

Bangor University

DOCTOR OF PHILOSOPHY

An Integrated Low-Cost Agent-Based Environment for Simulating And Validating the Design of Interesting Behaviours for Robots

Aslam, Syed Kaleem

Award date:
2022

Awarding institution:
Bangor University

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Bangor University

DOCTOR OF PHILOSOPHY

An Integrated Low-Cost Agent-Based Environment for Simulating And Validating the Design of Interesting Behaviours for Robots

Aslam, Syed Kaleem

Award date:
2022

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.



PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

**An Integrated Low-Cost Agent-Based
Environment for Simulating And
Validating the Design of Interesting
Behaviours for Robots**

Syed Kaleem Aslam

Submitted in partial satisfaction of the requirements for the
Degree of Doctor of Philosophy
in Computer Science

Supervisor Dr. William J. Teahan

12 May 2022

Acknowledgements

“ *People are fascinated by robots because they are machines that can mimic life.*

— **Colin Angle**

First and foremost, praises and thanks to God, the Almighty, for His showers of blessings throughout my research work.

I would like to express my sincere gratitude to my supervisor, Dr. William J. Teahan, for guiding me on every single point of my research. THANK YOU for your continuous feedback and suggestions, without your guidance it was not possible to complete this research.

I would like to thank my whole family for encouraging me and taking care of me emotionally and financially. There were many occasions when I was about to give up, but your support always kept me going no matter what. Unfortunately, our mother passed away three months after the start of my PhD, you all were there to fulfil that unfillable gap. I love and thank you all for everything you have done for me.

I would like to thank my colleagues and friends: Ada, Ali Haider, Ali Khan, Alley, Anantha, Andi, Angel Tan, Asad, Azeer, Bing Yan, Diti, Dushyant, Faheem, Guru, Haider, Harshal, Jay Zupudi, Joe, Kalim, Khizar, Kiran, Lamiae, Marcel, Maria, Mubashir, Nadeem Rehman, Nadim, Neil Williams, Nivas, Rafia, Ravi, Rishab, Sahil, Sajid Ali, Shainece, Sidharth, Songkun Ji, Taimoor, and Yasharth for conversations regarding studies, dinner parties, cultural events, society events, visiting nearby places and everything else to keep me going

during my research by keeping my mind fresh. The best night was when we captured Bio-luminescent plankton at Penmon point.

Finally, I would like to thank all those students and friends, I have not named, who came to Bangor University and left after completing their studies, but their memories are still in my heart. Some of you encouraged me to stay strong during my difficult times, especially when I was hospitalised and convinced me to finish what I have started. I thank you all.

I dedicate this thesis

to my parent, who saw the dream of providing higher education for their son.

Although they are no longer of this world, their dream has come true;

to my elder brother who made that dream come true;

to my whole family for their prayers and love.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Syed Kaleem Aslam

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, the British Library ETHOS system, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Syed Kaleem Aslam

Abstract

Designing and developing autonomous robotics systems is quite a challenging and time-consuming task. Besides that, current robotic systems are expensive and mostly designed to perform a single task. Developing software environments to aid the design of low-cost robotic systems will open up development to people and parties which are less connected to the field of robotics.

This thesis proposes a robotic system design environment that is user-friendly, low-cost, and can be used to find solutions for several different tasks. The robotic design system that has been developed controls a Lego Mindstorms robot via a NetLogo model. It makes and enacts decisions by using robot commands and real-time sensor feeds such as demonstrated in NetLogo models (line-following and subsumption architecture roaming). The results have shown that the robotic system design environment is capable of sending commands to robots and getting real-time feed-backs from sensors.

This environment has also been modified to extend its capabilities to include evolutionary techniques using Grammatical Evolution. The code for the NetLogo application was evolved grammatically using simulated agents that were embodied within a virtual environment for the task of maze exploration. The approach was evaluated using a selection of mazes with a robot inserted into these unknown environments without any internal memory mechanisms. The solutions found during virtual experiments were extracted to a Lego Mindstorms robot and validated in the real world.

The robotic system design environment was further developed to discover and validate novel interesting behaviours. A combination of configurations and grammar used for these experiments resulted in producing several unusual

behaviours. The behaviours were evolved by using Grammatical Evolution and a novel compression-based metric was used as a fitness function to effectively discover interesting behaviour. These behaviours were validated by real-world experiments by comparing them against the simulated results.

In summary, the research undertaken in this thesis developed a novel robotic system design environment that is user-friendly, low-cost, and capable of performing several different tasks involving complex robotic behaviours such as maze exploration, and was able to discover and validate novel interesting behaviour.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Statement	2
1.3	Research Questions	3
1.4	Objectives	3
1.5	Contributions	4
1.6	Publications	5
1.7	Thesis Outline	5
2	Background and Related Work	7
2.1	Robotic System Design Environments	7
2.1.1	Introduction	7
2.1.2	Popular approaches to Robotics	8
2.2	Issues with Robotics Design	12
2.2.1	What is a Robot?	12
2.2.2	Design Issues for Robotics Design Environments	13
2.2.2.1	Design, Fabrication and Programming	13
2.2.2.2	Communication	14
2.2.2.3	Computation	14
2.2.2.4	Biohybrid and Bioinspired robots	14
2.2.2.5	Battery Power	15
2.2.2.6	Navigation and Exploration	15
2.2.2.7	Providing heterogeneity abstractions	16
2.2.2.8	Offering often-needed robot services	16
2.2.2.9	Limitation of component resources	16
2.3	Design Environments for Robotics	16
2.3.1	Player/Stage System	18
2.3.2	Orca	19
2.3.3	Robot Operating System (ROS)	19
2.3.4	Webots™	20
2.3.5	Open Robot Interface for the Network (ORiN)	20
2.3.6	Sensory Data Processing Middleware	20
2.3.7	Data-Centric Middleware	21
2.3.8	LEaRN – LEgo Robot and Netlogo	21

2.4	Evolutionary Robotics	23
2.4.1	Evolutionary Algorithms	24
2.4.2	Genetic Algorithms	25
2.4.3	Genetic Programming	26
2.4.4	Grammatical Evolution	27
2.4.4.1	Backus-Naur-Form (BNF)	29
2.4.4.2	Mapping Function	29
2.4.5	Current and Past Research	30
2.4.6	Implementations of Grammatical Evolution	31
2.4.6.1	Grammatical Evolution in Java (GEVA)	32
2.4.6.2	libGE	32
2.4.6.3	Java Grammatical Evolution (jGE)	33
2.4.7	Performance Evaluation of Grammatical Evolution	34
2.4.7.1	Artificial Ant Problems	34
2.4.7.2	Maze Searching Problems	37
2.5	Discovery and Validation of Interesting Robotic Behaviours	39
2.5.1	Behaviour-based Robotics	40
2.5.2	Interestingness	41
2.5.3	Discovering Interesting Robotic Behaviours	43
2.5.3.1	Visual Memorability for Robotic Interestingness	44
2.5.3.2	Discovering Interesting Behaviours in Complex Systems	45
2.6	Summary and Discussion	46
3	NXTLogo – A new Robotic Design Environment	50
3.1	Introduction	51
3.2	Background	52
3.2.1	Related Work	53
3.2.2	Lego Mindstorms Robotics	54
3.2.3	NetLogo	54
3.2.4	LeJOS	54
3.3	NXTLogo: Design and implementation	55
3.3.1	The Control File	56
3.3.2	The NetLogo Extension File, NXTLogo	58
3.3.3	Implementation and Class Breakdown	60
3.4	Sample NetLogo Models	61
3.4.1	Line Following	61
3.4.2	Subsumption Architecture Roaming	63
3.5	Results	65
3.5.1	Line Following	65
3.5.2	Subsumption Architecture Roaming	66
3.6	Conclusions	67

4	Evolving Robotic Behaviours using Grammatical Evolution	69
4.1	Introduction	70
4.2	Design of Lego Mindstorms Robot	71
4.2.1	Ultrasonic Sensor	72
4.2.2	Bump Sensor	73
4.2.3	RGB Colour Sensor	73
4.2.4	Motors	73
4.3	NetLogo Application for Evolving Solutions	73
4.3.1	Application Interface	74
4.3.2	The environments	75
4.3.3	The fitness function	76
4.3.4	BNF rules	76
4.3.5	Testing	77
4.4	An Interface between NetLogo and NXT	80
4.5	Mindstorms Control File	81
4.6	Real-World Implementation	82
4.6.1	The Environment	82
4.6.2	Three Experiments	83
4.6.3	Summary	84
4.7	Evaluation	85
4.7.1	Evolution & grammar	85
4.7.2	Experimental Observations	86
4.7.3	Robot's Design Improvements	87
4.8	Summary and Conclusion	87
5	Discovery and Validation of Novel Interesting Behaviours	89
5.1	Introduction	89
5.2	Experimental Setup for Discovering Interesting Behaviours	90
5.2.1	Design of Robot	91
5.2.1.1	Colour Sensor	92
5.2.1.2	Motors	92
5.2.1.3	Ultrasonic Sensor	92
5.2.1.4	Bump Sensors	93
5.2.2	The Environment	93
5.2.3	The BNF Grammar	94
5.2.4	Fitness Function	95
5.3	The Experiments	97
5.3.1	Virtual Experiments	97
5.3.1.1	Robotic Behaviours	98
5.3.1.2	Interesting Behaviours	100
5.3.1.3	iFinder Discovery	104
5.3.1.4	Experimental Setup and Results of Experiments	106

5.3.2	Real-World Experiments	111
5.3.2.1	Colour Sensing	111
5.3.2.2	Results and Discussion	113
5.3.2.3	Turn Angles	115
5.3.2.4	Results and Discussion	116
5.3.2.5	iFinder Validation	118
5.3.2.6	Results and Discussion	120
5.4	Summary	122
6	Conclusion and Future Work	124
6.1	Revisiting the Research Questions	124
6.2	Revisiting the Thesis Objectives	125
6.2.1	First Objective	126
6.2.2	Second objective	126
6.2.3	Third objective	127
6.2.4	Fourth objective	127
6.3	Limitations	128
6.4	Future Work	130
	References	131
	Appendices	149
A	Videos of Experiments	150

List of Figures

2.1	A generic approach to the use of middleware for robotic systems [39].	17
2.2	Flowchart of Genetic Algorithm [52].	25
2.3	(a) Crossover Operator, (b) Mutation Operator [52].	26
2.4	Modular components of grammatical evolution [110].	28
2.5	The components of GEVA v2.0 [110].	32
2.6	The Santa Fe trail problem where green squares represent food and brown squares represent gaps [74].	35
2.7	The grids of the Los Altos Hills trail problem [74].	36
2.8	The Hampton Court Maze, green dot represent the starting point and red dot represent the finishing point. [60].	38
2.9	The Chevening House Maze [169].	39
2.10	Structure of robotic behaviours [124].	40
2.11	A selection of interesting and uninteresting scenes [165].	44
2.12	Visual interestingness of footage and loss of interest in repetitive scenes is indicated by the arrows during online learning [165].	44
2.13	A variety of interesting behaviours was produced by Ahmed’s new variant of the Braitenberg Vehicle [2].	46
3.1	The design for the middleware based on some sample classes in the extension.	56
3.2	The Control File Instruction Buffer.	59
3.3	Successful execution of the line following behaviour.	66
3.4	The roaming ability of the robot is illustrated by it staying on the same colour and avoiding obstacles which are represented by solid colours.	67
4.1	The upgraded design of the bumper car for our low-cost Lego Mindstorms robot “Explorer”.	72
4.2	The interface of the application used to evolve the codes.	74
4.3	The design and layout of the three mazes environments (Maze I, Maze II, and Maze III).	75
4.4	The path taken by the best ‘Maze I, Maze II, and Maze III’ solutions.	78
4.5	The UI of the application that connects between NetLogo and NXT.	81
4.6	Basic Control flow for the NXT control file.	82
4.7	The real-world reconstruction of the Maze I environment.	83

4.8	A robot bump despite the use of the ultrasonic sensor.	86
4.9	A delay between detecting the goal and acknowledging a goal. .	86
5.1	Five Minute Bot design from Lego Projects [120].	91
5.2	The adapted "Five Minute Bot" design used for the final experiments.	92
5.3	A behaviour hand-crafted for the model Robotic-Behaviours. . .	99
5.4	Interface of the model Interesting-Behaviours.	101
5.5	Some of the different behaviours randomly produced by the model "Interesting-Behaviours" as ranked in ascending order by the compression Codelength metric.	103
5.6	The "iFinder-Discovery" application to evolve interesting behaviours	105
5.7	The results achieved for Example 1. Below each simulation output is the codelength fitness value.	109
5.8	The results achieved for Example 2.	110
5.9	The user interface of the Colour Sensing application.	112
5.10	The colour setup used for the new test environment.	113
5.11	Test Environment for the robot's turn angles and new behaviour to test robot's performance after correcting turn angles.	115
5.12	User interface of the Turn-Angles application.	116
5.13	Turn Angle robot behaviour successfully recreated in the real-world	117
5.14	The robot becoming stuck while taking a turn.	118
5.15	The user interface of the "iFinder-Validation" application. . .	119
5.16	The simulation results achieved for Example 1 and Example 2. .	120

List of Tables

2.1	A selection of robots that have popularised robotics development.	10
2.2	A comparison of robotic environments' features.	22
2.3	Properties required by mobile robots to judge the interestingness of scenes [165].	45
3.1	Commands currently available in NXTLogo.	61
3.2	Six procedures of the line following model.	63
3.3	Five procedures of the subsumption roaming model.	64
4.1	The results achieved from 12 test runs.	78
4.2	The configurations used for best results in each experiment. . .	78
4.3	The performance summary of evolved solutions in real-world experiments.	85
5.1	The summary and details of the experiments.	97
5.2	The time taken to generate solution code with different configurations.	107
A.1	The description of the experiments' videos uploaded to YouTube.	151

Chapter 1

Introduction

1.1 Motivation

Autonomous robot systems perform tasks in situations where human control is infeasible or not cost-effective. Robots use many different types of sensors, actuators, and degrees of freedom to run autonomously and provide sophisticated services. Robotic system environments allow you to program robots to perform desired actions. Unfortunately, these robots are tied to specific hardware, processing environments, and communication infrastructures which often makes their services limited to that specific application scenario only. The problem for robotics design is that the functions and constraints related to the task and environments are hard-coded and inaccessible which increases the complexity. This increased complexity of design and cost of the autonomous system has led to an increase in demands for modularity, productivity, re-usability, integration, and maintenance.

Designing and developing robotics environments is quite a challenging and complex task. When it comes to robotics, there is still much research work left to be done. A set of reliable software components integrated into a unified environment can kick start a remarkable robotics project. However, currently available environments are expensive and hard to design/customise. One of the reasons is that a large share of the robotic project's cost is spent on the control software. For example, business automation projects may spend up to 80% of their budget on system integration such as software development and customisation [146]. Educational institutions also need to customise their learning environment frequently to introduce new opportunities to students.

Thus, to reduce the cost of customisation, an important goal for any robotics system environment is to simplify the job of robotics software engineers.

Currently, robotics design and operation requires advanced programming skills for the end users which makes it complex and time-consuming. Therefore, a robotics system environment that is easy to program, install, and compile will open up development opportunities to the people and parties less connected to the field of robotics. For example, a plug and play robotics system environment and drag and drop user interface can take out the complexity and save valuable time. If it takes less time to implement, then more time can be spent investigating alternative designs.

Recently, as is the case in AI research in general, a large number of research projects have explored issues related to robotics design, especially in the evolutionary robotics sub-field [105], [102], [56], [81], [84], [24], [43], [129], [85], [140], [42], [57], [10], [38], [82]. But we still have a long way to go in the production of truly cost-effective and easy to use robotics design environments. While current research has explored how agents can create interesting behaviours and validate them in the real world, we now need to explore other components of the robotics system design environments to make them low-cost and easy to use. To do so, design issues must be considered and a design should be implemented which can be applied to multiple projects with very little customisation. The hope is that this will be cost-effective in terms of time and money and will be useful for businesses who wish to adopt automation in the future.

1.2 Problem Statement

Developing robot system environments with low-cost and easy to use features is a challenge. It is a problem for AI to design a robotic system environment that can be used for multiple projects and not specific to any single project. Current robot systems are specific to certain types of hardware and software and it is expensive to customise them according to new research. One of the main reasons is design issues – currently, robot systems are developed

according to the project requirements and are immensely dependent on a certain type of hardware and software which is only usable for that specific project. However, while there has been significant research undertaken in designing robotics system environments for autonomous robots, low-cost and easy to use environments have received relatively little attention. The main communities that could benefit from this research are small businesses and educational institutions.

Robotics has been used to perform many different tasks in recent years. During some of those tasks, interesting behaviours of robotics were noticed. Is it possible to develop a system which can automatically discover behaviour that is interesting or not? Can we validate these newly discovered behaviours in the real-world? These are very challenging issues while keeping the robotic system environment low-cost and easy to use.

1.3 Research Questions

The specific research questions are as follows:

1. Can a low-cost robotic system design environment be developed that is open-source and extendable for performing evolutionary robotics experiments?
2. Is the new robotic system design environment capable of automatically discovering novel interesting behaviours and validating these behaviours in the real-world?

1.4 Objectives

The specific objectives are as follows:

1. To conduct an extensive literature review on the current state of the art of robotic design and work related to the above research questions and point out current issues, findings, and future directions (see Chapter 2).

2. To develop an integrated low-cost robotic system design environment for evolutionary robotics experiments (see Chapters 3 and 4).
3. To use the new robotic environment to discover novel interesting robotic behaviours (see Chapter 5).
4. To use the new robotic environment to validate novel interesting robotic behaviours in the real world (see Chapter 5).

1.5 Contributions

The work undertaken in this thesis has led to several contributions and multiple publications which are detailed below:

A new low-cost robotic development environment The thesis proposes and implements an up to date robotic system environment which links NetLogo, an agent-oriented integrated development environment, with Lego Mindstorms Robots. This middleware can receive commands from NetLogo and pass signals to the robot, and receive feedback from the robot and pass them back to the NetLogo.

The novel application of Grammatical Evolution for robotic design The thesis applied the solution of Grammatical Evolution [136] to the problem of evolving behaviours for the task of maze exploration. A series of experiments were conducted, and results were evaluated. The results showed that the newly developed environment is capable of finding novel solutions using Grammatical Evolution efficiently. However, there were some shortcomings due to the limitations of the Lego Mindstorms robot hardware.

The discovery of novel interesting behaviours The thesis proposes a novel method for discovering interesting robotic behaviours that utilises the new robotic system design environment. The model used for Grammatical Evolution experiments was further developed to discover the novel interesting robotic behaviours.

The real-world validation of the novel interesting behaviours The new robotic design environment was also used to validate the discovered novel interesting behaviours in the real-world. Also, a new interface was developed to send commands to the robot and receive feedback from the robot. Its performance in the real-world was compared against the simulated results and a detailed analysis is provided in chapter 5.

1.6 Publications

The following papers have been published or accepted for publication during the course of this research:

1. Aslam, S.K, Faithful, W.J. and Teahan, W.J. "A Middleware to Link Lego Mindstorms Robots with 4th Generation Language Software NetLogo". In International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 416-430) 2018, Cambridge UK. Springer.
2. Aslam, S.K, Jones, L. and Teahan, W. J. "A Low-Cost Platform for the Grammatical Evolution of Exploratory Agents". Abstract accepted and talk presented at Artificial Intelligence International Conference (A2IC) 2020, Barcelona, Spain.

1.7 Thesis Outline

Chapter 2 reviews the background and related work to the field of robotic system development environments, evolutionary robotics, and interesting behaviours of robotics. The chapter begins with a survey of current robotic system environments and provides a historical review of the field. The chapter continues with the description of evolutionary robotics and outlines the main challenges in the field. The chapter provides a detailed background of interesting behaviours and some research to discover interesting behaviours using robotics and ends with a summary and discussion. Current issues and future directions are discussed as well. The chapter provides the past and current research of the robotic system environments and evolutionary robotics

with a focus on grammatical evolution and interesting robotic behaviours. The chapter concludes with a summary of past and present research and briefly describes related implementations and applications. The chapter closes with the presentation of the issues and problems which can be mitigated to improve robotic systems.

Chapter 3 presents the design of a middleware that uses a Java extension to NetLogo and LeJOS Java control file. The chapter first discusses the prior knowledge by reviewing the existing middlewares. Then the chapter presents the middleware which can receive commands from NetLogo and pass signals to the robot and receive feedback from the robot and pass them back to the NetLogo. The chapter concludes with some examples of how middleware can be used in NetLogo models and a discussion of some experimental results.

Chapter 4 reports on the experiments that were conducted on the use of Grammatical Evolution applied to the problem of evolving behaviour for the task of maze exploration. A series of NetLogo applications and a Lego Mindstorms kit (to examine the performance in the real-world) was used to investigate the application of Grammatical Evolution to exploratory robotics.

Chapter 5 investigates how to evolve interesting behaviours using the novel robotic system development environment, developed earlier in this thesis. The chapter discusses the discovery of novel interesting robotic behaviours and their validation in the real-world by using the newly developed environment. The chapter concludes with the evaluation of results and a summary of the success and shortcomings of the new environment.

Chapter 6 discusses the success of the thesis by reviewing the aims and objectives. Then the chapter summarises the findings and provides the conclusions. Finally, the chapter discusses the limitations of the thesis before suggesting future work.

Chapter 2

Background and Related Work

This chapter provides background to the area of robotics and identifies design issues for robotics by comparing currently available design environments for robotics. Related work is discussed in detail and the issues which still need to be resolved are identified. These will help us to design and upgrade our robotic system development environment. Some popular areas in Evolutionary Robotics research are discussed and grammatical solutions to the current issues are provided in detail which is later used to test the performance of our robotic system development environment. The notion of interestingness is then discussed, with a focus on robotics design, followed by the current issues and future directions. This has helped guide the development of solutions to automatically finding interesting behaviours in chapter 5.

2.1 Robotic System Design Environments

This section and its subsections provide background and work related to robotic design environments.

2.1.1 Introduction

The advancements in the field of computing, wireless communication, and sensors technologies are revolutionising the emerging field of robotics which is leading to an extraordinary opportunity to develop a variety of real-time applications. These new applications include performing tasks in dangerous and inaccessible terrains, search-and-rescue missions, assisting elders and physically challenged people, and assisting surgeons in operating theatres. In future, robots should be designed to facilitate rapid and easy implementation,

be flexible, maintainable, customisable, self-configuring, self-optimising, and have the ability to interact with other systems such as sensor networks and enterprise information systems [95].

The capabilities of robots are increasing every day whereas the costs of hardware and components are reducing rapidly at the same time. The current situation and progress look optimistic; however, the growth of industry seems delayed because of some factors such as the development environment. Fragmentation of hardware is the most important factor to be considered [73]. Another important factor is the portability of code and development tools are not up to standard [128]. Most robots are programmed to be used in unique environments and can only be used in those specific environments. Because of this reason, the process of robots' adoption in different fields is slow. For example, we can easily find a robot to sweep our home autonomously, and some other special task robots can be found performing specific military tasks. But those robots can only work in the specific environment they are designed for and cannot be programmed to swap duties.

To be truly useful, robots must have a set of basic abilities to adapt and tackle a variety of tasks in a wide range of environments. These abilities may include navigation to a goal location, avoiding obstacles, object recognition, and object manipulation [92]. To find out current issues and possible solutions a detailed literature review related to robotics' history, design and future are presented below.

2.1.2 Popular approaches to Robotics

The history of robotics is as old as the histories of technology and science, and attempts to reproduce an artificial, sentient being. The main reason for interest in robotics is the need to perform those tasks which cannot be performed by a human. Some reasons behind this can be that the environment is hazardous (such as radioactive field, deep-sea level) or it is too far away (such as space exploration). Therefore, researchers are trying to develop new robots every day [27]. A brief history and popular approaches

to robotics are provided below. A table summarising a selection of events that popularised robotics is shown in table 2.1.

Robotics research is not something new. Some historians stated that in Greek mythology Talos was a giant creature made of bronze given to Europe by Zeus [89]. However, Karel Capek used the term “robot” for the first time in his play called “Rossum’s Universal Robots (R.U.R)”. The story was simply that a man created a robot and was later killed by his creation [96]. This may be the reason behind human’s biggest fear that one day robots will take over the world.

Westinghouse created ELECKTRO, a robot human-like robot which was unveiled at the 1939 world’s fair. This robot could walk, talk, and smoke [4]. It was in 1942 when Willard Pollard and Harold Roselund designed the first “programmable” mechanism (a spray-painter) for the Devilbiss Company [45]. In the late 1940s, Elmer and Elsie (turtle robots) were created by Walter, robots that were capable of finding and returning to the charging station whenever the battery ran low [154]. Similar research was conducted between the 1960s and 1970s which led to one of the first mobile robots “Shakey” [151] and the computer-controlled Stanford arm [139]. Shakey was built by SRI which can be controlled by a computer and was equipped with a vision system.

Another similar robot called “Stanford Cart” successfully crossed a room full of obstacles (chairs were used as obstacles) without any assistance from humans. This cart was equipped with a camera to take pictures from several angles which were relayed to a computer. The computer then analysed the pictures and figured out the distance from obstacles [148].

During the late 1980s, Brooks and his team at MIT created robots that defied the cognitive approaches to AI, such as subsumption architecture couples sensory information to choose action rather than using the guiding behaviour by symbolic mental representations of the world. The robots created using these approaches were capable of using sensed information and taking

Table 2.1: A selection of robots that have popularised robotics development.

Robot Name	Features
ELECKTRO [4]	Capable of walking, talking, and smoking.
Paint-Sprayer [45]	First “programmable” mechanism.
Turtle robots [154]	Able to find their way back to the charging station whenever battery power run low.
Shakey [151]	First Mobile robot, equipped with a vision system
Stanford Cart [148]	Crossed a room filled with chairs without any human assistance.
Genghis [78]	Able to follow a person based on the readings of an infrared sensor.
LEGO [73]	LEGO plastic bricks to design/invent robots.
Myrmix [11]	Able to find and collect objects representing food.
iRobot [67]	First robot vacuum cleaner.
Spirit & Sojourner [138]	Mars exploration.
HUBO [59]	One of the smartest mobile robots in the world.
Robonaut2 [35]	First-ever space robot.
Sophia [69]	First ever robot to get citizenship.

action accordingly, such as Genghis could follow a person by utilising the information provided by the infrared sensor [78]. Unlike previous robots, Genghis showed that reactive behaviours could navigate complex, rugged and open environments while being much cheaper than previous robots.

LEGO released Mindstorms NXT in 1998, which is a system for designing/inventing robots using LEGO plastic bricks, to enable developers to design an inexpensive robotic environment [73]. More on this approach will be discussed in Chapter 3.

In 1999, the University of Zurich created a robot called “Myrmix” [11] with three basic principles layered behaviours which are collect, avoid, and safe-forward. The robot was supposed to avoid the objects which represent obstacles while finding the objects representing food. The behaviours were prioritised such as collecting an object, avoiding obstacles, and moving forward respectively.

It was the 2000s when robotics started spreading everywhere and human’s started using them in different aspects of regular life such as iRobot which was the first generation of the Roomba robotic vacuum cleaners that was

introduced in 2002 [67]. The year 2003 saw NASA launching twin robotic rovers called Spirit and Sojourner as part of their Mars exploration mission [138].

The Korean Institute of Science and Technology (KIST) created a robot called “HUBO” and claims it is the world’s smartest mobile robot. The robot is connected to a computer using high-speed wireless connecting and the computer performs all the thinking for the robot [59]. In 2005 self-driving cars made their first appearance; however, substantial improvements are still needed before wide-spread deployment of fully autonomous vehicles is possible.

Robonaut2, the latest generation of astronaut helpers, was launched to space in 2011. Its main purpose, for now, is to teach engineers how dextrous robots behave in space. However, it is expected that one day it could be helping space-walkers in making repairs and additions to the international space station [35].

At the Future Investment Summit in Riyadh 2017, a robot called Sophia stunned the whole world by displaying human-like expressions and interacting with people. It was also featured in many TV interviews and was also granted Saudi Arabian citizenship, becoming the first-ever robot to have a nationality [69].

In summary, over the years the approaches used for designing autonomous systems evolved, however, these systems are either expensive or complex to understand. Substantial time and/or money are required to learn and develop the small selection of robotic systems described above. Therefore a new approach is needed which is cost-effective and user-friendly without the requirement of knowledge of advanced programming.

2.2 Issues with Robotics Design

Robotics is the design and study of robots. It has been long predicted that robots will become mainstream public agents if they acquire basic intelligence. To demonstrate basic intelligence, the robots should have capabilities of navigating, communicating, and interacting in the real-world environment. Generally, these requirements can be fulfilled by currently available robotics hardware, however, this prediction is held back by the complexity of design and implementation of thoughtful intelligence. To identify the issues with robotics, first, it is important to answer the following question “what is a robot”?

2.2.1 What is a Robot?

There are several robots available nowadays. Some of them walk around on their two, four, six, or more legs while others fly. Some help physicians to do surgery, others help workers in industries. These robots can be the size of a coin or bigger than the size of a car. Some robots can make omelettes while others can explore Mars. This diversity in size, design, and capabilities makes it difficult to come up with a definition of what is a robot [51].

Despite this, there are various definitions provided by some scholars. For example, Goris [50] defined that a robot is *“a mechanical device which performs automated tasks, either according to direct human supervision, a predefined program or, a set of general guidelines, using artificial techniques”*. Weser [168] stated that *“the term robot has quite a broad meaning. It can refer to software that crawls web pages, has a manipulator arm, an artificial insect, an autonomous aeroplane and many other technical systems that somehow act more or less purposefully”*. Furthermore, there is a common apprehension that robots are to replace humans in almost every situation, especially in monotonous, heavy, and hazardous processes.

Nowadays, their use in industries is common where robots perform several tasks including welding, drilling, assembling, painting and packaging. These robots perform a fixed sequence of actions and the surrounding space is

designed to suit the robots. In contrast, mobile robots use different types of sensors to interact with their surroundings which helps them to decide their next action. Mobile robots need intelligent navigation systems to ensure efficient and collision-free movement which is still being researched in several projects [50].

In the context of this thesis, a robot is capable of displaying Brook's [22] definition of a behaviour based robot such as the robot does not need an internal model to operate, it utilises the sensors and actuators to act immediately upon real-world events including interaction, complex reaction, and forward planning.

2.2.2 Design Issues for Robotics Design Environments

Some of the important design issues of robotics design environments are discussed below.

2.2.2.1 Design, Fabrication and Programming

The length of time to design, fabricate, and program a new robot is one of the major issues for robotics. The current robotic design process is lengthy and it is also complex to make alterations in existing designs. Similarly, the fabrication of these designs is a lengthy process as well. Developers may need to wait days or months to get a custom part for their robot. These issues can be exacerbated by the limited resources of robots such as computation, mobility, manipulation capabilities, and size. Robots can only perform those tasks for which they have components. For example, if a robot does not have a colour sensor, it cannot recognise colour no matter how many commands you run on the system. Besides that, programming the robot takes a significant amount of time and sometimes after modification in the robot, it may require to be re-programmed from scratch [16].

Some of these procedures can be simplified by using a middle layer of design which should provide advanced developments with simple interfaces to be used by the developers. Software extension and reusability should also be increased by the middleware.

2.2.2.2 Communication

Communication is one of the most complex issues faced by robotics. When robots are working together, a reliable communication method is required for coordination. Regardless of advances in wireless communication, there is still no single reliable solution for robot-to-robot communication. The tricky part is that in swarms, robots need to sense not only the environment but also each robot in the swarm. Modelling and predicting communication is notoriously hard and robots need new reliable approaches to communication [71].

2.2.2.3 Computation

Current robots have limited resources especially when it comes to battery power and computation. Developers are trying to produce energy-inefficient robots to improve the battery timings, which is an important issue, especially for mobile robots and drones. An increase in the use of these systems has led to new developments for battery technologies that are safe, cost-effective, and lasts longer than before. However, still, they are not efficient for a robot to perform tasks which require high computation [71]. For example, currently, the robot cannot easily ask the question “Have I been here before?” because to do so robots need to utilise previously collected data. It is difficult to have data stored on the robot because it requires a large amount of battery power, storage, and computation.

Besides that, different tasks require different computations and resources utilisation. Robots are required to perform different tasks in real-time including vision processing, mapping, and navigation. Therefore, the components and resources should be utilised efficiently. Middleware can be used to help the robots use these resources efficiently and according to the requirements of different applications.

2.2.2.4 Biohybrid and Bioinspired robots

Recently, nature-inspired robots are becoming more common in robotics labs. The idea is to develop robots which perform tasks the same as efficient systems found in nature. However, studies show that the issues involved

in this area are far too great to be solved any time soon. For example, a battery to perform highspeed conversion, powerful actuators, self-healing material, perception like humans, computation and reasoning are required to develop biohybrid robots. These robots could provide useful features such as body support for elderly or physically challenged people, weight reduction for labour, impact protection, high speed computation, and mobility [170].

2.2.2.5 Battery Power

Robotics has seen many advances in recent years. Computer processing and sensors have become cheaper and miniaturised which has helped with the development of energy-efficient and computationally more powerful robots. However, the life span of batteries has limited the infiltration of robotics into daily life because current batteries are woefully inefficient and slow to recharge. This is the reason that researchers are developing new batteries which are affordable, safer and longer-lasting. A possible solution for the robot to extract energy from light, vibration, and mechanical movement is also under consideration [170].

2.2.2.6 Navigation and Exploration

One of the most common and important uses of robots is the exploration of unknown terrain or environments. A large amount of work has already been done in this area. To navigate through the environment and to sense its surrounding, the robot needs different types of sensors and components which need to be fixed at a suitable position. For example, if the collision detection sensor is too high it won't be able to detect obstacles that are lower than the sensor.

Furthermore, these types of robots are mostly used in places where it is not easy to access them frequently. Therefore, these robots should be able to handle failure and able to adapt, learn, and recover for successful navigation of the environment. For exploration, it should be able to differentiate between old and new discoveries [170].

2.2.2.7 Providing heterogeneity abstractions

Nowadays, robots are multitasking which requires the use of heterogeneous hardware and software components. Therefore, it is important to have reliable communication so that these components can work together. Usually, a middleware provides this type of functionalities by acting as a collaborative layer between all the modules involved, reducing the impact of poor communication and the heterogeneity of the modules [1].

2.2.2.8 Offering often-needed robot services

Re-implementations of current algorithms, and control services for robots may need to be repeated several times, which takes a great deal of time and effort. It can be due to the changes in hardware, changes in technical staff, implementation of new applications, operating system updates, or just when you want to add new functionalities. The middleware should facilitate these types of changes and services so that modules can be reused by offering these functionalities.

2.2.2.9 Limitation of component resources

Often robots are equipped with components that have limitations. These limitations include limited battery power, low storage memory, slow computation, and low-level connectivity. The functionalities provided by middleware can be used to manage these devices efficiently when needed. For example, if an equipped device is not in use, it should be in standby mode to save battery power [170].

2.3 Design Environments for Robotics

There are many surveys covering diverse topics in Robotics such as space robotics [18], evaluations and characteristics of robotic programming environments, [76], vision for mobile robot navigation [33], and robotic mapping techniques [152]. Also, there are some surveys regarding the use of middleware for rapidly growing technologies including ad hoc networks [53] and wireless sensor networks (WSN) [54]. However, the research concerning

robotic design environments are less frequent in comparison to the research mentioned above. In this section, some of the important projects regarding middleware for robotic design systems are explored and discussed.

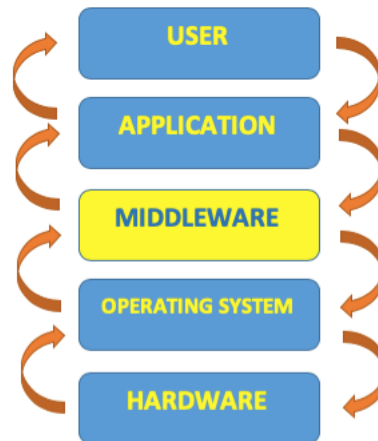


Figure 2.1: A generic approach to the use of middleware for robotic systems [39].

The middleware sits in-between the software application used by the user to program the robot and the operation used by the robot's hardware to receive commands from the application (see figure 2.1). There are many reasons behind the use of middleware in robotic design systems, such as it can be used to improve the quality of the software application, make the software design process simple, reduce the cost of development, and manage the heterogeneity of the hardware. To develop a middleware, the developers need to develop an algorithm that can be integrated with other components within the robotic design system. Furthermore, if they need to make any improvements, they only need to replace old components with new ones.

Some problems faced during the development of middleware are outlined by Smart [144]. Kramer and Scheutz [76] performed a survey regarding robot development environments (RDEs) in which they described, compared, and evaluated nine open-source, free of cost RDEs for robots. Mohammad et al. [95] explored research projects regarding middleware for robotics and provided a short overview of a few middlewares and their objectives. A year later, a survey of networked robots middlewares and their evaluation was provided by the same team [99]. Furthermore, Namoshee et al. [100], discussed some of the freely available middleware environments for robotics, including these environments, applications within the field of robotic systems.

In the past, robots were designed for specific tasks and will only perform that task, whereas, modern robots are autonomous and ubiquitous because of their design and implementation. New robot hardware components are heterogeneously interconnected. These components usually can be controlled using software modules developed in different programming languages by different manufacturers. These software modules can process the information received from sensors and send control commands to actuators to perform required tasks, including planning, navigation, and user interaction.

Despite the advantages provided by the module design, it still faces integration issues that can be mitigated by using middleware. These issues include communication, interoperability, and configuration [95].

Elkady and Sobh [39] suggested that to fulfil the modular design requirement, the middleware needs to have the following capabilities. It should not be limited to a specific device or software application. It should be easy to use, upgradeable, scalable, robust, flexible, maintainable, and reliable. Besides that, it should also be secure to use such as secure communication and authentication.

Some of the popular middleware projects proposed and implemented by researchers are presented below.

2.3.1 Player/Stage System

Player/Stage system [77] was developed, at the University of Southern California, to serve as an interface to robotic devices. It supports many types of robot devices and provides drivers, infrastructure, and algorithms for robotic applications. As alluded to its name, it consists of two parts: Player and Stage. The Player is a device repository server for robotic devices such as sensors, actuators, and robots, where all devices are assigned a drive and an interface. To receive, process, and send data, these drives implement algorithms. On the other hand, the Stage is a graphical simulator where users can model devices in an environment. This middleware is designed as a three-layer architecture:

1. Clients
2. Interfaces
3. Devices

The client represents software developed for specific robotic applications, interfaces represent robot devices and services, and actual devices are actuators, sensors, and robots. There are several client-side libraries written in programming languages (C, C++, and Python) that can provide access to the services offered by this middleware. These services include accessing data from devices, sending commands to a device, and changing configurations of an existing device.

2.3.2 Orca

Orca [88] is an open-source development tool for component-based robotics. It is used in several different applications including a single vehicle to distributed sensor networks. This middleware encourages the user to reuse the robotics software by allowing the users to implement a distributed component-based robotic system where the user can define interfaces and communication mechanisms. At the start, ORCA was implemented using CORBA. However, because of some complex problems, Ice [62] was developed. This new approach added the capabilities of smaller and more consistent API, better services, and improved performance.

2.3.3 Robot Operating System (ROS)

ROS was developed by Stanford University, which is an open-source software where many tools, libraries, and conventions work together to write robotic software. The functionalities provided by these libraries and tools include receiving, writing, compiling, and running codes across multiple computers. The services provided by ROS include designing hardware, controlling devices, implementing functionalities, sending, and receiving messages/commands, and package management. The main advantages of ROS are integration and

language independence. It allows you to use code written in other software frameworks and code can be in any modern programming language [132].

2.3.4 Webots™

Webots™[93] is a mobile robotic simulation software developed by Cyberbotics Ltd. It can quickly generate prototyping environments for modelling, programming, and simulating mobile robots. Besides, several commercially available robots can be controlled by generating a control program in Webots™. It allows the user to define, set up, and modify several robots by using the same environment, and the robots can be equipped with most of the available sensors and actuators. These robots are programmed in C, C++, and Java, simulate them and can also transfer them to your real robot. However, the software is commercial and not free-to-use. Also, users have to have high-level programming knowledge to use the software.

2.3.5 Open Robot Interface for the Network (ORiN)

ORiN [94] is an interface which was developed by the Japan Robot Association to access and control computer-based robotic applications. ORiN is written in web languages such as HTTP, XML, and SOAP. It mainly targeted industrial robots and separated specifications and implementation, which encouraged third parties to develop computer-based robotic applications. Hence it is possible to create low-cost multi-vendor systems. It can be used on any network and with any programming language because of its distributed object model design. It supports several types of specifications, and vendors can use XML to define specific options.

2.3.6 Sensory Data Processing Middleware

University of Tsukuba (Japan) developed Sensory Data Processing Middleware (SDPM) [157] to support service mobile robots by providing access to sensory information. Services provided by this middleware are:

- Services to provide information about obstacles.
- Services to determine the robot's position.

These services are provided by utilising the landmark information sensed by several external sensors. Its functionalities provide a single model with different configurations for external sensors on a service mobile robot. This module can be used with or without any changes for any service mobile robot.

2.3.7 Data-Centric Middleware

This environment [49] is a data-centric middleware to integrate wireless sensor networks and mobile robots. The universities of Seville, Spain and Stuttgart, Germany, developed it for the European project AWARE (platform for Autonomous self-deploying and operation of Wireless sensor-actuators networks cooperation with AeRial objEcts). The middleware provides simplified mechanisms to integrate the information gathered using different types of sensors. Such processes are required for applications that use robots to obtain data (temperature, light level, or high humidity) from their environments and feedback using a wireless network connection.

2.3.8 LEaRN – LEgo Robot and Netlogo

LEaRN was an attempt to improve students' classroom experience by the University of Split, Croatia. It is a combination of a simulated environment and a real physical robot. The main purpose of this environment is to make use of the advantages of a simulated environment with the possibility to demonstrate agent behaviour by using a physical robot. LEaRN allows two-way communication between simulation and robot. You can transfer data from the simulation to the robot and from the robot to the simulation.

Table 2.2: A comparison of robotic environments' features.

	Player/ Stage	ORCA	ROS	LEaRN
OS	Linux, Mac, Win	Linux, Mac, Win	Linux, Mac, Win	Linux, Mac, Win
Simulator Type	2-D	2-D, 3-D	2-D, 3-D	2-D, 3-D
Programming Language	Player(any) Stage (C, C++, Python, Java)	C++, Java, Python, PHP, Matlab	C, C++, Python, Octave, LISP, Java	C#, .NET, NetLogo
Documentation	Low-Level	High-Level	High-Level	Low-Level
Tutorial	Yes	Yes	Yes	No
Portability	Yes	Yes	Yes	Yes
Sensor	Odometry, range	Odometry, range, camera	Odometry, range, camera	colour, ultrasonic, bump
Debugging/ Logging	Yes	Yes	No	No
GUI	No	Yes	No	Yes

This environment consists of three parts – the simulated environment, a communications channel, and the physical environment. NetLogo [158] has been used as a tool for depicting the simulated environment of the robot. There are several reasons that it was chosen such as its simplicity for beginners and fewer code requirements. The communication channel to exchange information between NetLogo and the robot was a special application developed using .NET and C# programming languages. The robot was programmed using the NXC programming language to receive the commands from NetLogo [171]. In a case study, students were asked to use this framework in some selected assignments and some excellent results were achieved. However, there is no evidence of the experiments conducted.

The next section and its subsections provide the background of Evolutionary Robotics, more specifically Grammatical Evolution which is used in this thesis to evolve robotic behaviours using a fitness function. It also discusses the existing robotic environments which use Grammatical Evolution.

2.4 Evolutionary Robotics

This section provides a brief introduction to the field of evolutionary robotics and its algorithms, followed by a detailed background and description of Grammatical Evolution. It then continues to provide the outline, applications, implementation and evaluation of grammatical evolution.

Evolutionary Robotics is useful for exploring biological mechanisms and investigating robotic applications' design space and testing for scientific hypotheses [105]. Evolutionary Robotics (ER) is a sub-field of Behavioural Robotics (BR) that develops evolutionary computing applications for autonomous robotic systems. One of the main aims of ER is to develop automated applications that can evolve complex robotic behaviours. Therefore, recent ER research is mainly focused on mobile robots and robot colonies [102].

Evolutionary robotics utilise sensory-motor coordination to allow robots with limited resources (sensors, battery, computation) to solve complex problems [140]. Evolutionary computation is used by ER to design robots and control software automatically [42]. The ER field is divided into two main parts: methods for cognitive science [57] and biology [10], [38], [82], and methods for engineering purposes. This thesis mainly focuses on the second category along with first category add-ons to automatically design and maintain an efficient robotic design system [36]. Specifically, robotic behaviours will be governed and evolved by the new developed robotic design system. In order to attain the desired results, it is necessary to replace pre-programmed approaches with evolutionary principles where a robot is only provided with the specification of the task; see [56], [81], [84], [24], [43], [129], [85] for examples.

ER aims to design such a robot that can adapt to the changing environment and evolve the solution of a specific task. Evolutionary Algorithms (EA) have demonstrated that they can accomplish such goals [5]. Furthermore, these techniques have shown the potential to design autonomous adaptive robots

and generate modular robotic behaviours such as self-assembly of robots, self-reconfiguration after detecting a change in the environment, and self-repair after collusion. These behaviours allow the robots to explore dangerous and difficult to access terrains.

The next section briefly describes evolutionary algorithms and their applications. The purpose is to identify the most promising methods to design autonomous adaptive robots.

2.4.1 Evolutionary Algorithms

Evolutionary Algorithm (EA) is an optimisation algorithm that imitates the biological mechanisms behind natural evolution such as mutation, recombination, and natural selection in order to find an optimal design within specific constraints [12]. The number of robotic system design environments have increased along with the requirement for more automated processes in real-world optimization problems and require high efficiency and robustness while designing the algorithms. These applications include computer vision [104], robotics, big data analytics, and bioinformatics [70]. In order to fulfil these requirements, the current trend is to design optimisation processes by machine learning and search methodologies [12].

Evolutionary algorithms are classed as general, randomised search heuristics that can be used to perform many different tasks. A number of different parameters are required to perform each task which are very crucial for the success and efficiency of the search. Even though it is based on empirical experience mainly, however, finding the appropriate settings is still a difficult task.

In the evolutionary algorithm domain, the following main algorithms can be mentioned: genetic algorithms (GA) [52]; genetic programming (GP) [126]; and grammatical evolution (GE) [136]. Each of these algorithms has many different variants and are being used in several different industries.

2.4.2 Genetic Algorithms

Genetic algorithms are part of a large class of evolutionary algorithms which takes inspiration from natural evolution to generate solutions to optimisation problems [30], [127]. A genetic algorithm evolves a population set over and over to direct itself towards optimal solutions. That population is termed as candidate-solutions of an optimisation problem. Initially, the candidates are encoded in a specified format according to their real or binary genetic representation. In the case of binary encoding, a candidate-solution's properties are represented by a bit string consisting of 1s or 0s locations. The genetic algorithms evolve the population for subsequent generations by using three operators which are called reproduction, crossover and mutation (as shown in fig 2.2).

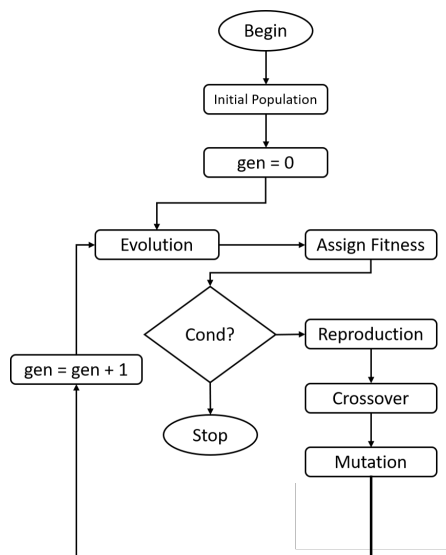


Figure 2.2: Flowchart of Genetic Algorithm [52].

A fitness function is used to measure the fitness of each candidate. The reproduction operator uses a probability-based technique to help in the selection of the best candidates in each generation. A mating pool is formed after the selection of the best candidates. These mating pools are used by crossover operators which randomly select a site at which crossover operations are carried out (as shown in fig 2.3a). Crossover operators are followed by the mutation operators which may change randomly selected features in the offspring. This operator flips the value of a binary digit in a string that represents the genome of the candidate-solution as shown in figure

2.3b, at a random location selected with a small probability. This may help the population to emerge out of a local minima during the search optimisation process.

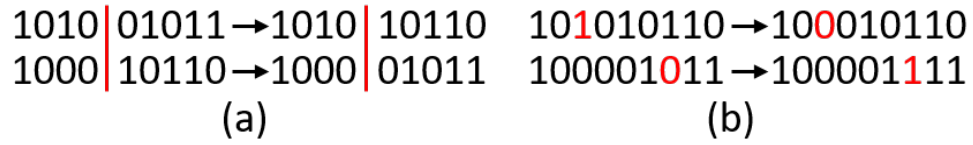


Figure 2.3: (a) Crossover Operator, (b) Mutation Operator [52].

2.4.3 Genetic Programming

Genetic Programming is an Evolutionary Algorithm which usually uses syntax trees to encode individuals. There are several research studies proposing various other forms of encoding genetic programming since the original tree-based genetic programming was invented by Koza [126]. The generational process of genetic programming is as follows: it randomly creates an initial population of genetic programming individuals using a tree generation method. A predefined fitness function is used to evaluate the fitness of every individual of that population. Koza [74] proposed three separate methods to generate the initial population which are full, grow, and ramped (half-and-half) methods.

After the first run, if the required results are not achieved, the individuals of the current population with high fitness value (parents) are selected to generate the new population (offspring). The process is repeated until the required results are achieved [40]. To generate offspring, the parents are modified using genetic operators called crossover and mutation. Crossover operates by exchanging sub-trees which are randomly selected from each of the two selected parents. For mutation randomly selected trees replace each other on the selected parent. The offspring replaces the initial population during each generation. Genetic Programming keeps repeating this process continuously until an individual achieves the required fitness level to terminate the process. A predefined set of functions and terminals is used by Genetic Programming to construct individuals.

2.4.4 Grammatical Evolution

Grammatical Evolution is a grammar-based form of genetic programming which evolves programs or rules in any language [108], [31], [135], [61], [21], [7], [85], [125]. Grammatical evolution is similar to genetic programming as it also generates variable-length computer programs by using the evolution process. However, the difference between the two is that grammatical evolution implements linear genotype population (binary or integer strings). A genotype-to-phenotype mapping process is used to transform these strings into functional phenotype programs. The whole process is observed by using a BNF grammar which is used to generate the produced solution code. Because grammatical evolution uses grammar to describe generated structures, a user can simply write or modify a grammar in a text file to manipulate the output structure. This provides ease of application and attractive flexibility which is not available in the standard approach to genetic programming. Furthermore, the solutions can be generated in a number of languages including but not limited to Java, C/C++, Lisp, Postscript, and English [32].

Grammatical evolution was introduced by Ryan et al. [136] as a new approach for evolving computer programs. In this method, a variable-length linear genome was used which governs the mapping of a Backus-Naur-Form grammar into a solution code (see figure 2.4) which can be executed to solve complex problems. The result is that expressions and programs of arbitrary complexity may be evolved. They successfully applied this newly proposed method to the symbolic regression problem.

There are three different aspects of grammatical evolution [114]:

1. Grammars
2. Grounded in molecular biology
3. Evolutionary automatic programming

They further explained that grammatical evolution is a form of genetic programming [74], but it is different to the traditional approach in three ways:

1. It employs linear genomes.
2. It uses a genotype to phenotype mapping.
3. Legal structures in phenotypic space are dictated by the use of grammar.

Furthermore, O’Neil and Ryan argued that grammatical evolution is different from genetic programming as it is closer to the process of natural evolution [116], [111].

O’Neil and Ryan noted that grammars provide a simple mechanism to describe any complex structure such as languages, graphs, mathematical expressions, molecules, and more. This makes grammatical evolution a powerful tool for the evolution of any arbitrary structure as long as context-free grammar can describe that structure.

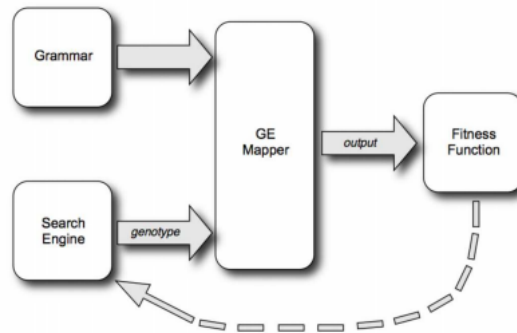


Figure 2.4: Modular components of grammatical evolution [110].

Different types of grammars are used with grammatical evolution including attributes grammars [28], Christiansen grammars [118], shape grammars [113], and tree-adjunct grammars [98]. The grammar used for experiments performed in this thesis is Backus-Naur-Form (BNF), which is briefly described in the next section.

2.4.4.1 Backus-Naur-Form (BNF)

The grammar notation most commonly used in Grammatical Evolution is Backus-Naur-Form (BNF). BNF expresses the grammar of a language as a set of production rules. It consists of three sets which are:

- A set of terminal symbols (items which appear in language, eg, + or -).
- A set of non-terminal symbols (can be expanded into one or more terminals).
- A set of production rules of the form (*Left-Hand-Side* ::= *Right-Hand-Side*).

According to the production rule, the left-hand side (non-terminal) can be replaced by the expression on the right-hand side (terminals or non-terminals) [29].

A BNF grammar G can be represented by a four-tuple $\langle N, T, P, S \rangle$, where N represents a set of non-terminals, T a set of terminals, P a set of production rules that map the elements of N to T and S (a member of N) represents the start symbol.

2.4.4.2 Mapping Function

Grammatical evolution draws inspiration from molecular biology where characteristics of phenotypes are defined by using a sequence of genetic material called deoxyribonucleic acid (DNA), also known as genotype. Ryan et al. [135] provided a detailed comparison of grammatical evolution and the biological process. In grammatical evolution, a binary string is used to translate an individual to a chromosome, where each gene of the chromosome is referred as a codon and contain on 8-bit binary string. The following mapping function is used to convert each codon to its decimal value from which a suitable set of production rules is selected [107]:

$$Rule = (codon\ decimal\ value) \% (No\ of\ production\ rules). \quad (2.1)$$

Grammatical evolution has provided solutions for a variety of problems and to several different fields since its invention. The research related to robotics is reviewed in the next section 2.4.5 and some related implementations have been explored in section 2.4.6.

2.4.5 Current and Past Research

This section will focus on applications of GE that are related to the applications discussed later and specifically robot-related applications. In 1999, O’Neil and Ryan introduced grammatical evolution to solve a real-world problem by evolving caching algorithms because the solutions generated by genetic programming were not good enough to outperform the solutions designed by humans [115], [117]. O’Neil and Ryan created a caching algorithm in the C programming language which showed a significant increase in performance when applied to a large cache size. The solutions produced by grammatical evolution were powerful enough to solve complex problems.

Tsoulos, Gavrilis and Glavas (2005) constructed artificial neural networks (ANN) by using a grammatical evolution based technique [160]. The proposed method was able to construct both types of neural networks: neural networks with an arbitrary number of hidden layers, and recurrent neural networks. A series of classification and regression problems was used to test the performance where the results were compared against the performance of a traditional Multilevel Perception.

Tavares and Pereira (2012) introduced a GE based technique to evolve configurations for Ant Colony Optimisation (ACO). To get better results, they used the features of crossover, mutation, and tournament selection to find the fittest individual. The performance of the system was evaluated against a human-designed ACO method, where the results showed that the automatically generated method by the presented approach outperformed the methods manually designed by humans [155].

Lourenço, Pereira, and Costa (2013) evolved solutions for knapsack problems by using a grammatical evolution based hyper-heuristic framework. The

features of this framework included single point crossover, mutation, and tournament selection to find the best solution. The solutions generated using this method outperformed the solutions generated by other traditional methods [86].

Peabody and Seitzer (2015) introduced a novel approach of grammatical evolution called Grammatical Evolution for the Finch (GEF). This approach provided a program written in Java programming language to control the robot. To handle contingencies, the robot was able to perceive, decide, and act whenever unplanned events and dynamic changes in the environment were detected. For example, the robot was able to change its marked location if the priorities marked location was not available, send an alert about the problem detected, and automatically download the new instructions wirelessly [122].

Lourenço, Pereira and Costa, applied novel Structured Grammatical Evolution (SGE) to three problems (symbolic regression, path-finding, and predictive modelling). The results showed the effectiveness of SGE against the standard grammatical evolution techniques [85].

Perez and Nicolau (2018) evolved behaviour trees to control a video game (the Mario AI Benchmark) by applying grammatical evolution. The results showed that these controllers performed well when manoeuvring to avoid challenging obstacles, and reactive behaviour capabilities to shoot enemies [125].

The next section provides some of the related implementations of grammatical evolution for robotics design systems.

2.4.6 Implementations of Grammatical Evolution

Grammatical evolution is implemented for numerous purposes in a variety of languages which PonyGE2 [41], PyNeurGen [145], GERET [121], GEM [162], gramEvol [106], GEVA [110], libGE [103], jGE [47]. Some of these which are related to this research are described briefly below:

2.4.6.1 Grammatical Evolution in Java (GEVA)

GEVA is implemented in the JAVA programming as an open-source library developed by UCD's Natural Computing Research & Application group [161]. GEVA provides features of implementing the mapping process of genotype-to-phenotype, a graphical user interface (GUI), and an evolutionary algorithm to convert a variable-length integer, which makes it a complete implementation of a grammatical evolution system. Also, it is capable of providing implementation of demonstration problems such as symbolic regression, the max problem, the royal tree problem, and the artificial ant problem (Santa Fe Trail, Los Altos, and San Mateo).

This software contains two parts: GUI and GEVA [110]. The GUI allows the user to configure and execute the demonstration of the problem (see figure 2.5), whereas, GEVA provides scripting features by providing a command-line interface (see figure 2.5).

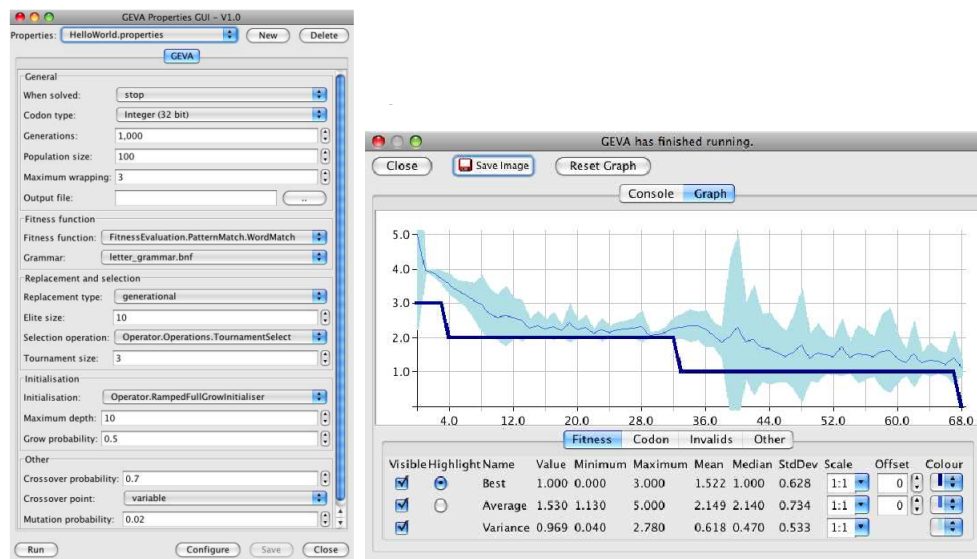


Figure 2.5: The components of GEVA v2.0 [110].

2.4.6.2 libGE

The libGE library is open-source software development of grammatical evolution systems that is one of the first GE software which was available to the general public. It is developed for the Linux operating system and written in the C++ programming language. The last version (0.26) of libGE

was released in September 2006 [47]. Any compatible evolutionary algorithm system can use libGE to implement the mapping process. The process of mapping genotype to phenotype is implemented by libGE where genotype is the result of the search algorithm and the phenotype is the solution to be evaluated. Nicolau stated in libGE documentation that it can take a string from a variable length-genetic algorithm and map it onto a syntactically correct program whose language is specified by the context-free grammar using BNF [103].

For example, binary strings are provided by a chosen Search Engine to libGE. These strings are mapped into programs by using a specified grammatical language such as BNF grammar. Then, the Evaluator (an interpreter or compiler) implements the problem specification by evaluating the resulting programs. And these evaluated results (based on the fitness score) are returned back to the Search Engine to create offspring and to set up new individuals (binary strings) for libGE [103].

Initially, libGE was designed for work related to Context-Free Grammars (CFG). Therefore, a specific set of codon values will only produce specific production rules for a non-terminal symbol [47]. To improve the control over the mapping process, libGE introduced a set of commands that can work together with grammar.

2.4.6.3 Java Grammatical Evolution (jGE)

The jGE [47] library was the first implementation of grammatical evolution for the Java programming language (Java SE versions 5 and 6) that was published and was available for free. Until recently, artificial intelligence and intelligent agent research groups at the School of Computer Science of Bangor University used jGE as a core component but this was no longer possible due to incompatibilities as a result of frequent software upgrades. A major effort was required to update the software in order to use it for various applications as described later in this dissertation.

The main aim of jGE is to implement a framework of an open Evolutionary Algorithm (EA), to facilitate further research into evolutionary algorithms, especially Grammatical Evolution (GE) because grammatical evolution facilitates the use of BNF grammar, arbitrary structures, and programming languages [46]. The main objectives of jGE were: to provide an open and extendable framework; to create an agent-based evolutionary system; to facilitate further research of nature and biological principles; and to evolve competent robot designs which are better than the robots designed by humans.

2.4.7 Performance Evaluation of Grammatical Evolution

Grammatical evolution has performed better than other grammatical algorithms in a series of benchmark problems including symbolic regression problems [149], [112], symbolic integration problems [74], [106], artificial ant problems [74], [75], [109], [112], and maze searching problems [147], [48]. Some of these performances are discussed below.

2.4.7.1 Artificial Ant Problems

This problem was created by Jefferson and his team [66] by using the tracker system which was built by the UCLA Artificial Life group. The aim of the problem was to design complex behaviours using evolutionary algorithms. The trail used in the original system was called the John Muir Trail [66], [80], [150]. The objective of this problem is to control an artificial ant by using a computer program so that it can find all the pieces of food lying along the trail within a reasonable amount of time [150]. The artificial ant could perform one of the three available actions (`move`, `turn-right`, and `turn-left`) where the first action moves the ant one square forward and the other two actions turn the ant 90 degrees right or left respectively. The artificial ant also uses a sensing operation called `food-ahead` to check if the square the ant is facing contains food or not.

Two instances of the artificial ant problems are discussed below in detail, the Santa Fe Trail and the Los Altos Hills:

Santa Fe Trail The Santa Fe Trail (see figure 2.6) is defined as a standard problem to be used for benchmarking in genetic programming [74], [75], and grammatical evolution [109], [112], [163].

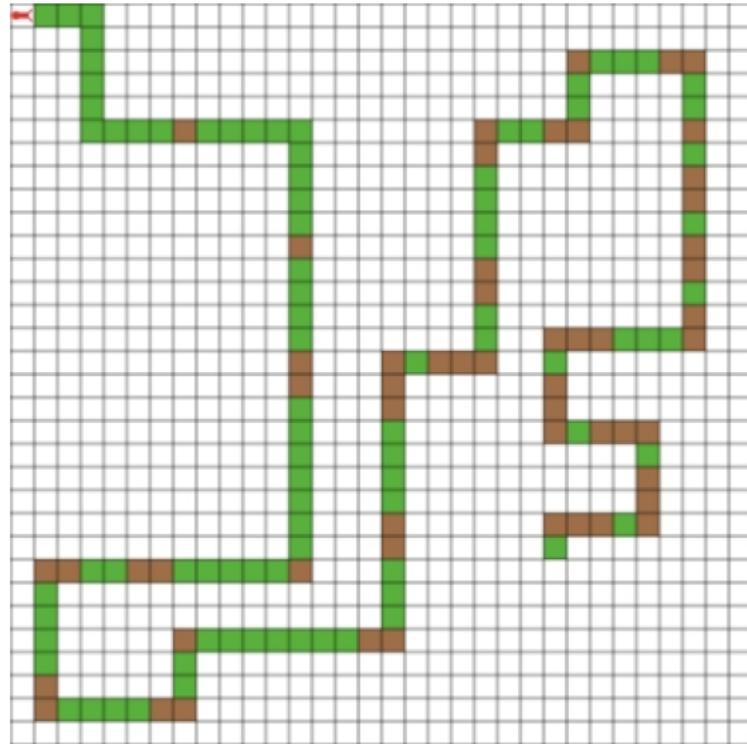


Figure 2.6: The Santa Fe trail problem where green squares represent food and brown squares represent gaps [74].

The objective of the Santa Fe Trail is to control an artificial ant using a computer program to find the 89 pieces of food laid on the trail, represented using green squares and gaps in the trail, represented using brown squares. The artificial ant starts exploring from the top left square of the grid (0, 0) in a square 32 by 32 toroidal grids. There are 144 squares in the trail with 21 turns and 89 pieces of food distributed randomly along with it. The grid contains the following irregularities [163]: single gaps in the food trail, double gaps in the food trail, single gaps at corners of the food trail, double gaps at the corner of the food trail, and triple gaps at the corner of the food trails. These gaps in the trail represent different situations to evaluate the performance of grammatical algorithms such as double gaps at the corners that require a chess' short knight move and triple gaps require long knight moves.

This problem was designed by Christopher Langton [74], [163], and is a more difficult problem than the original artificial ant problem "John Muir Trail" used

by Jefferson et. al. [66], [80]. The interesting features of this problem made it popular as a benchmark and it is still being used repeatedly in the field of grammatical programming [163]. Langdon and Poli [79] showed that the performance of genetic programming is not good enough for the Santa Fe Trail because of its multiple level deceptions. Similarly, Hugosson and his team [64] argued that Santa Fe Trail problem is a complex problem to solve which requires planning because it contains several local optima. Furthermore, they added that a limited genetic programming schema analysis showed that the problem is deceptive at all levels and there are no beneficial blocks to choose randomly. Therefore, it is a challenging task for an evolutionary algorithm to solve Santa Fe Trail problem.

Los Altos Hills The Los Altos Hills problem (see figure 2.7) is more challenging than the Santa Fe Trails problem as it contains 157 pieces of food, represented by green squares, located on a 100 by 100 toroidal grid [74], [85]. The artificial ant starts exploring from the top left square of the grid (0, 0) and explores through 221 squares with 29 turns and 157 pieces of food distributed randomly along the trail (see figure 2.7).

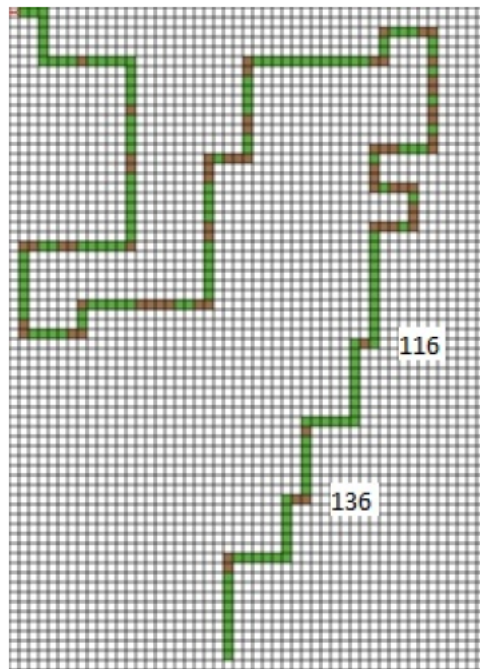


Figure 2.7: The grids of the Los Altos Hills trail problem [74].

This problem has a larger grid than the Santa Fe Trail problem which makes it more difficult. The trail begins in a similar fashion as the Santa Fe Trail such

as single gaps in the trail and corners, double gaps in the trail and corner, and triple gaps in the corners [90]. However, it introduces two novel irregularities towards the end of the trail. The first one appears at grid 116 as shown in the figure (see Figure 2.7) where the artificial ant is required to take two steps to the right or left of the existing food grid. The second one is difficult as it requires moving one step forward and then two steps to the left or right of the existing food grid as shown in grid 136 (see Figure 2.7).

2.4.7.2 Maze Searching Problems

A type of maze problem is a two-dimensional board where the agent (solver) needs to find a suitable path, from a series of confusing and difficult paths, from the starting point to the finish point. It consists of fixed walls and pathways where the agent can walk on pathways but can't see through walls.

One of the oldest mazes is the Cretan labyrinth [37]. Planning and making a strategy to reach the endpoint of a maze is a challenging problem in artificial intelligence. Different strategies are suggested for different mazes, for example, the "hand on the wall" solution [169] where the agent maintains its hand on one side of the wall and keeps exploring the maze until it reaches the goal.

If maze paths are divided into squares, it makes it easier to find the exit point. Developing a computer program which can control an artificial agent (solver) to help it find the exit point is the main objective of a maze searching problem. In a maze searching problem, an agent starts at the entry point facing towards the maze and can take only three actions (`move`, `turn right`, and `turn left`), same as the artificial ant problem. These actions take one-time unit each, however, no time units are taken by a sensing function of the agents such as `wall ahead`, `wall left`, and `wall right` [147]. These sensing functions check all three sides of the agent respectively and return the value `true` if there is a wall or `false` if there is a path.

It is a challenging task for evolutionary algorithms to solve maze searching problems because mazes contain local optima and obstacles block the way

towards the target regularly. Furthermore, walls are similar to each other which makes it confusing for the agent to find a way around them to reach the target.

Hampton Court Maze The Hampton court maze represents the Hampton court palace maze in the UK and its grid size is 39 by 23 as shown in figure 2.8 below [60]. The agent enters the maze at the middle bottom (at the location marked by the green cell) and its target is to reach the centre of the maze (marked by the red cell).

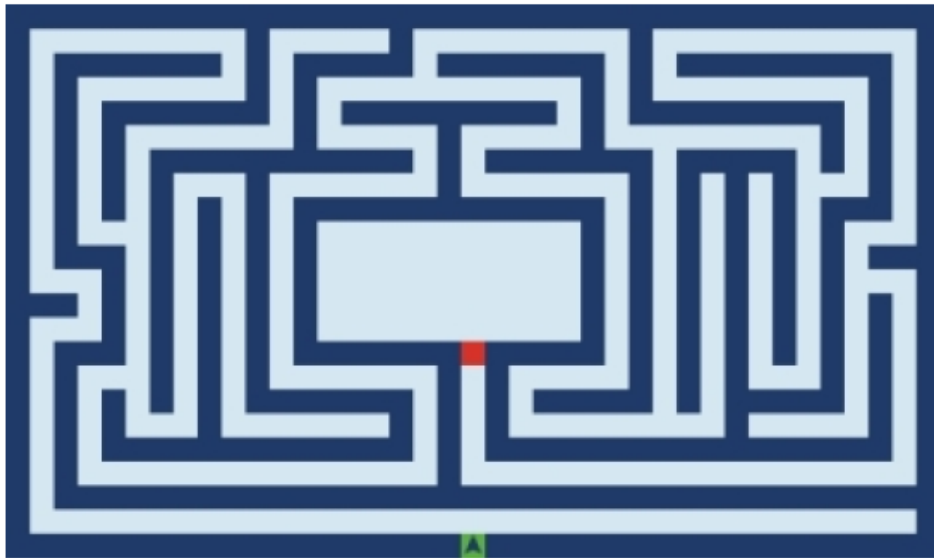


Figure 2.8: The Hampton Court Maze, green dot represent the starting point and red dot represent the finishing point. [60].

As shown in figure 2.8, the Hampton Court maze is a simply connected maze and a human can always reach the target by simply adopting the behaviour of keeping its left or right hand on the wall. Although the maze can be solved with this one simple technique, Teahan [156] stated that the presence of several local optima in the Hampton Court maze make it a challenging task.

Chevening House Maze The Chevening House maze (see figure 2.9) is a multiply-connected maze of grid size 47 by 47. Teahan [169] stated that the Chevening House maze was built in the 1820s and was one of the first multiply-connected mazes. These multiply-connected mazes are represented by one or more “islands” which are separated and disconnected from outer walls. An important difference between the Hampton Court Maze and Chevening

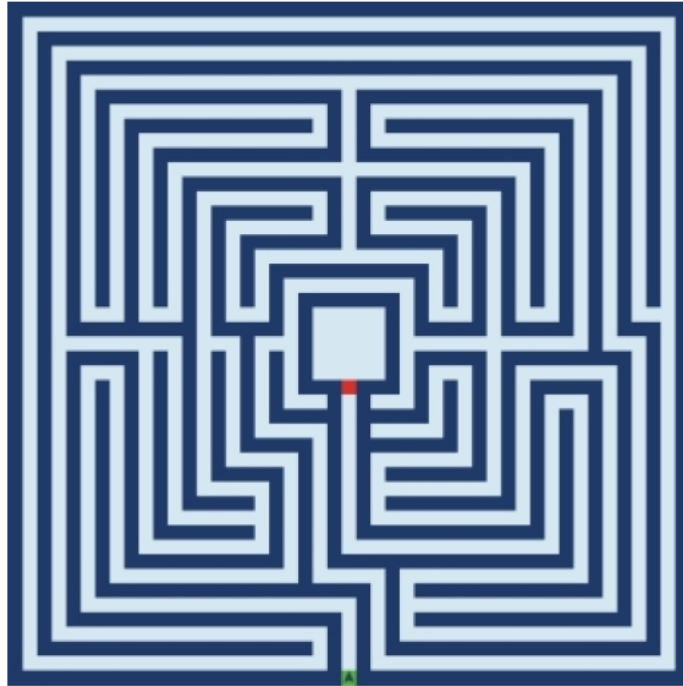


Figure 2.9: The Chevening House Maze [169].

House Maze is the design of the later doesn't allow the "hand on the wall" behaviour to solve the maze [169], [60].

The next section and its subsections provide the background of interestingness in robotics. It also discusses the current and past research to discover interesting robotic behaviours. Further related work will be discussed before summarising the chapter.

2.5 Discovery and Validation of Interesting Robotic Behaviours

This section and its subsections provide background and related work about the discovery of interesting robotic behaviours which will be evolved using the grammatical evolution approach. This relates to third and fourth objectives of this thesis which concern the use of new robotic system environment to discover novel interesting robotic behaviours which will be validated in the real-world.

The next section explores Behaviour-based Robotics in general, and then examines the concept of Interestingness concerning Robotics-based research, and then discusses how interesting robotic behaviours may be discovered.

2.5.1 Behaviour-based Robotics

Behaviour-based robotics is a biologically inspired technique for designing autonomous robots, where a collection of behaviours is implemented and acts in parallel to achieve goals using inputs and outputs. Most behaviours have a basic structure where input is taken from sensors and out is sent to actuators (see figure 2.10). A coordinator is required to make sure only one command is sent at a time. Sometimes, the internal structure of the behaviour can include different modules with various other modules interconnected by sensors and coordinators [22]. However, these behaviours must not depend on each other and can be implemented in parallel so that a real-time response can be sent back while using the least amount of computation. This methodology can be used to build low-cost autonomous robots. Behaviour-based robotics has reliably performed standard robotic activities including navigating an environment, avoiding obstacles, exploring, and learning maps [124].

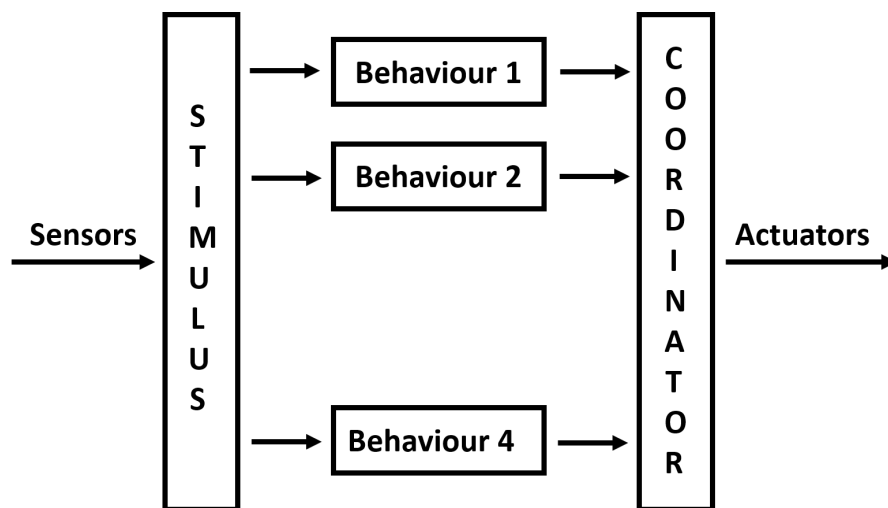


Figure 2.10: Structure of robotic behaviours [124].

Since Brooks subsumption architecture in 1986, several research proposals and reports have appeared in the field of behaviour-based robotics. A number of scientists classified these approaches into two sets according to their adaptability [173], [87]. At the start, non-adaptive approaches, also called engineering approaches, were popular as they were manually tuned to

implement a sophisticated action selection mechanism. Later on, adaptive approaches became more popular because of their simplicity of selecting action mechanism and learning techniques. These adaptive approaches include reinforcement learning and evolutionary techniques [124].

This thesis concentrates on adaptive approaches where robots are left in environments without any prior knowledge and robots adapt to the environment and change their behaviour according to the changing environments. Robots send this information back to the user where the user analyse the information to separate the interesting information. This process requires high bandwidth and computation by the robots. However, if robots are able to decide if their information is interesting or not and only sends back the information which is interesting, this would reduce the cost of computation and user involvement to make the robots more autonomous. For example, when an exploratory robot is exploring Mars and has the ability to decide if the information it has detected is interesting or not. This way it needs to send only relevant data and avoid sending unnecessary data. This can increase its life and exploration time by reducing unnecessary communication and computation.

Different research will have different requirements concerning what is deemed to be interesting or not; therefore, it is necessary to provide a definition of interestingness for our research.

2.5.2 Interestingness

Researchers have studied “interestingness” in the light of its use in data mining and knowledge discovery. This concept has been divided into two magnitudes of measurements which are: subjective (influenced by humans) and objective (based on quantitative data) [25]. Subjective measures correlate the current pattern of human behaviour with the previous scenario to notice any persisting change. However, objective measures, on the other hand, are a more generic and human biased free approach as it scales its measure on the data available and quantifies statistical analysis describing emerging trends in their behaviour. Surprisingly, the field of data

mining and knowledge discovery acknowledges the concepts and literature of “interestingness” but there has been a general lack in the study of its use for robotic systems.

According to Carvalho et al. [25], a behaviour can be considered interesting if it stands out from the previously observed behaviours. Such behaviour can often be classed as something novel or surprising; that is, one can say that “interesting discoveries are surprising” [44]. However, according to data mining and knowledge discovery requirements, an interesting discovery has to be useful. Otherwise, McGarry [91] points that unexpected behaviours could be considered as outliers or “noise” and may not be useful at all. Therefore, interesting discoveries can only be to one’s advantage if it’s useful. The relevancy of the behaviours is only considered useful if the interestingness measures are designed to eliminate non-relevant behaviours that need to be checked when data mining [63]. This endorses that measures of “interestingness” exploited by the data mining community may not be good enough to predict interestingness for robotic systems.

As mentioned above, some definitions of “interestingness” use an impression of surprise. Sometimes, a course of action predicts an outcome with high confidence, yet it fails to deliver its benchmark results which often underlines the scenario of large surprise due to different unpredicted results [15].

Hall and Mortan also conclude the key role of entropy estimators to construct measures of “interestingness” [55]. In fact, Blanchard et al. acknowledged Shannon’s conditional entropy as the most commonly used measure to calculate “interestingness” [20]. Moreover, Hussain et al. [65] adopts a relative entropy approach for “interestingness” using observations which examine the frequency of elements appearing in the data set along with comparison of unusual data sets compared with the preceding observations [91].

In contrast, Schmidhuber has a different outlook on the idea of what constitutes “interestingness” [141]. To examine the beautiful dimensions

of a face, he uses a prototype face which defines the standards of beauty. The more deviations are between prototypes, the less likely the face will be classed as beautiful. If one's dimensions are closer to its prototype, it is considered more beautiful than the one which results in higher deviations in comparison. He states that the value of a beautiful item can lose its "interestingness" over time when the items have been previously observed as new and beautiful [141].

A similar idea has been employed by Wang, C. et al. [165] where they explored the problem of predicting interesting scenes for mobile robots which is critical for several applications including autonomous exploration and decision making. A three-tier learning architecture (long-term, short-term, and online learning) was designed by them to enable their system to have human-like capabilities including experiencing the environment, adapting to the environment, and environmental knowledge. Their approach achieved better results than previous results of related experiments. One of the main reasons was that their approach was able to lose interest in repetitive scenes as well as identifying new interesting scenes whereas prior approaches struggled to do that.

The research presented above proves that it is possible to design a fully autonomous robotic system environment which can create and identify novel interesting behaviours. The following section discusses some research related to the problem of discovering interesting robotic behaviours.

2.5.3 Discovering Interesting Robotic Behaviours

There are several research and studies in the field of robotics where an attempt to discover interesting robotic behaviours have been made. For example, Ahmed (2020) [2] used Braitenberg vehicles and PPM to discover interesting behaviours. Wang et al, (2020) [165] designed a robot which can identify the interestingness of the scene by utilising visual memory and unsupervised online learning. Some of the recent and related applications which relied on the discovery of interesting robotic behaviours are reviewed in more detail below.

2.5.3.1 Visual Memorability for Robotic Interestingness

Wang, C. et al. (2020) [165] explored the problem of predicting interesting scenes for mobile robots which is critical for several applications including autonomous exploration and decision making. For example, when the robot finds the door shown in figure 2.11(f) this can affect future planning. Similarly, more attention can be attracted by that hole on the wall in figure 2.11(h). To recall and identify the interesting scenes, they proposed a new translation-invariant visual memory.



Figure 2.11: A selection of interesting and uninteresting scenes [165].

A three-tier learning architecture (long-term, short-term, and online learning) was designed by them to enable their system to have human-like capabilities including experiencing the environment, adapting the environment, and environmental knowledge. Their technique achieved better results than previous results of related experiments.

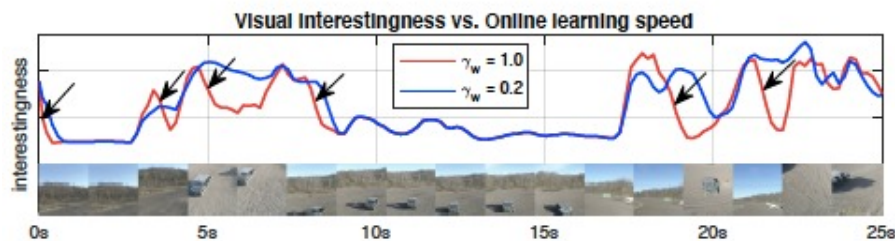


Figure 2.12: Visual interestingness of footage and loss of interest in repetitive scenes is indicated by the arrows during online learning [165].

One of the main reasons was because their approach was able to lose interest in repetitive scenes as well as identifying new interesting scenes in contrast

to prior approaches which did not focus on the more interesting scenes. For example, in Figure 2.12 when a truck appeared, it has a high level of interest, but it lost interest when it appeared repeatedly. To achieve this, they proposed a novel online learning approach to find interesting scenes for the robot exploration task whereas previous approaches depended on computationally expensive training via the back-propagation algorithm [134].

To solve the problem of autonomous exploration, they introduced a novel translation-invariant 4-D visual memory in order to find and recall interesting scenes. Human beings are considered to have an excellent capacity to direct visual attention and determine the interestingness of the scene. To replicate that in mobile robots, the following properties are necessary.

Table 2.3: Properties required by mobile robots to judge the interestingness of scenes [165].

Properties	Description
Unsupervised	Autonomous mobile robot.
Task-dependent	Task related knowledge.
Long-term learning	To acquire human-like experience.
Short-term	Learning for quick robot deployment.
Online learning	Environment adaption and real-time response.

2.5.3.2 Discovering Interesting Behaviours in Complex Systems

Ahmed (2020) [2] proposed a modified variant of Braitenberg vehicles to discover novel interesting behaviours in complex systems. In his modified version, proximity sensors were added to provide subsumption architecture functionalities. Subsumption architecture was used to avoid collisions with walls or other objects present in the environment. During his experiments, a large number of behaviours, from lack of movement to create simple patterns, were created by simply changing the location of light sensors. The visual output data produced by these experiments was compressed using the PPM compression scheme [3] in order to find when the output had noticeably changed during the simulation. These behaviours were later divided into four groups by using a k-means clustering algorithm using the PPM compression data. An example of the four different interesting behaviours that were automatically discovered is shown below in figure 2.13.

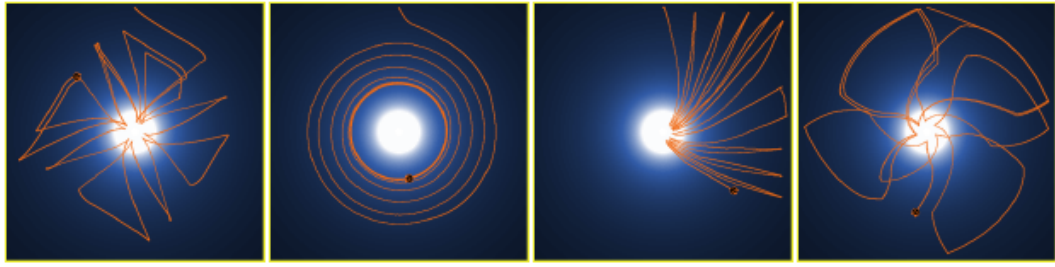


Figure 2.13: A variety of interesting behaviours was produced by Ahmed’s new variant of the Braitenberg Vehicle [2].

The results were gathered together after running 10,080 simulations. Many different possible combinations of configurations exhibited a large variety of behaviours. During these experiments, the k-means clustering produced a Davies-Bouldin index of 1.4533. These results show that this method was quite effective at automatically identifying the interesting behaviours out of a medium-sized data-set which would be extremely difficult for an individual person to have to classify manually. Real-world validation of these behaviours was left for future work.

A robotic system environment for predicting interesting behaviour has not been achieved, therefore the design of a new approach is an important contribution of the present work in order to detect interestingness, saliency, anomaly, and novelty.

2.6 Summary and Discussion

This chapter has discussed the background and related work that is relevant to this research. This includes a discussion of robotics, evolutionary robotics, grammatical evolution, interesting robotic behaviours and interestingness prediction by robots. This chapter is divided into three main sections:

1. Robotics design system.
2. Evolving behaviours using Grammatical Evolution.
3. Discovering and validating novel interesting robotic behaviours.

Robotics has a history as old as the history of science, technology and the basic principle of progress. Robotic systems are rapidly becoming an important part of our daily lives such as vacuum cleaners, medical/surgical robots, human assistance, exploratory robots. Potentially, robots may end up being one of the greatest accomplishments of human beings of creating an artificial sentient being. The main reason for interest in robotics is to design such robots which can be used to perform and make those tasks easy which humans cannot perform. This is especially true for those tasks which need to be performed in dangerous environments including radioactive places, deep-sea exploration, inaccessible terrains, or places too far away in distance and time (space exploration).

However, current robotic system environments are still task specific and require knowledge of programming languages to control them. Future robotics systems environments should be capable of quick and easy implementation, be flexible when performing different tasks, customisable to add functionalities, and interactive towards other systems such as sensor networks and enterprise information systems.

It has been long predicted that robots will become mainstream public agents often only requiring “basic intelligence”. Robots only need to be able to communicate, navigate and interact in the real-world environments. Currently, robotics hardware is advanced enough to fulfil these requirements. However, this prediction is held back by the complexity of design and implementation of thoughtful intelligence. Designing a robotic system environment is a challenging task which requires knowledge of robots (so that a suitable design can be chosen), and environments where robots are going to perform tasks (so that robots can be configured and programmed accordingly). One of the main reasons is the lack of research on the design and development of a design environment for robotics.

Some of the design issues currently present in the existing robotic system environments are identified and an effort to mitigate these issues is made by developing a low-cost robotics system environment. These issues include

design, fabrication, programming, communication, computation, battery power, navigation, flexibility, and multi-platform compatibility. These design issues were identified by reviewing the performance of currently available robotic system environments including Sensory Data Processing Middleware, Player/Stage Systems, Ocra, Robot Operating System (ROS), and Webots™.

After designing the robotic design system, it should be tested to check its capabilities. Evolutionary robotics is a useful method for investigating robotic applications as it allows the robots to perform complex tasks with limited resources by simple coordination of sensors and motors. It aims to automatically design adaptive robots which can evolve solutions to complete a task and adapt to the environment at the same time. However, it was challenging to design such robots which can adapt to the environment to complete the task. Evolutionary algorithms have demonstrated the feasibility to accomplish such goals. A detailed review of current research on evolutionary algorithms and their applications was conducted. Grammatical Evolution, which is a grammar-based form of genetic programming was selected to test the abilities of the new robotic design system after analysing its implementations in a variety of programming languages for various purposes.

Its applications related to our research were analysed deeply including PonyGE2, PyNeurGen, GERET, GEM, gramEvol, GEVA, libGE, jGE which were later used to compare the results achieved from our robotic system environment. The idea was to identify the most promising method to develop an autonomous adaptive robotic system. This novel robotic system should require minimum information on the designer side so that the robotic system environment can be made as user friendly as possible.

This thesis has concentrated on adaptive approaches because in this type of approach, robots enter an environment without any prior information and adapt to the environment and change their behaviour accordingly. Because these behaviours are sent back to the user to find useful information which requires high bandwidth and computation, however, sending back useful

information can reduce the cost of computation and user involvement to make the robots more autonomous. Thus, a detailed look into the ideas of interestingness for robotics has been explored and a variety of research related to finding interestingness is analysed.

The knowledge gained in this chapter, related to the identified issues and suggested solutions to mitigate these issues, will be applied in the next three chapters.

Chapter 3

NXTLogo – A new Robotic Design Environment

This chapter describes the development of middleware named NXTLogo where an agent-oriented 4th generation language is used to control NXT 2.0 robots which can provide additional functionalities along with the functionalities of 3rd generation middleware extensions/libraries. In addition, the developer has access to simulation and modelling facilities provided by NetLogo, and an ability to design using what Papert¹ called a 'body-syntronic' or first-person perspective—the developer designs the agent (imagined as a robotic 'turtle') to move using a perspective similar to their perspective. A couple of examples of NetLogo models to demonstrate the capabilities of this system (line-following ability and subsumption architecture roaming) have been developed and are described in this chapter.

This chapter is based on Paper 1 mentioned in the publications section of this thesis (section 1.6). Its purpose is to fulfil objective 2 of this thesis, developing an integrated low-cost robotic environment which is open-source and extendable. This is so that the research question 1 *"Can a low-cost robotic system design environment be developed that is open and extendable for easily performing evolutionary robotics experiments?"* can be answered partly.

¹Papert was the original designer of NetLogo's parent language, Logo.

3.1 Introduction

Robotics research has been conducted since the 1950s. The first industrial robot was introduced in 1961 which was used by GM for welding die casts onto car bodies [13]. Further research led to the development of many new and modern robots with more functionalities and features such as the computer controlled Stanford arm [139], first mobile robot Shakey [151], Genghis was capable of following a person based on the readings of an infrared sensor [68], Myrmex [11] was created with three principle layered behaviours: collect, avoid and safe-forward; and the winner of the 2005 DARPA Grand Challenge, Stanley [142] which was able to autonomously navigate a 175 mile off road course in under 10 hours.

The release of LEGO Mindstorms robots in 1998 increased amateur interests in the development of intelligent mobile robots. Nowadays, the LEGO kit is an essential part of educational institutions with many extensions created by the community provide additional functionalities. However, these extensions/libraries require the users to have internal knowledge of the system and of 3rd generation languages such as Java.

Lego Mindstorms has delivered low-cost amateur robotics to the public, where anyone can easily modify and develop new systems and extensions to extend its capabilities. Using the NXTLogo middleware, the robot can be controlled and linked directly to simulators in the agent-oriented programming language software, NetLogo. The technologies that the robotic system design environment relies on are heterogeneous since it is built in Java which runs on the JVM on all platforms as does NetLogo, being free to download and able to run on any platform. Also, the front end of the system is written in NetLogo, which is known for its simplicity of code and provides body-syntonic capabilities for real-time sensor feeds and robot commands to make and enact decisions. The robotic design environment does not require any programming skills from the end-user in order to implement behaviours (developed in NetLogo) on the robot. Significant results can be achieved by using the drag and drop feature and a modest amount of code. These features, design,

and cost of this robotic system environment fulfil the second objective of this thesis *"To develop an integrated low-cost robotic environment which is open-source and extendable"* in order to help answer the first research question *"Can a low-cost robotic system be developed which is open-source and extendable?"*. The next section provides background to this chapter.

3.2 Background

Robots have long been predicted to become mainstream public agents which will require (at least) basic intelligence (an ability to navigate, communicate and interact in the real-world environment). Generally, robotics hardware is advanced enough to fulfil these objectives; however, complexity of design and implementation of thoughtful intelligence are holding back this prediction.

Rodney Brooks outlined the idea of subsumption [22] in which architecture breaks down complicated behaviour to be the sum of several behaviour layers and each layer is subsumed in priority by the one above it. Sensor-based robots are characterised by four key phases, according to Brooks [97]: situatedness; embodiment; intelligence; and emergence.

To understand the Brooks approach, we need to understand cognitive and reactive behaviours. Cognitive behaviour for navigation maintains an internal model of an environment and comprehends relative position within it. In contrast, reactive behaviour does not maintain this relative position or internal model, it simply responds to stimuli. Wang [166] describes cognitive agents as being driven by intention and having a representation of their environment from which they can predict the future. For example, a robot vacuum cleaner can respond to simple stimuli to accomplish its tasks such as "Is this patch of floor dirty?", "Is there a wall in front of me?", or "Am I moving?" and this can be paired with appropriate instinctive responses.

The approach to designing autonomous systems has evolved over the years from being exclusively cognitive before the late 1980s to agents with more

reactive elements and behaviour. Reactive approaches to robotics provide a simple and effective means of realising complicated behaviours.

In the context of this chapter, a robot is capable of displaying aspects of Brook's behavioural approach if it can act immediately upon real-world events by utilising sensors and actuators and does not need an internal model to operate. Complex reaction, interaction and forward planning is all within scope. However, for our work, these capabilities are available if the robot is treated as a NetLogo turtle. NetLogo provides programming constructs using agents in the form of turtles, patches, links and the observer.

3.2.1 Related Work

NetLogo's API for Java extensions provides a wide range of possibilities especially for linking two independent systems such as NetLogo and MATLAB [131]. The approach for writing a NetLogo extension is seamless, with data passed from NetLogo, where it is acted upon in the secondary system and then passing back as a result. This is done in a way that the developer essentially has access to further NetLogo-like language commands without the need to be aware of the underlying implementation. Many other middleware extensions/libraries for Mindstorms NXT have been developed such as RWTH [14], cliRobust [167], and JCSPre [72].

RWTH — Mindstorms NXT toolbox for MATLAB is one of the few available 4th generation language extensions for Lego Mindstorms. RWTH is similar to having an application in which LeJOS commands can be written and run without compilation, whereas this project uses the capabilities of LeJOS to provide an abstraction of a robot in NetLogo instead of recycling LeJOS commands into another language [14].

cliRobust — Clojure Programming API for Lego Mindstorms NXT [167] – has been designed around clojure, which is a modern, concurrent dialect of LISP. The purpose of this extension is to allow control applications to be written in clojure. Extensions/libraries such as these have been designed to allow parallel programming elements to be used to control Mindstorms robots.

JCSPre – JCSPre [72] bears more resemblance to this project as it is also based on LeJOS firmware for NXT robots, but it links the robot and its LeJOS abstraction to a reduced version of the parallel programming environment JCSP (Communicating Sequential Processes for Java). Similar to cljRobust, this has been designed to provide a port for LeJOS functionality directly into the JCSP language.

3.2.2 Lego Mindstorms Robotics

The first version of the Lego Mindstorms kit was released in 1998, with subsequent versions released in 2006, 2009 and 2013. Lego Mindstorms comes with pre-installed firmware. However, a custom firmware can be installed to achieve the required functionalities. Because of its availability to the public and its relatively cheap price for a robotics kit, several interesting robots have been developed by the community. Two well-known ones are a Rubik's cube solver and a Sudoku solver. These puzzle solvers demonstrate an application of the cognitive approach to AI and also demonstrate the flexibility of the Lego Mindstorms kit, and the capabilities of the sensors included with it. The fact that a robot with a light sensor on a sweeping arm can accurately scan and read numbers on a piece of paper opens up many possibilities for completely autonomous input, even if it is a relatively slow process.

3.2.3 NetLogo

NetLogo is a programmable multi-agent modelling and simulation language with a wide range of practical uses [158]. It is free to download and provides a large number of example models which demonstrate the level of processes and behaviours that can be simulated. Its drag and drop interface makes it easy to use, create and monitor programs (called 'models') in real-time. Being an agent-oriented programming language makes it a uniquely interesting port destination for a robotics kit.

3.2.4 LeJOS

LeJOS is the custom firmware and API used in this project to program the control file on the robot and the communication in the extension. LeJOS is

a lightweight Java replacement firmware for Lego Mindstorms NXT and RCX robots. LeJOS has its own JVM (Java Virtual Machine) which allows robots to be programmed in Java. The advantages of using Java are twofold:

- Java provides cross-platform system portability and NetLogo is also free and portable.
- NetLogo is written in Java, meaning that NetLogo and LeJOS can be linked with relative simplicity.

LeJOS flexibility and scope has already been used widely. For example, a team from Portugal [19] successfully used LeJOS to implement subsumption based roaming and environment mapping on a Lego Mindstorms robot. They linked the control file (Robot) and Java program (Laptop) via Bluetooth.

The next section describes the design and implementation of the system and the files and commands which make up the extension.

3.3 NXTLogo: Design and implementation

The purpose of this middleware is to make the system heterogeneous, so that it works on all platforms and with all robot configurations, and to make the front end intuitive and easy to develop, without the user needing internal knowledge of the system or the processes involved. In order to achieve that, a great number of challenges need to be tackled through system design. The system needs to be efficient, configurable, dynamic and reliable while at the same time maintaining a simple, user-friendly front end. There needs to be an emphasis on the usability of LEGO Mindstorms extensions to enthusiasts without specific 3rd generation programming skills. Doing away with complicated programming, installations, and compiling should open up development to people and parties less directly connected to the field.

The design of the Middleware (see Figure 3.1) is divided into two parts: the Java extension to NetLogo; and the LeJOS Java control file which runs on the NXT. The extension handles the channelling and conversion of

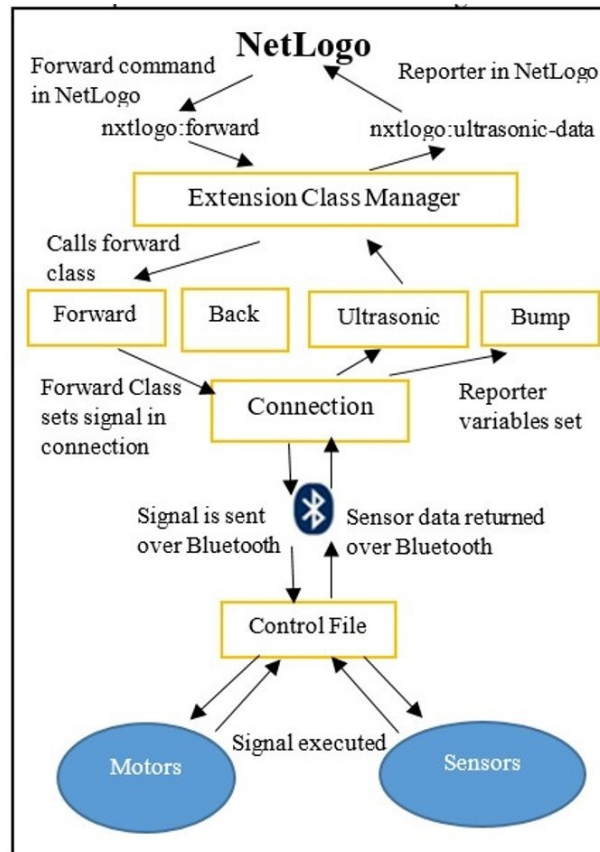


Figure 3.1: The design for the middleware based on some sample classes in the extension.

data both to the NXT over Bluetooth and to the extension via NetLogo reporters, and does no explicit processing of the data itself. The control file is the other end of the Bluetooth connection created in the extension, and runs on the LeJOS firmware on the NXT. It listens for incoming commands (forward, backward, turn-right, turn-right, and stop), executes these commands and returns feedback from all sensors (Ultrasonic sensor, Bump Sensor, colour sensor).

3.3.1 The Control File

There are many challenges in designing an efficient control file such as returning sensor data frequently, non-blocking, and data transfer. LeJOS used to run on a NXT robot via a control file. However, several steps are required in order to compile a program to run on the NXT.

Firstly, the written program needs to be compiled into a .class file by the LeJOS NXJ compiler, nxjc, and then needs to be converted into a binary .nxj program

to run on the robot. To achieve this, the program needs to be independent of its references and the wholly independent unit needs to be compiled into a .nxj binary instruction file using the nxjlink utility. Afterwards, the file is uploaded to the robot via Bluetooth or USB and executed as instructed.

The control file implements the LeJOS abstractions of robot peripherals and sits between the extension and the peripherals. For example, a pilot (NXT Brick) is used to control two motors, a colour sensor, an ultrasonic sensor, and two touch sensors. The sensors return their abstractions (boolean for the bump sensor, distance from 0-255 for the ultrasonic sensor, and three integer RGB tuple for the colour sensor) and the pilot sends back a boolean indicating if the robot is moving, the associated tachos count, and the approximate real turn angle after any given manoeuvre, and resets itself. The provided information is enough to allow simulation corrections in NetLogo which means we can move a turtle representation of the robot in the NetLogo simulation in a similar manner as the robot has moved itself.

The main polling loop of the control file is shown in Algorithm 1. In order to send back the sensor data, every polling loop ensures a constant data stream besides making sure that sending and receiving is synchronised at both ends which is a difficult task. Synchronisation is one of the biggest issues that needs to be overcome in order for the robot to be able to generate a relative map of its environment upon which it can later rely. A switch statement is used to process the signal in case anything other than NO-ACTION is received, but everything else is designed using LeJOS's instant return arguments in motor calls which avoids using while loops or hanging the program. This allows the sending of sensor data even while the robot is moving forward.

This architecture allows the sending of data at the end of every iteration of the main while loop without blocking, hanging, or pausing the robot. This fast and regular return of the data is necessary for visualising the sensors as event streams, otherwise, events can be reacted too slowly or missed completely.

Algorithm 1: Main Loop of the Control File

```
1 while signal != terminate do  
2   switch signal do  
3     case stop do  
4       stop robot;  
5       setsignal = NO-ACTION;  
6     case foward-constant do  
7       robot keep going forward;  
8       setsignal = NO-ACTION;  
9   send sensor data;
```

In an ideal world, sensor data would be placed in an array; however, available I/O streams (OutputStream and DataOutputStream) in LeJOS only support primitives which means output has to be sequential for each sensor. The former is able to send bytes of data (integers) one at a time which is not particularly useful because tachometer and motor turn-angle estimations are floating-point numbers. Although the later supports all the primitives (boolean, integer, float, double), it does not support an array. This leaves us with two choices:

1. Send back the numbers from each sensor individually and deal with them sequentially at the extension.
2. Write a serialising method to run on the robot which has limited power and memory.

Our current solution is sequential. However, this may complicate things if we want to extend the project to involve multiple robots. It is important to have separate input and output for each robot so that the source of the data can be determined.

3.3.2 The NetLogo Extension File, NXTLogo

The variables in the extension called NXTLogo are in the connection class and in the reporters. In the connection class, two methods are used to set the signal, buffered and checked. In buffered, if two different signals are sent at the same time, the second to arrive is placed in the buffer to be executed

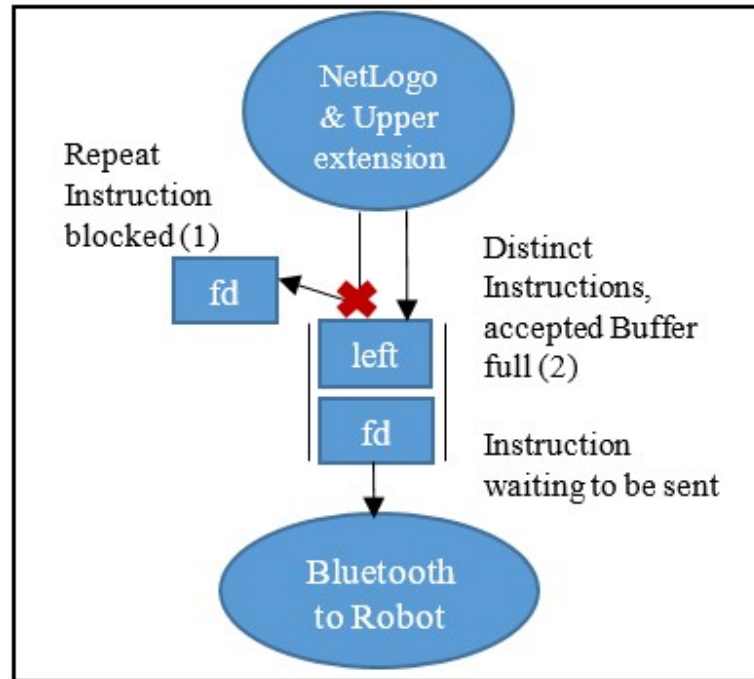


Figure 3.2: The Control File Instruction Buffer.

next. The buffer size is variable but two seems to operate best where the second signal is discarded if identical to the first one. In order to update the sensor streams frequently, the NetLogo simulation needs to keep executing which generates unwanted repeat instructions. Without discarding, the buffer would fill up quickly and the robot would not move as expected. Once the instruction is executed and removed from the buffer, a new one can take its place. In this way, repeat instructions can still work if required. However, they will not stack up into a backlog (see Figure 3.2). The checked method can accept integers from 1 to 359, and sets a variable as such which tells the robot how far it is expected to turn when given a turning command.

Instructions are each defined by a class and linked to an appropriate command in NetLogo. When an instruction is given in NetLogo, its defined class in the extension passes a representative integer to the signal setting method which checks the instruction duly and places it in the queue to be executed if accepted. Whenever the main program loop is repeated, the signal buffer is peeked (first element is investigated but not removed). If peek is null, the NO-ACTION signal is sent, otherwise, if peek is not null, the head of the queue is removed and sent to the robot along with the current turning angle.

These systems are synchronized by blocking each other. For example, the extension sends a signal and waits to hear a response from the robot to make sure the signal has been executed and the loop has been passed and the robot sends a response after executing and waits for the next signal. The steps between a command being sent and executed are shown in Figure 3.1.

3.3.3 Implementation and Class Breakdown

The extension consists of 15 Java classes with most of them using standard practice when writing a NetLogo extension where every extension command to be called in NetLogo needs its own class. All of these command classes are referenced by the extension class manager file, which tells NetLogo which command is linked with which class. Commands and Reporters are two types of classes called by NetLogo. Commands do not return any information and are only capable of executing codes whereas reporters return data when called. The currently available commands in NXTLogo are listed and briefly summarized in Table 3.1.

Each of the available NetLogo commands in Table 3.1 has a respective class in the NXTLogo extension. Most of the commands just pass data to or from the robot whereas the `get-errors` command and its associated java class make debugging much easier. Errors are stored in a buffer upon detection and can be accessed one-by-one from NetLogo with the `get-errors` command.

This extension architecture is sufficient for delivering the basic functionality required of the system. The control file and extension as described can perform connection and two-way Bluetooth communication between the external systems: NetLogo and Lego Mindstorms Robots. The robot can be completely controlled by NetLogo and it streams back real-time information not only from the sensors, but feedback from the actuators as well.

Table 3.1: Commands currently available in NXTLogo.

nxtlogo:connect <String robot-name>	Connects to the specified NXT.
nxtlogo:forward	Moves forward an arbitrary unit of distance.
nxtlogo:forward-until-bump	Moves forward until told to stop or bump sensor hit.
nxtlogo:backward	Move backward an arbitrary unit of distance.
nxtlogo:turn-left <int angle>	Turns left the specified angle.
nxtlogo:turn-right <int angle>	Turns right the specified angle.
nxtlogo:stop	Stops the NXT's current action.
nxtlogo:bump-data	Reporter for the touch sensor.
nxtlogo:colour-data	Reporter for the colour sensor.
nxtlogo:ultrasonic-data	Reporter for ultrasonic sensor.
nxtlogo:angle-data	Reporter for the approx. angle turned last manoeuvre.
nxtlogo:tacho-data	Reporter for the wheel turns last manoeuvre.
nxtlogo:moving	Reporter for whether the NXT is moving.
nxtlogo:get-errors	Reporter pops the top error off the stack trace, if any.
nxtlogo:terminate	Terminates the connection.

3.4 Sample NetLogo Models

In this section and its subsections, sample NetLogo models are designed and implemented using the new NXTLogo extension. The sample models are NetLogo simulations using a single robot in order to show some of the scopes of the system.

3.4.1 Line Following

Line following is a complicated problem as it requires analysis of the colour sensor data to detect the change from the background colour. Line following with one colour sensor is not an optimal solution and it is best implemented with two sensors, one on each side of the line [164]. It is assumed that we are following a looped path which means we can turn in one direction. The sample program can also be used to keep the robot in a specific colour area.

It is assumed that the robot is set up with a downward-facing colour sensor to measure in RGB 0-255 integers the colour of the floor and the line. The model is set up by running a control file on the robot and connecting to it in NetLogo. The robot should be placed with the colour sensor exactly over the line before the robot can follow the line and a button on the interface is pressed to learn the colour of the line. After deciding the line to be followed, the turn-angle (according to directions) and tolerance to the colour change settings (according to light settings) are adjusted before pressing the go button in the interface to run the NetLogo program.

This model contains just 50 lines of code which is reasonable for its accomplishments: line following and staying within a specific colour area. In order to illustrate the simplicity of the final NetLogo code used to control the robot, an extract of this code is shown below.

Listing 3.1: Simplicity of NXTLogo Line-following Code.

```
1 to go
2   ask turtles
3   [
4     set color using nxtlogo:colour-data
5     if color within tolerances
6     [
7       if (algorithm = 'left turn')
8         [ left-turn ]
9         [ right-turn ]
10    ]
11    [
12      nxtlogo:forward-until-bump
13    ] ; go forward
14  ]
15 end
16
```

```

17 to left-turn
18     nxtlogo:stop
19     nxtlogo:turn-left turn-angle
20 end
21
22 to right-turn
23     nxtlogo:stop
24     nxtlogo:turn-right turn-angle
25 end

```

The code for this model has just six procedures (some of these are used above) which are summarised in Table 3.2.

Table 3.2: Six procedures of the line following model.

connect	Connects to the specified NXT robot.
set-line-colour	Saves the current colour under the sensor as the colour to follow.
check-colour	Displays the current colour under the sensor without moving the robot.
go	Follows the line: moves forward by matching the current colour with saved colour, and if not it turns in the stated direction. Displays the current colour in the environment.
left-turn	Turns the robot left the specified number of degrees.
right-turn	Turns the robot right the specified number of degrees.

3.4.2 Subsumption Architecture Roaming

The purpose of this model is to show that the system is well suited to different AI paradigms, in this case, the subsumption architecture [22]. To control a robot, subsumption architecture employs a layered approach, with behaviours of different priorities subsuming one another. In other control systems for Lego Mindstorms Robots, this might be difficult to envision and implement, but in NXTLogo and NetLogo, it is very straightforward. The main loop of the program as defined in the go procedure is as follows:

Table 3.3: Five procedures of the subsumption roaming model.

connect	Connects to the specified NXT. Uses possible future convention for multi-robot simulations by keeping a record of the connected NXT in a hidden turtle. Keeps track of connected robots via simulation.
go	Executes layers of behaviours in order of subsumption. Reflexive behaviour has priority.
avoid-obstacles	Highest priority behaviour. Turns away from an obstacle if the ultrasonic sensor value is below a certain threshold.
stay-on-this-colour	Second priority behaviour. Turns if the colour sensor value is too far outside the tolerance.
explore	Lowest priority behaviour. Moves the robot forward unless the bump sensor is being pressed or another manoeuvre is attempted.

Listing 3.2: Main loop of the Subsumption Achitecture Roaming NXTLogo Code.

```
1 to go
2   avoid-obstacles
3   if not obstacle-detected?
4   [
5     stay-on-this-colour
6     if not colour-change?
7     [
8       explore
9     ]
10  ]
11 end
```

This code is inspired by the Myrmix pseudocode for a basic subsumption architecture robot [23]. The subsumption roaming model contains five procedures. These are summarised in Table 3.3.

By defining procedures for each behaviour, the main loop of the model becomes very easy to understand and can be implemented in close to natural language. In this case, the robot will prioritise the avoidance of physical obstacles, then try to remain on the same floor colour before it can move forward. This simple layered approach to behaviour means that we can see

that apparently complex environment-aware roaming behaviour is the sum of three very simple behaviours, a hallmark of the subsumption architecture.

3.5 Results

In this section and its subsections, the performance of the new robotic system environment is analysed and improvements are suggested. The different environments are developed with specific goals to test the performance and capabilities of the new robotic system environment.

3.5.1 Line Following

The line following program meets with mixed success based on a number of factors including but not limited to:

- Lighting conditions
- Remaining battery power
- Colour tolerance
- Excessive turn angle

The turn angle was set to 11 and the tolerance to 20. Based on the low light conditions, these settings were inadequate for line following in the environment the robot was situated in. A lower turn angle of five degrees was set to avoid overturning and higher tolerance of 42 was set to make the robot turn earlier when it is not completely over the line. The adjustment of these settings allowed for success as shown in figure 3.3.

A set of ten runs was conducted with the same settings as described above and the criteria of success for the robot was to make a complete run without any assistance. The model was successful in six out of ten runs.

Because of the factors which can affect the program's success, it is difficult to draw solid conclusions from these results, but certain trends are noticeable.

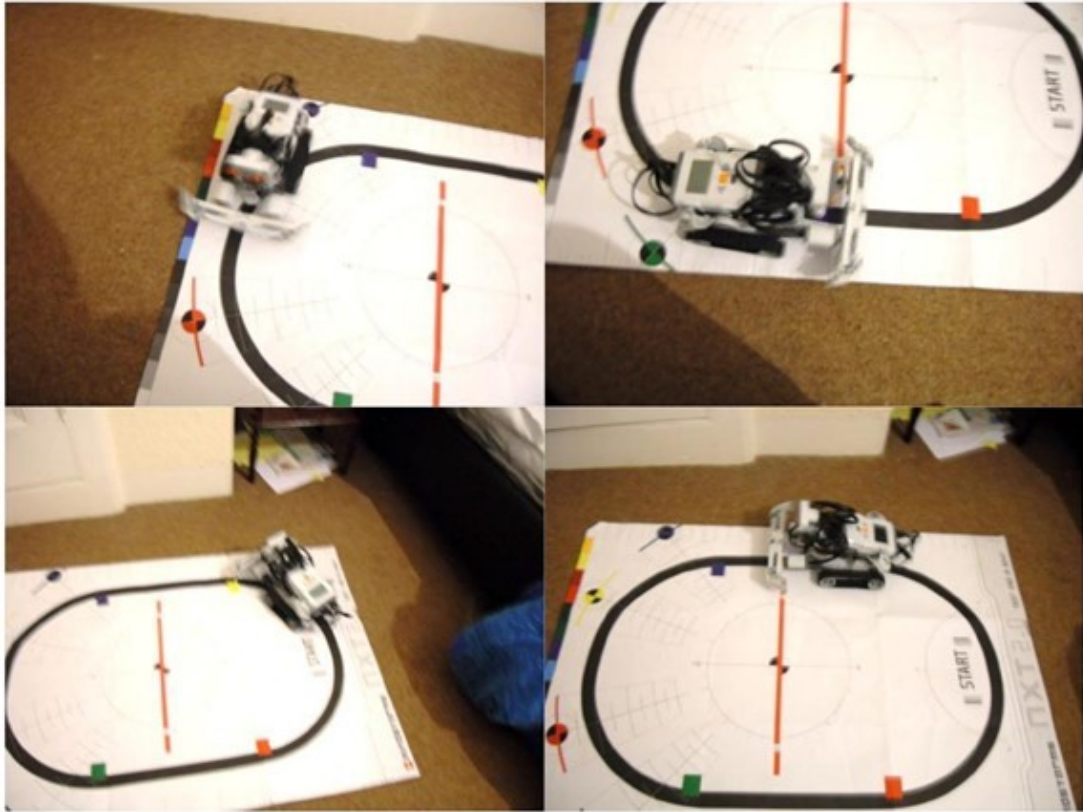


Figure 3.3: Successful execution of the line following behaviour.

For one, this model relies upon one-way turning, so overturning is the most common reason for a failed run. The problem can be overcome by setting a lower turn angle which may slow down the robot in corners but will increase accuracy. The introduction of turning while moving would benefit this model; currently the system can only process one instruction at a time but future versions may include this capability easily by varying motor speed on the robot.

3.5.2 Subsumption Architecture Roaming

In the first attempt at running this simulation, the settings described in the previous section were found to be inadequate for the conditions and needed to be changed. The robot turned before reaching the black coloured section of the floor in the test environment (see Figure 3.4) when there was no ultrasonic obstacle any detectable distance in front of it, suggesting that the chosen colour tolerance value was set too low. A colour tolerance of ± 20 means that if any of the three values (R, G and B) change by 20, the robot is told to turn and correct itself. After increasing the colour-tolerance value to 30,

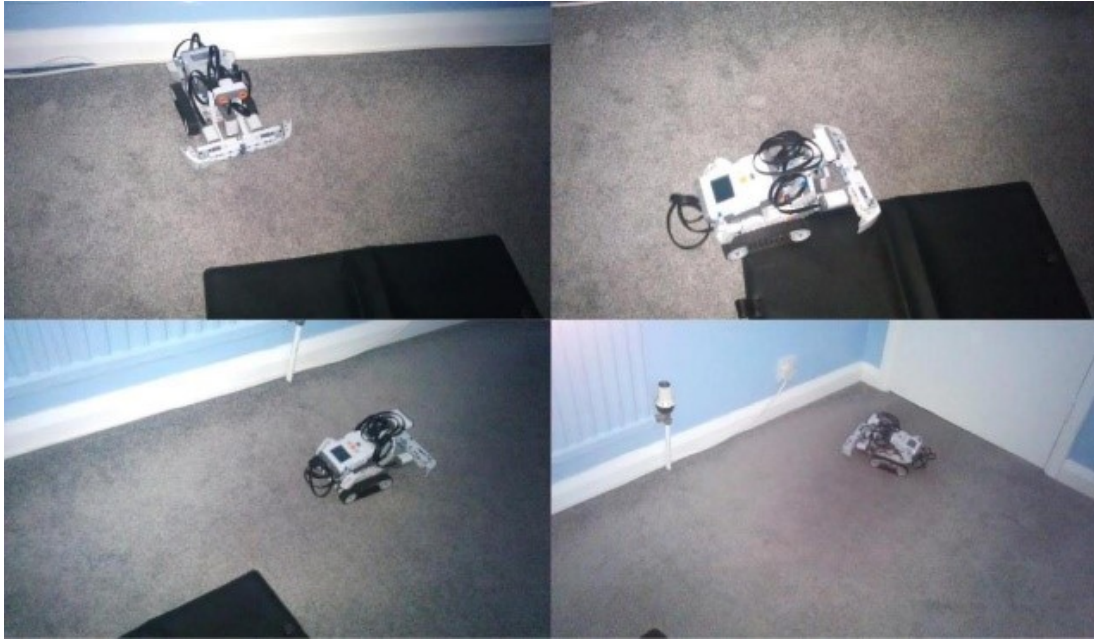


Figure 3.4: The roaming ability of the robot is illustrated by it staying on the same colour and avoiding obstacles which are represented by solid colours.

the robot could explore, turning when it detected the black coloured section of the floor and the far wall as shown in Figure 3.4. This exploration is an emergent property of the combination of the three very simple behaviours implemented by the model.

Further experiments with different tolerance levels provided mixed results where a higher tolerance of 45 could not even detect the black coloured section of the floor. However, this didn't falsely identify the wrong patch as a black coloured section of the floor. The best results were achieved by using a colour tolerance of 30.

It is worth noting that these runs were conducted on the carpet, which can yield varying results for a colour sensor based on wear and light. If the runs were conducted again on a smooth matte floor, better results would be expected.

3.6 Conclusions

This chapter has described the design and implementation of a middleware for Lego Mindstorms NXT robots where the robot can be controlled and linked

directly to simulators in the agent-oriented programming language, NetLogo. The technologies that the middleware relies on are heterogeneous since it's built in Java which runs on the JVM on all platforms as does NetLogo, being free to download and able to run on any platform. Also, the front end of the system is written in NetLogo, which is known for its simplicity of code. With regards to the Lego Mindstorms NXT hardware, sensor streams are implemented for all three native NXT sensors (touch, ultrasonic, colour). The middleware does not require any third-generation programming from the end-user in order to implement behaviours on the robot. Significant results can be achieved through minimal code and drag-and-drop interface creation for the controlling of simulations.

This chapter has also explored the feasibility and advantages of linking Lego Mindstorms robots with an agent-oriented multi-agent simulation language. Its significant contributions include the following:

- It has introduced a new and user-friendly way of implementing cognitive and reactive behaviour on a mobile robot built from the Lego Mindstorms NXT 2.0 kit.
- It has demonstrated that the capabilities of other Mindstorms middleware extensions/libraries can be recreated in the system using relatively modest amounts of code.
- And it has provided the means for some of the extensive NetLogo capabilities to be adapted to feature real-world agents and therefore laid the foundations for further research and development using NetLogo and Lego Mindstorms robots.

While the basic functionality of the extension is satisfactory, there are areas in which the work could be extended; for example, the extension of the system to involve multiple robots. A series of control files based on multi-threading can be implemented for the integrity and efficiency of the system. However, its effect on Mindstorms NXT and efficiency needs to be investigated before making further conclusions.

Chapter 4

Evolving Robotic Behaviours using Grammatical Evolution

This chapter is based on Paper 2 mentioned in the publication section of this thesis (section 1.6). Its purpose is to fulfil objective 2 *"To develop an integrated low-cost robotic environment which is open-source and extendable for evolutionary robotics experiments"* in order to partly help answer the first research question *"Can a low-cost robotic system be developed which is open-source and extendable for performing evolutionary robotics experiments?"*. The next section provides background to this chapter. This chapter builds on the work from the previous chapter and reports on experiments that were conducted on the use of Grammatical Evolution [108], [31], [135], [61], [21], [7], [85], [125] applied to the problem of evolving behaviour for the task of maze exploration. The robotic system environment developed in the previous chapter was used along with a series of NetLogo applications focusing on evolving exploratory robotic behaviours using Grammatical Evolution (GE). A low-cost platform in NetLogo for the evolution of behaviours for Lego Mindstorms robots was created which can produce a solution for robots to search for an abstract goal while navigating the environment.

The approach was evaluated using three different environments with a robot inserted into these unknown environments without any internal memory mechanisms. The performance of individual robot was evaluated using a fitness function to determine the fittest individual. The fittest individual's phenotype was stored and extracted to a Lego Mindstorms robot in order to

test them in the real world. The results showed that solutions developed for the complex environment outperformed all other solutions.

4.1 Introduction

Exploratory robotics has become one of the most popular and interesting areas in Robotics. One example project is the Mars “Curiosity” Rover, which is helping with the exploration of Mars [172]. This chapter will investigate a low-cost solution for rapid prototyping purposes for exploratory robotics by developing a low-cost robot capable of searching for an abstract goal while navigating the environment. This will be conducted along with a series of NetLogo applications focusing on evolving exploratory robotic behaviours using Grammatical Evolution (GE).

Several components were combined to achieve the following objectives:

1. Designing and constructing a Lego Mindstorms robot for experiments.
2. Developing NetLogo applications to evolve solutions for exploratory robotic behaviours to be used in a variety of environments.
3. Developing an interface to facilitate the extraction of the evolved solution to upload to the Lego Mindstorms robot.

Evolutionary robotics research has investigated addressing the issue of generalisation. Instead of developing specific solutions for particular problems, the purpose is to develop solutions that are useful to various scenarios [101]. In this chapter, Grammatical Evolution (GE) is applied to the problem of evolving behaviours for the task of maze exploration. A series of experiments is performed to find a suitable solution which can solve several navigational mazes.

GE was originally introduced by Ryan et al. in 1998 [135] and was improved by the same group. GE evolves solutions for given problems by combining a genetic algorithm and a context-free grammar [58].

The next section (section 4.2) discusses the design of the Lego Mindstorms robot. Section 4.3 discusses NXTLogo (NetLogo Extension). The NetLogo evolutionary application, the fitness function and testing are discussed in subsections of 4.3. Section 4.4 describes the additional NetLogo application which is created to facilitate the easy and modular design. Section 4.5 briefly describes the Mindstorms Control File used to control the robot. A real-world implementation and testing are described in section 4.6 followed by evaluation in section 4.7 and conclusions in section 4.8.

4.2 Design of Lego Mindstorms Robot

One of the first thing developers need to think about is to decide the robot to start with. Besides being expensive, domestic robots are very popular as you can build your own unique design. Commercial robots can also be used; however, it requires considerable work on libraries to use the robot in certain areas and the design is not easy to customise [83]. Lego Mindstorms programmable brick and Lego pieces allow you to design the robot of your choice and it is easy to customise because of its Lego structure. This provides an ideal system for advanced software development because of its specifications such as Bluetooth communication that can provide communications up to 30 feet, and the ability to send commands to actuators and receive feedback from sensors [73]. Lego Mindstorms comes with pre-installed firmware. However, custom firmware can be installed to achieve the required functionality. Because of its availability to the public and its relatively cheap price for a robotics kit, several interesting robots have been developed by the community. The fact that a robot with a light sensor on a sweeping arm can accurately scan and read numbers on a piece of paper opens up many possibilities for completely autonomous input, even if it is a relatively slow process [9].

In the real world, a robot can perform only those tasks which are allowed by its physical construct regardless of evolved and robust codes [159]. Keeping this in mind and with the aim of our project being that the robot will be inserted into an unknown environment without any internal memory mechanism,

several designs were studied. The most suitable for our task was chosen which is the NXT bumper car design. The original design of this robot had a motorised body and a bump sensor which was upgraded and equipped with an extra ultrasonic sensor and an RGB colour sensor as shown in figure 4.1 below.

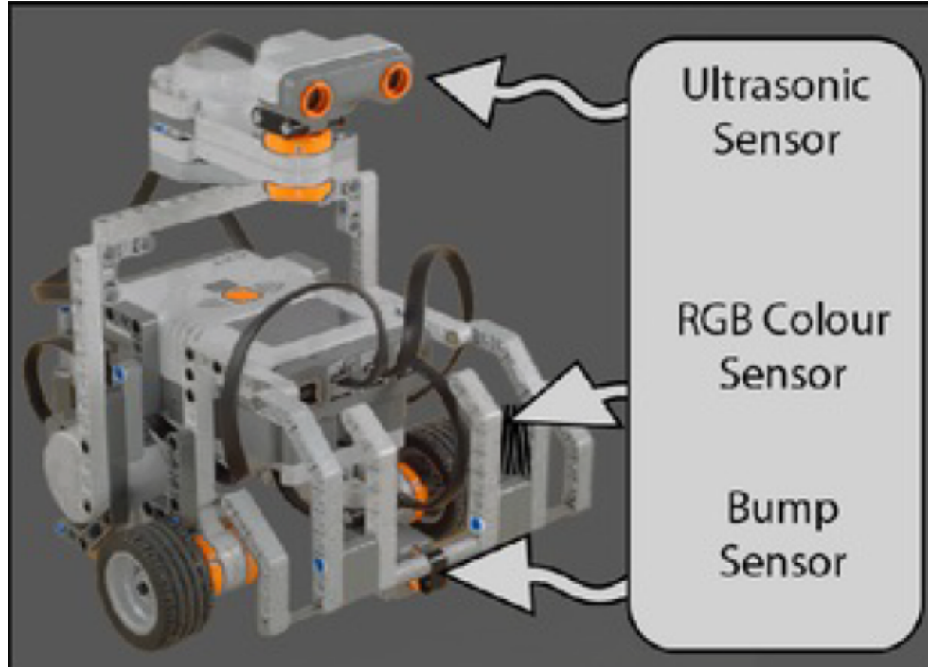


Figure 4.1: The upgraded design of the bumper car for our low-cost Lego Mindstorms robot “Explorer”.

To justify the design, the details of the components are explained below.

4.2.1 Ultrasonic Sensor

When the robot is placed in an unknown environment, it may break down and be left stranded. To reduce the risk of breakdown, it is necessary to minimise the contact of the robot with obstacles [153]. Therefore, the robot uses an ultrasonic sensor to sense the obstacles ahead and take preemptive action to avoid that obstacle. It was a difficult task to set the threshold at which a distant object becomes an obstacle; however, experimentation showed that 10cm works well for the robot. The robot displayed the qualities of self-preservation and increased reactivity to obstacles which are highly desired in a robotic system.

4.2.2 Bump Sensor

Unfortunately, it is not guaranteed that the ultrasonic sensor will protect the robot 100% of the time as it does not detect obstacles which are higher or lower than the position of the ultrasonic sensor [17]. Therefore, a bump sensor was also added to the robot which responds to actual collisions. A long barrier is hinged to the robot which triggers the sensor whenever the robot hits the wall and protects the actual sensor from any damage while increasing its width. The bump sensor width is only 2cm and without long barriers, so it may not detect the collision when the robot hits the wall in corners.

4.2.3 RGB Colour Sensor

After making sure that the robot can navigate the environment safely, the next important task was a mechanism for exploration to find specific objects (i.e. goal-seeking behaviour). Throughout this project, the object being searched for was a coloured shape that lay flat on the floor. By attaching an RGB colour sensor to the robot, this allows it to constantly check for the coloured shape and stop when it has been reached in order to avoid wasting any further resources.

4.2.4 Motors

There are three different motors in the design, one for each wheel and the third to react as the neck for the robot where the ultrasonic sensor is attached. Motors power the robot to move forward and turn by working symmetrically and allow zero-radius turns by making one wheel go forward and another backward.

4.3 NetLogo Application for Evolving Solutions

It is a challenging task to evolve a robot's behaviour when it is negotiating within an environment [133], especially, when the device is power-hungry and has limited resources. Therefore, a NetLogo application was developed to debug and test the environment to evolve simulated solutions where agents

are embodied within a virtual environment. Later the fittest evolved NetLogo simulated behaviour will be used as the Mindstorms robot's default behaviour.

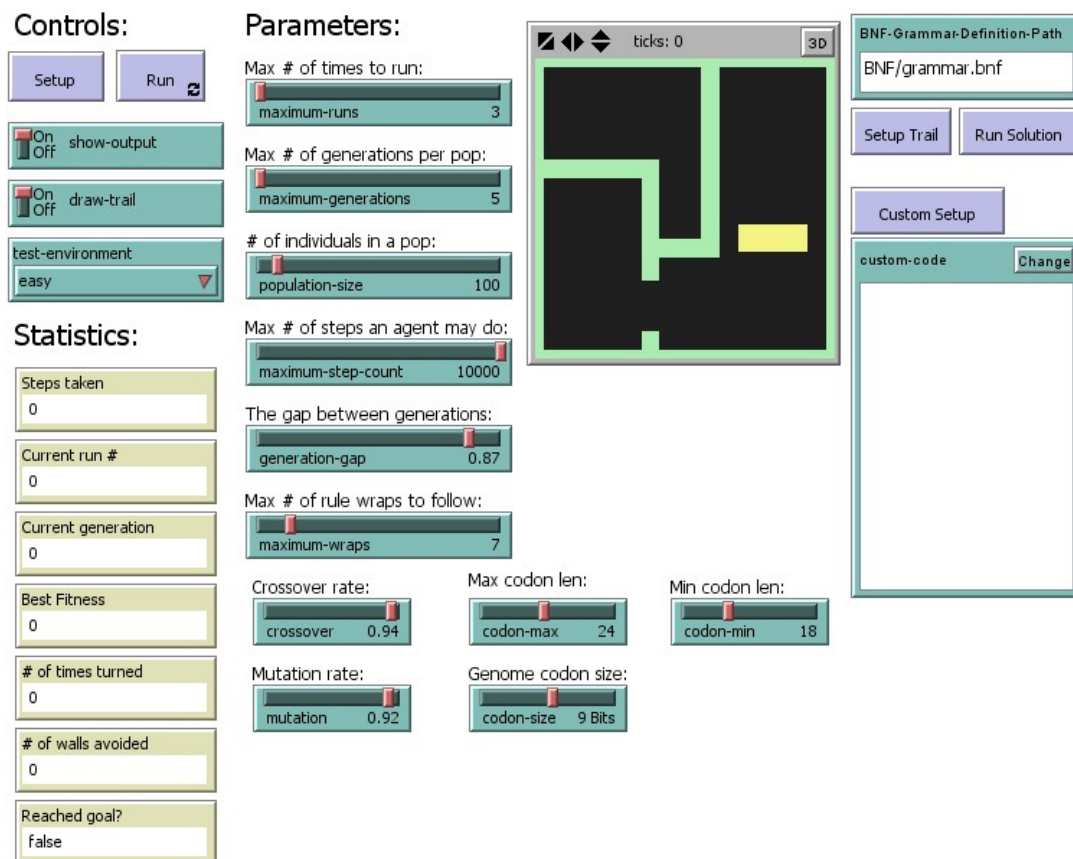


Figure 4.2: The interface of the application used to evolve the codes.

4.3.1 Application Interface

The “Mindstorms-Explorer” application is an implementation of the Grammatical Evolution algorithm which uses several components to allow users to select the desired configuration via using the interface (see figure 4.2). These configurations decide the state of the evolutionary process and evolve solutions accordingly. The application generates a population of agents and evaluates their performance by using a fitness function to decide the fittest individuals that produce the offspring for the next generation. Once the desired runs and generations are completed, the application extracts the fittest individual's phenotype (evolved solution) to the robot so that it can be tested in the real-world.

Most of the user interface controls consists of sliders which makes it easy to use and configurations can be changed by moving the slider from side

to side. After choosing the desired configurations, the draw-trail switch can be turned on to view the path taken by the robot. Then the user can decide between three environments (Maze I, Maze II, Maze III) for the sample application. Once the user has decided all the configurations, they can press the Setup button (top left), followed by the Run button (next to it, top left). Once the evolutionary process is complete, the fittest phenotype string will be displayed in the output box by the application. To see this fittest behaviour, the user simply needs to click the Setup Trial button and then click the Run Solution button.

4.3.2 The environments

It is possible that the evolved behaviour may work exceptionally well for a specific environment. Therefore, to monitor and mitigate the effects of this, the application used three different environment layouts [6]. The purpose is to check if solutions derived from one environment works in another. The three maze layouts are listed below:



Figure 4.3: The design and layout of the three mazes environments (Maze I, Maze II, and Maze III).

The basic rules are the same in all environments such as robots cannot cross walls (represented by green patches) while exploring the environment to search an abstract goal (represented by the yellow patch). The robots are not allowed to go through the walls, however, their reaction (turn left or turn right) is completely dependent on the evolutionary process.

4.3.3 The fitness function

The fitness function evaluates the performance of all agents and selects the fittest individuals to generate the offspring through crossover and mutation. In any standard evolutionary algorithm, this is a very important consideration that determines the final performance by evolving the solution (phenotype) because a bad fitness function will evolve an inappropriate solution which will lead to bad performance. Keeping this in mind, the fitness function was designed according to the desired behaviour (where a less amount of resources has been used in order to find the goal). The fitness function used is shown below:

$$B = \begin{cases} 5000 & \text{if goal-found is } true \\ 0 & \text{if goal-found is } false \end{cases} \quad (4.1)$$

$$f = (M_s - C_s) - (O_c \times 5) + B - (T \times 2) \quad (4.2)$$

where M_s is the maximum step count (i.e. the maximum steps the robotic agent can make), C_s is the number of steps the agent has taken, O_c is the amount of times an obstacle was detected, B is a bonus value used for rewarding the reaching of the goal state and T is the number of turns the robot performed, but not the angles turned.

According to equation 4.2, the best fitness is represented by a higher value. To reduce the effect from parameters on the fitness value, a bonus scheme of $B = 5000$ is introduced whenever an individual reached the goal which significantly exceeds all and values of (C_s , O_c and T), therefore ensuring that the goal-state will be deemed the highest fitness value.

4.3.4 BNF rules

The aim of the application is to evolve NetLogo code using grammar as per the Grammatical Evolution process. Therefore, Backus-Naur-Form notation [109] was used to define the possible syntax of the program code that could be evolved, and what rules governed the syntax. The recursive grammar

used in the application is shown below:

Listing 4.1: The grammar used to evolve robotic behaviours using Grammatical Evolution.

```
1 <expr> ::= <line> | <expr> <line>
2 <line> ::= ifelse (robot::obstacle-ahead?) [ <turn> ]
3 [ <move> ] | <turn> | <expr>
4 <turn> ::= robot::left | robot::right | robot::turn-around
5 <move> ::= robot::forward
```

These rules mean that code can be generated for the robot that includes avoiding obstacles and turning away (left or right) whenever an obstacle is detected. If no obstacle is detected, it commands the robots to move forward.

The reason behind splitting `move` and `turn` into separate groups is that the robot may choose to go forward while evolving commands. This will allow the agents to go through the walls and produce solutions which are impossible in the real-world. Therefore, splitting both of them can guarantee that only `turn` commands will be selected by the evolutionary process.

4.3.5 Testing

To test the application, 12 tests were conducted (4 in each maze). The goal was reached in all tests for each of the mazes. The results are provided in table 4.1 and the best results for each environment are highlighted in green. The columns of the table present the number of experiments, the environment used for that experiment, steps taken to reach the goal, number of turns taken while finding the goal, number of walls avoided while finding the goal, and whether the robot reached the goal, respectively.

The configurations used for the best results in each environment are provided below in table 4.2.

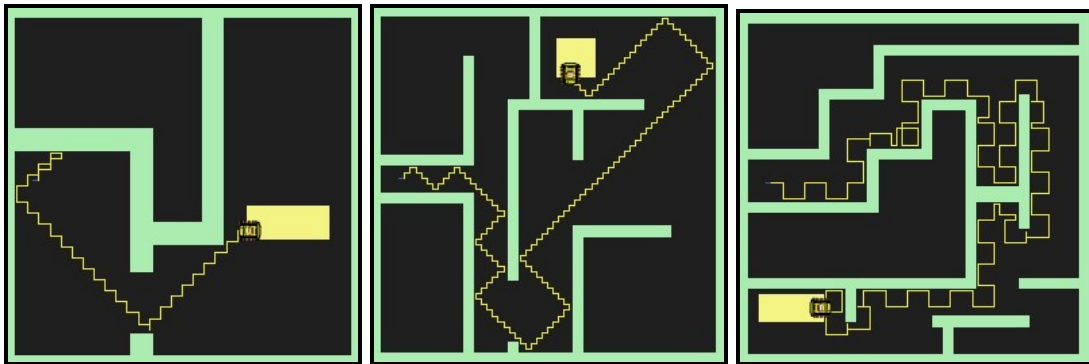
Table 4.1: The results achieved from 12 test runs.

Experiment	Environment	Steps	Turn Count	Walls Avoided	Reached Goal?
1	Maze I	157	60	6	✓
2	Maze I	212	108	2	✓
3	Maze I	181	79	1	✓
4	Maze I	244	101	4	✓
5	Maze II	605	173	64	✓
6	Maze II	571	113	42	✓
7	Maze II	311	110	7	✓
8	Maze II	509	267	13	✓
9	Maze III	489	146	48	✓
10	Maze III	547	154	63	✓
11	Maze III	456	101	36	✓
12	Maze III	318	81	41	✓

Table 4.2: The configurations used for best results in each experiment.

Experiment	Runs	Gens	Population	Gap	Crossover	Mutation
1 (Maze I)	3	5	100	0.87	0.94	0.92
7 (Maze II)	10	9	300	0.87	0.14	0.11
12 (Maze III)	22	20	700	0.89	0.90	0.97

Figure 4.4 also shows the behavioural pattern of each solution by having each agent draw the path it takes using the pen-down command in NetLogo.

**Figure 4.4:** The path taken by the best ‘Maze I, Maze II, and Maze III’ solutions.

An interesting result is that there does not seem to be any correlation between parameters. We see that 22 runs were used in Maze III to select the fittest individual, the highest number of runs used in our experiments. However, we see that the fittest individual selected after three runs in Maze I beats the performance of the fittest individual selected after 22 runs in Maze III.

Something intriguing from paths taken in all environments as shown in figure 4.4, is how each solution is well-suited to the environment. The figure shows that the paths each robot took are well-suited to the specific environment which means the robot may not perform well in other environments using the same solution. The resulting phenotypes are presented below.

Maze I Solution:

```

ROBOT::RIGHT  ROBOT::TURN-AROUND
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT ][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::TURN-AROUND]
[ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ROBOT::TURN-AROUND  ROBOT::TURN-AROUND
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT ][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::TURN-AROUND]
[ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ROBOT::TURN-AROUND      ROBOT::TURN-AROUND
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT ][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::TURN-AROUND]
[ ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]

```

Maze II Solution:

```

ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ROBOT::RIGHT
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT ][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::RIGHT][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT ][ROBOT::FORWARD]
ROBOT::LEFT

```

Maze III Solution:

```
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT][ROBOT::FORWARD]
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT][ROBOT::FORWARD]
ROBOT::LEFT
ifelse (ROBOT::OBSTACLE-AHEAD?) [ROBOT::LEFT][ROBOT::FORWARD]
ROBOT::RIGHT
```

An interesting characteristic of these solutions is that the longer the evolutionary process continues, the more refined the phenotype becomes, which is possibly over-fitted. However analysing the syntax, the code is adapted to a range of environments whereas the solution for the slightly easier maze (Maze I) which was generated after a couple of runs looks well-suited for the Maze I environment. This is interesting because phenotypes from early runs are expected to be more general and refined by evolving over and over.

These evolved solutions will be used to evaluate the real-world performance of the robots later in this chapter.

4.4 An Interface between NetLogo and NXT

An additional NetLogo application was designed to facilitate the execution of the evolved code between the NetLogo application and the Lego Mindstorms brick. Using the NXTLogo API described in the previous chapter, this program can connect to any Mindstorms robot and run any program generated through the first application. The interface of the additional application is shown below in figure 4.5.

This additional application provides an easy and modular design with several functionalities such as connection to the robot (second column second row first button), execution of commands, running behaviours (second column bottom button), termination of connection (second column second row second button), print state (button top left), and reset settings (button next to it, top left). Monitors on the right show the state of the robot, such as whether the

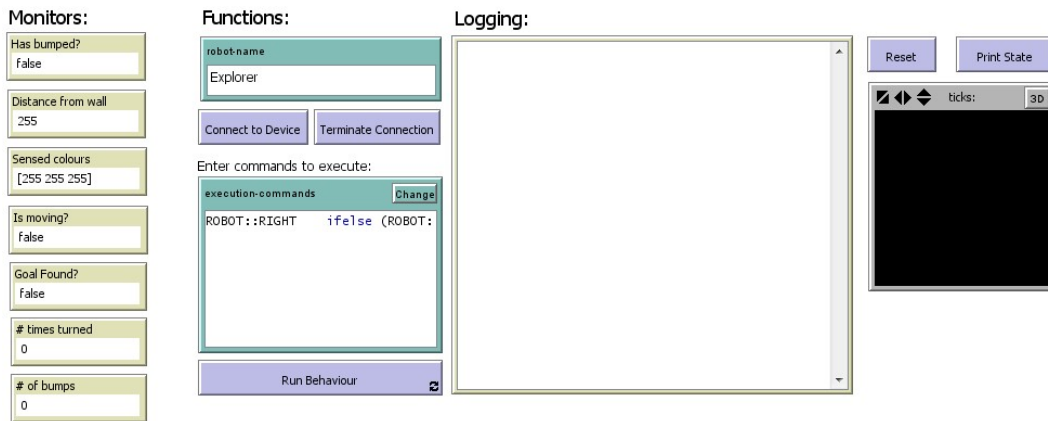


Figure 4.5: The UI of the application that connects between NetLogo and NXT.

robot has bumped a wall, the distance from the wall, the colour of the floor the robot is currently moving over, whether the robot is moving, whether the robot has reached its goal, the number of turns the robot has taken, and the number of times the robot has bumped into the wall. The `execution-commands` input box lets you enter the behaviour of the fittest individual generated earlier in the Mindstorms Explorer in order to investigate whether the robot can reproduce the same behaviour in the real-world. The logging output area shows the connection process and after that the current state of the robot. It also shows the statistic when you click the `Print State` button on the top right.

4.5 Mindstorms Control File

In order to execute the commands on the Lego Mindstorms robot and to receive feedback from the sensors, the `ControlFile` used in chapter 3 was adopted and upgraded.

How this program works is displayed below in figure 4.6. “Start” represents the opening of connection and flow of data stream between the NetLogo and the robot. This data is translated into commands which are executed by the robot. Whenever the robot changes its position, it sends data back to the application which usually contains information regarding whether the robot is moving, whether the robot bumped into anything, the distance from an obstacle, the colours that have been sensed, the distance travelled, and the total angle turned etc.

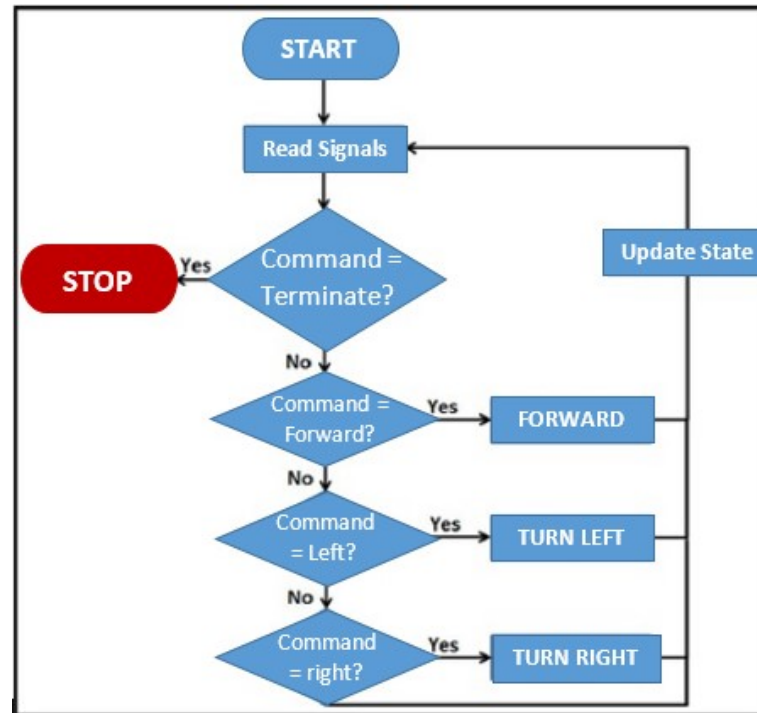


Figure 4.6: Basic Control flow for the NXT control file.

The NXT control file receives signals via the NetLogo extension and takes action according to the command signal sent. If it is `terminate` then it stops the robot, or if it is `forward` then it moves the robot forward, or if it is `left` then it turns the robot to the left or if it is `right` then it turns the robot to the right. Once the action is taken according to the command, the state the robot is in is updated and feedback is sent back via the NetLogo extension.

4.6 Real-World Implementation

In this section and its subsections, we will cover the real-world implementation of the robot and observations.

4.6.1 The Environment

The real-world implementation involved constructing one of the three environments. For our experiments, the Maze I environment was implemented in the real world (see figure 4.7). The reason behind choosing this environment is its simplicity.

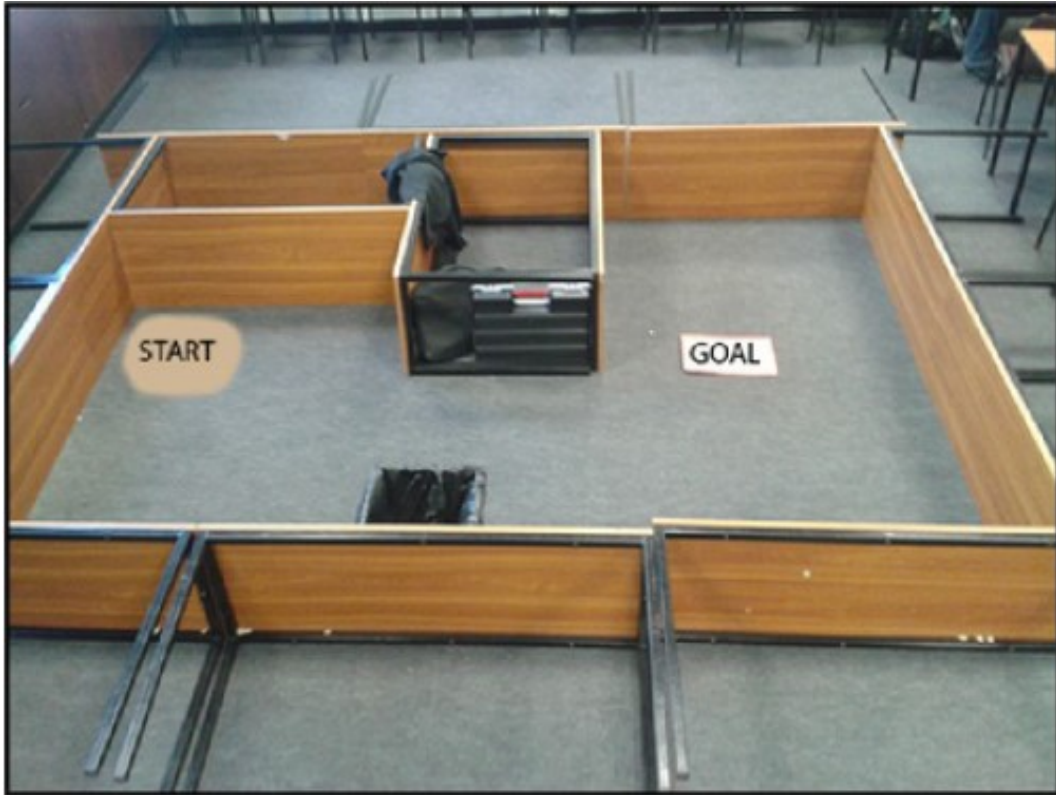


Figure 4.7: The real-world reconstruction of the Maze I environment.

The basic rules were the same for all experiments that the robot will start from the starting point highlighted in figure 4.7 and will finish at the goal location marked by placing a card on the floor. Even though the robot is in line with the goal location, it can not glide there straight as it is separated by obstacles and walls.

4.6.2 Three Experiments

The evolved solutions from each Maze environment were extracted to the robot respectively to test their performance in the real-world environment as described above. The results are as below.

Performance evaluation of 'Maze I' Solution

The Maze I environment solution (developed by the Mindstorms-Explorer application) was expected to perform best in this identical environment. However, its performance was opposite to what was expected. Sometimes the robot was unable to find the goal and it was not able to go past the first obstacle as it kept getting stuck in the corner where it started. However, most

of the time the robot reached the goal with the best time of 5.11 minutes. It is hard to know why the real-world experience did not reflect the virtual simulation to solve the maze quickly. The observed behaviour was not even close to the pattern reflected in the simulations (see figure 4.4) which proved that accuracy of reconstruction of the environment in the real-world had very little effect on the results.

Performance evaluation of ‘Maze II’ solution

Some good results were achieved using the solution generated to solve the Maze II as the robot was able to find the goal on all occasions within a reasonable time of under 5 minutes every time with the best time of 3.01 minutes. Considering that the program solution was developed for quite a different environment, this shows that evolving more general-purpose solutions have the ability to be better suited to overcome issues caused by a mismatch between simulations and real-world implementations.

Performance evaluation of ‘Maze III’ solution

The fittest program generated from the Maze III environment outperformed both the other solutions by quite a large margin as it only took 1 min 59 seconds for this program to lead the robot to the goal. The gradual performance increase from each experimental setup indicates as stated that a more general solution may be better suited to cope with different environments in the real-world. The Maze III solution was not expected to perform well in the Maze I environment because the evolved solution was generic, and the robot could have lost hours searching for a goal. However, the results showed the opposite as the Maze III solution performed best in the real-world test environment.

4.6.3 Summary

As discussed above and results presented in table 4.3 below, we can see that the best results were produced by the Maze III environment solution. One point to notice here is that the Maze III solution was generated after a

higher number of runs and generations than the other two solutions. There is a possibility that the performance of other solutions can be improved by using a higher number of runs and generations when evolving the solutions. Another factor which needs to be looked at is the effect of evolving in complex environments compared to the simple ones.

Table 4.3: The performance summary of evolved solutions in real-world experiments.

Solution	Found Goal?	Time
Maze I	40%	5.11 minutes
Maze II	100%	3.01 minutes
Maze III	100%	1.59 minutes

4.7 Evaluation

In this section and its subsections, we will further discuss how the evolved robots behaved because so far, we have discussed the results achieved from the simulations and real-world test environments.

4.7.1 Evolution & grammar

During all the real-world experiments, regardless of which solution is used out of the three, the robot hardly followed the patterns created in simulations. The Maze I solution couldn't even get near the marked location. The Maze II and Maze III solutions showed some better results, however, several times when the robot was heading straight to the marked location, it turned away and took longer to reach the goal location. Instead, if it had gone straight, the results would have looked extremely positive. The reason behind this could be that the simulations had tight paths with several turns. Also, it can be said that solutions generated in complex environments can perform better than solutions generated in simple environments, as is witnessed in the performance of the Maze I solution. Besides, the solution for Maze I was generated from a very low number of runs and generations, however, it performed best in simulations. This raises the question – What if we generate the Maze I solution by setting the configurations to a high number of runs and generations? Will it perform any better?

4.7.2 Experimental Observations

This section discusses further observations concerning the robot itself when compared to the observed behaviour and evolved solutions. For example, the robot could not detect the wall and ran right into it (see figure 4.8). The observations were even worse as the robot didn't stop and turned away, but it kept spinning motors a couple of times, even when it hit the wall.

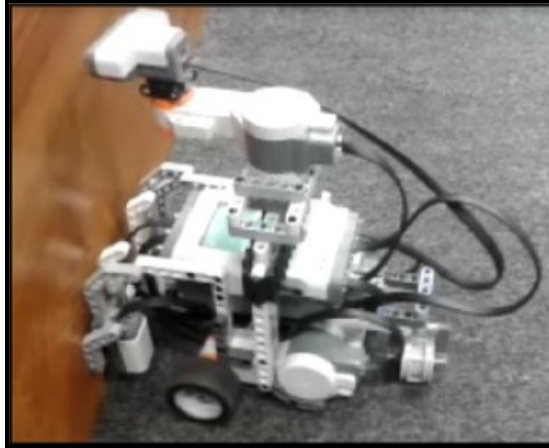


Figure 4.8: A robot bump despite the use of the ultrasonic sensor.

Another problem noticed was when the robot reached the marked goal colour, it kept moving forward (see figure 4.9) even after registering the goal state. Lastly, in the simulated code, only 90 or 180 degree turn angles were allowed, however, different turn angles were observed during real-world experiments.

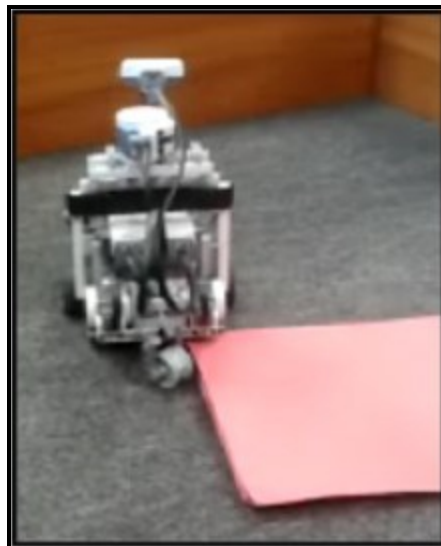


Figure 4.9: A delay between detecting the goal and acknowledging a goal.

Several issues can cause these problems including quality of hardware (Lego Mindstorms) and Bluetooth communication delay. Some of these problems

can be mitigated by using better hardware. The issue of the robot travelling into a wall was further investigated and the performance of the ultrasonic sensor was monitored by placing it against a solid object. The value changed whenever the object was moved closer or away from the object, however, jitter was noticed when moving the sensor in speed. The jitter combined with the delay of communication could be the cause that it kept going forward to a hit wall. Therefore, a bump sensor was added to avoid the robot being stuck against the obstacles. This also provides an explanation of the scenario detailed in figure 4.9 that after sending the acknowledgement back to the software, the robot kept moving forward until it received the next command from the software.

However, the main issue detected was turn angles being inaccurate. There can be several factors which can contribute to the problem such as the surface in the test environment, the power available for the motors, the design of the robot, and conflicting commands. An attempt to mitigate these issues will be made in the next chapter by utilising the knowledge and experience gained during these experiments.

4.7.3 Robot's Design Improvements

During the experiments, ultrasonic sensor was placed on top of the robot to facilitate the design and to avoid it from damage by hitting obstacles. However, it was observed that placing the ultrasonic sensor lower would detect smaller obstacles and robot won't get stuck.

4.8 Summary and Conclusion

This chapter reported the experiments that were conducted on the use of grammatical evolution applied to the problem of evolving behaviours for the task of maze exploration. The accomplishments of this chapter include the design and construction of a Lego Mindstorms robot; updating of a NetLogo extension; the creation of a NetLogo application to evolve solutions; and the creation of a NetLogo application to interface between the first application and the NXT Lego Mindstorms brick. The robot was designed and constructed

to examine the performance of behaviours in the real-world. The low-cost platform NXTLogo was updated to be compatible with NetLogo 6.1 API, in order to make use of NetLogo for the evolution of behaviours for Lego Mindstorms robots which can guide the robots to navigate an environment in search of an abstract goal. A series of NetLogo applications (focused on evolving NetLogo source code) was also used to investigate the use of Grammatical Evolution for exploratory robotics. For example, a NetLogo application was developed to grammatically evolve the codes for simulated agents embodied within a virtual environment.

The approach was evaluated using three different simulated environments and the robot was inserted into these unknown environments without any internal memory mechanisms. The application evaluated each individual of those robots with a fitness function to determine the fittest individual and stored the fittest individual's phenotype. Another NetLogo application was created as an interface to deploy the fittest individual's phenotype into a Lego Mindstorms robot in order to test it in the real-world. The results showed that solutions developed for complex environments outperformed all other solutions. Although the test environment in the real-world was like the simple simulated environment, the simple environment solution's performance was not good enough in comparison to the other environments' solutions. There is a possibility that the performance of other solutions can be improved by using a higher number of runs and generations when evolving the solutions.

Chapter 5

Discovery and Validation of Novel Interesting Behaviours

The purpose of this chapter is to address objective 3 *"To use the new environment to discover novel interesting robotic behaviours"* and objective 4 *"To use the new platform to validate novel interesting robotic behaviours in the real world"* of this thesis in order to help answer research question 2 *"Is the new robotic system design environment capable of automatically discovering novel interesting behaviours and validating these behaviours in the real-world?"*. The first half of the chapter is about the discovery of interesting behaviours and the second half of the chapter is about the validation of these novel interesting behaviours in the real-world. The first half also provides details of the robot design and the environment design to run the model, in order to discover interesting behaviours. These behaviours are also evolved, and a fitness function is used to find the most interesting behaviour. After discovering the interesting behaviour, it is also validated in a real-world environment. The results are discussed at the end of this chapter, followed by a summary.

5.1 Introduction

This thesis proposes an unsupervised method of discovering interesting robotic behaviours. Undoubtedly, a large number of simulations would be required to test this robotic system environment to check its accuracy of interestingness detection. A robotic system like this would save a large amount of time and labour cost, as it is nearly impossible for a human to

go through thousands of behaviours and find those behaviours that are interesting. Where a human might take several days to complete such a task, the goal is to develop a robotic system capable of finding interesting behaviour within a few minutes.

Discovering interesting behaviours is an emerging area of research for robotics. Currently, as stated in the literature review in section 2.7, this field is underexplored, but it can have a significant impact on decision making and autonomous exploration [119]. However, in the last few years, there have been various research concerning interestingness in different areas of robotics. Ahmed used Braitenberg vehicles to discover interesting behaviours by using clustering and compression [2]. Dhar et al. approximated the aesthetics and interestingness of images by using three hand-crafted rules, which are composition, content, and sky-illumination [34]. Wang et al. explored the problem of predicting interesting scenes for mobile robots, which is crucial for several practical applications such as autonomous exploration and decision making [165].

The next section and its subsections provide the design of the robot and environmental setup of the simulation experiments, which are followed by a discussion of results. The following section and its subsection provide the real-world environmental setup, experiments and results used for validation. The chapter concludes with a discussion of the results achieved, and future work is identified.

5.2 Experimental Setup for Discovering Interesting Behaviours

The discovery of interesting behaviours requires a large number of simulations with several different configurations and a design of a robot which can be remodelled in the real-world. A detailed design of the robot and the environment used for the virtual experiments are discussed below.

5.2.1 Design of Robot

To achieve the best results for these experiments, it is necessary to faithfully design the robot with functions which can be remodelled in the real-world. While designing the robot, it is important to keep in mind that the robot will be inserted in an unknown environment without any previous knowledge or training. Therefore, a robust design was required, which can be modified or upgraded easily while making sure the robot does not break down due to collisions.

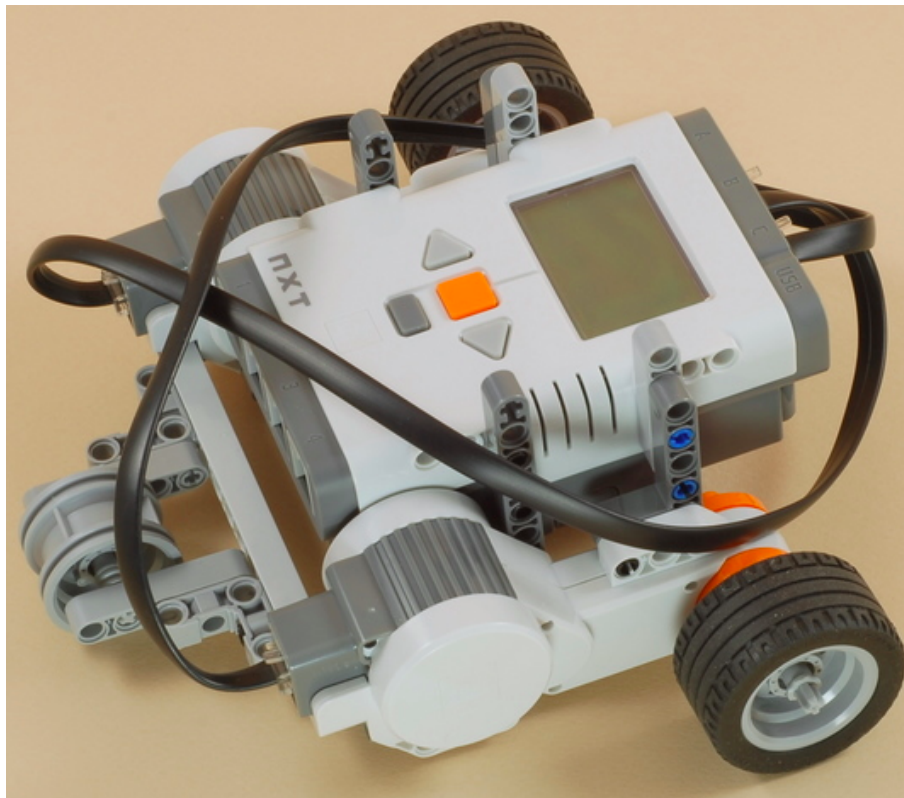


Figure 5.1: Five Minute Bot design from Lego Projects [120].

Several designs were studied from the Lego Mindstorms website, including bumper car, line following, Segway, mini rover, hammer car, explorer, and five minute bot [120]. It was decided to use the five minute bot design (shown in figure 5.1) and modify it according to our needs because the design is flexible for making changes such as adding another sensor, or moving sensors around.

The robot that was developed is named "*iFinder*" (see figure 5.2) and equipped with the following sensors to perform different tasks.

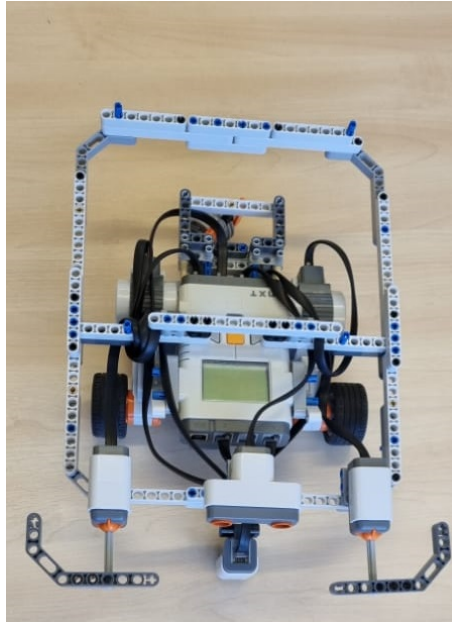


Figure 5.2: The adapted "Five Minute Bot" design used for the final experiments.

5.2.1.1 Colour Sensor

In these experiments, the robot reacts to the colour of the floor and chooses an action (left, right, forward, and turn-around) according to the floor colour. An RGB sensor is attached to the robot, which detects the floor colour continuously and then behaves according to the colour detected.

5.2.1.2 Motors

The design of the robot uses two motors (one for each wheel) to help the robot to move forward, take a right turn by slowing one motor, take a left turn by slowing the other motor, or turn around by making one wheel go forward and the other backward. Also, chains were not used in this design because they increase the weight on motors and will use extra battery power.

5.2.1.3 Ultrasonic Sensor

When the robot is placed in an unknown environment, especially one which is dangerous or inaccessible for humans, it is important to minimise the robot's contact with obstacles so that the risk of the robot breaking down can be reduced [153]. Therefore, the robot uses an ultrasonic sensor to sense the obstacles ahead and takes preemptive action to avoid those obstacles. It was previously observed in the Chapter 4 experiments that this can increase

the reaction time of the robot and also increases self-preservation which is a highly desired quality. Therefore, the ultrasonic sensor is employed in this new robot's design as well.

5.2.1.4 Bump Sensors

Unfortunately, it is not guaranteed that the ultrasonic sensor will definitely protect the robot all the time as it does not detect obstacles which are higher or lower than the position of the ultrasonic sensor [17]. Therefore, a bump sensor was also added to the robot, which responds to actual collisions. An anti-collision bar is added all around the robot, which also triggers the sensor whenever the robot hits the wall and protects the actual sensor and robot from any damage.

5.2.2 The Environment

The virtual environment created in NetLogo contains overlapping rectangles of different colours. The environment is surrounded by a boundary wall which is represented by the red colour in NetLogo. The vehicle shape is designed as close as possible to the robot design described above in section 5.2.1. The world wrapping option is disabled in NetLogo settings so that a realistic environment can be created in order to avoid issues during real-world experiments. The patches store the number of times the robot visits them and this information is used by the entropy calculation measure (described below in section 5.2.4) as a fitness function when evolving the most interesting behaviour. After setting up the environment, the next step is to choose the appropriate settings and grammar (see next section) that is used to evolve several different behaviours. A large number of simulations need to be compared in order to discover interesting behaviour. Initially, the environment was designed with different shades of grey colour to represent the real-world scenario where a light was placed in the centre with darker patches representing low light conditions. However during experiments, it was difficult to distinguish colours accurately, therefore more prominent colours were used in the environment (see figure 5.10).

5.2.3 The BNF Grammar

The purpose of these experiments is to find the most interesting behaviour using Grammatical Evolution. The possible syntax of the program code that could be evolved, was defined by using the Backus-Naur Form notation [109]. The following grammar was initially used in the application.

Listing 5.1: The initial grammar used to discover interesting robotic behaviours.

```
1
2 <expr> ::= <actions> | <expr> <actions>
3 <actions> ::= <sensingaction1> | <sensingaction2> |
4 <sensingaction3> | <sensingaction4> | <sensingaction5>
5 | <turn1> | <turn2> | <move>
6 <sensingaction1> ::= ifelse (ROBOT::OBSTACLE-AHEAD?)
7 [<turn1>] [<move>]
8 <sensingaction2> ::= ifelse (ROBOT::COLOUR-AHEAD-WHITE?)
9 [ <turn1>] [ <move>]
10 <sensingaction3> ::= ifelse (ROBOT::COLOUR-AHEAD-YELLOW?)
11 [ <turn1>] [ <move>]
12 <sensingaction4> ::= ifelse (ROBOT::COLOUR-AHEAD-BLUE?)
13 [ <turn1>] [ <move>]
14 <sensingaction5> ::= ifelse (ROBOT::COLOUR-AHEAD-GREEN?)
15 [ <turn2>] [ <move>]
16 <turn1> ::= ROBOT::RIGHT
17 <turn2> ::= ROBOT::LEFT
18 <move> ::= ROBOT::FORWARD
```

This grammar provides robots with capabilities of avoiding obstacles, sensing colours, and tanking turns (left, right, turn-around). Whenever a colour is detected via a sensing action, the robot will make a turn according to that specific colour and moves forward. Actions are divided into different sub-actions (sensing action). It is worth mentioning here that if a colour is sensed during a sensing action, the robot will “turn” but not move. This is to avoid the robot potentially crashing. Also, it will avoid virtual agents travelling

through the boundary wall which is not a desirable situation as in real-world experiments that behaviour cannot be replicated.

5.2.4 Fitness Function

A fitness function is used to find the individual with the most interesting behaviour from a population across generations through crossover and mutation. The easiest way was to assume that the behaviour with the highest entropy value is the most interesting behaviour (a similar approach was used by Ahmed and Teahan [3]). Keeping this in mind, the entropy was estimated using a compression-based codelength calculation for each simulation.

As stated, we can use a straightforward entropy-based metric for measuring Interestingness. The metric that has been devised uses a Markov model approach in order to predict the visit counts for the patches in the environment that is produced at the end of each simulation (after the simulation has run for a set number of ticks).

One of the simplest Markov models that we can investigate is to use order 0 probability estimations. (This is in contrast with the approach by Ahmed & Teahan [3] which used higher order models). The order 0 approach we have devised is where the probability estimation is based simply on the distribution of the visit counts without taking into account any prior context i.e. the visit counts of the preceding patches are ignored.

Formally, we can adapt the standard entropy-based formula devised by Shannon [143] that is well founded in Information Theory, to propose the following novel metric for measuring the ‘Interestingness’ I that is related to the compression codelength of encoding all the visit-counts in the patches of the simulation environment:

$$I = - \sum_{\rho \in \text{patches}} \log_2 P_\rho \quad (5.1)$$

where ρ is a patch in the environment and P_ρ is the probability that the patch has the specific visit-count associated with it.

The key to the success of this metric is the way the probabilities are estimated. The probability for patch ρ having a specific visit-count is estimated from the frequency of the visit-count for that patch:

$$P_{\rho} = \frac{(\omega \times C_{\rho}) + 1}{\sum_{u \in patches} (\omega \times C_u) + 1} \quad (5.2)$$

where C_{ρ} is the visit count associated with patch ρ and ω is a constant used for weighting non-zero frequency counts more heavily. (That is, a higher value for the ω constant will assign a greater weight to the non-zero counts).

Essentially, an order 0 model is being used to predict the specific visit count for each patch in the environment. Then, I provides a theoretically minimum estimate for the cost of encoding the visit counts for the whole environment. If the visit-counts were being physically encoded to a file on disk, an encoding close to the theoretical minimum can be achieved using arithmetic coding [130], [137].

The standard add-one smoothing [123], [26] is adopted here where $+1$ is added to the frequency count to ensure non-zero probability estimates in all cases (i.e. for all patches). This is required because a majority of the patches in the simulations being generated have zero visit-counts i.e. the simulated robots for the more interesting configurations will only visit a small number of patches in the environment.

The denominator, which sums across all the different possible visit counts, normalises the probabilities across the probability distribution so that they add up to 1.

However, in the experiments described below, this codelength calculation was not satisfactory enough to find the interesting behaviours as the model started returning similar behaviours. In any standard evolutionary algorithm, this is a very important consideration that determines the final performance when evolving the solution (phenotype) because a poorly designed fitness function will evolve inappropriate or undesirable solutions with poor performance. Keeping this in mind, the fitness function was modified and a bonus amount

of 500,000 was added to separate the small and large values of codelength (entropy based fitness measure). The behaviours displayed by the vehicles were processed by using codelength value as file name, so that when the images are sorted by name, the export-view of the behaviours will be sorted by most interesting at the top.

5.3 The Experiments

This section contains two main parts: part one describes the process of discovering interesting robotic behaviours; and part two presents the validation of these novel interesting behaviours in the real-world. Table 5.1 shows a summary of NetLogo models that were developed for the experiments.

Table 5.1: The summary and details of the experiments.

Name of Model	Description of Model
Virtual Experiments	
Robotic-Behaviours	Developed robot actions to create behaviours in the new environment.
Interesting-Behaviours	Used entropy based calculation to find interesting behaviours.
iFinder-Discovery	Grammatical Evolution applied to discover interesting robotic behaviours.
Real-World Experiments	
Colour-Sensing	Adjustment of colour sensor values and tolerance for accurate colour detection.
Turn-Angles	Adjusts Angles when the robot makes a turn so that behaviours can be recreated as close to the simulations as possible.
iFinder-Validation	Real-world validation of evolved interesting behaviours.

The code of these models have been made freely available in the project's Github repository [8].

5.3.1 Virtual Experiments

This section and its subsections present the design and results of the virtual experiments. There are three experiments in this section to examine different hand-crafted behaviours, identify interesting behaviours, and evolve interesting behaviours.

5.3.1.1 Robotic Behaviours

A NetLogo model (Robotic-Behaviours) was created as an initial experiment to simulate the behaviour of the robot. The objectives of this experiment are:

- Designing a simulated robot and environment with different colours so that the robot can react in a non-trivial and potentially interesting way according to the colour detected.
- Observing the robot's behaviour by experimenting with different sets of commands in NetLogo.

The model (see figure 5.3) was used as an initial setup for the main experiments. The model visualises the actions taken by the robot during the simulation by drawing its path in red in NetLogo's environment (shown in the middle of figure 5.3). The patch visit counts are also shown in white. These are used later by the compression codelength fitness function in the next model (Interesting-Behaviours) to help evolve the interesting behaviours.

It is important to confirm that the set of commands sent to the robot are able to produce interesting behaviours (validated in the real world). Several combinations of configurations were tried, and one of these produced the behaviour shown in figure 5.3. This behaviour is quite similar to one of the interesting behaviours produced by Ahmed's model. The behaviour commands executed by the robot is shown on the left of the figure.

The following guidelines show how these simple commands, and a variety of configurations, can be used to produce interesting behaviours.

How to Use the Model

The model's speed needs to reset to normal, so that the environment can be redrawn quickly. Then the experiment is initiated by pressing the "Setup" button. When this is pressed, the environment will appear in the 2D view of the NetLogo interface. The set of commands will also appear on the left side in multi-line input box "Robot-Actions" which is editable making

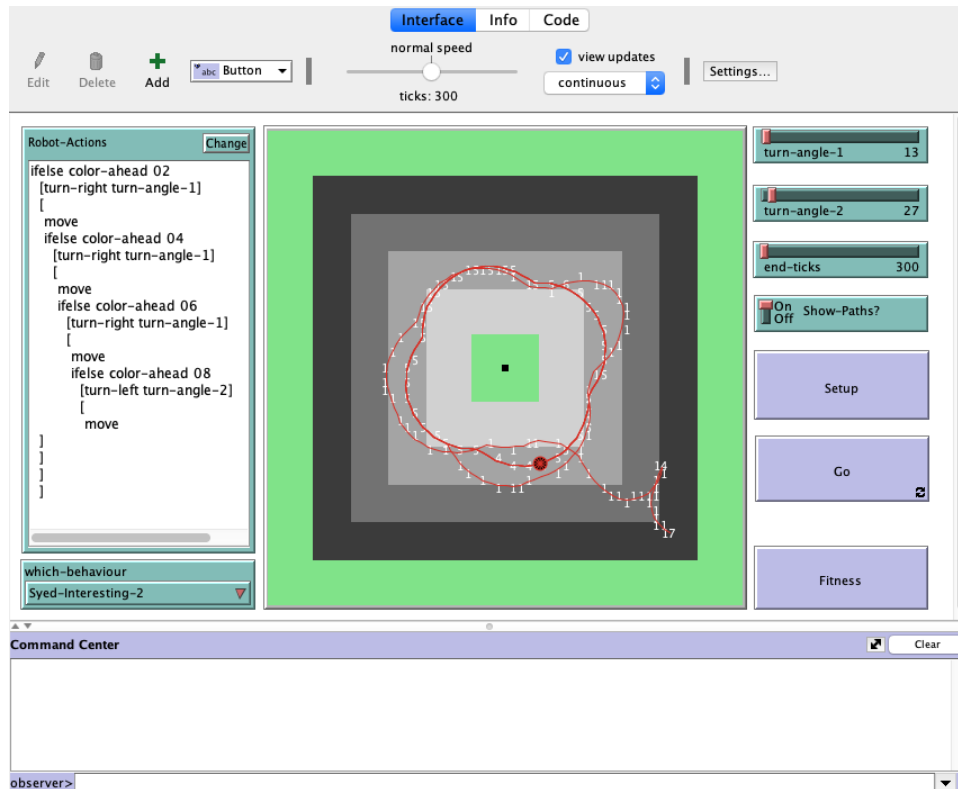


Figure 5.3: A behaviour hand-crafted for the model Robotic-Behaviours.

it possible to change any of the commands. Listing 5.2 shows the code (NetLogo commands) used to control the robot to turn right or turn left via the `turn-right` and `turn-left` commands (at the angle chosen in configuration) if there is detected the specific colour ahead (via the `colour-ahead` command) which will then result in the robot moving forward one patch.

Listing 5.2: Robot actions and control program in the NetLogo model Robotic-Behaviours.

```

1 NetLogo Robot Commands:
2         move, turn-right, turn-left
3 NetLogo Robot Reporter:
4         color-ahead
5 Sample Robot Actions:
6         ifelse color-ahead [colour]
7             [turn-right][turn-angle-1]
8             [move]
9         ifelse color-ahead [colour]
10            [turn-left][turn-angle-2]
11            [move]

```

Before starting the simulation, the ‘Max-Steps’ (ticks) slider must be set to the desired number of steps the user wants the robot to take. Also, the turning angles (turn-angle-1 and turn-angle-2) must be set to the desired settings as well. To follow the robot’s behaviour, we need to monitor the robot’s path by setting ‘Show-Paths?’ switch to on. Once the model is ready, the ‘Go’ button needs to be pressed in order to execute the ‘Robot-Action’ commands. The model’s speed must be set to very slow (by adjusting the speed slider) so that the robot’s behaviour can be observed easily. If the model’s speed is not set to very slow, the robot will run so fast that only the final behaviour can be seen.

5.3.1.2 Interesting Behaviours

A further NetLogo model (Interesting-Behaviours) was developed that improves the first model (Robotic-Behaviours) by adding capabilities to select random configurations of the interface settings when running the model. The model can be used to find the configuration of settings and internal robot code that produces the highest compression codelength via the fitness function described above in section 5.2.4.

The objectives of this experiment are:

- Designing a model capable of selecting random configurations to generate the behaviour and to be capable of repeating that behaviour as many times as required.
- Analysing all the behaviours generated by the different simulated robots, in order to manually determine whether interesting behaviours can be generated for this robot and environment.

Most of the interface configurations of the model ‘Interesting-Behaviours’ are the same as the model ‘Robotic-Behaviours’ including ‘Robot-Actions’, ‘Setup’, ‘Max-Steps’ and ‘Show-Paths?’. This model generates different turn angles unlike the Robotic-Behaviours model which were set manually via the interface between 1-360. To run the model several times, a repeat button

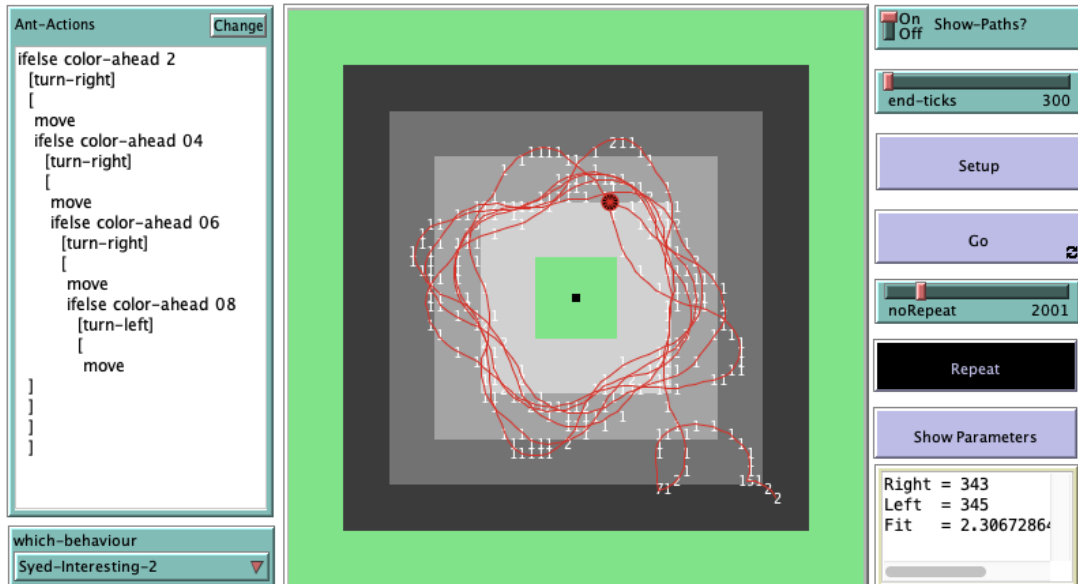


Figure 5.4: Interface of the model Interesting-Behaviours.

is added in the interface and the number of repeats can be decided by using the ‘noRepeats’ slider. By using these random configurations and repeating capabilities, it is possible to create 129,600 different behaviours (due to the possible variations in the different parameter settings). However, running the simulation for this amount of times was deemed too time consuming and also would have required an enormous amount of storage to store the results. Therefore, results for these initial experiments were gathered by conducting 10,000 simulations with randomly generated configurations.

The next step was to identify the interesting behaviours, which was achieved by using the compression codelength calculation (explained in detail in section 5.2.4). The output box in the interface shows the parameters for each run which are generated randomly. A view of the environment depicted in the interface is exported at the end of every run as shown in figure 5.4 above. The NetLogo code that performs the calculation of the compression codelength entropy-based metric to find the interesting robotic behaviour is as follows:

Listing 5.3: Calculation of Entropy to find the interesting robotic behaviour

```
1 to-report calculate
2
3   let counts [counter] of patches
4
5   let calc 0.0
6   let total 0
7   let adjusted-count 0
8   let prob 0.0
9   let neglogprob 0.0
10  let probfactor ProbFactor
11
12  foreach counts
13    [ my-count ->
14      set total total + my-count * probfactor + 1
15    ] ; +1 to ensure non-zero probability
16
17  foreach counts ;codelength calculation on all the counts
18    [ my-count ->
19      set adjusted-count my-count * probfactor + 1
20      set prob (adjusted-count / total)
21      set neglogprob (- log prob 2)
22      set calc calc + adjusted-count * neglogprob
23    ]
24  report calc
25 end
```

The results achieved are shown in figure 5.5 below and are taken from a selection of the robot behaviours with the lowest ranked deemed to be the Least-Interesting, and the higher ranked deemed to be Interesting and Most-Interesting respectively. Visual analysis of the latter clearly shows more complex and surprising behaviours than the former. The results confirm that this task-environment setup can produce interesting and unusual behaviours.

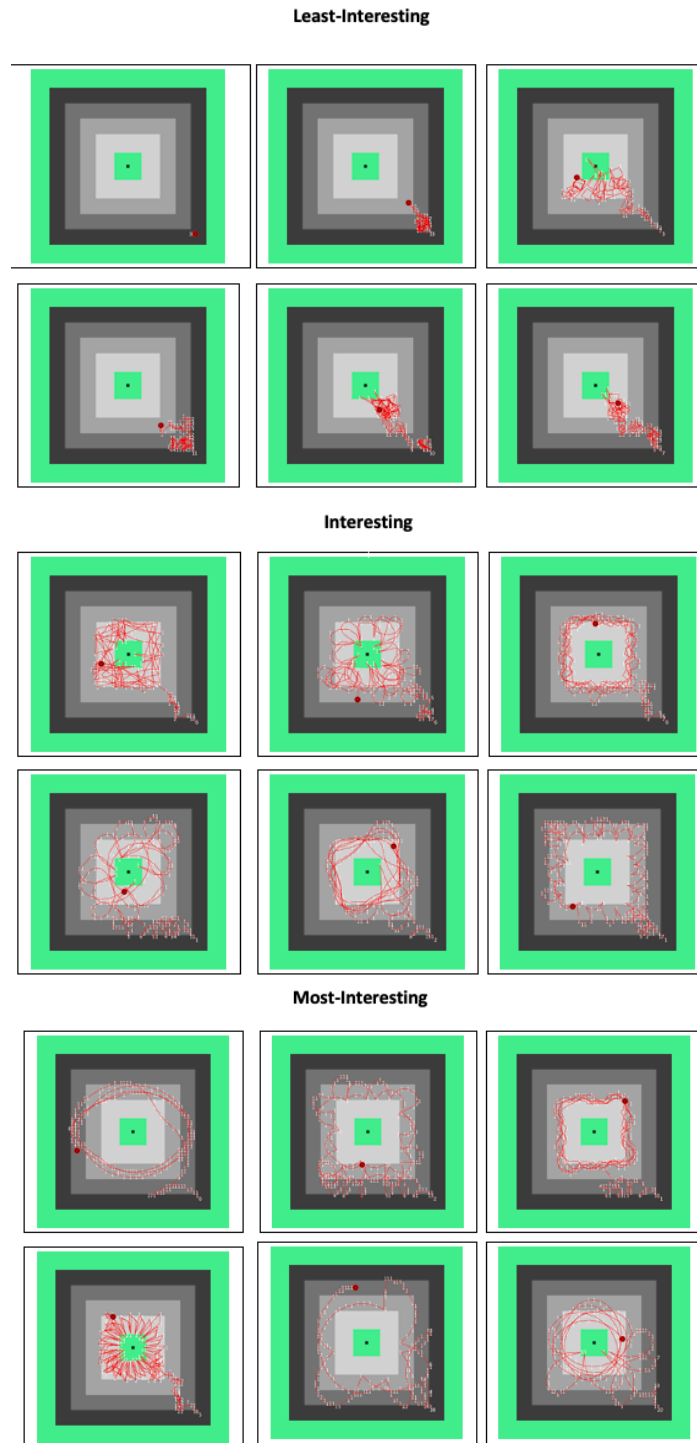


Figure 5.5: Some of the different behaviours randomly produced by the model "Interesting-Behaviours" as ranked in ascending order by the compression Codelength metric.

How to Use the Model

The model's speed must be set to fastest by adjusting the speed slider in the toolbar in order to complete 10,000 runs in a reasonable time, otherwise, it can take hours or days to run that many simulations at normal or slow speed. The experiment is initiated by pressing the "Setup" button. When this

button is pressed, the environment will appear in the 2D view of the NetLogo interface. A set of commands will appear on the left side in the multi-line input box ‘Robot-Actions’ which is editable and provides the user with the ability to change any of the commands the robot executes. Listing 5.2 shows the valid actions of the simulated robot and a sample robot control program which dictates the robot to `turn-right` or `turn-left` if there is a specific colour ahead via `colour-ahead` and moves forward one patch otherwise. Before starting the simulation, the ‘Max-Steps’ (ticks) slider must be set to the desired number of steps the user wants the robot to take.

To follow the robot’s behaviour, the user can monitor the robot’s path by setting the ‘Show-Paths?’ switch to on position. Once the model has been set up with the desired configuration, the ‘Go’ button is pressed in order to execute the ‘Robot-Actions’ commands. Once the model has stopped, further simulation runs can be produced by pressing the ‘Repeat’ button. The results are saved to files for later analysis.

5.3.1.3 iFinder Discovery

This section provides an overview of the main experimental model named ‘iFinder-Discovery’ (see figure 5.6), followed by an explanation of how to use the model. The experimental setup and results are also discussed. This is the final model of these virtual experiments which further develops the above mentioned models by using Grammatical Evolution, for the discovery of interesting robotic behaviours.

The model uses the newly developed system environment (described in Chapter 4) for the grammatical evolution genotype-to-phenotype mapping process, which involves a genetic algorithm as search engines, a roulette wheel for the selection of the parents, and the creation of the offspring population via a steady-state replacement strategy. Several buttons and sliders are used for the configuration of the grammatical evolution algorithm and for setting up the experiment.

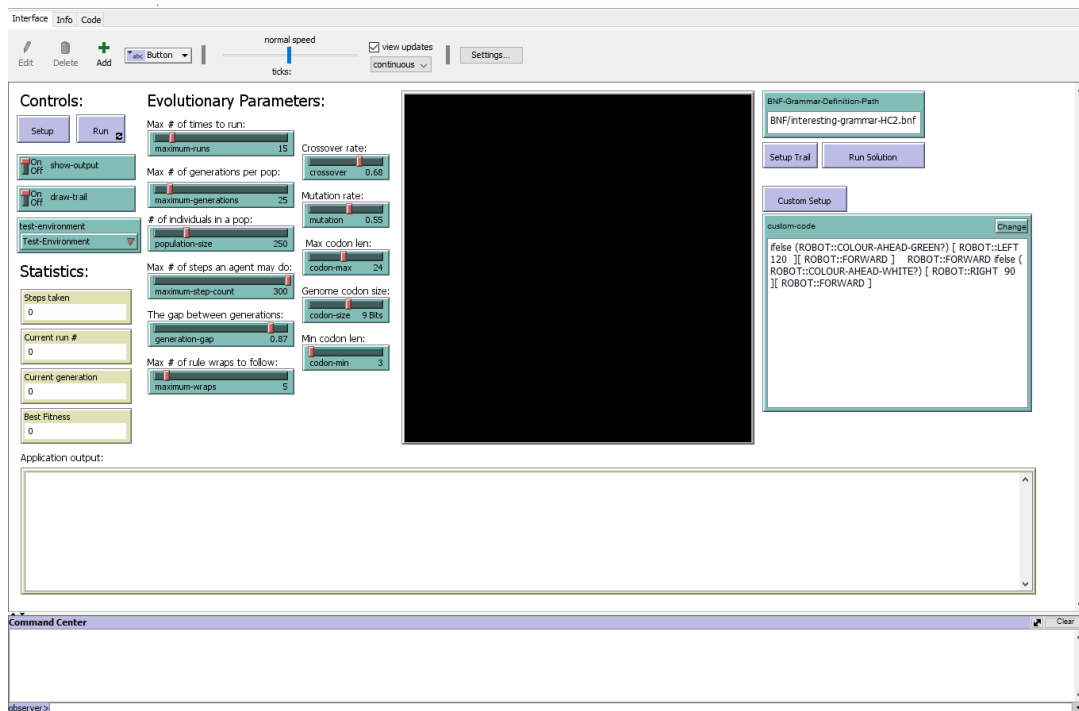


Figure 5.6: The “iFinder-Discovery” application to evolve interesting behaviours.

The execution of the simulation is on the top left and a live running result of the simulation can be seen in the window on the left side. The objectives of this experiment are:

- Developing and discovering interesting robotic behaviours using the new developed robotic system environment.

This “iFinder-Discovery” application provides the users with several configurations that can affect the process of evolution. During this simulation evolution process, the application creates a robotic population which is evaluated one by one by using the codelength fitness function. The fittest individuals generate the new offspring, and this process is repeated until a suitable solution is found. At the end of the process, it stores the phenotype of the fittest individual which can be extracted to a Lego Mindstorms control unit by the user as discussed previously in section 4.4.

The following guidelines show how this easy-to-use process can be used to produce interesting behaviours.

How to use the Model

The majority of this model's features are sliders which make it easy to use. The user can select required configurations by moving the sliders from side to side. The model's speed must be set to fastest by adjusting the speed slider in toolbars so that selected number of runs and generations can be completed as soon as possible, otherwise, it can take days to run that many simulations at normal or slow speed. The user can also choose if they want to view the path taken by the robot during simulations by using the "draw-trail" switch.

After selecting the desired configuration settings, when the user clicks the "Setup" button followed by the "Run" button, this initiates the evolutionary process. The application creates a robotic population which is evaluated by the fitness function. The fittest individuals generate the new offspring, and this process is repeated until a suitable solution (phenotype) is found. This solution is displayed in the output box and later extracted to the robot. Usually, this process will indicate the most interesting robotic behaviour produced during that set of simulations. However, we will also be analysing the top 20 interesting behaviours (examples shown in figure 5.5 above) to make sure we do not miss the more interesting ones.

When the generated solution code displays in the output box, it can be executed again for study and analysis by pressing the "Setup Trail" and "Run Solution" buttons respectively. The solution can be re-executed as many times as the user wants, or until the user presses the "Setup" button again to initiate a new experiment. During the study and analysis of the found solution, the speed slider can be adjusted to slow the speed of the robot for better observation and understanding of the behaviour.

5.3.1.4 Experimental Setup and Results of Experiments

A series of experiments have been conducted to determine how successful this design framework is for discovering interesting robotic behaviours. The details of two examples are discussed below.

The model “iFinder-Discovery” contains various controls and statistics, evolutionary parameters, custom-codes, and the test environment viewing box. The controls let the user set up the experiment, choose the desired test environment, choose the desired BNF grammar, run the experiment, and setup trial and run the solution to study the best solution produced. The statistics show the number of steps the robot has taken, the current number of the run, and the current number of the generation. It also shows the best fitness value achieved so far. The parameters let you decide the number of runs, number of generations, number of steps, size of population, generation-gap, maximum-wrap, crossover, mutation, codon size, minimum codon value, and maximum codon value. The custom setup lets you run the best code generated from any run or generation, or you can create your own code for further testing. The test environment viewing box shows you the output (behaviour) of the robot.

The time it takes to generate solution code depends on configurations used for that experiment and the speed selected using the speed slider. Some examples with different combinations of configurations (with highest speed possible) are provided in table 5.2 where the columns provide the number of runs, the number of generations, population size, and the time it took to run that experiment respectively.

Table 5.2: The time taken to generate solution code with different configurations.

No.	Runs	Generations	Population Size	Time (sec)
1	5	10	100	24.63
2	5	20	100	42.37
3	10	10	200	107.36
4	10	20	200	215.51
5	15	25	250	516.74

The results show that only relatively modest resources are required in terms of execution speed to run the experiments such as experiment no 1 in table 5.2 produced 5000 ($5 \times 10 \times 100$) behaviours in just 24.63 seconds.

To demonstrate the variety of results that are possible, two different grammars are used to produce solution code. The grammar used to generate Code for Example 1 is shown in listing 5.4.

Listing 5.4: The grammar used for Example 1.

```
1 <expr> ::= <actions> | <expr> <actions>
2 <actions> ::= <sensingaction1> | <sensingaction2> |
3 <sensingaction3> | <sensingaction4> | <sensingaction5> |
4 ROBOT::RIGHT <angle> | ROBOT::LEFT <angle>
5 | ROBOT::TURN-AROUND | ROBOT::FORWARD
6 <sensingaction1> ::= ifelse (ROBOT::COLOUR-AHEAD-WHITE?)
7 [ ROBOT::RIGHT <angle> ][ ROBOT::FORWARD ]
8 <sensingaction2> ::= ifelse (ROBOT::COLOUR-AHEAD-YELLOW?)
9 [ ROBOT::RIGHT <angle> ][ ROBOT::FORWARD ]
10 <sensingaction3> ::= ifelse (ROBOT::COLOUR-AHEAD-BLUE?)
11 [ ROBOT::RIGHT <angle> ][ ROBOT::FORWARD ]
12 <sensingaction4> ::= ifelse (ROBOT::COLOUR-AHEAD-GREEN?)
13 [ ROBOT::LEFT <angle> ][ ROBOT::FORWARD ]
14 <sensingaction5> ::= ifelse (ROBOT::COLOUR-AHEAD-RED?)
15 [ ROBOT::LEFT <angle> ][ ROBOT::FORWARD ]
16 <angle> ::= 30 | 60 | 90 | 120
```

The number of ‘max-steps’ is kept to 300 so that when testing in the real-world, it does not take long because the NXT brick is power-hungry, and may drain the battery before completing the whole run. The number of ‘maximum-runs’, ‘maximum-generations’, and ‘population-size’ is kept high to get the best results possible. The best results (highest fitness) achieved using this grammar is shown below in figure 5.7.

After analysing these results, it was decided that an interesting robotic behaviour to examine further was 5.7(a) because it also had the highest fitness value and the solution code (see listing 5.5) produced for that behaviour will be loaded to the real-world robot.

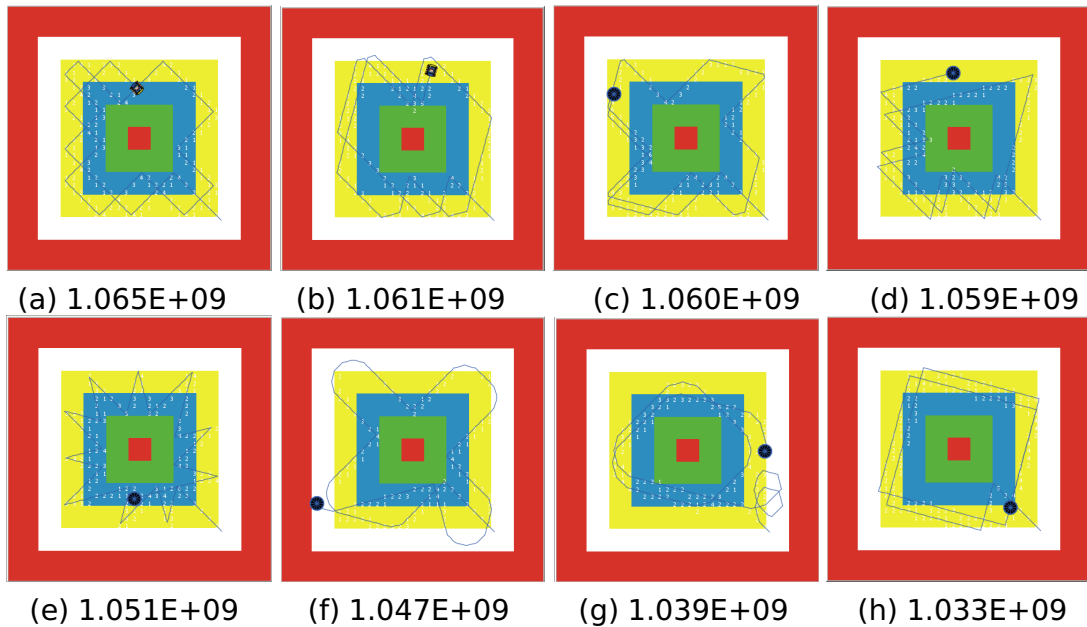


Figure 5.7: The results achieved for Example 1. Below each simulation output is the codelength fitness value.

Listing 5.5: The solution code for the behaviour with highest fitness value for Example 1.

```

1 ifelse (ROBOT::COLOUR-AHEAD-GREEN?)
2           [ ROBOT::LEFT 90 ][ ROBOT::FORWARD ]
3 ifelse (ROBOT::COLOUR-AHEAD-WHITE?)
4           [ ROBOT::RIGHT 90 ][ ROBOT::FORWARD ]

```

For Example 2, similar configurations were adopted but using a different grammar. The grammar is shown in listing 5.6.

Listing 5.6: The grammar used for Example 2.

```

1 <expr> ::= <actions> | <expr> <actions>
2 <actions> ::= <sensingaction1> | <turn> | <move>
3 <sensingaction1> ::= ifelse (ROBOT::COLOUR-AHEAD? <colours>)
4                       [ <turn> ][ <move> ]
5 <colours> ::= 5 | 15 | 35 | 55 | 95
6 <turn> ::= ROBOT::RIGHT <angle> | ROBOT::LEFT <angle>
7 <move> ::= ROBOT::FORWARD
8 <angle> ::= 05 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
9           55 | 60 | 65 | 70 | 75 | 80 | 85 | 90

```

The purpose for the grammar used for Example 1 was to allow the generation of behaviour involving basic robot actions (similar to initial experiments) where the robot is taking a right turn for white, yellow, and blue colours and a left turn for green and red colours which is followed by a step forward by the robot. Whereas the grammar for Example 2 provides freedom of turning left or right after detecting any colour followed by any type of action. Also, a wider range of turn angles is available. Essentially, the difference between both grammars is the way the robot reacts after detecting colours.

The results were analysed, and the interesting behaviour was discovered through the evolutionary process (see figure 5.8). The code (see listing 5.7) for the behaviour with the highest fitness value is as below:

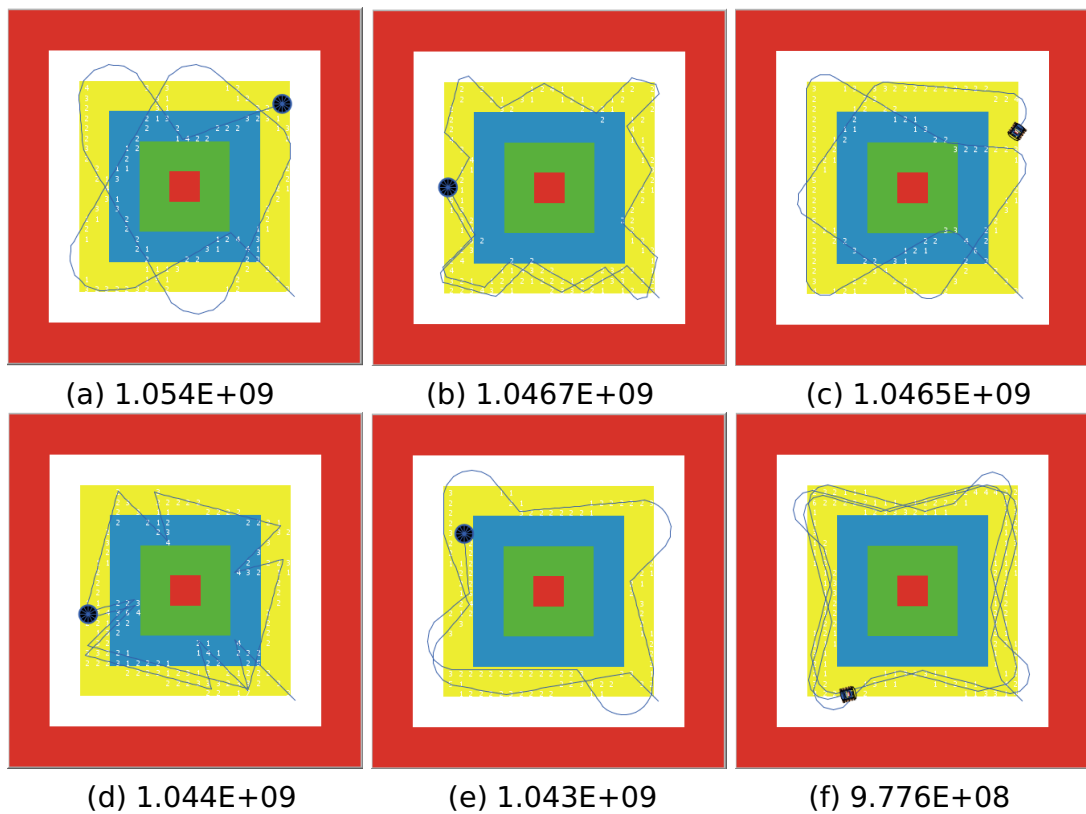


Figure 5.8: The results achieved for Example 2.

Listing 5.7: The solution code for highest valued behaviour for Example 2.

```

1 ROBOT::FORWARD ifelse (ROBOT::COLOUR-AHEAD? 5 )
2           [ ROBOT::RIGHT 25 ] [ ROBOT::FORWARD ]
3 ifelse (ROBOT::COLOUR-AHEAD? 55 ) [ ROBOT::LEFT 75 ]
4           [ ROBOT::FORWARD ] ROBOT::FORWARD

```

Both grammars produced some interesting results. However, the grammar and configurations used to generate these results are basic which shows the possibilities that can be achieved with this robotic environment. More complex grammar (such as choice of turn angles can be increased) and advanced configurations (higher number of runs, generations, and population size) can be used in future experiments to generate more complex results.

5.3.2 Real-World Experiments

This section and its subsections present the real-world validation of the above mentioned experiments. Note that, it is important to mitigate the issues (colour sensing and turning angles) detected in previous chapters in order to achieve more faithful reproduction of the behaviour for validation of the results.

5.3.2.1 Colour Sensing

In real-world experiments involving colour sensing, the results produced by the robot were different than expected. The robot was able to follow simple actions (Forward, Left-Turn, and Right-Turn); however, sensing via the `colour-ahead` command proved problematic due to issues involved with matching real colours.

Interface for Colour Sensing

A new NetLogo model called Colour-Sensing was designed to send commands to the robot and to receive and show results of the robot's colour sensor. The user interface of the new application is shown below in figure 5.9.

This new interface application provides the following functionalities.

Connect to the robot – Using the input box “Insert-Robot-Name”, the user can insert the name of its robot as it shows on the NXT brick. The application attempts to connect to a Mindstorms robot of that name via a Bluetooth connection.

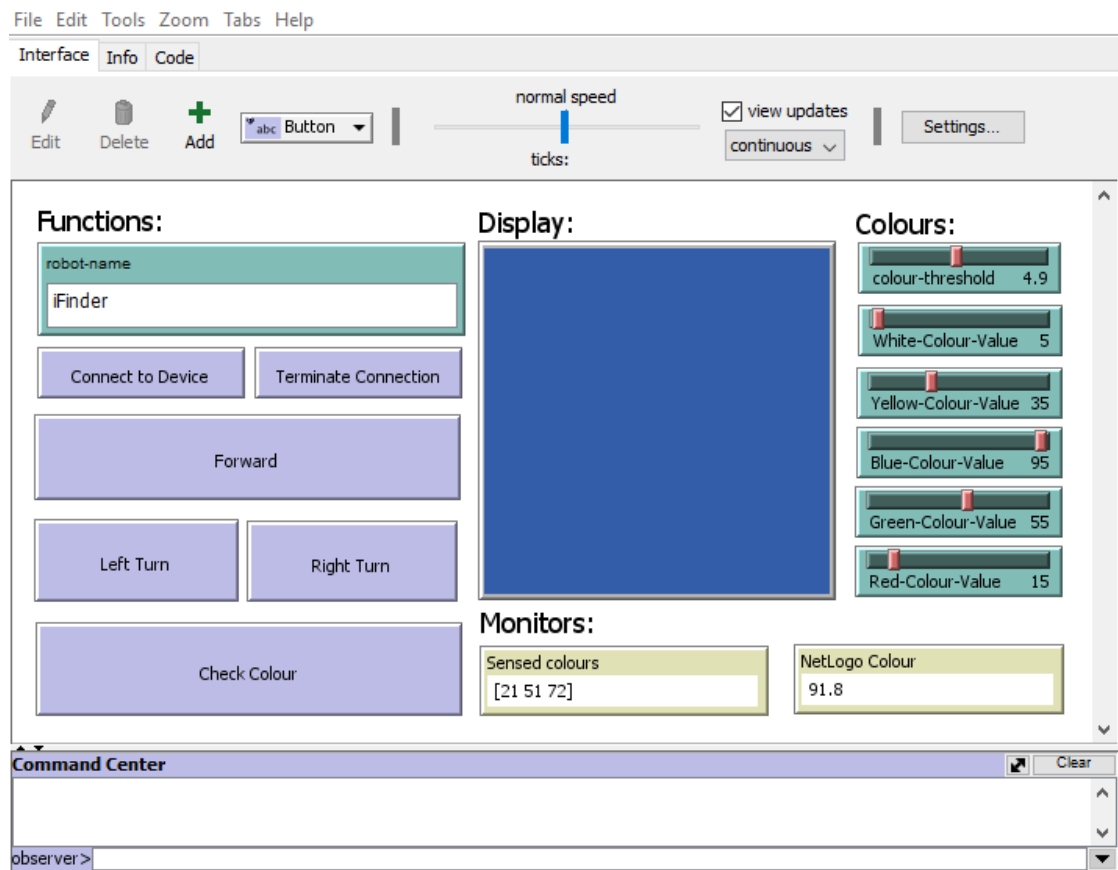


Figure 5.9: The user interface of the Colour Sensing application.

Terminate Connection – By pressing the ‘‘Terminate Connection’’ button, the user can stop sending commands to the robot and stops receiving feedback from the robot. All open channels and resources are closed when the user presses this button.

Forward – By pressing the ‘‘Forward’’ button, a command is sent to the robot and the robot moves one step forward.

Right Turn – The robot turns towards the right.

Left Turn – The robot turns towards the left.

Check Colour – The ‘‘Check Colour’’ button converts RGB values into NetLogo colour and prints out the colour detected by the robot.

5.3.2.2 Results and Discussion

The experimental results of chapters 3 and 4 identified the problem of colour detection based on a number of factors including but not limited to:

- condition of lighting in the room.
- colour tolerance.

By analysing the experiments performed in the last two chapters, it was decided to use a threshold for matching colours. However, based on the numerical values of the colours not exactly matching, we found that settings used in previous experiments were inadequate for the present real-life environment because of low light conditions. Therefore, the colours used in the test environment were changed to bright colours so that the boundaries between them could be distinguished more easily. The new test environment is shown in figure 5.10.



Figure 5.10: The colour setup used for the new test environment.

After some trial and error with this Colour Sensing model, a setup was found where the colours were correctly sensed and matched. The variable colour-to-sense is the target colour, and the colour-threshold sets the

threshold being used to determine if there has been a successful match. Both of these variables are specified via the interface. Matching was done between the sensed colour and the target NetLogo colour by using the following metric (see listing 5.8) between the RGB values sent by the colour sensor (see left monitor in figure 5.9).

Listing 5.8: Matching colours between sensed colours and the target NetLogo colours.

```
1 to-report ROBOT::COLOUR-AHEAD-WHITE?  
2   let colour nxtlogo:colour-data  
3   let r item 0 colour  
4   let g item 1 colour  
5   let b item 2 colour  
6   let this-approx-colour approximate-rgb r g b  
7   report (abs(this-approx-colour - White-Colour-Value) <  
8           colour-threshold)  
9 end
```

The success criteria for the robot was to complete two rounds of the environment while detecting and returning the exact colour. The model produced successful results, for example, by the blue colour matching as shown in figures 5.9 and 5.10. The video of these experiments can be seen on a YouTube channel (see Appendix A) using the following link <https://www.youtube.com/channel/UCbqDprtdpk3WIthZJ7f1hVg>.

It is difficult to draw solid conclusions from the results presented above because of various factors. For example, it can be seen in the video that colour values detected were slightly different in the areas covered by a shadow from a neighbouring table. The problem can be overcome by setting the test environment in a room without furniture. The results can be further improved by avoiding sunlight and using only room lights as a change in the sunlight also affected the results was also noticed during experiments.

5.3.2.3 Turn Angles

The Colour-Sensing application was further developed to fulfil the needs of the robot's turn angles real-world validation. The experimental results showed that, once again, the robot was behaving differently than expected. The robot was able to recognise colours and react accordingly (turn left, turn right, or turn around); however, the turn angle was different to what it was supposed to perform. To sort out this issue, a new test environment with more clearly defined angles was created (see figure 5.11(a)), and the robot was tested using a simple behaviour shown in figure 5.11(b).

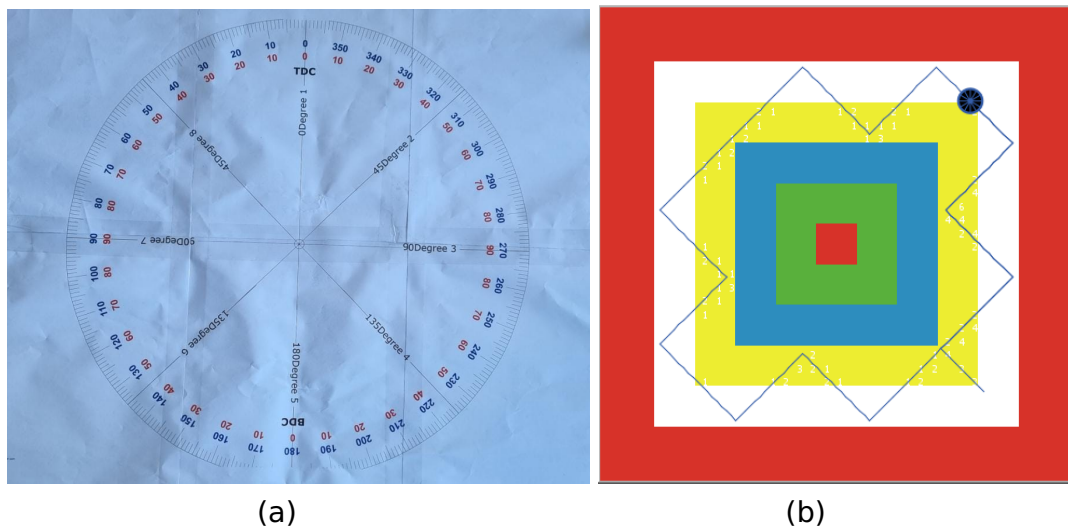


Figure 5.11: Test Environment for the robot's turn angles and new behaviour to test robot's performance after correcting turn angles.

Interface for Turn Angles

The NetLogo interface application designed above was modified to send commands to the robot and to allow the turn angle to be specified via an interface slider. The user interface of the modified application is shown below in figure 5.12.

This modified interface application provides the following extra functionalities.

Run Behaviour – Using the input box “execution-commands”, users can insert the code they wish to run on the robot. Usually, this box is used to place the default code from the “iFinder-Discovery” application (see section 5.3.1.3) to test the behaviours in the real-world. This button passes that code to the robot which keeps running the code until this button is pressed again.

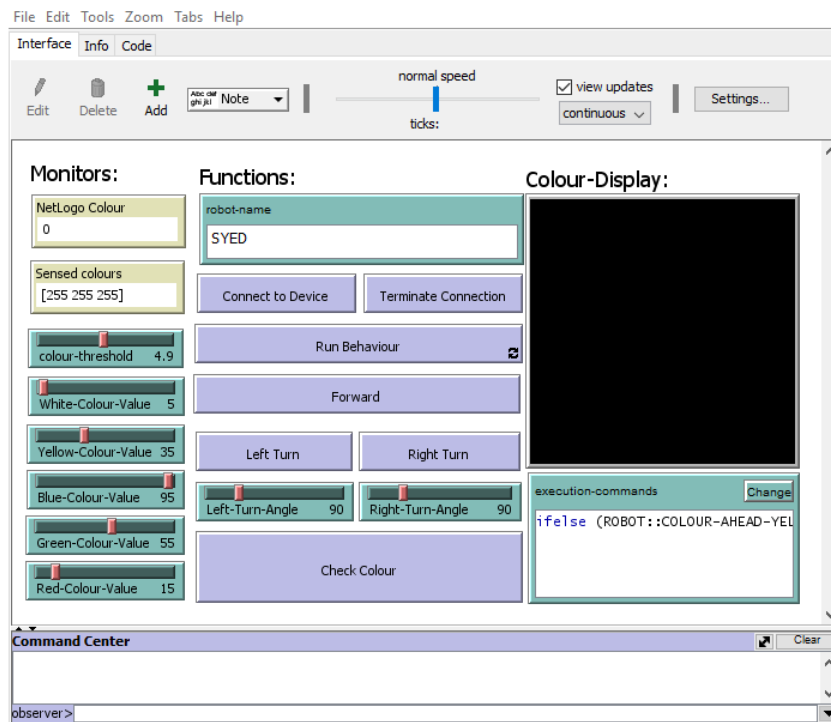


Figure 5.12: User interface of the Turn-Angles application.

Right Turn Angle – This slider lets the user decide the degrees of the angle they want their robot to turn towards the right.

Left Turn Angle – This slider lets the user decide the degrees of the angle they want their robot to turn towards the left.

5.3.2.4 Results and Discussion

As stated, the purpose of Turn-Angles model was to adjust the turn angles and successfully implement a simple behaviour in the real-world. The main reason behind this was excessive turning being performed by the robot.

When the initial experimental model was recreated in the real world, the robot could not recreate the same behaviour mainly because of the excessive turn angles made by the robot. For these modified experiments, a new robot was designed in section 5.2.1 which does not use chains-based tyres as used in the previous two chapters. Therefore, it was necessary to adjust the turn angles accordingly in the ControlFile loaded onto the NXT brick. After several test runs using trial and error, accurate turn angles were found. The updated code in the ControlFile was as follows:

Listing 5.9: Motors adjustment to get accurate turn angles.

```
1 Old Settings:
2 DifferentialPilot pilot =
3     new DifferentialPilot(30, 160, Motor.A, Motor.C, false);
4
5 New Settings:
6 DifferentialPilot pilot =
7     new DifferentialPilot(40, 155, Motor.A, Motor.C, false);
```

In listing 5.9, line number 3 provides the values used previously for the left and right motors turn ratio, and line number 7 presents the new values found to produce more accurate turn angles for the newly designed robot.

The success criteria for the robot were to make a turn (right or left) in real life according to the degrees of angles decided in the turn angle sliders. Once this was done, the simple behaviour presented above in figure 5.11(b) was successfully reproduced in the real world experiment as shown below in figure 5.13. The videos of these experiments can be seen via a YouTube channel (see Appendix A) using the following link <https://www.youtube.com/channel/UCbqDprtdpk3WIthZJ7f1hVg>.

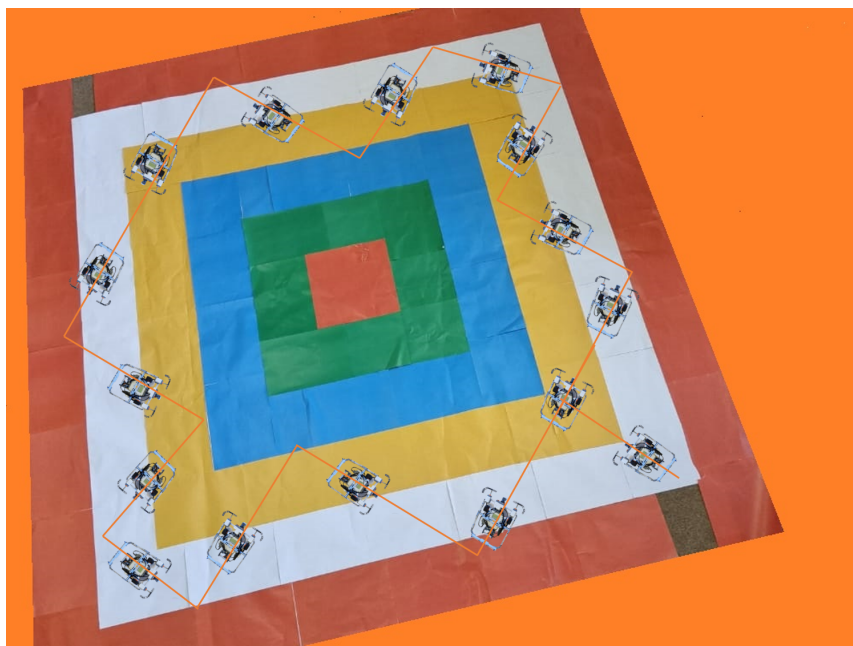


Figure 5.13: Turn Angle robot behaviour successfully recreated in the real-world.

Although the robot met the criteria for success, however, there is still room for improvement. For example, it can be seen in figure 5.14 below that the robot was stuck under the paper while making a turn which affected the degrees of turn. The problem can be resolved by not printing the test environment on A4 papers which are currently joined with sellotape.

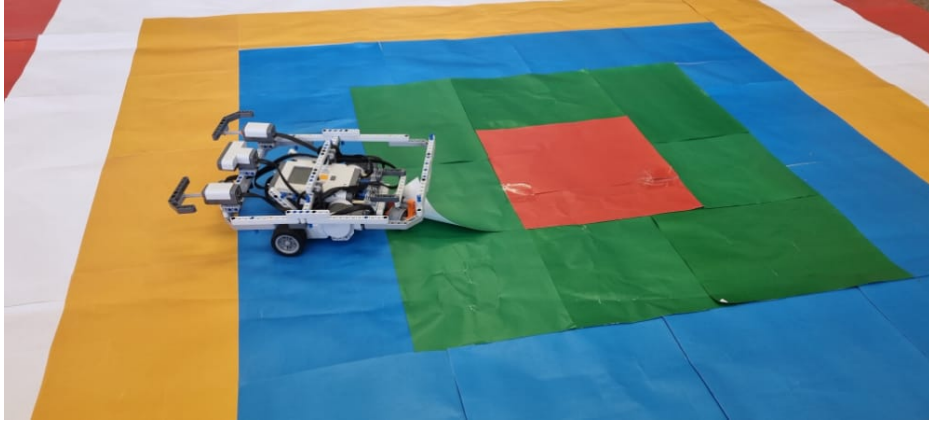


Figure 5.14: The robot becoming stuck while taking a turn.

5.3.2.5 iFinder Validation

This section continues the work done in section 5.3.1.4. A new NetLogo application was created for the real-world experiments and the grammar of the most interesting robotic behaviour is extracted to the “iFinder” control unit to test it in the real world.

iFinder Validation Interface

To extract the evolved code to the robot “iFinder”, a new NetLogo application was designed which is called “iFinder-Validation”. This new application can connect to any Lego Mindstorms’ robot and run any code generated via the “iFinder-Discovery” application. The user interface of the new application is shown below in figure 5.15.

This new interface application provides the following functionalities.

Connect to the robot – Using the input box “Insert-Robot-Name”, the user can insert the name of its robot as it shows on the NXT brick. The application attempts to connect to a Mindstorms robot of that name via a Bluetooth connection.

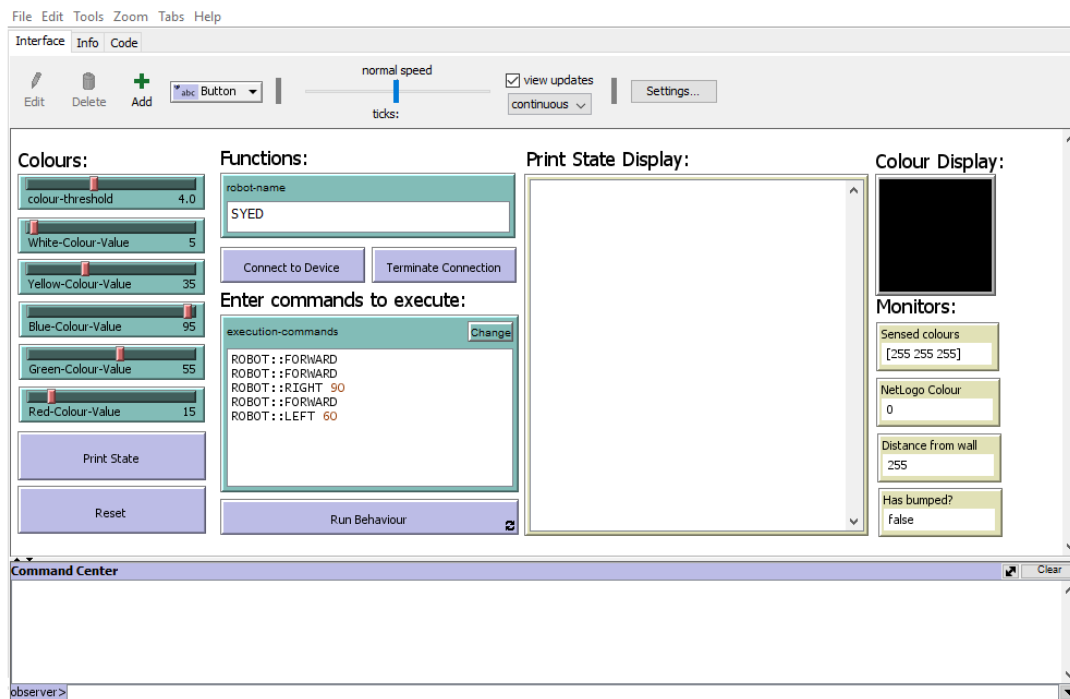


Figure 5.15: The user interface of the "iFinder-Validation" application.

Terminate Connection – By pressing the “Terminate Connection” button, the user can stop sending the commands to the robot and stop receiving feedback from the robot. All open channels and resources are closed when the user presses this button.

Run Behaviour – Using the input box “execution-commands”, users can place the code they wish to run on the robot. Usually, this box is used to place the code that was generated by the “iFinder-Discovery” application to test the behaviours in the real world. This button passes this code to the robot which keeps running it until the user presses this button again.

Colour Parameters – The colour parameters lets the user decide the colour they want to detect and also let them decide the colour threshold limit for matching purposes. These parameters save the user from the hassle of changing code every time and simply allows the user to change the colour value by moving the slider to the left or right.

Monitors – The monitors display data sent back from the robot sensors and displays the current state of the robot.

Print state – This button lets the user print out the data displayed on the left-hand side monitors. These printed values can be used later for analysis.

Logging – The logging box displays the data printed by the ‘Print State’ button.

Reset – This button resets all states and global variables, useful if the user wants to start over with the experiment.

5.3.2.6 Results and Discussion

This section will discuss the results obtained and what was observed during the real world experiments. A real world test environment was designed as close as possible to the simulation test environment (see section 5.2.2). The success criteria for these experiments was whether the robot behaved the same as it behaved in the simulations by using the solution code generated previously loaded onto the robot. The simulation results are reproduced below in figure 5.16:

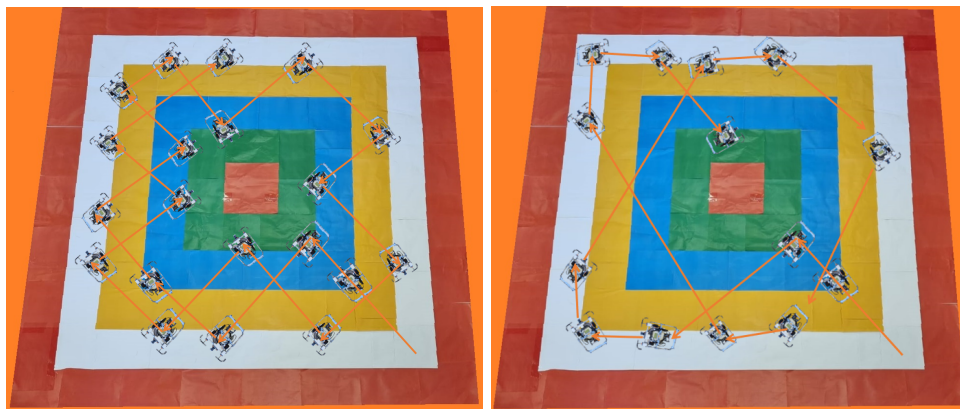


Figure 5.16: The simulation results achieved for Example 1 and Example 2.

The robot performed better than the previous two experiments by successfully recreating the interesting behaviours generated by the simulations. During the experiments, it was observed that the robot was able to recognise colours and take actions accordingly. The turn angles were accurate as well. A video implementation of these real-world experiments can be seen on a YouTube channel (see Appendix A) using the following link <https://www.youtube.com/channel/UCbqDprtdpk3WIthZJ7f1hVg>. These results prove that

the new robotic system environment can discover novel interesting robotic behaviours and validate them in the real-world.

Even though the robot performed well in these experiments and met the success criteria, there are a few other observations which need to be mentioned here. These observations include but not limited to issues concerning:

- Battery Power
- Colour Mismatches
- Test Environment Issues

One of the major problems faced during these experiments was the power-hungry unit of Lego Mindstorms. It was noticed that as soon as the charge in the batteries was less than a certain level, the robot slowed down and this affected the turn angles as well. Therefore, new batteries were used every time before running the experiment. It was also observed that sometimes when the colour reading was exactly in-between two target colours, it would misread the colour and produce values that were different than expected thereby matching one of the other colours that were present in the test environment. These false values could mislead the robot and make it turn in the wrong direction. As mentioned above in the Turn-Angle experiments, the test environment was printed on A4 sized papers that were joined together. The edges of these papers were cut using a paper-cutter, so there was no guarantee that all the papers were the same size which led to colours overlapping. Also, some papers were stuck on top of each other while applying Sellotape which produced a darker shade of the colour that could mislead the robot. A possible solution to avoid this is to re-conduct the experiments using a smooth matte floor. However, this could not be achieved during these experiments because of Covid-19 restrictions.

5.4 Summary

In this chapter, the experiments that were conducted to discover and validate interesting robotic behaviours were reported. The contributions of this chapter include: the design and construction of a new Lego Mindstorms robot; the creation of a NetLogo application to identify interesting robotic behaviours; updating of the Lego Mindstorm's ControlFile; and a new NetLogo application for controlling the final robot that was designed. The robot was constructed to recreate the interesting robotic behaviours in the real world so that its performance in a physical robotic system environment could be examined. A new test environment was designed for real world experiments that matched the environment used in the simulations. The NetLogo application was updated to use a new entropy-based metric using compression codelength to rate the robotic behaviours where interesting robotic behaviours were rated higher than others. A new NetLogo application was designed to facilitate the control of the new robot designed earlier in this chapter (see section 5.2.1). An update to the ControlFile was made to adjust the parameters (turn angles) according to the new robot design.

Various initial experiments were conducted to see if all the robot components were working as required. For example, an initial experiment helped to evaluate the performance of the colour sensor where it was determined that the colour tolerance threshold required was much lower than the experiments in previous two chapters. Another initial experiment mitigated the problem of excessive turn angles being made by the robot which was elevated because of the robot vehicle using wheels instead of chains. A third experiment extracted the evolved code deemed as the most interesting robotic behaviour and this was uploaded to the Lego Mindstorms robot and in order to observe its performance in the real world.

The results showed that the new robotic system design framework can discover novel interesting robotic behaviours and those behaviours can be validated in the real-world.

Although, the results help to answer the questions asked in this thesis, however, there is room for improvement. This is especially true for the test environment which can be constructed using a smooth matte floor where the colours are not overlapping, and lighting conditions are always the same. The battery power of Lego Mindstorms can also be improved to use a constant level of power for best performance.

Further experimentation could also be conducted using different (e.g more complex) grammars and different metrics for the compression codelength fitness calculation. However, the purpose of the experiments that were conducted was to demonstrate that the automatic discovery of interesting behaviour was possible and this has been achieved even when using very simple grammars and a very simple fitness calculation. The effect of using more complex grammars and a more sophisticated fitness calculation (such as used by Ahmed [3]) is left for future work.

Chapter 6

Conclusion and Future Work

This thesis has described the design and implementation of a novel low-cost system for the design of robots. The novel applications of Grammatical Evolution for robotic design were developed to extend the capabilities of the robotic design system. The scope of the robotic design system was demonstrated by developing applications to discover interesting robotic behaviours. The robotic design system was further tested to validate these novel behaviours in the real-world.

In this final chapter, the original objectives will be revisited to conclude the thesis while summarising the thesis findings at the same time and identifying its limitations. The chapter concludes the final thoughts about this newly developed robotic design environment, and future work for further research is identified if there is any.

6.1 Revisiting the Research Questions

In this section, the questions asked in section 1.3 will be revisited to check if the thesis has answered these questions.

- Can a low-cost robotic system design environment be developed that is open and extendable for easily performing evolutionary robotics experiments?

This thesis has successfully developed the low-cost robotic design system as required in question 1. Most of the components are free, and the Lego Mindstorms NXT brick costs only £295. Chapter 4 answered the second part

of the question by performing evolutionary robotics experiments. The robotic design system is written in Java, and anyone who knows Java can extend the robotic design system as it was extended in this thesis to perform grammatical evolution experiments. Therefore, the thesis has shown how to develop a low-cost robotic design system that is open and extendable for performing evolutionary robotics experiments.

- Is the new robotic system design environment capable of automatically discovering novel interesting behaviours and validating these behaviours in the real-world?

The robotic design system has successfully discovered interesting robotic behaviours by using the novel application in chapter 5. These behaviours were also reproduced successfully in real-world experiments using the new robotic design system. This shows that the new robotic design system environment can discover novel interesting robotic behaviours and validate them in the real-world experiments.

6.2 Revisiting the Thesis Objectives

The thesis was introduced with the following objectives:

1. To conduct an extensive literature review on the current state of the art of robotic design and work related to the above research questions and point out current issues, findings, and future directions.
2. To develop an integrated low-cost environment for evolutionary robotics experiments.
3. To use the new environment to discover novel interesting robotic behaviours.
4. To use the new platform to validate novel interesting robotic behaviours in the real world.

6.2.1 First Objective

The first objective was completed in chapter 2. The literature review presented in chapter 2 provided a detailed review of the current state of art for robotic design environments, grammatical evolution, and interesting behaviours for robotics. The chapter also highlighted a number of specific issues for robotic design, which were subsequently addressed in chapters 3, 4, and 5.

6.2.2 Second objective

The second objective was explored in chapter 3 and chapter 4. Chapter 3 has described the design and implementation of a robotic design environment for Lego Mindstorms NXT robots called NXTLogo where the robot can be controlled and linked directly to simulators in the agent-oriented programming language software, NetLogo. The technologies that the robotic design environment relies on are heterogeneous since it is built in Java which runs on the JVM on all platforms as does NetLogo, being free to download and able to run on any platform. Also, the front end of the system is written in NetLogo, which is known for its simplicity of code. The robotic design environment does not require fewer programming skills from the end-user in order to implement behaviours (developed in NetLogo) on the robot. Significant results can be achieved through minimal code and drag-and-drop interface creation for the controlling of simulations. This chapter was submitted and accepted as a conference paper.

Chapter 4 reported the experiments that were conducted on the use of grammatical evolution applied to the problem of evolving behaviours for the task of maze exploration. The accomplishments of this chapter included: the design and construction of a Lego Mindstorms robot; updating of a NetLogo extension; the creation of a NetLogo application to evolve solutions; and the creation of a NetLogo application to interface between the first application and the NXT Lego Mindstorms brick. The robot was designed and constructed to examine the performance of behaviours in the real-world. The low-cost platform NXTLogo was updated to be compatible with NetLogo 6.1 API, in order

to make use of NetLogo for the evolution of behaviours for Lego Mindstorms robots which can guide the robots to navigate an environment in search of an abstract goal.

A NetLogo application was developed to grammatically evolve the codes for simulated agents embodied within a virtual environment. The approach was evaluated using three different simulated Maze environments and the robot was inserted into these unknown environments without any internal memory mechanisms. The application evaluated each individual of those robots with a fitness function to determine the fittest individual and stored the fittest individual's phenotype. A NetLogo application was used as an interface to deploy the fittest individual's phenotype into a Lego Mindstorms robot in order to test it in the real-world. The results showed that the more generalised solutions developed for Maze III outperformed all other solutions.

6.2.3 Third objective

Chapter 5 explored the third objective. To discover the novel interesting robotic behaviours, the robotic system environment was customised and upgraded. A new grammar to run behaviours was created after analysing the new test-environment which was developed in chapter 5 as well. The fitness function was modified as well, and an entropy-based measure was used as the fitness function to identify interesting robotic behaviours. The grammar of these identified robotic behaviours was extracted for validation in the real-world.

6.2.4 Fourth objective

The fourth objective was also explored in chapter 5. A new robot was designed and equipped with sensors and motors according to the requirements of the design and test-environment. A new interface application was also developed to facilitate the new features of the robotic system environment. This was followed by a series of applications for troubleshooting purposes. The grammar was successfully extracted to the robot and its behaviour in the real-world was observed to validate the novel interesting robotic behaviours.

The results proved that the robotic system environment can discover and validating novel interesting robotic behaviours.

6.3 Limitations

It is important to consider the problems encountered in the real-life experiments using the Lego Mindstorms NXT devices for the robotic design system and potential limitations. There are some limitations encountered which affect a single model, however, the issues which have more to do with the system and can affect multiple applications will be discussed here.

Chapter 3 showed that the basic functionality of the robotic design environment fulfils objective 2 *"To develop an integrated low-cost environment for evolutionary robotics experiments"*. However, some improvements can be made such as the extending the system to work with a swarm of robots which can lead to a whole new range of experiments. A series of control files based on multithreading can be implemented for the integrity and efficiency of the system. However, its effect on Mindstorms NXT and efficiency needs to be investigated before making further conclusions.

During the subsumption architecture roaming experiments, a calibration problem was noted which refers to the factors that can affect the accuracy of colour detection. These factors include the texture of the experiment floor, conditions of light, and colour shade. The performance of the model depends on these factors because the robot cannot tell if the detection has been made correctly or not. For these purposes, we need NetLogo to monitor its detections in real-time.

Chapter 4 produced some interesting results in relation to the maze exploration. The results showed that generalised solutions developed for more complex environments outperformed all other solutions. More experiments need to be conducted to see if this effect is mainly due to the length of the evolutionary runs used for the different environments that were investigated

or is more due to general solutions being better suited to a wider range of environments.

Another common problem found in such devices is jitters which can reduce the quality of results. Especially when jitter is combined with the problem of delay, it is possible that the robot can keep going forward even after detecting an obstacle because data is yet to be communicated and the robot may end up hitting the obstacle. To resolve the issue, a backup sensor (bump sensor) was added to the robot which affected the battery life and increased the computation cost.

Chapter 5 further presented the capabilities of the robotic design system such as it is easy to modify, easy to extend, and capable of discovering and validating interesting robotic behaviours. However, there were some limitations in matching the simulation results to the real-life test environment. These include the roughness of the floor in the test-environment, colours overlapping, misreading of colours, the robot being stuck under the paper and the effect that the lighting condition has on the level of colour detection accuracy.

The main problems noted during these experiments was excessive turn angles and battery drainage. The turn angles problem resulted after a change in the robot design. This was solved easily by just making a change in the turn angles settings of the robot. However, the validation of these experiments depended on the correct simulation of the turn the robot takes, therefore, it was an important problem to mitigate any mismatch to get better results. An alternative battery source to improve energy usage during the experiments would provide better results.

Although the new robotic design system has some limitations, it can be used to rapidly design new robotic test configurations, simulating and validating robotic behaviours, identifying interesting robotic behaviours while being low-cost and easy to customise. Some ideas to extend and implement in this robotic design system are presented in the next section.

6.4 Future Work

The results of experiments performed in this thesis have proved that it is possible to develop an integrated low-cost agent-based robotic design system for simulating and validating the design of interesting robotic behaviours. While objectives 2, 3, and 4 has been met, there are areas in which the work could be extended, for example, the extension of the system to involve multiple robots. A series of control files based on multi-threading can be implemented for the integrity and efficiency of the system. However, its effect and efficiency need to be investigated before making further conclusions.

A further upgrade to the software is required to make it work with EV3 the latest robot introduced by Lego Mindstorms. This upgrade should be simple and straight-forward but could not be performed in this thesis due to funding issues and time constraints.

The new robotic system can discover novel interesting robotic behaviours and validate those behaviours in the real-world. Although, the results help to answer the research questions asked in this thesis, however, further experiments are needed to be conducted in more complex environments to confirm the findings. The experiments performed in this thesis do demonstrate the capabilities of this robotic design system and its easy to customise design, however, it would be interesting to see if this system could be adapted to design aerial vehicles, for example for the use of search and rescue purposes. There are many possibilities which can be explored using this robotic design system. Due to its low-cost, this makes it more accessible to educational institutions where students can easily come up with many new ideas without great expense in terms of both money and time. The students will benefit from performing their experiments in the real-world as well as in simulations at the same time.

References

- [1] T. Abukhalil, M. Patil, S. Patel and T. Sobh, 'Deployment Environment for a Swarm of Heterogeneous Robots,' *Robotics*, vol. 5, no. 4, pp. 1–22, 2016. DOI: 10.3390/robotics5040022, MDPI (p. 16).
- [2] N. Ahmed, 'Discovering Interesting Behaviours in Complex Systems,' Ph.D. dissertation, Bangor University (United Kingdom), 2019 (pp. xi, 43, 45, 46, 90).
- [3] N. Ahmed and W. J. Teahan, 'Using Compression to Discover Interesting Behaviours in a Hybrid Braitenberg Vehicle,' *IEEE Access*, vol. 9, pp. 11 316–11 327, 2021, IEEE (pp. 45, 95, 123).
- [4] M. Akhtaruzzaman and A. A. Shafie, 'Evolution of Humanoid Robot and Contribution of Various Countries in Advancing the Research and Development of the Platform,' *ICCAS 2010 – International Conference on Control, Automation and Systems*, pp. 1021–1028, 2010. DOI: 10.1109/ICCAS.2010.5669646, IEEE (pp. 9, 10).
- [5] R. J. Alattas, S. Patel and T. M. Sobh, 'Evolutionary Modular Robotics: Survey and Analysis,' *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3-4, pp. 815–828, 2019, Springer (p. 23).
- [6] Z. A. Algfoor, M. S. Sunar and H. Kolivand, 'A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games,' *International Journal of Computer Games Technology*, pp. 1–11, 2015, Hindawi (p. 75).
- [7] S. Amarteifio, 'Interpreting a Genotype-Phenotype Map with Rich Representations in XMLGE,' Ph.D. dissertation, University of Limerick, 2005 (pp. 27, 69).
- [8] S. Aslam, *NetLogo Source Code Used for Experiments Developed for this Thesis*, <https://github.com/Syed-72/A-robotic-design->

system-to-design-robots-using-NetLogo-and-LEGO-Mindstorms,
Accessed: 16-09-2021 (p. 97).

- [9] S. K. Aslam, W. J. Faithful and W. J. Teahan, 'A Middleware to Link Lego Mindstorms Robots with 4th Generation Language Software NetLogo,' in *Artificial Intelligence XXXV 38th SGAI International Conference on Artificial Intelligence, AI 2018*, 2018, pp. 416–430, Springer (p. 71).
- [10] J. E. Auerbach and J. C. Bongard, 'Environmental Influence on the Evolution of Morphological Complexity in Machines,' *Publis Library of Science (PLoS) Computational Biology*, vol. 10, no. 1, 2014, Public Library of Science (pp. 2, 23).
- [11] R. P. Bachega, R. Pires and A. B. Campo, 'Force Sensing to Control a Bio-Inspired Walking Robot,' *International Federation of Automatic Control Proceedings Volumes*, vol. 46, no. 7, pp. 105–109, 2013. DOI: 10.3182/20130522-3-br-4036.00052, Elsevier (pp. 10, 51).
- [12] C. Bagavathi and O. Saraniya, 'Evolutionary Mapping Techniques for Systolic Computing System,' in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, 2019, pp. 207–223, Elsevier (p. 24).
- [13] L. A. Ballard, S. Sabanovic, J. Kaur and S. Milojevic, 'George Charles Devol, Jr. [History],' *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 114–119, 2012, IEEE (p. 51).
- [14] A. Behrens *et al.*, 'MATLAB Meets LEGO Mindstorms—A freshman Introduction Course Into Practical Engineering,' *IEEE Transactions on Education*, vol. 53, no. 2, pp. 306–317, 2009, IEEE (p. 53).
- [15] F. Benureau, 'Self Exploration of Sensorimotor Spaces in Robots,' Ph.D. dissertation, Université de Bordeaux, 2015 (p. 42).
- [16] N. Bezzo, A. Mehta, C. D. Onal and M. T. Tolley, 'Robot Makers,' *IEEE Robotics and Automation Magazine*, vol. 22, no. 4, pp. 27–36, 2015, ISSN: 10709932. DOI: 10.1109/MRA.2015.2482838, IEEE (p. 13).
- [17] S. Bhardwaj, A. Warbhe and B. R. Kumar, 'Sensor System Implementation for Unmanned Aerial Vehicles,' *Indian Journal of Science and Technology*, vol. 8, no. 2, pp. 7–11, 2015. DOI: 10.17485/ijst/2015/v8iS2/57790, Indian Society of Education and Environment (pp. 73, 93).

- [18] G. Biggs and B. MacDonald, 'A Survey of Robot Programming Systems,' in *Proceedings of the Australasian Conference on Robotics and Automation*, 2003 (p. 16).
- [19] A. Björkelund, H. Bruyninckx, J. Malec, K. Nilsson and P. Nugues, 'Knowledge for Intelligent Industrial Robots,' in *AAAI Spring Symposium: Designing Intelligent Robots*, 2012, AAAI (p. 55).
- [20] J. Blanchard, F. Guillet, R. Gras and H. Briand, 'Using Information-Theoretic Measures to Assess Association Rule Interestingness,' in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, 2005, IEEE (p. 42).
- [21] A. Brabazon, M. O'Neill and I. Dempsey, 'An Introduction to Evolutionary Computation in Finance,' *IEEE Computational Intelligence Magazine*, vol. 3, no. 4, pp. 42–55, 2008, IEEE (pp. 27, 69).
- [22] R. Brooks, 'A Robust Layered Control System For A Mobile Robot,' *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986, IEEE (pp. 13, 40, 52, 63).
- [23] E. Burattini, A. de Francesco and M. De Gregorio, 'NSL: A Neuro-Symbolic Language for a Neuro-Symbolic Processor (NSP),' *International Journal of Neural Systems*, vol. 13, no. 02, pp. 93–101, 2003, World Scientific (p. 64).
- [24] A. Cangelosi and M. Schlesinger, *Developmental Robotics: From Babies to Robots*. MIT press, 2015 (pp. 2, 23).
- [25] D. R. Carvalho, A. A. Freitas and N. Ebecken, 'Evaluating the Correlation Between Objective Rule Interestingness Measures and Real Human Interest,' in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2005, pp. 453–461 (pp. 41, 42).
- [26] S. F. Chen and J. Goodman, 'An Empirical Study of Smoothing Techniques for Language Modeling,' *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999, Academic Press (p. 96).
- [27] X. Chen, J. Chase and Y. Chen, *Mobiles Robots-Past Present and Future*. IntechOpen, 2009 (p. 8).

- [28] R. Cleary, 'Extending Grammatical Evolution with Attribute Grammars: An Application to Knapsack Problems,' M.S. thesis, University of Limerick, Ireland, 2005 (p. 28).
- [29] K. Cooper and L. Torczon, *Engineering a Compiler*. Elsevier, 2011 (p. 29).
- [30] K. Deb, *Optimization For Engineering Design: Algorithms and Examples*. PHI Learning Pvt. Ltd., 2012 (p. 25).
- [31] I. Dempsey, M. O'Neill and A. Brabazon, 'Constant Creation in Grammatical Evolution,' *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 23–38, 2007, Inderscience (pp. 27, 69).
- [32] I. Dempsey, M. O'Neill and A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, 2009 (p. 27).
- [33] G. N. DeSouza and A. C. Kak, 'Vision for Mobile Robot Navigation: A Survey,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237–267, 2002, IEEE (p. 16).
- [34] S. Dhar, V. Ordonez and T. L. Berg, 'High Level Describable Attributes for Predicting Aesthetics and Interestingness,' in *Conference on Computer Vision and Pattern Recognition (CVPR) 2011*, IEEE, 2011, pp. 1657–1664 (p. 90).
- [35] M. A. Diftler et al., 'Robonaut 2 - The First Humanoid Robot in Space,' in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2178–2183 (pp. 10, 11).
- [36] S. Doncieux, J.-B. Mouret, N. Bredeche and V. Padois, 'Evolutionary Robotics: Exploring New Horizons,' in *New Horizons in Evolutionary Robotics*, Springer, 2011, pp. 3–25 (p. 23).
- [37] P. R. Doob, *The Idea of the Labyrinth from Classical Antiquity through the Middle Ages*. Cornell University Press, 2019 (p. 37).
- [38] S. Elfwing and K. Doya, 'Emergence of Polymorphic Mating Strategies in Robot Colonies,' *Public Library of Science (PloS) one*, vol. 9, no. 4, 2014, Public Library of Science (PLOS) (pp. 2, 23).
- [39] A. Elkady and T. Sobh, 'Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,' *Journal of*

- Robotics*, vol. 2012, pp. 1–15, 2012, ISSN: 1687-9600. DOI: 10.1155/2012/959013, Hindawi (pp. xi, 17, 18).
- [40] P. G. Espejo, S. Ventura and F. Herrera, 'A Survey on the Application of Genetic Programming to Classification,' *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 2, pp. 121–144, 2009, IEEE (p. 26).
 - [41] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg and M. O'Neill, 'PonyGE2: Grammatical Evolution in Python,' in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 1194–1201 (p. 31).
 - [42] D. Floreano and L. Keller, 'Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection,' *Publis Library of Science (PLoS) Biology*, vol. 8, no. 1, pp. 1–8, 2010, Public Library of Science (PLoS) (pp. 2, 23).
 - [43] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT press, 2008 (pp. 2, 23).
 - [44] B. R. Gaines, 'Transforming Rules and Trees into Comprehensible Knowledge Structures,' *Advances in Knowledge Discovery and Data Mining*, pp. 205–226, 1996, Citeseer (p. 42).
 - [45] A. Gasparetto and L. Scalera, 'A Brief History of Industrial Robotics in the 20th Century,' *Advances in Historical Studies*, vol. 08, no. 01, pp. 24–35, 2019, ISSN: 2327-0438. DOI: 10.4236/ahs.2019.81002 (pp. 9, 10).
 - [46] L. Georgiou and W. J. Teahan, 'Implications of Prior Knowledge and Population Thinking in Grammatical Evolution: Toward a Knowledge Sharing Architecture,' *World Scientific and Engineering Academy and Society (WSEAS) Transactions on Systems*, vol. 5, no. 10, pp. 2338–2345, 2006, WSEAS (p. 34).
 - [47] L. Georgiou and W. J. Teahan, 'jGE-A java Implementation of Grammatical Evolution,' in *10th World Scientific and Engineering Academy and Society (WSEAS) International Conference on Systems, Athens, Greece*, WSEAS, 2006, pp. 406–411 (pp. 31, 33).

- [48] L. Georgiou and W. J. Teahan, 'Constituent Grammatical Evolution,' in *Twenty-Second International Joint Conference on Artificial Intelligence, AAAI*, 2012, pp. 1261–1268 (p. 34).
- [49] P. Gil, I. Maza, A. Ollero and P. Marrón, 'Data Centric Middleware for the Integration of Wireless Sensor Networks and Mobile Robots,' in *7th Conference on Mobile Robots and Competitions, ROBOTICA*, IEEE Robotics and Automation Society, 2007 (p. 21).
- [50] K. Goris, 'Autonomous Mobile Robot Mechanical Design,' Ph.D. dissertation, Vrije University Brussels, 2005 (pp. 12, 13).
- [51] E. Guizzo, *What Is a Robot? - ROBOTS: Your Guide to the World of Robotics*. [Online]. Available: <https://robots.ieee.org/learn/> (visited on 12th May 2020) (p. 12).
- [52] S. Gupta and E. Singla, 'Evolutionary Robotics in Two Decades: A Review,' *Sadhana*, vol. 40, no. 4, pp. 1169–1184, 2015, Springer (pp. xi, 24–26).
- [53] S. Hadim, J. Al-Jaroodi and N. Mohamed, 'Trends in Middleware for Mobile Ad Hoc Networks,' *The Journal of Communications*, vol. 1, no. 4, pp. 11–21, 2006, ACADEMY (p. 16).
- [54] S. Hadim and N. Mohamed, 'Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks,' *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 0603–0626, 2006, IEEE (p. 16).
- [55] P. Hall and S. C. Morton, 'On The Estimation of Entropy,' *Annals of the Institute of Statistical Mathematics*, vol. 45, no. 1, pp. 69–88, 1993, Springer (p. 42).
- [56] I. Harvey, P. Husbands, D. Cliff, A. Thompson and N. Jakobi, 'Evolutionary Robotics: The Sussex Approach,' *Robotics and Autonomous Systems*, vol. 20, no. 2-4, pp. 205–224, 1997, Elsevier (pp. 2, 23).
- [57] I. Harvey, E. D. Paolo, R. Wood, M. Quinn and E. Tuci, 'Evolutionary Robotics: A New Scientific Tool for Studying Cognition,' *Artificial Life*, vol. 11, no. 1-2, pp. 79–98, 2005, MIT Press (pp. 2, 23).
- [58] P. He, Z. Deng, C. Gao, X. Wang and J. Li, 'Model Approach to Grammatical Evolution: Deep-Structured Analyzing of Model and

- Representation,' *Soft Computing*, vol. 21, no. 18, pp. 5413–5423, 2017, Springer (p. 70).
- [59] W. He, W. Ge, Y. Li, Y.-J. Liu, C. Yang and C. Sun, 'Model Identification and Control Design for a Humanoid Robot,' *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 45–57, 2016, IEEE (pp. 10, 11).
- [60] C. Headleand, A. Cenydd and W. Teahan, 'Benchmarking Grammar-Based Genetic Programming Algorithms,' in *Research and Development in Intelligent Systems XXXI. SGAI 2014*, Springer, Oct. 2014. DOI: 10.1007/978-3-319-12069-0_9 (pp. xi, 38, 39).
- [61] E. Hemberg, C. Gilligan, M. O'Neill and A. Brabazon, 'A Grammatical Genetic Programming Approach to Modularity in Genetic Algorithms,' in *European Conference on Genetic Programming*, Springer, 2007, pp. 1–11 (pp. 27, 69).
- [62] M. Henning, 'A new approach to object-oriented middleware,' *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, 2004, IEEE (p. 19).
- [63] R. J. Hilderman and H. J. Hamilton, *Knowledge Discovery and Interestingness Measures: A Survey*. CiteSeer, 1999 (p. 42).
- [64] J. Hugosson, E. Hemberg, A. Brabazon and M. O'Neill, 'Genotype Representations in Grammatical Evolution,' *Applied Soft Computing*, vol. 10, no. 1, pp. 36–43, 2010, Elsevier (p. 36).
- [65] F. Hussain, H. Liu, E. Suzuki and H. Lu, 'Exception Rule Mining with a Relative Interestingness Measure,' in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2000, pp. 86–97 (p. 42).
- [66] D. Jefferson *et al.*, 'The Genesys System: Evolution as a Theme in Artificial Life,' in *Proceedings of Second Conference on Artificial Life*, Addison-Wesley Publishing Company, 1992, pp. 549–578 (pp. 34, 36).
- [67] J. L. Jones, 'Robots at the Tipping Point: The Road to the iRobot Roomba,' *IEEE Robotics & Automation Magazine*, vol. 13, no. 1, pp. 76–78, 2006, IEEE (pp. 10, 11).
- [68] T. Karakurt, A. Durdu and N. Yilmaz, 'Design of Six Legged Spider Robot and Evolving Walking Algorithms,' *International Journal of Machine Learning and Computing*, vol. 5, no. 2, pp. 96–100, 2015. DOI: DOI :

- 10.7763/IJMLC.2015.V5.490, International Association of Computer Science and Information Technology (p. 51).
- [69] R. Karunasena, P. Sandarenu, M. Pinto, A. Athukorala, R. Rodrigo and P. Jayasekara, 'DEVI: Open-Source Human-Robot Interface for Interactive Receptionist Systems,' in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, 2019, pp. 378–383 (pp. 10, 11).
 - [70] H. Kashyap, H. A. Ahmed, N. Hoque, S. Roy and D. K. Bhattacharyya, 'Big Data Analytics in Bioinformatics: A Machine Learning Perspective,' *ArXiv*, vol. 13, no. 9, pp. 1–20, 2015, arXiv (p. 24).
 - [71] G. Keramidas *et al.*, 'Computation and Communication Challenges to Deploy Robots in Assisted Living Environments,' in *2016 Design, Automation & Test in Europe Conference & Exhibition*, IEEE, 2016, pp. 888–893 (p. 14).
 - [72] J. M. Kerridge, A. Panayotopoulos and P. Lismore, 'JCSPre: The Robot Edition to Control LEGO NXT Robots,' in *Communicating Process Architectures*, 2008, pp. 255–270 (pp. 53, 54).
 - [73] S. H. Kim and J. W. Jeon, 'Programming LEGO Mindstorms NXT with Visual Programming,' *ICCAS 2007 - International Conference on Control, Automation and Systems*, pp. 2468–2472, 2007. DOI: 10.1109/ICCAS.2007.4406778 (pp. 8, 10, 71).
 - [74] J. R. Koza, *Genetic Programming: On The Programming Of Computers by Means of Natural Selection*. MIT press, 1992, vol. 1 (pp. xi, 26, 28, 34–36).
 - [75] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT press, 1994 (pp. 34, 35).
 - [76] J. Kramer and M. Scheutz, 'Development Environments for Autonomous Mobile Robots: A Survey,' *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007, Springer (pp. 16, 17).
 - [77] M. Kranz, R. B. Rusu, A. Maldonado, M. Beetz and A. Schmidt, 'A Player/Stage System for Context-Aware Intelligent Environments,' *Proceedings of System Support for Ubiquitous Computing Workshop (UbiSys)*, vol. 6, no. 8, pp. 17–21, 2006, IEEE (p. 18).

- [78] A. Krishna, K. Nandan, S. P. Kumar, K. Srihari and P. Sivraj, 'Design and Fabrication of a Hexapod Robot,' in *2014 International Conference on Embedded Systems (ICES)*, IEEE, 2014, pp. 225–230 (p. 10).
- [79] W. B. Langdon and R. Poli, 'Better Trained Ants for Genetic Programming,' *School of Computer Science Research Reports–University of Birmingham CSR*, 1998, University of Birmingham (p. 36).
- [80] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. Springer Science & Business Media, 2013 (pp. 34, 36).
- [81] C. Leger, *Darwin2K: An Evolutionary Approach to Automated Design for Robotics*. Springer Science & Business Media, 2012 (pp. 2, 23).
- [82] J. Lehman and K. O. Stanley, 'Evolvability is Inevitable: Increasing Evolvability without the Pressure to Adapt,' *Public Library of Science (PLOS) one*, vol. 8, no. 4, 2013, Public Library of Science (PLOS) (pp. 2, 23).
- [83] M. W. Lew, T. B. Horton and M. S. Sherriff, 'Using LEGO Mindstorms NXT and LeJOS in an Advanced Software Engineering Course,' *2010 23rd IEEE Conference on Software Engineering Education and Training*, pp. 121–128, 2010, IEEE Computer Society (p. 71).
- [84] H. Lipson and J. Pollack, 'Automatic Design and Manufacture of Robotic Lifeforms,' *Nature*, vol. 406, no. 6799, pp. 974–978, 2000, Nature Publishing Group (pp. 2, 23).
- [85] N. Lourenço, F. B. Pereira and E. Costa, 'Unveiling the Properties of Structured Grammatical evolution,' *Genetic Programming and Evolvable Machines*, vol. 17, no. 3, pp. 251–289, 2016, Springer (pp. 2, 23, 27, 31, 36, 69).
- [86] N. Lourenço, F. B. Pereira and E. Costa, 'The Importance of the Learning Conditions in Hyper-Heuristics,' in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, Association for Computing Machinery, 2013, pp. 1525–1532 (p. 31).
- [87] P. Maes, 'Modeling Adaptive Autonomous Agents,' *Artificial Life*, vol. 1, no. 1_2, pp. 135–162, 1993, MIT Press (p. 40).
- [88] A. Makarenko, A. Brooks and T. Kaupp, 'Orca: Components for Robotics,' in *International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 163–168 (p. 19).

- [89] A. Mayor, *The World's First Robot: Talos*. [Online]. Available: <http://www.wondersandmarvels.com/2012/03/the-worlds-first-robot-talos.html> (visited on 12th May 2020) (p. 9).
- [90] D. McDonald, *Early Los Altos and Los Altos Hills*. Arcadia Publishing, 2010 (p. 37).
- [91] K. McGarry, 'A Survey of Interestingness Measures for Knowledge Discovery,' *The Knowledge Engineering Review*, vol. 20, no. 1, pp. 39–61, 2005, Cambridge University Press (p. 42).
- [92] L. Meeden and D. Kumar, 'Trends in Evolutionary Robotics,' in *Soft Computing for Intelligent Robotic Systems*, Springer, 1998, pp. 215–233 (p. 8).
- [93] O. Michel, 'WebotsTM: Professional Mobile Robot Simulation,' *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004, ISSN: 1729-8806. arXiv: 0412052 [cs]. [Online]. Available: <http://arxiv.org/abs/cs/0412052> (p. 20).
- [94] M. Mizukawa et al., 'ORiN: Open Robot Interface for the Network-The Standard and Unified Network Interface for Industrial Robot Applications-,' in *Proceedings of the 41st Society of Instrument and Control Engineers (SICE) Annual Conference*, IEEE, vol. 2, 2002, pp. 925–928 (p. 20).
- [95] N. Mohamed, J. Al-Jaroodi and I. Jawhar, 'Middleware for Robotics: A Survey,' *2008 IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 736–742, 2008. DOI: 10.1109/RAMECH.2008.4681485, IEEE (pp. 8, 17, 18).
- [96] S. Mohanty, S. S. Samal and S. Sathyamurthy, 'Futuristic Humanoid Robot of Twenty First Century,' *International Conference on "Emerging Trends in Robotics and Communication Technologies", INTERACT-2010*, pp. 185–188, 2010. DOI: 10.1109/INTERACT.2010.5706223, IEEE (p. 9).
- [97] V. Müller, 'Is There a Future for AI Without Representation?' *Minds and Machines*, vol. 17, no. 1, pp. 101–115, 2007, Springer Science and Business Media (p. 52).

- [98] E. Murphy, M. O'Neill, E. Galván-López and A. Brabazon, 'Tree-Adjunct Grammatical Evolution,' in *IEEE Congress on Evolutionary Computation*, IEEE, 2010, pp. 1–8 (p. 28).
- [99] M. Nader, A. Jameela and J. Imad, 'A Review of Middleware for Networked Robots,' *International Journal of Computer Science and Network Security*, vol. 9, no. 5, pp. 139–148, 2009, ACADEMIA (p. 17).
- [100] M. Namoshe, N. Tlale, C. Kumile and G. Bright, 'Open Middleware for Robotics,' in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, IEEE, 2008, pp. 189–194 (p. 17).
- [101] E. Naredo, P. Urbano and L. Trujillo, 'The Training Set and Generalization in Grammatical Evolution for Autonomous Agent Navigation,' *Soft Computing*, vol. 21, no. 15, pp. 4399–4416, 2017, Springer (p. 70).
- [102] A. L. Nelson, G. J. Barlow and L. Doitsidis, 'Fitness Functions in Evolutionary Robotics: A Survey and Analysis,' *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, 2009, Elsevier (pp. 2, 23).
- [103] M. Nicolau and D. Slattery, *LibGE Documentation. Bio Computing Developmental Systems Centre, University of Limerick, Version 0.27 Alpha1 edition*, 2006. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=536845FD9ABE36259B7848921129824E?doi=10.1.1.153.962&rep=rep1&type=pdf> (visited on 12th Jun. 2020) (pp. 31, 33).
- [104] D. K. Nithin and P. B. Sivakumar, 'Generic Feature Learning in Computer Vision,' *Procedia Computer Science*, vol. 58, pp. 202–209, 2015, Elsevier (p. 24).
- [105] S. Nolfi, J. Bongard, P. Husbands and D. Floreano, 'Evolutionary Robotics,' in *Springer Handbook of Robotics*, Springer, 2016, pp. 2035–2068 (pp. 2, 23).
- [106] F. Noorian, A. M. de Silva, P. H. Leong *et al.*, 'gramEvol: Grammatical Evolution in R,' *Journal of Statistical Software*, vol. 71, no. 1, pp. 1–26, 2016, Foundation for Open Access Statistics (pp. 31, 34).
- [107] T. Nyathi and N. Pillay, 'Comparison of a Genetic Algorithm to Grammatical Evolution for Automated Design of Genetic Programming

- Classification Algorithms,' *Expert Systems with Applications*, vol. 104, pp. 213–234, 2018, Elsevier (p. 29).
- [108] M. O'Neill, 'Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution,' Ph.D. dissertation, University of Limerick, 2001 (pp. 27, 69).
- [109] M. O'Neill and C. Ryan, 'Grammatical Evolution,' *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001, IEEE (pp. 34, 35, 76, 94).
- [110] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. Mcdermott and A. Brabazon, 'GEVA: Grammatical Evolution in Java,' *Association for Computing Machinery SIGEVolution*, vol. 3, no. 2, pp. 17–22, Jul. 2008. DOI: 10.1145/1527063.1527066, Association for Computing Machinery (pp. xi, 28, 31, 32).
- [111] M. O'Neill and C. Ryan, 'Under the Hood of Grammatical Evolution,' in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, Association for Computing Machinery, 1999, pp. 1143–1148 (p. 28).
- [112] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003, vol. 4, ISBN: 1402074441. DOI: 10.1007/978-1-4615-0447-4 (pp. 34, 35).
- [113] M. O'Neill et al., 'Shape Grammars and Grammatical Evolution for Evolutionary Design,' in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, Association for Computing Machinery, 2009, pp. 1035–1042, ISBN: 9781605583259 (p. 28).
- [114] M. O'Neil and C. Ryan, 'Grammatical evolution,' in *Grammatical Evolution*, Springer, 2003, pp. 33–47 (p. 27).
- [115] M. O'Neill and C. Ryan, 'Automatic Generation of Caching Algorithms,' *Evolutionary Algorithms in Engineering and Computer Science*, vol. 30, pp. 127–134, 1999, John Wiley & Sons (p. 30).
- [116] M. O'Neill and C. Ryan, 'Automatic Generation of High Level Functions using Evolutionary Algorithms,' *Proceedings of the First International Workshop on Soft Computing Applied to Software Engineering*, pp. 21–29, 1999 (p. 28).

- [117] M. O'Neill and C. Ryan, 'Automatic Generation of Programs with Grammatical Evolution,' *Proceedings of Artificial Intelligence and Computer Science (AICS)*, pp. 72–78, 1999, IEEE (p. 30).
- [118] A. Ortega, M. De La Cruz and M. Alfonseca, 'Christiansen Grammar Evolution: Grammatical Evolution With Semantics,' *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 1, pp. 77–90, 2007, IEEE (p. 28).
- [119] S. Oßwald, M. Bennewitz, W. Burgard and C. Stachniss, 'Speeding-Up Robot Exploration by Exploiting Background Information,' *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 716–723, 2016, IEEE (p. 90).
- [120] D. Parker, *Projects for NXT 2.0*, <https://www.nxtprograms.com/index2.html>, Accessed: 13-04-2021 (pp. xii, 91).
- [121] S. Pavel, *Grammatical Evolution Ruby Exploratory Toolkit (GERET)*, 2009. [Online]. Available: <http://geret.org/> (visited on 12th Jun. 2020) (p. 31).
- [122] C. Peabody and J. H. Seitzer, 'GEF: A Self-Programming Robot Using Grammatical Evolution,' *Twenty-Ninth AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015, Association for the advancement of Artificial Intelligence (AAAI) (p. 31).
- [123] F. Peng and D. Schuurmans, 'Combining Naive Bayes and n-Gram Language Models for Text Classification,' in *European Conference on Information Retrieval*, Springer, 2003, pp. 335–350 (p. 96).
- [124] M. C. I. PÉREZ and J. Grabulosa, 'An Overview of Behavioural-Based Robotics with Simulated Implementations on an Underwater Vehicle,' Ph.D. dissertation, Institut d'Informàtica i Aplicacions Universitat de Girona, Girona, 2000 (pp. xi, 40, 41).
- [125] D. Perez-Liebana and M. Nicolau, 'Evolving Behaviour Tree Structures Using Grammatical Evolution,' in *Handbook of Grammatical Evolution*, Springer, 2018, pp. 433–460 (pp. 27, 31, 69).
- [126] R. Poli, W. B. Langdon, N. F. McPhee and J. R. Koza, *A Field Guide to Genetic Programming*. Lulu.com, 2008 (pp. 24, 26).
- [127] D. K. Pratihari, *Soft Computing: Fundamentals and Applications*. Alpha Science International, Ltd, 2013 (p. 25).

- [128] F. Proctor, S. Balakirsky, Z. Kootbally, T. Kramer, C. Schlenoff and W. Shackleford, 'The Canonical Robot Command Language (CRCL),' *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 495–502, 2016, Emerald Group Publishing Limited (p. 8).
- [129] M. Quinn, L. Smith, G. Mayley and P. Husbands, 'Evolving Controllers for a Homogeneous System of Physical Robots: Structured Cooperation with Minimal Sensors,' *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1811, pp. 2321–2343, 2003, The Royal Society (pp. 2, 23).
- [130] J. Rissanen and G. G. Langdon, 'Arithmetic Coding,' *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979, IBM (p. 96).
- [131] S. H. Robertson *et al.*, 'Multiscale Computational Analysis of *Xenopus Laevis* Morphogenesis Reveals Key Insights of Systems-Level Behavior,' *BioMed Central (BMC) Systems Biology*, vol. 1, no. 1, p. 46, 2007, BioMed Central (p. 53).
- [132] ROS.org, *Robot Operating System (ROS)*. [Online]. Available: <https://www.ros.org/> (visited on 12th May 2020) (p. 20).
- [133] L. Royakkers and R. van Est, 'A Literature Review on New Robotics: Automation from Love to War,' *International Journal of Social Robotics*, vol. 7, no. 5, pp. 549–570, 2015, Springer (p. 73).
- [134] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning Representations by Back-Propagating Errors,' *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, Nature Publishing Group (p. 45).
- [135] C. Ryan, J. Collins and M. O'Neill, 'Grammatical Evolution: Evolving Programs for an Arbitrary Language,' *In: Proceedings of the First European Workshop on Genetic Programming*, vol. 1391, no. 1, pp. 83–96, 1998, Springer (pp. 27, 29, 69, 70).
- [136] C. Ryan, M. O'Neill and J. Collins, 'Grammatical Evolution: Solving Trigonometric Identities,' in *Proceedings of Mendel: 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets held in Brno, Czech Republic*, vol. 98, 1998, pp. 111–119 (pp. 4, 24, 27).

- [137] A. Said, 'Introduction to Arithmetic Coding - Theory and Practice,' *Hewlett Packard Laboratories Report*, pp. 1057–7149, 2004, Academic Press (p. 96).
- [138] K. Sakamoto, M. Otsuki, T. Maeda, K. Yoshikawa and T. Kubota, 'Evaluation of Hopping Robot Performance With Novel Foot Pad Design on Natural Terrain for Hopper Development,' *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3294–3301, 2019, IEEE (pp. 10, 11).
- [139] V. Scheinman, *The Stanford Arm*. [Online]. Available: <http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm> (visited on 12th May 2020) (pp. 9, 51).
- [140] K. Y. Scheper, S. Tijmons, C. C. de Visser and G. C. de Croon, 'Behavior Trees for Evolutionary Robotics,' *Artificial Life*, vol. 22, no. 1, pp. 23–48, 2016, MIT Press (pp. 2, 23).
- [141] J. Schmidhuber, 'Driven by Compression Progress: A Simple Principle Explains Essential Aspects of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes,' in *Workshop on Anticipatory Behavior in Adaptive Learning Systems*, Springer, 2008, pp. 48–76 (pp. 42, 43).
- [142] T. Sebastian et al., 'Stanley: The Robot That Won The DARPA Grand Challenge,' *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006, Wiley InterScience (p. 51).
- [143] C. E. Shannon, 'A Mathematical Theory of Communication,' *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948 (p. 95).
- [144] W. D. Smart, 'Is a Common Middleware for Robotics Possible?' In *Proceedings of the IROS 2007 Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, vol. 1, 2007 (p. 17).
- [145] D. Smiley, *PyNeurGen: Python Neural Genetic Algorithm Hybrids*, <https://jacksonpradolima.github.io/PyNeurGen/> Accessed: 01-07-2021 (p. 31).
- [146] M. Somby, *Robotics Software Platforms Review*, <http://www.automation.com/library/articles-whitepapers/robotics/robotics-software-platforms-review>, Accessed: 11-03-2021 (p. 1).

- [147] F. Sondahl, *Solving Mazes with Genetic Programming*. [Online]. Available: <https://users.cs.northwestern.edu/~fjs750/netlogo/final/> (visited on 12th Jun. 2020) (pp. 34, 37).
- [148] W. Song, X. Zhou, P. Si and C. Wang, 'Research on Mobile Chassis System of Live Working Robot,' in *2016 4th International Conference on Mechanical Materials and Manufacturing Engineering*, Atlantis Press, 2016 (pp. 9, 10).
- [149] M. A. Sotelo-Figueroa, A. Hernández-Aguirre, A. Espinal, J. A. Soria-Alcaraz and J. Ortiz-López, 'Symbolic Regression by Means of Grammatical Evolution with Estimation Distribution Algorithms as Search Engine,' in *Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications*, Springer, 2018, pp. 169–177 (p. 34).
- [150] A. V. Spirov, 'Memetic Algorithms in Evolutionary Robotics on Example of Virtual Bots,' *International Federation of Automatic Control (IFAC)*, vol. 51, no. 30, pp. 586–591, 2018, Elsevier (p. 34).
- [151] Sri-International, *AI Center :: Shakey*. [Online]. Available: <http://www.ai.sri.com/shakey/> (visited on 12th May 2020) (pp. 9, 10, 51).
- [152] C. Stachniss, *Robotic Mapping and Exploration*. Springer, 2009, vol. 55 (p. 16).
- [153] M. Strandberg, 'Robot Path Planning: An Object-Oriented Approach,' in *Doctoral Dissertation*, sensor och system, Signaler, 2004, p. 254 (pp. 72, 92).
- [154] R. Taukeer, G. Das, A. Nath, A. Paul and J. U. Ahamed, 'Design and Fabrication of a Teleoperated Explorer Mobile Robot,' *2017 IEEE International Conference on Imaging, Vision and Pattern Recognition, icIVPR 2017*, pp. 3–8, 2017. DOI: 10.1109/ICIVPR.2017.7890890, IEEE (pp. 9, 10).
- [155] J. Tavares and F. B. Pereira, 'Automatic Design of Ant Algorithms with Grammatical Evolution,' in *European Conference on Genetic Programming*, Springer, 2012, pp. 206–217 (p. 30).
- [156] W. J. Teahan, *Artificial Intelligence-Agent Behaviour*, 2010 (p. 38).
- [157] V. Ng-Thow-Hing, T. List, K. R. Thórisson, J. Lim and J. Wormer, 'Design and Evaluation of Communication Middleware in a Distributed

- Humanoid Robot Architecture,' in *IROS'07 Workshop Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, vol. 29, 2007 (p. 20).
- [158] S. Tisue and U. Wilensky, 'Netlogo: A Simple Environment for Modeling Complexity,' in *International Conference on Complex Systems*, New England Complex Systems Institute, vol. 21, 2004, pp. 16–21 (pp. 22, 54).
- [159] S. Tjiharjadi and E. Setiawan, 'Design and Implementation of a Path Finding Robot Using Flood Fill Algorithm,' *International Journal of Mechanical Engineering and Robotics Research (IJMERR)*, vol. 5, no. 3, pp. 180–185, 2016, IJMERR (p. 71).
- [160] I. G. Tsoulos, D. Gavrilis and E. Glavas, 'Neural Network Construction using Grammatical Evolution,' in *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005.*, IEEE, 2005, pp. 827–831 (p. 30).
- [161] UCD-NCRA, *Grammatical Evolution in Java (GEVA)*, v2.0. [Online]. Available: <http://ncra.ucd.ie/Site/GEVA.html> (visited on 12th Jun. 2020) (p. 32).
- [162] UCD-NCRA, *Grammatical Evolution in MATLAB (GEM)*. [Online]. Available: <http://ncra.ucd.ie/Software.html> (visited on 12th Jun. 2020) (p. 31).
- [163] P. Urbano and L. Georgiou, 'Improving Grammatical Evolution in Santa Fe Trail using Novelty Search,' in *Artificial Life Conference Proceedings 13*, MIT Press, 2013, pp. 917–924 (pp. 35, 36).
- [164] R. Vannoy, *Designing and Building a Line Following Robot*, <https://www.cs.york.ac.uk/micromouse/Papers/Building-a-line-following-robot.pdf>, Accessed: 21-04-2021 (p. 61).
- [165] C. Wang, W. Wang, Y. Qiu, Y. Hu and S. Scherer, 'Visual Memorability for Robotic Interestingness via Unsupervised Online Learning,' in *European Conference on Computer Vision (ECCV)*, Springer, 2020 (pp. xi, xiii, 43–45, 90).
- [166] Y. Wang, 'Cognitive robots,' *IEEE Robotics Automation Magazine*, vol. 17, no. 4, pp. 54–62, 2010, IEEE (p. 52).

- [167] J. Wąs and K. Kułakowski, 'Agent-based approach in evacuation modeling,' in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, Springer, 2010, pp. 325–330 (p. 53).
- [168] M. Weser, 'Hierarchical Memory Organization of Multimodal Robot Skills for Plan-based Robot Control,' Ph.D. dissertation, University of Hamburg, 2009, p. 138 (p. 12).
- [169] T. William John, *Artificial Intelligence-Agents and Environments*, 2010 (pp. xi, 37–39).
- [170] G. Yang et al., 'The Grand Challenges of Science Robotics,' *Science Robotics*, vol. 3, no. 14, 2018, Science Robotics (pp. 15, 16).
- [171] G. Zaharija, A. Grubač and A. Granić, 'LEARN–LEgo Robot and Netlogo,' in *Contemporary Issues in Economy and Technology (CIET) 2014*, University of Split, CROATIA, 2014 (p. 22).
- [172] C. Zieliński et al., 'Reconfigurable Control Architecture for Exploratory Robots,' in *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*, IEEE, 2015, pp. 130–135 (p. 70).
- [173] T. Ziemke, 'Adaptive Behavior in Autonomous Agents,' *Presence*, vol. 7, no. 6, pp. 564–587, 1998, MIT Press (p. 40).

Appendices

Appendix A

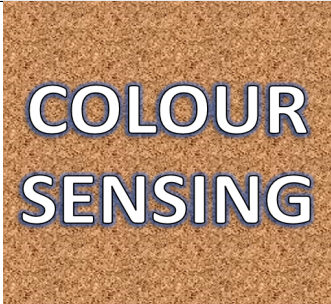


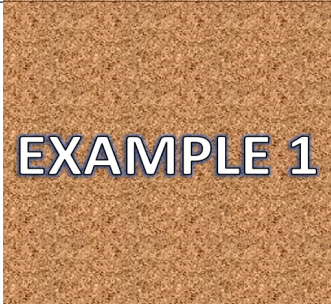
Videos of Experiments

The videos of experiments are uploaded to YouTube channel and a brief explanation is provided in the following table.

Link to YouTube Channel:

<https://www.youtube.com/channel/UCbqDprtdpk3WIthZJ7f1hVg>

Table A.1: The description of the experiments' videos uploaded to YouTube.

Description	Thumbnail
<p>Title: Colour Sensing Description: This video contains the experiments involving colour sensor adjustment and colour-threshold tolerance determination.</p>	
<p>Title: Turn Angles Description: This video contains the experiments involving Turn Angles where the robot performed turns at different degrees of angles towards left and right.</p>	
<p>Title: Simple Behaviour Description: This video contains the experiments involving a simple behaviour created by the robot to initially test the performance of the robot in the real world.</p>	
<p>Title: Example 1 Description: This video contains the experiments involving the validation of interesting behaviour created by the iFinder-Discovery model for Example 1.</p>	
<p>Title: Example 2 Description: This video contains the experiments involving the validation of interesting behaviour created by the iFinder-Discovery model for Example 2.</p>	