments using Iris dataset, it has been found that the results of the experimental material (material sample 1) were better than the results of Cartesian genetic programming when analogue inputs, outputs and configuration inputs were used by Mecobo 3.5. The number of generations was 50.
- In the case of classification experiments using Iris dataset and material sample 1, it was found that the performance of using analogue inputs, outputs and configuration inputs was better than the performance of using digital inputs, outputs and configuration inputs according to comparison results of Mecobo 3.5 and Mecobo 3.0.
- In the case of classification experiments using Iris dataset and Mecobo 3.5, it was found that the input signals (inputs and configuration inputs) having only static analogue voltages showed better results than the results obtained by mixtures of static analogue voltages and digital square waves for inputs and configuration inputs. These experiments were performed using material sample 1.
- Two sets of classification experiments were performed using Iris dataset to investigate whether different organisations of electrodes matter or not. These experiments used the same mixture of material (1.0% single-walled carbon nanotubes in PBMA), but the organisations of electrodes were different. These experiments were performed using Mecobo 3.0. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test in the case of both training and test datasets.
- Two sets of experiments were performed using Iris dataset to investigate whether the choice of polymer in material mixture plays any role in computation or not. These experiments used the same percentage (1.0%) of single-walled carbon nanotubes, but the polymers (PBMA or PMMA) were different. These experiments were performed using Mecobo 3.0 and Iris dataset. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test in the case of both training and test datasets.
- An experimental investigation examined which weight percentage of single-walled carbon nanotubes in PMMA appeared to be the most effective for classification using Iris dataset. It appeared that the best results were obtained with 0.05%

single-walled carbon nanotubes. However, the differences of the results with various mixtures appeared to be statistically not significant according to U-test and KS-test.

It should be noted that for comparing solutions of classification problems using different mixtures of materials, different hardware platforms, different electrode organisations, different signals (wave, static, digital, analogue), the Iris dataset has been selected over the Contact lens dataset for various reasons:

- The Iris dataset has much higher number of instances than the Contact lens dataset;
- The Iris is a balanced dataset, i.e. an equal number of instances for each of the output classes;
- The Iris dataset can be divided into training and test sets having equal number of instances in each set, and also both training and test sets contain the same number of instances for each of the output classes.

5.3 Discriminating Tones

The tone discriminator is a device which takes two different (different frequencies) signals as inputs and returns a different response for each of the input signals. It was first described in the work of Thompson [Thompson (1998)] (Section 3.1), and later on Harding and Miller applied evolution-in-materio to solve tone discriminator problems using an LCD by many pairs of frequencies [Harding and Miller (2004a); Harding (2006)] (Section 3.3.1). The same pairs of input frequencies have been used here to compare with the results of Harding's tone discriminator experiments.

5.3.1 Methodology

Nine different sets (sets A-I) of tone discriminator experiments have been performed. All of these experiments were performed using Mecobo 3.0. In the first set of tone discriminator experiments (A), the same pairs of frequencies as in [Harding (2006)] were investigated using material sample 4 (according to Table 4.3). Mecobo does not support

Table 5.14: The experimental settings for all sets of tone discriminator experiments. All of these experiments were performed using Mecobo 3.0. The second column shows the number of pairs of frequencies on which the experiments were performed. The third column shows the material sample number (according to Table 4.3). All of these experiments used 12 electrodes of the electrode array, a 25 KHz output sampling frequency and 250 milliseconds for the input-output timing. The number of runs was 5 and the number of generations was 500. However, the evolutionary run was terminated if fitness score gave 100% correct result.

Set	Number of pairs of frequencies	Material sample
A	119	4
B	45	1
C	45	2
D	45	3
E	45	5
F	45	6
G	45	7
H	45	8
I	1	1

frequencies lower than 500 Hz, so no frequency lower than 500 Hz was tried in any of the tone discriminator experiments here. Harding tried 114 pairs of frequencies ranging from 500 Hz to 4500 Hz. The experiments of set A used 119 pairs of frequencies, of these 114 pairs are the same as his 114 pairs of frequencies of tone discriminator experiments. However, in experiments B-H, only 45 pairs have been used in order to reduce the number of experiments. The 45 pairs were selected in a way so that they cover most of the parts of the full distribution of 119 pairs. All experiments were performed using the evolutionary process except the set I which used a random process. This experiment (set I) used only one pair of frequencies. All the populations of all generations were selected randomly in this experiment.

The experiments of set A were performed to compare the performance of a mixture of single-walled carbon nanotubes and a polymer against the performance of an LCD, i.e. the experimental results of set A were compared with the results of Harding's experiments using 114 pairs of frequencies. In tone discriminator experiments using LCD, the number of runs was 5 and each input signal was sent to the material for 250 milliseconds, so to compare the performances, the number of runs in each of the tone discriminator experiments was 5 and the input-output timing was 250 milliseconds. The experimental settings of all sets of tone discriminator experiments are described in Table 5.14 and the motives for performing the tone discriminator experiments are described in Table 5.15.

Table 5.15: The motives for performing the tone discriminator experiments. The first column shows the sets of experiments. The second column shows the motive.

Experiments	Motive
Set A	Comparison of results using a mixture of single-walled carbon nanotubes with a polymer against the results using an LCD.
Sets B, C	Comparison of results using different organisations of electrodes (the same mixture of material, but organisations of electrodes are different)
Sets C, D	Comparison of results using different polymers (the same percentage of single-walled carbon nanotubes, but in different polymers).
Sets A, D-H	Comparisons of results using different percentages of single-walled carbon nanotubes in PMMA.
Sets B, I	Comparison of the performance of using the evolutionary process against the performance of using a random process

All of these experiments were performed with electrode arrays having 12 electrodes. However, in the case of material sample 1, one electrode array was used, where only 12 electrodes were used from the 16 electrodes of that electrode array and these were the middle 6 electrodes from each side of one sample. Examples of electrode arrangements used in tone discriminator experiments are shown in Figure 5.3.

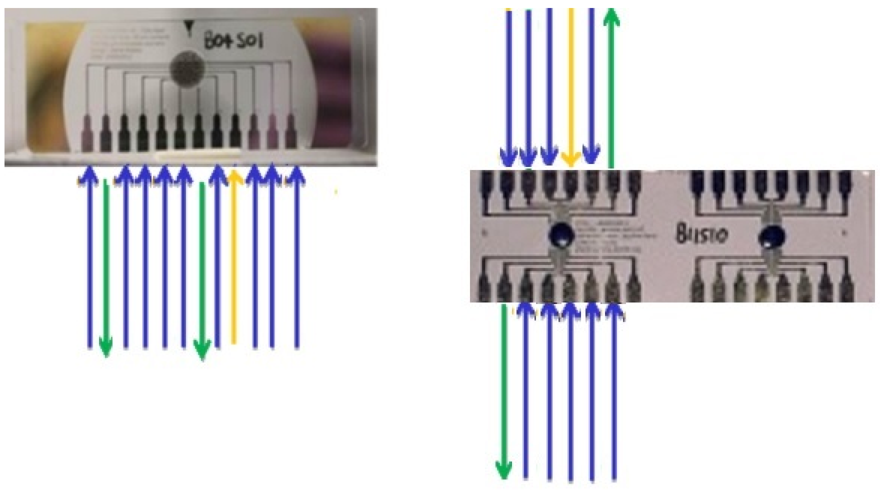


Figure 5.3: Examples of electrode arrangements used in tone discriminator experiments using two different material samples having different organisations of electrodes. Green arrows are used to indicate reading outputs from the output electrodes, a yellow arrow is used to show input being sent to an input electrode and blue arrows are used to show configuration inputs being sent to 9 electrodes.

For all experiments, one electrode was used for inputting the signal to be discriminated, two electrodes were used as outputs and nine electrodes were used as configuration inputs. Each chromosome defined which electrodes were outputs, inputs (received square waves) or received the configuration inputs (square waves or static voltages). The frequency applied to the input electrode was the input frequency to be discriminated. The mark-space ratio of input square wave signal was set to 50% and its amplitude was set to one (i.e. 3.5 V).

5.3.2 Genotype Representation

Each chromosome used $n_e = 12$ electrodes at a time. The values that genes could take are shown in Table 5.16, where i takes values 0, 1, . . . 11. The description of the genotype for the experiments is shown in Table 5.17.

TABLE 5.16: Description of the genes for tone discriminator experiments.

Gene symbol	**Signal applied to, or read from the i^{th} electrode**	**Allowed values**
p_i	Which electrode is used	0, 1, 2 . . . 11
s_i	Type	0 (static) or 1(square wave)
a_i	Amplitude	0 , 1
f_i	Frequency	500 ,501 . . . 10K
c_i	Mark-space ratio	0, 1, . . . 100

Mutated children were created from a parent genotype by mutating a single gene (i.e. one gene of 60). In the input and output genes, only the p_i (here the values of i are 0, 10, 11) has any effect, others do not have any effect. The gene p_i decides which electrode will be used for the inputs and outputs of the device. Thus, mutations in these genes can choose a different electrode to be used as an input or output.

TABLE 5.17: Description of the genotype for tone discriminator experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each genotype. The 'Genotype representation' column shows the representation of a genotype. The 'Genes related to inputs' column shows the gene values of a genotype, which are related to inputs. The 'Genes related to outputs' column shows the gene values of a genotype, which are related to outputs.

No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Genotype representation	Genes related to inputs	Genes related to outputs
5	p_i, s_i, a_i, f_i, c_i	12X5 =60	$p_0 s_0 a_0 f_0 c_0 \ldots p_{11} s_{11} a_{11} f_{11} c_{11}$	First 5 genes: $p_0 s_0 a_0 f_0 c_0$	Last 10 genes: $p_{10} s_{10} a_{10} f_{10} c_{10}$ $p_{11} s_{11} a_{11} f_{11} c_{11}$

5.3.3 Output Mapping

The output was determined using the average transition gap by examining the output buffers of output electrodes. This is the same as the average transition gap calculation of classification experiments and was described in Section 5.2.4.

The tone discriminator problem was interpreted as two-class problem. Each output was associated with a particular class. The class associated with an output electrode was determined by the output buffer with the *lower* average transition gap. If the contents of the buffer from the first output electrode had the lower average transition gap, it was designated to be class one, otherwise it was designated to be class two. So, the buffer contents from the first output electrode were expected to have the lower average transition gap only if the input frequency was low frequency.

Thus, if the tone discriminator works as desired, it would have class one when the first electrode buffer has the lower average transition gap whenever the input frequency is low. It would have class two when the first electrode buffer has the higher average transition gap at the time when input frequency is high.

It should be noted that the output was decided to be class one in the case that both output buffers had the same average transition gap.

5.3.4 Fitness Score

The fitness score was measured here using the similar method used by Harding and Miller in their evolution-in-materio experiments to solve tone discriminator problems [Harding and Miller (2004a); Harding (2006)]. Their fitness calculation method was described in Section 3.3.1. The fitness calculation for tone discriminator experiments of this thesis is described as follows:

Let, S is the vector containing the input sample and L is the length of S. The elements of the two-component vector, O decides the output class at a given time. The value of S can be either HIGH or LOW. The i^{th} element of the input sample is $S[i]$ and output vector is $O[i]$. The elements of the two component vector, x can be decided by the Equation 5.9.

$$x(i) = \begin{cases} 1, & S[i] = LOW \text{ and } O[i] = 1 \\ 1, & S[i] = HIGH \text{ and } O[i] = 2 \\ 0, & Otherwise \end{cases} \tag{5.9}$$

The fitness value is calculated using Equation 5.10.

$$fitness = 100\frac{\sum_{i=1}^{L} x(i)}{L} \tag{5.10}$$

In the experiment, L=2, i.e. one input signal is low and another input signal is high. Two examples of calculating fitness score are shown in Figure 5.4.

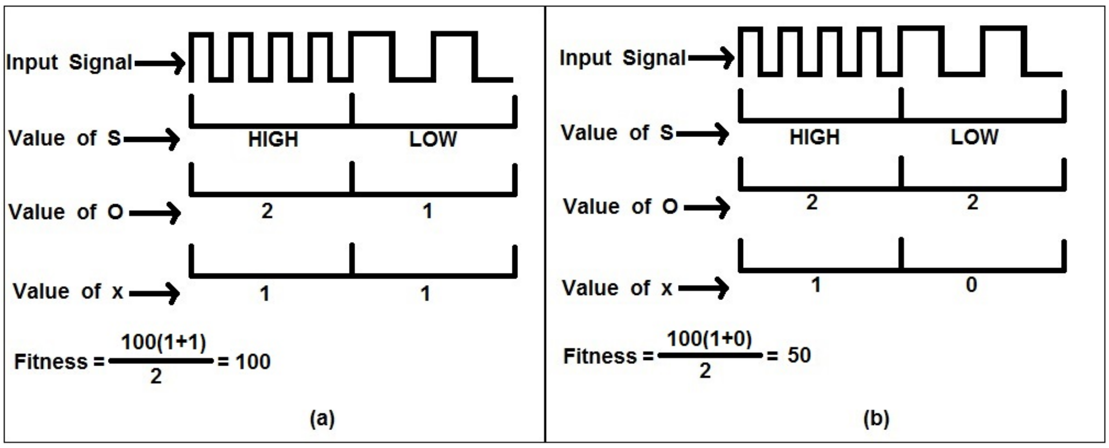


FIGURE 5.4: Examples of calculating fitness score. (a) Both tones are classified correctly and fitness score is 100. (b) Only one tone (high-frequency tone) is classified correctly and fitness score is 50.

5.3.5 The Experiments and the Results

For each of the experiments, a (1+4)-evolutionary algorithm was used. In experiments, a child replaced the parent if its fitness was greater than or equal to the parent. The total time required for all 5 runs of 119 pairs of frequencies took almost 1 day.

An image is drawn for the first experiment in Figure 5.5 showing average number of generations to find solutions for 114 pairs of frequencies. The experiment A was performed with material sample 4, i.e. 0.71% single-walled carbon nanotubes with PMMA. The image is then compared with the results of Harding's experiments with an LCD.

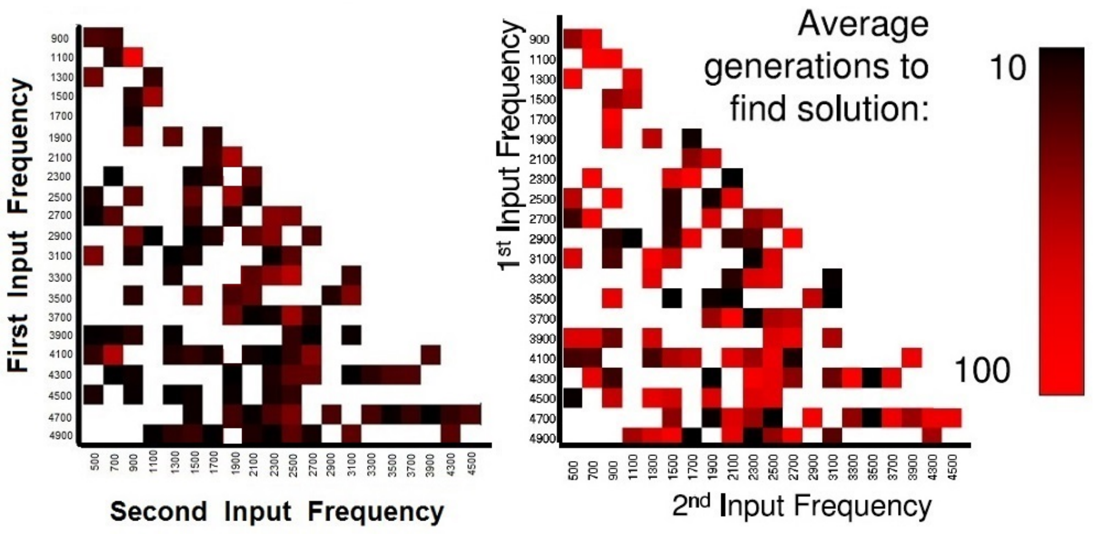


FIGURE 5.5: Graph showing a comparison of results using a mixture of single-walled carbon nanotubes with PMMA against the results using an LCD (a) The experimental results with a mixture of single-walled carbon nanotubes and PMMA. (b) Harding's experimental results using an LCD [Harding (2006)].

Different shades have been used to show average number of generations to find solutions (100% correct result). The shades are from black to red. Black is used for generation number 10 (and less than 10) and red is used for generation number 100 (and more than 100).

A tone discriminator experiment was performed with material sample 10, i.e. material with only PMMA (0% single-walled carbon nanotubes) using one pair of input frequencies to investigate whether the single-walled carbon nanotubes are required in the material mixture for computation or not. It has been observed that no evolution took place with this material sample. The measured results from the output electrodes were always zero, and this result was observed over all 100 generations for one run. In further investigations, the results of tone discriminator experiments with the same percentage of single-walled carbon nanotubes (1.0%) (material sample 2 and material sample 3) in PBMA and in PMMA have also been compared to discover whether the type of polymer plays any role in computation or not using the results of experiments C and D. Statistical significance tests have been performed for the comparison using the U-test and KS-test [Hollander and Wolfe (1973)]. The effect size statistic [Vargha and Delaney (2000)] has also been computed. The statistical significance tests have been performed over all pairs of frequencies using the average (averaged over 5 runs) number of generations to find

100% correct result for each of the pairs of frequencies. According to U-test and KS-test, the difference of the results is statistically not significant.

As experiments of 0% single-walled carbon nanotubes showed that no evolution happened without the single-walled carbon nanotubes, further investigations were performed using material sample 8 and material sample 9 (with 0.02% and 0.01% single-walled carbon nanotubes in PMMA respectively). It has been found from the investigations that no evolution took place when material sample 9 (0.01% single-walled carbon nanotubes in PMMA) was tried and the output buffers were full of zeroes always (this experiment was carried out with one pair of frequencies, the number of generations was 100 and the number of runs was 1). When an experiment was started with material sample 8 (i.e. with 0.02% single-walled carbon nanotubes in PMMA), evolution happened with mixtures of 0 and 1 in the output buffers. All of these investigation results showed that single-walled carbon nanotubes are required in the material mixture for computation.

Experiments A, D-H were used to compare results obtained by different percentages of single-walled carbon nanotubes in PMMA. Statistical significance tests (U-test, KS-test and effect size) have been performed for comparing the results. The comparison results of tone discriminator experiments with different percentages of single-walled carbon nanotubes in PMMA (material samples 3-8 according to Table 4.3) are shown in Table 5.18. The best mixture of single-walled carbon nanotubes in PMMA was material sample 5 (0.50% single-walled carbon nanotubes in PMMA), was decided using the average number of generations to find 100% correct results in all 5 runs in the case of all 45 pairs of frequencies (material sample 5 took the least number of generations on average of obtaining 100% correct results, and the value is 26.16). An image is shown for tone discriminator experiments with material sample 5 in Figure 5.6 showing the average number of generations (averaged over 5 runs) to find solutions for 45 pairs of frequencies.

Further investigations were carried out to see whether different organisations of electrodes with the same mixture of material (1.0% single-walled carbon nanotubes in PBMA) (material sample 1 and material sample 2) play any role in computation or not using experiments B and C. According to U-test and KS-test, the difference of the results is statistically not significant. The statistical significance tests have been performed over the average number of generations required to give 100% correct results in all 5 runs for 45 pairs of frequencies.

Table 5.18: Statistical significance tests on results of tone discriminator experiments with different percentages of single-walled carbon nanotubes in PMMA (material samples 3-8 according to Table 4.3). The first column shows the pair of material samples (material sample numbers are according to Table 4.3). 'U-test', 'KS-test' and 'Effect size' columns show results of the statistical significance tests. The statistical significance tests have been performed over the average number of generations required to give 100% correct results in all 5 runs for 45 pairs of frequencies. '✓' in 'U-test', 'KS-test' columns indicates that the difference between the two data samples is statistically significant and 'X' indicates that the difference is statistically not significant. It should be noted that the average number of generations required to give 100% correct results in all 5 runs for 45 pairs of frequencies are 31.19, 34.56, 26.16, 28.80, 43.16 and 57.57 in the case of material samples 3, 4, 5, 6, 7 and 8 respectively.

Pair of material samples	U-test	KS-test	Effect size
Sample 3-Sample 4	X	X	Small
Sample 3-Sample 5	X	X	Small
Sample 3-Sample 6	X	X	Small
Sample 3-Sample 7	✓	✓	Large
Sample 3-Sample 8	✓	✓	Large
Sample 4-Sample 5	X	✓	Medium
Sample 4-Sample 6	X	X	Medium
Sample 4-Sample 7	✓	✓	Medium
Sample 4-Sample 8	✓	✓	Large
Sample 5-Sample 6	X	X	Small
Sample 5-Sample 7	✓	✓	Large
Sample 5-Sample 8	✓	✓	Large
Sample 6-Sample 7	✓	✓	Large
Sample 6-Sample 8	✓	✓	Large
Sample 7-Sample 8	X	X	Medium

An investigation was carried out to compare the evolutionary process with a random process using experiments B and I. This comparison was performed using one pair of frequencies (500Hz-900Hz). In the case of both of these experiments, the number of runs was 5 and the number of generations was 500, however the run was terminated if fitness score gave 100% correct result. The experimental results and the comparison results are shown in Table 5.19.

It should be noted that in the case of all of these experiments, the results acquired 100% success in all runs. Furthermore, the average number of generations to find solutions is no more than 100.

A summary of outcomes from these tone discriminator experiments is given as follows:

- In the case of all pairs of frequencies in all tone discriminator experiments, the results acquired 100% success in all runs.

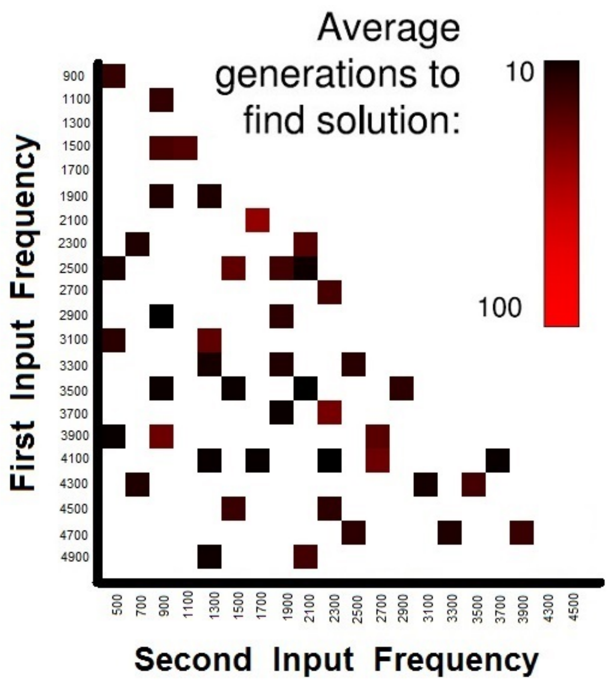


FIGURE 5.6: Graph showing tone discriminator results for 45 pairs of frequencies with material sample 5.

TABLE 5.19: Comparative results of the evolutionary process (set B) with a random process (set I). The comparison was performed using one pair of frequencies (500Hz-900Hz) with material sample 1 and Mecobo 3.0. The 'Average number of generations using evolutionary process' column shows the average number of generations required to give 100% correct results in all 5 runs for this pair of frequencies using the evolutionary process. The 'Average number of generations using random process' column shows the average number of generations required to give 100% correct results in all 5 runs for this pair of frequencies using the random process. The 'U-test' and 'Effect size' columns show results of the statistical significance tests. The statistical significance tests have been performed over the number of generations required to give 100% correct results in all 5 runs for this pair of frequencies. '✓' in 'U-test' column indicates that the difference between the two data samples is statistically significant and 'X' indicates that the difference is statistically not significant. It should be noted that the KS-test could not be performed due to the small sample size.

Average number of generations using evolutionary process	Average number of generations using random process	U-test	Effect size
17.8	19.6	X	Small

- The average number of generations to find solutions is no more than 100 in the case of all pairs of frequencies in all sets of tone discriminator experiments.
- Two sets of tone discriminator experiments were performed to investigate whether different organisations of electrodes matter or not. These experiments used the same mixture of material (1.0% single-walled carbon nanotubes in PBMA), but the organisations of electrodes were different. These experiments were performed by Mecobo 3.0 using 45 pairs of frequencies. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.
- Two sets of tone discriminator experiments were performed to investigate whether the choice of polymer in material mixture plays any role in computation or not. These experiments used the same percentage (1.0%) of single-walled carbon nanotubes, but the polymers (PBMA or PMMA) were different. These experiments were performed by Mecobo 3.0 using 45 pairs of frequencies. The statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.
- No evolution is possible with a material having 0% single-walled carbon nanotubes (only PMMA), where the output buffers are always full of zeroes, which shows that single-walled carbon nanotubes are required in the material mixture for computation.
- It has been found from an investigation (this experiment was carried out with one pair of frequencies of tone discriminator problem, the number of generations was 100 and the number of runs was 1) that no evolution took place when a mixture containing 0.01% single-walled carbon nanotubes with PMMA was tried and the output buffers were full of zeroes always. When an experiment was started with a mixture containing 0.02% single-walled carbon nanotubes with PMMA, evolution happened with mixtures of 0 and 1 in the output buffers.
- An experimental investigation examined which weight percentage of single-walled carbon nanotubes in PMMA appeared to be the most effective for tone discriminator problems. It appeared that the best result was obtained with 0.50% single-walled carbon nanotubes, this was decided using the average number of generations to find 100% correct results in all 5 runs in the case of all 45 pairs of frequencies.

5.4 Classifying Frequency

The frequency classifier problem is an extended version of tone discriminator problem, where a sequence of frequencies are used as input signals instead of only two tones and the output is decided using a threshold value. That means, the idea of a frequency classifier problem is to classify whether an applied frequency is above or below a user-defined threshold. Thus, all frequencies lower than or equal to the threshold belong to one class and frequencies higher belong to the other class.

5.4.1 Methodology

All of the experiments were performed with an electrode array having 12 electrodes with material sample 4 (Table 4.3) and Mecobo 3.0. One electrode was used for inputting the signal to be classified, 2 electrodes were used as outputs and 9 electrodes were used as configuration inputs. An example of electrode arrangement used in the frequency classification experiment is shown in Figure 5.7. Each chromosome defined which electrodes were outputs, inputs (received square waves) or received the configuration inputs (square waves or static voltages). The frequency of the signal applied to the input electrode was the input frequency to be classified. The mark-space ratio of input electrode was set to 50% and its amplitude was 1 (3.5 V). The input-output timing was 128 milliseconds and the output sampling frequency was 25 KHz.

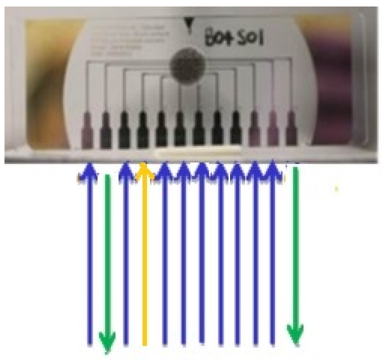


FIGURE 5.7: An example of electrode arrangement used in the frequency classification experiment. Green arrows are used to indicate reading outputs from the output electrodes, a yellow arrow is used to show input being sent to an input electrode and blue arrows are used to show configuration inputs being sent to 9 electrodes.

The frequency classification problem was interpreted as two-class problem. Each output was associated with a particular class.

Table 5.20: Description of the genes for frequency classification experiments.

Gene symbol	**Signal applied to, or read from the i^{th} electrode**	**Allowed values**
p_i	Which electrode is used	0, 1, 2 . . . 11
s_i	Type	0 (static) or 1(square wave)
a_i	Amplitude	0 , 1
f_i	Frequency	500 ,501 . . . 10K
ph_i	Phase	1, 2 . . . 10
c_i	Mark-space ratio	0, 1, . . . 100

In the evaluation of each chromosome, ten input frequencies were used: 1 KHz-10 KHz in 1 KHz intervals. At the end of each evolutionary run, the final evolved configurations of electrodes were tested using 10 *different* input frequencies: 0.5 KHz - 9.5 KHz in 1 KHz intervals. In each evolutionary run and test run, a fixed input threshold was used. Seven different sets (A-G) of experiments were performed with seven different thresholds and these were: 1.375 KHz, 2.750 KHz, 4.125 KHz, 5.500 KHz, 6.875 KHz, 8.250 KHz, 9.625 KHz respectively.

5.4.2 Genotype Representation

Each chromosome used $n_e = 12$ electrodes at a time. The values that genes could take are shown in Table 5.20, where i takes values 0, 1, . . . 11. The description of the genotype for the experiments is shown in Table 5.21.

Mutated children were created from a parent genotype by mutating a single gene (i.e. one gene of 72). In the input and output genes, only the p_i (here the values of i are 0, 10, 11) has any effect, others do not have any effect. The gene p_i decides which electrode will be used for the inputs and outputs of the device. Thus, mutations in these genes can choose a different electrode to be used as an input or output.

5.4.3 Output Mapping

The output was determined using the average transition gap by examining the output buffers of output electrodes. This is the same as the average transition gap calculation of tone discriminator and classification experiments and was described in Section 5.2.4.

Table 5.21: Description of the genotype for frequency classification experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each genotype. The 'Genotype representation' column shows the representation of a genotype. The 'Genes related to inputs' column shows the gene values of a genotype, which are related to inputs. The 'Genes related to outputs' column shows the gene values of a genotype, which are related to outputs.

No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Genotype representation	Genes related to inputs	Genes related to outputs
6	$p_i, s_i, a_i,$ $f_i, ph_i c_i$	12X6 =72	$p_0 s_0 a_0 f_0 ph_0 c_0 \ldots$ $p_{11} s_{11} a_{11} f_{11} ph_{11} c_{11}$	First 6 genes: $p_0 s_0 a_0 f_0 ph_0 c_0$	Last 12 genes: $p_{10} s_{10} a_{10} f_{10} ph_{10} c_{10}$ $p_{11} s_{11} a_{11} f_{11} ph_{11} c_{11}$

The class associated with an output electrode was determined by the output buffer with the lower average transition gap. If the contents of the buffer from the first output electrode had the lower average transition gap, it was designated to be class one, otherwise it was designated to be class two. So, the buffer contents from the first output electrode were expected to have the lower average transition gap only if the input frequency was less than or equal to the threshold.

Thus, if the frequency classifier works as desired, it would have class one when the first electrode buffer has the lower average transition gap whenever the input frequency is less than or equal to the threshold. It would have class two when the first electrode buffer has the higher average transition gap at the time when the input frequency is higher than the threshold.

It should be noted that the output was decided to be class one in the case that both output buffers had the same average transition gap.

5.4.4 Fitness Score

The fitness calculation required counts to be made of the number of true positives TP, true negatives TN, false positives, FP and false negatives, FN. There are four possible cases which are described in Table 5.22.

Table 5.22: The four possible cases for the fitness calculation of frequency classification experiments. The first row shows the condition related to input frequency. The first column shows the condition related to outputs obtained from output electrode buffers. The second and third columns of all rows except the first row show the actions.

	The input frequency is less than or equal to the threshold	The input frequency is more than the threshold
The first electrode buffer has the lower average transition gap	TP = TP + 1; TN = TN + 1;	FP = FP + 1; FN = FN + 1;
The second electrode buffer has the lower average transition gap	FP = FP + 1; FN = FN + 1;	TP = TP + 1; TN = TN + 1;

In frequency classification problems, the set of value TP, TN, FP, FN accumulated over all instance data (in the case of experiments, different applied frequencies) defines the so-called confusion matrix. To obtain the fitness value, the Matthews correlation coefficient (MCC) was used. The MCC is recognised as one of the best single number measures of the quality of a classification algorithm based on the confusion matrix [Baldi *et al.* (2000)]. It can be applied to balanced or unbalanced datasets. The MCC is calculated using Equation 5.11 and was the fitness function adopted in the experiments.

$$MCC = \frac{TP.TN - FP.FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5.11}$$

If all results are correct, the fitness is 1 since in this case, $FP = 0$ and $FN = 0$. In the case that all results are incorrect, $TP = 0$ and $TN = 0$, so fitness is -1.

It can be shown that TN has the same value as TP since when frequencies are correctly classified, both TP and TN are incremented by the same amount (it is shown at the beginning of this section). Similarly, FN has the same value as FP as when frequencies are incorrectly classified, both FP and FN are incremented by the same amount. If TN is replaced by TP and FN by FP in Equation 5.11, it can be found that

$$MCC = \frac{TP - FP}{TP + FP} \tag{5.12}$$

As either the device has the correct response to a square wave of a particular frequency or not, in each case, either TP has been incremented by one or FP has been incremented by one. The sum must be equal to the number of frequencies applied. This is ten in both training and test situations. Thus, the Equation 5.13 is:

$$TP + FP = 10 \tag{5.13}$$

Solving Equations 5.12 and 5.13,

$$TP = 5(MCC + 1) \tag{5.14}$$

The number of correctly classified frequencies, N_c is the same as the value of TP, thus $N_c = 5(MCC + 1)$, so for instance, if $MCC = 0.8$, nine out of ten input frequencies are correctly classified.

5.4.5 The Experiments and the Results

To evaluate each chromosome, ten input frequencies were applied and the responses were measured. The evolutionary runs were carried out using seven different thresholds in experiments A-G.

For each of these experiments, a (1+4)-evolutionary algorithm was used. In experiments, a child replaced the parent if its fitness was greater than or equal to the parent. The evolutionary algorithm was run for a maximum of 5000 generations. However, the evolutionary run was terminated if fitness score reached at value 1.0 (when all the input frequencies were correctly classified).

In all of the experiments of sets A-G, the number of runs was 20. The total time required for all 20 runs of all experiments was almost 4 days.

The experiments show that in the case of all of these 7 sets of experiments, the average result of all evolutionary runs has accuracy 100% in training. However, the average test accuracy of all 20 runs is not 100%, but the best test accuracy of all 20 runs is 100% in the case of 7 sets of experiments. The detailed results are shown in Table 5.23.

Figure 5.8 shows the change in fitness values in different generations for five evolutionary runs using a threshold 2.750 KHz (set B).

Inspections were carried out on all the final gene values of configuration inputs of one frequency classifier problem to see if there was a common pattern, however none was

TABLE 5.23: Experimental results with different input thresholds. The first column shows the set of experiments and the second column shows the input threshold used in experiments. The third column shows average Matthews correlation coefficient (MCC) for test data over all 20 runs.

Set	Input threshold	Average test result (MCC)
A	1375 Hz	0.79
B	2750 Hz	0.79
C	4125 Hz	0.61
D	5500 Hz	0.53
E	6875 Hz	0.66
F	8250 Hz	0.62
G	9625 Hz	0.85

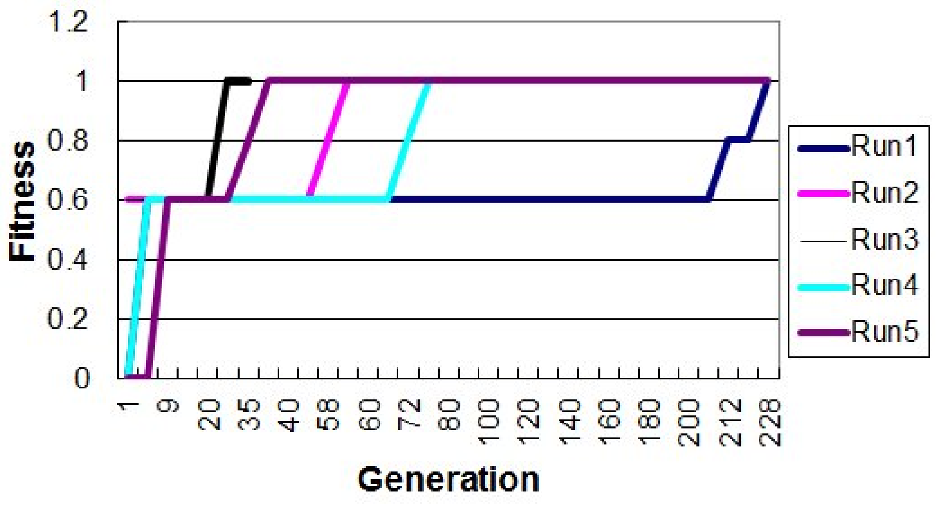


FIGURE 5.8: Fitness value vs. number of generations for five evolutionary runs using a threshold of 2.750 KHz (set B).

found. The data of one run of frequency classifier experiments of set D (having threshold 5500 Hz, and both training and test accuracies are 100%) is shown in Table 5.24.

The frequency classifier experiment of set C (using a threshold 4125 Hz) was repeated for a single evolutionary run of 100 generations using an electrode array containing no material. It was found that no evolution happened with this electrode array. The fitness value of any configurations was found to be the same after 100 generations.

A summary of outcomes from these experiments is given as follows:

- The experiments show that in the case of all sets of experiments, the average result of all evolutionary runs has accuracy 100% in training. The average test accuracy of all runs is not 100%, but the best test accuracy of all runs is 100% in the case of all sets of experiments.

Table 5.24: Final gene values of configuration inputs in one run of frequency classifier problem using threshold 5500 Hz (set D). This run achieved training and test accuracies of 100%. 'Not applicable' is used when mark-space ratio, frequency and phase values are not applicable, i.e. for static voltages.

Configuration input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio	Phase
1	3	Wave	0	4755	85	7
2	9	Static	1	Not applicable	Not applicable	Not applicable
3	8	Wave	0	4528	16	9
4	1	Wave	1	7026	34	7
5	11	Static	0	Not applicable	Not applicable	Not applicable
6	10	Static	1	Not applicable	Not applicable	Not applicable
7	0	Wave	1	3112	77	2
8	2	Wave	1	2422	31	8
9	6	Static	0	Not applicable	Not applicable	Not applicable

- No evolution happened using an electrode array containing no material.
- Inspections were carried out on all the final gene values of configuration inputs of one frequency classification experiment (obtained 100% correct result on average in both training and test set) to see if there was a common pattern, however none was found.

5.5 Solving Even Parity Problems

The n-bit parity function has n binary inputs and a single binary output. In the case of even parity problem, the output is one if there are an even number of ones in the input stream. The problem gets harder as the value of n increases, i.e. the number of inputs increases. The even parity functions with a given number of variables are very difficult functions to find solutions when carrying out a random search of all GP trees with function set {AND, OR, NAND, NOR} [Koza (1992)].

Even parity-3 and 4 have been investigated here using a mixture of single-walled carbon nanotubes with a polymer, where no logic gates have been used at all. This is the first time that a material has been used to solve these even parity problems.

Table 5.25: The experimental settings of all sets of even parity experiments. The 'In. sig.' and 'Conf. volt.' columns show the input signals (SW=square waves, S=static voltages) and configuration inputs (SW=square waves, S=static voltages, M=mixtures of static voltages and square waves) of the experiments respectively. The 'No. of in.', 'No. of out.' and 'No. of conf.' columns show the number of inputs, number of outputs and number of configuration inputs respectively. The 'In. map.' and 'Out. map.' columns show the input mapping (C=mark-space ratio mapping, F=frequency mapping, A=amplitude mapping) and output determination method (PO=percentage of ones, TG=average transition gap) respectively. The experiments of all sets were performed using Mecobo 3.0 and material sample 1 (according to Table 4.3). Sixteen electrodes were used in total. The input-output timing was 25 milliseconds, and the output sampling frequency was 25 KHz.

Set	Problem	In. sig.	Conf. volt.	No. of in.	No. of. out.	No. of conf.	In. map.	Out. map.
A	Even Parity-3	SW	M	3	2	11	F	AT
B	Even Parity-3	SW	M	3	2	11	C	PO
C	Even Parity-3	S	M	3	2	11	A	PO
D	Even Parity-3	S	S	3	2	11	A	PO
E	Even Parity-3	S	S	3	1	12	A	PO
F	Even Parity-3	SW	SW	3	2	11	C	PO
G	Even Parity-4	SW	M	4	2	10	F	AT
H	Even Parity-4	S	S	4	2	10	A	PO

5.5.1 Methodology

Eight sets (A-H) of experiments were performed. The experimental settings of all sets of even parity experiments are described in Table 5.25 and the motives for performing the even parity experiments are described in Table 5.26.

Each chromosome defined which electrodes were outputs, inputs (received square waves or static voltages) or received the configuration inputs (square waves or static voltages). In the evaluation of each chromosome, 8 test cases were used in the case of even parity-3 problem and 16 test cases were used in the case of even parity-4 problem.

5.5.2 Genotype Representation

In the case of all experiments, each chromosome used $n_e = 16$ electrodes at a time. The values that genes could take are shown in Table 5.27. i takes values 0, 1, . . . 15. The description of the genotype for the experiments is shown in Table 5.28.

In all experiments, the mutated children were created from a parent genotype by mutating a single gene (i.e. one gene of 80 in experiments A-C, G; one gene of 64 in experiment

TABLE 5.26: The motives for performing the even parity experiments. The first column shows the sets of experiments. The second column shows the motive.

Experiments	Motive
Sets A, B, C	Comparisons of the performances of different input mappings and output determination methods (frequency, mark-space ratio and amplitude for input mapping; average transition gap and percentage of ones for output determination method).
Sets C, D	Comparison of the performances of different types (static voltages against mixtures of static voltages and square waves) of configuration inputs.
Sets B, F	Comparison of the performances of different types (square waves against mixtures of static voltages and square waves) of configuration inputs.
Sets D, E	Comparison of the performances of using different numbers of output electrode(s).
Sets G, H	Comparison of the performances of solving even parity-4 problem.
Set A	Comparison of experimental results against the results of Cartesian genetic programming.
Set G	Comparison of experimental results against the results of Cartesian genetic programming.

TABLE 5.27: Description of the genes for even parity experiments.

Gene symbol	**Signal applied to, or read from the i^{th} electrode**	**Allowed values**
p_i	Which electrode is used	0, 1, 2 . . . 15
s_i	Type (Irrelevant for sets: D-F, H)	0 (static), 1(square wave)
a_i	Amplitude	0 , 1
f_i	Frequency (Irrelevant for sets: D, E, H)	500 ,501 . . . 10K
c_i	Mark-space ratio (Irrelevant for sets: D, E, H)	0, 1, 2 . . . 100

F; one gene of 32 in experiments D, E, H). In these input and output genes, only the first p_i has any effect, others do not have any effect. The gene p_i decides which electrode will be used for the input or output of the device. Thus, mutations in this gene can choose a different electrode to be used as an input or output.

Examples of electrode arrangements of even parity-3 and even parity-4 experiments are shown in Figures 5.9 and 5.10 respectively.

5.5.3 Input Mapping

In experiments A and G, each of the inputs to the electrode array was a square wave of a particular frequency. The frequency was determined by a linear mapping of attribute data. In experiments B and F, each of the inputs to the electrode array was a square

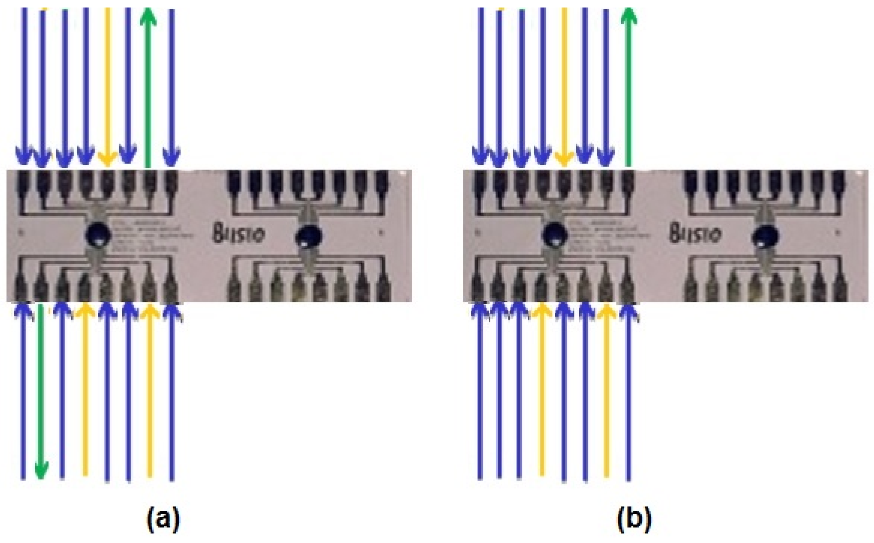


FIGURE 5.9: Two examples of electrode arrangements used in the even parity-3 experiments. (a) This type of electrode arrangement is used in sets A-D and F. Green arrows are used to indicate reading outputs from output electrodes, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 11 electrodes. (b) This type of electrode arrangement is used in set E. A green arrow is used to indicate reading output from an output electrode, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 12 electrodes.

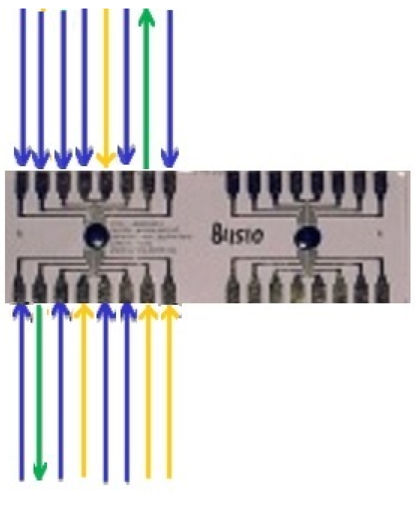


FIGURE 5.10: An example of electrode arrangement used in even parity-4 experiments. This type of electrode arrangement is used in sets G and H. Green arrows are used to indicate reading outputs from output electrodes, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 10 electrodes.

TABLE 5.28: Description of the genotype for even parity experiments. The ‘Exp.’ column shows the set(s) of experiments. The ‘No. of gen. in each elec.’ column shows the number of genes associated with each electrode. The ‘Gen. ass. with each elec.’ column shows the genes that are associated with each electrode. The ‘Total no. of genes’ column shows the total number of genes in each genotype. The ‘Genotype representation’ column shows the representation of a genotype. The ‘Genes related to inputs’ column shows the gene values of a genotype, which are related to inputs. The ‘Genes related to outputs’ column shows the gene values of a genotype, which are related to outputs.

Exp.	No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Genotype representation	Genes related to inputs	Genes related to outputs
Sets A-C	5	p_i, s_i, a_i, f_i, c_i	16X5 =80	$p_0s_0a_0f_0c_0 \dots p_{15}s_{15}a_{15}f_{15}c_{15}$	First 15 genes: $p_0s_0a_0f_0c_0$ $\dots$ $p_2s_2a_2f_2c_2$	Last 10 genes: $p_{14}s_{14}a_{14}f_{14}c_{14}$ $\dots$ $p_{15}s_{15}a_{15}f_{15}c_{15}$
Set D	2	$p_i, a_i,$	16X2 =32	$p_0a_0 \dots p_{15}a_{15}$	First 6 genes: p_0a_0 $\dots$ p_2a_2	Last 4 genes: $p_{14}a_{14}$ $\dots$ $p_{15}a_{15}$
Set E	2	$p_i, a_i,$	16X2 =32	$p_0a_0 \dots p_{15}a_{15}$	First 6 genes: p_0a_0 $\dots$ p_2a_2	Last 2 genes: $p_{15}a_{15}$
Set F	4	p_i, a_i f_i, c_i	16X4 =64	$p_0a_0f_0c_0 \dots p_{15}a_{15}f_{15}c_{15}$	First 12 genes: $p_0a_0f_0c_0$ $\dots$ $p_2a_2f_2c_2$	Last 8 genes: $p_{14}a_{14}f_{14}c_{14}$ $\dots$ $p_{15}a_{15}f_{15}c_{15}$
Set G	5	p_i, s_i, a_i, f_i, c_i	16X5 =80	$p_0s_0a_0f_0c_0 \dots p_{15}s_{15}a_{15}f_{15}c_{15}$	First 20 genes: $p_0s_0a_0f_0c_0$ $\dots$ $p_3s_3a_3f_3c_3$	Last 10 genes: $p_{14}s_{14}a_{14}f_{14}c_{14}$ $\dots$ $p_{15}s_{15}a_{15}f_{15}c_{15}$
Set H	2	$p_i, a_i,$	16X2 =32	$p_0a_0 \dots p_{15}a_{15}$	First 8 genes: p_0a_0 $\dots$ p_3a_3	Last 4 genes: $p_{14}a_{14}$ $\dots$ $p_{15}a_{15}$

wave of a particular mark-space ratio. The mark-space ratio was determined by a linear mapping of attribute data. In experiments C-E and H, each of the inputs to the electrode array was a static voltage of a particular amplitude. The amplitude was determined by a linear mapping of attribute data. The input mappings of these experiments are described as follows:

Denote the i^{th} attribute in a dataset by I_i, where i takes values $\{1, 2, 3\}$ in the case of sets A-F and $\{1, 2, 3, 4\}$ in the case of sets G and H. Denote the maximum value and minimum value taken by this attribute in the whole dataset by $I_{i_{max}}$ and $I_{i_{min}}$

TABLE 5.29: Input mappings of even parity experiments

Exp.	Equation	Variables	Other parameters of input signals
Sets A, G	$F_i = a_i I_i + b_i$ (5.15) Here, $a_i = \frac{(F_{max} - F_{min})}{(I_{i_{max}} - I_{i_{min}})}$ (5.16) $b_i = \frac{(F_{min} I_{i_{max}} - F_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})}$ (5.17)	I_i is mapped to a square wave frequency F_i, where the maximum allowed frequency is F_{max} and the minimum allowed frequency is F_{min}. Here F_{max}=10KHz and F_{min}=500Hz	Mark-space ratio =50% and amplitude=1
Sets B, F	$M_i = a_i I_i + b_i$ (5.18) Here, $a_i = \frac{(M_{max} - M_{min})}{(I_{i_{max}} - I_{i_{min}})}$ (5.19) $b_i = \frac{(M_{min} I_{i_{max}} - M_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})}$ (5.20)	I_i is mapped to a mark-space ratio M_i, where the maximum allowed mark-space ratio is M_{max} and the minimum allowed mark-space ratio is M_{min}. Here M_{max} =75% and M_{min}=25%	Frequency =5KHz and amplitude=1
Sets C-E, H	$A_i = a_i I_i + b_i$ (5.21) Here, $a_i = \frac{(A_{max} - A_{min})}{(I_{i_{max}} - I_{i_{min}})}$ (5.22) $b_i = \frac{(A_{min} I_{i_{max}} - A_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})}$ (5.23)	I_i is mapped to a static digital voltage with amplitude A_i, where the maximum allowed amplitude is A_{max} and the minimum amplitude is A_{min}. Here A_{max}=1 and A_{min}=0	

respectively. Then the linear input mappings of the even parity experiments are shown in Table 5.29.

5.5.4 Output Mapping

In experiments A and G, the output was determined using the average transition gap by examining the output buffers of output electrodes. This is the same as the average transition gap calculation of tone discriminator, frequency classification and machine learning classification experiments and was described in Section 5.2.4. The thinking behind using the average transition gap is that it may be useful as it is frequency related.

However, in experiments B-F and H, percentage of ones in an output buffer was used for determining output classes. The thinking behind using a percentage of ones is that it may be useful as it is mark-space ratio and amplitude related.

The even parity problem (both even parity-3 and even parity-4) was interpreted as two-class problem. If the output value of even parity problem is 0, it was interpreted to be class 1, otherwise it was class 2.

In the case of all experiments except the E, two electrodes were used for outputs. The class associated with an output electrode was determined by the output buffer with the lower value (average transition gap or percentage of ones). If the contents of the buffer from the first output electrode had the lower value, it was designated to be class one, otherwise it was designated to be class two. So, the buffer contents from the first output electrode were expected to have the lower value only if the output class was 1. Thus, if the solution works as desired, it would have class one when the first electrode buffer has the lower value whenever the output value of the even parity problem is 0. It would have class two when the first electrode buffer has the higher value at the time when the output value of the problem is 1. It should be noted that the output was decided to be class one in the case that both output buffers had the same value.

In experiment E, only one electrode was used for output. Percentage of ones in an output buffer was used in this case. The class associated with the output electrode was determined by a threshold value, which was set to 50. If the output electrode buffer had a value of the percentage of ones less than or equal to 50, it was designated to be class one, otherwise it was designated to be class two. So, the output electrode buffer was expected to have a value of the percentage of ones less than or equal to 50 only if the output class was one. Thus, if the solution works as desired, it would have class one,

Table 5.30: The four possible cases for the fitness calculation of even parity experiments. The first column shows the set(s) of experiments. The first row shows the condition related to output (Boolean value) of the problem. The second column shows the condition related to output obtained from output electrode buffer(s). The third and fourth columns of all rows except the first row show the action.

		The output of the problem is 0	The output of the problem is 1
Sets A, G	The first electrode buffer has the lower average transition gap	TN = TN + 1;	FN = FN + 1;
	The second electrode buffer has the lower average transition gap	FP = FP + 1;	TP = TP + 1;
Sets B-D, F, H	The first electrode buffer has the lower percentage of ones	TN = TN + 1;	FN = FN + 1;
	The second electrode buffer has the lower percentage of ones	FP = FP + 1;	TP = TP + 1;
Set E	The output electrode buffer has a percentage of ones less than or equal to 50	TN = TN + 1;	FN = FN + 1;
	The output electrode buffer has a percentage of ones more than 50	FP = FP + 1;	TP = TP + 1;

when the output electrode buffer has a value (percentage of ones) less than or equal to 50 whenever the output value of the even parity problem is 0. It would have class two when the output electrode buffer has a value more than 50 at the time when the output value of the problem is 1.

5.5.5 Fitness Score

In the case of all experiments, the fitness calculation required counts to be made of the number of true positives TP, true negatives TN, false positives, FP and false negatives, FN. There are four possible cases which are described in Table 5.30.

In even parity problems, the set of value TP, TN, FP, FN accumulated over all instance data (test cases) defines the so-called confusion matrix. To obtain the fitness value, the Matthews correlation coefficient (MCC) [Baldi *et al.* (2000)] was used. The MCC is calculated using Equation 5.24 and was the fitness function adopted in the experiments of all sets.

$$MCC = \frac{TP.TN - FP.FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5.24}$$

If all results are correct, the fitness is 1 since in this case, $FP = 0$ and $FN = 0$. In the case that all results are incorrect, $TP = 0$ and $TN = 0$, so fitness is -1. It should be noted that MCC was used in the frequency classification experiments (Section 5.4) to calculate fitness score.

In addition to the fitness calculation, the number of classes (test cases), which were predicted correctly, were recorded by summing up the values of TP and TN.

5.5.6 The Experimental Details

For each of these experiments, a (1+4)-evolutionary algorithm was used. The total time required for each of the runs, where the number of generations was 5000 (i.e. the evolutionary run was not stopped before completing 5000 generations due to not obtaining fitness value 1.0), was more than 5 hours in the case of experiments B, C, E-F and more than 9 hours in the case of experiments G and H. The later took a longer time due to the fact that the experiments G and H solved even parity-4 problem, and the even parity-4 problem has twice as many test cases as the even parity-3 problem. It should be noted that none of the experiments of sets A and D was carried out for up to 5000 generations due to obtaining 100% correct result before completing the full evolutionary run.

To evaluate the effectiveness of the evolution-in-materio method for solving even parity-3 and 4 problems, the results of experimental material were compared with the results of Cartesian genetic programming using the same (1 + 4)-evolutionary algorithm over the same number of generations (5000) and the same number of runs (20), but the evolutionary run was terminated when accuracy reached 100%. The comparison results of experimental material and Cartesian genetic programming are shown in detail in Section 5.5.6.1. In Cartesian genetic programming, the maximum number of nodes was 200, and the mutation rate was 2.00%, i.e. 12 genes per chromosome were mutated. In the case of all experiments of the experimental material and Cartesian genetic programming, a child replaced the parent if its fitness was greater than or equal to the parent. The function set chosen for this study was {AND, OR, NAND, NOR}. Evolving even parity problems using a function set {AND, OR, NAND, NOR} is a standard benchmark problem in genetic programming [Koza (1992)].

Cartesian genetic programming was applied on even parity problems before. Miller solved even parity-3,4 and 5 successfully with Cartesian genetic programming with function set {AND, OR, NAND, NOR}. He showed that extremely low populations are most effective. (1+λ)-evolutionary algorithm (probabilistic hillclimber) was proved to be more efficient than genetic programming and evolutionary programming [Miller (1999)]. However, Cartesian genetic programming did not evolve any general solution in that experiment.

Harding et al. used self-modifying Cartesian genetic programming for the first time on even parity problems (with function set {AND, OR, NAND, NOR}), where parity circuits were evolved with up to 8 inputs [Harding *et al.* (2007)]. Self-modifying Cartesian genetic programming is a form of Cartesian genetic programming, which includes primitive functions that modify the program. Later on, Harding et al. showed that using self-modifying Cartesian genetic programming it is easier to solve difficult parity problems than using either Cartesian genetic programming or modular Cartesian genetic programming with function set {AND, OR, NAND, NOR}, where the efficiencies of solutions grow with problem sizes. Most importantly, they showed that it is possible to evolve general solutions to arbitrary-sized even parity problems using self-modifying Cartesian genetic programming [Harding *et al.* (2009)].

5.5.6.1 The Results

The average accuracies of all sets of experiments are shown in Table 5.31. Accuracy is the percentage of the test cases correctly predicted.

The results of experiments A-C were used to compare the performances of different input mappings and output determination methods. It has been found from the comparisons that the result of experiment A was the best of all. That means, it has been found that in the case of even parity-3 experiments, using frequency for input mapping and average transition gap for classifying outputs performed the best of all types of mappings. The result of experiment C was better than the result of experiment B, i.e. in the case

Table 5.31: Experimental results of all sets of even parity experiments. The second column shows the accuracy averaged over 20 runs. Accuracy is the percentage of the test cases correctly predicted.

Set	Average accuracy
A	100%
B	97.5%
C	99.38%
D	100%
E	78.75%
F	99.38%
G	94.38%
H	87.5%

Table 5.32: The statistical significance test on results of even parity experiments of sets A-C. The first column shows the pair of sets on which the comparison is performed. 'U-test', 'KS-test' and 'Effect size' columns show results of the statistical significance tests. The statistical significance tests have been performed using the number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two data samples is statistically significant, and 'X' indicates that the difference is statistically not significant.

Pair of sets	U-test	KS-test	Effect size
Set A - Set B	✓	X	Medium
Set B - Set C	X	X	Medium
Set A - Set C	X	X	Small

of even parity-3 experiments, the performance of using amplitude was better than the performance of using mark-space ratio for input mapping.

The results of each pair of sets A-C have been compared using the U-test and KS-test [Hollander and Wolfe (1973)]. The effect size statistic [Vargha and Delaney (2000)] has also been computed. It should be noted that the statistical significance tests have been performed using the number of correct instances of all runs. The statistical significance test results comparing experiments A-C are shown in Table 5.32.

The results of experiments C and D were used to compare the performances of different types of configuration inputs, where experiment D used only static voltages and experiment C used mixtures of static voltages and square waves for configuration inputs. In both of these cases, the input signals were static voltages. It has been found that in the case of even parity-3 experiments, the performance of using only static voltages was better than the performance of using mixtures of static voltages and square waves for configuration inputs. Statistical significance tests have also been performed, and it has been found that the difference is statistically not significant according to U-test and

KS-test. The same type of comparison (performances of different types of configuration inputs) was performed using experiments B and F, where experiment F used only square waves and experiment B used mixtures of static voltages and square waves for configuration inputs. In both of these cases, the input signals were square waves. It has been found that in the case of even parity-3 experiments, the performance of using only square waves was better than the performance of using mixtures of static voltages and square waves for configuration inputs. Statistical significance tests have also been performed, and it has been found that the difference is statistically not significant according to U-test and KS-test.

Comparing the results of experiment C with D and experiment B with F, it has been found that in the case of even parity-3 experiments, for configuration inputs, the performance of using only static voltages was better than the performance of using mixtures of square waves and static voltages when input signals were static voltages. Furthermore, the performance of using only square waves was better than the performance of using mixtures of square waves and static voltages for configuration inputs when input signals were square waves. Thus, it appears that the performances of different types of configuration inputs may be influenced by the input signals. If the input signals and configuration inputs are combined together and defined as input signals, then it is found that in the case of even parity-3 experiments, for input signals, the performance of using either all static voltages or all square waves was better than the performance of using mixtures of static voltages and square waves. However, the difference of the results is statistically not significant according to U-test and KS-test, i.e very little difference. This little difference might be due to the influences of some other factors such as input mappings, output determination methods and types (digital or analogue) of input signals. This requires further investigation.

The results of experiments D and E were used to compare the performances of using different numbers of output electrode(s) to classify Boolean output values of even parity-3 problem, where experiment D used two output electrodes and experiment E used only one output electrode. It has been found that in the case of even parity-3 experiments, the performance of using two output electrodes was better than the performance of using one output electrode. Statistical significance tests have also been performed, and it has been found that the difference is statistically significant according to U-test and KS-test, and also the effect size is large.

After analysis of the results of all sets (A-F) of even parity-3 experiments, it has been found that both experiments A and D acquired 100% accuracy on average, i.e. all the experiments of sets A and D predicted all the 8 test cases correctly. In the case of experiment A, the 100% correct result was obtained on average within 275.75 generations (averaged over 20 runs) and the lowest number of generations to obtain 100% correct result was 27. In the case of experiment D, the 100% correct result was obtained on average within 899.9 generations (averaged over 20 runs) and the lowest number of generations to obtain 100% correct result was 43. If the average number of generations or the lowest number of generations to obtain 100% correct result is considered, it can be said that the performance of experiment A was better than the performance of experiment D. That means, in the case of even parity-3 experiments, the performance of using frequency for input mapping, average transition gap for output mapping, square waves for input signals, mixtures of static voltages and square waves for configuration inputs was better than the performance of using amplitude for input mapping, percentage of ones for output mapping, static voltages for input signals and configuration inputs. The same outcome has been obtained in the case of experiments G and H, where experiment G acquired 94.38% accuracy on average, 100% accuracy in the case of 6 out of 20 runs and experiment H acquired 87.5% accuracy on average, but none of the 20 runs of experiment H acquired 100% accuracy.

The experimental settings of sets G and H were exactly the same as the experimental settings of sets A and D respectively. Only those sets of experiments of even parity-3 problem were applied on the even parity-4 problem, which acquired 100% accuracy on average.

It has been found from the results that the performance of experiment G was not as good as the performance of experiment A and the performance of experiment H was not as good as the performance of experiment D. This might be due to the fact that the experiments G and H might need more generations to obtain 100% accurate result than the experiments A and D because of the higher number of test cases. If experiments G and H would be run for more generations, better results could be obtained.

The performance of experiment A was the best of all of the even parity-3 experiments and the performance of experiment G was better than the other (set H) even parity-4 experiment. That is why the results of experiments A and G were compared with the

Table 5.33: Comparative results of experimental material (sets A and G) with Cartesian genetic programming on even parity-3 and 4 problems. The experiments A and G were performed using material sample 1 and Mecobo 3.0. In all of these cases, the number of runs was 20 and the number of generations was 5000. However, in both of these cases, the evolutionary run was terminated when the accuracy reached 100%. The input-output timing of each experiment was 25 milliseconds. The first column shows the set of experiments. The second column shows the problem (even parity-3 or even parity-4) on which the experiments were performed. The third and fourth columns show the average accuracy of the experimental material and Cartesian genetic programming respectively. Accuracy is the percentage of the test cases correctly predicted. 'U-test', 'KS-test' and 'Effect size' columns show results of the statistical significance tests. The statistical significance tests have been performed using the number of correct instances of all runs. '✓' in 'U-test' and 'KS-test' columns indicates that the difference between the two results is statistically significant, and 'X' indicates that the difference is statistically not significant.

Set	Problem	Average accuracy of experimental material	Average accuracy of Cartesian genetic programming	U-test	KS-test	Effect size
A	Even Parity-3	100%	100%	X	X	Small
G	Even Parity-4	94.38%	97.19%	✓	✓	Large

results of Cartesian genetic programming. The comparison results are shown in Table 5.33. It should be noted that both of these experiments (A and G) used frequency for input mapping, average transition gap for classifying outputs, square waves for input signals and mixtures of static voltages and square waves for configuration inputs.

It has been found that in the case of even parity-3 problem, both Cartesian genetic programming and experimental material acquired accuracy 100% on average. In the case of experimental material, the 100% correct result was obtained on average within 275.75 generations (averaged over 20 runs) and the lowest number of generations to obtain 100% correct result was 27. In the case of Cartesian genetic programming, the 100% correct result was obtained on average within 845 generations (averaged over 20 runs) and the lowest number of generations to obtain 100% correct result was 142. If the average number of generations or the lowest number of generations to obtain 100% correct result is considered, it can be said that in the case of even parity-3 problem, the experimental material performed better than Cartesian genetic programming. It has been observed from the comparison results of the Table 5.33 that in the case of even parity-4 problem, Cartesian genetic programming performed better than the experimental material.

A summary of outcomes from these experiments is given as follows:

- In the case of even parity-3 problem, using frequency for input mapping and average transition gap for classifying outputs performed the best of all types of mappings. It should be noted that this outcome was obtained using digital inputs, outputs and configuration inputs (i.e using Mecobo 3.0).

- In the case of even parity-3 problem, the performance of using amplitude was better than the performance of using mark-space ratio for input mapping. It should be noted that this outcome was obtained using digital inputs, outputs and configuration inputs (i.e using Mecobo 3.0).

- In the case of even parity-3 problem, for configuration inputs, the performance of using only static voltages was better than the performance of using mixtures of square waves and static voltages when input signals were static voltages. Furthermore, the performance of using only square waves was better than the performance of using mixtures of square waves and static voltages for configuration inputs when input signals were square waves. Thus, it appears that the performances of different types of configuration inputs may be influenced by the input signals. If the input signals and configuration inputs are combined together and defined as input signals, then it is found that for input signals, the performance of using either all static voltages or all square waves was better than the performance of using mixtures of static voltages and square waves. However, the difference of the results is statistically not significant according to U-test and KS-test, i.e very little difference. This little difference might be due to the influences of some other factors such as input mappings, output determination methods and types (digital or analogue) of input signals. This requires further investigation.

- The results of even parity-4 experiments were not as good as the results of even parity-3 experiments. This might be due to the fact that the experiments of the even parity-4 problem need more generations to get 100% accurate result than the even parity-3 problem because of a higher number of test cases. If experiments of even parity-4 problem would be run for more generations, better results could be obtained.

- In the case of even parity-3 problem, the performance of using two output electrodes was better than the performance of using one output electrode.

- It has been found that in the case of even parity-3 problem, both Cartesian genetic programming and experimental material acquired 100% accuracy on average. However, if the average number of generations or the lowest number of generations to obtain 100% correct result is considered, it can be said that in the case of even parity-3 problem, the experimental material performed better than Cartesian genetic programming.
- It has been found that Cartesian genetic programming performed better than the experimental material in the case of even parity-4 problem.

5.6 Summary

Four experiments have been described in detail in this chapter: machine learning classification, tone discriminator, frequency classification and even parity experiments. This is the first time that machine learning classification, frequency classification and even parity problems have been attempted by the manipulation of physical materials. It cannot be said that solving classification problems using evolution-in-materio is very efficient, but these are the initial experiments. More investigations are required to be performed in future, which might make evolution-in-materio as a useful and efficient technique to solve these problems. Also, the physical material plays a very important role in this regard. This is the first time that mixtures of single-walled carbon nanotubes and polymers have been used to solve classification problems.

The tone discriminator experiments obtained 100% accuracy on average in all of the cases. The tone discriminator problem has been extended here to the more general problem of frequency classification. This is a more difficult problem than the tone discriminator since more than two tones have to be classified. In addition, in the frequency classification experiments, ten unseen test frequencies have been used to test the generality of evolved solutions. These experiments obtained 100% accuracy in most cases. The even parity-3 experiments also obtained 100% accuracy on average in two cases. Both tone discriminator and frequency classifier problems have advantages to solve using Mecobo as no input mapping is required for these two problems. Square waves with specific frequency can be used as input directly in the case of these two problems.

Different mixtures of material and different organisations of electrodes were used in machine learning classification (using Iris dataset) and tone discriminator experiments to find the best mixture of material and the best organisation of electrodes. Different hardware platforms (Mecobo 3.0 and Mecobo 3.5) were also used in classification experiments using Iris dataset, which showed that the performance of Mecobo 3.5 was better than the performance of Mecobo 3.0.

An investigation using a frequency classifier problem has shown that no evolution can be possible using an electrode array containing no material. The tone discriminator problem has shown that single-walled carbon nanotubes are required in the material mixture for solving computational problems.

The next chapter describes those experiments which require many outputs, where the number of outputs of a problem is more than the total number of electrodes of an electrode array.

Chapter 6

Solving Many Output Computational Problems Using Materials

Contents

Chapter 5 described experiments requiring few outputs, where the number of inputs, outputs and configuration inputs of a problem is less than the total number of electrodes of an electrode array. This chapter describes the experiments having many outputs, where the number of outputs of a problem is more than the total number of electrodes of an electrode array. As mentioned in Chapter 4, the electrode arrays used have either 12 or 16 electrodes. Function optimisation and bin packing problems typically have many dimensions, thus requiring many outputs to define a solution. The technique used in these experiments to solve problems having more outputs is called *split genotype technique*. The description of this technique and the characteristics of the problems, which are suitable for the application of the split genotype technique, are discussed in this chapter.

6.1 Split Genotype Technique

In the split genotype technique, a genotype consisting of multiple chromosomes is used, each of which is applied to a number of electrodes. On each application of a chromosome, the configuration inputs are applied to a number of electrodes of an electrode array and the values are read from the output buffer(s) of samples from the remaining electrode(s) of that electrode array. This means, each chromosome defines which electrode(s) will be read and which electrodes will receive the input signals or configuration inputs (square waves or static voltages). The final fitness is calculated from all the outputs of all chromosomes of the genotype. In this technique, each of the chromosomes of a genotype (genotype of each individual of the population) is applied in turn. This is referred to as an iteration. An iteration means applying the evolved configuration inputs to an electrode array, applying inputs and reading outputs. This is repeated for each evolved chromosome. For instance, for a 30 variable function optimisation problem, 30 chromosomes have been used in the genotype, where each chromosome defines which electrode will be read and which electrodes will be used as configuration inputs (as no input signal was needed).

6.2 Optimising Functions

Benchmark function optimisation problems are functions, $f(x_i)$ of a number (n) of real-valued variables, where $i = 1, 2, \ldots n$. The goal is to obtain the values of x_i, which make $f(x_i)$ to be a minimum. In evolutionary computation, many complex, multi-modal functions have been designed, whose minima are known, but these are challenging functions to minimise using search algorithms. An example of such a function is given in Equation 6.1 [Vesterstrom and Thomsen (2004)], and the two-dimensional version is illustrated in Figure 6.1. In general, these functions have many dimensions (typically 30).

$$f_8(x) = \sum_{i=1}^{d} -x_i \sin(\sqrt{|x_i|}) \tag{6.1}$$

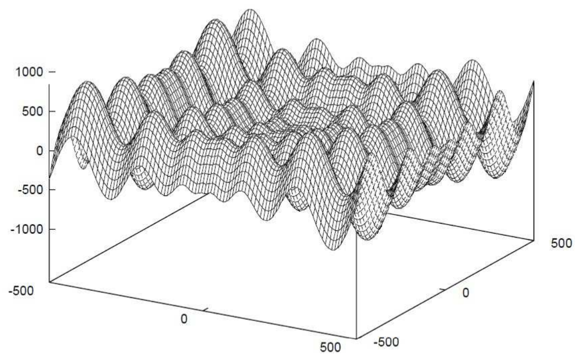


FIGURE 6.1: Function optimisation problem $f_8(x_1,\, x_2)$.

Optimisation functions are typically defined over a variety of ranges for each variable, x_i. For instance, f_8 is defined over $-500 \leq x_i \leq 500$ and has a global minimum given by $min(f_8(x)) = f_8(420.9687, ..., 420.9687) = -12,569.5$ [Vesterstrom and Thomsen (2004)].

Here in these experiments, 17 out of 23 benchmark functions (functions 1, 3 - 11, 14 - 16, 18, 21 - 23) have been chosen from [Vesterstrom and Thomsen (2004)] and 6 out of 23 functions (functions 2, 12 - 13, 17, 19, 20) from [Yao and Liu (1996)]. These benchmark functions are defined in Appendix C.

6.2.1 Methodology

Four different sets (A-D) of experiments were performed with function optimisation problems. The experimental settings of all sets of function optimisation experiments are

Table 6.1: The experimental settings of all sets of function optimisation experiments. The ‘Fun. set’ column shows the benchmark function numbers on which the experiments were performed. The ‘No. of gen.’ and ‘No. of run’ columns show the number of generations and number of runs respectively. The ‘Mat. sam.’ and ‘No. of el.’ columns show the material sample number (according to Table 4.3) and number of electrodes used in the experiments respectively. The ‘SG’ column shows whether the split genotype technique has been used or not. ‘✓’ in this column indicates that the split genotype technique has been used and ‘X’ indicates that the split genotype technique has not been used. The ‘Mec. ver.’ column shows the version of Mecobo. The ‘Out. sig.’, ‘Con. vol.’ and ‘Out. map.’ columns show the types of output signals (A=analogue, D=digital), configuration inputs (AS=analogue static voltages, M=mixtures of digital square waves and digital static voltages) and output mapping (AB=average of absolute values of output buffer, PO=percentage of ones) respectively. The ‘In. pop.’ column shows the information of initial population (R=random, F=fixed) of the evolutionary run. In the case of fixed initial population, the population was seeded based on some initial investigations (the detailed description is given in Section 6.2.3). The last column shows the input-output timing used in the experiments (measured in milliseconds). In all sets of experiments, a 25 KHz output sampling frequency was used. It should be noted that in the case of set A, some initial investigations were performed, which showed that no more than 40% ones can be obtained in output buffer with input-output timing 128 milliseconds, that is why the value 2.5 was multiplied with the value of percentage of ones in an output buffer in this case (the detailed description is given in Section 6.2.3). The outcomes of the same initial investigations were used to seed the initial populations of the experiments of set A.

Set	Fun. set	No. of. gen.	No. of run	Mat. sam.	No. of el.	SG	Mec. ver.	Out. sig.	Con. vol.	Out. map.	In. pop.	Time
A	1-23	5000	10	4	12	✓	3.0	D	M	PO	F	128
B	14-19, 21-23	1000	20	1	16	✓	3.0	D	M	PO	R	50
C	14-19, 21-23	1000	20	1	16	✓	3.5	A	AS	AB	R	50
D	14-17	1000	20	1	16	X	3.5	A	AS	AB	R	50

described in Table 6.1. The motives for performing the function optimisation experiments are described in Table 6.2.

A series of output values (these are digital in the case of Mecobo 3.0 and analogue in the case of Mecobo 3.5) were read from a buffer of samples taken from a single electrode. These values were used to define the value of a variable ‘x_i’ in the function optimisation problem. As the optimisation functions have more than one dimension, more than one output from the device was needed. In these cases, the split genotype technique was used, where one electrode of an electrode array was used as an output and the remaining electrodes of that electrode array were used as configuration inputs. However, the experiments of set D did not use the split genotype technique as these experiments were applied on four functions that require few outputs. The fitness function

Table 6.2: The motives for performing the function optimisation experiments. The first column shows the sets of experiments. The second column shows the motive.

Experiments	Motive
Set A	Comparison of experimental results against the results of Cartesian genetic programming
Sets B, C	Comparison of the performance of using all analogue configuration inputs and outputs against the performance of using all digital configuration inputs and outputs (comparison of the performances of two Mecobo platforms)
Sets C, D	Comparison of the performance of using the split genotype technique against the performance without using the split genotype technique.

of a function optimisation benchmark was the mathematical function defined in the benchmark. The process of generating outputs in function optimisation experiments using the split genotype technique is shown in Figure 6.5.

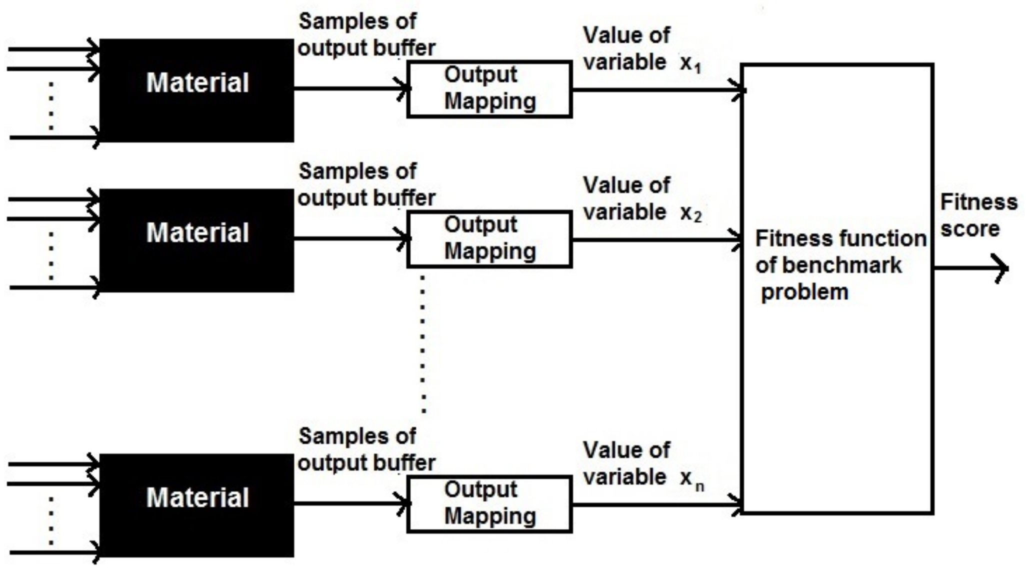


Figure 6.2: The process of generating outputs in function optimisation experiments using the split genotype technique.

It should be noted that no more than 8 static analogue input signals (inputs and configuration inputs) can be sent via Mecobo 3.5. If more than 8 static analogue signals are needed to be sent to the material, the remaining input signals are set to static -2.3V by default by Mecobo 3.5. As 15 configuration inputs were used in the experiments of set C, the remaining 7 configuration inputs were set to static -2.3V by Mecobo 3.5 irrespective of the voltage levels set by the genotype for these configuration inputs.

In the case of functions 14, 16 and 17 of set D, two electrodes were used as outputs

Table 6.3: Description of the genes for function optimisation experiments.

Gene symbol	**Signal applied to, or read from the i^{th} chromosome and the j^{th} electrode**	**Allowed values**
$p_{i,j}$	Which electrode is used	0, 1, 2 . . . 11 (for set: A) 0, 1, 2 . . . 15 (for sets: B-D)
$s_{i,j}$	Type (Irrelevant for sets: C, D)	0 (static) or 1(square wave)
$a_{i,j}$	Amplitude	0 , 1 (for sets: A, B) 1, 2 . . . 254 (for sets: C, D)
$f_{i,j}$	Frequency (Irrelevant for sets: C, D)	500 ,501 . . . 10K
$ph_{i,j}$	Phase (Irrelevant for sets: B-D)	1, 2 . . . 10
$c_{i,j}$	Mark-space ratio (Irrelevant for sets: C, D)	0, 1, . . . 100

and the remaining 14 electrodes were used as configuration inputs. As 14 configuration inputs were used in the experiments, the remaining 6 configuration inputs were set to static -2.3V by Mecobo 3.5 irrespective of the voltage levels set by the genotype for these configuration inputs. In the case of function 15 of set D, four electrodes were used as outputs and 12 electrodes were used as configuration inputs. As 12 configuration inputs were used in the experiments, the remaining 4 configuration inputs were set to static -2.3V by Mecobo 3.5 irrespective of the voltage levels set by the genotype for these configuration inputs.

6.2.2 Genotype Representation

In experiment A, each chromosome used $n_e = 12$ electrodes at a time. In experiments B-D, each chromosome used $n_e = 16$ electrodes at a time.

The values that genes could take are shown in Table 6.3. The electrode index, j takes values $0, 1, \ldots n_e - 1$. The chromosome index, i takes values $0, 1, \ldots d - 1$ for experiments A-C, where d is the number of dimensions of the function optimisation problem. However, in experiment D, the value of i is always 0. The description of the genotype for the experiments is shown in Table 6.4.

In the case of all sets of experiments, the mutated children were created from a parent genotype by mutating a single gene. For example, in experiment A, one gene of 2106 was mutated in the case of the genotype which had 30 chromosomes. In the output genes, only the first $p_{i,j}$ has any effect, others do not have any effect. The gene $p_{i,j}$ decides

Table 6.4: Description of the genotype for function optimisation experiments. The 'Exp.' column shows the set(s) of experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each chromosome of a genotype. The 'Representation of the i^{th} chromosome' column shows the representation of the i^{th} chromosome of a genotype. The 'Representation of a genotype' column shows the representation of a genotype. The 'Genes related to outputs' column shows the gene values of a chromosome, which are related to outputs.

Ex.	No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Representation of the i^{th} chromosome (C_i)	Representation of a genotype	Genes related to output(s)
Set A	6	$p_{i,j}, s_{i,j}, a_{i,j}, f_{i,j}, ph_{i,j}, c_{i,j}$	12X6 =72	$p_{i,0}s_{i,0}a_{i,0}f_{i,0}$ $ph_{i,0}c_{i,0}\dots$ $p_{i,11}s_{i,11}a_{i,11}f_{11}$ $ph_{i,11}c_{i,11}$	$C_0C_1\dots C_{d-1}$	Last 6 genes of i^{th} chromosome: $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$
Set B	5	$p_{i,j}, s_{i,j}, a_{i,j}, f_{i,j}, c_{i,j}$	16X5 =80	$p_{i,0}s_{i,0}a_{i,0}$ $f_{i,0}c_{i,0}\dots$ $p_{i,15}s_{i,15}a_{i,15}$ $f_{15}c_{i,15}$	$C_0C_1\dots C_{d-1}$	Last 5 genes of i^{th} chromosome: $p_{i,15}s_{i,15}a_{i,15}$ $f_{i,15}c_{i,15}$
Set C	2	$p_{i,j}, a_{i,j}$	16X2 =32	$p_{i,0}a_{i,0}\dots p_{i,15}a_{i,15}$	$C_0C_1\dots C_{d-1}$	Last 2 genes of i^{th} chromosome: $p_{i,15}a_{i,15}$
Set D	2	$p_{i,j}, a_{i,j}$	16X2 =32	$p_{i,0}a_{i,0}\dots p_{i,15}a_{i,15}$ (i is always 0)	C_0	Last 4 genes for functions 14,16,17: $p_{i,14}a_{i,14}p_{i,15}a_{i,15}$ Last 8 genes for function 15: $p_{i,12}a_{i,12}\dots$ $p_{i,15}a_{i,15}$

which electrode will be used for the output of the device. Thus, mutations in this gene can choose a different electrode to be used as an output.

Two examples of electrode arrangements associated with chromosomes used in function optimisation experiments with the split genotype technique are shown in Figure 6.3. Two examples of electrode arrangements associated with chromosomes used in function optimisation experiments without the split genotype technique are shown in Figure 6.4.

6.2.3 Output Mapping

As Mecobo 3.0 supports only digital outputs, to determine a real-valued output from a collection of ones in an output buffer, it was decided to use the fraction of ones in

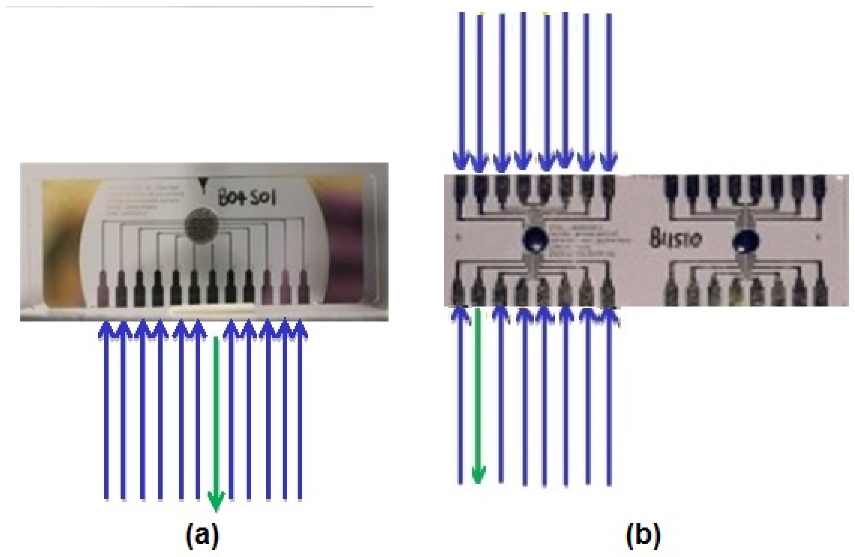


FIGURE 6.3: Two examples of electrode arrangements associated with chromosomes used in function optimisation experiments with the split genotype technique. (a) This type of electrode arrangement is used in set A. The green arrow is used to indicate reading output from the output electrode and blue arrows are used to show configuration inputs being sent to 11 electrodes. (b) This type of electrode arrangement is used in sets B and C. The green arrow is used to indicate reading output from the output electrode and blue arrows are used to show configuration inputs being sent to 15 electrodes.

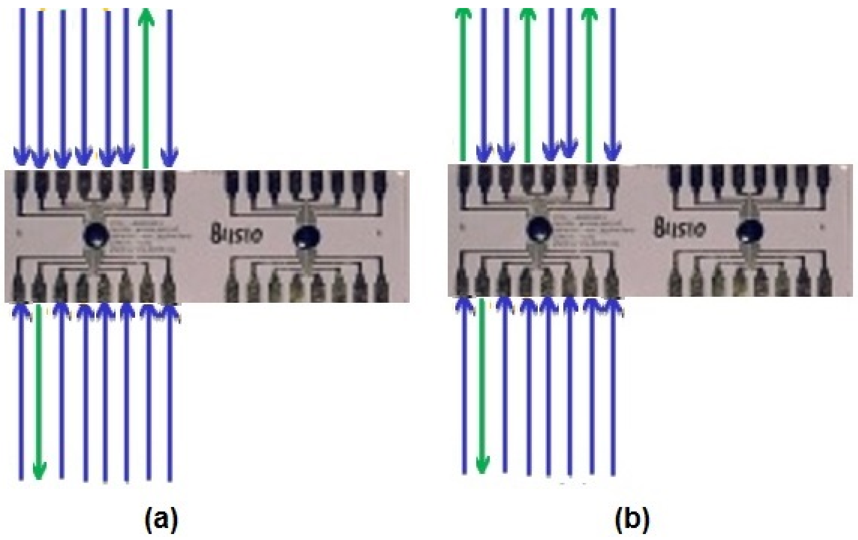


FIGURE 6.4: Two examples of electrode arrangements associated with chromosomes used in function optimisation experiments without the split genotype technique. (a) This type of electrode arrangement is used in set D for functions 14, 16 and 17. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 14 electrodes. (b) This type of electrode arrangement is used in set D for function 15. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 12 electrodes.

experiments A and B.

In experiment A, the input-output timing was 128 milliseconds. Initial findings revealed that the output buffer with an input-output timing of 128 milliseconds never contains more than 40% ones. As a result, before running the function optimisation experiments of set A, an initial evolutionary investigation was performed to discover the typical contents of an output buffer under various conditions. The fraction of ones in the output buffer was calculated to obtain the values of the variables required to optimise functions. However, because the buffer contains a maximum of 40% ones in the case of the input-output timing 128 milliseconds, the fraction of ones was multiplied by 2.5 so that a real-valued output would take values between 0 and 1.

In the initial investigation in experiment A, evolutionary runs were carried out to find the electrode configurations (which electrode is used as an output or configuration input, signal type, amplitude, phase, mark-space ratio, frequency of configuration inputs) that gave different percentages of ones in the output buffer. The different percentages were 0%, 10%, 20%, 30% and 40%. The evolved electrode configurations that gave these percentages were used to seed the initial populations for the evolutionary runs for the function optimisation problems. This was done to ensure a diversity of values in the initial population. No initial investigation was performed for experiments B-D as it is time-consuming and these sets of experiments were performed to compare results with each other. The initial population was selected randomly in these experiments.

The real values (percentage of ones or average of absolute values of samples) determined from the output buffers were linearly mapped to the range of values variables were allowed to take in various optimisation functions. This was done as follows:

Let, max_i be the maximum value and min_i be the minimum value allowed for a variable, x_i in a function optimisation problem. Then the equations used for calculating the linearly mapped output value, x_i for these experiments are shown in Table 6.5.

6.2.4 The Experimental Details

The results of experiment A were compared with Cartesian genetic programming using a (1+4)-evolutionary algorithm over the same number of generations with the same number of runs. It should be noted that in all experiments using both the experimental

Table 6.5: Output mappings of function optimisation experiments. The 'Exp.' column shows the set of experiments. The 'Equation' column shows the equation used for calculating output value. The 'Variable' column defines the variables used in the output mapping equation.

Exp.	Equation	Variables
Set A	$x_i = min_i + 2.5(max_i - min_i)q$ (6.2)	q is the value of the fraction of ones in the output buffer
Set B	$x_i = min_i + (max_i - min_i)q$ (6.3)	q is the value of the fraction of ones in the output buffer
Set C	$x_i = min_i + \frac{(max_i - min_i)r}{4096}$ (6.4)	r is the average of the absolute values of samples in the output buffer (the maximum output value in an output buffer is 4096 (Section 4.1))
Set D	$x_i = min_i + \frac{(max_i - min_i)r_i}{4096}$ (6.5)	r_i is the average of absolute values of samples in the i^{th} output buffer (the maximum output value in an output buffer is 4096 (Section 4.1))

material and Cartesian genetic programming, a child replaced the parent if its fitness was greater than or equal to the parent.

In experiments of set A, twenty-three benchmark functions of function optimisation problem were investigated. The input-output timing was 128 milliseconds, the number of generations was 5000 and the number of runs was 10 in the case of each benchmark function. Only 10 runs were undertaken as it took over 7 days for these experiments. Different functions took different times due to different numbers of dimensions. Elapsed time increased with the number of dimensions.

In Cartesian genetic programming function optimisation experiments, five constant inputs (terminals) were generated randomly in the interval [-1, 1] at the start of each evolutionary run. The function set chosen for the study was defined over the real-valued interval [-1.0, 1.0]. The number of outputs was $n_o = d$, where d is the dimensionality of the optimisation problem. Since the terminals and functions all returned numbers in the interval [-1, 1], the program outputs, q_i also had values defined in this range. However, as the optimisation functions are defined over a variety of intervals, the program

TABLE 6.6: Node function gene values and their definition.

Value	Definition
0	$\sqrt{\lvert z_0 \rvert}$
1	${z_0}^2$
2	${z_0}^3$
3	$(2exp(z_0+1) - e^2 - 1)/(e^2 - 1)$
4	$\sin(z_0)$
5	$\cos(z_0)$
6	$\lvert z_0 \rvert^{\lvert z_1 \rvert}$
7	$\sqrt{({z_0}^2 + {z_1}^2)/2}$
8	$(z_0 + z_1)/2$
9	$(z_0 - z_1)/2$
10	$z_0 z_1$
11	**if** $\lvert z_1 \rvert < 10^{-10}$ **then** 1 **else if** $\lvert z_1 \rvert > \lvert z_0 \rvert$ **then** z_0/z_1 **else** z_1/z_0
12	**if** $z_0 > z_1$ **then** $z_2/2$ **else** $1 - z_2/2$

outputs, q_i, were needed to be mapped to the intervals (min_i and max_i) defined in the optimisation problem for a variable, x_i. Equation 6.6 gives the mapping.

$$x_i = \frac{max_i - min_i}{2} q_i + \frac{max_i + min_i}{2}. \tag{6.6}$$

Three mutation parameters (defined in percentages) were used in the case of all function optimisation experiments of Cartesian genetic programming: a probability of mutating connections, μ_c, functions, μ_f and outputs, μ_o. In all experiments, $\mu_c = 0.01$, $\mu_f = 0.03$, $\mu_o = 0.04$, the number of rows, $n_r = 1$ and the number of columns, $n_c = 100$ with nodes being allowed to connect to any previous node. The function set chosen for the function optimisation experiments of Cartesian genetic programming is shown in Table 6.6.

It should be noted that Cartesian genetic programming was applied to 20 function optimisation problems [Vesterstrom and Thomsen (2004)] before, and the results were compared[1] with differential evolution (DE) [Storn and Price (1997); Price *et al.* (2005)], particle swarm optimisation (PSO) [Kennedy and Eberhart (1995); Eberhart *et al.* (2001)] and an evolutionary algorithm (SEA) [Miller and Mohid (2013)]. In comparison results, it was found that in 15 out of 20 benchmarks, Cartesian genetic programming is the

[1]Based on average results over 30 independent runs and 5,00,000 evaluations for each run

same or better than DE and in 19 out of 20 cases, Cartesian genetic programming is the same or better than PSO or SEA.

6.2.4.1 The Results

The results of experiment A and the results of Cartesian genetic programming were compared using the U-test and KS-test [Hollander and Wolfe (1973)]. The effect size [Vargha and Delaney (2000)] statistic was also computed.

The experiments of set A show that in 15 out of 23 functions, the best results with the experimental material are within 8% of the optimum. Of these, in 7 cases, the best results with the experimental material are equal to optimum results. In 13 out of 23 functions, the average results with the experimental material are within 8% of the optimum. Of these, in 5 cases, the average results with the experimental material are equal to optimum results. In 10 out of 23 functions, the best results of experimental material are better than or equal to the best results of Cartesian genetic programming. In the case of 12 out of 23 functions, the average results of experimental material are better than or at least equal to the average results of Cartesian genetic programming. The detailed results and the statistical significance tests are shown in Table 6.7.

To compare the performances of two hardware platforms (Mecobo 3.0 and Mecobo 3.5), the results of experiments B and C have been used. Statistical significance (U-test, KS-test and effect size) tests have been performed to compare the results. The detailed results and the statistical significance tests are shown in Table 6.8. It has been found from the results that no average result of Mecobo 3.5 is equal to optimum in the case of any of these nine functions, the average result of Mecobo 3.0 is equal to optimum in function 16. The best results of Mecobo 3.5 are equal to optimum in the case of 5 out of 9 functions (functions 14, 16, 17, 18, 21) and the best results of Mecobo 3.0 are equal to optimum in the case of 6 out of 9 functions (functions 14, 16, 17, 18, 19, 22). If the average results are compared, the performance of Mecobo 3.5 is better than the performance of Mecobo 3.0 as the average results with Mecobo 3.5 are better than the average results with Mecobo 3.0 for 5 out of 9 functions (functions 14, 17, 18, 21, 23). However, if the best results are compared, Mecobo 3.0 performs better than Mecobo 3.5 as the best results of Mecobo 3.0 are better than the best results of Mecobo 3.5 in the

Table 6.7: Comparative results of the experimental material (experiments of set A) with Cartesian genetic programming on 23 benchmark optimisation functions. The evolution-in-materio experiments were performed using material sample 4 and Mecobo 3.0. The first column indicates the benchmark function number associated with the function optimisation problem. In the case of experimental material and Cartesian genetic programming, the number of generations was 5000 and the number of runs was 10. The 'Res.' column shows whether the results of the experimental material are within 8% of the optimum or not. The first result of this column shows the comparison between the best result of the experimental material and the optimum, and the second result shows the comparison between the average result of the experimental material and the optimum. '✓' in this column indicates the result is within 8% of the optimum and 'X' indicates the result is not within 8% of the optimum. The 'Co. res.' column shows the comparisons of the best and average results of the experimental material with Cartesian genetic programming. The '=' in this column indicates the result with the material is equal to the result of Cartesian genetic programming, '+' indicates the result with the material is better than the result of Cartesian genetic programming, and '-' indicates the result with the experimental material is worse than the result of Cartesian genetic programming. The first result of this column shows the comparison of the best results and the second result of this column shows the comparison of the average results. 'U-t', 'KS-t' and 'E. s.' (effect size) (L = large, M = medium, S = small) columns show results of statistical significance tests. The statistical significance tests have been performed over the results of all 10 runs of all 23 functions. '✓' in 'U-t' and 'KS-t' columns indicates that the difference between the two datasets is statistically significant and 'X' indicates that the difference is statistically not significant.

F. n.	Expected output	Best result of exp. material	Average result of exp. material	Best result of Cartesian genetic prog.	Average result of Cartesian genetic prog.	Res.	Co. res.	U-t	KS-t	E. s.
1	0	1.548E-05	3.262E-05	0	0	✓ ✓	- -	✓	✓	L
2	0	1.014E-02	2.372E-02	0	0	✓ ✓	- -	✓	✓	L
3	0	127.902	1614.87	3.938	3.065E+03	X X	- +	X	X	L
4	0	2.038E-01	5.600E-01	0	0	X X	- -	✓	✓	L
5	0	1.137E-01	3.871E-01	27.690461	37.668384	X X	+ +	✓	✓	L
6	0	0	0	0	0	✓ ✓	= =	X	X	S
7	0	3.813E-03	1.071E-02	1.399E-03	1.083E-02	✓ ✓	- +	X	X	S
8	-12569.487	-12451.028	-12255.608	-12569.450	-12569.330	✓ ✓	- -	✓	✓	L
9	0	2.992018	5.634806	0	0	X X	- -	✓	✓	L
10	0	1.166E-02	3.491E-02	1.440E-16	1.440E-16	✓ ✓	- -	✓	✓	L
11	0	3.395E-03	8.345E-02	0	0	✓ ✓	- -	✓	✓	L
12	0	1.047E-01	1.151E-01	3.072E-01	6.514E-01	X X	+ +	✓	✓	L
13	0	4.336E-05	1.587E-04	7.416E-06	1.176E-03	✓ ✓	- +	X	X	S
14	0.9980038	0.9980038	0.9980038	0.9980038	0.9980038	✓ ✓	= =	X	X	S
15	0.0003075	3.085E-04	3.431E-04	4.437E-04	1.059E-03	✓ X	+ +	✓	✓	L
16	-1.0316	-1.0316	-1.0316	-1.0316	-1.0308	✓ ✓	= +	X	X	M
17	0.397887	0.397887	0.397916	0.397887	0.397994	✓ ✓	= +	X	X	M
18	3.0	3.0	21.906419	3.0	3.0	✓ X	= -	✓	X	L
19	-3.86	-3.86	-3.86	-3.86	-3.86	✓ ✓	= =	X	X	S
20	-3.32	-3.32	-3.32	-3.32	-3.29	✓ ✓	= +	X	X	L
21	-10.15	-5.10	-5.10	-10.15	-8.64	X X	- -	✓	✓	L
22	-10.40	-5.13	-5.13	-10.40	-8.82	X X	- -	✓	✓	L
23	-10.54	-5.18	-5.18	-10.54	-9.46	X X	- -	✓	✓	L

Table 6.8: Comparative results of material sample 1 on different hardware platforms (Mecobo 3.0 and Mecobo 3.5) on 9 benchmark optimisation functions using experiments B and C. Experiment B used Mecobo 3.0 and experiment C used Mecobo 3.5. The first column indicates the benchmark function number associated with the function optimisation problem. In the case of experiments B and C, the number of generations was 1000 and the number of runs was 20. The 'Res. of Mec. 3.5' column shows whether the results of Mecobo 3.5 are equal to optimum or not and 'Res. of Mec. 3.0' column shows whether the results of Mecobo 3.0 are equal to optimum or not. The first results of these columns show the comparisons between the best results of the experimental material and the optimum, and the second results show the comparisons between the average results of the experimental material and the optimum. '✓' in these columns indicates the result is equal to optimum and 'X' indicates the result is not equal to optimum. The 'Co. res.' column shows the comparisons of the best and average results of Mecobo 3.5 with Mecobo 3.0. The '=' in this column indicates that the results of both hardware platforms are equal, '+' indicates the result with Mecobo 3.5 is better than the result of Mecobo 3.0 and '-' indicates the result with Mecobo 3.5 is worse. The first result of this column shows the comparison of the best results and the second result of this column shows the comparison of the average results. 'U-t', 'KS-t' and 'E. s.' (effect size) (L = large, M = medium, S = small) columns show results of statistical significance tests. The statistical significance tests have been performed over the results of all 20 runs of all 9 functions. '✓' in 'U-t' and 'KS-t' columns indicates that the difference between the two datasets is statistically significant and 'X' indicates that the difference is statistically not significant.

F. n.	Expected output	Best result of Mecobo 3.5	Average result of Mecobo 3.5	Best result of Mecobo 3.0	Average result of Mecobo 3.0	Res. of Mec. 3.5	Res. of Mec. 3.0	Co. res.	U-t	KS-t	E. s.
14	0.9980038	0.9980038	1.1465161	0.9980038	26.24430	✓ X	✓ X	= +	X	X	S
15	0.0003075	0.0014874	0.0721622	0.0012233	0.0180017	X X	X X	- -	✓	✓	L
16	-1.0316	-1.0316	-0.91	-1.0316	-1.0316	✓ X	✓ ✓	= -	X	X	L
17	0.397887	0.397887	0.492799	0.397887	1.135623	✓ X	✓ X	= +	X	X	M
18	3.0	3.0	149.5	3.0	769.9	✓ X	✓ X	= +	X	X	S
19	-3.86	-1.00	-1.00	-3.86	-1.25	X X	✓ X	- -	X	X	M
21	-10.15	-10.15	-5.31	-5.06	-3.80	✓ X	X X	+ +	✓	✓	L
22	-10.40	-5.09	-5.09	-10.40	-5.34	X X	✓ X	- -	✓	✓	L
23	-10.54	-5.13	-4.93	-5.13	-4.78	X X	X X	= +	✓	✓	L

case of functions 15, 19 and 22 and the best results are equal in the case of functions 14, 16, 17, 18 and 23.

To compare the performance of using the split genotype technique against the performance without using the split genotype technique, the results of experiments C and D were used. Statistical significance (U-test, KS-test and effect size) tests have been performed to compare the results. The detailed results are shown in Table 6.9. It has been found from the results that no average result of any of these experiments is equal to optimum in the case of any of these four functions. The best results of experiments C and D are equal to optimum in the case of 3 out of 4 functions (functions 14, 16, 17). If the average results are compared, the performance of experiment C (with the split genotype technique) is better than the performance of experiment D (without using the

Table 6.9: Comparative performance results of the split genotype technique (experiment C) versus the performance without using the split genotype technique (experiment D). The first column indicates the benchmark function number associated with the function optimisation problem. The comparisons are performed using material sample 1 and Mecobo 3.5 on 4 benchmark optimisation functions. In the case of experiments C and D, the number of generations was 1000 and the number of runs was 20. The 'Res. of set C' column shows whether the results of set C (experiments with the split genotype technique) are equal to optimum or not and 'Res. of set D' column shows whether the results of set D (experiments without the split genotype technique) are equal to optimum or not. The first results of these columns show the comparisons between the best results of the experimental material and the optimum and the second results show the comparisons between the average results of the experimental material and the optimum. '✓' in these columns indicates the result is equal to optimum and 'X' indicates the result is not equal to optimum. The 'Co. res.' column shows the comparisons of the best and average results of sets C and D. The '=' in this column indicates that the results of both sets are equal, '+' indicates the result of set C is better than the result of set D and '-' indicates the result of set C is worse. The first result of this column shows the comparison of the best results and the second result of this column shows the comparison of the average results. 'U-t', 'KS-t' and 'E. s.' (effect size) (L = large, M = medium, S = small) columns show results of statistical significance tests. The statistical significance tests have been performed over the results of all 20 runs of all 4 functions. '✓' in 'U-t' and 'KS-t' columns indicates that the difference between the two datasets is statistically significant and 'X' indicates that the difference is statistically not significant.

F. n.	Expected output	Best result of set C	Average result of set C	Best result of set D	Average result of set D	Res. of set C	Res. of set D	Co. res.	U-t	KS-t	E. s.
14	0.9980038	0.9980038	1.1465161	0.9980038	1.5051768	✓ X	✓ X	= +	✓	✓	L
15	0.0003075	0.0014874	0.0721622	0.0004352	0.1115308	X X	X X	- +	X	X	S
16	-1.0316	-1.0316	-0.91	-1.0316	-0.7789	✓ X	✓ X	= +	✓	✓	L
17	0.397887	0.397887	0.492799	0.397887	0.410195	✓ X	✓ X	= -	✓	✓	L

split genotype technique) as experiment C obtained better results than the results of experiment D in 3 out of 4 functions (functions 14-16). However, if the best results are compared, the performance of experiment D is better than the performance of experiment C as the best result of experiment D is better than the result of experiment C in the case of function 15 and the best results of both experiments are equal in the case of other three functions.

A summary of outcomes from these experiments is given as follows:

- The comparison results of experimental material (material sample 4 according to Table 4.3) and Cartesian genetic programming show that in 10 out of 23 functions, the best results of experimental material are better than or equal to the best results of Cartesian genetic programming. In the case of 12 out of 23 functions, the average results of experimental material are better than or equal to the average results of Cartesian genetic programming.

- In the case of 15 out of 23 functions, the best results of experimental material (material sample 4) are within 8% of the optimum and in the case of 13 out of 23 functions, the average results of the experimental material are within 8% of the optimum.
- According to the experiments of function optimisation problems (functions 14-19, 21-23), if the average results are compared, the performance of Mecobo 3.5 is better than the performance of Mecobo 3.0. However, if the best results are compared, the performance of Mecobo 3.0 is better than the performance of Mecobo 3.5.
- According to the experiments of function optimisation problems (functions 14-17) using Mecobo 3.5, if the average results are compared, the performance of using the split genotype technique is better than the performance without using the split genotype technique. However, if the best results are compared, the performance without using the split genotype technique is better than the performance of using the split genotype technique.

6.3 Solving Bin Packing Problems

Bin packing is a well-studied NP-hard problem [Coffman *et al.* (1997)]. In the bin packing problem, each item, i with weight w_i from a total number of items, n_t, has to be placed in a bin. Each bin, however has a maximum weight capacity, c. The objective is to place all the items in the least number of bins such that no bin has its weight limit exceeded [Horowitz *et al.* (2007)].

Scholl and Klein have collated bin packing benchmarks [Scholl and Klein (2003)]. The datasets are divided into three classes according to difficulty. The best result for each dataset has been obtained by Scholl et al. [Scholl *et al.* (1997)] using an algorithm called BISON which combines a successful heuristic meta-strategy tabu search and a branch and bound procedure. Many pieces of work were carried out using these bin packing benchmarks by many researchers [Schwerin and Wäscher (1997); Fleszar and Hindi (2002); Alvim *et al.* (2002); Schoenfield (2002)]. For the experiment used here, four instances from each difficulty class have been chosen.

Table 6.10: The experimental settings of all sets of bin packing experiments. The 'Benchmark' column shows the bin packing benchmark used in the experiments. The 'No. of gen' and 'No. of run' columns show the number of generations and number of runs of the experiments respectively. The 'No. of out.', 'No. of conf.' and 'Mut. rate' columns show the number of outputs, number of configuration inputs and mutation rate respectively. The mutation rate (m_n) is defined to be the number of discrete mutations made in the entire collection of chromosomes. It should be noted that the experiments of all sets used Mecobo 3.0, material sample 4 (according to Table 4.3), 12 electrodes of the electrode array, 128 milliseconds for the input-output timing and 25 KHz for the output sampling frequency.

Set	Benchmark	No. of gen.	No. of run	No. of out.	No. of conf.	Mut. rate
A	HARD0	5000	20	1	11	1
B	HARD0	5000	20	2	10	1
C	HARD0	5000	20	4	8	1
D	HARD0	5000	20	5	7	1
E	HARD0	5000	20	10	2	1
F	HARD0	5000	20	2	10	2
G	N1C1W1_A	5000	20	2	10	1
H	N2C2W2_B	5000	20	2	10	1
I	N3C3W4_R	5000	20	2	10	1
J	N4C2W1_M	5000	20	2	10	1
K	N1W1B1R0	5000	20	2	10	1
L	N3W1B2R3	5000	20	2	10	1
M	N4W2B3R5	5000	20	2	10	1
N	N2W3B1R9	5000	20	2	10	1
O	HARD3	5000	20	2	10	1
P	HARD4	5000	20	2	10	1
Q	HARD9	5000	20	2	10	1
R	N4W2B3R5	25000	10	2	10	1

Table 6.11: The motives for performing the bin packing experiments. The first column shows the sets of experiments. The second column shows the motive.

Experiments	Motive
Sets A-E	To find out the best combination (combination of the number of outputs and configuration inputs) of electrodes, which gives the best result.
Sets B, F	To find out the best mutation rate, which gives the best result.
Sets B, G-Q	To investigate the performances of evolution-in-materio on different bin packing benchmarks.
Set R	To investigate the performance of evolution-in-materio on bin packing problem with more generations.

6.3.1 Methodology

Eighteen different sets (A-R) of experiments were performed. The experimental settings of all sets of bin packing experiments are described in Table 6.10. The motives for performing the bin packing experiments are described in Table 6.11.

All of the experiments were performed with an electrode array having twelve electrodes with material sample 4 (according to Table 4.3) and Mecobo 3.0. Bin packing problems require no inputs. The total number of outputs must equal the number of items that need to be packed into bins. Since bin packing problems typically have 50 or more items, multiple chromosomes must be used, where each chromosome defines a number of configuration inputs to the electrode array and the remaining outputs supply recorded values in output buffers. This means, the split genotype technique (Section 6.1) is required to be used. In bin packing problems, the number of chromosomes required is given by the number of items to be packed into bins divided by the number of outputs chosen per chromosome. For instance, for a problem with 50 items, if two outputs are chosen, the genotype requires 25 chromosomes. In this case, there will be 10 configuration inputs applied for each chromosome processed.

In all sets of experiments, a series of output values (0 or 1) were read from buffer(s) of samples taken from output electrode(s). The output values read from the electrode(s) were linearly mapped between values -1.0 to 1.0. These values were then used to define the index of the bin (bin_i) in which the item i of the bin packing problem would be placed. So, the total number of outputs must be equal to the total number of items (n_t). The electrode array that was used in the experiments has only twelve electrodes, and the split genotype technique was used to obtain the required number of outputs from the device so that the number of required values defined by the computational problem (bin packing problem) could be handled. The process of generating bin indexes in bin packing experiments using the split genotype technique is shown in Figure 6.5.

6.3.2 Genotype Representation

In the case of all experiments, each chromosome used $n_e = 12$ electrodes at a time. The values that genes could take are shown in Table 6.12. The electrode index, j takes values $0, 1, \ldots n_e - 1$. The chromosome index, i takes values $0, 1, \ldots d - 1$. If the total number of outputs (number of items) of bin packing problem is n_t and the number of outputs in each chromosome is n_o, then $d = n_t/n_o$.

The description of the genotype for the experiments is shown in Table 6.13.

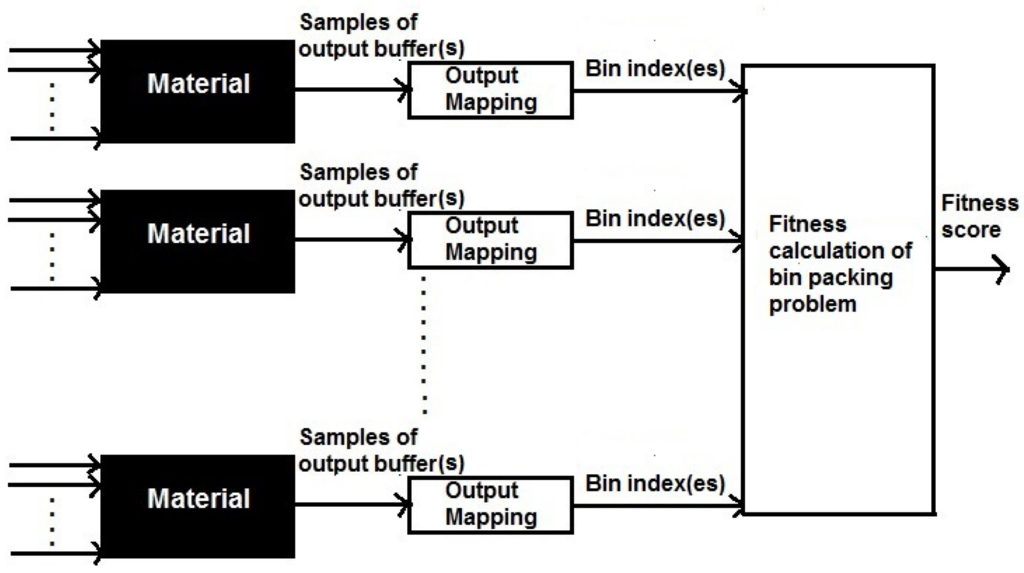

Samples of output buffer(s)
Material
Output Mapping
Bin index(es)
Fitness calculation of bin packing problem
Fitness score

FIGURE 6.5: The process of generating bin indexes in bin packing experiments using the split genotype technique.

TABLE 6.12: Description of the genes for bin packing experiments.

Gene symbol	**Signal applied to, or read from the i^{th} chromosome and the j^{th} electrode**	**Allowed values**
$p_{i,j}$	Which electrode is used	0, 1, 2 . . . 11
$s_{i,j}$	Type	0 (static) or 1(square wave)
$a_{i,j}$	Amplitude	0 , 1
$f_{i,j}$	Frequency	500 ,501 . . . 10K
$ph_{i,j}$	Phase	1, 2 . . . 10
$c_{i,j}$	Mark-space ratio	0, 1, . . . 100

In the output genes, only the $p_{i,j}$ has any effect, others do not have any effect. The gene $p_{i,j}$ decides which electrode will be used for an output of the device. Thus, mutations in these genes can choose a different electrode to be used as an output.

Five examples of electrode arrangements associated with chromosomes used in bin packing experiments are shown in Figure 6.6.

6.3.3 Output Mapping

In the case of all experiments, to determine a real-valued output from a collection of ones in an output buffer, it was decided to use the fraction of ones. This was chosen

Table 6.13: Description of the genotype for bin packing experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each chromosome of a genotype. The 'Representation of the i^{th} chromosome' column shows the representation of the i^{th} chromosome of a genotype. The 'Representation of a genotype' column shows the representation of a genotype. The 'Exp.' column shows the set(s) of experiments. The 'Genes related to outputs' column shows the gene values of a chromosome, which are related to outputs.

No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Representation of the i^{th} chromosome (C_i)	Representation of a genotype	Ex.	Genes related to output(s)
6	$p_{i,j}, s_{i,j}, a_{i,j}, f_{i,j}, ph_{i,j}, c_{i,j}$	12X6 =72	$p_{i,0}s_{i,0}a_{i,0}f_{i,0}$ $ph_{i,0}c_{i,0}\dots$ $p_{i,11}s_{i,11}a_{i,11}f_{11}$ $ph_{i,11}c_{i,11}$	$C_0C_1\dots C_{d-1}$	Set A	Last 6 genes of i^{th} chromosome: $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$
					Set B, F-R	Last 12 genes of i^{th} chromosome: $p_{i,10}s_{i,10}a_{i,10}f_{i,10}$ $ph_{i,10}c_{i,10}$ $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$
					Set C	Last 24 genes of i^{th} chromosome: $p_{i,8}s_{i,8}a_{i,8}f_{i,8}$ $ph_{i,8}c_{i,8}\dots$ $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$
					Set D	Last 30 genes of i^{th} chromosome: $p_{i,7}s_{i,7}a_{i,7}f_{i,7}$ $ph_{i,7}c_{i,7}\dots$ $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$
					Set E	Last 60 genes of i^{th} chromosome: $p_{i,2}s_{i,2}a_{i,2}f_{i,2}$ $ph_{i,2}c_{i,2}\dots$ $p_{i,11}s_{i,11}a_{i,11}f_{i,11}$ $ph_{i,11}c_{i,11}$

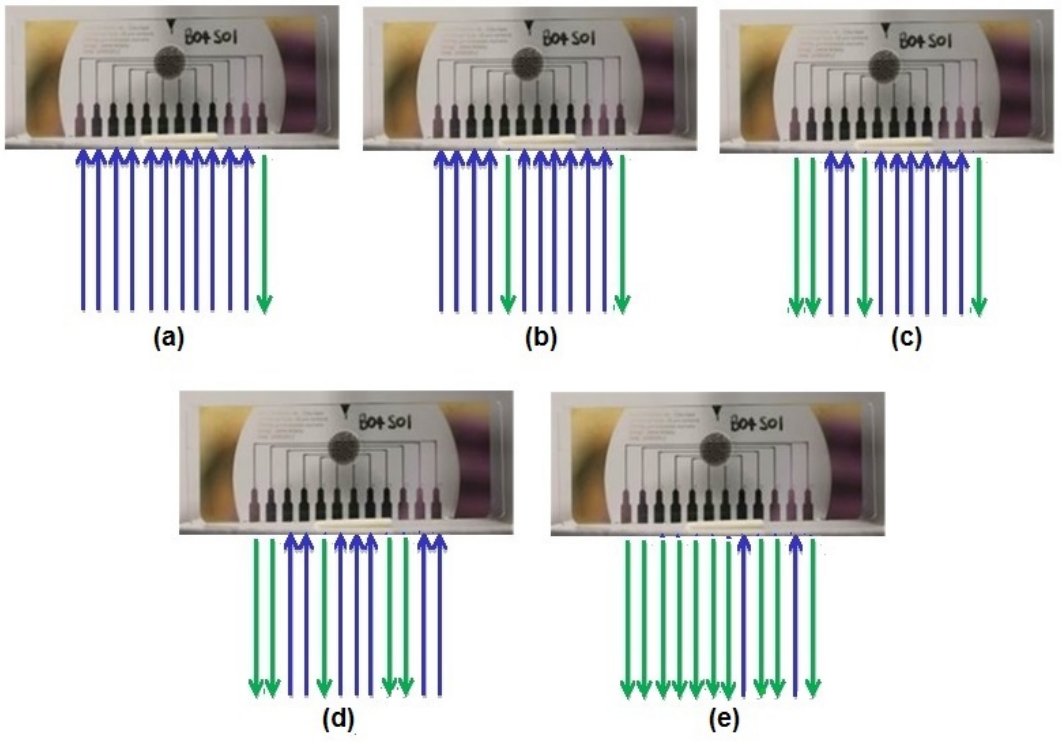


FIGURE 6.6: Five examples of electrode arrangements associated with chromosomes used in bin packing experiments (a) This type of electrode arrangement is used in set A. The green arrow is used to indicate reading output from the output electrode and blue arrows are used to show configuration inputs being sent to 11 electrodes. (b) This type of electrode arrangement is used in sets B and F-R. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 10 electrodes. (c) This type of electrode arrangement is used in set C. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 8 electrodes. (d) This type of electrode arrangement is used in set D. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 7 electrodes. (e) This type of electrode arrangement is used in set E. The green arrows are used to indicate reading outputs from the output electrodes and blue arrows are used to show configuration inputs being sent to 2 electrodes.

purely for simplicity and it is possible that other mappings of bits of the buffer to a real number could have worked better. As initial findings revealed that the output buffer with an input-output timing of 128 milliseconds never contains more than 40% ones, the fraction of ones was multiplied by 2.5 so that a real-valued output would take values between 0 and 1. This value is denoted for the buffer, buf_i by q_i.

The value, q_i was linearly mapped to value, x_i in the interval [-1.0, 1.0] using Equation 6.7.

$$x_i = -1.0 + 2.0q_i \tag{6.7}$$

The linearly mapped output value, x_i corresponding to each chromosome was used to decide the bin index, bin_i which denotes which bin item, i will be placed in. Assuming the total number of items is n_t, the bin index is given by Equation 6.8.

$$bin_i = \lfloor n_t \frac{(x_i + 1.0)}{2 + \epsilon} \rfloor \tag{6.8}$$

The floor function, $\lfloor z \rfloor$ returns the nearest integer less than or equal to its argument, z. *epsilon* is a very small positive quantity. Essentially, Equation 6.8 divides the interval [-1, 1] into n_t equal intervals corresponding to bins, so that the mapped output value decides which bin an item will be placed in.

For instance, assuming the number of items, $n_t = 50$, if x_i is -1.0, bin_i is 0, and if x_i is 1.0, bin_i is 49.

6.3.4 Fitness Calculation

The fitness calculation for bin packing experiments is described as follows:

Let, c is the bin capacity of each bin, b_j is total weight of j^{th} overflowing bin, i.e. the summation of weights of all items placed in that bin, n_{ob} is the number of bins that have exceeded their capacity, n_{ub} is the number of unused bins. The fitness is calculated by the Equation 6.9.

$$fitness = \begin{cases} n_{ub}, & \text{If there is no bin overflowing} \\ \sum_{j=1}^{j=n_{ob}} (c - b_j), & \text{Otherwise} \end{cases} \tag{6.9}$$

Thus, a fitness less than zero indicates at least one bin is over capacity. If all bins are used and no bin overflows, then the fitness is zero. If no bin overflows and some bins are unused, then the fitness is a positive value equal to the number of unused bins. Thus,

Table 6.14: Comparative results of different electrode combinations associated with chromosomes (sets A-E). The experiments were performed using the bin packing benchmark HARD0 [Scholl and Klein (2003)] and in each of the experiments, a single mutation was used to generate a child. The first column shows the set of experiments. In the second column, $p_{y,z}$ denotes the electrode combination, where y is the number of electrodes used as outputs and z is the number of electrodes used as configuration inputs. The third column indicates the average result of 20 runs. The fourth column indicates the best result of all 20 runs. 'Overflow' is used where at least one bin was filled beyond its capacity.

Set	Pin configuration	Average minimum number used bins	Best minimum number used bins
A	$p_{1,11}$	Overflow	69
B	$p_{2,10}$	70.75	68
C	$p_{4,8}$	Overflow	72
D	$p_{5,7}$	Overflow	73
E	$p_{10,2}$	Overflow	71

maximisation of fitness drives genotypes towards representing the smallest number of non-overflowing bins.

6.3.5 The Experiments and the Results

In all sets of experiments, a (1+4)-evolutionary algorithm was used and a child replaced the parent if its fitness was greater than or equal to the parent.

Twelve benchmarks of bin packing problems were used in the experiments (four from each difficulty class). The benchmark HARD0 (the first benchmark in the hardest category) was used to determine how many outputs to use in each chromosome and the best mutation rate to use in the later experiments. It should be noted that the mutation rate, m_n, is defined to be the number of discrete mutations made in the entire collection of chromosomes. Five different sets (sets A-E) of experiments were performed with HARD0 to determine the best number of outputs per chromosome. For all of the experiments of sets A-E, a single mutation (m_n=1) was chosen. The results are shown in Table 6.14.

Investigations were performed using HARD0 with two different values of m_n to see which gave the best result using experiments B and F. The mutation rate used was either one or two ($m_n = 1$ or $m_n = 2$). The results are shown in Table 6.15.

It has been found that the best mutation rate is one (m_n=1) and the best electrode combination is to use two electrodes as outputs and ten as configuration inputs. These

Table 6.15: Comparative results of two different mutation rates (m_n) applied in experiments B and F. Here, m_n is defined to be the number of discrete mutations made in the entire collection of chromosomes. The experiments have been performed using the bin packing benchmark HARD0 [Scholl and Klein (2003)]. The first column shows the set of experiments. The second column shows the mutation rate. The third column shows the average result of 20 runs. The fourth column indicates the best result of all 20 runs. The electrode combination used for all the experiments mentioned in this table is 2 electrodes as outputs and 10 as configuration inputs as this electrode combination gave the best result according to Table 6.14.

Set	Mutation rate	Average Minimum number used bins	Best minimum number used bins
B	1	70.75	68
F	2	73.3	71

parameter settings were used in all other sets (G-R) of experiments. In the case of experiments A-Q, the number of generations was 5000 and the number of runs was 20. Experiments B and G-Q were used to investigate the performances of evolution-in-materio on different bin packing benchmarks. The results are shown in Table 6.16. It should be noted that 20 evolutionary runs (5000 generations each) on each benchmark problem took more than 2 days.

The experiments show that on 3 out of 12 benchmarks, the average and best results of experimental material are within 15% of the optimum. In two cases, the results of all runs were unable to find a solution in which all bins were within capacity. In one set of evolutionary runs, some of the runs could not find solutions in which all bins were within capacity but other did.

In further experiments, one of the difficult benchmark problems (N4W2B3R5) was investigated over longer evolutionary runs. In experiment R, ten runs of 25000 generations were performed with the N4W2B3R5 benchmark. The average result of 10 runs was 128 and best result in 10 runs was 125, where the optimum is 101.

The final gene values of configuration inputs were examined for one single bin packing problem and no recognisable pattern was found. This is not surprising as the number of gene values is very large. For example, for 50 item bin packing problem with 10 configuration inputs and 2 outputs, requires 25 chromosomes and 25*72=1800 genes (per genotype).

In addition, it was also examined whether the material was an essential part of the experiment by attempting to evolve solutions using an electrode array containing no

Table 6.16: The experimental results on twelve bin packing benchmark problems (sets B, G-Q). In all of these experiments, the number of generations was 5000 and the number of runs was 20. Two electrodes were used as outputs and ten were supplied with configuration inputs. A single mutation was used to generate a child in the evolutionary algorithm ($m_n = 1$). The first column shows the set of experiments. The second column shows the benchmark of bin packing problem on which experiments were performed. The third column shows the optimum result (expected result) of the bin packing problem. The fourth column indicates the difficulty class to which the benchmark data belongs to (according to [Scholl and Klein (2003)]). The fifth column shows the number of items of bin packing problem. The sixth column indicates the average result of all runs. The seventh column indicates the best result of all runs. 'Overflow' is used where at least one bin overflowed. The 'Result' column shows whether the results of the experimental material are within 15% of the optimum or not. The first result of this column shows the comparison between the average result of the experimental material and the optimum and the second result shows the comparison between the best result of the experimental material and the optimum. '✓' in this column indicates the result is within 15% of the optimum and 'X' indicates the result is not within 15% of the optimum.

Set	Benchmark	Optimum result	Class	Total number of items	Minimum number of used bins (average)	Minimum number of used bins (best)	Result
G	N1C1W1_A	25	1	50	27.4	27	✓ ✓
H	N2C2W2_B	56	1	100	63	60	✓ ✓
I	N3C3W4_R	87	1	200	Overflow	103	X X
J	N4C2W1_M	217	1	500	Overflow	Overflow	X X
K	N1W1B1R0	18	2	50	20.2	20	✓ ✓
L	N3W1B2R3	65	2	200	81.7	78	X X
M	N4W2B3R5	101	2	500	Overflow	Overflow	X X
N	N2W3B1R9	15	2	100	20.45	19	X X
B	HARD0	56	3	200	70.75	68	X X
O	HARD3	55	3	200	71.15	68	X X
P	HARD4	57	3	200	70.05	69	X X
Q	HARD9	56	3	200	72.4	70	X X

material. One of the bin packing benchmarks (dataset N1C1W1_A) was chosen. For this experiment, two electrodes were used as outputs and ten as configuration inputs. A single evolutionary run was carried out for up to 1000 generations using a single mutation to create each child. It has been found that no evolution happened in this case. The fitness value in the first generation is the same as the fitness value of the 1000^{th} generation. The fitness value indicates that all items went into the same bin. Output buffers of samples were also investigated and it was found that the buffers were always full of zeroes.

Another experiment was performed using the same bin packing benchmark problem (dataset N1C1W1_A) to see whether single-walled carbon nanotubes are required in the material mixture for computation or not. Material sample 10 (according to Table 4.3)

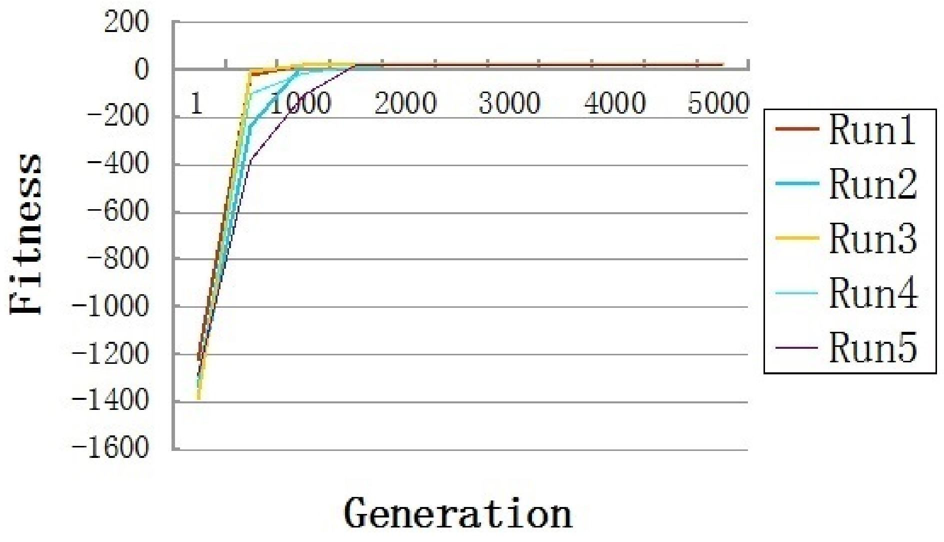


Figure 6.7: Fitness value vs. number of generations for five evolutionary runs using experiment G (benchmark N1C1W1_A).

has been used in this experiment, where the material contains only polymer (0% single-walled carbon nanotubes in PMMA). For this experiment, two electrodes were used as outputs and ten as configuration inputs. A single evolutionary run was carried out for up to 1000 generations using a single mutation to create each child. It has been found that no evolution happened in this case. The fitness value in the first generation was the same as the fitness value of the 1000^{th} generation. Output buffers of samples were also investigated and it was found that the buffers were always full of zeroes.

Investigations were performed using material sample 8 and material sample 9 (with 0.02% and 0.01% single-walled carbon nanotubes in PMMA respectively). These investigations were carried out for up to 1000 generations for one run with dataset N1C1W1_A (2 electrodes as outputs and 10 as configuration inputs). A single mutation was used to create a child in these investigations. It was found that no evolution took place with 0.01% single-walled carbon nanotubes (material sample 9) and the values of output buffers were always zero. When an experiment was started with material sample 8 (i.e. with 0.02% single-walled carbon nanotubes), evolution happened with mixtures of 0 and 1 in the output buffers. All of these investigation results showed that single-walled carbon nanotubes are required in the material mixture for computation.

Figure 6.7, 6.8, 6.9, 6.10 and 6.11 show the changes in fitness values in different generations for five evolutionary runs using experiments G, I, K, M and O.

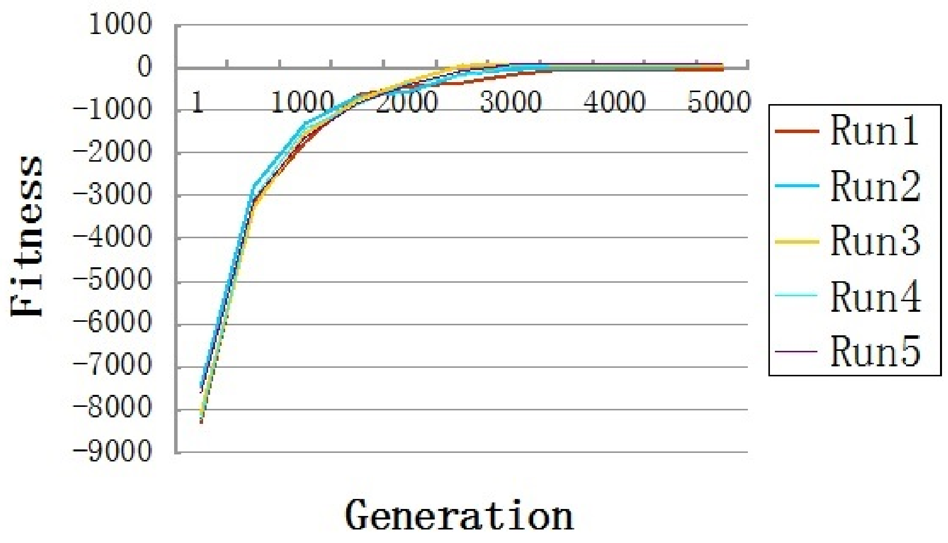


FIGURE 6.8: Fitness value vs. number of generations for five evolutionary runs using experiment I (benchmark N3C3W4_R).

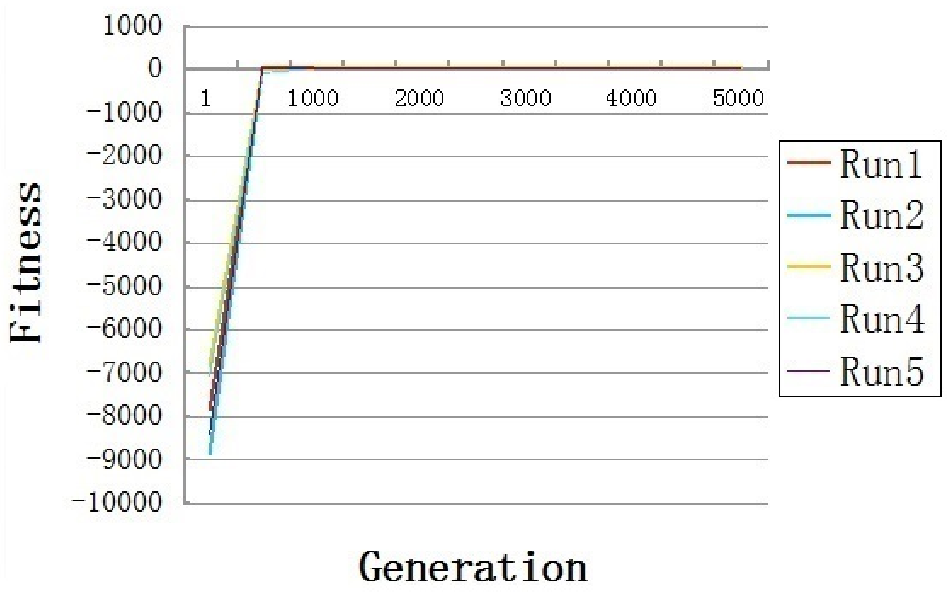


FIGURE 6.9: Fitness value vs. number of generations for five evolutionary runs using experiment K (benchmark N1W1B1R0).

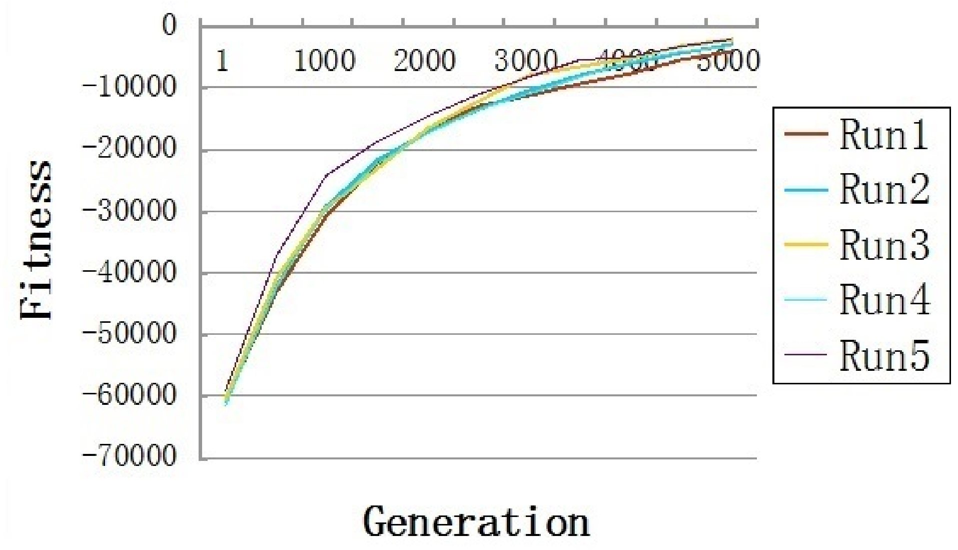


FIGURE 6.10: Fitness value vs. number of generations for five evolutionary runs using experiment M (benchmark N4W2B3R5).

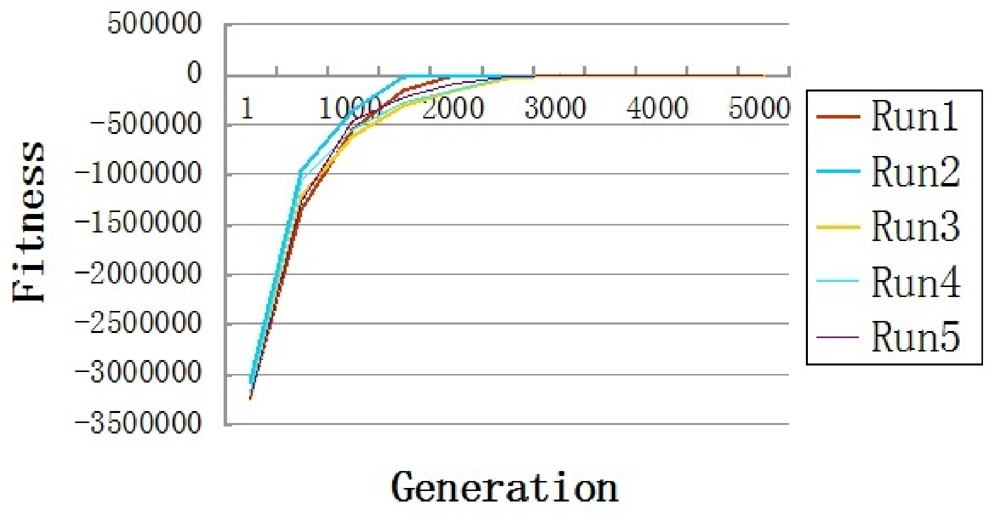


FIGURE 6.11: Fitness value vs. number of generations for five evolutionary runs using experiment O (benchmark HARD3).

A summary of outcomes from these experiments is given as follows:

- In the case of bin packing experiments using benchmark HARD0, it has been found that using two electrodes for outputs and ten for configuration inputs performed the best of all combinations of electrodes.
- In the case of bin packing experiments using benchmark HARD0, it has been found that the performance of using mutation on only one gene in the genotype was better than the performance of using mutation on two genes.

- No evolution happened using an electrode array containing no material, where the output buffers were always full of zeroes.

- No evolution was possible with a material having 0% single-walled carbon nanotubes (only PMMA), where the output buffers were always full of zeroes, which shows that single-walled carbon nanotubes are required in the material mixture for computation.

- Investigations were performed using material sample 8 and material sample 9 (with 0.02% and 0.01% single-walled carbon nanotubes in PMMA respectively). These investigations were carried out for up to 1000 generations for one run with dataset N1C1W1_A (2 electrodes as outputs and 10 as configuration inputs). A single mutation was used to create a child in these investigations. It was found that no evolution took place with 0.01% single-walled carbon nanotubes (material sample 9) and the values of output buffers were always zero. When an experiment was started with material sample 8 (i.e. with 0.02% single-walled carbon nanotubes), evolution happened with mixtures of 0 and 1 in the output buffers.

6.4 When the Split Genotype Technique is Applicable

The split genotype technique has been used previously in two computational problems (function optimisation and bin packing) having many outputs. However, this technique cannot be applied to all computational problems with many outputs. The problems must have some characteristics or properties that need to be suitable for applying the split genotype technique. The split genotype technique can be used in following situations:

- When there is no input, outputs obtained from one chromosome should be independent from outputs of another chromosome. The configuration inputs of one chromosome are independent from configuration inputs of another chromosome, and the configuration inputs of each chromosome control the outputs of that chromosome only. Function optimisation problem and bin packing problem fall in this category.

- When there are one or more inputs, outputs obtained from one chromosome using some inputs should be independent from inputs and outputs of another chromosome and the configuration inputs of one chromosome are independent from configuration inputs of another chromosome. The inputs and configuration inputs of each chromosome control the outputs of that chromosome only.

6.5 Summary

In this chapter, two computational problems, i.e. function optimisation and bin packing problems have been investigated using evolution-in-materio. This is the first time it has been shown that such an approach can be used to solve well-known benchmark function optimisation and bin packing problems.

Function optimisation problems were used to compare the effectiveness of evolution-in-materio against the well-known search technique named Cartesian genetic programming. These problems were also used to compare the performances of two hardware platforms. In function optimisation problems, one electrode was used as an output and the remaining electrodes were used as configuration inputs. There can be many other ways of using the electrodes. Two electrodes could have been read and evolved configuration inputs could have been applied to the remaining electrodes and other choices could be possible. Examining other choices remains for future work. However, in the case of bin packing problem, various numbers of outputs and configuration inputs were investigated using one of the hard bin packing benchmarks. Two different mutation rates were also investigated.

The bin packing problem was also applied to see whether single-walled carbon nanotubes are required in the material mixture for computation or not and the test result showed that without single-walled carbon nanotubes, no evolution can be possible, this is the same result obtained from the tone discriminator experiment (Section 5.3) with a material having no single-walled carbon nanotubes (only polymer).

The next chapter describes experiments that deal with applying evolution-in-materio to robot control to obtain desired behaviours. Both simulated and real robots have been used in these experiments.

Chapter 7

The Evolution-In-Materio Controlled Robot

Contents

Controlling an obstacle avoiding robot is a well-known problem, which has been tried and solved by many researchers using hardware evolution, evolutionary algorithms and even using evolution-in-materio with an LCD by Harding and Miller [Harding and Miller (2005)]. The robot controlling experiments, that are described in this chapter, had a total number of inputs, outputs and configuration inputs of not more than 16, so a split genotype technique was not needed. Often the task for the robot is to travel around a closed environment avoiding the obstacles and walls and to cover as much floor space as possible within a limited number of time steps. Here this task has been modified or extended, where the robot has to cope with faults and environmental changes and to reach a specific location in a map from a specific starting position. The control system is able to use the distance sensors on the robot and use this information in the control of the motion of the robot using motors. Both the simulated Khepera robot and a real Pi-Swarm robot have been used in these experiments. These experiments were performed using material sample 1 (Table 4.3) and Mecobo 3.0. The Khepera robotic platform has eight short range infra red sensors and two motors. The Pi-Swarm robot has also eight short range infra red sensors and two motors like Khepera robot. The positions of the sensors and motors of the Pi-Swarm robot are also exactly the same as the Khepera robot.

7.1 The Khepera Simulator

A Khepera robot simulator (version 2.0), that has been adapted in this research, was written by Marcin Pilat[1]. Pilat rewrote a Unix-based Khepera written by Olivier Michel [Michel (1996)]. The simulated robot has a diameter of 55 nominal units, and obstacles or walls are made from small bricks having width and height 20 units. The map is 1000 X 1000 $unit^2$.

The Khepera simulated robot together with the placement of sensors (S0-S7) and motors (M1-M2), which has been used in the experiments, is illustrated in Figure 7.1. The sensor distance value is calculated as a function of the presence (or the absence) of obstacles[1]. The sensor distance value has a range [0, 1023], where 0 means no object is found and 1023 means an object is in the nearest position. Random noise of $\pm10\%$ is added to the amplitude of the distance value of a sensor. In experiments, the distance values of sensors have been used as inputs to the material. The robot moves according to the speeds (in a range [-10, 10]) of the motors. Random noise of $\pm10\%$ is also added to the amplitude of the speed of a motor while random noise of $\pm5\%$ is added to the direction resulting from the difference of the speeds of two motors. The motor speeds are decided by the outputs of the experimental material (a mixture of single-walled carbon nanotubes and a polymer).

[1]http://www.pilat.org/khepgpsim/

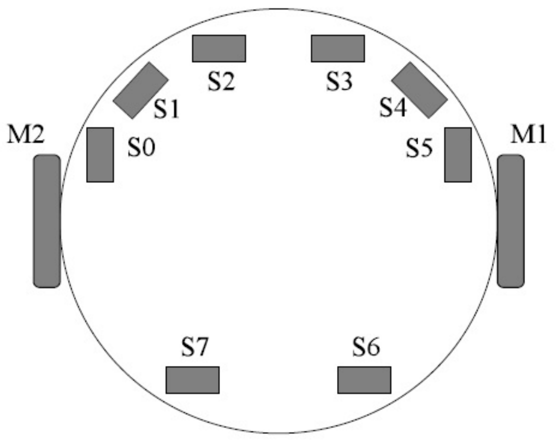


FIGURE 7.1: Schematic view of the Khepera robot with the positions of the IR proximity sensors (S0-S7) and motors (M1, M2).

7.2 The Pi-Swarm Robot

A Pi-Swarm robot [Hilder *et al.* (2014)] has been used in real robot experiments in this thesis. The Pi-Swarm robot has been built in the University of York. The Pi-Swarm robot is designed as part of the Pi-Swarm System, which itself is an extension for the Pololu 3-Pi robot[2] which enables the Pi-Swarm robot to feature as part of a fully autonomous swarm [Hilder (2014)]. An mbed LPC1768 rapid-prototyping microcontroller board is one of the main parts of the system of the robot. This allows code to be easily created on any system with a USB-port and web-browser without the need for any dedicated driver or software. The 3-Pi robot, which is manufactured by Pololu[2], is a circular mobile platform featuring five IR reflectance sensors, two micro metal gear motors, an 8x2 character LCD display, three user push buttons and a buzzer. It is powered by four AAA batteries. The actuators and peripherals are all connected to a programmable ATmega328 microcontroller, which features 2KB of RAM, 32KB of flash program memory and 1KB of persistent EEPROM memory and operates at a 20 MHz clock-speed. The platform can be programmed using the GNU C/C++ compiler. The Atmel Studio can be used as a development environment. Generic firmware is used with the Pi-Swarm system in typical use. This handles messages sent over a serial-bus from the mbed board in order to operate all core functions of the robot.

The mbed LPC1768 is a small PCB, which is designed to allow rapid prototyping for general microcontroller applications. It is based around the NXP1768, a 32-bit ARM

[2]http://www.pololu.com

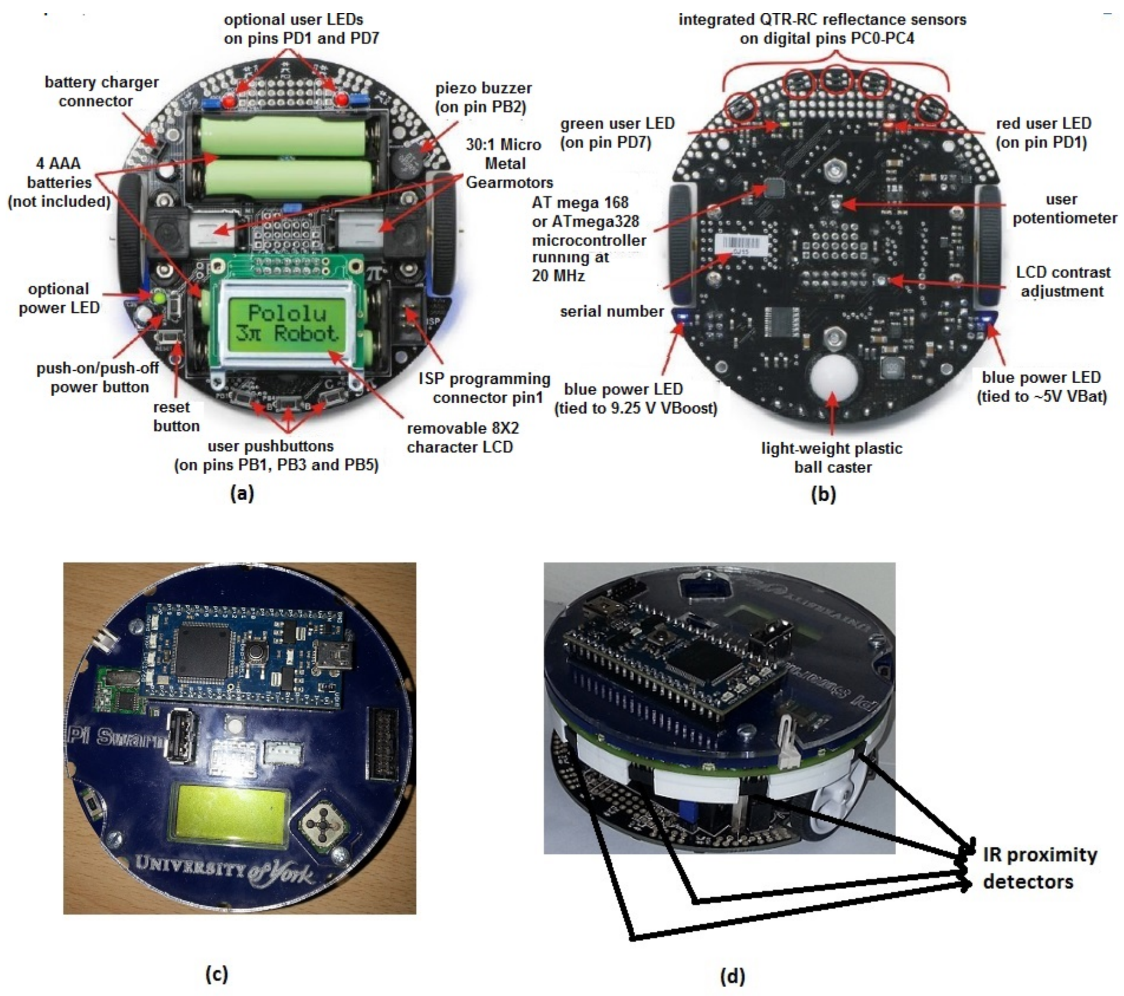

optional user LEDs on pins PD1 and PD7
battery charger connector
piezo buzzer (on pin PB2)
4 AAA batteries (not included)
30:1 Micro Metal Gearmotors
optional power LED
Pololu 3π Robot
push-on/push-off power button
reset button
user pushbuttons (on pins PB1, PB3 and PB5)
ISP programming connector pin1
removable 8X2 character LCD
(a)
integrated QTR-RC reflectance sensors on digital pins PC0-PC4
green user LED (on pin PD7)
red user LED (on pin PD1)
AT mega 168 or ATmega328 microcontroller running at 20 MHz
user potentiometer
LCD contrast adjustment
serial number
blue power LED (tied to 9.25 V VBoost)
blue power LED (tied to ~5V VBat)
light-weight plastic ball caster
(b)
(c)
IR proximity detectors
(d)

FIGURE 7.2: The Pi-Swarm robot (a) Top view of Pololu 3-Pi robot [Hilder (2014)]. (b) Bottom view of Pololu 3-Pi robot [Hilder (2014)]. (c) Top view of Pi-Swarm robot after adding the top cover and mbed. (d) Side view of Pi-Swarm robot.

Cortex-M3 microcontroller that includes 32KB RAM and at least 512KB FLASH memory (on newer boards, it is 1MB) and operates with a 96 MHz clock. The FLASH memory appears as a USB-FLASH drive when the mbed is connected to a computer. The mbed microcontroller has an online compiler[3]. This compiler provides the tool chain and libraries to create C++ programs that can be compiled and installed onto the mbed. After reset, the most recent binary file, which has been saved onto the FLASH memory, is loaded by the bootstrap loader on the mbed board.

The Pi-Swarm extension board connects to the 3-Pi base using the 14-pin peripheral connector. There are two additional 2-pin connectors that allow the duplication of the recharging pins and the reset switch. The top of the board contains a socket to attach the mbed, a number of actuators and sensors. On the underside of the board, eight

[3]at http://www.mbed.org

IR optocouplers are arranged around the edge. These act as proximity detectors and are spaced with sensors at $\pm15^o$, $\pm45^o$, $\pm90^o$ and $\pm144^o$. The sensors and the two motors are organised in the same way as the Khepera robot (Figure 7.1). The robot's built-in library has a function that can calculate distances between objects and the 8 IR proximity detectors. The distance value has a range [0.0, 100.0], where 100.0 means no object is found and 0 means an object is in the nearest position. The built-in library has a function that accepts motor speed values to drive two motors. The robot moves according to the speed (in a range [-1.0, 1.0]) of the motor.

A socket is present at the front, which allows the connection of a separate ultrasonic range detector. Ten RGB LEDs are arranged in a ring around the edge of the board. An additional high-power RGB LED (facing up) is also connected in the middle of the board. Other than these IR proximity detectors and LEDs, a MEMS 3-axis accelerometer, a MEMS yaw gyroscope, a MEMS 3-axis magnetometer, a 433 MHz RF transceiver, a 64 kilobit EEPROM, a digital temperature and ambient light sensor are connected with the robot. A set of 5-DIL switches is also connected with the robot, which is used to set ID of the robot within the swarm. A 5-way directional switch is used to trigger interrupt and control the robot. The Pi-Swarm robot is shown in Figure 7.2.

7.3 Methodology

Ten sets (A-J) of experiments were performed with the simulated robot. The experimental settings of all sets of simulated robot experiments are described in Table 7.1.

In experiments A-I, the maps shown in Figure 7.3 were used. The robot's starting positions in experiments varied. In maps (a) and (c) shown in Figure 7.3, the starting position was the centre of the map and for map (b), the starting position of the robot was the upper left corner of the map. However, in the case of experiment C, the starting position of the robot was chosen randomly.

The complexity of electronic circuits and hardware is increasing day by day. As complexity in hardware is increasing, the chances of error and faults are also increasing. Faults can cause serious problems and can lead to damage. So, fault tolerance has become an important characteristic for an electronic circuit. Tyrrell et al. performed robot controlling experiments, where a continuous evolutionary process was used to help

Table 7.1: The experimental settings of all sets of simulated robot experiments. The 'Task' column shows the task (WA=exploring areas of maps by obstacle avoidance behaviour, FT=fault tolerance behaviour, IE=incremental evolution, MS=maze solving task) of the robot. The 'Map' column shows the map (the reference to the figure number) where the robot was allowed to move. The 'St. pos' column shows the starting position (CN=centre, UL=upper left corner, R=random, M= marked in the map) of the robot in the map. The 'No. of gen.' and 'No. of run' columns show the number of generations and number of runs respectively. The 'No. of el.', 'No. of in.' and 'No. of con.' columns show the total number of electrodes, the number of inputs and number of configuration inputs respectively. In all experiments, 2 electrodes were used for outputs. The 'In. map.' and 'Out. map.' columns show the input mapping (C=mark-space ratio, F=frequency) and output mapping (PO=percentage of ones, TG=average transition gap) respectively. The last column shows the input-output timing (measured in milliseconds) for the experiments. In all sets of experiments, a 25 KHz output sampling frequency was used.

Set	Task	Map	St. pos.	No. of gen.	No. of run	No. of el.	No. of. in.	No. of con.	In. map.	Out. map.	Time
A	WA	7.3 (a)	CN	100	10	12	6	4	C	PO	20
B	WA	7.3 (b)	UL	100	10	12	6	4	C	PO	20
C	WA	7.3 (b)	R	100	10	12	6	4	C	PO	20
D	WA	7.3 (b)	UL	100	10	12	6	4	C	TG	20
E	WA	7.3 (a)	CN	100	10	12	6	4	F	TG	20
F	WA	7.3 (b)	UL	100	10	16	8	6	C	PO	25
G	FT	7.3 (a)	CN	200	5	16	8	6	C	PO	32
H	WA	7.3 (c)	CN	300	5	12	6	4	C	PO	20
I	IE	7.3 (c)	CN	300	5	12	6	4	C	PO	20
J	MS	7.4 (a)-(f)	M	300	5	16	8	6	C	PO	25

cope with environment changes or faults [Tyrrell *et al.* (2004)]. This was described in Section 3.1. Experiments G and I were performed to evolve robot controllers that could cope with faults and environment changes. In experiment G, each of five evolutionary runs of 100 generations was carried out before a fault was introduced in the robot by switching off one sensor (S1). This was done by setting the corresponding mark-space ratio to zero. Then each evolutionary algorithm was run for another 100 generations. In the incremental evolution experiments (set I), the number of generations was 300. Other than the 4 side walls, there are 7 obstacles in the middle of the map, which are distributed all over the map. These 7 obstacles were not placed at the same time. They were placed one by one in intervals of 30 generations, starting with the 20^{th} generation. No obstacle was added during the last 100 generations.

The tenth set (J) of experiments was concerned with maze solving task and six different maps were used in these experiments. The fitness of the robot controller in experiment J was gathered in three stages using three maze maps. The maps in the sequence

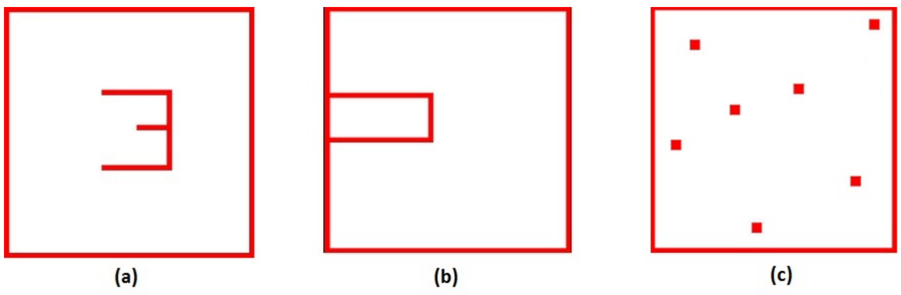


FIGURE 7.3: Task environments used in simulated robot experiments A-I. In (a)-(c), the obstacles or the walls are shown in red and the white area of the map is the area where the robot is allowed to move.

increased in complexity. The complexity has been judged using the number of turns required for a robot to move from its starting position to the goal. The simplest map has the least number of turns and the hardest map has the highest number of turns. The evolutionary run started from the simplest map (Figure 7.4 (a)). After an individual in the population was found, which could successfully solve the maze, the population was immediately evaluated on the next more complex maze map (Figure 7.4 (b)) for further evolution. Once a population member could solve the second maze, the full population was evaluated on the third maze map (Figure 7.4 (c)). Once the period of evolution that used the third maze map was completed, the final population of the robot controller was tested on three previously unseen other maze maps (Figure 7.4 (d), (e) and (f)) to test the generality of the evolved controller.

In all sets of experiments except the F, G and J, only 12 electrodes from the 16 electrodes were used (the middle 6 electrodes from each side of one sample). In these experiments, the number of inputs was 6. These inputs were provided by sensors S0, S2, S3, S5, S6 and S7 (Figure 7.1). In F, G and J, all 16 electrodes were used, where 8 electrodes (all 8 sensors) were used as inputs.

For all sets of experiments, each chromosome defined which electrodes were outputs, inputs (received square waves) or received the configuration inputs (square waves or static voltages).

The input-output timing for experiments F, G and J was higher due to using a greater number of electrodes. Sampling over longer times is necessary as the scheduling in Mecobo is serial. This means that several sequences of actions (i.e. sending inputs, configuration inputs, reading outputs) do not take place at the same instant. Mecobo

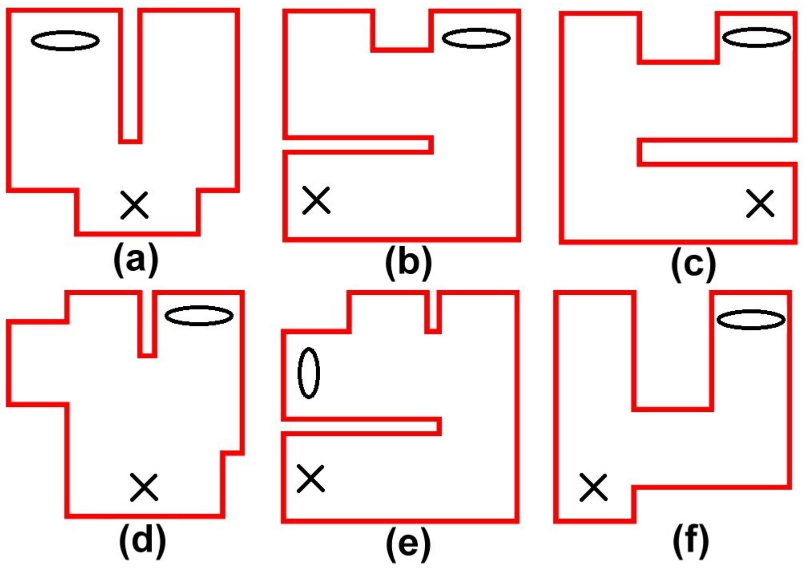


FIGURE 7.4: Task environments used in maze solving experiment (J). In (a)-(f), the obstacles or the walls are shown in red and the white area of the map is the area where the robot is allowed to move. The starting position of the robot is marked with a cross and the goal is marked with an oval.

maintains a schedule. Thus, it takes some time for Mecobo to circulate signal to each electrode.

In experiments A-F, the number of runs was 10 and in experiments G-J, the number of runs was 5. The experiments G-J were performed over a lower number of runs as these experiments were performed over a higher number of generations. The experiments G-J required more generations to obtain better results. This was due to the fact that the experiments G and I were designed to investigate fault tolerance and incremental evolution (environmental changes) respectively, both the experiments H and I were performed on a more complex map and experiment J was performed to solve mazes.

7.4 Genotype Representation

In experiments A-E, H and I, each chromosome used $n_e = 12$ electrodes at a time. In experiments F, G and J, each chromosome used $n_e = 16$ electrodes. The values that genes could take are shown in Table 7.2, where i takes values 0, 1, . . . 11 for experiments A-E, H, I and values 0, 1, . . . 15 for experiments F, G, J. The description of the genotype for the experiments is shown in Table 7.3.

Table 7.2: Description of the genes for robot controlling experiments.

Gene symbol	**Signal applied to, or read from the i^{th} electrode**	**Allowed values**
p_i	Which electrode is used	0, 1, 2 . . . 11 (for sets: A-E, H, I) 0, 1, 2 . . . 15 (for sets: F, G, J)
s_i	Type	0 (static) or 1(square wave)
a_i	Amplitude	0 , 1
f_i	Frequency	500 ,501 . . . 10K
c_i	Mark-space ratio	0, 1, . . . 100

Table 7.3: Description of the genotype for robot controlling experiments. The 'Exp.' column shows the set(s) of experiments. The 'No. of gen. in each elec.' column shows the number of genes associated with each electrode. The 'Gen. ass. with each elec.' column shows the genes that are associated with each electrode. The 'Total no. of genes' column shows the total number of genes in each genotype. The 'Genotype representation' column shows the representation of a genotype. The 'Genes related to inputs' column shows the gene values of a genotype, which are related to inputs. The 'Genes related to outputs' column shows the gene values of a genotype, which are related to outputs.

Exp.	No. of gen. in each elec.	Gen. ass. with each elec.	Total no. of genes	Genotype representation	Genes related to inputs	Genes related to outputs
Sets A-E, H, I	5	p_i, s_i, a_i, f_i, c_i	12X5 =60	$p_0s_0a_0f_0c_0 \ldots p_{11}s_{11}a_{11}f_{11}c_{11}$	First 30 genes: $p_0s_0a_0f_0c_0$ $\ldots$ $p_5s_5a_5f_5c_5$	Last 10 genes: $p_{10}s_{10}a_{10}f_{10}c_{10}$ $\ldots$ $p_{11}s_{11}a_{11}f_{11}c_{11}$
Sets F, G, J	5	p_i, s_i, a_i, f_i, c_i	16X5 =80	$p_0s_0a_0f_0c_0 \ldots p_{15}s_{15}a_{15}f_{15}c_{15}$	First 40 genes: $p_0s_0a_0f_0c_0$ $\ldots$ $p_7s_7a_7f_7c_7$	Last 10 genes: $p_{14}s_{14}a_{14}f_{14}c_{14}$ $\ldots$ $p_{15}s_{15}a_{15}f_{15}c_{15}$

In all experiments, mutated children were created from a parent genotype by mutating a single gene (i.e. one gene of 60 in the case of sets A-E, H and I and one gene in 80 in the case of sets F, G and J). In these input and output genes, only the p_i (here the values of i are 0-5 and 10-11 for solutions with 12 electrodes and values of i are 0-7 and 14-15 for solutions with 16 electrodes) has any effect, others do not have any effect. The gene p_i decides which electrode will be used for the input and output of the device. Thus, mutations in this gene can choose a different electrode to be used as an input or output.

The examples of electrode arrangements of robot controlling experiments are shown in Figures 7.5.

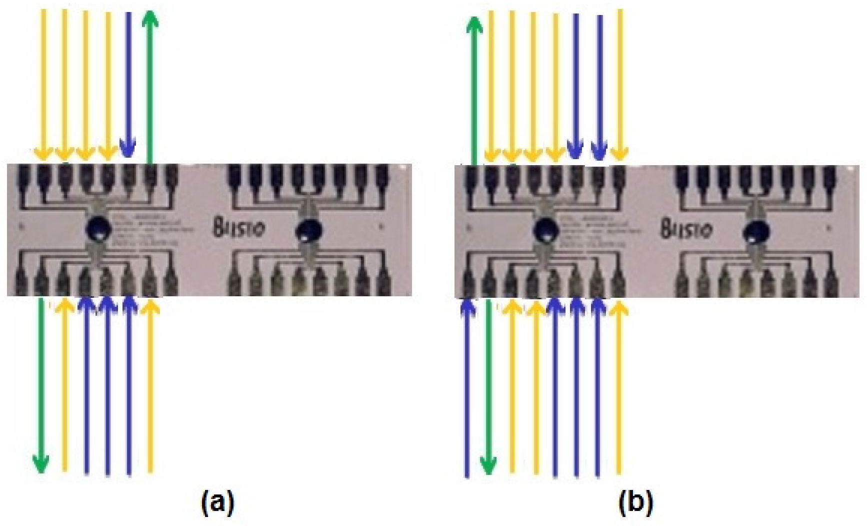


FIGURE 7.5: Two examples of electrode arrangements used in robot controlling experiments. (a) This type of electrode arrangement is used in sets A-E, H and I. Green arrows are used to indicate reading outputs from output electrodes, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 4 electrodes. (b) This type of electrode arrangement is used in sets F, G and J. Green arrows are used to indicate reading outputs from output electrodes, yellow arrows are used to show inputs being sent to input electrodes and blue arrows are used to show configuration inputs being sent to 6 electrodes.

7.5 Input Mapping

In the experiments A-D, F-J, each of the inputs to the electrode array was a square wave with a fixed mark-space ratio. The mark-space ratio (the examples of mark-space ratio are shown in Figure 4.2) was determined by a linear mapping of the distance value of the sensor. The examples of mark-space ratio mapping for the experiments A-D, F-J are shown in Figure 7.6. In experiment E, each of the inputs to the electrode array was a square wave with a fixed frequency. The frequency was determined by a linear mapping of the distance value of the sensor. The input mappings for these experiments are described as follows:

Denote the distance value of the i^{th} sensor by I_i, where i takes integer values in a range 0-5 (corresponding to six sensors) in experiments A-D, H, I and 0-7 (corresponding to 8 sensors) in experiments F, G, J. Denote the maximum distance value and minimum distance value of any sensor by $I_{i_{max}}$ and $I_{i_{min}}$ respectively. Then the linear input mappings of the robot controlling experiments are shown in Table 7.4.

TABLE 7.4: Input mappings for robot controlling experiments

Exp.	Equation	Variables	Other parameters of input signals
Sets A-D, F-J	$$M_i = a_i I_i + b_i \quad (7.1)$$ Here, $$a_i = \frac{(M_{max} - M_{min})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.2)$$ $$b_i = \frac{(M_{min} I_{i_{max}} - M_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.3)$$	I_i is mapped to a mark-space ratio M_i, where the maximum allowed mark-space ratio is M_{max} and the minimum allowed mark-space ratio is M_{min}. Here M_{max} =100%, M_{min}=0%, $I_{i_{min}}$=0 and $I_{i_{max}}$=1023	Frequency =5KHz and amplitude=1
Set E	$$F_i = a_i I_i + b_i \quad (7.4)$$ Here, $$a_i = \frac{(F_{max} - F_{min})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.5)$$ $$b_i = \frac{(F_{min} I_{i_{max}} - F_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.6)$$	I_i is mapped to a square wave frequency F_i, where the maximum allowed frequency is F_{max} and the minimum allowed frequency is F_{min}. Here F_{max}=10KHz, F_{min}=500Hz, $I_{i_{min}}$=0 and $I_{i_{max}}$=1023	Mark-space ratio =50% and amplitude=1
Real Robot Exp.	$$M_i = a_i I_i + b_i \quad (7.7)$$ Here, $$a_i = \frac{(M_{max} - M_{min})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.8)$$ $$b_i = \frac{(M_{min} I_{i_{max}} - M_{max} I_{i_{min}})}{(I_{i_{max}} - I_{i_{min}})} \quad (7.9)$$	I_i is mapped to a mark-space ratio M_i, where the maximum allowed mark-space ratio is M_{max} and the minimum allowed mark-space ratio is M_{min}. Here M_{max} =100%, M_{min}=0%, $I_{i_{min}}$=0 and $I_{i_{max}}$=100	Frequency =5KHz and amplitude=1

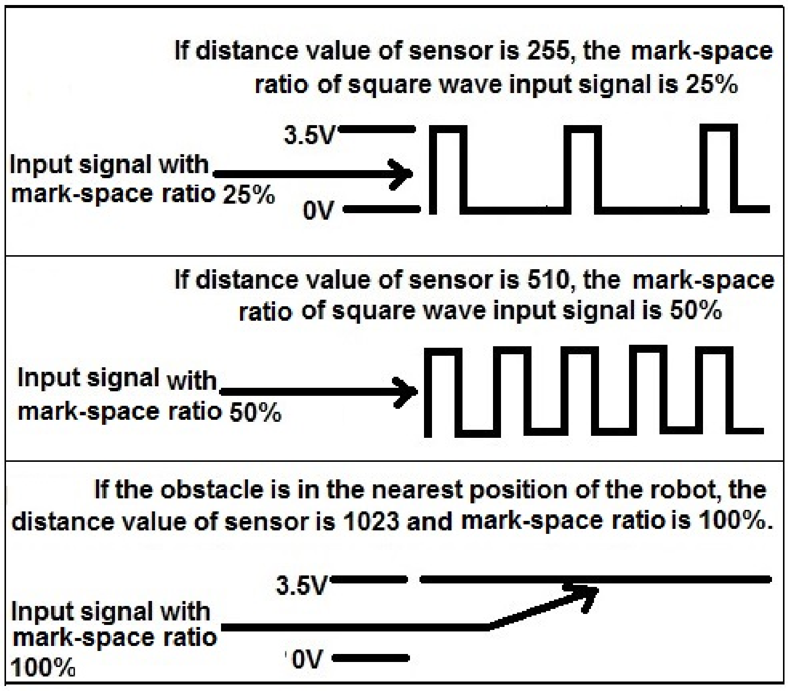

If distance value of sensor is 255, the mark-space ratio of square wave input signal is 25%
3.5V
Input signal with mark-space ratio 25%
0V
If distance value of sensor is 510, the mark-space ratio of square wave input signal is 50%
Input signal with mark-space ratio 50%
If the obstacle is in the nearest position of the robot, the distance value of sensor is 1023 and mark-space ratio is 100%.
3.5V
Input signal with mark-space ratio 100%
0V

FIGURE 7.6: Three examples of mark-space ratio mappings for the experiments A-D, F-J. (a) If the distance value of the sensor is 255, the mark-space ratio of the input signal is 25%.(b) If the distance value of the sensor is 510, the mark-space ratio of the input signal is 50%. (c) If the obstacle is in the nearest position of the robot, the distance value of the sensor is 1023 and the mark-space ratio of the input signal is 100%.

It should be noted that the distance value of IR detector was subtracted from value 100.0 to maintain the similarity with the input mapping used in simulated robot experiments. In simulated robot experiments, the lowest distance value (according to the range [0, 1023]) of IR sensor was used when there was no object and the highest value was used when an object was right next to the IR sensor. In the Pi-Swarm experiments, a value 0 was used when no object was found and 100.0 when an object was in the nearest position.

7.6 Output Mapping

The output was determined by examining the output buffers containing samples taken from the output electrodes. Since the Mecobo 3.0 platform can only recognise binary values, the output buffers contain bitstrings. In all experiments except the D and E, the fraction of ones was used to obtain output values. This mapping was used as it seems more closely related to mark-space ratio. In the experiments D and E, a different output

Table 7.5: Output mappings of robot controlling experiments. The ‘Exp.’ column shows the set of experiments. The ‘Equation’ column shows the equation used for calculating motor speed value. The ‘Variable’ column defines the variables used in the output mapping equation.

Exp.	Equation	Variables
Sets A-C, F-J	$o_i = M_{min} + (M_{max} - M_{min}) num_i / buf_i$ (7.10)	num_i is the number of ones in the i^{th} output buffer. Here, M_{max}=10 and M_{min}=-10.
Sets D,E	$o_i = -10 + 20 avg_i / (buf_i - 2)$ (7.11)	avg_i is the average transition gap in the i^{th} output buffer. Here, M_{max}=10 and M_{min}=-10.
Real Robot Exp.	$o_i = M_{min} + (M_{max} - M_{min}) num_i / buf_i$ (7.12)	num_i is the number of ones in the i^{th} output buffer. Here, M_{max}=1.0 and M_{min}=-1.0

mapping was used corresponding to the *transitions* from 0 to 1 in the output buffers. This was done so that the relative merits of the two mappings could be ascertained. In the transition-based mapping, the transitions in each output buffer were recorded and the gaps between consecutive transitions were measured and an average calculated. The thinking behind using a transition-based mapping is that it may be useful as it is mark-space ratio and frequency related. An example of average transition gap calculation for an output electrode is shown in Figure 5.2. The output mappings of these experiments are described as follows:

Let, buf_i is the total number of samples of the i^{th} output buffer, where i takes values 0, 1 (corresponding to two motors). M_{max} is the maximum motor speed and M_{min} is the minimum motor speed. Then the equations used for calculating the linearly mapped i^{th} motor speed of the robot, o_i for these experiments are shown in Table 7.5.

In experiments D and E, the highest average transition gap is $(buf_i - 2)$ when the first transition happens at index 0 (output value 0 at index 0 and value 1 at index 1) in the output buffer and the last transition happens at index $(buf_i - 2)$ (output value 0 at index $(buf_i - 2)$ and value 1 at index $(buf_i - 1)$) and no other transition happens in between.

7.7 Fitness Calculation

To calculate a fitness of an evolved robot controller, each individual of the population is executed for a number of time steps. Experiments A-I used 5000 time steps and experiment J (maze solving) used 10,000 time steps. In most cases, if the robot collides with an obstacle, it is stopped immediately, resulting in a lower fitness value. However, in the case of maze solving experiment, the robot, which moves near to the goal, is allowed to run after a collision for up to 1000 collisions. If a robot rotates in the same place for 1000 consecutive moves, it is also stopped. The latter is assessed using the x and y coordinates of robot's position over all time steps in the past. However, the previous (immediate) 50 positions of the robot are not used to prevent a slowly moving robot being penalized. If the differences between the old and new x and y coordinates are ≤ 30 units (approximately half the diameter of the robot), it is assumed that the robot visits the same place as before, otherwise it is assumed that the robot is exploring a new area of the map. If the robot rotates in the same place for 1000 consecutive moves, it is stopped immediately and its overall fitness is assessed. However, in the case of experiment J (maze solving), a robot is not stopped when it rotates in the same place for 1000 consecutive moves if there is no obstacle between robot's position and the goal. If a robot has not been stopped early and it is exploring a new area of the map, the distance between the previous position and the new position is added to the fitness score. The distance is calculated using Euclidean distance with x and y coordinates of the previous and new positions. The better individuals are decided by higher fitness values. In the case of maze solving experiment (J), robots are rewarded for reaching points nearer to the goal. This is done by measuring the Euclidean distance between the position of the robot and the goal. This Euclidean distance is subtracted from the value $1415 \approx 1414.214$ so that the value becomes higher as the robot reaches closer to its goal. It should be noted here that the largest distance between any two points on the map is 1414.214 corresponding to the points (0, 0) and (1000, 1000). The obtained value is multiplied by a constant and then added to the fitness value. The constant value is chosen to be 10. It has been found that if 10 is multiplied by the Euclidean distance and added to the fitness value, the obtained fitness value becomes high enough to reward the robot to survive in next generation, i.e. the obtained fitness value becomes higher than any other fitness values of the robots that have explored all the areas of the map without collision and this was done by observing the fitness values of experiments A-I.

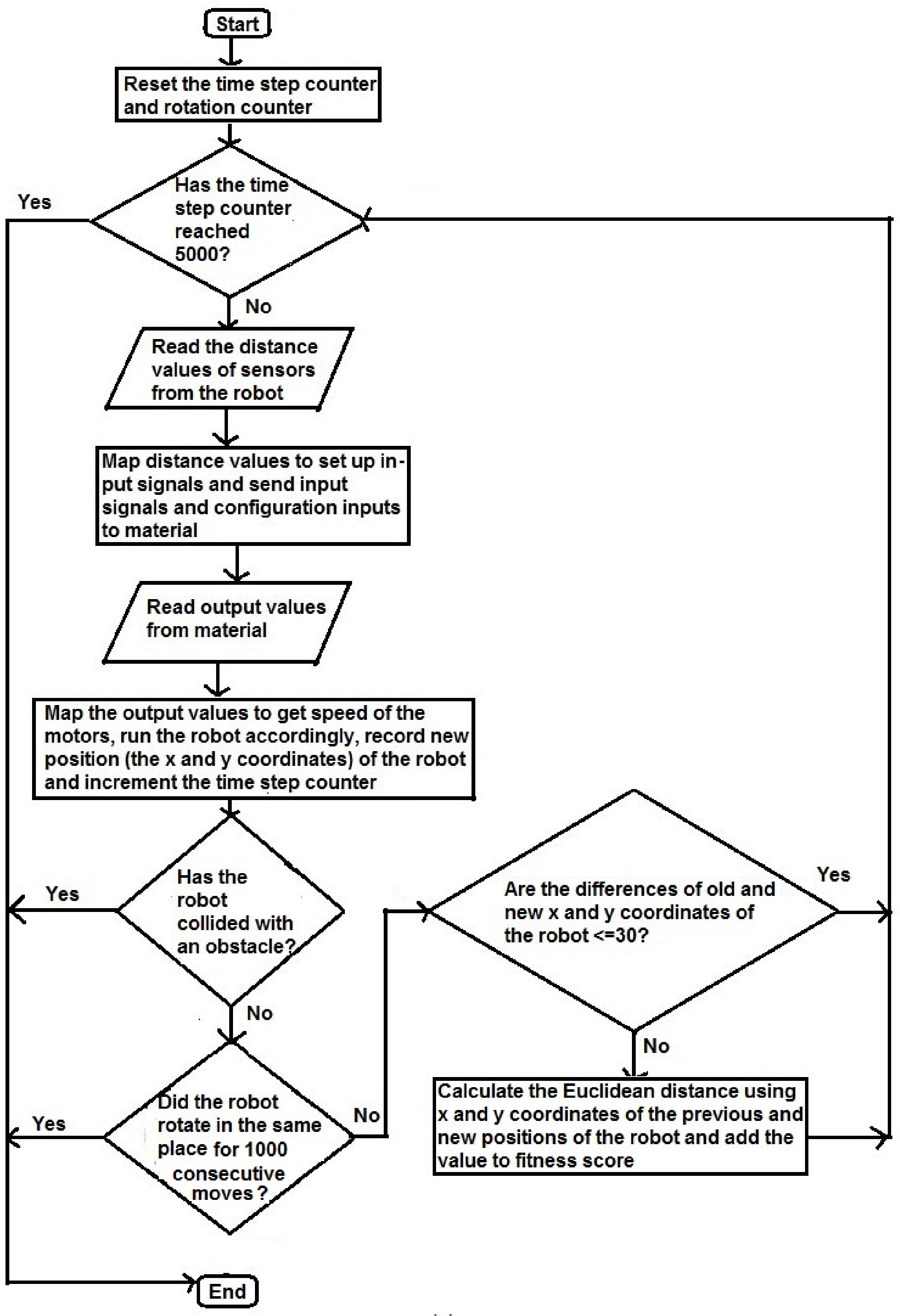

Start
Reset the time step counter and rotation counter
Has the time step counter reached 5000?
Yes
No
Read the distance values of sensors from the robot
Map distance values to set up in-put signals and send input signals and configuration inputs to material
Read output values from material
Map the output values to get speed of the motors, run the robot accordingly, record new position (the x and y coordinates) of the robot and increment the time step counter
Has the robot collided with an obstacle?
Yes
No
Did the robot rotate in the same place for 1000 consecutive moves ?
Yes
No
Are the differences of old and new x and y coordinates of the robot <=30?
Yes
No
Calculate the Euclidean distance using x and y coordinates of the previous and new positions of the robot and add the value to fitness score
End

Figure 7.7: The flow chart of fitness calculation for simulated robot experiments A-I.

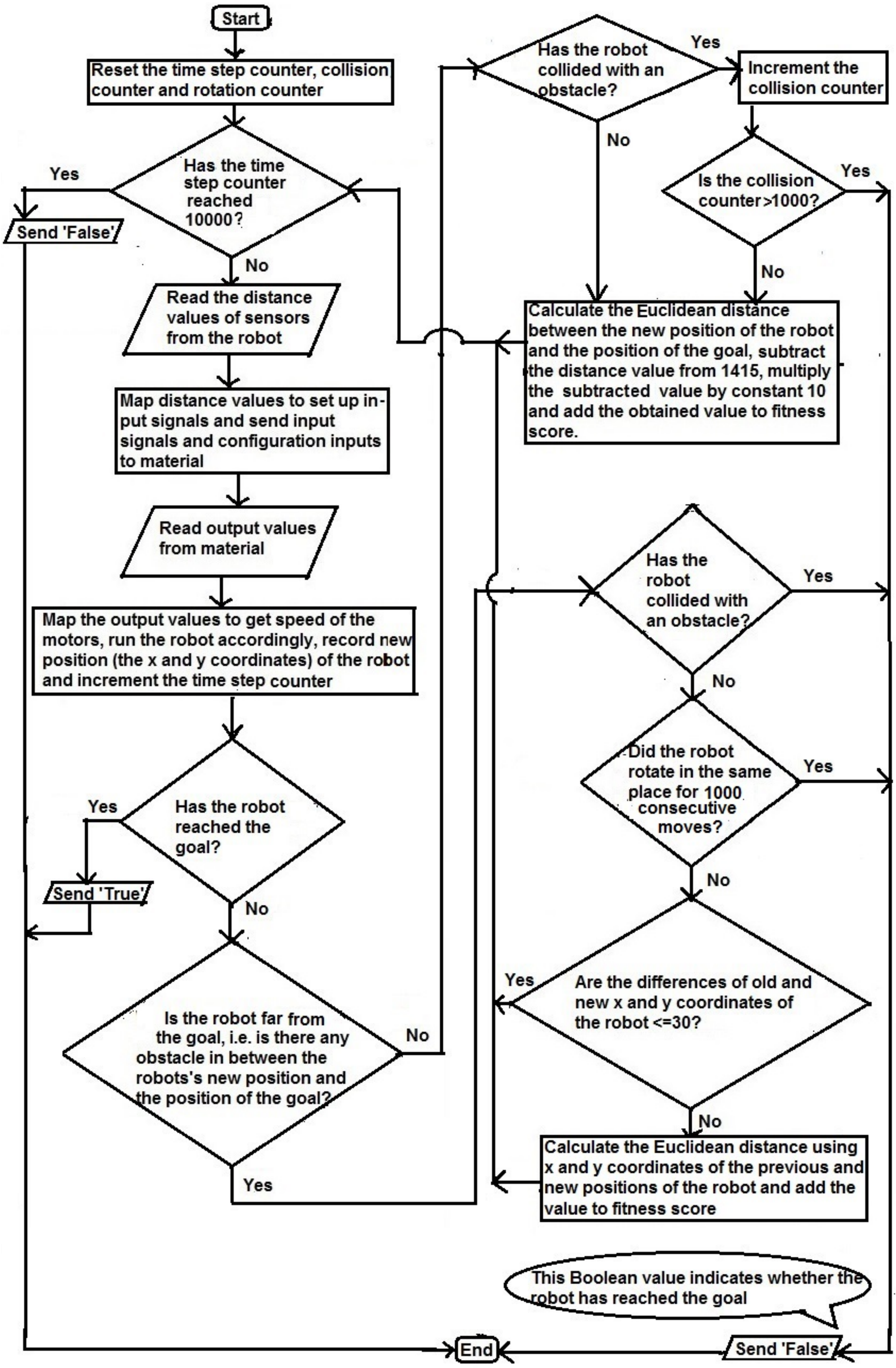

Start
Reset the time step counter, collision counter and rotation counter
Has the time step counter reached 10000?
Yes
Send 'False'
No
Read the distance values of sensors from the robot
Map distance values to set up input signals and send input signals and configuration inputs to material
Read output values from material
Map the output values to get speed of the motors, run the robot accordingly, record new position (the x and y coordinates) of the robot and increment the time step counter
Has the robot reached the goal?
Yes
Send 'True'
No
Is the robot far from the goal, i.e. is there any obstacle in between the robots's new position and the position of the goal?
No
Yes
Has the robot collided with an obstacle?
Yes
No
Increment the collision counter
Is the collision counter>1000?
Yes
No
Calculate the Euclidean distance between the new position of the robot and the position of the goal, subtract the distance value from 1415, multiply the subtracted value by constant 10 and add the obtained value to fitness score.
Has the robot collided with an obstacle?
Yes
No
Did the robot rotate in the same place for 1000 consecutive moves?
Yes
No
Are the differences of old and new x and y coordinates of the robot <=30?
Yes
No
Calculate the Euclidean distance using x and y coordinates of the previous and new positions of the robot and add the value to fitness score
This Boolean value indicates whether the robot has reached the goal
Send 'False'
End

Figure 7.8: The flow chart of fitness calculation for simulated robot experiment J (maze solving).

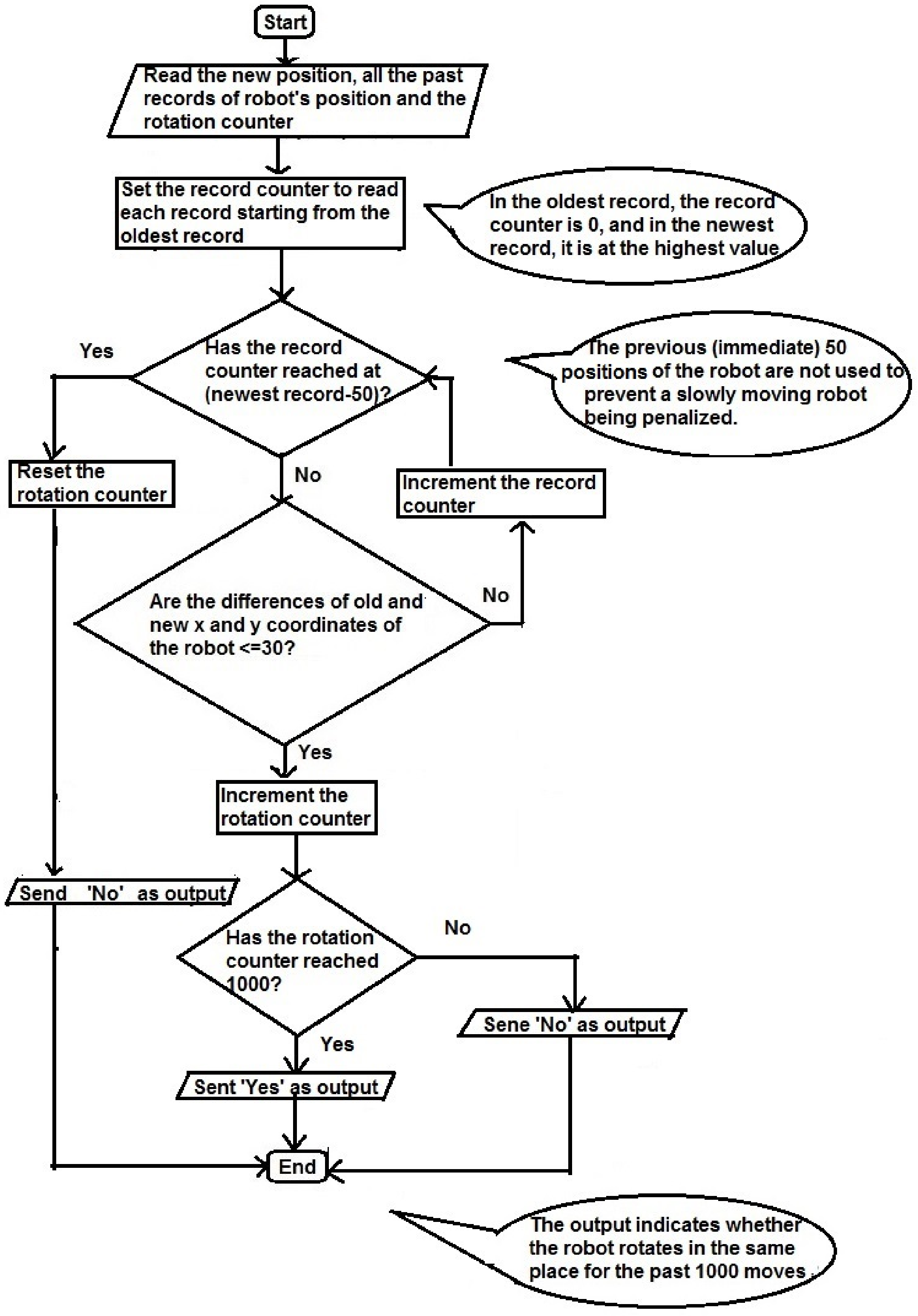

Start
Read the new position, all the past records of robot's position and the rotation counter
Set the record counter to read each record starting from the oldest record
In the oldest record, the record counter is 0, and in the newest record, it is at the highest value
Has the record counter reached at (newest record-50)?
The previous (immediate) 50 positions of the robot are not used to prevent a slowly moving robot being penalized.
Yes
No
Reset the rotation counter
Increment the record counter
Are the differences of old and new x and y coordinates of the robot <=30?
No
Yes
Increment the rotation counter
Send 'No' as output
Has the rotation counter reached 1000?
No
Yes
Sene 'No' as output
Sent 'Yes' as output
End
The output indicates whether the robot rotates in the same place for the past 1000 moves

FIGURE 7.9: The flow chart of the function that decides whether the robot is rotating in the same place for 1000 consecutive moves or not.

Table 7.6: The elapsed time of experiments A-I. The 'No. of gen.' column shows the number of generations of the evolutionary run. The 'Able to exp. the map' column shows the average time taken by those robots that could explore the full map without colliding with obstacles. 'Not applicable' of this column indicates that none of the robots could explore the full map. The 'Unable to exp. the map' column shows the average time taken by those robots that could not explore the full map due to colliding with obstacles or rotating in the same place for 1000 consecutive moves within the given number of generations.

Set	No. of gen.	Able to exp. the map	Unable to exp. the map
A	100	9 hours	More than 5 hours
B	100	More than 14 hours	8 hours
C	100	4 hours	1 hour
D	100	Not applicable	Less than 1 hour
E	100	Not applicable	More than 3 hours
F	100	More than 8 hours	More than 4 hours
G	200	Not applicable	More than 20 hours
H	300	More than 13 hours	More than 10 hours
I	300	Not applicable	30 hours

The flow charts of fitness calculation of simulated robot experiments A-I and J are shown in Figures 7.7 and 7.8 respectively, and Figure 7.9 shows the flow chart of the function that decides whether the robot is rotating in the same place for 1000 consecutive moves or not.

There can be many ways to decide whether the robot reaches close to its goal or not. The Euclidean distance between robot's current position and the goal can be used, but it has a drawback, especially if there is any obstacle between these two positions, in that case, there is no direct path for the robot to reach the goal, thus the robot might need to move a significant amount to reach the goal. So, merely calculating the Euclidean distance between robot's current position and the goal is not always a good measure. In the experiment, this has been taken into account by analysing the positions of all the obstacles. If there is no obstacle between the robot's current position and the goal, the robot is judged to be near to its goal.

In the case of experiments A-I, the elapsed time of an evolutionary run varied according to the length of time that a robot could survive without colliding with an obstacle. Individuals of a population would take longer to run if they did not rotate in the same place or collide with obstacles. The elapsed times of experiments A-I are shown in Table 7.6.

It has been found that the time was higher in experiments involving incremental evolution (experiment I). In the maze experiment (set J), robots were evolved that solved the simplest map in an average of 2 hours, while to evolve robots to solve the second map took a further 6 hours, but a robot was evolved that could solve the hardest map in a further 45 minutes.

7.8 The Simulated Robot Experiments and the Results

For each of the experiments, a (1+4)-evolutionary algorithm was used. In experiments, a child replaced the parent if its fitness was greater than or equal to the parent.

The results of experiments A-I are described in Table 7.7. The paths of the robot exploring the full map without colliding with obstacles are shown in Figure 7.10 using three results from experiments A-I on three different maps (Figure 7.3 (a)-(c)). The movements of the robot that could not explore the full map due to colliding with obstacles are shown in Figure 7.11 using three results from experiments A-I on three different maps (Figure 7.3 (a)-(c)).

After comparing the results of experiments B and C (the experimental settings of B and C were the same, but the starting position of the robot was upper left corner in experiment B and random in experiment C), it can be noted that an upper left corner starting position of the robot was suitable for robots to explore the full map in the case of map (b) (Figure 7.3) as the results of experiment B are better than the results of experiment C. It has been found from the records of robot's movement that in the case of experiment C, only three robots could explore the full map, these started from the upper left corner of the map.

The experimental results of D and E have shown that an output mapping based on average transition gap did not provide good results. It should be noted that experiments D and E used average transition gaps for output mappings and other experiments used percentages of ones.

It has been found that the results of experiment F with 6 configuration inputs were not as good as the results of experiment B with 4 configuration inputs. It should be noted that the experiments B and F used the same experimental settings, but experiment B

Table 7.7: The results of experiments A-I. The 'No. of run' column shows the number of runs. The 'Total number of robots exploring the full map' column shows the total number of robots that explored the full map (with or without colliding with obstacles) and the 'Number of robots exploring the full map without colliding' column shows the number of robots that explored the full map without colliding with obstacles. The 'Min. gen.' column shows the minimum number of generations required to explore the full map without colliding with an obstacle. 'Not applicable' of this column indicates that none of the robots could explore the full map without colliding. The last column shows some additional results.

Set	No. of run	Total number of robots exploring the full map	Number of robots exploring the full map without colliding	Min. gen.	Note
A	10	5	4	21	3 robots could not escape from the middle zone of the map due to collision and 2 robots explored almost half of the map.
B	10	7	3	33	1 robot could explore the 4/5 of the map but was stopped due to rotating in the same place for 1000 consecutive moves.
C	10	3	2	47	The starting positions of the best 3 robots were the upped left corner of the map.
D	10	0	0	Not applicable	The highest moves that the robot could continue without colliding with obstacles was 247.
E	10	0	0	Not applicable	The highest moves that the robot could continue without colliding with obstacles was 91.
F	10	4	2	32	1 robot could explore the 4/5 of the map but was stopped due to rotating in the same place for 1000 consecutive moves.
G	5	1 (before and after the fault injection) and 1 (after the fault injection)	1 (after the fault) injection)	Not applicable	1 robot did not collide with an obstacle but could not explore the full map (before and after the fault injection). In 3 runs, the fitness value that was obtained prior to the fault injection was recovered or even exceeded in an average within 42 generations.
H	5	2	2	198	1 robot explored the 8/9 of the map.
I	5	3	0	Not applicable	2 robots explored the 7/9 of the map.

used 12 electrodes (6 as inputs, 4 as configuration inputs) and experiment F used 16 electrodes (8 as inputs, 6 as configuration inputs).

After the first nine sets of experiments, four generalization experiments were performed. The target was to find the robots that could explore all of the three maps (Figure 7.3) without colliding with obstacles. In these generalization experiments, the successful robots of experiments A, B, C and H were transferred to other two maps of Figure 7.3, where the corresponding robots did not move at the time of the evolution. This means, in the evolutionary run, maps (a) and (c) (Figure 7.3) were used in experiments A and H respectively and map (b) was used in experiments B and C. In the generalisation experiments, robots of experiment A were tested on maps (b) and (c), robots of experiment H were tested on maps (a) and (b), and robots of experiments B and C were tested on maps (a) and (c). The successful robots are the robots which could explore the full map without colliding with obstacles in the evolutionary run. The four generalization experiments are described as follows:

- In total four robots of experiment A were successful. In the case of the first generalization experiment, these four robots were run on the second (Figure 7.3 (b)) and third maps (Figure 7.3 (c)). Of these, only one robot could explore the second map without colliding with an obstacle. The other three robots of experiment A could not explore any of the two maps due to colliding with obstacles.
- In total three robots of experiment B were successful. In the case of the second generalization experiment, these were run on the first (Figure 7.3 (a)) and third (Figure 7.3 (c)) maps. Of these, none of the robots could explore any of the two maps due to colliding with obstacles.
- In total two robots of experiment C were successful. In the third generalization experiment, these were run on the first and third maps. Of these, one robot did not collide with an obstacle when transferred to the first map but could not explore the full map within 5000 time steps. The same robot could not explore the third map due to rotating in the same place for 1000 consecutive moves. The other robot of experiment C collided with an obstacle in the case of both maps.
- In total two robots of experiment H were successful. In the fourth generalization experiment, these were run on the first and second maps. In this experiment, both of the robots collided with obstacles in the first map, and these robots could not explore the second map due to rotating in the same place for 1000 consecutive moves.

As before, in the four generalization experiments, the robots were allowed to move up to 5000 time steps, however they were stopped as soon as they collided with obstacles or if they rotated in the same place for 1000 consecutive moves (Section 7.7). It has been found from the generalization experiments that none of the robots could explore all the three maps (Figure 7.3) due to colliding with obstacles or rotating in the same place for

TABLE 7.8: The results of experiment J (maze solving). The 'Map' column shows the maze (according to Figure 7.4) used in the experiment. The 'Use of the map' column shows the use (training or test) of the map. The 'No. of individuals' column shows the number of individuals solved the maze. The 'No. of runs' column shows the number of runs in which the maze was solved. The 'Average no. of generations' column shows the average number of generations to solve the maze. 'Not applicable' of this column indicates that the result corresponding to this column is not applicable for the maze. The value of the column is not applicable in the case of last three mazes, which were used for testing.

Map	Use of the map	No. of individuals	No. of runs	Average no. of generations
a	Training	At least 1 in each run	5	20.6
b	Training	At least 1 in each run	5	56.8
c	Training	At least 1 in each run	5	6.6
d	Test	7 (2 individuals in the first, second and fifth runs; 1 individual in the third run)	4	Not applicable
e	Test	4 (2 individuals in the second and fourth runs)	2	Not applicable
f	Test	6 (3 individuals in the first run; 2 individuals in the second run; 1 individual in the fifth run)	3	Not applicable

1000 consecutive moves. However, only one robot (one successful robot of experiment A) could explore two maps (Figure 7.3 (a) and (b)). It should be noted that the starting position of the robot in the first, second and fourth generalization experiments was the centre of the map in the case of the first and third maps and upper left corner in the case of the second map, but the starting position was selected randomly in the case of the third generalization experiment.

Experiment J was concerned with maze solving. The results of experiment J are described in Table 7.8. The paths of the robot solving the mazes are shown in Figure 7.12 using the results of experiment J on six mazes (Figure 7.4 (a)-(f)).

From these results it can be concluded that the maze (d) was solved by more robots than the other two mazes ((e) and (f)), this is due to the fact that it is the simplest maze. Although the maze (f) was the hardest, it was still solved by 6 robots, which was more than the number of robots that solved the maze (e).

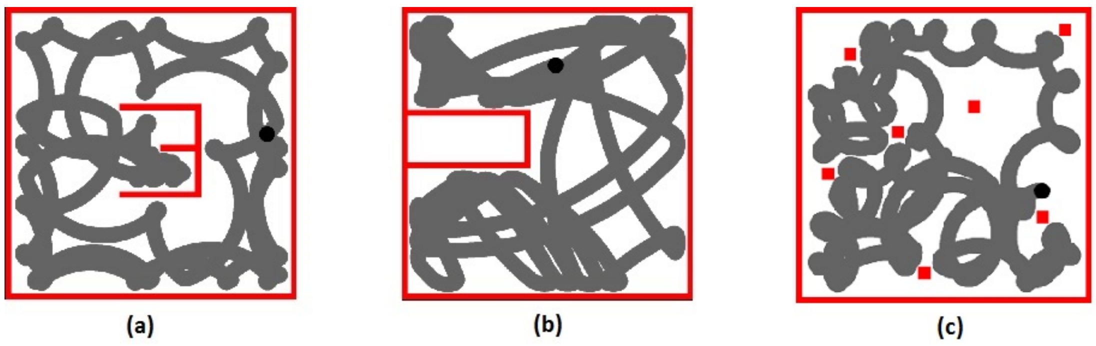


FIGURE 7.10: Paths of robots exploring the full map without colliding with obstacles. (a) The Path of the robot in the case of one run of experiment A. (b) The Path of the robot in the case of one run of experiment B. (c) The Path of the robot in the case of one run of experiment H. In (a)-(c), obstacles are shown in red, the robot's current position is shown using a black circle and the path through which the robot has already visited the map is shown in grey.

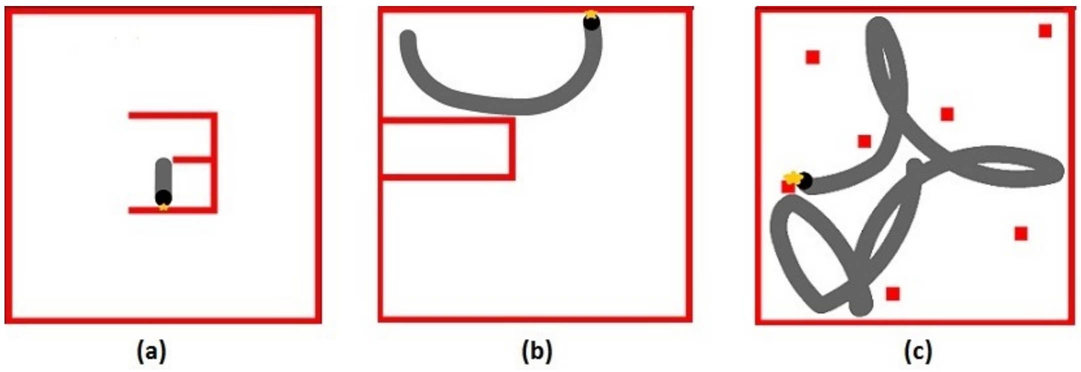


FIGURE 7.11: Paths of robots colliding with obstacles. (a) The Path of the robot in the case of one run of experiment A. (b) The Path of the robot in the case of one run of experiment D. (c) The Path of the robot in the case of one run of experiment H. In (a)-(c), obstacles are shown in red, the robot's current position is shown using a black circle, the path through which the robot has already visited the map is shown in grey and the place where the robot has collided is shown as a yellow star.

7.9 The Real Robot Experiments and the Results

The mbed microcontroller can communicate with a host PC through a 'USB Virtual Serial Port' over the same USB cable which is used for programming. In experiments, the Pi-Swarm robot was run by establishing communication between the Pi-Swarm robot (through mbed) and the computer program which communicates with material via Mecobo. The mbed sent distance values from 8 IR proximity detectors on the robot to the computer which sent the two motor speed values back to the robot. The mbed ran the robot using the motor speed values for 10 milliseconds and then stopped the robot. Then mbed sent 8 distance values to the computer again and another cycle begins. This sequence of operations was performed 5000 times. It should be noted that

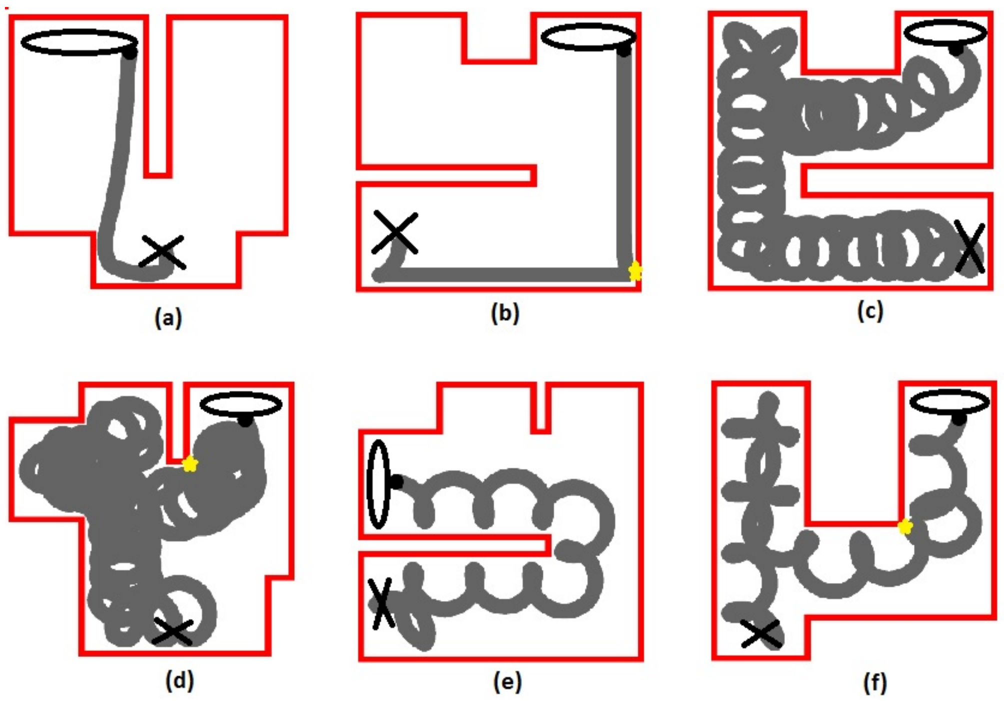


Figure 7.12: Paths of robots solving mazes. In (a)-(f), obstacles are shown in red, the robot's current position is shown using a black circle, the path through which the robot has already visited the map is shown in grey, the place where the robot has collided is shown as a yellow star, the starting position of the robot is marked with a cross and the goal is marked with an oval.

the robot was not stopped if it collided with an obstacle or rotated in the same place for 1000 consecutive moves, unlike the simulated robot. This means that the robot was given the chance to free itself if it was stuck. The robot was halted every 10 milliseconds so that the robot waited to allow Mecobo to finish sending input signals and reading outputs. This was done to make sure that after sending sensor values, the robot would not have moved to a different position when the new motor speeds were obtained. Prior investigation was performed to arrive at the timing of 10 milliseconds. Times larger than 10 milliseconds allowed the robot too much time to move resulting in the harder control and more collisions. Using times smaller than 10 milliseconds meant the robot moved very slowly.

The final solutions of experiments A-C, F and H were tested on a real Pi-Swarm robot to investigate whether solutions evolved with a simulator perform exactly the same or not. Only successful solutions of simulated robot experiments were used in real robot experiments, i.e. only those solutions were used which explored the full map without colliding with obstacles. The real robot experiments focused on observing the behaviour of the robot to explore the full map. This is why solutions obtained in experiments G and

I of simulated robot experiments were not used on the real robot as these experiments were designed to test the robot's ability to cope with a simulated sensor fault and environmental changes. The experiment J was concerned with maze solving problem and this was not used in the Pi-Swarm experiment either. As none of the robots of experiments D and E could explore the full map, none of the solutions of these two sets was used in real robot experiments. Only the successful final solutions of the remaining five sets (sets A-C, F and H) of simulated robot experiments were tested on the real Pi-Swarm robot, in total 13 solutions were tested from these five sets, i.e. in total 13 solutions were successful from these five sets, and these comprised four solutions from set A, three solutions from set B, two from set C, two from set F and two solutions from set H.

It should be noted that the environmental settings of each real robot experiment were arranged to try and make a similar set of environmental settings as the corresponding simulated robot experiment (same map, same starting position of the robot, same input-output timing, same input and output mapping).

The results are discussed as follows:

- Four solutions were successful in experiment A. These four solutions were tested on the real Pi-Swarm robot. None of the robots could escape from collisions. As the robot was not stopped if it collided and it was left to run for up to 5000 moves, one of the four robots did explore half of the map. However, the robot collided several times but did manage to escape. In the case of the other three solutions, the robots collided very soon after starting their journey and could not free themselves within these 5000 moves.
- The three successful solutions in experiment B were tested on the real Pi-Swarm robot. None of the robots could escape from collisions. Only one robot could explore half of the map in spite of colliding several times with obstacles and then escaping. In the case of the other two solutions, the robots collided very soon after starting their journey and could not escape within these 5000 moves.
- Two solutions were successful in experiment C. One robot explored half of the map after colliding several times with obstacles and then escaping. The other

robot collided very soon after starting its journey and could not extricate itself within the full life period (within 5000 moves).

- Two solutions were successful in experiment F. In the case of both solutions, the robot collided very soon after starting its journey and could not make an escape within the 5000 moves.
- Two solutions were successful in experiment H. As the robot was not stopped if it collided, only one robot explored half of the map. The robot collided several times but could escape. The other robot collided very soon after starting its journey and could not free itself within its life period (within 5000 moves).

The analysis showed that the real robot did not perform as well as the simulated robot. This was due to the fact that the simulator was written for Khepera robot and the real robot was performed with the Pi-Swarm robot. A common problem in evolutionary robotics is that if the solution obtained by simulation is transferred in the real world, the behaviour of the solution might not be exactly the same as the behaviour obtained during the simulated evolution. This is called reality gap. The other reasons for the worse performance of the Pi-Swarm robot than the simulated robot were the reality gap, presence of noise and also the extra disturbance caused by having a wire connected to the on-board mbed robot controller and the computer while the robot was running. Wireless communication can be possible, but the sent and received messages have a limitation on sizes (limited number of bytes). This limitation meant the robot would communicate with the computer program too infrequently. Although it was attempted to make the settings of simulated and real robot experiments as similar as possible, the organisation of a map along with obstacles, the size of a map, the proportion of the size of the robot in a map, the distance traversed by the robot by different motor speed values in a map were not exactly the same as those in the simulated robot experiments. These dissimilarities have also affected the performance of Pi-Swarm robot. Figure 7.13 shows an image of real robot (Pi-Swarm) experiment.

7.10 Summary

A summary of outcomes from these experiments is given as follows:

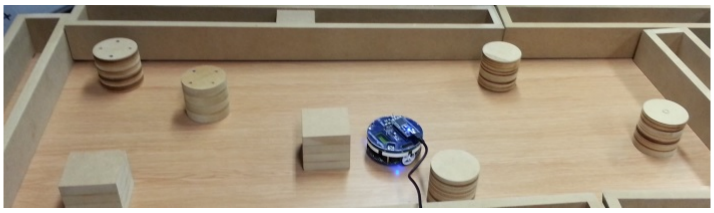


Figure 7.13: The Pi-Swarm robot moving within a map. The image was taken in the middle of one experiment.

- It has been found from the simulated robot experiments that an output mapping based on average transition gap did not provide good results.
- In the case of simulated robot experiments, the results with 6 configuration inputs were not as good as the results with 4 configuration inputs.
- After the four generalization experiments, none of the robots could explore all of the three maps (Figure 7.3 (a)-(c)) without colliding with an obstacle. However, one robot could explore two maps ((a) and (b)).
- None of the robots could explore the full map without colliding with an obstacle in simulated robot experiments of incremental evolution.
- In the fault tolerant simulated robot experiments, one robot collided with an obstacle and could not explore the full map before adding the fault, but after adding the fault, it could explore the full map without colliding with an obstacle. However, none of the robots could explore the full map without colliding with an obstacle before adding the fault.
- In maze solving simulated robot experiments, in training, the third maze (Figure 7.4 (c)) was solved with the lowest number of generations on average than the other two mazes and it took the highest number of generations to solve the second maze (Figure 7.4 (b)) on average. In testing, the maze (d) (Figure 7.4) was solved by more robots than the other two mazes (Figure 7.4 (e), (f)) and the maze (f) was solved by more robots than the maze (e).
- In the case of real robot experiments, none of the robots could explore the full map within its life period. However, in total four robots (out of 13 successful robots of sets A-C, F and H) could explore half of the map in spite of colliding several times with obstacles. The real robot did not perform as well as the simulated robot.

This chapter shows that using purpose-built hardware it is possible to evolve voltages and signals applied to a physical material to control robots, both simulated (Khepera) and real (Pi-Swarm). This is the first time that a mixture of single-walled carbon nanotubes and a polymer has controlled a robot. In some cases, it was found that the robot could explore an environment without colliding with an obstacle or solve all six mazes. Robot control problems using a simulated robot were investigated in the past using evolution-in-materio, where the computational material was liquid crystal [Harding and Miller (2005)]. However, the number of sensors used was two. The simulated robot experiments have been performed here with at least six sensors. Further, the task of the robot has been extended compared to the task performed by the evolution-in-materio controlled robot using an LCD.

The next chapter deals with investigation and analyses regarding the stability of materials, useful characteristics of input signals that are effective for computation, the solutions obtained for various computational problems and the computational problems unsuitable for solving using evolution-in-materio.

Chapter 8

Analysis of Results and Further Investigations

Contents

This chapter contains investigations and analysis regarding the solutions obtained for the computational problems described in Chapters 5, 6 and 7. Previous chapters described solutions of many computational problems, but these solutions cannot be said to be reliable or repeatable unless the used materials are proved to be stable. Some stability tests have been performed using mixtures of single-walled carbon nanotubes and polymers. These stability tests are described in detail in this chapter.

Different input parameters (Table 4.1) were used to generate different types of input signals with the Mecobo hardware platform, which were evolved to get final solutions of the computational problems. Investigations have been performed on these input parameters to see which are effective and which are redundant. Other than the input parameters, different output parameters (Table 4.2) were also used to read outputs from an electrode using Mecobo. These have also been investigated. The investigations and the results are described here.

An analysis has been done on the results of the experiments performed in this thesis on various types of computational problems. This chapter discusses in detail the analysis and the outcomes of the analysis. The outcomes of the analysis along with the problems faced in this research reveal some guidelines on choosing computational problems for future work, which are also discussed in this chapter.

8.1 The Stability Test

In the case of evolution-in-materio, when a number of signals are applied to the material, it cannot be measured whether the molecules go back to exactly the same state that they were in before applying the signals. However, it is important for a physical material to be able to be 'reset' in some way before applying new input signals on it, otherwise it might preserve some memory and might give fitness scores and solutions of the problem, that are dependent on the past behaviour. Thus, it is important in evolution-in-materio to investigate the stability of the material so that if an evolved configuration is applied again, it will give the same output. If the material does not give the same output for the same problem using the same inputs and configuration inputs, the solution obtained during the evolutionary run will not be applicable again. Thus, it cannot be reliably said that the material can give the same solutions always for the same problem under the same condition.

Harding performed some stability tests on the LCD used in his evolution-in-materio experiments [Harding (2006)], which were described in Section 3.3.1. Some stability tests have been performed here to find out whether the results obtained using the mixture of single-walled carbon nanotubes and a polymer are repeatable.

To investigate whether the behaviour of the experimental material is stable and reliable, the final evolved configurations of electrodes from one run of one tone discriminator experiment were re-applied under different circumstances and the responses of the material were measured. The configurations of electrodes for configuration inputs used in that tone discriminator experiment are shown in Table 8.1. The configurations of electrodes for inputs and the average transition gap in an output electrode buffer are shown in Table 8.2. This average transition gap was recorded at the time of the tone discriminator experiment.

The circumstances that are used in the stability tests are described as follows:

- Signals were applied to the electrodes, then Mecobo was stopped and started again (by switching it off and on again), and then subsequently the same signals were re-applied to the same electrodes.
- Signals were applied to the electrodes, then Mecobo was left idle for 1 day, and then subsequently the same signals were re-applied to the same electrodes.
- Signals were applied to the electrodes, then different signals using different configurations of electrodes were applied, and then subsequently the original signals were re-applied to the same electrodes.
- Signals were applied to the electrodes twice one after the other.

In these investigations, the final evolved configurations of electrodes were re-applied under each of these circumstances five times, i.e. in total 20 responses were recorded. It has been observed that in 18 cases, the average transition gap in that output buffer was 14 and in 2 cases, it was 13. It should be noted that these tests were performed after intervals of nine months from the original tone discriminator experiment. The mixture used in these tests was material sample 6 (Table 4.3).

To further investigate stability, other type of output measurement was also examined. In this case, the percentage of ones in an output buffer was measured using a single output electrode. The configurations of electrodes, which were used to seed one of the individuals of the initial population of function optimisation experiment A (Section 6.2), were used in this stability test. An initial evolutionary investigation was performed to discover the typical contents of an output buffer before function optimisation experiment

Table 8.1: Final configurations of electrodes for configuration inputs in one run of tone discriminator experiment using pair of frequencies 500Hz-900Hz. In this experiment, the input-output timing was 250 milliseconds. 'Not applicable' is used when mark-space ratio and frequency are not applicable, i.e. for static voltages.

Configuration input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio
1	5	Wave	1	2886	6
2	11	Static	0	Not applicable	Not applicable
3	7	Wave	1	8748	22
4	0	Static	1	Not applicable	Not applicable
5	3	Static	0	Not applicable	Not applicable
6	9	Wave	1	8485	19
7	6	Wave	1	6584	46
8	4	Wave	1	4987	75
9	2	Wave	1	3738	62

Table 8.2: The configurations of electrodes for inputs and the average transition gap in an output electrode buffer in one run of tone discriminator experiment using pair of frequencies 500Hz-900Hz. This average transition gap was recorded at the time of the tone discriminator experiment. In this experiment, the input-output timing was 250 milliseconds. 'Not applicable' is used when mark-space ratio and frequency are not applicable, i.e. for static voltages.

Input					Output	
Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio	Electrode no.	Average transition gap
10	Wave	1	900	50	1	14

A to seed the initial population of that experiment. In the initial investigation in experiment A, evolutionary runs were carried out to find the electrode configurations that gave different percentages of ones in the output buffer. The different percentages were 0%, 10%, 20%, 30% and 40%. The configurations of electrodes, which gave 40% ones in that investigation, were used in this stability test. The configurations of electrodes used for configuration inputs are shown in Table 8.3. The 9^{th} electrode (electrode number: 8) was used for measuring the percentage of ones. After intervals of three months, the configurations of electrodes were re-applied and varied the environments as before (each of the circumstances for five times and in total 20 responses). It was observed that in 19 out of 20 cases, the percentage of ones was 40% and in one case, it was 39%. The mixture used in these tests was material sample 4 (Table 4.3).

From these stability tests, it has been found that in most cases, the evolved behaviours

Table 8.3: The configurations of electrodes for configuration inputs used to obtain 40% ones in the output buffer of the 9th electrode (electrode no.: 8) of the electrode array of material sample 4 (Table 4.3). In this experiment, the input-output timing was 128 milliseconds. 'Not applicable' is used when mark-space ratio, frequency and phase values are not applicable, i.e. for static voltages.

Configuration input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio	Phase
1	9	Wave	1	3797	92	2
2	2	Static	0	Not applicable	Not applicable	Not applicable
3	4	Wave	1	1090	64	9
4	0	Wave	0	4887	55	4
5	10	Wave	0	1810	69	5
6	5	Wave	1	2161	65	1
7	6	Static	1	Not applicable	Not applicable	Not applicable
8	3	Wave	0	2635	53	4
9	11	Wave	1	2562	54	8
10	7	Wave	1	2815	83	6
11	1	Static	0	Not applicable	Not applicable	Not applicable

are stable. However, in some cases, variations were observed. Further experiments would be needed to investigate the reasons for these variations.

8.2 Investigations On Input and Output Parameters

There are two types of input signals that can be sent using Mecobo: static signals and square wave signals. Many input parameters are used to generate input signals with many variations, which were described in Table 4.1. Of these, some input parameters cause changes in measured outputs and some do not. So, there should be an investigation to find out which input parameters have effects on outputs and which do not have any effect at all. The input parameters that were used in these experiments are electrode number, signal type, amplitude, frequency, mark-space ratio, phase, start time and end time.

Five different sets of investigations were performed on these input parameters. These investigations and the outcomes of the investigations are described as follows:

- In the first (A) set, the electrode number was chosen as the input parameter on which an investigation was performed. Records of the electrode configurations of

Table 8.4: The electrode configurations of configuration inputs before the mutation happened on an electrode number. In this experiment, the input-output timing was 50 milliseconds. The output electrode number was 7. All the configuration inputs were static analogue voltages. Electrodes of the 3^{rd} and the 13^{th} configuration inputs were swapped.

Configuration input	Electrode no.	Amplitude
1	6	121
2	3	178
3	***15***	***25***
4	8	59
5	13	20
6	10	192
7	4	189
8	14	52
9	11	136
10	5	25
11	9	61
12	12	62
13	***0***	***172***
14	2	243
15	1	3

the best-fit individual in each generation for each run were kept in all experiments. After investigation of the records, it was found that electrode number has an effect on outputs. It was found that the average of absolute values in an output buffer of previous generation was increased in the next generation when only a single mutation happened on an electrode number of the full genotype, which swapped an electrode of an configuration input with an electrode of another configuration input. The average of absolute values in the output buffer was 698 before the mutation and 1639 after the mutation. One of the function optimisation experiments (set C) using the Mecobo 3.5 and material sample 1 (Table 4.3) was used in this investigation. The electrode configurations of configuration inputs are shown in Table 8.4.

- In the second (B) set, the phase was chosen as the input parameter on which an investigation was performed. After investigation, it was found that phase does not have any effect on outputs. For this investigation an experiment was performed using 14 inputs and one output. The inputs were a mixture of digital square waves and digital static voltages. In this experiment, at first, a sequence of inputs (a mixture of square waves and static voltages) were sent to the material and the percentage of ones in an output buffer was measured. The phase value of all square

TABLE 8.5: The electrode configurations of 14 inputs before the modification on the phase values. In this experiment, the input-output timing was 25 milliseconds. The output electrode number was 15. 'Not applicable' is used when mark-space ratio, phase and frequency values are not applicable, i.e. for static voltages. The phases of all square wave inputs were modified to the value 10.

Input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio	Phase
1	2	Static	0	Not applicable	Not applicable	Not applicable
2	***0***	***Wave***	***1***	***3186***	***10***	***1***
3	***4***	***Wave***	***1***	***4077***	***10***	***1***
4	5	Static	0	Not applicable	Not applicable	Not applicable
5	7	Static	0	Not applicable	Not applicable	Not applicable
6	***10***	***Wave***	***1***	***3203***	***10***	***1***
7	11	Static	0	Not applicable	Not applicable	Not applicable
8	***3***	***Wave***	***1***	***2210***	***10***	***1***
9	***8***	***Wave***	***1***	***3552***	***10***	***1***
10	1	Static	0	Not applicable	Not applicable	Not applicable
11	9	Static	0	Not applicable	Not applicable	Not applicable
12	6	Static	0	Not applicable	Not applicable	Not applicable
13	12	***Wave***	***1***	***1000***	***10***	***1***
14	13	Static	0	Not applicable	Not applicable	Not applicable

wave inputs was 1. Then phases of all square wave inputs were modified to the value 10 leaving other static inputs the same as before, the inputs are sent to the material and then subsequently the percentage of ones in the same output buffer was measured again. It was found that the percentage of ones in the output buffer of the 16th electrode (electrode number: 15) was the same before and after the modification of the phase values and the percentage of ones was 0.11%. Mecobo 3.0 and material sample 1 (Table 4.3) were used in this investigation. The electrode configurations of 14 inputs are shown in Table 8.5.

- In the third (C) set, the mark-space ratio was chosen as the input parameter on which an investigation was performed. After investigation, it was found that mark-space ratio has an effect on outputs. For this investigation an experiment was performed using 14 inputs and one output. The inputs were a mixture of digital square waves and digital static voltages. In this experiment, at first, a sequence of inputs (a mixture of square waves and static voltages) were sent to

Table 8.6: The electrode configurations for 14 inputs before the modification on the mark-space ratio value. In this experiment, the input-output timing was 25 milliseconds. The output electrode number was 12. 'Not applicable' is used when mark-space ratio and frequency values are not applicable, i.e. for static voltages. The mark-space ratio of the 8th input (electrode number: 1) was modified from 41% to 31%.

Input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio
1	15	Wave	1	6778	90
2	6	Static	0	Not applicable	Not applicable
3	7	Static	0	Not applicable	Not applicable
4	14	Wave	1	5683	56
5	3	Wave	1	829	66
6	2	Static	0	Not applicable	Not applicable
7	11	Static	0	Not applicable	Not applicable
8	***1***	***Wave***	***1***	***8691***	***41***
9	10	Wave	1	9810	49
10	5	Static	0	Not applicable	Not applicable
11	0	Static	0	Not applicable	Not applicable
12	9	Wave	0	3225	20
13	13	Static	0	Not applicable	Not applicable
14	4	Wave	1	10000	50

the material and average transition gap in an output buffer was measured. Then the mark-space ratio of a square wave input was modified leaving other inputs the same as before, the inputs are sent to the material and then subsequently the average transition gap in the same output buffer was measured again. It was found that the average transition gap in the output buffer of the 13th electrode (electrode number: 12) was increased when a modification happened on a mark-space ratio value of one input. The mark-space ratio of the 2nd electrode (electrode number: 1) was modified from 41% to 31% and consequently the average transition gap in the buffer of the 13th electrode (electrode number: 12) was modified from 23 to 32. Mecobo 3.0 and material sample 1 (Table 4.3) were used in this investigation. The electrode configurations for 14 inputs are shown in Table 8.6.

- In the fourth (D) set, the amplitude was chosen as the input parameter on which an investigation was performed. After investigation, it was found that amplitude

Table 8.7: The electrode configurations for configuration inputs before the mutation happened on an amplitude value. In this experiment, the input-output timing was 250 milliseconds. The output electrode number was 1. 'Not applicable' is used when mark-space ratio and frequency are not applicable, i.e. for static voltages. The amplitude of the last configuration input was mutated from the value 1 to 0 (i.e. from 3.5V to 0V). The electrode number of the input was 2. The input was a square wave signal having frequency 2500Hz, amplitude value 1 and mark-space ratio 50%.

Configuration input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio
1	3	Static	1	Not applicable	Not applicable
2	7	Static	1	Not applicable	Not applicable
3	10	Wave	1	1537	20
4	8	Static	0	Not applicable	Not applicable
5	5	Static	1	Not applicable	Not applicable
6	6	Static	0	Not applicable	Not applicable
7	4	Static	0	Not applicable	Not applicable
8	9	Wave	0	2317	60
9	***0***	***Static***	***1***	***Not applicable***	***Not applicable***

has an effect on outputs. It was found that the average transition gap in the output buffer of the second electrode (electrode number: 1) of previous generation was increased in the next generation when only a single mutation happened on an amplitude of the full genotype, which changed the amplitude value of a configuration input from 1 to 0 (i.e. from 3.5V to 0V). The average transition gap values in the output buffer were 4 before the mutation and 36 after the mutation. One of the tone discriminator experiments (pair of frequencies was 500Hz-2500Hz) using the Mecobo 3.0 and material sample 6 (Table 4.3) was used in this investigation. The electrode configurations for configuration inputs are shown in Table 8.7.

- In the fifth (E) set, the frequency was chosen as the input parameter on which an investigation was performed. After investigation, it was found that frequency has an effect on outputs. It was found that the average transition gap in the output buffer of the 7^{th} electrode (electrode number: 6) of previous generation was increased in the next generation when only a single mutation happened on a frequency of the full genotype, which changed the frequency of a configuration input from 3753Hz to 952Hz. The average transition gap values in the output

Table 8.8: The electrode configurations for configuration inputs before the mutation happened on a frequency value. In this experiment, the input-output timing was 250 milliseconds. The output electrode number was 6. 'Not applicable' is used when mark-space ratio and frequency are not applicable, i.e. for static voltages. The frequency of the 7th configuration input was mutated from 3753Hz to 952Hz. The electrode number of the input was 3. The input was a square wave signal having frequency 900Hz, amplitude value 1 and mark-space ratio 50%.

Configuration input	Electrode no.	Signal type	Amplitude	Frequency (Hz)	Mark-space ratio
1	10	Static	0	Not applicable	Not applicable
2	9	Static	1	Not applicable	Not applicable
3	4	Static	0	Not applicable	Not applicable
4	7	Static	1	Not applicable	Not applicable
5	1	Static	0	Not applicable	Not applicable
6	5	Static	0	Not applicable	Not applicable
7	***8***	***Wave***	***1***	***3753***	***25***
8	2	Static	1	Not applicable	Not applicable
9	11	Static	0	Not applicable	Not applicable

buffer were 71 before the mutation and 77 after the mutation. One of the tone discriminator experiments (pair of frequencies was 500Hz-900Hz) using the Mecobo 3.0 and material sample 4 (Table 4.3) was used in this investigation. The electrode configurations for configuration inputs are shown in Table 8.8.

After the five sets of investigations, it has been found that the phase of the square wave input signal does not have any effect on outputs. Other characteristics (electrode number, amplitude, mark-space ratio, frequency) of the input signal have an effect on outputs.

Some experiments were performed prior to these investigations of input parameters and some experiments were performed afterward. The experiments, which were performed before these investigations, used phases as genes in the genotypes and others did not. Set A of function optimisation experiment, sets A, C-M of classification experiments, all frequency classification experiments and all bin packing experiments used phases as genes in the genotypes.

Using the Mecobo platform the time (using start time and end time), by which a signal is applied to the material (Table 4.1), can be controlled. The number of samples stored in the output buffer can be controlled by the start time, end time and the sampling frequency of the output electrode. Inputs are applied here for a number of milliseconds and the outputs are accumulated in a buffer for the same number of milliseconds. This has been referred to as input-output timing in all of the experiments in this research. Some investigations were performed on the input-output timing and output sampling frequency using Mecobo 3.0.

From the investigations, it has been found that if input-output timing is less than 16 milliseconds, no more than 10 samples can be obtained in an output buffer regardless of the output sampling frequency and in the case of input-output timing less than 14 milliseconds, the buffer size is 0 regardless of the output sampling frequency. It should be noted that the number of electrodes used in these investigations was 12. It was also found that if the input-output timing was 16 milliseconds, the buffer contained no less than 150 samples for an output sampling frequency 10KHz. The scheduling in Mecobo (versions 3.0 and 3.5) is serial. This means that sequenced actions do not take place at the same instant of time. It maintains a schedule. It happens even if the same start times and end times are used for all inputs, configuration inputs and output(s). It takes some time for Mecobo to circulate a signal to each electrode, which is ≈ 1 millisecond. Usually, it is more than 1 millisecond, but the actual time was not measured. That is why according to observation, for 12 electrodes, it took 16 milliseconds to get a buffer containing no less than 150 samples for an output sampling frequency 10KHz. Less than this input-output timing (16 milliseconds), the buffer contains no more than 10 samples and less than 14 milliseconds input-output timing, the buffer size becomes 0 regardless of the output sampling frequency.

The time taken by Mecobo to circulate inputs to electrodes and read output buffer(s) from electrode(s) has been referred to as response time of Mecobo (both versions of 3.0 and 3.5) in this thesis. The greater the number of electrodes, the longer the response time of Mecobo (both versions of 3.0 and 3.5).

8.3 The Problems

Some problems have been faced in this research while undertaking experiments. Both Mecobo platforms (Mecobo 3.0 and Mecobo 3.5) sometimes stopped working due to some loose connections in wires or for some other reason. This meant it had to be restarted manually or by using the software. It has been observed that the Mecobo stopped working frequently if experiments were running consecutively for $\approx$ 7 days. This restricted experiments to a duration of less than $\approx$ 7 days. Due to the 16 milliseconds response time of Mecobo, each individual was evaluated no less than 16 milliseconds in the experiments. For these reasons, the number of possible generations was restricted in the experiments. This has resulted in poor results in some of these experiments. Better results could be obtained if experiments could be run for more generations. These problems restricted the size of the datasets that were feasible in machine learning classification problems. The even parity problems were tried only with 3 and 4 inputs, more inputs (i.e. more test cases) need more generations to find better results.

Mecobo 3.5 does not support more than 8 analogue input signals (inputs and configuration inputs). This has caused many problems. Due to this structure, some configuration inputs were forced to have static -2.3V (set B of classification experiments and sets C and D of function optimisation experiments). This means, some mutations were redundant if mutations happened on the genes that dealt with the voltage levels (amplitudes) of those configuration inputs. As a limited number of analogue inputs (inputs and configuration inputs) are allowed in the case of the Mecobo 3.5, many computational problems could not be investigated using Mecobo 3.5, such as classification problems with a higher number of attributes, even parity problems with many inputs, where the number of inputs is more than 8. Even, the computational problems having the number of inputs less than 8, could not be investigated as these would leave very few configuration inputs that could be used in the evolutionary process by having a freedom to choose any voltage level in a range [-5V, 5V]. However, in these cases, square wave input signals could be used.

The Mecobo 3.5 that was used in the experiments was located in Norway. It was possible to run Mecobo 3.5 over internet, but it caused some extra time for the program to communicate with Mecobo 3.5. The long response time of Mecobo along with this extra time of communication caused a delay, which made experiments slow. That is why the experiments could not be run for more generations. For example, classification experiments were run for only 50 generations, and function optimisation experiments were run only for 1000 generations.

Other than the problems of the hardware platform, the number of electrodes of the electrode array in the material slide was limited to 16. This meant many computational problems could not be investigated. For function optimisation and bin packing problems, a split genotype technique was required, which made the experiments longer to complete. Machine learning classification with a large number of attributes or classes could not be investigated due to the limited number of electrodes.

8.4 Analysis of Experiments

This section describes different factors relating to experiments, such as the reasons for choosing the computational problems that were tried in this research, analysis of results obtained in different computational problems and the factors affecting these results. The factors that were found to have effects on results are different types of input mappings, output determination methods, input signals and configuration inputs. A detailed analysis has been performed here on input mappings and output determination methods. Other than the mappings, the types (digital or analogue, static voltages or square waves) of inputs (both input signals and configuration inputs) sent to the materials have also an important role on the measured output. A detailed analysis has been performed here on the input signals based on the comparisons of the results obtained in different experiments.

8.4.1 Choices of Computational Problems

Many computational problems have been investigated using evolution-in-materio with mixtures of single-walled carbon nanotubes and a polymer. These computational problems have been chosen from different perspectives. The reasons behind the choices of these computational problems are described in Table 8.9.

Each of the computational problems which have been solved using evolution-in-materio in this research is different from each other. These different types of problems were

Table 8.9: The choices of computational problems. The 'Problem' column shows the name of the computational problem. The 'Reason for choice' column shows the reason behind the choice of the computational problem.

Problem	Reason for choice
Machine Learning Classification	To assess the use of evolution-in-materio in the field of artificial intelligence.
Tone Discriminator	To compare the results of an LCD [Harding (2006)] against the results of a mixture of single-walled carbon nanotubes and a polymer.
Frequency Classifier	To evaluate the evolution-in-materio on an extended tone discriminator problem.
Even Parity-3 and 4	To assess the application of evolution-in-materio on Boolean problems.
Function Optimisation	To evaluate the effectiveness of the application of evolution-in-materio on function optimisation problems by comparing the experimental results with the results of a famous algorithm (Cartesian genetic programming).
Bin Packing	To assess the application of evolution-in-materio on NP-hard problem.
Robot Controlling	To observe the behaviours of robots (simulated and real) in performing various tasks.

attempted here with a motive of identifying those problems that can be solved using a mixture of single-walled carbon nanotubes and a polymer. Of course, not all computational problems were found to be suitable. It is quite difficult to define the set of problems that is suitable to be solved using evolution-in-materio unless various types of problems are investigated.

8.4.2 Analysis of Experimental Results

In the evolutionary experiments reported in this thesis, it was found that the fitness improved with increasing number of generations and that initial random populations always gave bad results. Evidence was given in Figures 5.8, 6.7-6.11 (frequency classification and bin packing). Many sets of experiments were performed on each of the computational problems. The best experimental result(s) of each of these computational problems is described in Table 8.10.

Table 8.10: The analysis of results of the experiments of this thesis. The 'Problem' column shows the name of the computational problem. The 'Result' column shows the best experimental result(s) of each of these computational problems. The experimental results are described in detail in Chapters 5-7. The optimum results of function optimisation and bin packing problems are given in Chapter 6.

Problem	Result
Machine Learning Classification	Results using analogue signals by Mecobo 3.5 obtained accuracy 91.3% in training and 86.6% in testing in the case of Iris dataset. These results were better than the results obtained by Cartesian genetic programming.
Tone Discriminator	100% accuracy in all runs in all experiments.
Frequency Classifier	100% accuracy in all runs in training and the best (the best of all runs) accuracy 100% in testing in all experiments.
Even Parity-3 and 4	100% accuracy in all runs in 2 sets of even parity-3 experiments and the best (best of all) runs) accuracy 100% in 1 set of even parity-4 experiment.
Function Optimisation	In 7 out of 23 cases, the best results were equal to the optimum results and in 5 cases, the average results were equal to the optimum results. In 10 cases, the best results of the experimental material were better than or equal to the best results of Cartesian genetic programming and in 12 cases, the average results of the experimental material were better than or equal to the average results of Cartesian genetic programming.
Bin Packing	In 3 out of 12 benchmarks, the average and best results of the experimental material are within 15% of the optimum.
Robot Controlling	The results of the robot controlling problems were found to be dependent on choices of tasks, complexities of maps and also on input and output mappings. Detailed results of simulated robot experiments are described in Tables 7.7 and 7.8. The performance of the real robot experiments were worse than the performance of the simulated robot experiments.

8.4.3 Analysis of Input and Output Mappings

Other than different types of computational problems, different types of input and output mappings were used and their suitabilities were assessed. Frequencies were used for input mappings in most experiments, especially classification-based problems (all experiments of tone discriminator and frequency classifier problems, sets C-M of machine learning classification experiments and sets A, G of even parity experiments). Average transition gaps were used for classifying outputs in those classification-based problems which used frequencies for input mappings. This is due to the fact that a transition-based output is frequency related. Other than the classification-based problems, frequency mappings were used for inputs in robot controlling experiments, but in those experiments, none of the robots could explore the full map without colliding with obstacles (set E of robot controlling experiments of Chapter 7) and this might be due to the fact that average transition gaps were used for output mappings. Computing the average transition gap proved to be a better method for determining output classes by comparison (according to sets A, B and C of even parity experiments), but this was not effective for linear mapping to get output values within a specific range (according to sets D and E of robot controlling experiments). In that case, the percentage of ones was better, i.e. the percentage of ones performed better for linear output mapping, which was used for output mapping in function optimisation (sets A and B), bin packing (all experiments) and robot controlling (all sets except the sets D and E) experiments. However, the percentage of ones was effective for classifying outputs other than the average transition gap, especially when mark-space ratio or amplitude was used for input mapping (according to sets B-D, F and H of even parity experiments). The percentage of ones was used in these experiments as it is mark-space ratio and amplitude related.

In the set E of even parity-3 experiments, output class was determined by observing values obtained from the buffer of one output electrode. This was achieved by fixing a threshold, below which the output was decided to be one class, otherwise it was decided to be another class. The result was worse than the result obtained by using two output electrodes in the set D of even parity-3 experiments. In experiment D, the problem instances were classified by comparing the output values obtained from these output electrode buffers. So, experiments D and E showed that in the case of even parity-3 problem, it was better to use as many output electrodes as the number of output classes.

Other than the frequencies, mark-space ratios (used in sets A-D, F-J of robot control experiments and sets B, F of even parity experiments) and amplitudes (used in sets C-E and H of even parity experiments) were used for input mappings. Based on comparison results of sets A, B and C of even parity-3 experiments, frequency was the best of all types of input mappings, and the performance of using amplitude was better than the performance of using mark-space ratio for input mapping. The amplitude input mapping has a drawback in the case of Mecobo 3.0. As Mecobo 3.0 supports only two values for amplitude, amplitude input mapping can only be used in those computational problems, where inputs can have no more than two values, such as Boolean problems.

The input mappings and output determination methods discussed previously are related to the Mecobo 3.0. The experiments of Mecobo 3.5 used amplitudes for input mappings and the average values of output buffers for classifying outputs in machine learning classification experiments using Iris dataset. This proved to be efficient as the results obtained by Mecobo 3.5 were better than the results of Mecobo 3.0.

8.4.4 Analysis of Inputs, Outputs and Configuration Inputs

An analysis has been performed on different types of inputs, outputs and configuration inputs, which has compared different types of configuration inputs, analogue with digital signals and digital square waves with digital static voltages for inputs. These comparisons are described in following sections.

8.4.4.1 Comparison Between Analogue and Digital Signals

As noted in Section 8.4.3, the results obtained by Mecobo 3.5 were better than the results of Mecobo 3.0, which showed that in the case of classification experiments using Iris dataset, the performance of using analogue inputs, output(s) and configuration inputs was better than the performance of using digital inputs, output(s) and configuration inputs (according to comparison results of sets B and C of machine learning classification experiments). It should be noted that Mecobo 3.0 used digital square waves and Mecobo 3.5 used static analogue voltages as input signals in the classification experiments that compared the performances of two Mecobo platforms. By comparing the results of Mecobo 3.0 and Mecobo 3.5, in the case of input signals, static analogue voltages were more effective than digital square waves. However, no strong conclusion can be made regarding input signals using this comparison as configuration inputs, input and output mappings were different in these experiments of Mecobo 3.0 and Mecobo 3.5, and this might have a strong effect on the result.

8.4.4.2 Comparisons Between Digital Static Voltages and Digital Square Waves for Inputs

Two comparisons were performed between the static voltages and square waves as input signals in even parity experiments using Mecobo 3.0, where both the static voltages and the square waves were digital. In the first comparison, the results obtained by square wave inputs were better than the results using static input signals (according to the comparison of sets A and C of even parity experiments). In this comparison, frequencies and amplitudes were used for input mappings in the case of square wave inputs and static input signals respectively. In another comparison, the results obtained by static input signals were better than the results using square wave inputs (according to the comparison of sets B and C of even parity experiments). In this comparison, mark-space ratios and amplitudes were used for input mappings in the case of square wave inputs and static input signals respectively. It should be noted that in all of these even parity experiments (sets A-C), mixtures of digital square waves and digital static voltages were used for configuration inputs. So, comparisons between digital square waves and digital static voltages for inputs cannot draw any conclusion. Other than the input signals, choices of configuration inputs also influence the results of experiments, these are discussed in following section.

8.4.4.3 Comparisons of Different Types of Configuration Inputs

Machine learning classification experiments using Iris dataset showed that the performance of using static analogue voltages was better than the performance of using mixtures of static analogue voltages and digital square waves for configuration inputs (according to sets A and B). These experiments were performed using Mecobo 3.5. A similar outcome was obtained in the case of Mecobo 3.0 using even parity-3 experiments (sets C and D), which showed that the performance of using digital static voltages was better than the performance of using mixtures of digital square waves and digital static voltages for configuration inputs. It should be noted that in the case of the comparison of these even parity-3 experiments, the difference of the results is statistically not significant. In addition, in the case of both of these outcomes regarding configuration inputs of Mecobo 3.0 and Mecobo 3.5, the experiments used static voltages (digital in the case of Mecobo 3.0 and analogue in the case of Mecobo 3.5) for inputs.

Another analysis was performed using the results from the even parity-3 experiments (sets B and F) with Mecobo 3.0, where digital square waves were used for inputs instead of digital static voltages, which showed that the performance of using digital square waves was better than the performance of using mixtures of digital square waves and digital static voltages for configuration inputs. However, the difference of the results is statistically not significant. After analysis of the outcomes of Mecobo 3.0 and Mecobo 3.5 regarding configuration inputs, it can be said that the performance of using only static voltages was better than the performance of using mixtures of square waves and static voltages when input signals were static voltages. Furthermore, the performance of using only square waves was better than the performance of using mixtures of square waves and static voltages for configuration inputs when input signals were square waves. Thus, the performances of different types of configuration inputs may be influenced by the input signals. If the input signals and configuration inputs are combined together and defined as input signals, the performance of using either all static voltages or all square waves was better than the performance of using mixtures of static voltages and square waves. However, the difference in the results is statistically not significant in the case of Mecobo 3.0. This requires further investigation. This outcome might be influenced by some other factors such as input mappings, output determination methods and types (digital or analogue) of input signals.

8.4.5 Analysis of the Influences of Material Samples

Machine learning classification (with Iris dataset) and tone discriminator experiments compared the performances of different mixtures of materials (different percentages of single-walled carbon nanotubes in PMMA, same percentage of single-walled carbon nanotubes in different types of polymers) and different organisations of electrodes, however it was not possible to draw any strong conclusion from these experiments. Tone discriminator experiments and machine learning classification experiments using Iris dataset were performed to investigate whether different organisations of electrodes matter or not. These experiments used the same mixture of material (1.0% single-walled carbon nanotubes in PBMA), but the organisations of electrodes were different. These experiments were performed using Mecobo 3.0. In the case of both of these experiments, the statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.

Tone discriminator experiments and machine learning classification experiments using Iris dataset were performed to investigate whether the choice of polymer in material mixture matters or not. These experiments used the same percentage (1.0%) of single-walled carbon nanotubes, but the polymers (PBMA or PMMA) were different. Both of these experiments were performed using Mecobo 3.0. In the case of both of these experiments, the statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.

No evolution happened using an electrode array containing no material (Sections 6.3 and 5.4), where the output buffers were always full of zeroes. No evolution is possible with a material having 0% single-walled carbon nanotubes (only PMMA), where the output buffers are always full of zeroes (Sections 6.3 and 5.3). This shows that single-walled carbon nanotubes are required in material mixture for computation using the configuration reported in this thesis.

The bin packing and tone discriminator experiments found that no evolution took place when a mixture containing 0.01% single-walled carbon nanotubes with PMMA was used and the output buffers were always full of zeroes. When experiments were started with a mixture containing 0.02% single-walled carbon nanotubes with PMMA, evolution happened with mixtures of 0 and 1 in the output buffers.

8.4.6 Summary of Outcomes Obtained by Analysis of Experiments

Section 8.4 shows different types of analyses regarding the experiments performed in this research. The outcomes of all these analyses are summarised as follows:

- The computational problems solved in this research were chosen from different perspectives so that the applications of evolution-in-materio can be assessed on various types of problems. The reasons behind the choices of these computational problems are described in Table 8.9. The best experimental result(s) of each of these computational problems is described in Table 8.10.
- In the case of even parity-3 experiments using Mecobo 3.0, frequency was the best of all types of input mappings and the performance of using amplitude was better than the performance of using mark-space ratio for input mapping.

- In the case of Mecobo 3.0, amplitudes can be used for input mappings only in those computational problems, where inputs can have no more than two values.
- In the case of classification experiments using Iris dataset and Mecobo 3.5, amplitude for input mapping and average of output values of buffers for classifying outputs were very effective.
- In the case of even parity-3 experiments using Mecobo 3.0, the performance of using two output electrodes was better than the performance of using one output electrode.
- In the case of classification experiments using Iris dataset, the performance of using analogue inputs, output(s) and configuration inputs was better than the performance of using digital inputs, output(s) and configuration inputs according to comparison results of Mecobo 3.5 and Mecobo 3.0.
- In the case of classification experiments using Iris dataset and Mecobo 3.5 and even parity-3 experiments using Mecobo 3.0, for configuration inputs, the performance of using only static voltages was better than the performance with mixtures of square waves and static voltages when input signals were static voltages. Furthermore, the performance of using only square waves was better than the performance of using mixtures of square waves and static voltages for configuration inputs when input signals were square waves. Thus, the performances of different types of configuration inputs may be influenced by the input signals. If the input signals and configuration inputs are combined together and defined as input signals, the performance of using either all static voltages or all square waves was better than the performance of using mixtures of static voltages and square waves.
- Tone discriminator experiments and machine learning classification experiments using Iris dataset were performed to investigate whether different organisations of electrodes matter or not. These experiments used the same mixture of material (1.0% single-walled carbon nanotubes in PBMA), but the organisations of electrodes were different. These experiments were performed using Mecobo 3.0. In the case of both of these experiments, the statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.

- Tone discriminator experiments and machine learning classification experiments using Iris dataset were performed to investigate whether the choice of polymer in material mixture matters or not. These experiments used the same percentage (1.0%) of single-walled carbon nanotubes, but the polymers (PBMA or PMMA) were different. Both of these experiments were performed using Mecobo 3.0. In the case of both of these experiments, the statistical significance tests have shown that the difference of results is statistically not significant according to U-test and KS-test.
- No evolution happened with an electrode array containing no material, where the output buffers were always full of zeroes.
- No evolution is possible with a material having 0% single-walled carbon nanotubes (only PMMA), where the output buffers are always full of zeroes. This shows that single-walled carbon nanotubes are required in material mixture for computation using the configuration reported in this thesis.
- The bin packing and tone discriminator experiments found that no evolution took place when a mixture containing 0.01% single-walled carbon nanotubes with PMMA was used and the output buffers were always full of zeroes. When experiments were started with a mixture containing 0.02% single-walled carbon nanotubes with PMMA, evolution happened with mixtures of 0 and 1 in the output buffers.

8.5 Guidelines of Choosing Future Computational Problems

After the analysis of the experiments and the discussion regarding problems faced, some guidelines can be given in choosing computational problems suitable for future work:

- Those classification-based problems, which have a large number of test cases, are not suitable for solving with a hardware whose response time is large, such as machine learning classification problems with many instances, n-bit even parity problems with large values of n (number of test cases increases with the value of n), image filtering problem. The problem, which has a large number of test cases, will need a long time for the evaluation of each individual in the case of each generation and then finally will need a long time to complete the evolutionary run. For instance, in the case of image filtering problem, evaluation is performed on each of the pixels of the image in the case of each individual in each generation, which then takes a long time to complete each evolutionary run.

- It is better if the electrode array used in the evolution has a significant number of electrodes. After using a number of electrodes for inputs and outputs by the experiments, there should be reasonable number of configuration inputs for the evolution, otherwise evolution will stagnate after a certain generation and better results might not be obtained.

- For solving problems having many outputs, especially when the number of outputs is more than the number of electrodes of an electrode array, split genotype technique will be required. Solving a problem using the split genotype technique may need more generations to find a better solution, and more generations will require more time to complete the evolutionary process. In this case, bin packing problem is a good example, which has been addressed in this thesis.

- Those computational problems, which need a very small response time, cannot be performed using a hardware whose response time is higher than the response time required for the problem, such as solving pole balancing problem, especially if a real pole is used, which needs a very small response time.

8.6 Summary

This chapter focuses on the issues relating to all the experiments of this research, rather than any particular experiment in solving a computational problem. These issues are generated by the analysis of the results of experiments, by the problems faced in solving computational problems and other investigations relating to experiments. This discussion will assist in drawing up guidelines for future work.

The next chapter draws some conclusions of this thesis and gives some ideas for future work.

Chapter 9

Conclusions and Future Work

Contents

This thesis has shown that it is possible to solve a number of computational problems using mixtures of single-walled carbon nanotubes and polymers using evolution-in-materio. The problems examined are: machine learning classification, tone discriminator, frequency classification, even parity, function optimisation, bin packing and robot control.

The first part (Chapters 2 and 3) of this thesis laid the foundations for this research, describing a variety of techniques and aspects that motivated the work of the thesis. These were evolutionary algorithms, evolution-in-materio, former published research on evolvable hardware, evolvable motherboards, and physical computation.

Chapter 4 described the experimental system of this thesis. This includes the description of the hardware platform, the interface software and the experimental material used in the experiments of this thesis.

Chapters 5-7 described the experiments for solving machine learning classification, tone discriminator, frequency classification, even parity, function optimisation, bin packing and robot controlling problems. The experimental mixtures used in the experiments of this thesis are the mixtures of single-walled carbon nanotubes and polymers. This is the first time that these mixtures have been used to solve such problems using evolution-in-materio.

The hypothesis that was presented in Chapter 1 has been robustly supported as solutions to machine learning classification, tone discriminator, frequency classification, even parity, function optimisation, bin packing and robot controlling problems, which have been obtained using computer controlled evolution of signals applied to electrode arrays containing a mixture of single-walled carbon nanotubes and a polymer. Also, the work of this thesis has been presented in some publications through conference and workshop proceedings or journal papers. Of these, four papers have already been published, and four papers have been reviewed and accepted.

Chapter 8 discussed the analysis of the results obtained in the experiments concerning these computational problems. Other than the analysis of the experimental results, Chapter 8 analysed the input-output mappings and input-output signals used in the experiments, revealing some analytical results about the suitable mappings and signals. These analytical results have demonstrated that this research has devised and investigated suitable input signals and input-output mappings which allow various computational problems to be solved using electrode arrays. Summaries of outcomes from these experiments have also been presented. In this chapter, the possibilities for future work are discussed.

9.1 Future Work

Although the experimental results obtained here using evolution-in-materio and a mixture of single-walled carbon nanotubes and a polymer are promising, it is unlikely that this can be applied on any serious application. Subsequently, there is a vast field of research ahead in evolution-in-materio. Chapter 8 described the problems faced in these evolution-in-materio experiments and has drawn some guidelines for choosing future computational problems. The analysis of the experiments, the problems and the guidelines of choosing future problems have revealed some future work which is listed as follows:

- There are a vast number of other materials that could be used in evolution-in-materio experiments to solve various types of computational problems, and one or more of these may lead to useful systems.

- There can be many other properties of mixtures of single-walled carbon nanotubes and polymers, which have not been exploited here. So, future evolution-in-materio experiments may exploit those properties and may solve various computational problems efficiently.

- There should be more investigations on the features of the physical materials required for evolution-in-materio experiments, which then help to identify the set of materials that can be exploited to solve problems.

- It is still unknown what types of computational problems are most suitable to be solved by the evolution-in-materio method using a suitable physical material. There should be more investigations to identify those computational problems and features (e.g. how complex a problem can be) of the problems which are most suitable to be investigated using evolution-in-materio.

- The computational problems solved in this thesis can be repeated with some variations. The robot controlling experiments for example can be repeated with more maps, different starting positions and various tasks, and the machine learning classification experiments can be performed with different datasets. Here in the experiments, the successful solutions of simulated robot experiments were tested on the real robot. In future, the evolutionary experiment can be performed directly using the real robot.

- The factors (different robots used in the case of simulated and real robot experiments, dissimilarities of experimental settings of simulated and real robot experiments, disturbance caused by the wire) that were responsible for the worse (worse than the performance of the simulated robot) performance of the real robot can be overcome in future and then the real robot experiments can be repeated again. The factors can be overcome by making the experimental settings of the real robot experiments much closer to the experimental settings of the simulated robot experiments, using a wireless communication. Even the real robot experiments can be repeated using a different robot such as Khepera or e-puck.

- Here evolution-in-materio has largely solved single instances of problems. For example, even parity-3 or a single bin packing problem. However, evolution-in-materio should be able to solve any instance of a problem. There should be an investigation on the issue as to whether evolution-in-materio can be applied to any instance of these solved problems (in other words, can evolution-in-materio be used to develop algorithms?)

- So far, only 12 or 16 electrodes have been used as no electrode array contains more than 16 electrodes. More electrode arrays can be provided with a large number of electrodes (such as 50, 100, more than 100 electrodes), which then can be used to solve those problems that consist of a large number of inputs and outputs. For example, machine learning classification problems having a large number of attributes and even parity problems with many inputs.

- It has never been investigated as to how much material is required for computation or whether the amount of material matters or not. There should be an investigation on this issue.

- So far, digital square waves, static digital voltages and static analogue voltages have been used as input signals with various amplitudes, frequencies and mark-space ratios. There should be more investigations on the characteristics of the input signals which allow better results to be obtained. The investigation on the ranges of amplitudes of analogue voltages and the ranges of frequencies or mark-space ratios of square waves can be further examined.

  Different combinations of inputs and configuration inputs have been used and their performances have been compared here. However, some combinations have not been investigated yet, such as static digital voltages for inputs and square waves for configuration inputs; square waves for inputs and static digital voltages for configuration inputs; static analogue voltages for inputs and square waves for configuration inputs; square waves for inputs and static analogue voltages for configuration inputs. These combinations can be used in future experiments which then compare their performances.

- The square waves that have been applied as inputs in these experiments using both Mecobo 3.0 and Mecobo 3.5 are digital (with amplitudes 0V and 3.5V). The future interface hardware should provide the option of using analogue square wave input signals which will support various amplitudes instead of only two values.

- It would be helpful to see if the final gene values of configuration inputs of the evolutionary experiments generate a common pattern, which may help to understand and exploit the physical properties of the material to solve problems. On this matter, an investigation was performed in this thesis using one of the frequency classification experiments, which was unable to find any common pattern, but more investigations should be performed in future.

- The computation performed here to solve problems required a physical material. An investigation can be performed without the Mecobo board and the PC to see whether the material can give the same solutions that have been obtained in this research in various computational problems. This could be achieved by building a standalone system (i.e. no PC or Mecobo board) that uses some interfacing electronics and the experimental material with an electrode array.

- The standalone device (i.e. no PC, or Mecobo board) that has been mentioned above can be built to transfer the solution of frequency classifier problem to frequency filter using some interfacing electronics and the experimental material with an electrode array.

- Also, an n-bit even parity standalone device with an input and output module can be built using the solutions obtained here in the even parity problems. The objective would be to make the same standalone device for n-bit even parity problem with the same input and output module for various values of n. The input module will handle the input mappings and output module will handle the output mappings. The standalone device will not include any PC or Mecobo hardware. It will include some electronic circuitry having a common input module (supports different numbers of inputs), a common output module and the experimental material with an electrode array.

  However, it cannot be decided at this stage whether building the n-bin even parity standalone device is worthwhile as no even parity problem has been solved with a higher number of inputs (more than 4) here. In future, there should be more experiments that will attempt to solve more even parity problems with a higher number of inputs. Also, there should be some investigations in future as to how efficiently even parity problems can be solved using the standalone device and how complex the solutions can be.

- The experiments performed here can be repeated with more generations using new hardware platforms that have shorter response times. In this way, better results might be obtained with a faster evolutionary run. Even new computational problems can be tried.

- It has never been investigated as to how fast the material (a mixture of single-walled carbon nanotubes with a polymer) computes, i.e. the response time of the material. If a standalone device will be built or any new fast (low response time) interface hardware will be used to perform experiments with the material in future, the response time of the material could be measured.

- Two investigations have been performed here to find out the best mixture of material (single-walled carbon nanotubes with PMMA) using the machine learning classification and tone discriminator experiments. However, no conclusion can be drawn regarding the best mixture. There should be more investigations to find out the best mixture of single-walled carbon nanotubes in a polymer.

- The stability test proved the material to be stable in most cases, however sometimes a little variation was observed in two results with the same configurations of electrodes. This requires more investigations as to how much variation occurs between results achieved under the same circumstances, the possible reasons why the results differ and so on. Also, as mentioned before that the stability of the material was examined by re-applying evolved configurations after intervals of 3 or 9 months, but it should be investigated the stability of the material by using a longer time period. Also the sensitivity of the material to other environmental conditions could be investigated (e.g. temperature).

- Different methods of evolutionary algorithm for recombination, mutation, evaluation, selection, termination criteria can be tried in future on the same computational problems that were solved here (or on different computational problems). These include $(1, \lambda)$-evolutionary algorithm, $(\mu + \lambda)$-evolutionary algorithm with various values of μ and λ, experiments with various population sizes, experiments using recombination (crossover), mutation by probability having different probabilities of mutation and so on.

- Different input and output mappings can be tried in future on the same computational problems that were solved here (or on different computational problems).

Appendix A

The Components of Mecobo Hardware

The contents of the description of different components of interface hardware are provided from NASCENCE project reports.

A.1 The Motherboard

The main top level block diagram of the motherboard is shown in Figure A.1. The block diagram shows the main components and communication buses. The USB interface is the external communication port. The microcontroller runs the system software of the interface hardware communicating with the FPGA on the internal bus of the board and the host computer over the USB bus; maintains queue and schedule stimuli to the material. If the motherboard is to be used without any daughter board, the input and output signals between the FPGA and the material are in principle a 110 channel crossbar matrix allowing any connection to the material to act as a configurable input or output port. The FPGA provides a reconfigurable interface to materials and daughter boards. Mecobo is equipped with debug interfaces if the motherboard is to be used without any daughter boards, in this case material samples are connected directly to the connection headers. The pins of header N and header W can be configured as inputs or outputs. Two debug interfaces are used to ease debugging of the system software of the microcontroller and to add a possibility to use on-chip debugging tools on the

FPGA, i.e. chip scope. The same headers, i.e. header N and header W are used to communicate with daughterboards. The use of an FPGA enables the users to adapt the logical interface to daughter boards without redesigning the basic interface. Several daughter boards can be stacked on top of each other. The limit for maximum number of daughterboards is set by the available pins on header N and header W.

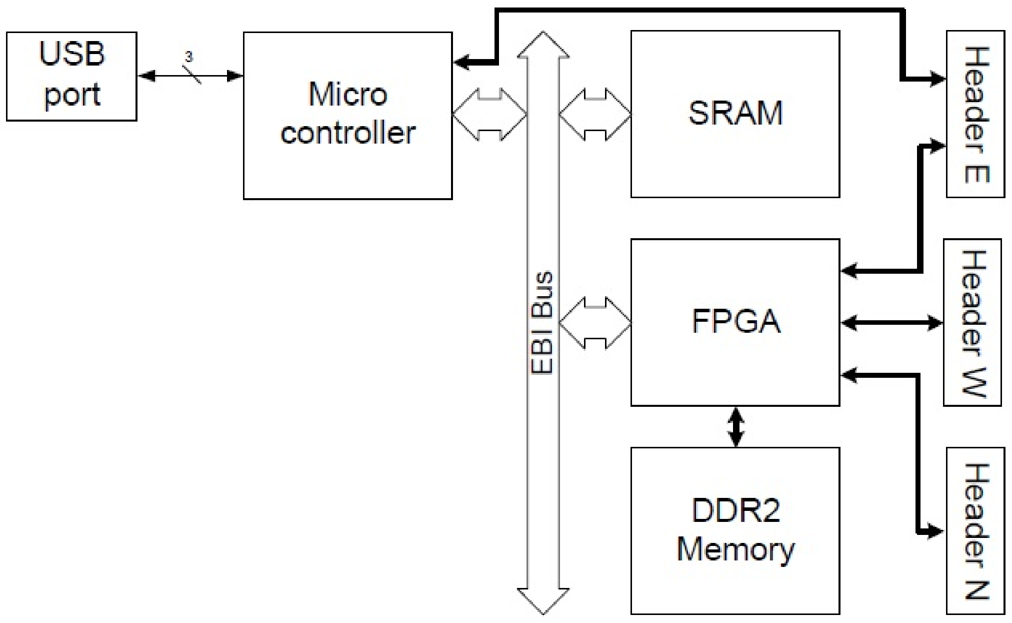


Figure A.1: Block diagram of the top level of the interface motherboard

Photo of motherboard is shown in figure A.2. The main components are labeled in this figure.

The main components used in the design:

- FPGA: XC6SLX25-2CSG324I in a 324 pin BGA package from Xilinx.
- Microcontroller: EFM32GG990F1024 in a 112 pin BGA package from Silicon LABS.
- SRAM: CY7C1061DV33 in a 48 pin BGA package (2 chips 32Mbit total) from Cypress Semiconductor.
- DDR2 DRAM: MT47H128M8CF-25E in a 60 pin BGA package (1Gbit) from Micron Technology.
- Headers: standard 2.54 mm double row 60 pins header.

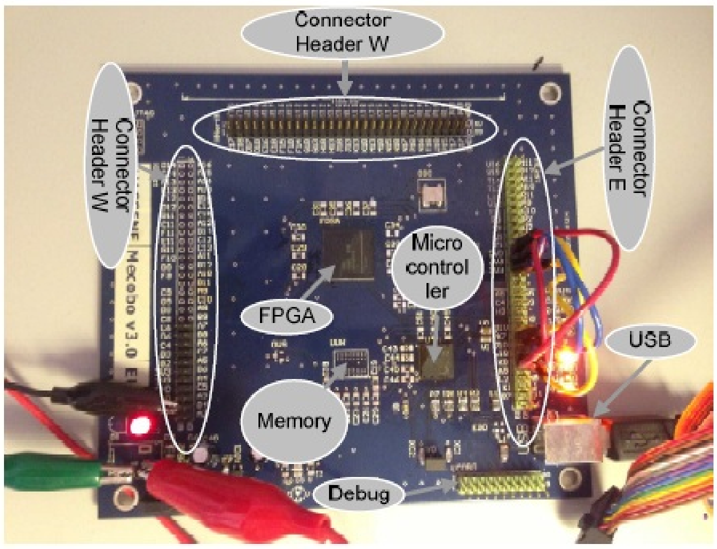


FIGURE A.2: The motherboard of Mecobo.

.

A.2 The FPGA

The block diagram of the functional units of FPGA is shown in figure A.3.

The description of different components of FPGA is given as follows:

- Communication: The communication module handles all communications with the microcontroller.
- Control: The control block decodes commands and outputs control signals on the control bus to set up executing units to handle data correctly. Control also watches other units to carry on their operations correctly.
- Memory (gene): Memory stores genetic information.
- Memory (buffer): Buffer stores data, such as response from material, waveforms. The sample buffer is filled with samples. The sampling rate is configurable and specified by software commands.
- Signal generator: Components applying signals to the material. Each pin has a signal generator. It generates waveform sequences internally or generates waveforms from samples in memory.

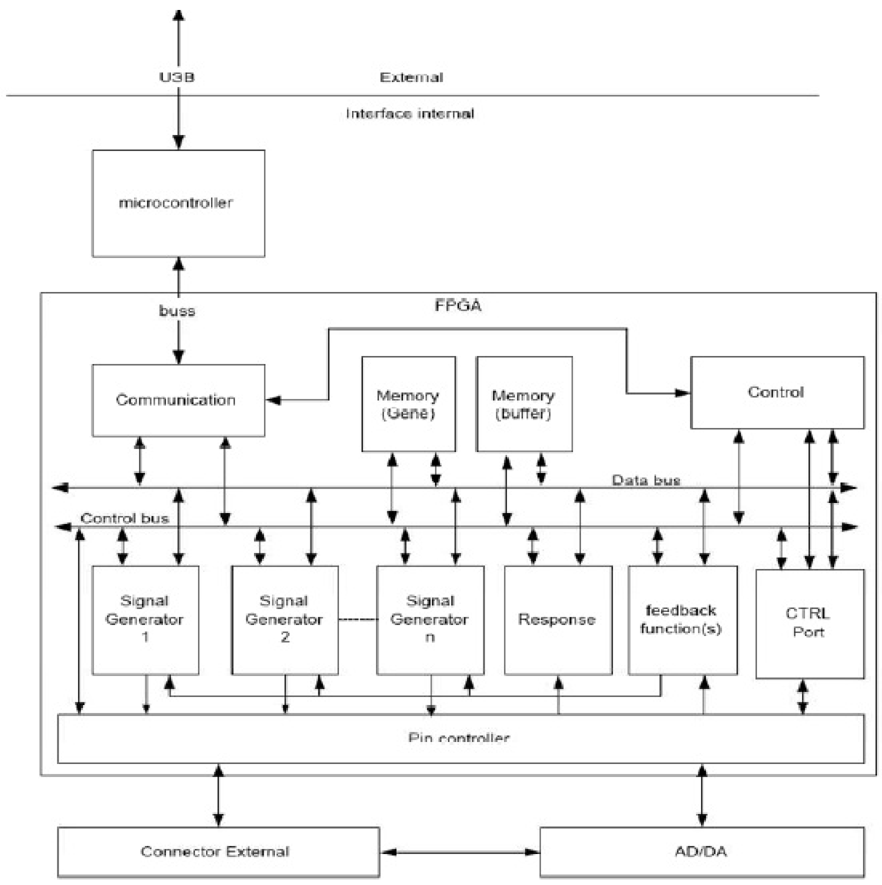


FIGURE A.3: Block diagram of the functional units of the FPGA

- Response: The response data from the material will be stored in this module. It handles all signals that are defined as output from the material. It is possible to include experiment specific extra logic for post processing in this unit.

- Feedback function(s): The module supports the use of responses from the material to influence on the operation of the signal generators, or opens for defining feedback loops including material and signal generator(s).

- Pin controller: It is a switch matrix. Any pin can go to any signal generator or to other modules connected to the pin controller. The functionality of the pin controller is partly defined in the interface software. The software ensures that the pin mapping is valid.

- CTRL port: Pins connected to the CTRL port can be used to steer external equipment and open for the use of external triggers.

- Buses: The USB bus is the interface to the microcontroller. Control bus is the internal bus used for the configuration and status of all units in the FPGA. The internal data bus is used to transfer data internally between units. The two external buses, one of which connects the FPGA pins to headers (connector external) and another connects to some special connection points that are connected to AD/DA I/O.

The PCB for the interface includes a power supply providing 1.2V for the FPGA core. A 1.8V supply voltage is used to power the DDR2 module. The microcontroller is powered by a 3.5V supply voltage. The 1.8V and 3.5V are also used to provide power to the FPGA I/O banks. A quartz oscillator is used to clock the microcontroller. The FPGA is clocked by an active clock generator.

A.3 The Mixed Signal Daughterboard

The mixed signal daughter board is designed to expand the experimental system to include dedicated analogue signals. These analogue signals are produced by DA/AD converters instead of the pulse width modulated (PWM) channels supported by a stand-alone Mecobo.

Figure A.4 shows the block diagram of a single daughter board connected to a Mecobo motherboard. One mixed signal daughter board has 8 digital to analogue channels, 8 analogue to digital channels and 16 digital I/O channels. The configurable crossbar enables any combination of AD, DA or digital I/O signal to be connected to any (from one to all) of the 16 I/O pins (on the right of the figure). All communications between the daughter board and Mecobo are done by synchronous serial interfaces (SSIs) and 16 configurable digital I/O lines. The number of available lines depends on the number of pins dedicated to daughter boards.

The mixed signal daughter board is designed to add new features to the experimental system of Mecobo 3.0. That is, from a user perspective all of the previous principles regarding applying or sampling data and configuration signals are kept as before (such as digital inputs and outputs, square wave inputs). The user can now use any pin connected to the material in the same way as with previous versions (Mecobo 3.0) of the

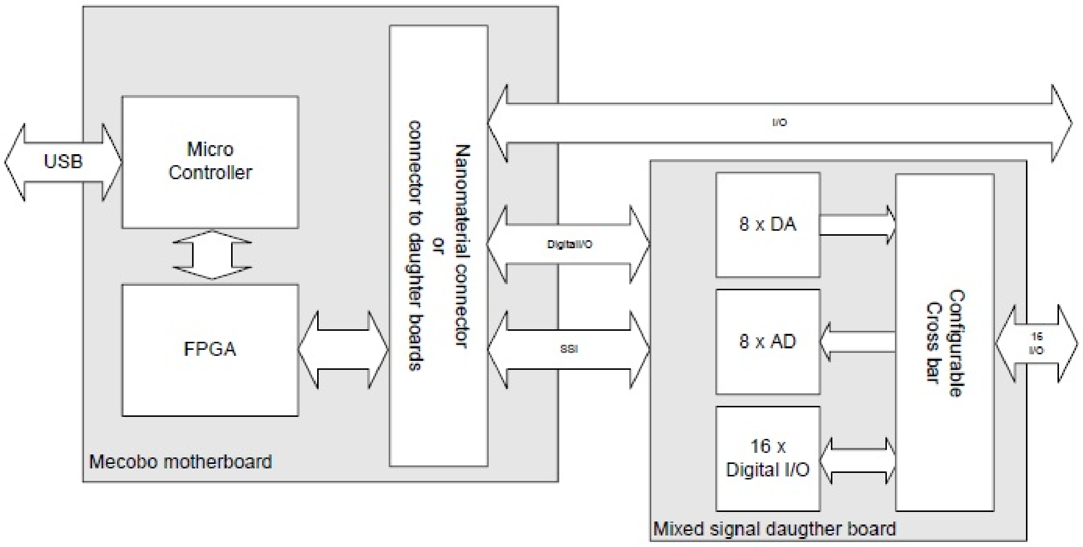


FIGURE A.4: Block diagram of mixed signal daughter board connected to the Mecobo motherboard

system. The added features are analogue inputs and outputs that can be defined and passed on to the material using interface software. The motherboard with the mixed signal daughterboard is named as Mecobo 3.5. Figure A.5 shows a photo of a mixed signal daughter board piggybacked on the Mecobo. The material sample is plugged in to the daughter board's sample holder.

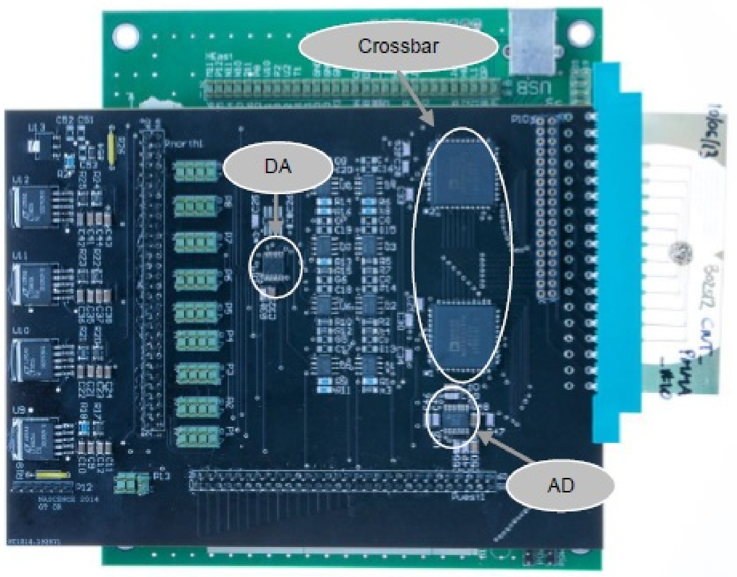


FIGURE A.5: Mixed signal daughter board

.

A.4 The Cross Switch PCB

The cross switch PCB is designed to expand the experimental hardware by adding a possibility to connect any device to provide signals, i.e. data and configuration. The PCB enables the users to look at parallelism. The PCB gives a possibility to let a pair of material samples to be connected and communicate. The cross switch PCB can be used with the Mecobo interface or as an interface card to other equipment. The cross switch PCB can hold up to four material samples, i.e. the board enables multiple material samples (up to four) to operate in parallel. The parallel operation can be configured by setting the cross switch array, i.e. data inputs can be split over several materials or the output of all used materials can be interpreted as a single output result. Block diagram of cross switch PCB is shown in figure A.6.

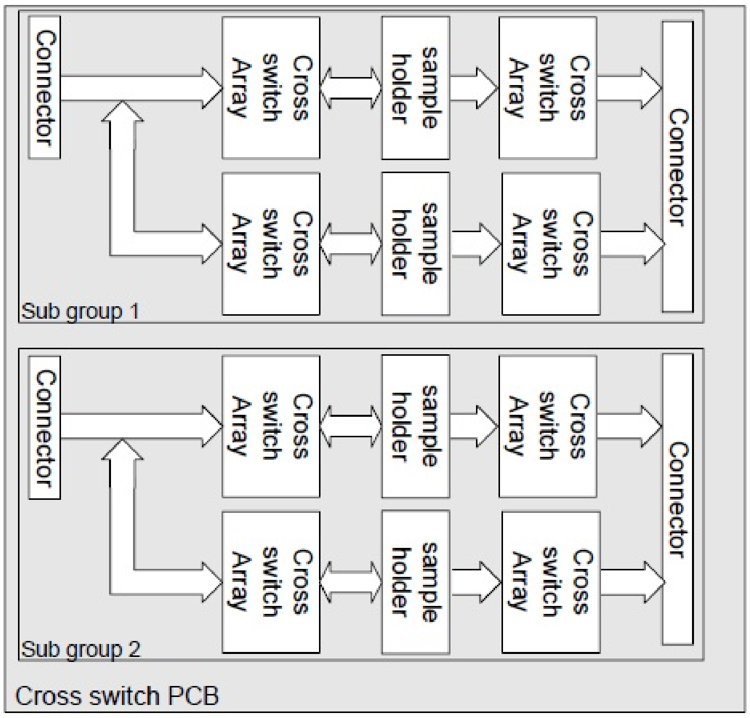


FIGURE A.6: Block diagram of cross switch PCB

Appendix B

The Interface Software

The interface software [Lykkebø*et al.* (2014)] works with a client application and a control software. The control software implements the application programming interface (API) as a Thrift server[1]. Thrift is a technology maintained by Apache which is designed to allow applications written in different programming languages, running on different operating systems and on different computers to communicate with each other. Thrift provides a language that is used to define the functionality used by the server. This language is then compiled by the Thrift compiler into skeleton code which contains all the functionality needed for a server and accepting connections, but the functional components remain missing. Afterward those functional components are added to complete the server implementation.

On the client side, the interface is compiled by Thrift into a library which exposes all the methods in the API. Thrift is able to generate the client and server codes in many programming languages, such as C++, C#, Java and so on. The client library is then connected to server through shared memory if the client and server are both in the same PC and through TCP if the client and server are in different PCs. Client applications only need to implement their functionality. As the communication between the Thrift server and client applications can be based on TCP, it is not necessary for all components to run on the same computer, they can operate over internet. The API and Thrift language are object-oriented, with the objects behaving the same for each client. This object is designed to store signals that are applied to or recorded from the material

[1]https://thrift.apache.org/

under evolution. Functionality is exposed for basic signal processing and statistical measures. Data logging API is intended to provide consistent logging of information.

Figure B.1 shows the complete software architecture of the system. Although only one evolvable motherboard is shown in this figure, but it is possible to add more. Client applications can connect to multiple servers (and Mecobos), and hence can handle a number of systems in parallel.

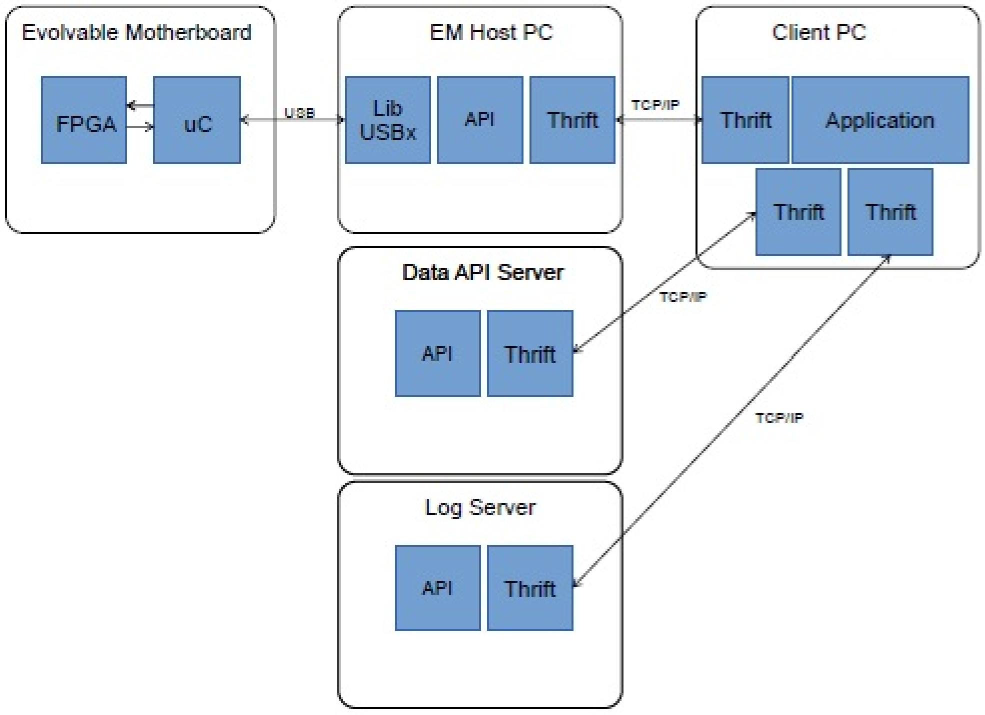


Figure B.1: Overview of the complete software architecture. Here, Mecobo hardware is on the left, the client application is on the right, the main API components are in the middle. Here, the evolutionary algorithm, i.e. user application runs on a client PC, communicating over TCP/IP to the evolvable motherboard host PC. The Mecobo platform is connected to and communicates with the host PC over USB. The log servers communicate with the client PC.

Some sample commands of interface software have been given in following section. Some commands (specially setting up electrodes, i.e. defining signals) are different for Mecobo 3.0 and for Mecobo 3.5. Both types of commands have been given here.

B.1 Commands of Interface Software

Here, an example session of accessing evolvable motherboard through the NASCENCE API is shown using pseudo code[2]. At first the code (after connecting to the Thrift server) pings the board to check whether the connection is alive and resets it. Then, three signals are defined. The signals are defined according to Mecobo 3.0 and Mecobo 3.5 separately. The first one, that stimulates the material with a square wave of amplitude value 1 (3.5 V), frequency 100 Hz, generates 1024 samples to be assigned to Mecobo's first pin. The second one, that stimulates the material with a static voltage (the voltage level is 3.5V for Mecobo 3.0 and the voltage level is 5V for Mecobo 3.5), generates 1024 samples to be assigned to Mecobo's second pin. The third is the signal to be recorded for 1024 samples, and is being recorded from the third pin. The recorded signal is filtered and a fitness is calculated as a sum of squared differences of the filtered recording and a target signal.

```
//Check the board exists
int pingResponse = emEvolvableMotherboard.ping();
//Make it nice and clean
emEvolvableMotherboard.reset();
//Wait until the board is ready
emEvolvableMotherboard.waitUntilReady();
//Get its name
string motherboardID = emEvolvableMotherboard.getMotherboardID();
print Connected to  + motherboardID;

//Definition of three signals: According to Mecobo 3.0
//Define a square wave signal to output
emSequenceItem output0 = new emSequenceItem();
output0.operationType = emSequenceOperationType.PREDEFINED;
output0.pin = 1;
output0.startTime = 0;
output0.endTime = 1024;
output0.frequency = 100;
```

[2]The pseudo code is provided from the NASCENCE project report

```
output0.amplitude = 1;
output0.waveFormType = emWaveFormType.PWM;

//Define a static digital signal to output
emSequenceItem output1 = new emSequenceItem();
output1.operationType = emSequenceOperationType.CONSTANT;
output1.pin = 2;
output1.startTime = 0;
output1.endTime = 1024;
output1.amplitude = 1;

//Define recording a signal
emSequenceItem input0 = new emSequenceItem();
input0.operationType = emSequenceOperationType.RECORD;
input0.pin = 3;
input0.startTime = 0;
input0.endTime = 1024;

//Definition of three signals: According to Mecobo 3.5
//Define a square wave signal to output
emSequenceItem output0 = new emSequenceItem();
output0.operationType = emSequenceOperationType.DIGITAL;
output0.pin = new List<int>();
output0.pin.add(1);
output0.startTime = 0;
output0.endTime = 1024;
output0.frequency = 100;
output0.amplitude = 1;
output0.waveFormType = emWaveFormType.PWM;

//Define a static analogue signal to output
emSequenceItem output1 = new emSequenceItem();
output1.operationType = emSequenceOperationType.CONSTANT;
output1.pin = new List<int>();
```

```
output1.pin.add(2);
output1.startTime = 0;
output1.endTime = 1024;
output1.amplitude = 255;

//Define recording a signal
emSequenceItem input0 = new emSequenceItem();
input0.operationType = emSequenceOperationType.RECORD;
input0.pin = new List<int>();
input0.pin.add(3);
input0.startTime = 0;
input0.endTime = 1024;

//Next download the instructions
emEvolvableMotherboard.clearSequences();
emEvolvableMotherboard.appendSequenceAction(output0);
emEvolvableMotherboard.appendSequenceAction(output1);
emEvolvableMotherboard.appendSequenceAction(input0);
//Run the instructions
emEvolvableMotherboard.runSequences();
//Wait until they are finished
emEvolvableMotherboard.joinSequence();
//Retrieve the sample data for analysis
emWaveForm rawdata = emEvolvableMotherboard.getRecording(input0.pin);
//Push the data to the data processing API
string emOutputData = emDataApi.setBuffer(rawdata);
//Load a waveform to compare
string expectedData = emDataApi.loadBuffer(target.values);
//Do some signal processing to remove noise
emOutputData = emDataApi.medianFilter(emOutputData, 5);
//Compare the target to the actual data
double error = emDataApi.sumSquaredDifference(emOutputData, exepectedData);
//Use this score for fitness
print Fitness score =  + error;
```

Appendix C

Function Optimisation Benchmarks

The function optimisation benchmarks that were used in this thesis are listed below. The dimensions, d and the intervals over which the benchmark optimization functions are defined are presented in Table C.1. It should be noted that the optima of these benchmarks are given in Table 6.7 of Chapter 6.

$f_1(x) = \sum_{i=1}^{d} {x_i}^2$

$f_2(x) = \sum_{i=1}^{d} |x_i| + \prod_{i=1}^{d} |x_i|$

$f_3(x) = \sum_{i=1}^{d} (\sum_{j=1}^{i} x_j)^2$

$f_4(x) = max_i\{|x_i|, 1 \leq i < d\}$

$f_5(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$

$f_6(x) = \sum_{i=1}^{d} (\lfloor x_i + 0.5 \rfloor)^2$

$f_7(x) = \sum_{i=1}^{d} i x_i^4 + random[0, 1)$

$f_8(x) = \sum_{i=1}^{d} -x_i \sin(\sqrt{|x_i|})$

$f_9(x) = \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i) + 10]$

$f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - \exp(\frac{1}{d}\sum_{i=1}^{d} \cos(2\pi x_i)) + 20 + e$

$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 + \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$

$f_{12}(x) = \frac{\pi}{d}\left\{10\sin^2(\pi y_i) + \sum_{i=1}^{d-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]\right\}$

$+\frac{\pi}{d}\left\{(y_d - 1)^2\right\} + \sum_{i=1}^{d} u(x_i, 10, 100, 4)\ ;\ y_i = 1 + \frac{1}{4}(x_i + 1)$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

$f_{13}(x) = \frac{1}{10}\left\{\sin^2(3\pi x_i) + \sum_{i=1}^{d-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]\right\}$
$+ \frac{1}{10}(x_d-1)[1+\sin^2(3\pi x_d)] + \sum_{i=1}^{d} u(x_i, 5, 100, 4)$

$$u(x_i, a, k, m) = \begin{cases} k(x_i-a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i-a)^m, & x_i < -a \end{cases}$$

$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}\right]^{-1}$ where

$a_{1j} = 16(j-3-5\lfloor j/5.1\rfloor) \quad a_{2j} = 16(\lfloor j/5.1\rfloor - 2)$

$f_{15}(x) = \sum_{i=1}^{i=11}\left[a_i - \frac{x_1(b_i^2+b_ix_2)}{b_i^2+b_ix_3+x_4}\right]^2$
where,

$a_i =$(0.1957, 0.1947, 0.1735, 0.16, 0.0844, 0.0627,
0.0456, 0.0342, 0.0323, 0.0235, 0.0246)

$b_i =$(1/0.25, 1/0.5, 1, 1/2, 1/4, 1/6, 1/8, 1/10, 1/12, 1/14, 1/16)

$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 - \frac{5}{\pi}x_1 - 6\right)^2 + 10(1-\frac{1}{8\pi})\cos x_1 + 10$
$f_{18}(x) =$
$\left[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right] \times$
$\left[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right]$

$f_{19}(x) = -\sum_{i=1}^{4} c_i exp\left[-\sum_{j=1}^{d} a_{ij}\,(x_j - p_{ij})\right]$

$$\text{with } a_{ij} = \begin{pmatrix} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{pmatrix}$$

$$p_{ij} = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}$$

$$c_i = \begin{pmatrix} 1.0 & 1.2 & 3.0 & 3.2 \end{pmatrix}$$

$f_{20}(x) = -\sum_{i=1}^{4} c_i exp\left[-\sum_{j=1}^{d} a_{ij}\left(x_j - p_{ij}\right)\right]$

with $a_{ij} = \begin{pmatrix} 10.0 & 3.0 & 17.0 & 3.5 & 1.7 & 8.0 \\ 0.05 & 10.0 & 17.0 & 0.1 & 8.0 & 14.0 \\ 3.0 & 3.5 & 1.7 & 10.0 & 17.0 & 8.0 \\ 17.0 & 8.0 & 0.05 & 10.0 & 0.1 & 14.0 \end{pmatrix}$

$$p_{ij} = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1415 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

$$c_i = \begin{pmatrix} 1.0 & 1.2 & 3.0 & 3.2 \end{pmatrix}$$

$f_{21}(x), f_{22}(x), f_{23}(x) = -\sum_{i=1}^{m}\left[(x - a_i)(x - a_i)^T + c_i\right]^{-1}$

with m =5,7,10 for f_{21}, f_{22} and f_{23}, respectively, and,

$$c_i = \begin{pmatrix} 0.1 & 0.2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.3 & 0.7 & 0.5 & 0.5 \end{pmatrix}$$

$$a_{ij} = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}$$

Table C.1: Benchmark optimization functions: dimensions and intervals.

f_i	Interval	f_i	Interval	f_i	Interval	f_i	Interval
1	$[-5.12,5.12]^{30}$	2	$[-10,10]^{30}$	3	$[-100,100]^{30}$	4	$[-100,100]^{30}$
5	$[-30,30]^{30}$	6	$[-100,100]^{30}$	7	$[-1.28,1.28]^{30}$	8	$[-500,500]^{30}$
9	$[-5.12,5.12]^{30}$	10	$[-32,32]^{30}$	11	$[-600,600]^{30}$	12	$[-50,50]^{30}$
13	$[-50.0,50.0]^{30}$	14	$[-65.536,65.536]^{2}$	15	$[-5,5]^{4}$	16	$[-5,5]^{2}$
17	$[-5,15]^{2}$	18	$[-2,2]^{2}$	19	$[0.0,1.0]^{3}$	20	$[0.0,1.0]^{6}$
21	$[0,10]^{4}$	22	$[0,10]^{4}$	23	$[0,10]^{4}$		

Abbreviations

ADC	**A**nalogue-to-**D**igital **C**onverter
API	**A**pplication **P**rogramming **I**nterface
BZ	**B**elousov-**Z**habotinsky
DAC	**D**igital-to-**A**nalogue **C**onverter
DE	**D**ifferential **E**volution
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
KS	**K**olmogorov-**S**mirnov
LCD	**L**iquid **C**rystal **D**isplay
MCC	**M**atthews **C**orrelation **C**oefficient
NASCENCE	**NA**no**SC**ale **E**ngineering for **N**ovel **C**omputation using **E**volution
NEWS	**N**orth, **E**ast, **W**est, **S**outh
PBMA	**P**oly **B**utyl **M**eth**a**crylate
PMMA	**P**oly**m**ethyl **M**eth**a**crylate
PSO	**P**article **S**warm **O**ptimisation
PWM	**P**ulse **W**idth **M**odulation

Bibliography

Adamatzky, A. (2009). Reaction-diffusion computing. In Meyers, R. A., editor, *Encyclopedia of Complexity and Systems Science*, pages 7548–7565. Springer.

Akbarzadeh, V., Sadeghian, A., and dos Santos, M. V. (2008). Derivation of relational fuzzy classification rules using evolutionary computation. In *IEEE Int. Conf. on Fuzzy Systems*, pages 1689–1693.

Alvim, A., Ribeiro, C., Glover, F., and Aloise, D. (2002). A hybrid improvement heuristic for the one-dimensional bin packing problem. Working Paper, Catholic University of Rio de Janeiro, Brazil.

Ashby, W. R. (1960). Design for a brain:the origin of adaptive behavior. Chapman & Hall.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1st edition.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. F., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics*, 16:412–424.

Banzhaf, W., Beslon, G., Christensen, S., Foster, J., Képès, F., Lefort, V., Miller, J. F., Radman, M., and Ramsden, J. (2006). Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7:729–735.

Barricelli, N. (1954). Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68.

Bechmann, M., Sebald, A., and Stepney, S. (2010). From binary to continuous gates - and back again. In *Proc. 9th international conference on Evolvable systems: from biology to hardware*, pages 335–347.

Beer, S. (1962). A progress note on research into a cybernetic analogue of fabric. In Harnden, R. and Leonard, A., editors, *How Many Grapes Went into the Wine? Stafford Beer on the Art and Science of Holistic Management.*

Broersma, H., Gomez, F., Miller, J. F., Petty, M., and Tufte, G. (2012). Nascence project: Nanoscale engineering for novel computation using evolution. *International Journal of Unconventional Computing*, 8(4):313–317.

Bull, L., Budd, A., Stone, C., Uroukov, I., de Lacy Costello, B., and Adamatzky, A. (2008). Towards unconventional computing through simulated evolution: Control of nonlinear media by a learning classifier system. *Artificial Life*, 14:203–222.

Campbell, D. T. (1956a). Adaptive behavior from random response. *Behavioral Science*, 1:105–110.

Campbell, D. T. (1956b). Perception as substitute trial and error. *Psychological Review*, 63:330–342.

Campbell, D. T. (1960). Blind variation and selective survval as a general strategy in knowledge-processes. *Self-Organizing Systems*, pages 205–231.

Cariani, P. (1993). To evolve an ear: epistemological implications of Gordon Pask's electrochemical devices. *Systems Research*, 10(3):19–33.

Cendrowska, J. (1987). Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27:349–370.

Clegg, K., Miller, J. F., Massey, M. K., and Petty, M. C. (2014a). Practical issues for configuring carbon nanotube composite materials for computation. In *2014 IEEE International Conference on Evolvable Systems (ICES)*, pages 61–68.

Clegg, K. D., Miller, J. F., Massey, M. K., and Petty, M. C. (2014b). Travelling salesman problem solved 'in materio' by evolved carbon nanotube device. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Proceedings*, volume 8672 of *LNCS*, pages 692–701. Springer.

Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S. (1997). Approximation algorithms for bin packing: A survey. In Hochbaum, D. S., editor, *Approximation Algorithms for NP-hard Problems*, pages 46–93. PWS Publishing Co.

Conrad, M. (1988). The price of programmability. In Herken, R., editor, *The Universal Turing Machine A Half-century Survey*, pages 285–307. Oxford University Press, Inc.

Darwin, C. (1859). *On the origin of species by means of natural selection or the preservation of favoured races in the struggle for life.* John Murray.

Davis, L. D. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm Intelligence.* Morgan Kaufmann.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing.* Springer.

Eshelman, L. and Schaffer, J. (1992). Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms*, pages 187–202. Morgan Kaufmann.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7(2):179–188.

Fleszar, K. and Hindi, K. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29:821–839.

Fogel, D. B. (1998). *Evolutionary Computation: The Fossil Record.* Wiley-IEEE Press.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10:484–491.

Friedman, G. J. (1956). Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, University of California, Los Angeles.

Friedman, G. J. (1959). Digital simulation of an evolutionary process. *General Systems: Yearbook of the Society for General Systems Research*, 4:171–184.

Greenwood, G. W. and Tyrrell, A. M. (2006). *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems.* Wiley-IEEE Press.

Haddow, P. C. and Tyrrell, A. M. (2011). Challenges of evolvable hardware: Past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12(3):183–215.

Harding, S. (2006). *Evolution in materio.* PhD thesis, University of York.

Harding, S., Graziano, V., Leitner, J., and Schmidhuber, J. (2012). MT-CGP: mixed type Cartesian genetic programming. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 751–758. ACM.

Harding, S. and Miller, J. F. (2004a). Evolution in materio: A tone discriminator in liquid crystal. In *Proceedings of the Congress on Evolutionary Computation 2004 (CEC'2004)*, volume 2, pages 1800–1807.

Harding, S. and Miller, J. F. (2004b). Evolution in materio: initial experiments with liquid crystal. In *Proc. 2004 NASA/DoD Conference on Evolvable Hardware.*, pages 298–305.

Harding, S. and Miller, J. F. (2005). Evolution in materio : A real time robot controller in liquid crystal. In *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pages 229–238.

Harding, S., Miller, J. F., and Banzhaf, W. (2007). Self-modifying Cartesian Genetic Programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1021–1028. ACM Press.

Harding, S., Miller, J. F., and Banzhaf, W. (2009). Self modifying Cartesian Genetic Programming: Parity. In *2009 IEEE Congress on Evolutionary Computation*, pages 285–292. IEEE Press.

Harding, S. L. and Miller, J. F. (2007). Evolution in materio: Evolving logic gates in liquid crystal. *International Journal of Unconventional Computing*, 3(4):243–257.

Hilder, J. (2014). Pi swarm system manual. `http://www.york.ac.uk/media/robots-lab/PiSwarm-v091.pdf`. Accessed: 01-06-2014.

Hilder, J., Naylor, R., Rizihs, A., Franks, D., and Timmis, J. (2014). The pi swarm: A low-cost platform for swarm robotics research and education. In *Advances in Autonomous Robotics Systems*, volume 8717 of *Lecture Notes in Computer Science*, pages 151–162. Springer.

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

Hollander, M. and Wolfe, D. (1973). *Nonparametric statistical methods.* Wiley.

Horowitz, E., Sahni, S., and Rajasekaran, S. (2007). *Computer Algorithms.* Silicon Press.

Kang, S. J., Kocabas, C., Ozel, T., Shim, M., Pimparkar, N., Alam, M. A., Rotkin, S. V., and Rogers, J. A. (2007). High performance electronics based on dense, perfectly aligned arrays of single walled carbon nanotubes. *Nature Nanotechnology*, 2(4):230–236.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press.

Kotsialos, A., Massey, M. K., Qaiser, F., Zeze, D. A., Pearson, C., and Petty, M. C. (2014). Logic gate and circuit training on randomly dispersed carbon nanotubes. *International journal of unconventional computing.*, 10(5-6):473–497.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

Layzell, P. (1998). A new research tool for intrinsic hardware evolution. In *Proceedings of The Second International Conference on Evolvable Systems: From Biology to Hardware*, volume 1478, pages 47–56.

Layzell, P. (2001). *Hardware Evolution: On the Nature of Artificial Evolved Electronic Circuits.* PhD thesis, University of Sussex.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`. University of California, Irvine, School of Information and Computer Sciences. Accessed: 01-10-2013.

Lykkebø, O. R., Tufte, G., Harding, S. L., and Miller, J. F. (2014). Mecobo: A hardware and software platform for in materio evolution. In *Proc. Conf. on Unconventional Computation and Natural Computation*, pages 267–279. Springer International Publishing.

Mahdavi, S. H. and Bentley, P. (2003). Evolving motion of robots with muscles. In *Applications of Evolutionary Computing*, volume 2611, pages 651–660. Springer Berlin Heidelberg.

Massey, M. K., Kotsialos, A., Qaiser, F., Zeze, D. A., Pearson, C., Volpati, D., Bowen, L., and Petty, M. C. (2015a). Computing with carbon nanotubes: Optimization of threshold logic gates using disordered nanotube/polymer composites. *Journal of Applied Physics*, 117(13).

Massey, M. K., Volpati, D., Qaiser, F., Kotsialos, A., Pearson, C., Zeze, D. A., and Petty, M. C. (2015b). Alignment of liquid crystal/carbon nanotube dispersions for application in unconventional computing. *AIP Conference Proceedings*, 1648.

McEuen, P., Fuhrer, M. S., and Park, H. (2002). Single-walled carbon nanotube electronics. *IEEE Transactions on Nanotechnology*, 1(1):78–85.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structure = Evolution Programs.* Springer.

Michel, O. (1996). Khepera simulator version 2.0 user manual. `http://www2.deec.uc.pt/~pm/RM/manual.pdf`. Accessed: 01-05-2014.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors (1994). *Machine Learning, Neural and Statistical Classification.*

Miller, J. F. (1999). An empirical study of the efficiency of learning Boolean functions using a Cartesian Genetic Programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142. Morgan Kaufmann.

Miller, J. F., editor (2011). *Cartesian Genetic Programming.* Springer.

Miller, J. F. and Downing, K. (2002). Evolution in materio: Looking beyond the silicon box. In *The 2002 NASA/DoD Conference on Evolvable Hardware*, pages 167 – 176. IEEE.

Miller, J. F., Harding, S. L., and Tufte, G. (2014). Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7:49–67.

Miller, J. F. and Mohid, M. (2013). Function optimization using Cartesian Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 147–148.

Miller, J. F. and Thompson, P. (2000). Cartesian genetic programming. In *Proc. European Conference on Genetic Programming, LNCS*, volume 1802, pages 121–132. Springer Berlin Heidelberg.

Miller, J. F., Thomson, P., and Fogarty, T. (1997). Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. In *D. Quagliarella, J. Periaux, C. Poloni, G. Winter (eds.) Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications.* Wiley.

Mills, J. W. (1995a). The continuous retina: Image processing with a single sensor artificial neural field network. Technical report. TR443, Department of Computer Science, University of Indiana.

Mills, J. W. (1995b). Polymer processors. Technical report. TR580, Department of Computer Science, University of Indiana.

Mills, J. W. (1995c). Programmable VLSI extended analog computer for cyclotron beam control. Technical report. TR441, Department of Computer Science, University of Indiana.

Mills, J. W., Beavers, M. G., and Daffinger, C. A. (1990). Lukasiewicz logic arrays. In *20th International Symposium on Multiple-Valued Logic*, pages 4–10.

Mills, J. W., Parker, M., Himebaugh, B., Shue, C., Kopecky, B., and Weilemann, C. (2006). "empty space" computes: the evolution of an unconventional supercomputer. In *3rd conference on Computing frontiers*, pages 115–126. ACM.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* MIT Press.

Mohid, M. and Miller, J. F. (2015a). Evolving robot controllers using carbon nanotubes. In *European Conference On Artificial Life.* Springer. In Press.

Mohid, M. and Miller, J. F. (2015b). Evolving solutions to computational problems using carbon nanotubes. *International journal of unconventional computing.* In Press.

Mohid, M. and Miller, J. F. (2015c). Solving even parity problems using carbon nanotubes. In *15th UK Workshop on Computational Intelligence (UKCI).* In Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014a). Evolution-in-materio: A frequency classifier using materials. In *Proceedings of the 2014 IEEE International Conference on Evolvable Systems (ICES): From Biology to Hardware.*, pages 46–53. IEEE Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014b). Evolution-in-materio: Solving bin packing problems using materials. In *Proceedings of the 2014 IEEE International Conference on Evolvable Systems (ICES): From Biology to Hardware.*, pages 38–45. IEEE Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014c). Evolution-in-materio: Solving function optimization problems using materials. In *14th UK Workshop on Computational Intelligence (UKCI)*, pages 1–8. IEEE Press.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2014d). Evolution-in-materio: Solving machine learning classification problems using materials. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Proceedings*, volume 8672 of *LNCS*, pages 721–730. Springer.

Mohid, M., Miller, J. F., Harding, S. L., Tufte, G., Lykkebø, O. R., Massey, M. K., and Petty, M. C. (2015). Evolution-in-materio: Solving computational problems using carbon nanotube-polymer composites. *Soft Computing*. In Press.

Oltean, M. (2006). Switchable glass: a possible medium for evolvable hardware. In *First NASA/ESA Conference on Adaptive Hardware and Systems*, pages 81–87.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*.

Price, K., Storn, R., and Lampinen, J. A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer.

Rechenberg, I. (1971). *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis. Reprinted by Fromman-Holzboog (1973).

Roselló-Merino, M., Bechmann, M., Sebald, A., and Stepney, S. (2010). Classical computing in nuclear magnetic resonance. *International Journal of Unconventional Computing*, 6:163–195.

Schoenfield, J. (2002). Fast, exact solution of open bin packing problems without linear programming. Working Paper.

Scholl, A. and Klein, R. (2003). Bin packing. `http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm`. Accessed: 01-10-2013.

Scholl, A., Klein, R., and Jürgens, C. (1997). BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24:627 – 645.

Schwefel, H. (1974). *Numerische Optimierung von Computer-Modellen Mittels Der Evolutionsstrategie: Mit Einer Vergleichenden Einführung in Die Hill-Climbing- Und Zufallsstrategie*. PhD thesis. Reprinted by Birkhäuser in 1977.

Schwerin, P. and Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4:377–389.

Stewart, R. M. (1969). Electrochemically active field-trainable pattern recognition systems. *IEEE Transactions on Systems Science and Cybernetics*, 5:230–237.

Storn, R. and Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, pages 341–359.

Theis, M., Gazzola, G., Forlin, M., Poli, I., Hanczyc, M. M., and Bedau, M. (2006). Optimal formulation of complex chemical systems with a genetic algorithm. In *European Conference on Complex Systems (ECCS 06)*, page 193.

Thompson, A. (1998). *Hardware Evolution - Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer-Verlag.

Thompson, A. and Layzell, P. (1999). Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79.

Toth, R., Stone, C., Adamatzky, A., de Lacy Costello, B., and Bull, L. (2008). Dynamic control and information processing in the Belousov-Zhabotinsky reaction using a coevolutionary algorithm. *Journal of Chemical Physics*, 129:184,708.

Toth, R., Stone, C., de Lacy Costello, B., Adamatzky, A., and Bull, L. (2011). Simple collision-based chemical logic gates with adaptive computing. In *Theoretical and Technological Advancements in Nanotechnology and Molecular Computation*, pages 162–175.

Tyrrell, A. M., Krohling, R. A., and Zhou, Y. (2004). A new evolutionary algorithm for the promotion of evolvable hardware. *IEE Proceedings of Computers and Digital Techniques*, 151(4):267–275.

Vargha, A. and Delaney, H. D. (2000). A critique and improvement of the "CL" common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132.

Vesterstrom, J. and Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Congress on Evolutionary Computation, 2004. CEC2004.*, volume 2, pages 1980 – 1987.

Völk, K., Miller, J. F., and Smith, S. L. (2009). Multiple network CGP for the classification of Mammograms. In *Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, EvoWorkshops '09, pages 405–413. Springer-Verlag.

Volpati, D., Massey, M. K., Johnson, D. W., Kotsialos, A., Qaiser, F., Pearson, C., Coleman, K. S., Tiburzi, G., Zeze, D. A., and Petty, M. C. (2015). Exploring the alignment of carbon nanotubes dispersed in a liquid crystal matrix using coplanar electrodes. *Journal of Applied Physics*, 117(12).

Walter, W. G. (1953). The living brain. Gerald Duckworth & Co. LTD.

Yao, X. and Liu, Y. (1996). Fast evolutionary programming. In *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press.