

# **Designing Limited Autonomous Robotic Systems**

In One Volume

**Niall McCurry B.Eng.**

(September) 2010

Submitted for the award of  
Masters in Engineering

Submitted to: Galway-Mayo Institute of Technology.  
Research Supervisor: Dr. John Owen-Jones.

Research funded under “The Irish Research Council for Science, Engineering  
and Technology” (IRCSET) Embark programme.

Submitted to the Higher Education and Training Awards Council,

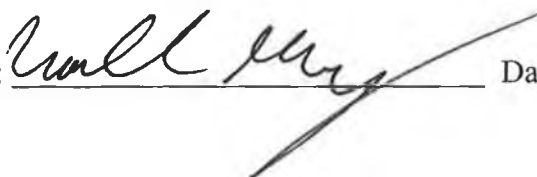


## Copyright and DAI Statement

'I hereby grant the Galway Mayo Institute of Technology or its agents the right to archive and to make available my thesis or dissertation in whole or part in the Institute libraries in all forms of media, now or here after known, subject to the provisions of the Copyright & Related Rights Act, 2000. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

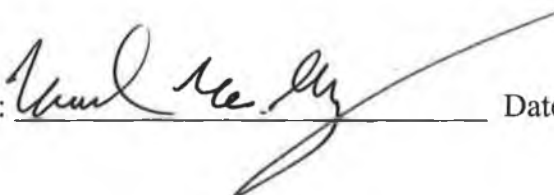
I also authorise Institute Microfilms to use the 350 word abstract of my thesis in Dissertation Abstracts International.

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.'

Signed:  Date: 24 Sep 2010

## Authenticity Statement

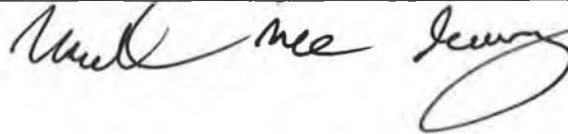
'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed:  Date: 24 Sep 2010

## Originality Statement

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at GMIT or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at GMIT or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed: Niall McCurry

A handwritten signature in black ink, written over a horizontal line. The signature is cursive and reads "Niall McCurry".

## Abstract

The aim of this research is to identify and select a cost effective platform and approach to the design and development of low cost autonomous mobile robotic systems. A requirement for such a platform is that it should be of reasonable cost, extendible and that its functionality allows for modification, with reasonable effort. Such an approach may enable organisations with limited resources in funding, limited specialist AI knowledge and scarce hardware expertise, to begin the design of robotic controllers and the application of AI to mobile robots. This solution should prove to be attractive to departments, not historically involved in developing robotic systems, to investigate this area of the application of AI allowing them to apply their own expertise to generate unique solutions to the problem sets.

By providing such a platform this research will enable developers to provide more flexible solutions for various application areas for the entertainment, industrial and domestic product markets.

The approach was to carry out an in-depth analysis of currently available solutions, modify them as appropriate and evaluate them wherever possible. Testing was carried out by designing robotic controllers for each of these platforms. The platforms investigated include both low cost hardware with embedded software solutions and software simulation environments.

The research identified several hardware candidates which featured frequently in publications, these being LEGO's MINDSTORMS RCX and iRobots Roomba platforms. The use of custom built hardware platforms was not considered for investigation.

Eight different simulators were investigated and two main candidates were selected for comparison, "Webots" from Cyberbotics Ltd and "Pyro" an open source simulation project. These two simulators used different approaches and development work and evaluations were carried out on each of them. The thesis arrives at the conclusion that "Pyro" is the most flexible approach of the two environments, but "Webots" is the more established one.

## **Acknowledgements**

I would like to thank IRCSET through their Embark Initiative for providing me with the funding for this research which allowed me to further my knowledge in the area of AI in robotics.

I would especially like to thank my supervisor Dr John Owen-Jones for his help and guidance throughout this project, and for his perseverance, which even extending into his retirement from the GMIT, in aiding me to complete this work. I would also like to thank my friends, family and girlfriend Chiara for their encouragement, help and support and furthermore for their belief that I could complete this work, with a special note to my father Frank whose advice and support were invaluable in completing this work.

I would finally like to thank Paul Dunne for his valuable assistance in the early stages of this research.

## Table of Contents

Copyright and DAI Statement.....	ii
Originality Statement .....	iii
Abstract .....	iv
Acknowledgements .....	v
Table of Contents .....	i
List of Figures .....	x
List of Tables .....	xiii
Chapter 1 - Introduction.....	1
1.1. Background .....	1
Robotic Operation .....	2
1.2. Project Goals .....	3
1.3. Overview .....	3
1.4. Thesis Structure.....	4
Chapter 2 - Literature Review.....	7
2.1. Introduction.....	7
2.2. Hardware Solutions with Embedded Software .....	7
2.2.1. Hardware Platforms .....	7
2.2.1.1. RCX .....	8
Physical Hardware of RCX Module .....	8
2.2.1.2. iRobot's Roomba and Create Robots.....	10
2.2.2. Embedded Software for Hardware Platforms .....	11
2.2.2.1. RCX Hardware Platform.....	12
LEGO® Robotics Invention System Environment.....	12
2.2.2.1.1. LeJOS Java for LEGO® Mindstorms.....	13
2.2.2.1.2. Not Quite C (NQC) .....	14
2.2.2.2. iRobot's Roomba and Create Hardware Platform .....	15
2.2.2.2.1. The Erdos Project.....	15
2.2.3. The Khepera Robot .....	17
2.3. Software Simulation Approach.....	19
2.3.1. Khepera Simulator .....	20
2.3.2. Wright State University (WSU) Java Khepera Simulator....	21
2.3.3. Evorobot.....	23

2.3.4.	Yet Another Khepera Simulator – YAKS.....	24
2.3.5.	EasyBot .....	25
2.3.6.	Webots .....	26
2.3.6.1.	Competitions .....	28
2.3.7.	Erdos .....	29
2.3.8.	Pyro: Python Robotics .....	30
2.4.	Summary .....	33
2.4.1.	Hardware Direction.....	33
2.4.2.	Software Simulation Direction.....	33
Chapter 3 - Investigation and Comparison of Approaches .....		35
3.1.	Introduction.....	35
3.2.	Hardware Solution .....	35
3.2.1.	RCX Physical Robot .....	36
3.2.1.1.	RCX Robot Sensory Layout .....	38
3.2.2.	Real time Embedded Software For RCX Robot Solution....	40
3.2.2.1.	Internal Memory Structure of RCX .....	40
3.2.2.2.	Development Work on LeJOS .....	41
3.2.2.2.1.	Outcome of development with LeJOS on RCX.....	42
3.2.2.3.	Development Work on NQC.....	43
3.2.2.3.1.	Outcome of NQC development for RCX Robot.....	43
3.3.	Software Simulation Approach.....	45
3.3.1.	Olivier Michel’s Khepera Simulator - KSim .....	45
3.3.2.	Wright State University (WSU) Java Khepera Simulator....	46
3.3.3.	Evorobot.....	46
3.3.4.	Yet Another Khepera Simulator – YAKS.....	47
3.3.5.	EasyBot .....	47
3.3.6.	Webots .....	48
3.3.7.	Erdos .....	49
Conclusions in Use of the Erdos .....		49
3.3.8.	Pyro .....	49
Conclusions in Use of Pyro.....		50
3.4.	Overall Conclusions .....	50
3.4.1.	Hardware Embedded Software Direction .....	50
3.4.2.	Software Simulation Direction.....	51

Chapter 4 - Software Design and Implementation.....	53
4.1. Introduction to the Software Design .....	53
4.2. Webots' Environment .....	53
4.2.1. Software Designing in Webots.....	53
4.2.2. Software Design for the Project .....	54
4.2.3. Development Work of Piglet Controller .....	56
4.2.3.1. Outcome from Modification of the Piglet Controller ...	58
4.2.4. Outcome of Webots .....	59
4.3. Project Work with Pyro.....	60
4.3.1. Pyro Outcome .....	68
4.4. Results of Implementation .....	68
Chapter 5 - Analysis and Results of Implementation .....	69
5.1. Webots .....	69
5.1.1. Analysis of Webots .....	69
5.1.2. Results and Observations with Webots.....	69
5.2. Pyro .....	70
5.2.1. Analysis of Pyro.....	70
5.2.2. Results with Pyro .....	71
5.3. Assessment Criteria.....	72
Chapter 6 - Conclusions and Recommendations .....	74
6.1. Conclusions .....	74
6.2. Recommendations .....	76
Appendix A - Hardware Code.....	77
A.1. LeJOS Code .....	77
A.1.1. Communication Testing with LeJOS .....	77
A.1.2. Mapping ability Testing of RCX using LeJOS .....	83
A.2. NQC Code.....	92
A.2.1. Navigation Testing with NQC.....	92
A.2.2. Map Storage Testing using NQC .....	96
Appendix B - Simulation Code.....	99
B.1. Olivier Michel's Khepera Simulator - KSim .....	99
B.2. Webots .....	100
B.2.1. Webots World File .....	103
B.3. Erdos .....	105



B.4. Pyro .....	105
Appendix C - Java.....	107
Java Background.....	107
Appendix D - Pyro Setup.....	108
Glossary .....	110
Poster Publications.....	112
Bibliography.....	113

## List of Figures

Figure 1 RCX module [26] .....	9
Figure 2 Rubik's Cube Solver [25] .....	9
Figure 3 iRobot's Roomba Red Vacuum image <a href="http://www.smarthome.com">www.smarthome.com</a> .....	10
Figure 4 iRobot Create Robot .....	11
Figure 5 RIS Programming Environment [26] .....	12
Figure 6 Erdos State Slate Roomba interface .....	16
Figure 7 Khepera Robot [51] .....	17
Figure 8 Khepera Robot Desk Set-up .....	18
Figure 9 Olivier Michel's Khepera Simulator .....	21
Figure 10 WSU Khepera Robot Simulator V7.2 .....	22
Figure 11 Evorobot GUI interface .....	23
Figure 12 YAKS Simulator [68] .....	25
Figure 13 EasyBot Simulator .....	26
Figure 14 Webots: World Environment .....	27
Figure 15 Webots Khepera Simulated Data .....	27
Figure 16 Alife charge station fully charged [72] .....	28
Figure 17 Alife charge station empty [72] .....	29
Figure 18 The Erdos project's pyRoomba output .....	30
Figure 19 Pyrobot .....	31
Figure 20 The Pyrobot Simulation Environment .....	32
Figure 21 Differential Gear Drive    Figure 22 Differential Gear Drive .....	37
Figure 23 Differential Gear Drive    Figure 24 Differential Gear Drive .....	37
Figure 25 Robot with Differential Drive used in this project .....	39
Figure 26 RCX Internal Memory Map .....	41
Figure 27 Webots arena .....	55
Figure 28 Webots piglet world data .....	55
Figure 29 Webots environment .....	57
Figure 30 Output of simulated camera .....	57
Figure 31 Camera false colour and internal data .....	57
Figure 32 Maze data .....	58
Figure 33 Camera information only .....	58
Figure 34 IR sensor collision data .....	58

Figure 35	Ground information .....	58
Figure 36	Pyrobot environment .....	61
Figure 37	UML Diagram of the simulator environment.....	62
Figure 38	Simulation environment with maze file loaded .....	63
Figure 39	Internal data of robot including the particle locations .....	65
Figure 40	Robot's internal data particles increasing weighting .....	66
Figure 41	UML of Robot Brain .....	67
Figure 42	TestRCXComm flowchart .....	77
Figure 43	SerialListenerTest Flowchart.....	78
Figure 44	Sender Flowchart .....	79
Figure 45	SendDistanceValues Flowchart .....	80
Figure 46	RunSendDisVal Flowchart .....	81
Figure 47	Receiver Flowchart.....	81
Figure 48	TestRCXComm4 Flowchart .....	82
Figure 49	Nothing Flowchart .....	83
Figure 50	BehaviorSendData Flowchart.....	84
Figure 51	LowLevelDrive Flowchart.....	85
Figure 52	LeftRightBumper Flowchart.....	86
Figure 53	FrontBumper Flowchart.....	87
Figure 54	Arbitrator Flowchart .....	88
Figure 55	MazeArea Flowchart A.....	89
Figure 56	MazeArea Flowchart B.....	90
Figure 57	AbstractSearchEngine Flowchart .....	91
Figure 58	motors_navigation Flowchart A.....	92
Figure 59	motors_navigation Flowchart B.....	93
Figure 60	motors_no_navigation Partial Flowchart .....	95
Figure 61	motors_navigation_single_array Partial Flowchart .....	97
Figure 62	Khepera Simulator Maze environment.....	99
Figure 63	User's info box .....	100
Figure 64	Flow Chart DrawSurrond .....	100
Figure 65	Piglet Controller UML part A .....	102
Figure 66	Piglet Controller UML part B.....	103
Figure 67	Mapped Environment .....	103
Figure 68	Output of Simulated Camera .....	103

Figure 69 False Colour Camera ..... 103  
Figure 70 Avoid Flowchart..... 106

## List of Tables

Table 1	NQC limitations constrained by RCX .....	15
Table 2	Khepera Robot Specifications [57] .....	18
Table 3	Simulators under Investigation .....	20
Table 4	Simulators Overview.....	34
Table 5	External Environmental Sensors of Roomba.....	36
Table 6	Event Record Bit Index.....	44
Table 7	Installation process of Simulators.....	52
Table 8	Assessment criteria of simulator's environment.....	73
Table 9	NQC Variable Storage .....	98

## Chapter 1 - Introduction

### 1.1. Background

The word “robot” first came into the popular domain with the first performance of Karel Čapek’s play *R.U.R.* (Rossum's Universal Robots) (1920) where he coined the word *robot* meaning "forced labour". This word was translated to “robot” when the play was translated into English for the 1923 production, and thus this new concept spawned our imagination. Stories from Isaac Asimov’s collection “I, Robot” (1950) presented the world with the three “laws of robots”:

1. A robot may not injure a human being, or through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

As robots have moved from being largely of science fiction to that of science fact, our perception has also changed to a willingness to accept robots among us, at a certain level. Researchers are now developing robots to mimic human reactions for applications as receptionists in buildings. There is something unnerving about staring a “being” in the face and wondering whether or not it is real.

We are accustomed to the use of robots in the industrial automation, underwater missions and space applications, and we are beginning to become more accepting of robots in the area of household help for example: vacuum cleaners, lawn mowers and pool cleaners from companies such as iRobot. The designs of such robots bear no resemblance to their former occupant of jobs such as, the pool cleaner, the grounds keeper or the home help. Educational and children’s toys are also an emerging market where robots are readily accepted and are, in some cases, highly sought after.

A second area where huge development is currently under way is in the search and rescue and the defence contractors, as is visible by the competitions

run by DARPA, 1<sup>st</sup> and 2<sup>nd</sup> Grand Challenge [1] [2], which took place over a desert track navigated by autonomous robots and 3<sup>rd</sup> Grand Challenge for autonomous robots in the Urban Challenge Moves to the City [3].

### **Robotic Operation**

If robots are to operate in real environments they require an ability to interrogate their surrounding world and make reasonable judgments based on the information they receive. The reasoning judgments are as a result of processing by Artificial Intelligence (AI). For the robot to employ this AI it needs to remember or record information of current and past information received, thus construct a map of the area it has already traversed in its environment. As the robot moves around its imperfect interpretation of the environment, the robot will experience errors due to, for example wheel slippage, leading to flawed kinematics data and possible collision with other stationary or moving obstacles. The AI control has to allow for these errors to occur and to update its internal data with its current position and orientation.

There are a large number of mobile robot platforms currently available on the market; these platforms range in price from the very expensive €100,000s to much more reasonably priced platforms of approx. €200. There are also custom designs developed to fit the needs of particular projects which also vary hugely in cost and complexity, such as those developed for the DARPA grand challenge desert races or less expensive designs developed for specific tasks in education.

The complex nature involved in developing in-house systems along with the pitfalls in the integration of different hardware, requires a large investment in time and effort, which can mitigate against their use in student projects [4]. The development time can be at the expense of the overall goal, which is the investigation of developing AI controllers for robotic systems.

If the decision is made to purchase a robotic platform, budgetary constraints can play a large part in minimising choice. There is also a cost where a department changes hardware platforms and these new platforms present different architecture from the previous system, presents a new set of requirements in the development of controllers.

## **1.2. Project Goals**

The main goal of this research is to identify a cost effective platform and an approach to the design and development of low cost autonomous robotic systems. A cost effective platform in the context of this research refers to one that reduces both development time and costs and provides better value for money in terms of features.

The use of custom built hardware platforms will not be considered for investigation as these require organisations to have in-house expertise in design, development and debugging and as such may preclude the organisations from developing control algorithms on these platforms.

The identification of such a platform would facilitate the research and design of AI based mobile robotic systems. This should facilitate the development of autonomous systems by small research teams with limited funding, skills in AI and in hardware development. This can assist research teams previously not investigating AI in mobile robotics, to expand into the area of developing AI systems. The inclusion of these smaller and more diverse organisations, together with the current community effort into AI for robotics research, could help to extend current boundaries in the development of minimal autonomous robotic systems.

The approach taken in this research began with a literature review to identify appropriate solutions in terms of hardware and software systems. This was followed with an evaluation of each system, using such criteria as suitability for the development of mobile autonomous robotic systems, ease of development, availability and cost. This was followed by bench testing of hardware and software systems where possible. A comparison between systems was undertaken and a selection of the optimum systems was made.

## **1.3. Overview**

This research involved a desk study of both hardware and software solutions for the provision of a test bed environment. It contrasted the two different approaches to the development of minimal autonomous robotic systems and selected the more appropriate one.



- Outlines the different software simulators available for investigation.
- Descriptions are given of each of the specific simulators included in this chapter, Table 4 identifying the specific hardware platforms covered by each of the simulators.

**Chapter 3 – Investigation and Comparison of Approaches:** This chapter reviews and compares in more detail the hardware and software systems identified in the literature review:

- Hardware with Embedded solution:
  - The competing hardware platforms are assessed and a choice is made to select the most appropriate one.
  - The different embedded solutions presented in chapter 2 for the chosen hardware platform are evaluated and critiqued with reference to the project goals. Work carried out in the evaluation process is outlined, and UML diagrams are presented in Appendix A - Hardware Code.
- Software platforms solution:
  - The software platforms presented in chapter 2 are also evaluated, and a comparison is made between each to select the more relevant one. Code used in this evaluation process is presented in Appendix B - Simulation Code.
- Project Direction:
  - The competing hardware and the software solutions are presented at the end of the chapter and a selection is made of the most appropriate solution to achieve the goals of the project.

**Chapter 4 – Software Design and Implementation:** At the end of chapter 3 two different simulators were identified as appropriate candidates to be further investigated. This chapter carries out this investigation, using much more detailed designs and implementations. The simulators are presented here, with their

controller's outputs from the simulated robots. UML diagrams for the implemented controllers are included in the chapter under the simulator headings.

**Chapter 5 – Analysis and Results:** Chapter 5 presents the results for the investigation into both the Webots and Pyro simulators. It discusses the results obtained from each simulator's differing architecture approaches, Webots being a specific solution and Pyro being a generic solution to the development of robotic controllers. The chapter also highlights the assessment criteria applied to these two simulators in Table 8.

**Chapter 6 – Conclusions and Recommendations:** This chapter highlights the shortcomings of the hardware approach to the goals of the project. It lists the headings that were used in the critique section of the software simulators in establishing the best candidates as a solution to the goals of this research.

In particular it highlights the difference between the Pyro and Webots approaches in developing robotic controllers, and the advantage of generic approach over the specific approach in development of controller for different platforms.

The thesis then presents recommendations which include further work to extend Pyro to improve the visual sensor simulation compatibility with other operating systems.

**Appendix A – Hardware Code:** This appendix includes flowcharts for the code for LeJOS and NQC, used in the evaluation of the RCX hardware platform.

**Appendix B – Simulator's Simulation Code:** This appendix includes flowcharts and some UML diagrams of the code developed in the comparison stage of chapter 3 in this research. <sup>1</sup>

---

<sup>1</sup> The use of general descriptive names, trade names, trademarks, etc., in this thesis, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

## Chapter 2 - Literature Review

### 2.1. Introduction

The literature review focuses on the two different approaches taken in the evaluation of a minimal autonomous robotic system. It investigates previous work done with respect to a hardware implementation and also to a software simulation approach. This work is presented in two sections:

a. Hardware Solutions:

The Hardware investigation identified different low cost hardware platforms that are readily available. This investigation includes an examination of the various embedded software and the implementation of approaches used on each platform.

b. Software Solutions:

The Software investigation identifies the various robotic simulation environments which are available for research. These are compared and the simulators are summarised in tabular form.

### 2.2. Hardware Solutions with Embedded Software

This section is divided into parts. The hardware platforms are presented firstly, and then the embedded software for each platform is then presented.

#### 2.2.1. Hardware Platforms

There are many different platforms utilised in research departments, but many of these require a large capital investment for the platform and its associated software. The pioneer robot which has featured as part of many research laboratories [5] robotic platforms costs €5,995 (2009) [6], for the robot base. Other platforms include the more powerful PowerBot base at a cost of €19,995 (2009) [6], the AmigoBot base at €2,245 (2009) [7] and the Khepera base robot at ~€1,600 (2009) [8]. These high costs can present an obstacle to the development of AI in budgetary constrained departments, especially where multiple systems are required.

The main focus of the hardware research was to identify affordable solutions for an in-depth investigation. Once one has eliminated custom built hardware platforms from the goals of the project, there are only two platforms

remaining which fit the selection criteria and cited frequently in the literature, and these are described in the following sections:

- RCX – LEGO Mindstorms RCX module
- Roomba – iRobot's home vacuum Robot

#### **2.2.1.1. RCX**

The LEGO® Mindstorms RCX module from the LEGO's Robotics Invention System shown in Figure 1 is both low cost and readily available. It has already been used in courses run in the Electronics department in the GMIT. This module has been presented by numerous different researchers [9-13] for diverse areas of investigation, from undergraduate research groups, programmes to encourage interest in robotics, and GMIT's programme in teaching robotics to primary and second level students.

Although it has featured extensively in the hobby, school and undergraduate levels, it also has potential at postgraduate level as a suitable platform for further studies as documented in research papers from: the University of Kent [14], University of Wollongong [15], Institute of Advanced Computer and Research [16], Computer Science Department [17], Institute of Operating Systems and Networks [18] and University of Twente [19].

The Mindstorms' RCX is based on a design developed at the Massachusetts Institute of Technology (MIT), adapted from designs such as, the "LEGO Robot Design Competition 1992" [20], "The Art of LEGO Design" [21] and "Introduction to Engineering Design ELEC 201 Course Notes" [22]. As the RCX module has proved itself useful in other postgraduate research project areas, it formed part of the investigation.

#### **Physical Hardware of RCX Module**

The RCX module consists of a Hitachi micro-controller with 3 inputs, 3 motor outputs and an Infra Red (IR) serial communication port [23]. The RCX is packaged in what looks like an oversized LEGO Block that can be included in any Lego design, but what makes this different is that it can be programmed to perform any task "only limited by the creator's imagination" (and code memory size). There are designs using RCX ranging from rudimentary image scanners

[24] to designs using multiple RCXs to solve the very frustrating Rubik's Cube problem [25] shown in Figure 2.



Figure 1 RCX module [26]

The RCX module is shown in Figure 1 with its accompanying sensors and motors.



Figure 2 Rubik's Cube Solver [25]

### 2.2.1.2. iRobot's Roomba and Create Robots

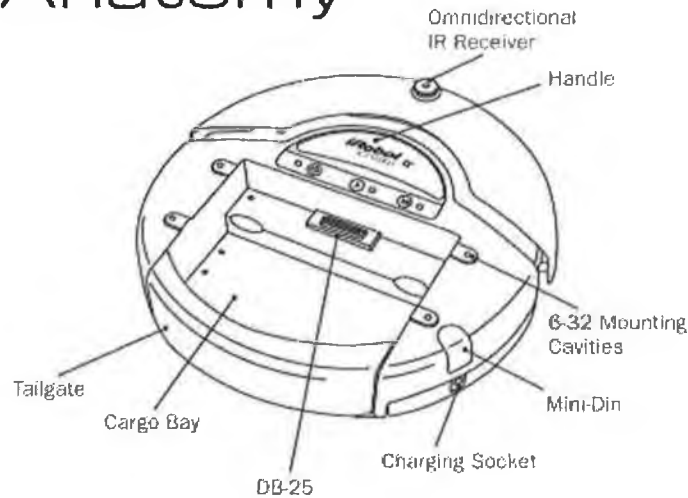
The iRobot Roomba® Figure 3 Vacuuming Robot launched in 2002 [27] was designed to be an affordable home cleaning robot. Initially, iRobot only produced versions of this robot that were complete vacuum cleaner units. However, after external user community groups grew up and began reverse engineering of the design [28], the company provided the open source interface called “Roomba Open Interface” (ROI) released in 2006 [29] to allow users to interface to Roomba. With the release of this ROI [30] and its predecessor the “Serial Command Interface” [31] in 2005, the user groups now had access to the protocols that control a robot over a serial interface, enabling experimentation on unaltered versions of the Roomba [32, 28, 33-36].



**Figure 3 iRobot's Roomba Red Vacuum image [www.smarthome.com](http://www.smarthome.com)**

Figure 3 illustrates the iRobot's Roomba Red with charger and remote control. This robot can be combined with a charging home base that allows the robot to recharge itself when it detects that its batteries require charge. It will return to the charging home base if it can still locate the base. The Company iRobot also released a version of Roomba called “Create” [29, 37, 38]. The Create robot Figure 4 is marketed at the research and hobby market. It's design differs from the Roomba in that it has removed the vacuum and dust collection parts of the design and replaced them with a cargo bay. This cargo bay allows for additional hardware to be attached to the robot, extending its possible uses.

# Anatomy



**Figure 4 iRobot Create Robot**

The iRobot and the Create with their existing hardware configurations (for external sensors see section 3.2 Table 5) do not provide the user with onboard processing, but provide the user with a serial interface to interact with the robot's inputs and outputs. The serial interface is defined in the iRobot's Roomba [30] and for the iRobot's Create [39]. These designs require a base station with a wired / wireless link (the robot requires a wireless dongle for this mode) to control the robot's operation, or the use of additional hardware added on top of the robot to provide onboard processing power. Researchers such as Dickenson [32], Tribelhorn [35] and Conbere [40] have placed a laptop on the Roomba to provide the robot with processing, Kurt [33] used a wireless router running embedded Linux and Matarić [34] an embedded Linux module, to allow them to construct their mobile robots with it own controlling algorithm. Others such as Dodds [41] have operated their Roomba with wireless dongles, providing the processing and robot control from a base computer.

## 2.2.2. Embedded Software for Hardware Platforms

This section details the investigation of the different approaches used in the development of controllers for the different hardware available under investigation. The literature presented a selction of different languages used to

control the RCX module. It also investigates the development of a controller for the iRobot's Roomba and Create modules.

### 2.2.2.1. RCX Hardware Platform

The investigation into the development of a controller for the RCX platform presented three different programming environments. These environments are:

LEGO Robotics Invention System Environment

LeJOS - Java for LEGO Mindstorms

NQC - Not Quite C

#### LEGO® Robotics Invention System Environment

The LEGO® Robotics Invention System (RIS) is sold as part of the development environment provided by LEGO with their RCX modules. It provides the user with a block diagram environment for the development of controllers for their robots, see Figure 5.



Figure 5 RIS Programming Environment [26]

The user selects which block they require and then connects them together to form a flowchart for the programme. The RCX module is programmable from



the computer via the connected Robotics Invention System's IR tower. Testing with this configuration was carried out during this project to establish how the real world interactions with RCX operating in this programming environment. It was also very useful in identifying appropriate motor speeds for the particular gearing ratio used.

The RIS environment also proved useful in assessing the properties of the communication link between the RCX module and the IR tower in investigate the requirements for valid connection. It showed that unless the RCX was positioned in direct line of sight, reliable communication could not be achieved. This would negate the possibility of continuous communication between the computer acting as a base station and the RCX Robot.

As a programming environment the Robotics Invention System provides its users with a very intuitive approach to create controllers owing to the block diagram development nature. It has however large limitations when it comes to the development of more complex tasks because of the limited variety of the blocks available. As a result the use of this language was not continued in this project.

#### **2.2.2.1.1. LeJOS Java for LEGO® Mindstorms**

LeJOS [42] is a programming language for the RCX module. It is a replacement firmware for the LEGO Mindstorms RCX brick. LeJOS is an Open Source Java based operating system of the LEGO'S Mindstorms RCX module. It runs on the RCX internal processor the Hitachi H8300 [43]. The software environment consists of three different parts:

1. A Virtual machine (VM) for the execution of Java bytecode.
2. An API for RCX programming on top of this VM.
3. Additional software tools.

LeJOS has been identified and used by researchers [9, 12] and authors have written books on the subject [44-46]. It has also featured in a project in GMIT to solve a Rubik cube using a RCX modules [47]. LeJOS is a tiny Java Virtual Machine (JVM) for the RCX. It enables the RCX to be programmed using the Java language. This allows for much more complex programmes to be written

for the RCX module that can be produced with the RCX's original Robotics Invention System software.

The programmer writes code in the Java language and then compiles this into Java byte code, which is downloaded using the IR tower to the RCX module. The RCX module required to be preloaded with LeJOS, which is a miniature operating system. LeJOS interprets the byte code control programmes stored on the RCX and carries out their instructions. The LeJOS provides a reduced instruction set to account for the limited nature of the RCX environment as Java is an extensive language, with the facility to be utilised in diverse programming areas.

#### **2.2.2.1.2. Not Quite C (NQC)**

NQC is another programming language for the RCX. It was presented by Dave Baum [48] as an alternative programming language to the RIS language provided by the LEGO kit. The NQC programming language is specifically designed for the LEGO robots. It was written by Dave Baum to allow the programmer much greater freedom from the RIS environment. Each of the robots in the LEGO Mindstorms series has its own bytecode interpreter. The NQC compiler translates the source code programme written by the user into LEGO's bytecodes, which are then executed on the robot. Research teams [49, 50, 12] have utilised this programming language in course work. Mario Ferrari's book [45] "Building Robots with LEGO MINDSTORMS The ULTIMATE Tool for MINDSTORMS Maniacs" referring to the use of NQC with the RCX module is another example.

NQC programming language was designed for the RCX module. It is quite similar to the C programming language, but it is more restrictive due to the restrictions placed on it by the RCX module. Programming in this language like LeJOS also enables much more flexibility over the block style in Robotics Invention System. It also does not require a Virtual Machine to run on the RCX module as the user's programme is translated into bytecode, which runs on the firmware inside the RCX module. The programming on the RCX hardware presents certain limitations of the code as shown in the Table 1, this is due to the resources available on the RCX module.

**Table 1 NQC limitations constrained by RCX**

Tasks	Subroutine
Up to 10 tasks allowed in programme	Up to 8 Subroutines per programme
One Task has to have name "main" which runs all other must be started by running task with "start"	Cannot use arguments with subroutines
	Cannot call another subroutine from within subroutine
All tasks run simultaneously	Must use semaphore if call subroutine from different tasks
Tasks restart from beginning	

Tasks are equivalent to threads in C and C++; they run at the same time, once a start command is issued with their name and are terminated with a stop command. Subroutines are equivalent to C and C++ functions; they are small pieces of code that can be repeatedly called from different places in the programme.

#### **2.2.2.2. iRobot's Roomba and Create Hardware Platform**

As previously mentioned different researchers have approached the task of controlling the iRobot's hardware in different ways shown in Figure 3 and Figure 4. Some have provided the robot with an on-board brain by mounting additional hardware on top of the existing robot structure, while others have left the robot unchanged only including a dongle to remotely control the robot from a base computer. Zachary Dodds and Ben Tribelhorn developed the Erdos project for controlling the iRobot's hardware using such a dongle.

##### **2.2.2.2.1. The Erdos Project**

The Erdos [41] project was designed around an unmodified Roomba utilising the API realised by iRobot in their SCI [31] and their further release of the Open Interfaces for Roomba [30] and Create [39]. This API enables the Roomba to act as serial device, which can respond to requests with actions or data from another serially equipped device. The physical Roomba has been equipped with wireless capabilities using an additional hardware of an external Bluetooth dongle. The "state slate" software interface shown in Figure 6 enables the user to interact with all the features available on the Roomba robot. It presents the

sensory data available of the robot, and also enables the user to use the virtual onboard buttons. The project further includes a simulation interface allowing for the development and testing of control programmes for robot prior to real operation. The Erdos simulator is explained in more detail in section 2.3.7 below.



ERDOS: State State Slate					
Updating	Update All	Frame 1	Frame 2	Frame 3	STOP
Left Bumper	Right Bumper	Left Cliff	LFnt Cliff	RFnt Cliff	Right Cliff
Right Wall	Virtual Wall	Left Wheel Drop	Caster Wheel Drop	Right Wheel Drop	OC/DD ok
Vacuum	Side Brush	Main Brush	Drive mm/s mm	0	0
Spot LED	Clean LED	Power LED	Status LED	DirtDetect LED	Max LED
Spot Button	Clean Button	Power Button	Remote Command	0	Max Button
Play Song	Song Set	Set & Play	00 16 64 16 67 16 72 32		
0					
Reset Odometry	x in mm	y in mm	theta in deg	raw distance	raw angle
	Unknown	Unknown	Unknown	Unknown	Unknown
					
Charging State	Voltage mV	Current mA	Charge mAh	Capacity mAh	Temperature C
Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Off Mode	Passive Mode	Safe Mode	Full Mode	RoTooth Power	RTS Power
Quit	COM Port		Server Port		User

Figure 6 Erdos State Slate Roomba interface

In Figure 6 the current location of the robot is represented by the circular dot, shown in the white panel which is a representation of its physical environment.

### 2.2.3. The Khepera Robot

This Khepera robot [51] manufactured by the K-Team [www.K-Team.com](http://www.K-Team.com), although excluded for the selected hardware solutions owing to its cost, nevertheless it is included here because it has formed the core subject of many software robot simulation environments.

This robot has featured in numerous publications [52-55] and competitions like Kheperasot [56]. Prof. Jean-Daniel Nicoud team based at the Microcomputing Laboratory (LAMI) of the Swiss Federal Institute of Technology of Lausanne (EPFL), designed the Khepera robot. The initial design, developed in 1991, was to have been a small inexpensive robot. One of the design requirements was for a robot of 1 cubic inch, which could facilitate investigation into mobile robots. Robots available at the time tended to be large and expensive. Another advantage of having a small robot is the dynamics change with respect to the physical nature, a large robot of diameter 1m moving at 1m/s colliding with a wall can have catastrophic results as opposed to a small robot of diameter of 1cm moving at 1cm/s, which generally results in negligible damage shown in Figure 7.



Figure 7 Khepera Robot [51]

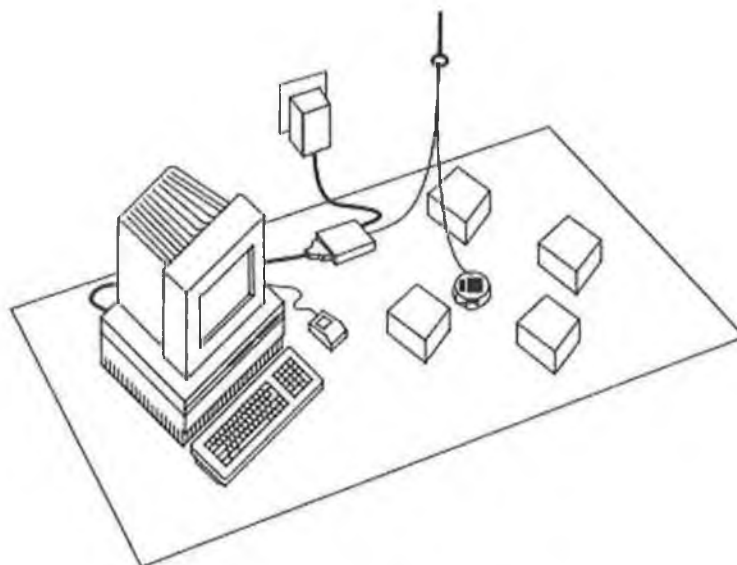
The Khepera hardware includes 8 infra-red (IR) proximity sensors that provide both measurement of the absolute ambient IR light, and by reflection of the emitted IR light, an estimation of the relative position of an object from the robot. The platform also included both wheels with wheel encoder information.

Table 2 lists the hardware specifications of the initial released Khepera robot. Later models improved on these specifications.

The initial implementation of the Khepera robot can be seen in Figure 7. The design provided the research community with a compact robot device that would allow investigation into how real robots interact with the real environment. A possible experimental setup is shown in Figure 8. The Khepera allowed for expansion modules to be installed on top of the Khepera base robot to further extend the possible task usage of the robot.

**Table 2 Khepera Robot Specifications [57]**

Elements	Technical Information
Processor	Motorola 68331, 16MHz
RAM	256 Kbytes
EPROM	128 or 256 Kbytes Reprogrammable
Motion	2 DC brushed servo motors with incremental encoders (12 pulses per mm)
Speed Max:	60 cm/s, Min: 2 cm/s
Sensors	8 Infra-red proximity and ambient light sensors with up to 50mm range
Power	Power Adapter Rechargeable OR NiCd Batteries OR Power Adapter.
Autonomy	30 minutes moving continuously
Communication	Standard Serial Port, up to 38kbps
Size	Diameter: 55 mm Height: 30 mm
Weight	Approx 70 g



**Figure 8 Khepera Robot Desk Set-up**

Figure 8 [51] illustrates an experimental set-up of the Khepera robot using it on a bench connected to a computer. The diagram shows a Khepera robot tethered to a computer using a hanging cable. The robot is surrounded with 4 obstacles. This layout where the robot can be operated on a bench has greater advantages, as a large space does not have to be provided for experimentation purposes where the user is modelling a maze environment. This contrasts with larger robots that may require outdoor space or a large indoor arena. The design was further commercialised by the K-Team [58] which provides a number of different variations based on this initial design. The Khepera robot and its variants are used widely in research centres. There were 58 centres listed in over 15 countries using the Khepera robot [59]. The Robot has been utilised in many different research areas from robotic soccer [56] to swarm mapping [60]. K-Team provides extension modules for the Khepera robot to enhance the functionality of the robot. These include: a gripper arm, radio communication turret, linear vision turret, video turret, a matrix view turret and additional processor. These enable the Khepera to perform much more completed tasks, but add to the expense of the robot's kit.

The ease of use of this robot coupled with the availability of its many add on modules have been important factors in its use for many research programmes. Thanks to this widespread use it has spawned a large body of simulation software developed both in-house by the team who designed it and independently by other researchers to meet their specific simulation needs.

### **2.3. Software Simulation Approach**

Eight different simulators currently utilised by the research community were identified during the literature review, as possible candidates. The Table 3 lists the simulators under investigation. A detailed description is given of each in the sections below. Table 4 at the end of the chapter includes a complete overview of the different simulators. These simulation environments were critiqued under 12 different headings ranging from real world representation to operating simulation support by the simulator. These are detailed in chapter 6.

**Table 3 Simulators under Investigation**

<b>Simulators</b>
Khepera Simulator
Wright State University (WSU) Java Khepera Simulator
Evorobot
Yet Another Khepera Simulator – YAKS
EasyBot
Webots
Erdos
Pyro: Python Robotics

### **2.3.1. Khepera Simulator**

The Khepera Simulator [61] is also referred to as the Olivier Michel's 2D Khepera Simulator [62]. Olivier Michel developed the simulator at the University of Nice Sophia-Antipolis. Since then he is based at the LAMI in Switzerland, which is where the Khepera was initially developed. This was presented to the research community as the first simulator designed entirely around the newly released Khepera robot (at the time of simulator release). Olivier Michel has since launched a company called Cyberbotics. This simulator is designed to run on a UNIX operating system. It allows the user to write control algorithms in C or C++ language. These algorithms are run by the simulator, in simulation of a Khepera robot operating in a 2D world. There is also a feature to transfer this controller algorithm to a real robot, to assess the controller's responses in the real environment.

The user selects a map in which the robot is required to operate, which is displayed by the simulator GUI. The user can use one of the predefined maps provided by the simulator or they may choose to create a new map or edit an existing one. The simulator allows the user to add bricks to form walls that will be detectable by the robot after the "scan" button is pressed. They can also choose the start location and orientation of the robot.



The GUI also presents to the user the simulated internal information of the robot, Figure 9 that is used to aid in the diagnostics of the controller throughout simulation. The GUI presents information on the distance sensors, light sensors and the speed and direction of both motors.

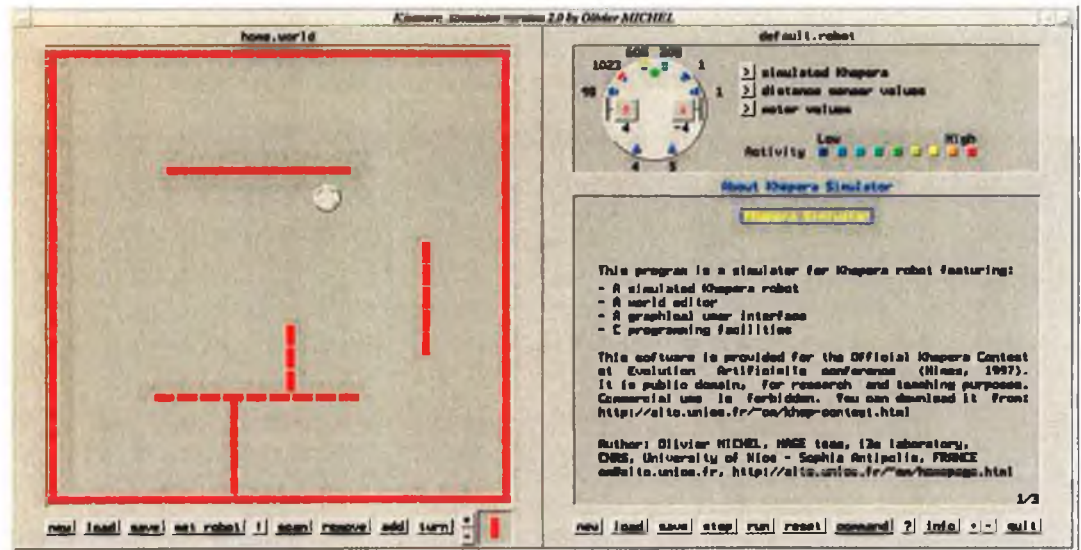
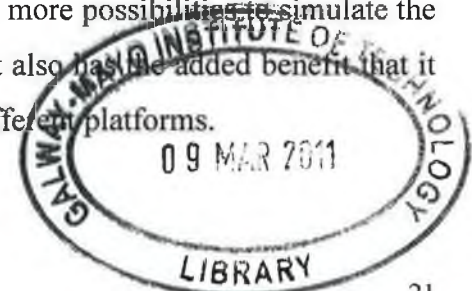


Figure 9 Olivier Michel's Khepera Simulator

### 2.3.2. Wright State University (WSU) Java Khepera Simulator

WSU Java Khepera Simulator [63] shown in Figure 10, was developed at the Wright State University College of Engineering And Computer Science's Evolvable Hardware Research Lab. The "KSim" was developed to provide a platform for the "coarse-grained" development of controllers for the Khepera Robot prior to testing on an actual Khepera robot. The simulator is written in Java (Appendix C - Java), which enables it to run on both Windows as well as Linux platforms. The KSim simulates the 8 IR sensors, which convey both proximity and light intensity, wheel encoder information and a gripper object sensor. This gripper sensor indicates the presence of an object between the grippers.

This simulator is based largely around the original Khepera Simulator. This provides the same functionality but provide more possibilities to simulate the addition module of the Khepera robot gripper. It also has the added benefit that it is written in Java which allows it to be run on different platforms.



The WSU simulator defines walls to be static objects thus the robot cannot have any effect on these and collision with them can result in the robot getting stuck. It also allows for simulation of dynamic objects such as “caps” (simulated representation of a light weight object) and “balls” (representation of a ball object) which will be effected by the robot if it collides with them. The simulator allows the robot to use its gripper to collect the caps or the balls if the controller equips the robot with sufficient behaviour to interact with them. As can be seen in Figure 10 the simulator allows the user to view the values that are measured by the robot’s sensors during simulation, this can enable the user to debug their controller should an unintended action be carried out by their controller. Users write their code in Java. A controller template is provided by the simulator installation, which indicates the important methods that are required to be implemented to allow the simulator to run the controller.

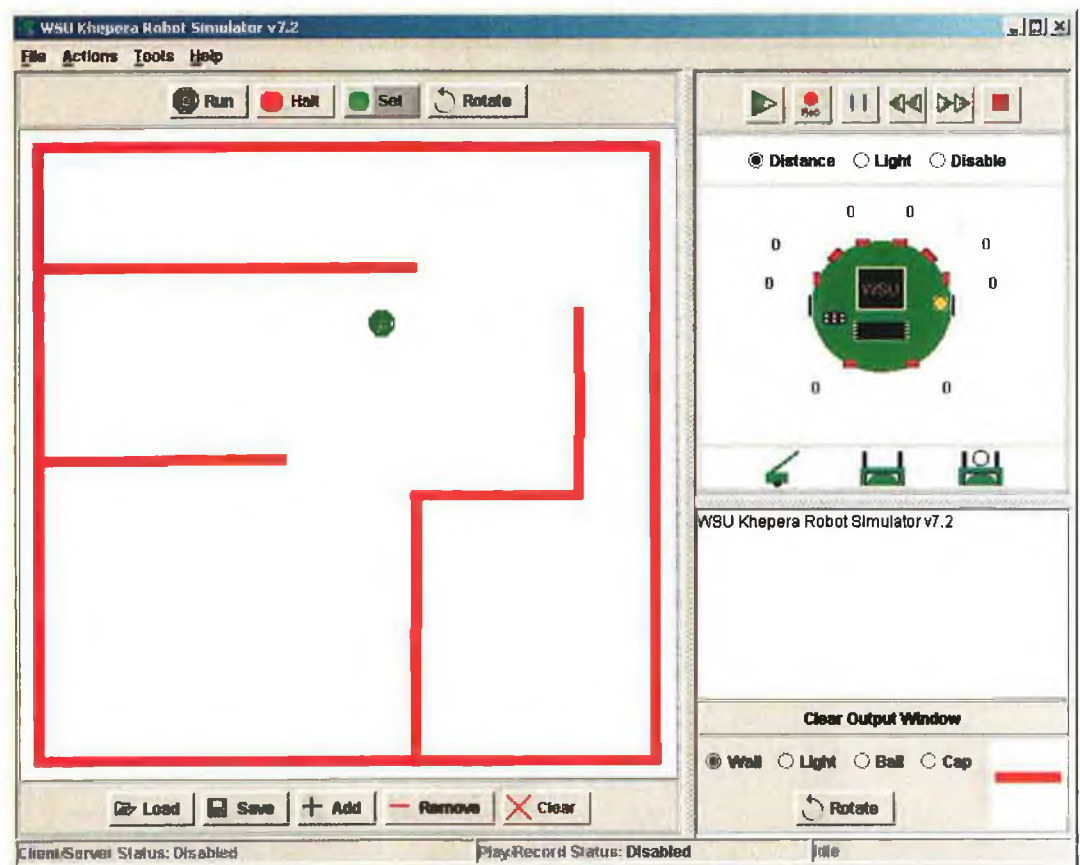


Figure 10 WSU Khepera Robot Simulator V7.2

In Figure 10 the robots sensory inputs are displayed in the top right of the image. The position of the gripper is displayed directly below this image, where

the presence of a “cap” or “ball” is indicated. On the left the robot’s maze is displayed and the robot position is indicated by the circle.

### 2.3.3. Evorobot

The Evorobot simulator [64] also referenced in [62] is a software programme for investigating evolutionary robotics. It runs on a Windows operating system. Stefano Nolfi [64] developed Evorobot. The author has “copyleft”ed this software under the GNU General Public License to make it available for use. “Copyleft is a general method for making a program or other work free of charge, and requiring all modified and extended versions of the program to be free as well” [65]. The simulator shown in Figure 11, allows the user to run evolutionary experiments in simulation or on a real robot. Stefano Nolfi worked on the “kepsim” robotic simulator [66] as mentioned in YAKS in section 2.3.4 and incorporated the functionality, into this simulator, of using the real world data to construct “look-up” tables to enable faster simulation of robots.

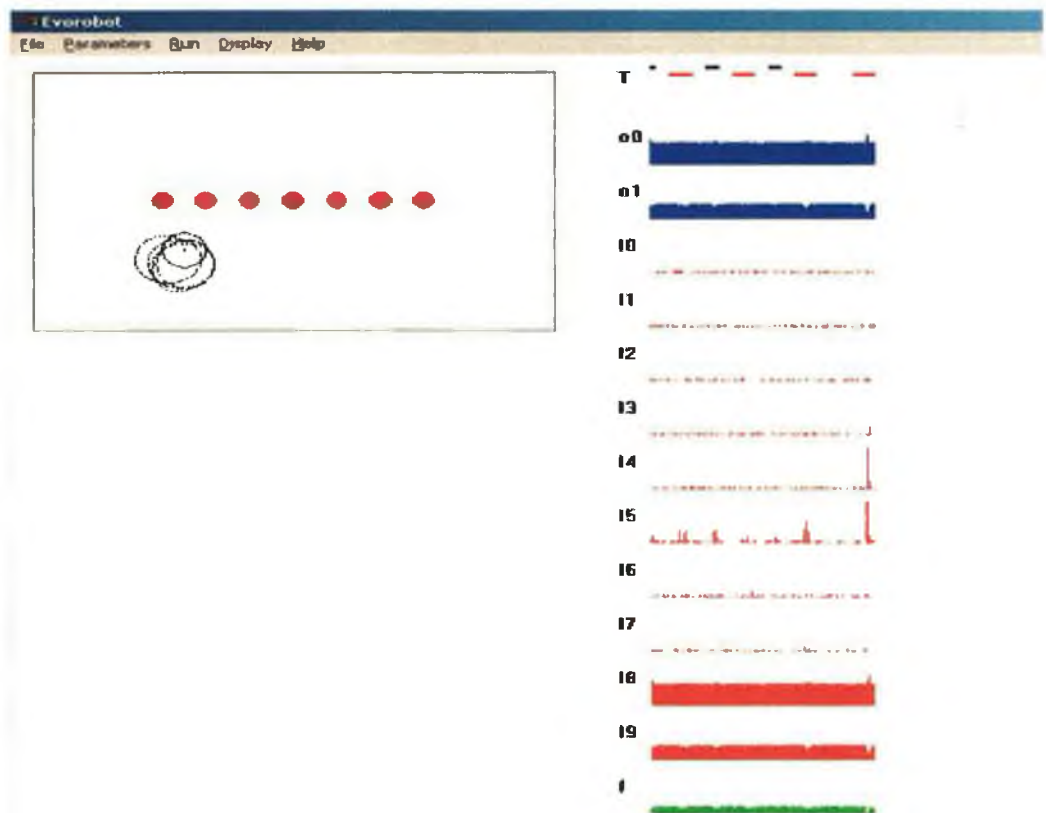


Figure 11 Evorobot GUI interface

In Figure 11 the robot's simulated environment is displayed the top left of the window; the robot is shown here with the path it has traversed in the current run. The environment consists of a enclosed space which also contains seven cylindrical objects. On the right is a graph of the information produced by the robot from its sensors. The top and bottom two wide bands are information from the current and past state of the motors (the simulated robot records it past motor state onboard). The eight bands in between indicate the information acquired from the eight distance measurement sensors. The other information is to display information about the current run's fitness function. A run ends when the robot collides with a wall or after a predefined time.

#### **2.3.4. Yet Another Khepera Simulator – YAKS**

The “Yet Another Khepera Simulator” [67] (YAKS) simulator was designed as a robot simulator to allow research into Genetic Algorithms (GA) in combination with Neural Networks. The simulator designers take the approach that if a robot's sensors are mathematically modelled during simulation then, although this is an accurate representation of a real robot, it produces a very slow simulation. The solution that they implemented was borrowed from the designer of “kpsim”, using pre-recorded sensor values for real world objects measured by the physical robot and applying these to a “lookup” table. During simulation of the robot the “simulated” sensor values are calculated by accessing appropriate values from this lookup table. This approach greatly reduces processing time in simulation, which in GA research is very important where a “run” can be in the order of 1000 generations, with each typically having 100 individuals, each of which is required to be simulated a given number of times, and from a number of different locations. The simulator is shown in Figure 12.

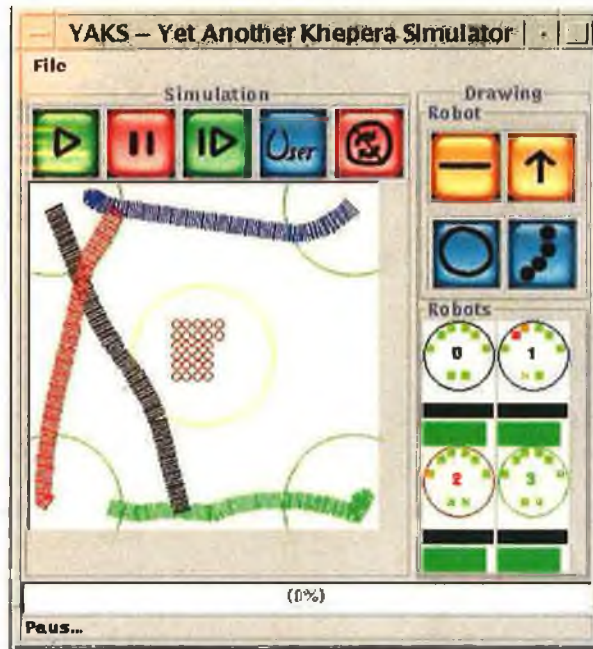


Figure 12 YAKS Simulator [68]

In Figure 12 different information for each run is displayed, this is a multi robot simulation environment so each robot is displayed in the bottom right along with the state of their sensors. Their different paths are displayed in the environment on the left.

### 2.3.5. EasyBot

EasyBot [69] is a robot simulator shown in Figure 13, which allows for the design and testing of control algorithms for mobile robots.

EasyBot is an extension for LightVision3D [70] to allow for the manipulation of mobile robots. LightVision3D by Oliver Michel [70] (not “Olivier Michel” of the Khepera Simulator) is a 3D viewer and modelling software. EasyBot is designed to integrate with LightVision3D to allow the environment to model robot objects. Robots are defined in the environment as objects containing sensor objects. All group-objects in the environment containing sensor objects are controlled by user selected Dynamic Linked Libraries (DLLs), which specify the actions that will be carried out by each particular robot.

EasyBot is referred to as the “Universal Robot Simulator” [71]. The EasyBot extension for LightVision3D allows the designer to create any type of robot and/or environment that they require. The designer is then required to produce a description file to map EasyBot’s sensor data onto their robot’s data

and the robot's controller onto EasyBot's position and orientation requirements. The interface file for the Khepera robot is included for EasyBot's mapping. EasyBot requires the user's controller to carry out collision detection, as this is currently not included in the simulation engine.

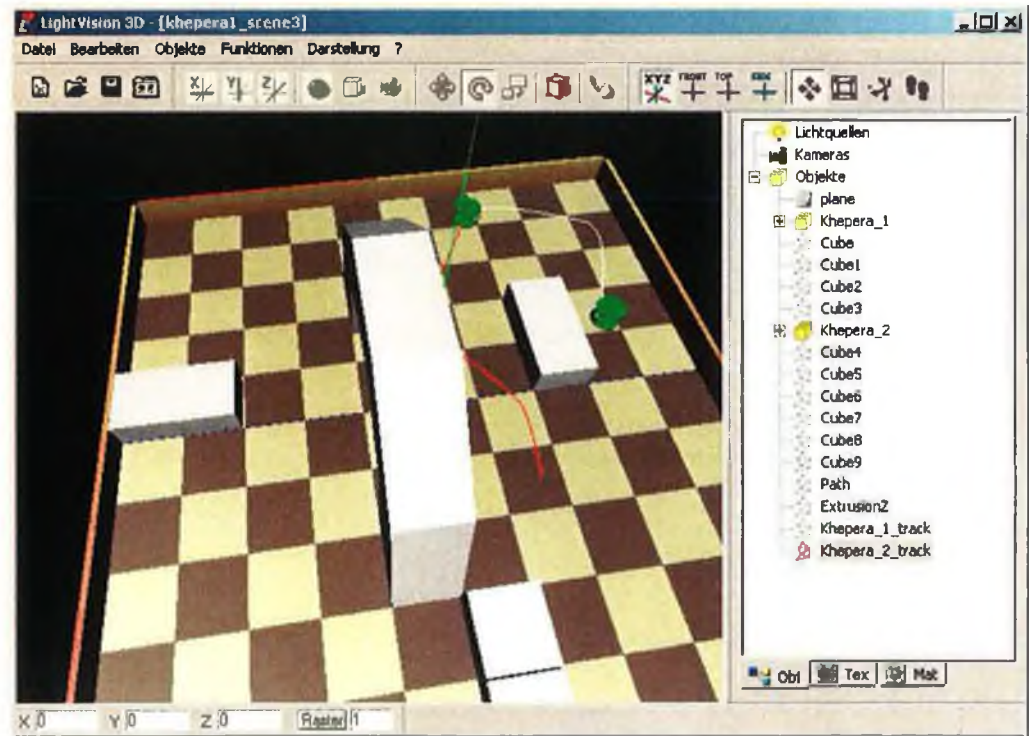


Figure 13 EasyBot Simulator

### 2.3.6. Webots

Webots [72], [73-75] is a proprietary software simulation environment by Cyberbotics, <http://www.cyberbotics.com/>, which allows for the design and simulation of different robotic layouts with a range of different sensors. It is the commercialisation of Olivier Michel's free "2D Khepera Simulator" for the research design market. It allows the user to create a physical robot in simulation defining its size and shape. Also accommodated are the robots wheel size and rotation angle forming the robot's propulsion system. The user then defines the robot's complement of sensors their location and type. The simulator environment allows for ultrasonic and IR sensors and camera image simulation. The user then constructs an environment in which they require their robot to operate. They specify the shape, size and contents of the required robot's environment.

Figure 14 illustrates an example of a maze world environment in which a simulated Khepera robot is required to operate.



Figure 14 Webots: World Environment

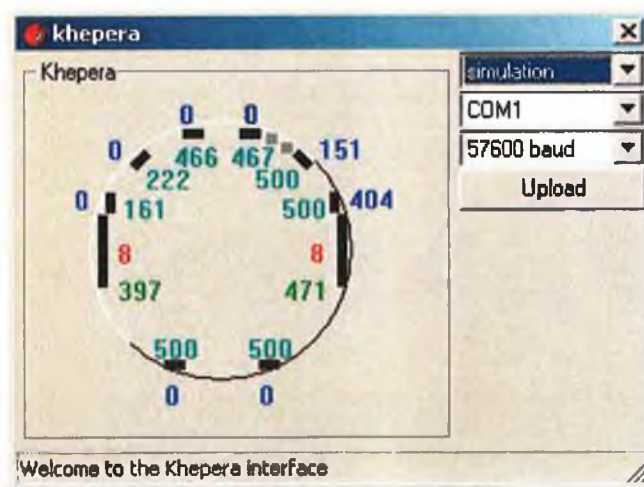


Figure 15 Webots Khepera Simulated Data

Figure 15 is the simulated internal information of the Khepera robot that can prove useful in debugging erroneous controller responses to sensory data.

### 2.3.6.1. Competitions

The company Cyberbotics that produce Webots has run a number of different competitions with different goals to encourage the community to further develop programming techniques using Webots.

One competition run by Cyberbotics was called Alife [72], this is where the community was required to enter a controller for a robot, which would compete against another entry for survival in a maze environment [76]. The objective of the competition is for two robots to be initialised in a maze environment so that they seek out energy feeder stations with green front side, the colour indicating that they have energy available for charging a robot see Figure 16.



**Figure 16 Alife charge station fully charged [72]**

These charge stations are positioned through out the maze and they may be difficult to see. When a robot visits a charge station this charge station will be unavailable for some time, their front colour will change to red see Figure 17. The robot is then required to locate a new charge station elsewhere in the maze. If a robot battery power runs out before it finds another energised charge station it will die. The remaining robot will be declared the winner. On visiting a feeder the robots batteries will be recharged by a certain amount, depending on the length of time the charge station indicates available power (green colour). The maze included a number of different charging stations but the robot could only charge if it reached the charge station first after the station turned green and stayed until it was no longer green.

The object of the entrant's controller was to locate an energised charge station, plan a path to this charge station while beating competitors to that charge



station. It then continued to locate the next station and continue this process until a winner outlasts the other competitor.



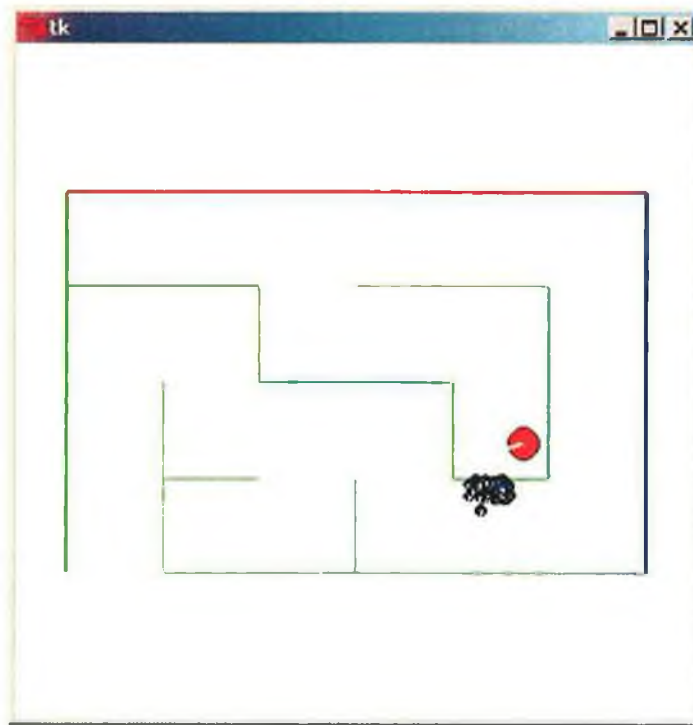
**Figure 17 Alife charge station empty [72]**

Another competition run by Webots involved simulated humanoid robots involved in a “judo like” competition. The objective this time was to push or knock an opponent out of a defined ring area.

### **2.3.7. Erdos**

The Erdos project [41] as previously referred to in section 2.2.2.2.1 was developed by Zachary Dodds and Ben Tribelhorn is based around the commercially launched robot designed to clean floors by the name of Roomba produced by the company “iRobot” described in section 2.2.1.2.

The Erdos simulator shown in Figure 18 is a 2D simulation for the Roomba robot. Erdos can operate on a real Roomba provided one is connected over the serial port or on a simulated version of one. Erdos incorporates a particle filtering algorithm to provide more accurate modelling of the odometry information from the robot. The simulator is written in the Python language.



**Figure 18** The Erdos project's pyRoomba output

Figure 18 shows the simulated iRobot in a maze environment. The robot is indicated by the circle and the clump of smaller circles indicated a possible location identified by the control algorithm. The algorithm uses the information from the wheel distance covered and the information from forward sensor.

### 2.3.8. Pyro: Python Robotics

Pyro stands for PYthon for Robotics [77]. Pyro is a different approach to the design and development of intelligent robots [78]. It is created using the Python programming language. The Pyro approach is in the use of abstraction when designing a controller for a robot design. This abstraction will remove the requirements for the designer to understand the specifics of the robot that the controller will be required to run on. This abstraction applies to both the propulsion and to the sensory array with which the robot is equipped. It allows the designer to ignore the hardware specifics and concentrate on the operating algorithm. This logic means that the user should not have to worry about how large the robot is or how it moves. This has enabled authors such as [79, 40, 4, 12, 80] to take advantage in the courses to enable students to investigate much more detailed behaviours in a shorter time period than experienced previously. Users can normalise and work with “robot units” that define the world in relation to the

robots size. The logic also applies to the sensors of the robot where the user may not know the layout of the sensors but can query the robot's "front", "left", "right" or "back" sensors. The "driver interface" which the controlling programme interfaces to, returns the appropriate values again which can be in relation to robot units. The user constructs a controller to drive a robot, which can be seamlessly applied to a real or simulation version of that robot. Figure 19 the user can select the required world map, the robot type, the sensing devices that are required and the control algorithm the "brain".

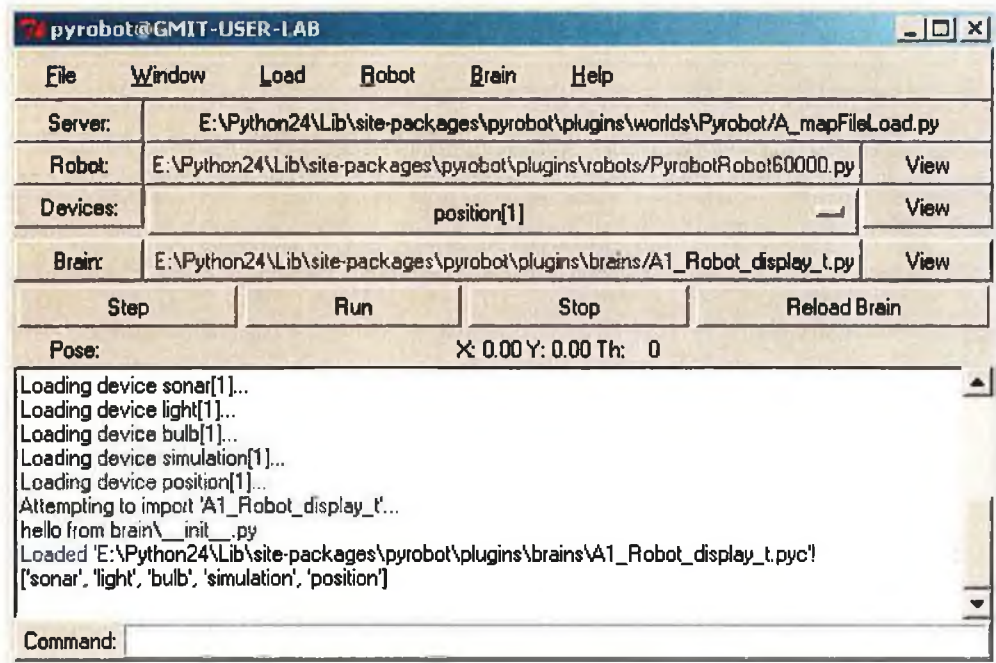
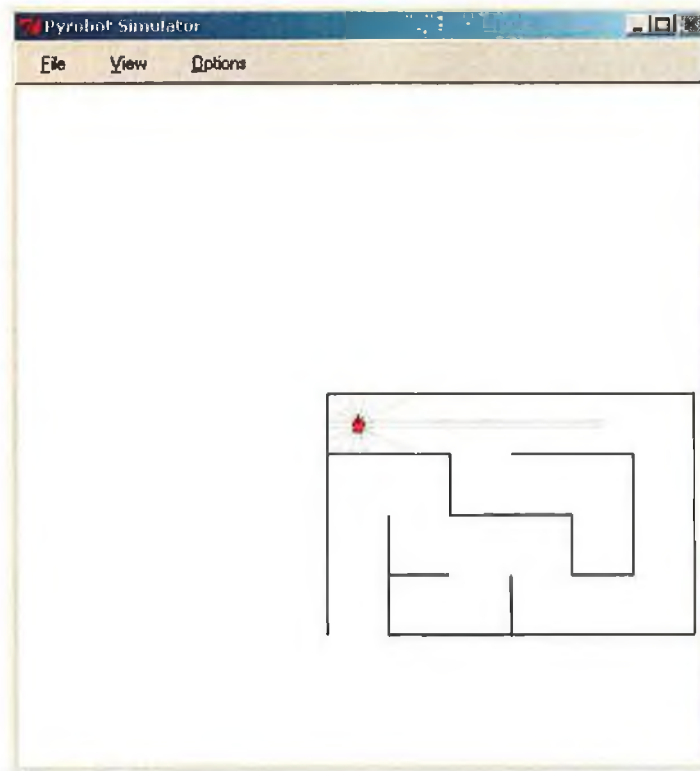


Figure 19 Pyrobot

In Figure 19 the "Server" button select the server which communicates with the environment. This server allows for a real world robot to be controlled or a simulated robot to be controlled. The robot type is then selected with "robot" button. The devices of interest available for that robot can be selected by "Devices". The control algorithm is then selected by "Brain". Figure 20 displays the environment in which the robot will operate. It has been defined in the world file. All locations of obstacles location of the robot are defined there.



**Figure 20 The Pyrobot Simulation Environment**

In Figure 20 the robot is shown in its maze environment the "rays" represent all of the distance sensors available on the robot. Their line length represents the relative value returned to the robot indication proximity to an object.

## **2.4. Summary**

### **2.4.1. Hardware Direction**

The hardware direction identified from approx fifty candidates, two possible low cost platforms used by researchers in AI mobile robotic development. The iRobot's hardware provides a suitable base to use as a platform for mounting further hardware to enable development of a mobile robot but the base itself does not provide adequate sensory data as it only provides two bump sensors [30]. The inclusion in the design of the "cliff top" sensors does allow for more design ideas where a real world is much more 3D than 2D. As a result of the lack of sensors available on the base, this platform was not investigated further.

The development of the RCX hardware platform will be presented in the next chapter section 3.2.1. Of the three embedded software approaches described above, only the LeJOS and NQC will be compared and contrasted in-depth, the LEGO RIS environment being dismissed due to its inadequacies, as presented previously in chapter.

### **2.4.2. Software Simulation Direction**

The eight different simulators under investigation in this project described in section 2.3 above are listed in Table 4. This table lists each simulator's attributes as related to programming language, Operating System (OS) and the physical robot simulated.

Comparisons are also made in the next chapter of these eight simulators to assess appropriateness to the direction of the project. This helped the selection of simulators for further investigation in this project.

Table 4 Simulators Overview

Simulator	Program language	Unix compatible	Windows compatible	Open Source	Simulated Robots	Commercial	Source code available	Requires other software
Khepera Simulator	C & C++	Yes	No	Yes	Khepera	No	Yes	C/C++ compiler
WSU Java Khepera simulator	Java	Yes	Yes	Yes	Khepera	No	Yes	JDK
Evorobot	Specific	No	Yes	Yes	Khepera	No	Yes	No
YAKS	C & C++	Yes	Yes	Yes	Khepera	No	Yes	C/C++ compiler
EasyBot	C & C++	No	Yes	No	Many Different types	Yes	No	C/C++ Compiler Light-Vision3D
Webots	C & C++ Java	Yes	Yes	No	Many Different types	Yes	No	C/C++ Compiler JDK
ErDOS	Python	Yes	Yes	Yes	Roomba Create	No	Yes	Python
Pyro	Python	Yes	Yes	Yes	Many Different types	No	Yes	Python

In Table 4 each of the simulators under investigation is listed along with their attributes. The restrictive nature of their simulation is identified. Their computer platform on which they can run is also included, as are the availability of their underlying source code. It also lists whether they are proprietary or open source.

## **Chapter 3 - Investigation and Comparison of Approaches**

### **3.1. Introduction**

The hardware and software systems identified for further investigation in the literature review will now be reviewed and compared in more detail. This chapter is organised as follows:

- Hardware Solution
  - Description of the physical hardware platforms under investigation.
  - Investigation into different Embedded Hardware Solutions for the RCX platform.
  
- Software Simulation Solution
  - Comparison of the different simulation environments identified in the Literature review.
  
- Comparing the “Hardware with Embedded Software” solutions with the “Software Simulation” solutions.
  - Presentation of the criteria used in the selection of one solution over the other.

### **3.2. Hardware Solution**

Of the two candidates identified for possible investigation in this research it was identified that the RCX showed the most promise for further investigation. The RCX allowed the developer to upload the controller software to the memory and then run it. The Roomba by iRobot did present as a very sturdy base but it would require additional hardware and onboard processing at a minimum to equip it with the requirements for this research. The Roomba base unit (see Table 5 below) only provides two collision sensors and an offset wall sensor and as such was considered inadequate in its current state.

**Table 5 External Environmental Sensors of Roomba**

Sensors	Number	Type	Location
Bumper	2	Press buttons in bumper	Front
Wheel Drop	3	Detect the wheel have dropped down	One each of the drive wheels and on caster wheel
Cliff sensor	4	Detect the sensor has gone over a cliff	Front of robot 2 to the right and 2 on the left
Wall Sensor	1	IR transmitter and detector	Positioned on the front right
IR Sensor	1	IR sensor	Front top of robot

On choosing the RCX module, the investigation into this platform focused on the two parts to the development on this platform:

- Physical Hardware
  - Propulsion design of robot
  - Sensory layout in the detection of obstacles
- Embedded Software investigated
  - LeJOS
  - NQC

The physical hardware is described below; the embedded software investigation is described in the section 3.2.2:

### 3.2.1. RCX Physical Robot

As the RCX module features in the project, a description of its physical robot is now given. The Robot is required to traverse a maze avoiding obstacles and recording maze locations. To produce a working robot it is necessary to incorporate the RCX in a physical embodiment to provide locomotion, sensing etc. There are a plethora of implementations of the RCX physical robot presented by different authors [81], mainly due to its suitability for undergraduate and postgraduate projects. Because of this it would be almost impossible to conduct an exhaustive search and report on all of them. Instead, a representative sample has been chosen.



In a robot where localisation is directly related to robot orientation and wheel movement it is vitally important that movement and direction control are rigidly controlled. Thus, in this project the design used for the drive system went through a number of design iterations, each time improving the system performance of the mobile robot. Initially the design involved connecting the wheels of the robot directly to both drive motor shafts to allow the robot to move. The motors were driven at a very slow rate of rotation to facilitate controlled forward, reverse and turning motion. This design was further improved by adding in some gearing between the motor shafts and the wheels. These first propulsion designs produced very poor accuracy in motion, which was characterised by a lot of wheel slippage resulting in skewed motion.



**Figure 21 Differential Gear Drive**



**Figure 22 Differential Gear Drive**



**Figure 23 Differential Gear Drive**



**Figure 24 Differential Gear Drive**

The propulsion system was improved by using a design for differential gearing presented by [82], which allowed for a more accurate way of driving the robot using the 2 motors to control forward, backwards or rotation. This enabled the robot to rotate by spinning on its own axis.

Figure 21-Figure 23 illustrate the Differential gearing system adapted by Mark Overmars [82] from other designers. This gearing system operates as follows: if only one motor is rotating it drives both wheels forward, or both wheels backward, while the other motor is responsible for steering the robot; turning it left or right. If on the other hand both motors operate at the same time one of the wheels will remain motionless while the other wheel rotates at twice-normal speed enabling fast turning.

This gearing arrangement was slightly changed for the design used in this project to allow the robot to occupy only the space between the two wheels (see Figure 25). This allows the robot to spin on its own axis with no unnecessary protrusions beyond the wheel arc.

The ability of the robot to drive in a straight line was improved in this arrangement over that, where each wheel was driven by a separate motor. The single motor had the disadvantage that to drive a robot forward each motor was required to be energised for a specific time. On completion of this time the power would be removed from the motors. The motors would settle by rotating clockwise or anticlockwise to the nearest permanent magnet contained within each motor assembly, resulting in the trajectory of the robot being upset slightly. This may be exacerbated if the motors are rotating at higher speeds (when over-run occurs) or with a lower gear ratio to the wheels. With the new gearing arrangement this settling would result in both wheels travelling the same amount in a forward direction as only one motor is driving both wheels. As a consequence of these slight inaccuracies, the robot travels slightly greater or less than the distance required by the programme, but produces no variation in rotation of the robot from forward motion.

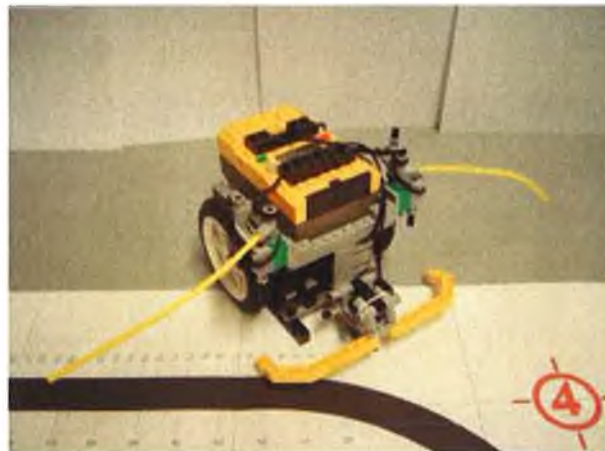
Unfortunately, none of the hardware designs include any feedback from wheel encoder information, as the RCX interface only provides 3 inputs and all three are used in ascertaining information about its environment.

#### **3.2.1.1. RCX Robot Sensory Layout**

The RCX module presents the designer with the ability to read 3 input parameters and control 3 output parameters. Each of the outputs can drive a single motor. The inputs can be connected to analogue inputs or switches. The RCX

programme is informed by the user as to the nature of the input, be it an analogue or switch. The RCX robot base proceeded through a number of different sensory layouts. As a robot is required to detect its environment and a mobile has a greater requirement to detect its environment to provide for collision free locomotion. The layout proceeded through a number of design refinements. The initial design use 3 bumpers positioned to the left, front and right of the robot. This allowed the robot to detect obstacles on the left, front and right hand sides. The design proved to be unworkable due to the fact that the robot would not be able to track along beside an obstacle due to the fact that when it detected an object to the side it would have to change course or the contact between the side bumper and the object would upset the trajectory of the robot.

This design was further refined to a system where the three sensors consisted of one bumper sensor to the front and the left and right sensor resembled “whiskers” that protruded out each side, see Figure 25 below. This layout meant that an object to the side could be detected but would not hinder the forward motion of the robot until it collided with the front bumper. This design would enable the robot to follow walls by detecting them with its “whiskers”.



**Figure 25 Robot with Differential Drive used in this project**

Figure 25 shows the robot configuration used in this research. The front bumper extends beyond the robot on either side to detect if an obstacle gets too close to the robot to unduly interfere with the current trajectory. Both right and left “whiskers” extend out much further than the bumper so that objects can be detected, allowing the robot to trace the periphery of the object if required without

causing a “bump” to the system. Such a bump will change direction of the robot, and will generate a discrepancy between expected and actual path.

As stated earlier, the RCX set-up did not provide for a wheel encoder to be incorporated into the design as only 3 inputs parameters were available but already used for the detection of surrounding obstacles. Distance calculations are carried out by recording the time and direction that the wheels are turning for a given motor speed. This method is referred to as “dead reckoning” but without the benefit of wheel encoder information. The wheel encoder information would offset the effect that changes in battery voltage have on the distance calculations.

### **3.2.2. Real time Embedded Software For RCX Robot Solution**

Development work on the LeJOS and NQC based embedded solutions for the RCX platform identified in the literature review (section 2.2.2.1) is presented here as part of the investigation into controller designs for the platform. This section contains:

- Internal Memory structure of RCX
- Development work in LeJOS language
  - Outcomes of LeJOS
- Development work in NQC language
  - Outcomes from NQC language
- Presents Conclusions from comparison of these two approaches at the end of the chapter in section 3.4.1.

#### **3.2.2.1. Internal Memory Structure of RCX**

Figure 26 outlines the memory structure available in the RCX module. The 16 Kb of ROM contains a driver for the RCX module that remains after power loss. It handles low level commands to the internal hardware of the RCX and allows for the downloading of the firmware. The firmware resides in the first 16Kb of the RAM. This is a bytecode interpreter, which acts like the operating system for the RCX. It interprets the user’s programmes which are written in bytecode and runs them by calling the appropriate opcode in the ROM. The user’s programmes reside in the next 6Kb of space in the RAM. The final 10Kb of

memory space is used by the firmware as interpretive and execution space. This is where local and global variables will reside during operation.

10 Kb Interpreting byte code & Execution space	6 Kb RAM User programmes
16 Kb RAM Firmware	
16 Kb ROM	

**Figure 26 RCX Internal Memory Map**

### 3.2.2.2. Development Work on LeJOS

The original RCX RIS programming environment was limited by the restricted nature of the available programming blocks. As a result a new programming language was chosen to drive the RCX module. This LeJOS requires a different firmware to be installed on the RCX module. This firmware is a JVM for the RCX environment. The new memory layout is shown in Figure 26. The developed code is then downloaded into the user programme space onboard the RCX.

During the development phase on the RCX module using LeJOS programming environment testing was carried out on the module running the LeJOS Virtual Machine. Physical measurements were carried out to determine the time it took for the robot to drive forward 1 metre and rotate through 360°. These were calculated for a RCX module reporting a voltage of 8.1V;

- Time to drive 1 metre is 5 seconds
- Time for a full rotation is 3.21 seconds

These values were then used in the code to perform the calculations for the “dead reckoning” to determine the robot’s current angle and location, relative to its start location. Code developed for the testing process is presented in Appendix A - A.1.

The LeJOS’s inbuilt process management system operates on the basis of a weighted process list. The process of higher weighting always takes precedence over a process of lower weighting. If the higher process requests operation it

returns a true from a call to its “takeControl()” function. The designer using this structure to construct their code, can form a list of different tasks that require operation and assign a different weighting to them by adding them to the list with the first being of lowest weighting.

Work commenced with this environment, writing code to test out the operation of the RCX running LeJOS. The programmes were designed to utilise the different inputs and outputs configurations on the RCX to investigate the advantages of LeJOS over the Robotics Invention System programming language approach.

The software design then continued into the application of the RCX running LeJOS for this project. This involved implementing a routine which detected surrounding objects of the robot, see Appendix A - Figure 52 and Figure 53. Work was conducted on the development of a mapping system to record detected objects around the robot, see Appendix A - Figure 55 and Figure 56, which is described later.

#### **3.2.2.2.1. Outcome of development with LeJOS on RCX**

The development on the RCX robot using the LeJOS environmental language proved that the robot could detect objects using its “whiskers” on the left and right and react to a front bumper collision with the appropriate stop motors running to minimise navigational errors from the collisions. It could calculate distance travelled and direction using dead reckoning technique of recording the length of time the motors were operational at a particular speed and direction.

The LeJOS environment could not however provide adequate storage space to map the environment travelled by the robot. The robot system would be required to store a large amount of data on the RCX to provide the system with a viable map. This is because the map would store the location data of all objects encountered around the maze as well as the path taken by the system. This map would be required to provide the robot with knowledge of its surroundings to enable it to perform educated decisions as to the direction to turn next, to best reduce the uncertainty about the current robot’s world view. The approach used by the Java software to store its information also adds to the overhead in the storage of the vital mapping information. As a result of LeJOS’ approach

requiring more memory than that available in the limited environment of RCX, it precluded continuation with LeJOS as a viable solution for the project.

### 3.2.2.3. Development Work on NQC

The NQC environment operates using the original RIS firmware on the RCX module. This firmware is required to be re-downloaded each time the battery power is disconnected from the RCX as it resides in the onboard volatile RAM.

Here again experimental calculations were required to establish the time taken for the robot to travel the distance of 1 metre and rotate 360°;

- Time to travel 1 body length = 100 counts of 10ms duration = 1s
- Time to rotate 360° = 70 counts of 10ms duration = 0.7s

Programmes were written in NQC to assess the ability of the RCX robot to map out a maze environment. Testing of the detection by the robot's "whiskers" of objects on both the left and right was carried out. This involved assigning events to each of the sensors on the robot and detecting when an event occurred. When one of the "whiskers" made contact with, or lost contact with, an object on the left or right this generated an event. This event was then captured and the time of the event was recorded, see Appendix A - Figure 60 and Figure 61. Code presented for this testing process is in Appendix A - A.2. The NQC in the RCX environment supports 32 global variable locations and 16 local variable locations. The results of investigating the storage restrictions are presented in Appendix A.2.2- Figure 60 and Figure 61. The results from this test indicate that you can effectively increase the number of storage locations using arrays but it ultimately cannot provide sufficient storage locations to store a map.

#### 3.2.2.3.1. Outcome of NQC development for RCX Robot

In this development using NQC the RCX robot was equipped with the ability to detect the presence of objects on the left and/or right and also to detect when that object was no longer present. This ability would enable the robot to record obstacles and their continued presence, over the number of unit robot body lengths. This ability could be used to construct a map using the robot's world view in robot sized units. As was identified while programming the RCX using

the LeJOS environment, the internal memory is of limited size. In the NQC approach an attempt to compensate for the lack of internal memory, lower level variables were chosen to minimise the required storage of data, this involved bit masking of a memory location. This would enable a variable to utilise only the required number of bits to record the variable's possible different states. An example using the 16 bit integer "event\_occured" to store 8 different variables referring to the state of object detection by the sensory inputs is shown in Table 6:

**Table 6 Event Record Bit Index**

Bit index	Bit name
1	FRONT_BUMPER_OBJ
2	FRONT_BUMPER_OBJ_GONE
4	LEFT_EAR_OBJ
8	LEFT_EAR_OBJ_GONE
16	RIGHT_EAR_OBJ
32	RIGHT_EAR_OBJ_GONE
256	RIGHT_EAR_PRESSED
512	LEFT_EAR_PRESSED

The storage location "event\_occured" enables the controller to record the occurrence of each of the different events independently. This approach allows the controller on servicing the highest event to clear its occurrence from the variable and then service the next highest event. Lower priority events are always recorded in the variable and are not affected by updating other events.

Using the approach of minimising the bits needed by each variable proved to be an appropriate way of minimising memory usage, by individual variables. It did not however achieve sufficient memory savings to provide enough memory to store the location of objects detected by the robot and produce a map of any usable size.

It was decided that, due to the limited nature of the RCX memory storage, as well as it only having 3 inputs for the detection of the environment, that the investigation into its use as the basis for the project could not yield adequate results.



### 3.3. Software Simulation Approach

This software simulation approach required the selection of a suitable simulation environment. The eight different simulators identified in Chapter 2 - Literature review were:

- Olivier Michel's Khepera Simulator
- Wright State University (WSU) Java Khepera Simulator
- Evorobot
- Yet Another Khepera Simulator – YAKS
- EasyBot
- Webots
- Erdos
- Pyro

Each simulator is evaluated to identify its appropriateness and suitability for the project. A report was given in the literature review on the operation of each simulation environment. Simulators presenting as good candidates for the project direction will be highlighted here and further developed in chapter 4. A conclusion is given at the end of the chapter in relation to the simulator choice made.

#### 3.3.1. Olivier Michel's Khepera Simulator - KSim

The Khepera Simulator which was the first simulator produced for the Khepera Robot was investigated. As described in 2.3.1 this simulator became the foundation for subsequent authors in the design of later simulators for the Khepera Robot. Code developed during the investigation of this simulator is described in Appendix B - B.1.

It was concluded that the Khepera Simulator environment provided a good introduction to evaluating the simulation principles used when dealing with the Khepera robot. Tasks available include: accessing data representing the simulated IR values for the robot, performing decisions making based on these values and varying the speed and direction of the motors. These gave a good impression of the versatility of the robot but it identified limitations in the simulation

environment when it came to adjusting the maze. The environment required a complete recompile of all files when the controller was changed. Updating of the maze layout also proved similarly cumbersome. This would preclude the investigation into a dynamic environment where structures in the maze were moved around while the robot is in operation.

### **3.3.2. Wright State University (WSU) Java Khepera Simulator**

This simulator was investigated to identify if it suited further developmental work for this project. The simulator presented a nice interface and allowed the development of robot controllers in multiple different Operating Systems (OS) due to the simulator environment being in the Java language which operates similarly across different platforms. The simulator was designed around the original Khepera Simulator by Olivier Michel but did provide the user with some advantages over the original. The user no longer needed to recompile all of source code each time a change was made and with the added benefit that Java software did not require a specific OS. The disadvantages of this environment are that it only allows for objects resembling short walls to be added to the robot's environment, the user cannot interact with the robot during operation it does not allow for the robot to be kidnapped<sup>2</sup> during a "run". As referred to in the simulators documentation it only allows for coarse grained simulation<sup>3</sup>. Owing to these limitations it was deemed not appropriate to continue developmental work on this simulator.

### **3.3.3. Evorobot**

The Evorobot simulator is designed to run evolutionary robotics experiments. This is where a control strategy is developed from multiple "runs" of competing controllers and where the controller that best approaching the target is selected to seed the next generation of controllers. As this simulation software is only suited to a Genetic Algorithms (GA) approach it was rejected as an appropriate simulator to this project, as other simulators investigated also allow for GA investigation as well as other approaches.

---

<sup>2</sup> See Kidnapped in Glossary

<sup>3</sup> See Coarse Grained Simulation in Glossary

#### **3.3.4. Yet Another Khepera Simulator – YAKS**

The documentation [67] for this simulator presented a programme that warranted investigation. This investigation was to identify whether the platform was appropriate to continue research. The K-Team referenced this as a viable simulator [62]. During the investigation into this simulator, the installation process required the simulator source code to be compiled. This is common on freeware code but in this instance the compilation process presented errors which could not be overcome. These errors may have been due to the use, by the simulator of outdated library files. Because of the difficulty in resolving these issues it was decided that further work on this programme would not resolve these problems in the short term, and as a result this simulation environment was not continued with.

#### **3.3.5. EasyBot**

The EasyBot simulator presented the user with a nice interface enabling the design of a self contained environment in a 3D modelling language. The disadvantage with this environment is that the user interface of LightVision3D (LV3D) which EasyBot is an extension of, is provided only in the German language, as are the help files for LV3D. The EasyBot's extension to LV3D does not perform any collision detection which would prevent the robot going through walls or other objects, which the developer have expressed is required in future work [69]. As a result of the lack of collision detection provided by the simulation engine (which appears standard in most other simulators) this operation requires the user's controller to carry out collision detection. This results in the controller being burdened with the added complication which would not exist if the controller is run on a real robot. This provides the user the dilemma of not being able to design a controller as would be envisaged for a real robot. These disadvantages resulted in the simulator not being continued with as both presented real obstacles to the developmental work in this project.

### 3.3.6. Webots

In the Alife “Webots” competition run by Cyberbotics Ltd. the makers of Webots, Petr Stepan achieved the winning entry with his Piglet controller. This competition as previously outlined in Chapter 2.3.6.1, required the entrant’s controller to locate “feeding stations” around a maze environment. These feed stations charge the simulated internal batteries of the robot, while the feed station was green, see Figure 16. After a feed station had exhausted its power it changed colour to red, see Figure 17. The purpose of the competition was to develop a strategy to locate green charge stations and reach one of these charge stations before the other robot, depriving the competing robot of power from that charge station thereby outlasting them. A depleted charge station requires time to “recharge” itself, requiring the robot to locate the next station.

Petr Stepan’s code constructed a map of the surrounding obstacles and “feeder stations”. It then planned a path to these feed stations. It also carried out exploration of the incomplete areas of its internal map. The exploration would cease when it identified that its map represented an enclosed environment, one that is surrounding by boundary walls on all sides with no possible escape. The code would identify the status of each visible feed station by its colour, identifying whether charging was possible from that station.

Petr Stepan’s code provided a very interesting approach to navigation using a simulated camera-equipped Khepera robot. This code provided a promising basis for this research project. Petr Stepan’s original code is released under the GPL licence [83].

Code investigated during the investigation is presented in Appendix B - B.2.

It was concluded that the Webots simulation environment was identified as one that showed great promise for further investigation in this research. The competition entry by Petr Stepan identified very novel ideas in its winning entry in the Alife competition. Due to these two promising facets of Webots and Piglet controller this simulator was identified as one that warranted more in-depth investigation in this research.

### 3.3.7. Erdos

The authors of Erdos present their code to the world in somewhat of a partially finished state. The code presents to the user a number of different files one of which refers to a simulation mode. This simulation's functionality had not been fully implemented as the authors provide a more complete implementation in their code designed for connection to a real Roomba robot. Owing to the incomplete nature of the simulation environment, work was carried out to incorporate into the simulator the updated functionality available for controlling the full functioning robot driver. The Erdos platform is written in the Python programming language.

Code developed for the investigation is presented in Appendix B - B.3

### Conclusions in Use of the Erdos

Erdos simulator implemented some search algorithms for the Roomba robot. Roomba performed object detection with collision sensors and cliff edge detectors. These sensors are only placed on the front half of the robot, resulting in no rear sensors. Due to the limited nature of Roomba, this presented limitations on Erdos environment, and as a result it was not continued with.

### 3.3.8. Pyro

Investigation into this Pyro environment commenced with the development of a simple controller. This controller was designed to generate a reactive robot and enables the examination of the abstraction behaviour of the Pyro environment. This system abstraction enables designers to focus on the algorithm nature of the controller and allow the system to provide the relevant information of the specific hardware currently being controlled. If, for example the Pyro controller was asked to supply the information about the nearest object. If the hardware has just a few or even many forward sensors the user's controller algorithm may not need to know how many were available, it may just require the distance between the front of the robot and an object. The system processes the request for "front" sensors and provides the requested distance information.

This indicates that the standard approach to the design of controllers for robots is not required for Pyro because the designer does not require the

knowledge of the physical robot. This abstraction allows for the use of relative terms like “front”, “side” etc to refer to the sensor positions. This abstraction makes Pyro a very powerful tool. The Pyro simulator is programmed in the Python programming language.

Code developed for this investigation is presented in Appendix B - B.4.

### **Conclusions in Use of Pyro**

The Pyro simulation interface was identified as one that showed great promise, as the interface presented the user with a selection of different real or simulated robots. It provided for the development of controllers that could be utilised to control different robots, by the use of abstraction. The interface enabled the controller to request the “Front” sensors and the specific robot selected front sensors were returned, whether this was one sensor or many sensors.

## **3.4. Overall Conclusions**

### **3.4.1. Hardware Embedded Software Direction**

The investigation into the embedded software side of the project in section (3.2.2), identified that both the LeJOS approach in section (3.2.2.2) and the NQC approach in section (3.2.2.3), performed the tasks of utilising the RCX robot presented in chapter 2 shown in Figure 25, of detecting surrounding objects adequately well. It was also identified that these two approaches failed to provide enough memory for the provision of a map storing locations of sufficiently large maze. Memory was still an issue even though the NQC direction provided for much smaller variables to be used, by specifying individual bits in a single memory location.

The overall conclusion was to reject the Hardware Embedded Software approach as a viable platform for low cost autonomous robotic systems due to its shortcoming demonstrated above.

### 3.4.2. Software Simulation Direction

Table 7 below gives an overview of the different simulators and the setup required for each. It identifies whether modifications are required to enable the individual simulator environment to function as described in its documentation.

It was identified during the comparison stage of this project that of the 8 investigated simulators, 6 simulators did not present as good candidates to form the basis of the research due to the following reasons:

1. The Khepera Simulator KSim had the disadvantage of requiring a recompile each time something changed.
2. The WSU simulator provided an easy to use interface but had limitations in its environment.
3. The Evorobot simulator was only suited to a GA approach.
4. The YAKS simulator did not function as described.
5. The EasyBot simulator also presented a sophisticated but usable interface but its disadvantages outweighed its usefulness to the project.
6. The Erdos project had limitations in its simulated robot's sensory complement.

The two remaining simulators, the Webots simulator and the Pyro simulator provided a worthy platform to further the investigation. These two simulators are presented in chapter 4 along with the work developed during the research within each simulator.

**Table 7 Installation process of Simulators**

<b>Simulator</b>	Source code compilation required	Source code required editing	EXE file supplied	Functioned after editing source code	Operating System
Khepera Simulator	Yes	No	No	Yes	Linux
WSU Java Khepera Simulator	No	No	No	Yes	Windows
Evorobot	No	No	Yes	Yes	Windows
YAKS	Yes	Yes	No	No	Windows
EasyBot	No	No	Yes	Yes	Windows
Webots	No	No	Yes	Yes	Windows Linux
Erdos	Yes	Yes	No	Yes	Windows
Pyro	Yes	Yes	No	Yes	Windows Linux



## Chapter 4 - Software Design and Implementation

### 4.1. Introduction to the Software Design

In the previous chapter two simulators were identified as valid candidates for inclusion in the software design stage of this research. Each simulator would require more software to be developed during the research for it. This developed software is presented below each of the simulators sections. Outcomes resulting from the developmental work are also presented below each simulator code.

### 4.2. Webots' Environment

This environment provides the user with a very adaptive environment. The user can construct a 3D world that they wish their robot to operate in and this is called the "world file". The user would construct the robot they require for their own world which is also added to the world file. The designs for different robots are provided in the example designs of the Webots environment, but the user has to recreate these designs in their new world file, specifying size, shape and orientation of each of the different faces required to construct the robot. The user develops a controller in software for their robot and assigns this controller's name to the "controller" field under the robot's definition in the world file.

#### 4.2.1. Software Designing in Webots

The design in this environment involves the selection of a pre-existing world or the construction of a new world and, as mentioned already, the world file also defines the robot. The robot can be designed with different means of locomotion; wheels, legs or flight. Its sensory complement is also defined with the location, the number of sensors and type of the individual sensors.

The controller for the robot can be written in C, C++ or Java programming languages. The name of the resulting executable (.exe) \ Java class (.class) file is assigned in the world file under the "controller" field. Each sensor added to the robot in the world file can be accessed by the controller using its individual name from the "name" field in the world file Appendix - B.2.1.

Robots can be designed with the following types of sensors and actuators; distance sensors (infra-red, sonar, Laser), motor wheels (differential wheels),

cameras (colour camera, monochrome camera, range finder device), servos, touch sensors, grippers, emitters (infra-red, radio), receivers (infra-red, radio) etc.

#### **4.2.2. Software Design for the Project**

The world chosen for this investigation was a modified version of one used during the Alife competition (see section 2.3.6.1). The robot featured in this competition was a simulated Khepera robot fitted with a colour camera. The controller used for this environment was a modified version of the Petr Stepan's original Piglet controller. The design in this strand of the project looks at the modification and further development of the Piglet controller for an implementation of a system to investigate sensory redundancy. This controller was selected because it presented an integrated solution to the operation of combining different sensory data and mapping of the arena (shown in Figure 27). The original controller displayed its search strategy as well as its explored areas to the user while it was operating shown in Figure 28.

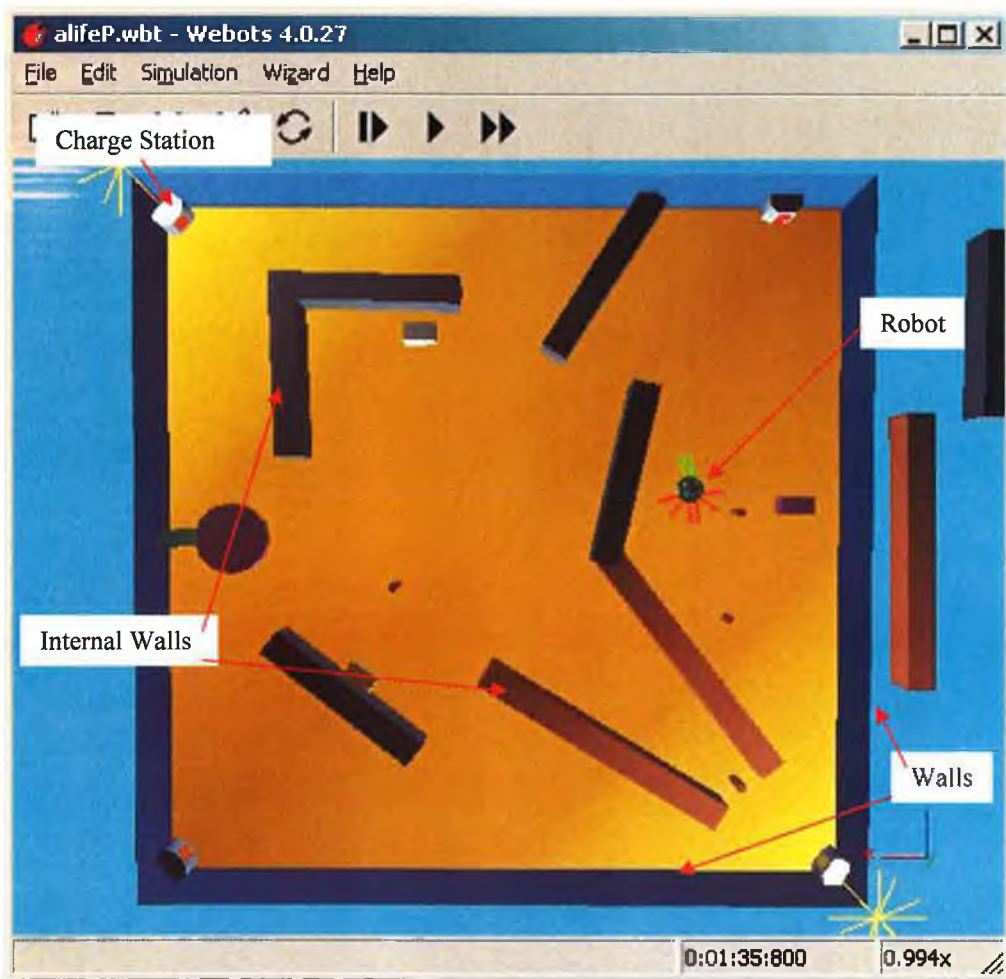


Figure 27 Webots arena

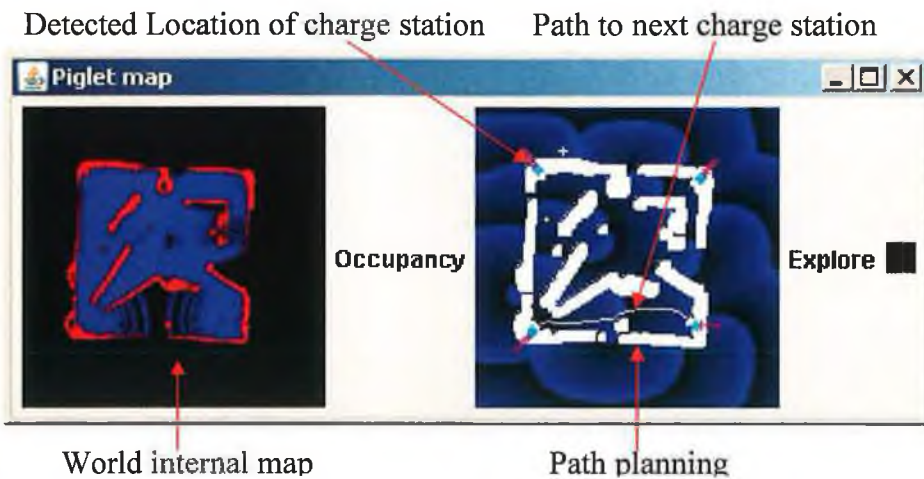


Figure 28 Webots piglet world data

### 4.2.3. Development Work of Piglet Controller

The following files were modified from their original, during the investigation into sensory redundancy:

- **Grid.java:** This is a modified version of the original code. Additional information has been added to this thread to enable the isolation and independent representation of the different information sources being detected by the robot. This information was present to the user in new maps representing only the information from a specific stimulus. Debug information was also generated to identify specifically how information relating to the environment is detected and stored.
- **Outputs.java:** The original purpose of this file was to display to the user the internal data collected by the simulated robot as it traverses the maze. The data displayed to the user was the false colour camera image shown in Appendix B.2-Figure 69 and the combined information from all the sensory data shown in Appendix B.2-Figure 67. This file was modified to separate out the information streams from each of the different sensors. It also displays other internal information in Figure 31. The original combined sensory information is still displayed as shown in Figure 32. Maze information identified by the camera only plus the ground information is shown in Figure 33. Sensory detection of obstacles by the IR proximity sensors is displayed in Figure 34. The robot registered that an object was detected by the proximity IR sensors when the sensor's value indicated that an object was within 40cm or closer. The more pronounced the red colour is at locations in the image shown in Figure 34, indicates the more an obstacle is detected at that area. This results from the robot being close enough to detect and spending long enough to produce multiple readings of that object. Figure 35 represents the combination of the camera decoded data and the IR sensors detecting free space around the robot. The information displayed in these images was utilised to establish the results from alterations in sensory responses.

- `Piglet_edited_v3.java`: Supervisory functionality was envisaged to be advantageous to the exploits of the robot but owing to the missing functionality between Java and CPP implementation in the Webots this was not available.

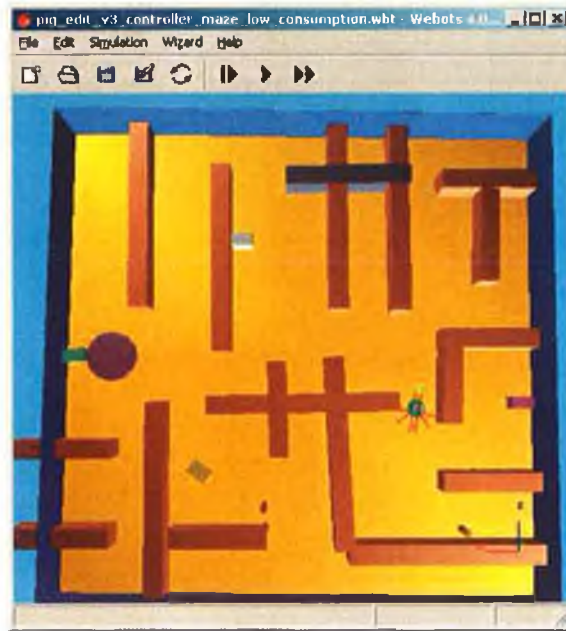


Figure 29 Webots environment

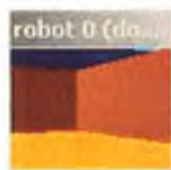


Figure 30 Output of simulated camera

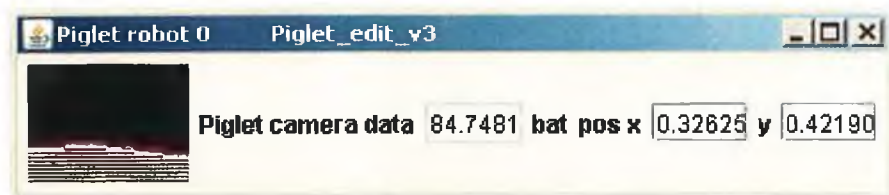


Figure 31 Camera false colour and internal data



Figure 32 Maze data



Figure 33 Camera information only



Figure 34 IR sensor collision data



Figure 35 Ground information

#### 4.2.3.1. Outcome from Modification of the Piglet Controller

The controller implemented an algorithm that produces a false colour image from the simulated camera image. It removes much of the maze detail in

the camera image, except where the walls meet the floor and colour associated with the charging stations.

As the code author remarked, the code is not well commented. This resulted in a challenging undertaking to identify how the different areas in the code operated. This code was investigated by adapting it to produce different maps from the two different stimuli available to the robot. During the investigation it was identified that, while the code performed very well in the specific world constructed for the competition, it would not handle subtle changes in colour, e.g. even where shadows were cast on walls. As a result of these shadows the robot's ability to detect the presence of a wall by the robot's camera is disabled by the controller. The ground of the maze was also required to be of a specific colour. The result is that the code would require adaptation to enable it to operate in a world of different coloured walls and floors. It would also require further changes to enable it to fully decode the camera image; the operation of the code scanned the middle line of the false colour image from the camera for the height above the bottom of the image where floor and wall/object meet. This height measurement is used to calculate the distance this floor-wall junction is away from the robot's current position by producing a distance measurement. The current structure of the code would require extensive rewrites to enable it identify more of the important information from its surroundings from both the simulated camera and the IR proximity data. The current code does not take into account of the possibility that objects may have a gap between themselves and the ground. This code would interpret them as being further away from the robot than their actual location is and this would greatly depend on the distance the robot is from the actual location and their height relative to the robot's camera.

#### **4.2.4. Outcome of Webots**

The code developed in this research highlighted Webots as very powerful simulation engine. The ability to simulate different robots with different sensory arrays presents a very good development platform of a research environment. The investigation into the modification of Piglet highlighted the power of Webots and also the interaction between controllers and sensory array and motors.

### 4.3. Project Work with Pyro.

The Pyro project provides a number of different server environments to allow for the development of different approaches to the development of controllers, these are:

- Player Server
- Pyrobot Simulator

This project focused on the Pyrobot simulator as the server environment to investigate controller development. Under this server the Pyrobot simulator is selected and assigned a world file. Player Server is a robot device interface server which provides a network interface to robotic hardware. It, like Pyro, provides a level of abstraction but has not been ported to the Windows OS. It does not afford the same level of abstraction when designing controllers as Pyro. It can be utilised by the Pyrobot interface when run on a Linux OS.

The code written for this project in the Pyrobot Python simulator falls into two distinct areas. Part of it is to define the simulated arena in which the robot will operate, and the other part is the brain controller used by the robot.

To run the Pyrobot environment the following setup is required see Figure 36, the user selects the different options under the following headings:

- Under the *Server* selection: The user selects the “PyrobotSimulator” and loads the world file “A\_mapFileLoad.py”
- Under the *Robot* selection: The user select the robot referred to as “PyrobotRobot60000.py”
- Under the *Device* selection: The user select the hardware they require to be supported in the particular simulation run, int this case “AllSupported.py” hardware is selected
- Under the *Brain* selection: The user select the controller they require to operate with the selected robot and hardware choice, in this case the “A1\_Robot\_display\_t2.py” controller is selected.



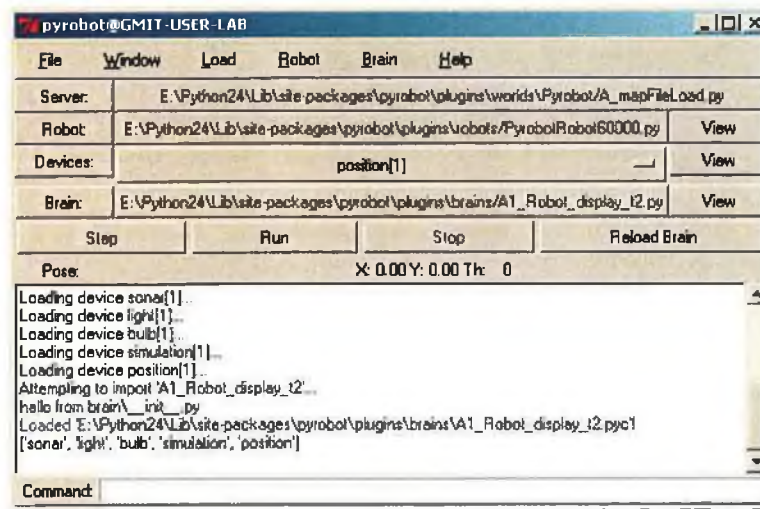


Figure 36 Pyrobot environment

The code developed to define the arena is as follows:

- `globalEnvironmentalVariables.py`: This programme is used to store common data that will be required by a number of different sources in the code. Information such as environmental size, centre position and map file name.
- `mazeExitMap.txt`: This is the file containing the information on wall positions in the maze. It defines the start x and y, and the end x and y position of each wall section. This maze is constructed in the simulated world. See in Figure 38 for resultant maze in simulator.
- `__init__.py`: This is a blank file that needs to be in a folder to indicate to python that it is a package directory. It is run first and thus any methods placed in this file will overwrite existing methods in this folder.
- `pysim_extension.py`: This file defines the object “TkSimulator\_extend” that extends “TkSimulator”. It provides the functionality of loading maze file. The object reads in the file and creates the walls from the provided x and y coordinates. The maze file, is passed to the function `__init__()` as “mapname” parameter.

- `A_mapFileLoad.py`: This file defines the simulation world. It constructs the simulation environment by defining an instance of “TkSimulator\_extend”, adds a robot to this environment, specifies the devices the robot will possess and adds light objects to the world.

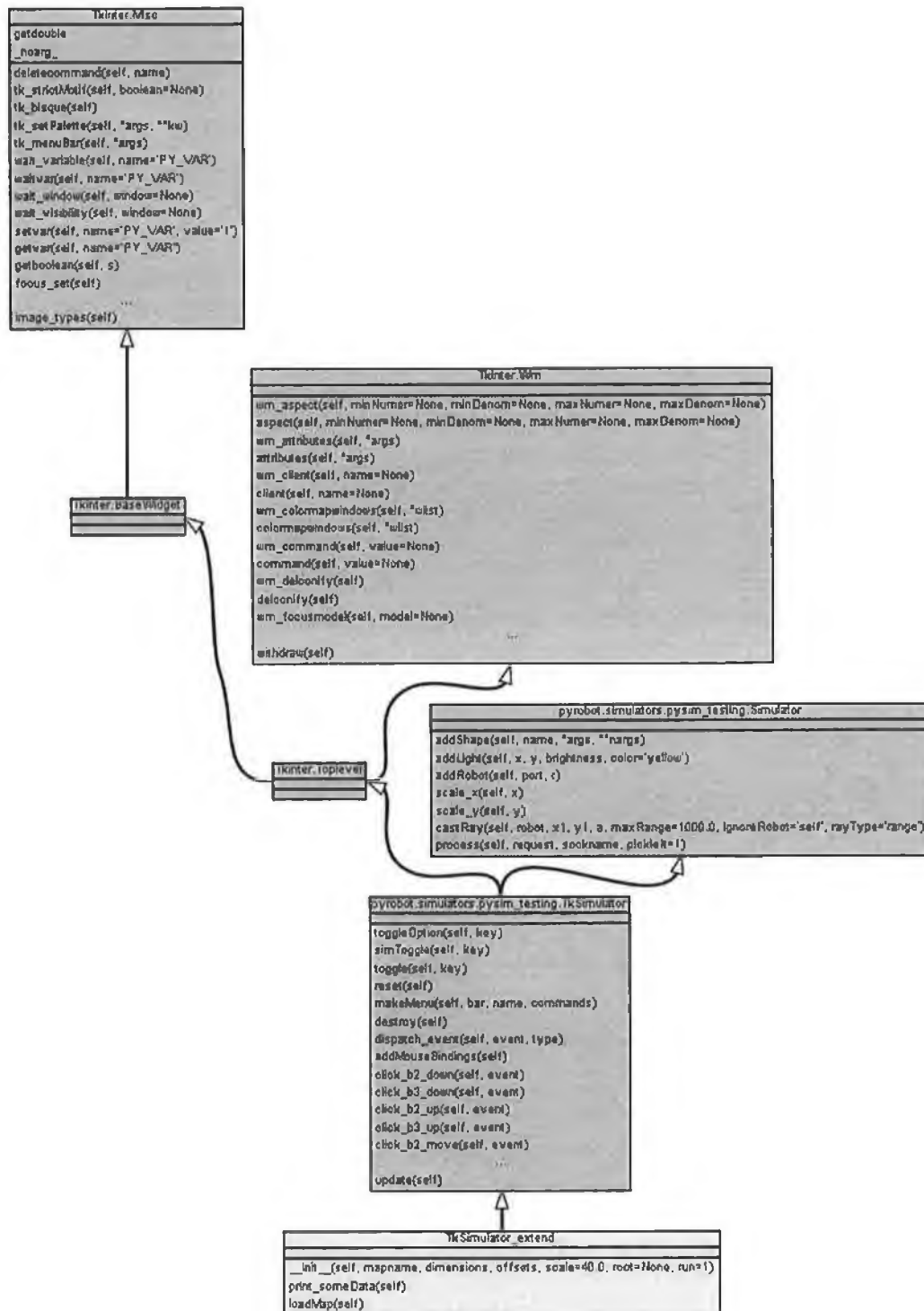
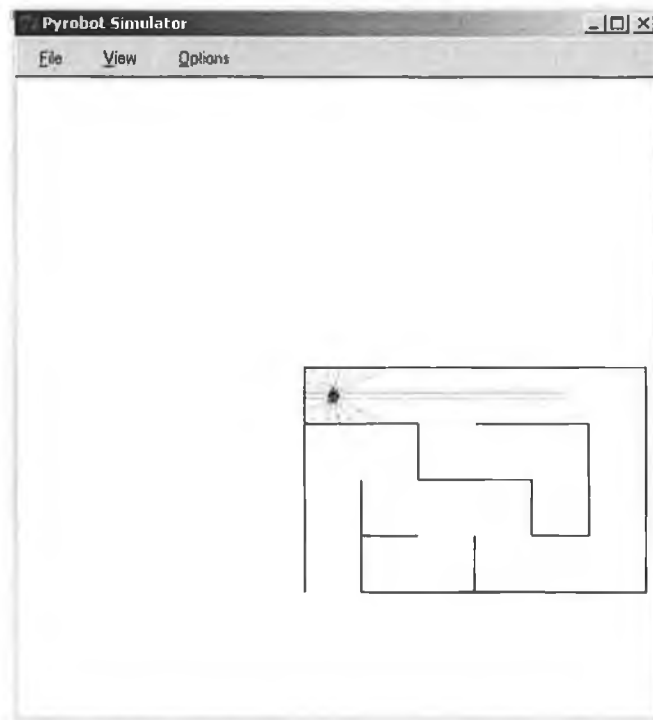


Figure 37 UML Diagram of the simulator environment

Figure 37 is the UML diagram of the code constructed to operate with the Pyrobot Simulator's simulation of the environment. It extends the original object "TkSimulator" to add the aforementioned ability to load a maze map into the simulated world.



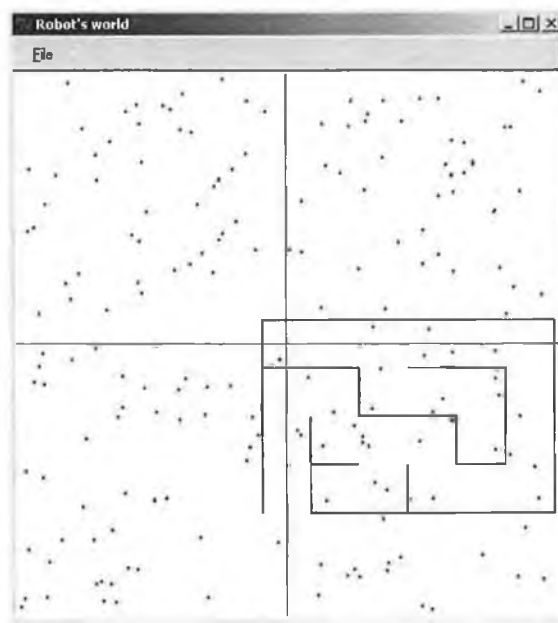
**Figure 38 Simulation environment with maze file loaded**

The code developed to define the robot's brain controller is as follows:

- `mazeExitMap.txt`: This is the same file used in the construction of the simulation arena. It is loaded by the controller to provide the prior knowledge of the maze environment.
- `globalEnvironmentalVariables.py`: This file is also used by the controlling and the simulator parts alike. It is used to define global parameters for the system. It defines the size and mapping scale of the environment, it also includes the name and directory path of the maze file "`mazeExitMap.txt`" in use

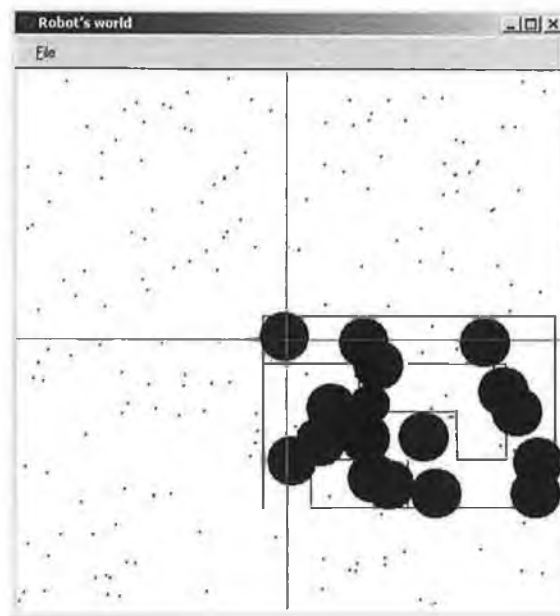
- `robot_data_v4.py`: This file performs the backbone of the controller for the brain of the robot. It holds the methods by which the robot updates its particles. Particles are possible locations for the robot pseudo randomly placed through the environmental space in which the robot is intending to operate. Each particle has a location (x, y), direction and size representing the probability that this particle is the current location of the robot. Figure 39 below shows a typical spread of initially placed random particles. This file includes the methods to perform the calculations to identify the proximity of individual particle location to the relative maze walls. It knows the structure of the maze as this is prior knowledge to the robot, and this is loaded into the environments display, but does not know its current location in the environment. It identifies ghost particles, those which pass through walls as a result of the update process, translating each particle by the amount the robot has moved. These ghost particles cannot represent the current location of the robot as the robot cannot pass through walls and as a result the particle is reinitialised and assigned a new random location. Particles that have similarities in the distance they are from objects, to the robot's calculated distance sensory data, have their weighting increased, as they represent higher probability that their location represents the robot's current location. Conversely particles distance measurements not comparing favourably to the sensory distance data, have their weighting decreased. If a weighting becomes too low the particle is also reinitialised and assigned a new random location. The code displays the internal information to the user by displaying all the particles with representative sized dots indicating their respective particle weighting, see Figure 40 for result of updated particles after a number of initial steps. The code also checks whether the newly created particles reside on a wall, if this result is true they are reassigned different random location values.
- `A1_Robot_display_t2.py`: This file is used to define the brain of the robot. It extends the `TkRobotDisplay` object defined in

“robot\_data\_v4.py” as well as the Brain object. It performs the simple action of driving the robot forward or turning it depending on the values received from the sensor groups: front, left-front and right-front. It updates the particles representing possible locations of the robot in the simulation arena. The particles are moved by the value that the robot has altered its original position and orientation as it performs each step. The controller also updates the weighting of all particles depending on the values received by the robot's sensors, see Figure 40 below. If the particle values tend to agree with measured robot values there is a higher possibility that their locations are the same so the particle gets a higher weighting. The inverse is also the same so particle values that tend not to agree with the robot get a decreased weighting.



**Figure 39 Internal data of robot including the particle locations**

Figure 39 above displays the internal data at the offset. The particles representing possible locations of the robot are pseudo randomly arranged around the simulation arena. These particle weightings will be increased or decreased, if they tend to agree or disagree respectively, with the values measured by the robot's sensors. This is shown in Figure 40.



**Figure 40** Robot's internal data particles increasing weighting

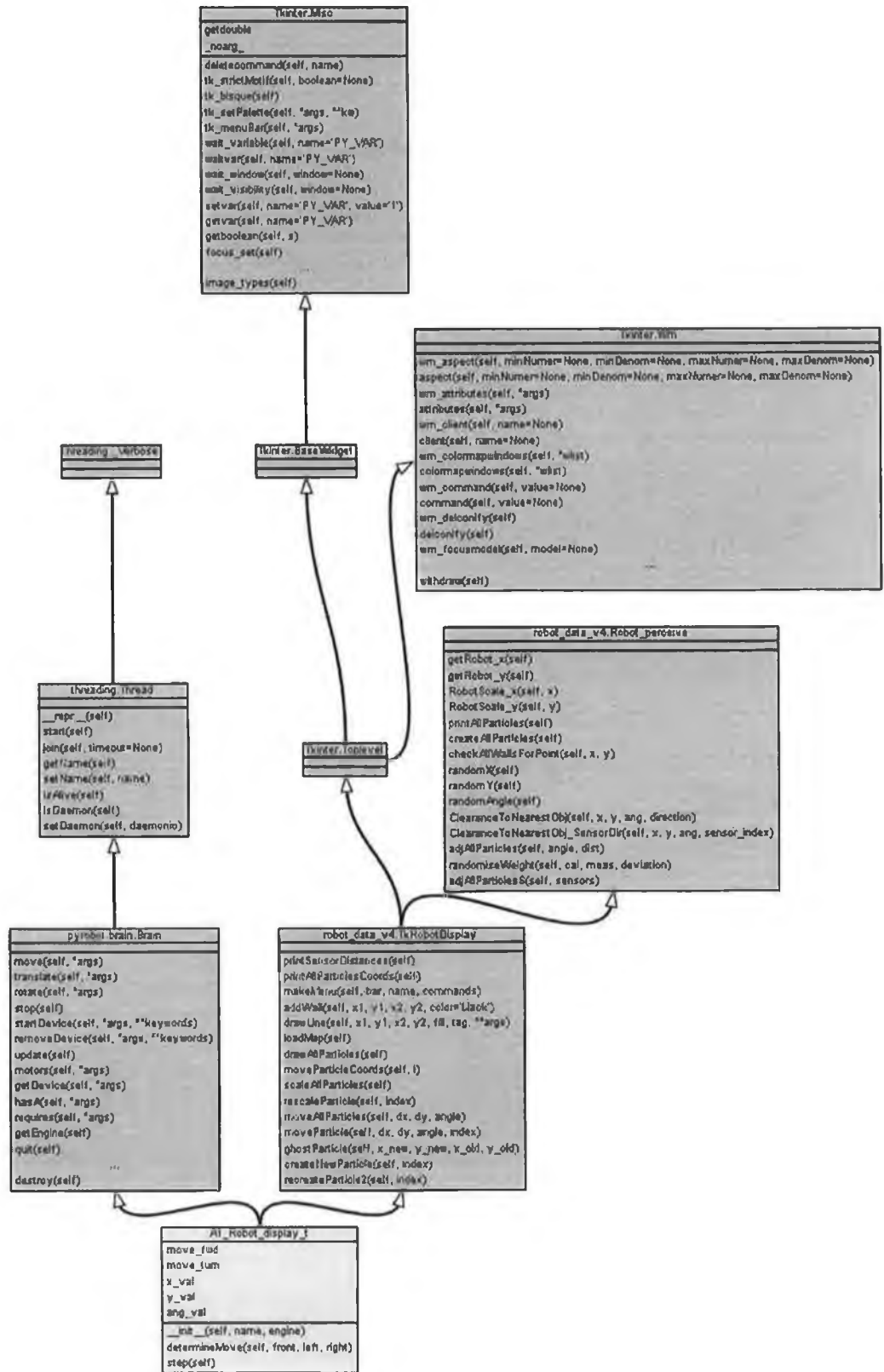


Figure 41 UML of Robot Brain

Figure 41 represents the UML diagram for the Brain controller of the robot. The controller inherits methods from the Brain and “Tkinter” base objects.

This enables the controller to request data from, and cause actions in the Pyrobot simulator

#### **4.3.1. Pyro Outcome**

The work with the Pyrobot Python simulator was in the development of a Monte Carlo Localisation (MCL) algorithm to assist the robot controller to localise itself within the maze environment. The controller used the information detected by its sensors to adjust the weighting of the particles each time it altered its course to identify a possible current location for the robot.

#### **4.4. Results of Implementation**

The results for this chapter's software design and implementation stage will be analysed and presented in the next chapter. An assessment criteria is also presented, which lists the attributes of each simulator under different headings.



## **Chapter 5 - Analysis and Results of Implementation**

### **5.1. Webots**

#### **5.1.1. Analysis of Webots**

The Webots simulation environment provides the user with a very adaptable approach to mobile robot design. The user can perform a design layout for the arena choosing the different coloured structures required for simulation. Each structure that is required to be placed in the arena must include a boundary object. This boundary object is the 3D region that will be detected by the robot sensors as the object's boundary. Inaccurate boundary objects will produce ghost walls and ghost objects resulting in the controller producing erroneous results, confusing the users until they spot their errors.

The physical robot's design also depends on user's choice; the user can design their own robot with any number of sensors and physical shapes and define the wheel placement and rotational direction and orientation. They must also provide the robot with its own boundary object as it also becomes an object needing detection by other robots in the environment. The simulation environment prevents a robot's boundary object from passing through another boundary object.

#### **5.1.2. Results and Observations with Webots**

The Webots software is proprietary software which requires a license fee to be paid. The company also provides a package which includes both the software and a Khepera robot for development purposes. These high costs can dissuade individuals or less resourced centres from developing controllers using this environment.

The Piglet controller using the Webots environment was designed for the Alife competition described in chapter 2.3.6.1. Its use of the simulated camera and proximity sensors by the controller algorithms resulted in it being a winning entry in the competition, initially identified this as a very promising direction of this research. However, following further investigation into this controller, this did not result in a valid approach for the project because the algorithm was coded to work only with specific colours in the maze arena. In addition, the algorithm only partially decoded the information from the simulated camera. This algorithm was

sufficient for the competition entry, beating off the other entries but lacking the extendibility to other tasks without significant changes to the algorithm. These changes would require extensive rewrites due to the complex nature of the code and the author's lack of using descriptive comments in it, identifying how specific functions are used, and the reasons for the choice of variables. Code revisions were attempted during the project, but it became somewhat of a trial and error situation to alter the code to achieve desirable results. This resulted in painfully slow progress and ultimately identified the limitations of the approach without a significant work effort to rewrite the controller code.

## **5.2. Pyro**

### **5.2.1. Analysis of Pyro**

This approach to controller design for mobile robots, addresses the problem from the point that the designer (user) may not want to, or need to, know the specifics of the robot. The user requires the robot to move forward, but they may not know that for a specific robot A to move forward one of its specific motors needs to turn in a specific direction, or, that for the specific robot B, 2 motors need to turn. For example, the user may just want the robot that is currently selected to drive forward. Pyro takes the view that each robot with different motor configurations for driving forward, will be handled by a single "robot" object function. This function will enable a user's controller to seamlessly control different robots with no modifications, whether the robot is wheeled or legged, requiring one or many different motors to facilitate forward motion. This allows for the abstractionisation of the user's controller for the underlying physical hardware of the robot.

The Pyro environment takes this abstraction approach also to the measurement of distance. The units used in measurement can be defined in terms of robot units, one unit being equal to the size of that robot. This also allows the controller to operate varying sizes of robots. For example, if a robot was of size 55mm, as in the case of the Khepera robot, or 44cm as in the case of the Pioneer Robot, it makes little sense to define physical metric measurements in order to specify that the robot should travel a fixed distance in centimetres or metres, because the Khepera speed is rated in cm/s compared to the Pioneer which is rated

in m/s or much larger robots which maybe rated in km/s. This enables the controller to operate on the environmental scale of the physical robot. This enables the user to design one controller that can investigate in the scale of a room with a Khepera robot, a building with the Pioneer or a much larger area outdoors such as an urban area with a robot of much greater size than the Khepera robot.

The robot units can also be utilized in the measurements acquired from the robots sensors. The sensor measurements can be returned in robot units. The sensors themselves can be treated with abstraction. The user is not required to know what and how many sensors the robot has. They can request sensor measurements that represent regions in the robots sensory complement, e.g. sensors that form the forward object detection, left side, right side or rear detection. This also extends to groups within these e.g. right-front and left-front. The user can request, from the sensor information returned, the particular type of sensor the information is coming from, if desired, but it may not be necessary for the operation.

### **5.2.2. Results with Pyro**

This abstraction approach does come with certain caveats, a controller that requires a large number of ultrasonic distance sensors to perform its operation will not perform as expected where a robots physical “distance” detection is performed by tactile collision sensors. The large variation between a graduated knowledge to the proximity of an object and a “can’t see” vs. bang, the robot has stopped, approach, requires an altered controller design [84].

Pyro also enables the user to design with the specifics of the robot in mind. The user has access to the raw data of the sensory outputs displaying the measurements in metric units. They can also request that the robot move forward a required metric distance if so required. The controller can also access the specific number of sensors or, more powerfully, they can query the number of sensors available and utilise this information during operation. This ability to request the available sensors during operation enables the user to adapt their design during operation depending of availability of sensory information.

Pyro is presented to the world as an open source programme, unlike Webots which does require a licence. This enables more limited research departments and hobbyist alike to experiment and develop controllers and worlds, allowing them to test their designs and provide them with the ability to realise their aspirations in the design of mobile controllers. They can then interact with a real robot, if one is available, with the controller they have developed in simulation.

### **5.3. Assessment Criteria**

The assessment of the simulation environments is carried out under the headings in Table 8.

**Table 8 Assessment criteria of simulator's environment**

<b>Assessment Criteria</b>	<b>Webots</b>	<b>Pyro</b>
Adaptability of robotic Arena	Ability to design complex shapes. Each shape requires an additional boundary object	Ability to create complex shapes.
Portability to real Robotic Hardware	Controller can be used without alterations on the physical robot it has been designed for.	Controller can be used without alterations on a VARIETY of physical robotic platforms.
Development of the Environment	VRML97 3D description language	Python Programming language
Controller Development	C/C++ and or Java can be used to develop a controller for a specific robot	Python is used to develop a controller for a VARIETY of physical robotic platforms
Extendibility	Does not allow for alterations as this is proprietary software.	Owing to open source nature the Environment is fully adaptable
Allows for simulated flight	Yes	Currently not implemented
Simulated Robotic platforms	Variety	Variety

## Chapter 6 - Conclusions and Recommendations

### 6.1. Conclusions

This thesis has presented and reviewed a number of different approaches generally available for the design and development of autonomous robotic systems. It compared the development of various hardware systems with differing embedded software environments to the development of robotic systems using a computer simulation approach. The robotic simulation approach involved the investigation of available software simulation packages, and the selection of the most appropriate one for the development of autonomous robotic systems.

This investigation identified that the RCX platform provides the user with a good prototyping platform but it does not contain sufficient resources on board to store adequate information required to store a map of its environment. This is a major requirement for an autonomous mobile robotic system. This shortcoming can be overcome by the inclusion of a base station computer to store information, but this means that the design is no longer completely autonomous.

The simulation investigation in the project assessed different simulation environments in the application of the development of autonomous robots. These simulation environments were critiqued using the following headings:

- Simulation of complex shapes in the environment to better representing Real Worlds.
- Adaptability of the environment
- Controllers portability to real Robotic Hardware
- Ease of development of the Environment
- Ease of the development of the Controller and controller languages
- Extendibility of the Simulation Environment to allow for further development
- Simulation of different robot Platforms. These represent real robot platforms and / or purely simulation created ones.
- Modifiability of the Robot under simulation.
- Interaction with the Robot during simulation, this will allow for the robot to be kidnapped.

- Simulation of different Stimuli
- Licence Cost of Simulator.
- Operating System supported by the simulator

From the results of the assessment stage the most promising two were selected for further investigation. Of these two simulators, Webots is proprietary software requiring a licence fee and the other Pyro is presented as an open source programme.

Webots presents to the user a simulation environment where the user can design their world and robot and develop a controller for that robot and simulate it in this environment and assess the validity of their controller.

Pyro presents the user with a similar ability to design a world, which has been further extended by the work in this project to allow the Pyro simulator to load a file containing wall locations information. The interface then allows the user to select a robot to run their controller on. The difference between Pyro and Webots comes in the design of the controller. In Webots the controller must be designed for the specific robot, in which it is intended to run. The individual sensors must explicitly be identified to allow the controller to operate. Pyro on the other hand uses an abstract approach to a robot. This means that it allows the user to interact with the sensors but without explicitly identifying a particular robot's configuration. The user just needs to refer to the distance sensors and the Pyro environment identifies the particular robot currently being controlled and returns its equivalent sensors. This removes the need for the user to constantly keep in mind that a particular robot has X number of sensors and that number Y sensor is pointing in a given direction. The huge advantage of this abstraction is it allows a controller designed for one robot to be installed in a different robot and it will still operate as intended.

Pyro, because of the use of abstraction, is highlighted as the most appropriate to the design of autonomous systems. Where any further development is envisaged, changes in the hardware of the robot or the robot itself will not result in the controller becoming void.

The code developed in this project amounted to 8000+ lines consisting of original code and modification of other author's code. This does not include the significant code refinement iterations required during testing and evaluation of the controllers. This is contained on the included CD and is available to other researchers.

The original files provided at [www.pyrorobotics.org](http://www.pyrorobotics.org) required adaptation to allow them to properly operate under the windows environment. The updates files are available on the CD and the changes are referenced in the appendix to this thesis.

## **6.2. Recommendations**

Pyro simulation environment is not fully implemented to run under the windows environment because the simulated camera driver has not been compiled for this operating system. It is only available under the Linux operating system environment which can be accessed through installation of Pyro on a Linux box or using the Live CD option which provides the user with a non invasive way to run Linux on a Microsoft Windows computer. The computer boots off the CD loading the appropriate files into ram leaving the hard drive unchanged, unless the user chooses to store certain files on the computer's hard drive.

Owing to the availability of Pyro under the open source licence the environment allows users and developers to adapt Pyro in any way that they require.

On the hardware side a newer version of the LEGO MINDSTORMS Bricks has since been released viz. LEGO MINDSTORMS NXT which provides extra memory and faster processing speed [85]. This may provide an interesting platform for the further investigation to the development in autonomous robotic environments.



## Appendix A - Hardware Code

### A.1. LeJOS Code

The following software code was constructed to implement the desired behaviours. Testing was carried to verify that the desired functionality was achieved.

#### A.1.1. Communication Testing with LeJOS

Testing of the communication between the RCX and the base station was carried out with the following procedure.

- TestRCXComm.java: This programme tested the communication between the RCX and the base station by sending characters '1', '2', '3' across the IR communication port. The flow chart is illustrated in Figure 42 below.

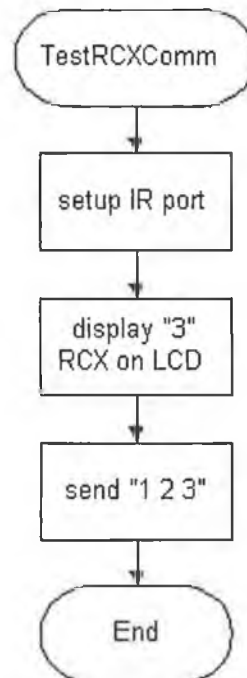


Figure 42 TestRCXComm flowchart

- SerialListenerTest.java: This programme tests to see if a packet of data is available on the IR port. It reads in this data and tests to see if the first byte is an Op-code command if so it combines the next 2 bytes lower byte first then higher byte to form integer. It checks to establish if this integer is valid and

displays this number on the RCX's screen for visual conformation that the communication link is operation. Figure 43 below is a flowchart of the code.

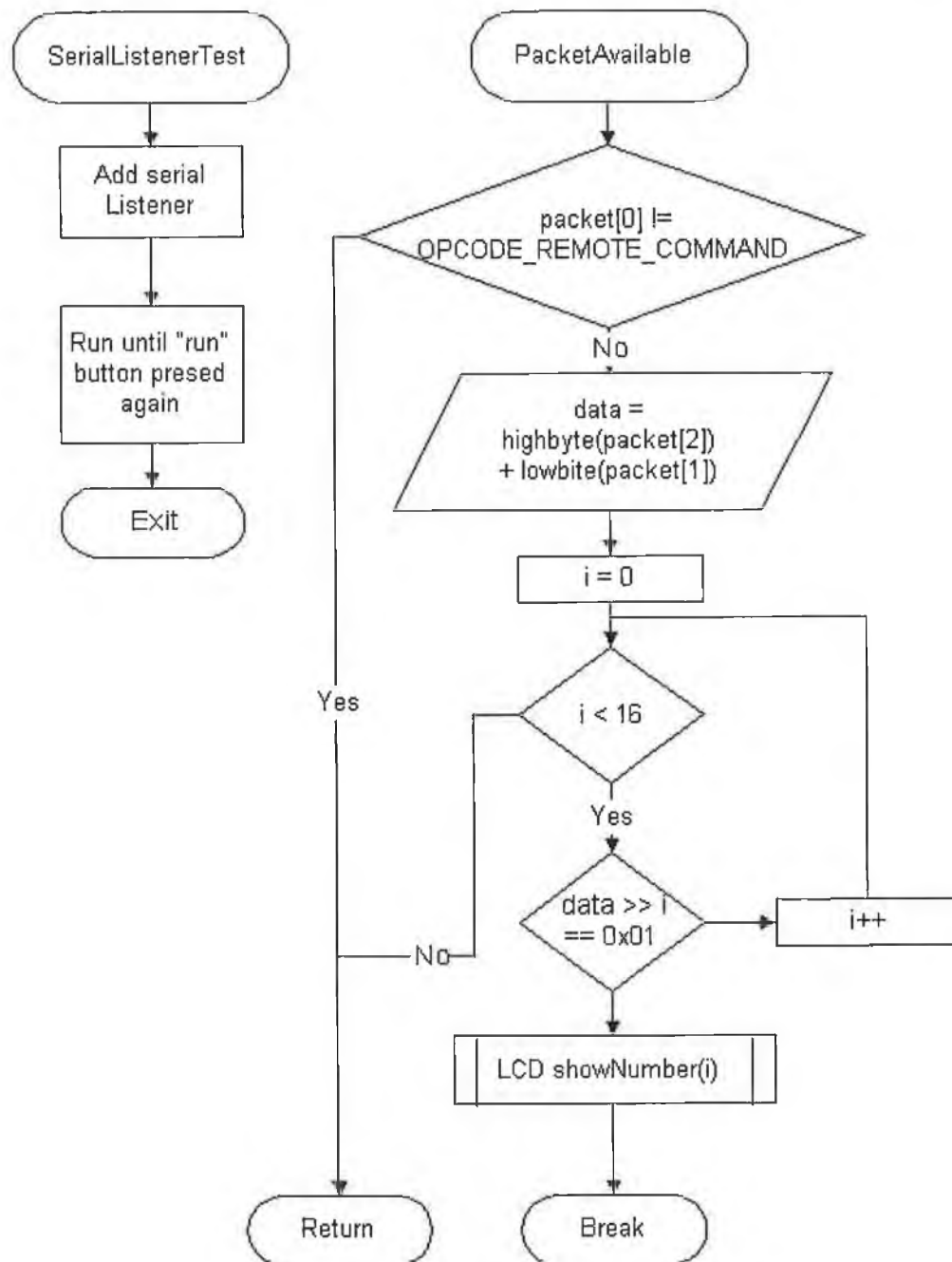
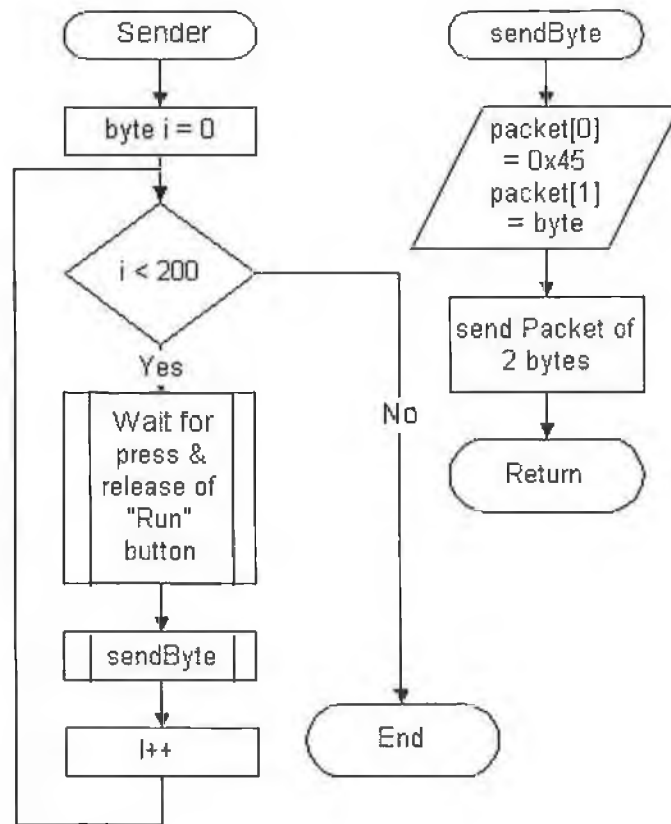


Figure 43 SerialListenerTest Flowchart

- Sender.java: This programme tests the ability to send a packet of data to the base station. A flowchart is given for the code in Figure 44 below.



**Figure 44 Sender Flowchart**

- **SendDistanceValues.java:** This programme drives the robot around and detects collisions with obstacles on the left, front and right, transmitting on collision the location of the RCX robot to the base station. The code generates the transmission packet from 14 bytes of information. The information sent is the: X and Y position and the angle values. Each value is a float, which consists of 4 bytes of data. These 3 float values, plus 2 additional bytes as packet start and end byte make up the 14 bytes sent. The flowchart is illustrated in Figure 45 below.

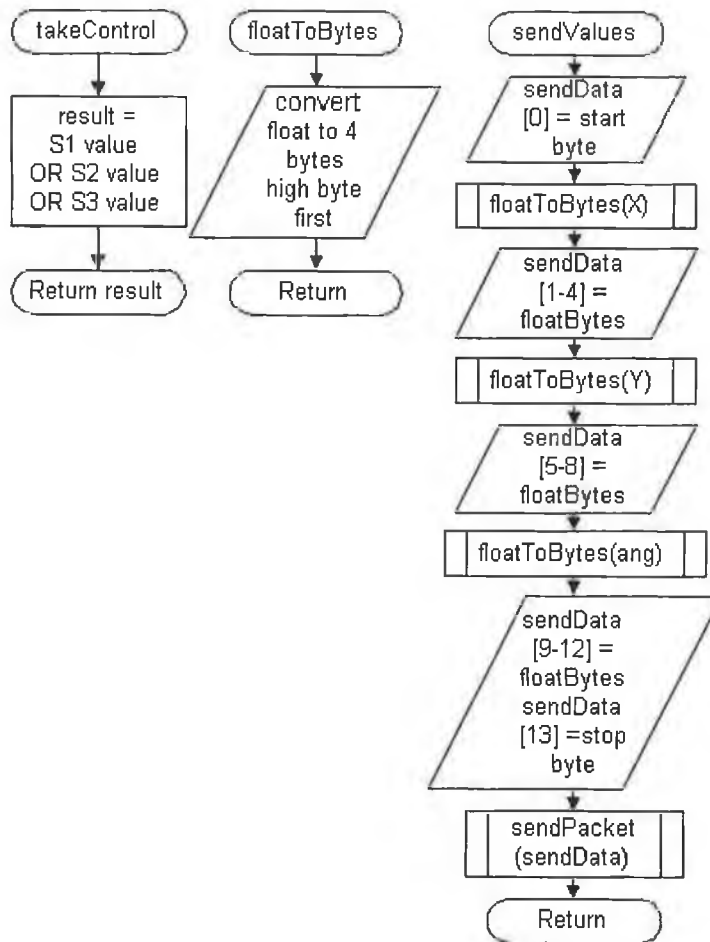
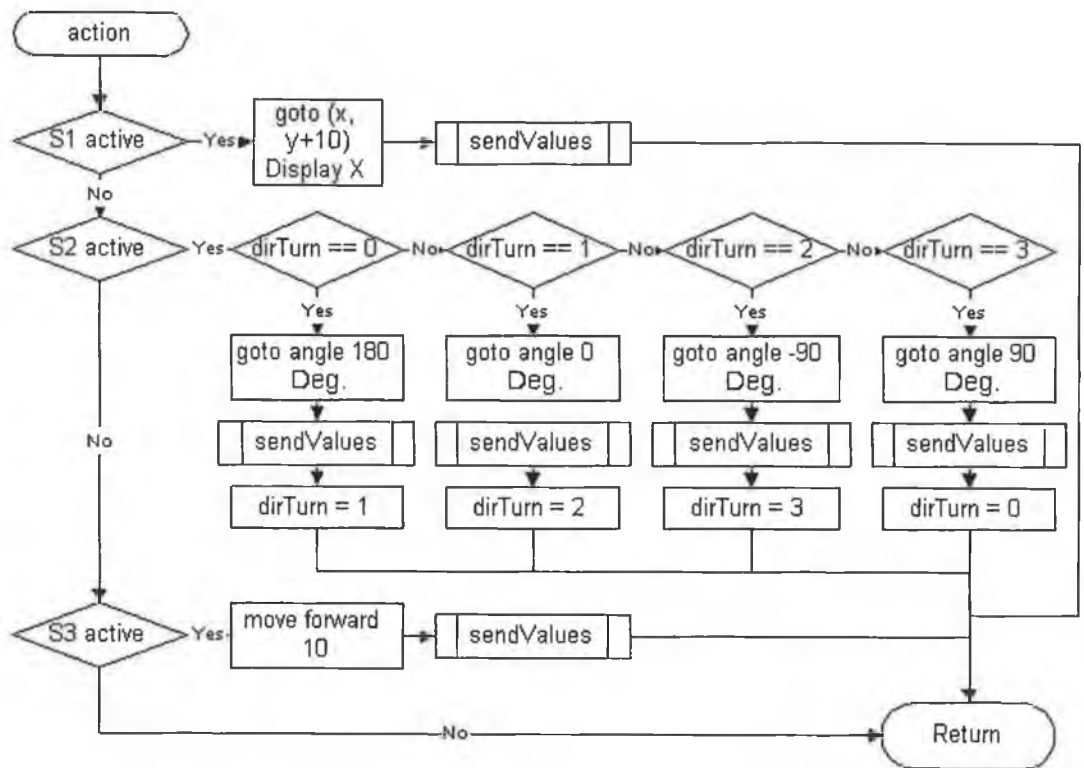


Figure 45 SendDistanceValues Flowchart

- `RunSendDistVal.java`: This programme is the driving programme to assign the behaviour of `SendDistanceValues` as the top-level task in the RCX. It is used to test the correctness of operation of the `SendDistanceValues` code. Figure 46 is the flowchart for this code.

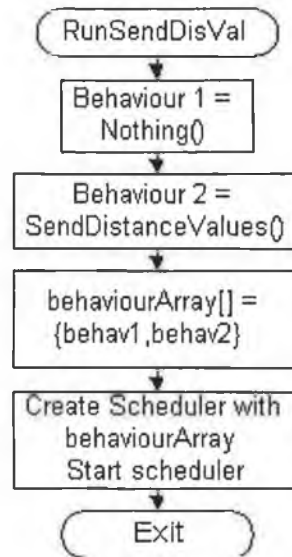


Figure 46 RunSendDisVal Flowchart

- `Receiver.java`: This is another programme to test the IR comm. port. It generates a beep if the port receives data. It also displays the number value of the received byte on the RCX LCD display. The flowchart is given in Figure 47.

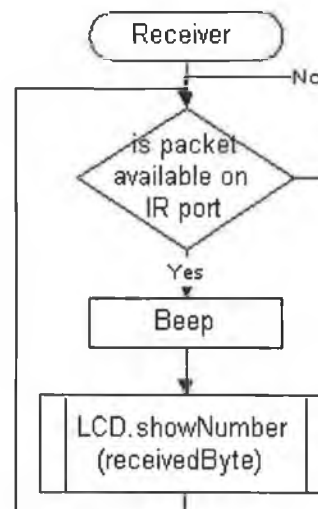


Figure 47 Receiver Flowchart

- **TestRCXComm4.java:** This code is designed to run on the PC side of the communication link. It tests whether or not the IR tower is connected, and if there is a connection to the RCX unit. It then receives the information via the IR tower from this RCX unit and displays this on screen to the user. The flowchart is in Figure 48 below.

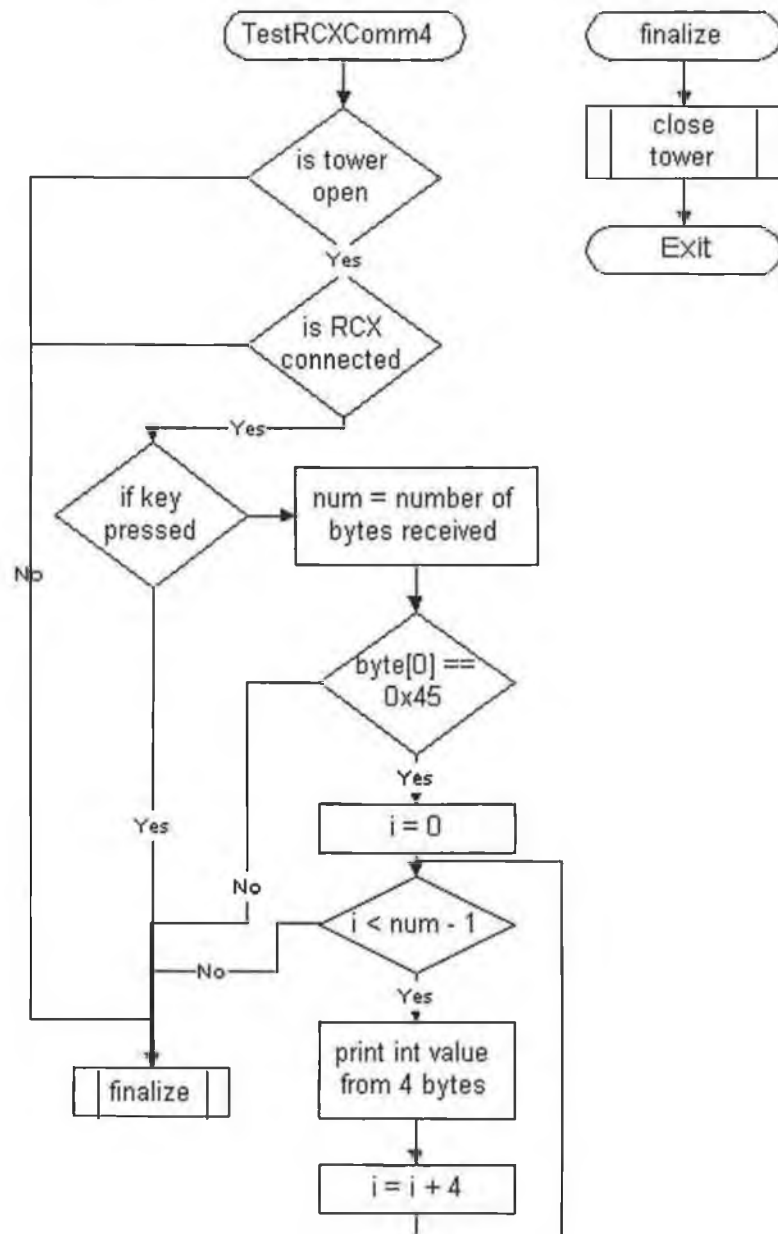
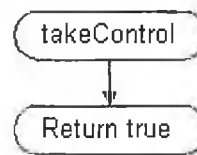


Figure 48 TestRCXComm4 Flowchart

### A.1.2. Mapping ability Testing of RCX using LeJOS

Testing on a mapping approach was carried out with the procedures below. The robot storing detected objects to the left, front and right in an internal map. It was then programmed to transmit this saved map to the computer for verification. The user would then view the received map to check it for completeness

- **Nothing.java:** This programme is designed as the lowest level task, which does, as the name says nothing. This task is operational when no other task requires control of the process. The behaviour takes control by returning “true” from the method “takeControl()” Figure 49.



**Figure 49 Nothing Flowchart**

- **BehaviourSendData.java:** This programme is a modified version of the inbuilt Behaviour.java file. It adds the functionality to each behavioural process. The functions that are added in the update are “floatToBytes(float)”, “sendCoordinates()” and “sendMaze(MazeArea)” see Figure 50 below for flowchart of these functions. This equips each defined behavioural process in the driver file, allowing the process to update the base station if it requires such functionality.

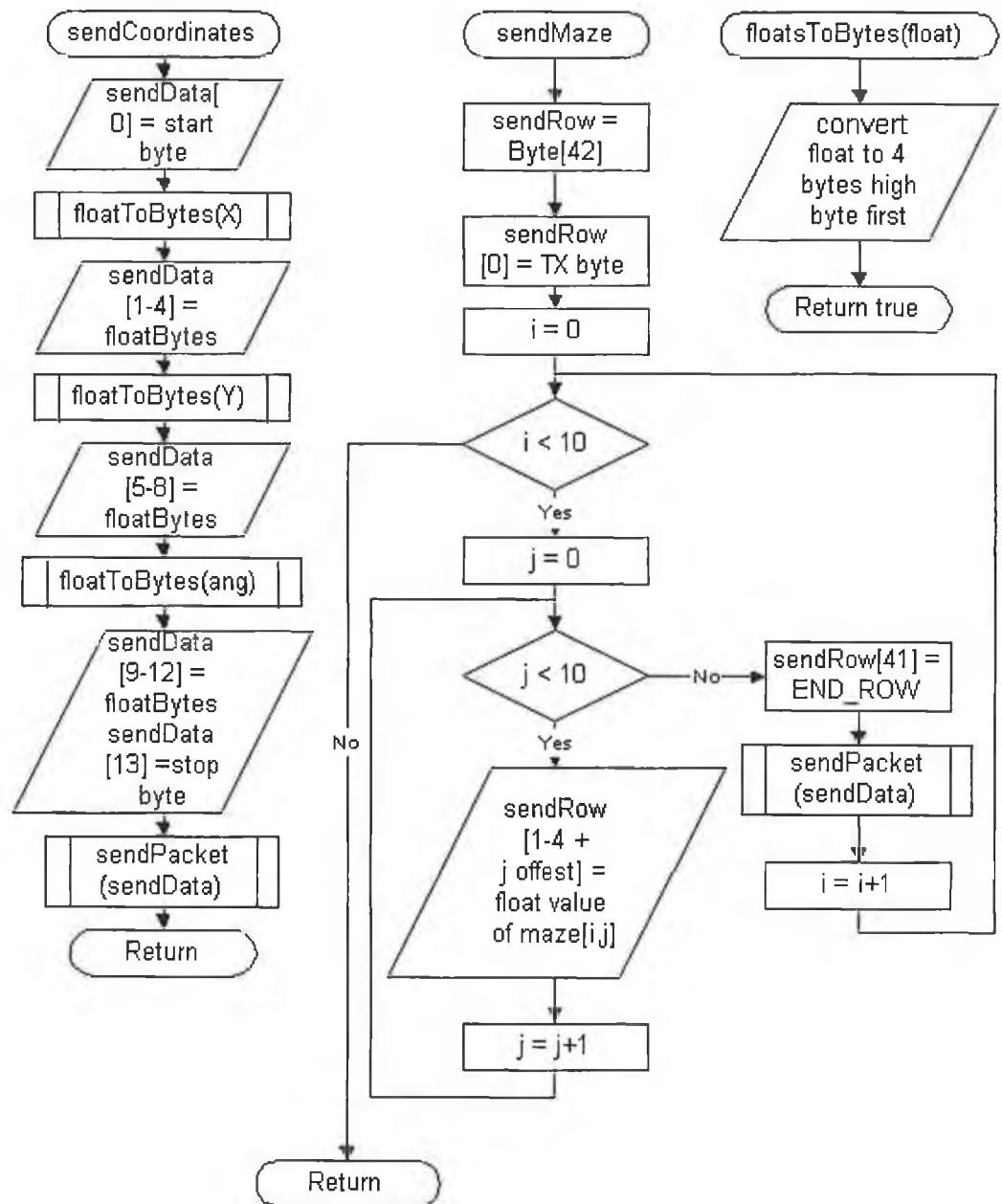
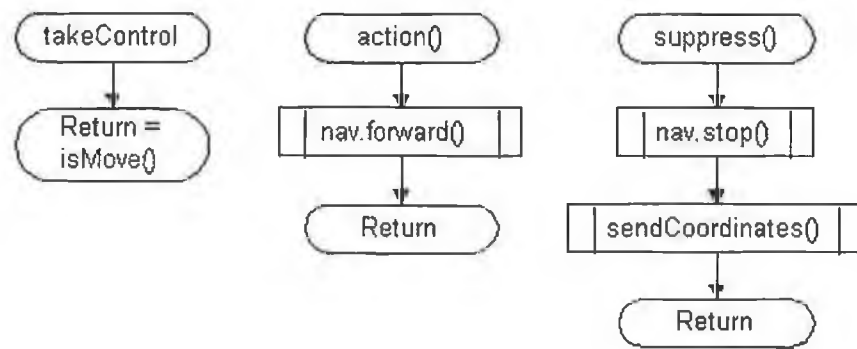


Figure 50 BehaviorSendData Flowchart

- **LowLevelDrive.java:** This programme performs the task of driving the RCX robot forward. The flowchart is shown in Figure 51 below. It inherits from BehaviourSendData super class.





**Figure 51 LowLevelDrive Flowchart**

- **LeftRightBumper.java:** This code requests process time, by returning true on a call to its “action” function when it detects an object on the right and/or on the left. The current location of that object is then added to the maze using the set setObstacleLeft and / or setObstacleRight. The code also indicates the presents of such obstacles by highlighting the appropriate input indicators on the LCD screen of the RCX. Flowchart for this code’s functions is in Figure 52 below.

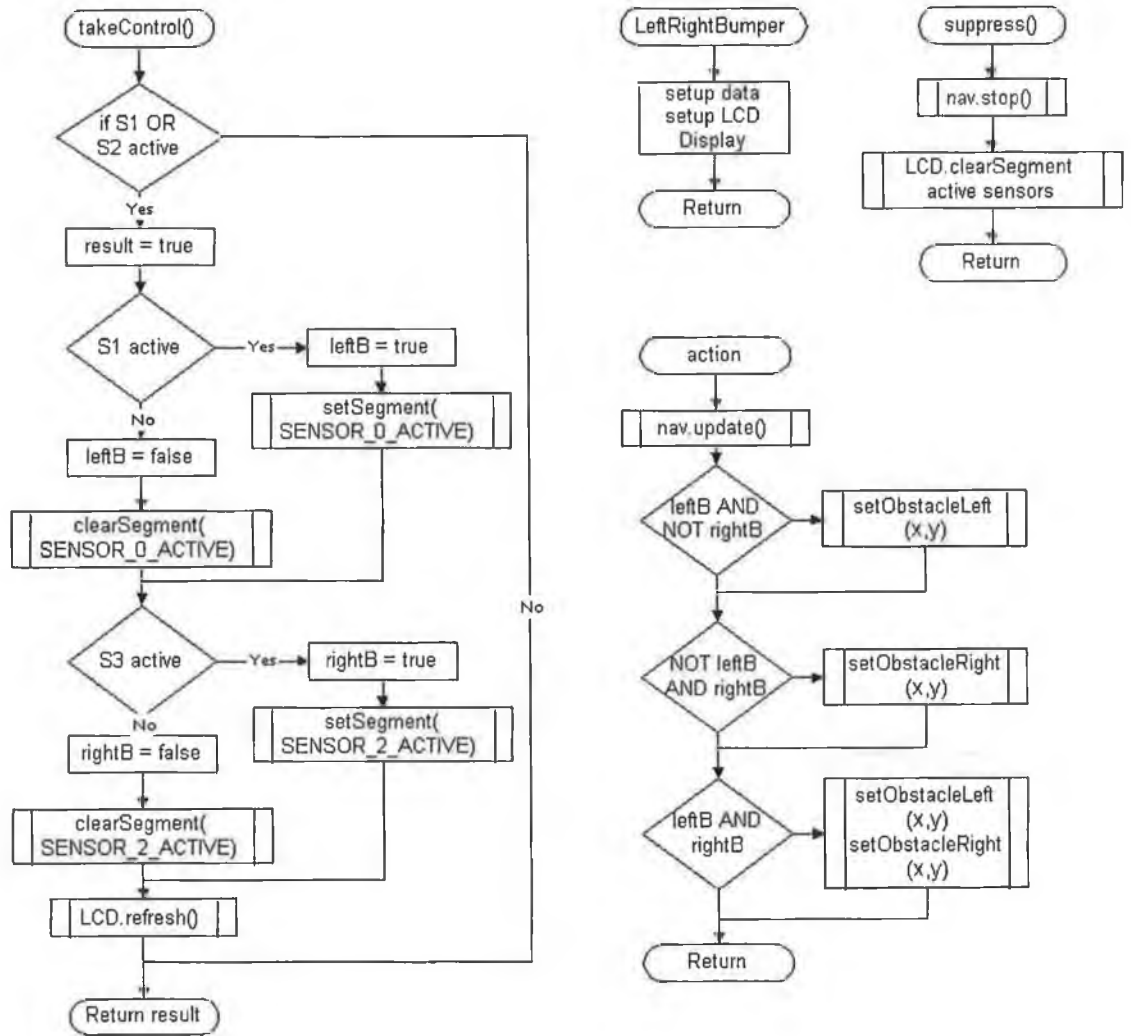


Figure 52 LeftRightBumper Flowchart

- FrontBumper.java: This is the highest priority process task. It requires highest priority owing to the fact that a front collision stops physical forward progress of the robot and it requires immediate stopping of the wheels turning to limit error to dead reckoning navigation. Once a front bumper collision is detected the procedure in Figure 53 is activated. During the “takeControl()” function, the process updates the path count in the in the internal representation of the maze, from the last detected front collision.

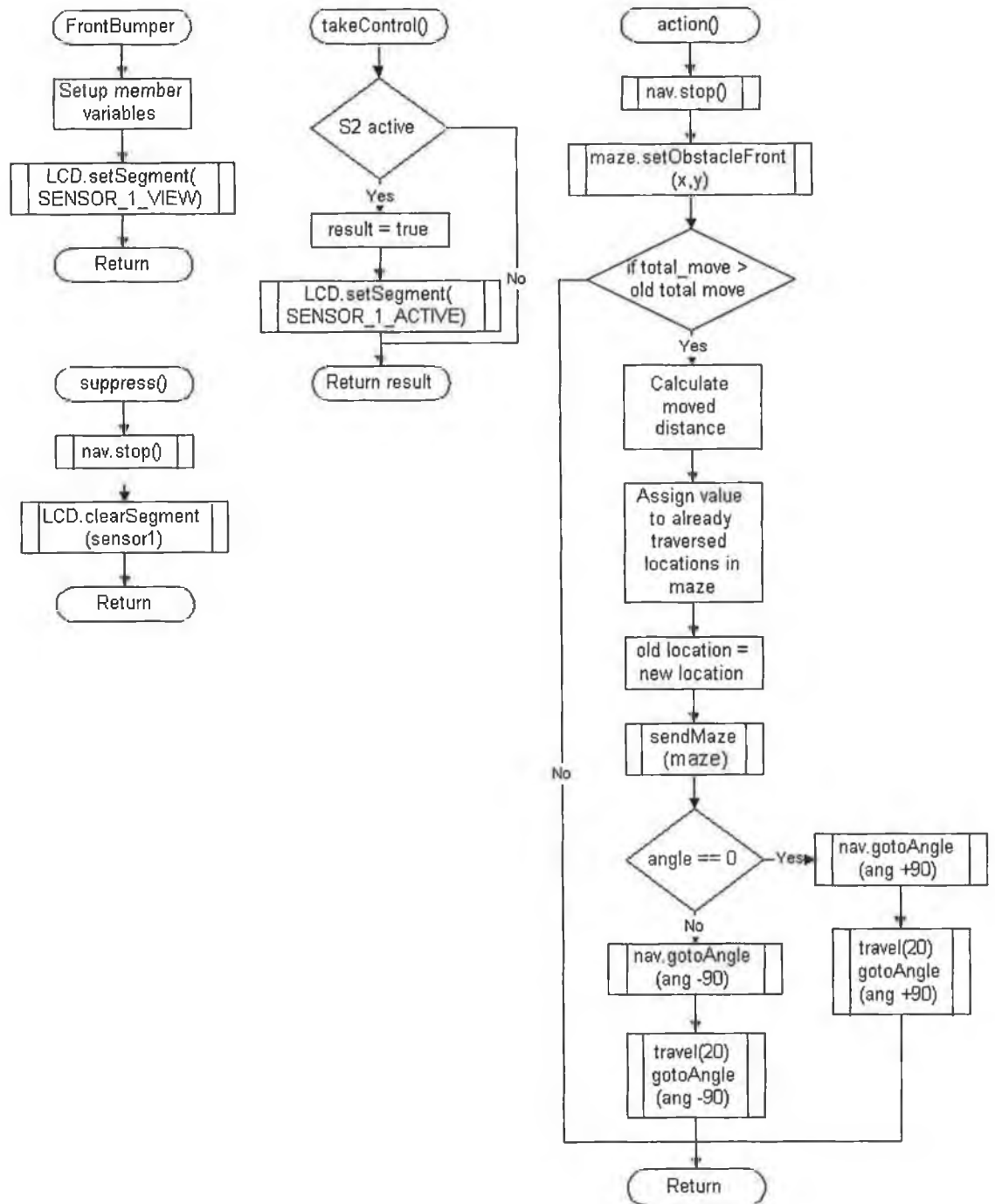
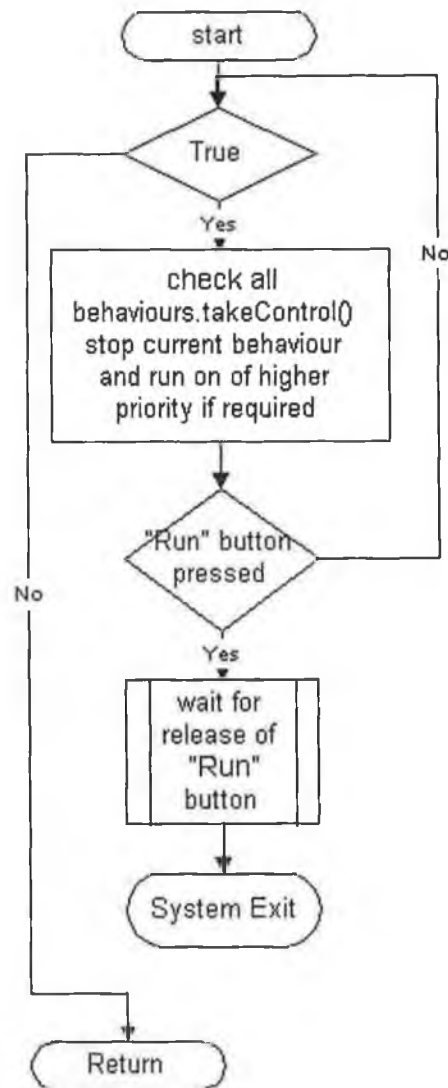


Figure 53 FrontBumper Flowchart

- Arbitrator.java: This original code has been edited, to add the functionality, which empowers the “run” button on the RCX module as a reset button. A push resets the programme flow back to the start awaiting the next push of the run button. The flowchart in Figure 54 includes the new functionality added. The “RUN” button is checked throughout programme operation, enabling the user to interrupt execution by stopping the code executing.



**Figure 54 Arbitrator Flowchart**

- **MazeArea.java:** This code allows for the creation of an object of type `MazeArea`. This object will hold the internal map produced by the robot of its surroundings. It represents the robot's view of the world as discrete areas blocks. Each block can be assigned a value indicating that location to be an obstacle or if traversed by the robot, the "path count" on traversing it. Figure 55 and Figure 56 illustrate the methods provided by the object to update the knowledge of the maze.

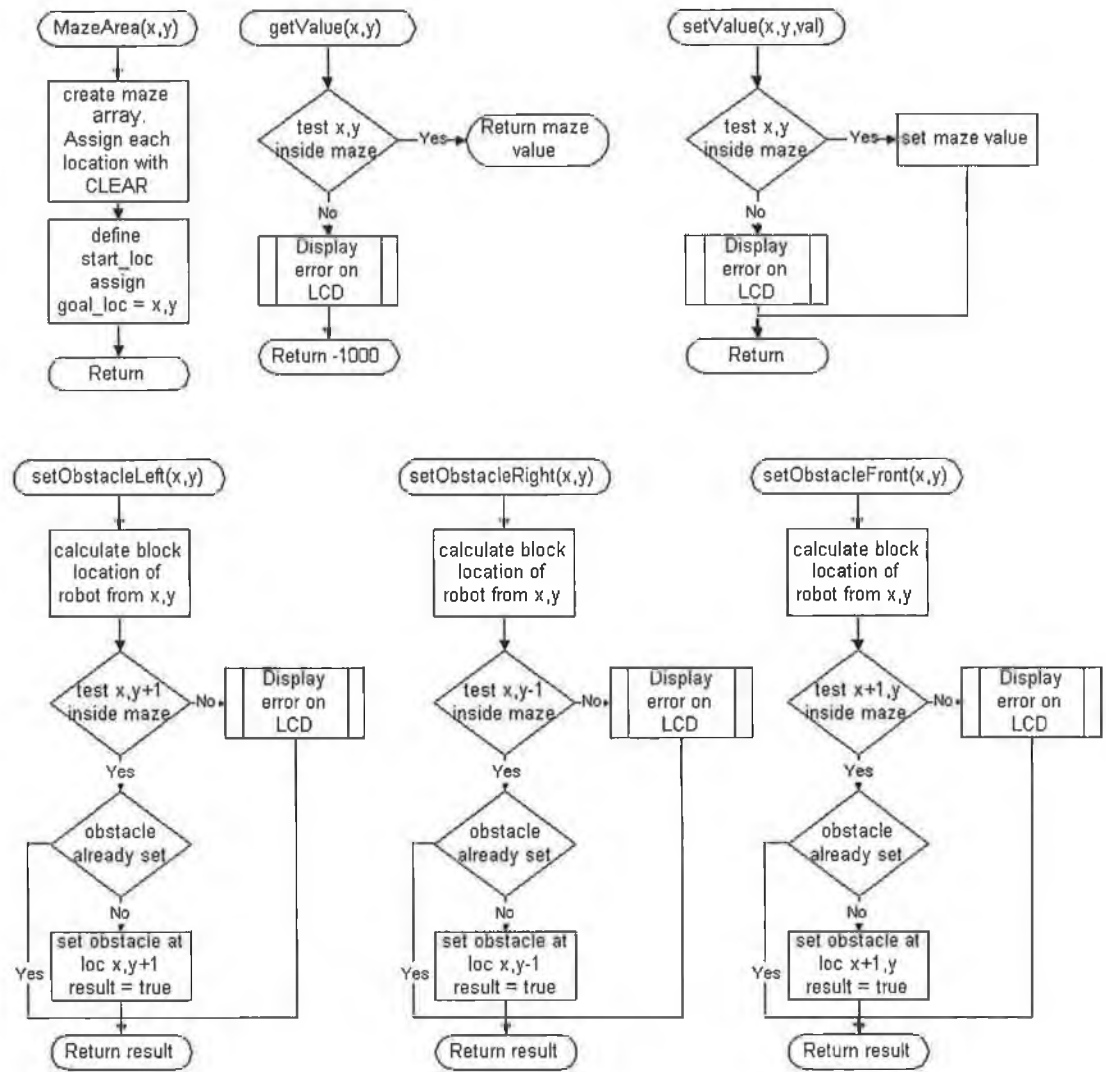


Figure 55 MazeArea Flowchart A

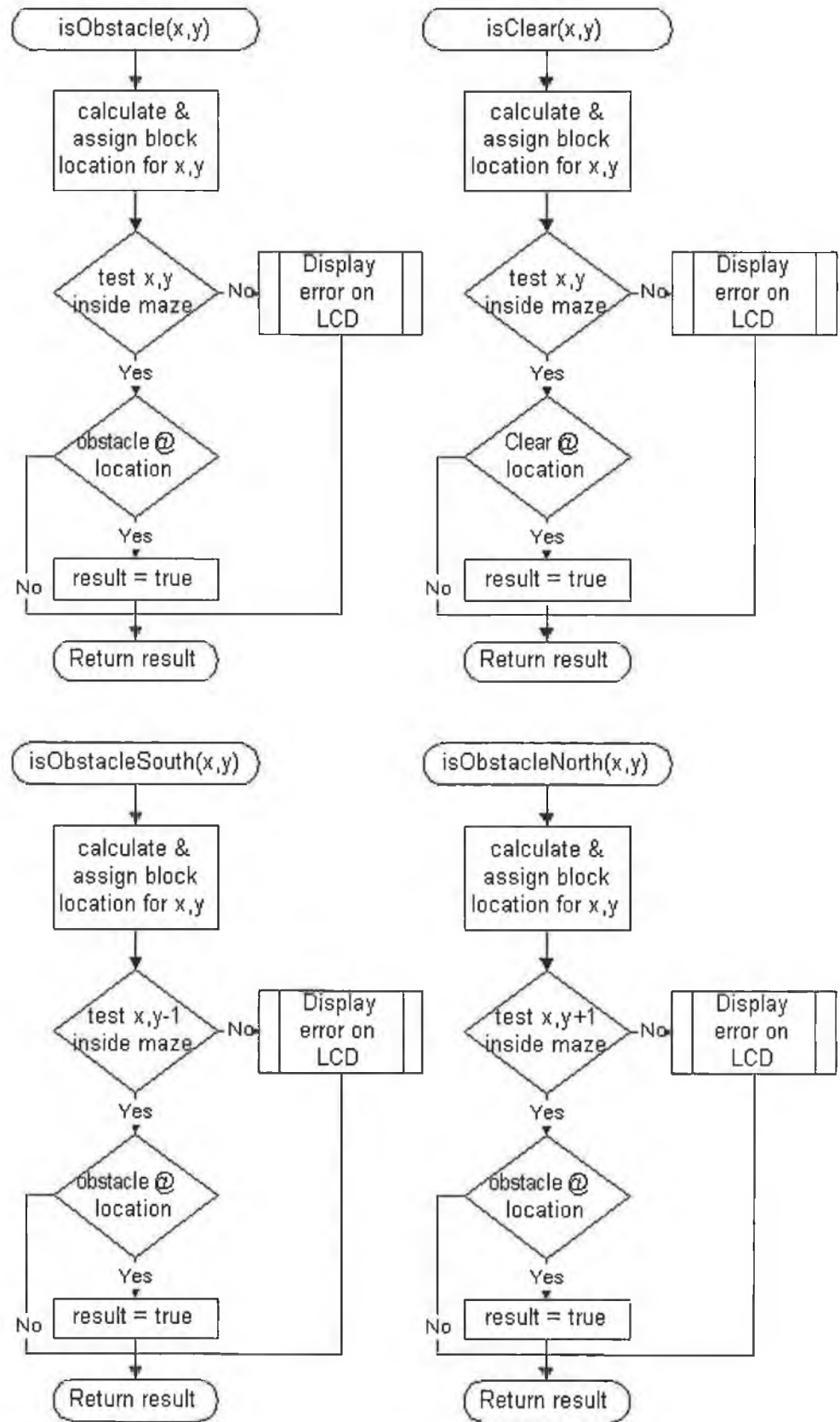


Figure 56 MazeArea Flowchart B

- `AbstractSearchEngine.java`: This code is used to interact with a `MazeArea` object. It is adapted from Mark Watson's book [86] the code from which is released under the Open Source Artistic License. The Code was used to assess the possibility of allowing the RCX to search its internally self constructed maze. It holds the current "path count". It interacts with the maze to allow searching of that maze to determine possible next moves. This is shown in Figure 57.

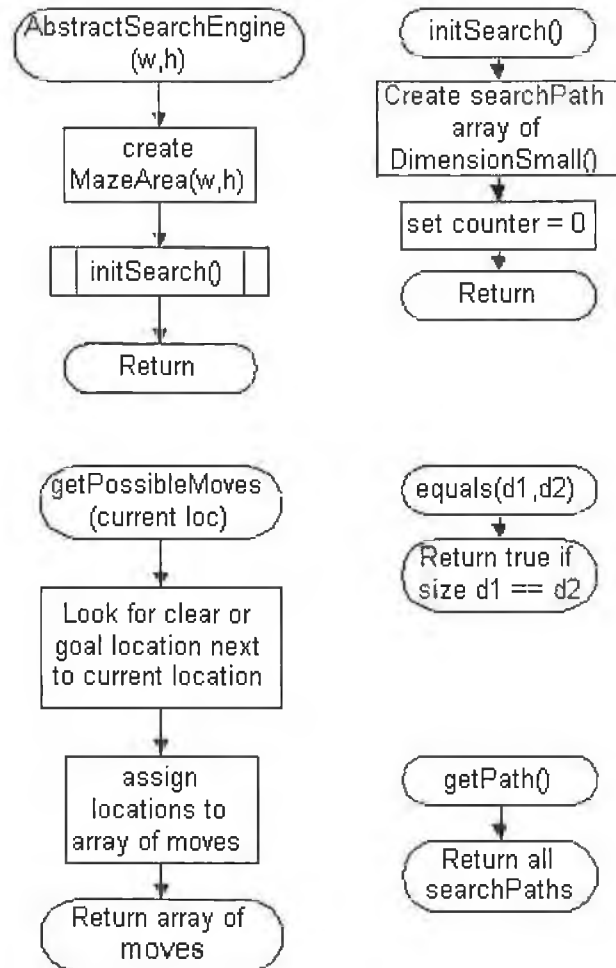


Figure 57 AbstractSearchEngine Flowchart

## A.2. NQC Code

### A.2.1. Navigation Testing with NQC

The following procedures contain the code produced to investigate this hypothesis:

- `motors_navigation.nqc`: The development with NQC started with investigating the ability of the RCX robot to respond appropriately to events. The code assigns two events to each of the 3 different inputs on the RCX as shown in Figure 58 function “`setupEvents()`”. These inputs are assigned as input buttons “`setup()`”. One event is assigned to a press on an input, second for releasing that input. Detecting and recording these would enable the robot to identify when objects come into contact with one of its inputs or when it lost contact with the object. Implementing the event management in Figure 59 enables the robot to detect obstacles on the left and right and collisions between obstacles and the front bumper. This code assigned the variable “`event_occurred`” the value of the event and proceeded to implement the response to the event in the event management function.

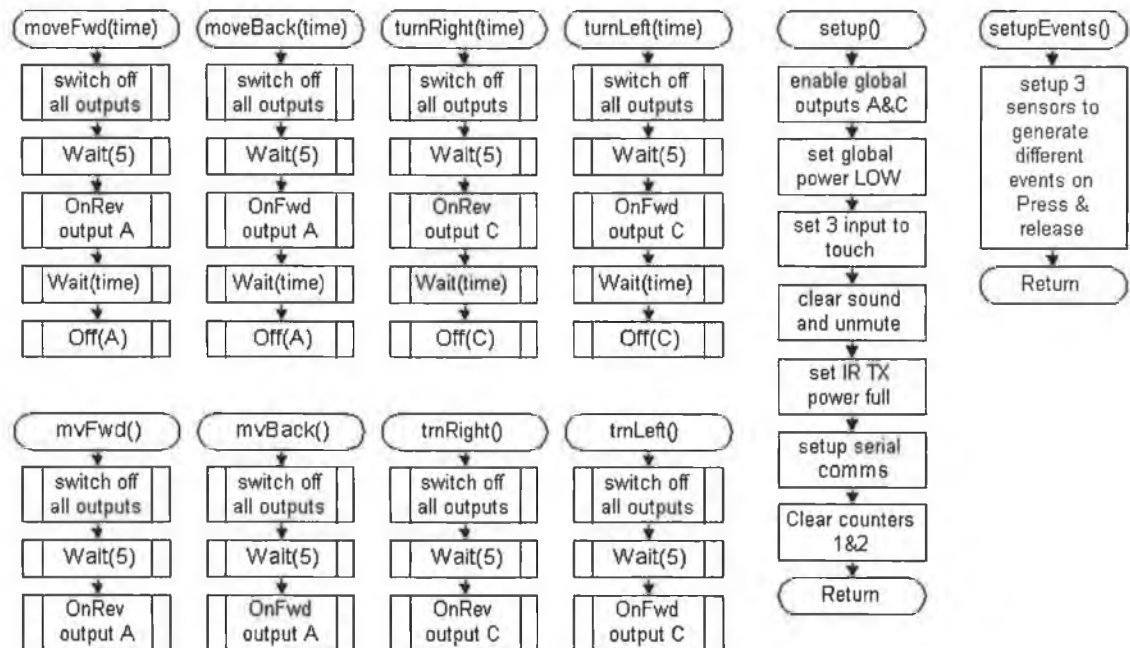


Figure 58 `motors_navigation` Flowchart A



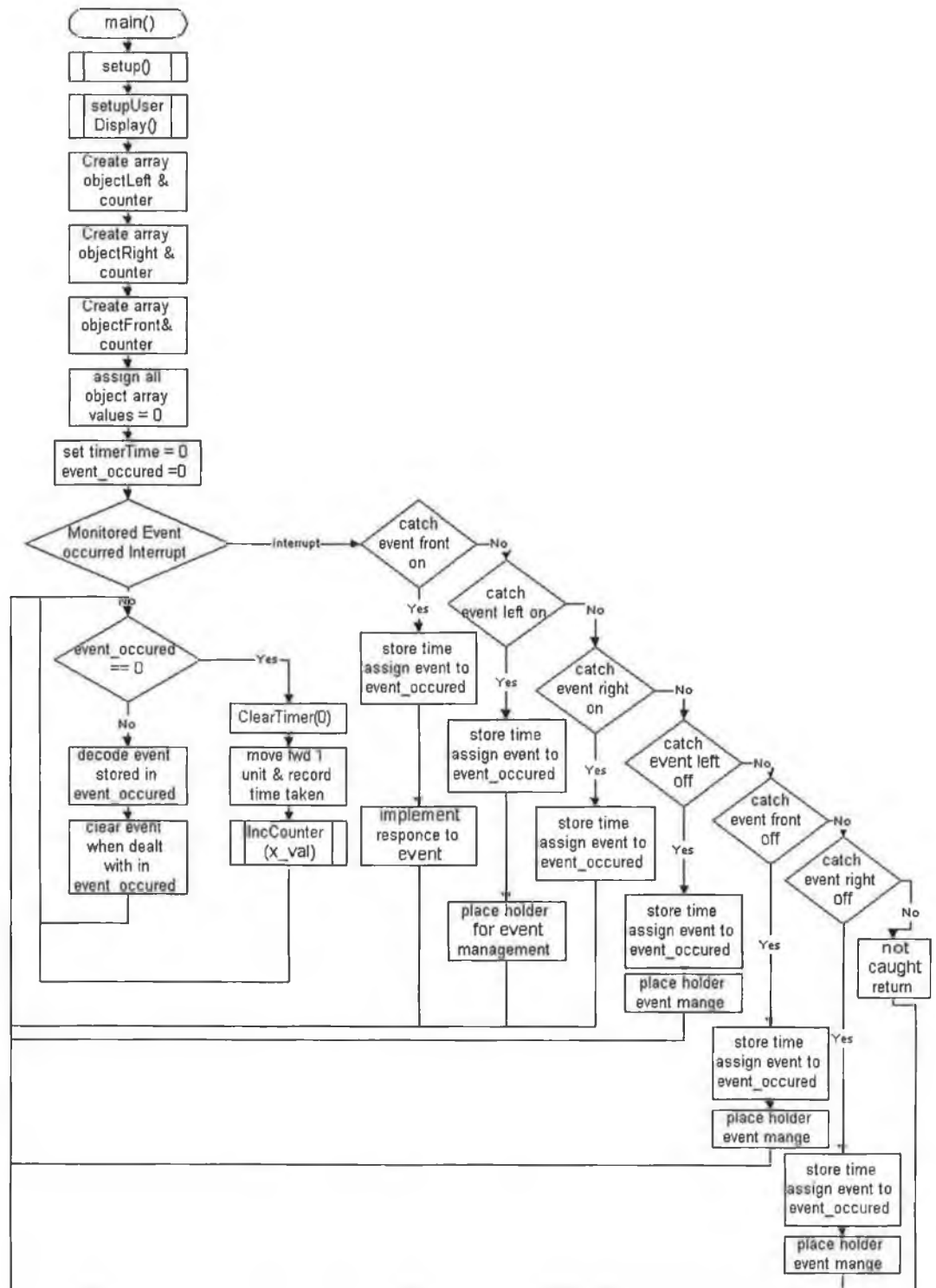


Figure 59 motors\_navigation Flowchart B

- motors\_no\_navigation.nqc: This programme improved the previous version in its event management as shown in Figure 60. The events in this programme are again recorded in an “event\_occured” variable, which enables the events to be managed by the main programme while

still allowing the robot to detect new events. The difference this time is that the event management only sets or clears the appropriate bit in the variable `event_occurred`, the advantage this presents to the operation is that the setting or clearing on one event has no effect of the status of the other event. The operation then returns to the main programme, which dealing with the specific event. The main programme detects that an event occurred by checking the variable “`event_occurred`” bits, if the appropriate variable bit is set the main task can carry out the required operation.

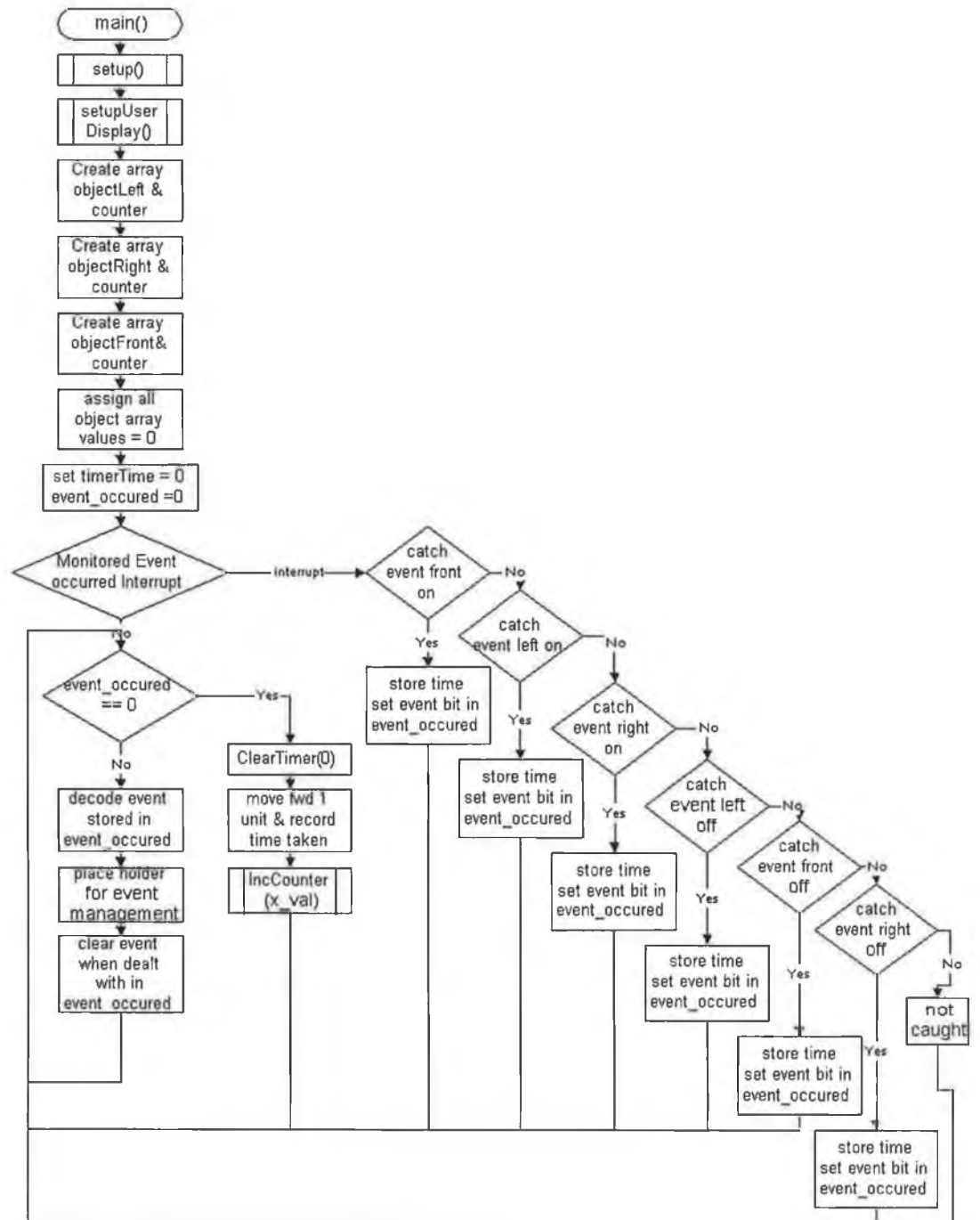


Figure 60 motors\_no\_navigation Partial Flowchart

Using this method of passing back control quickly to the main process as opposed to one where the event handler manages the complete response to the event has the following advantages. The system is able to respond to multiple events and if an event of higher priority occurs before service that, the current

event can be recorded and servicing after the higher priority event has been dealt with.

### **A.2.2. Map Storage Testing using NQC**

Storing the map of surrounding objects was assessed using the following code:

- `motors_navigation_single_array.nqc`: This code was used to investigate the possibility of storing a map on the RCX internal memory. This map would allow the RCX robot to store locations of obstacles around the RCX's environment. It was deemed that while attempting to create an array of sufficient memory locations that the RCX internal accessible memory using the NQC approach was insufficient and this code could not be used on the robot. Figure 61 below illustrates the updated approach with location of object storage in the array "map". Arrays for `objectLeft` `objectRight` and `objectFront` would allow for object tracking.

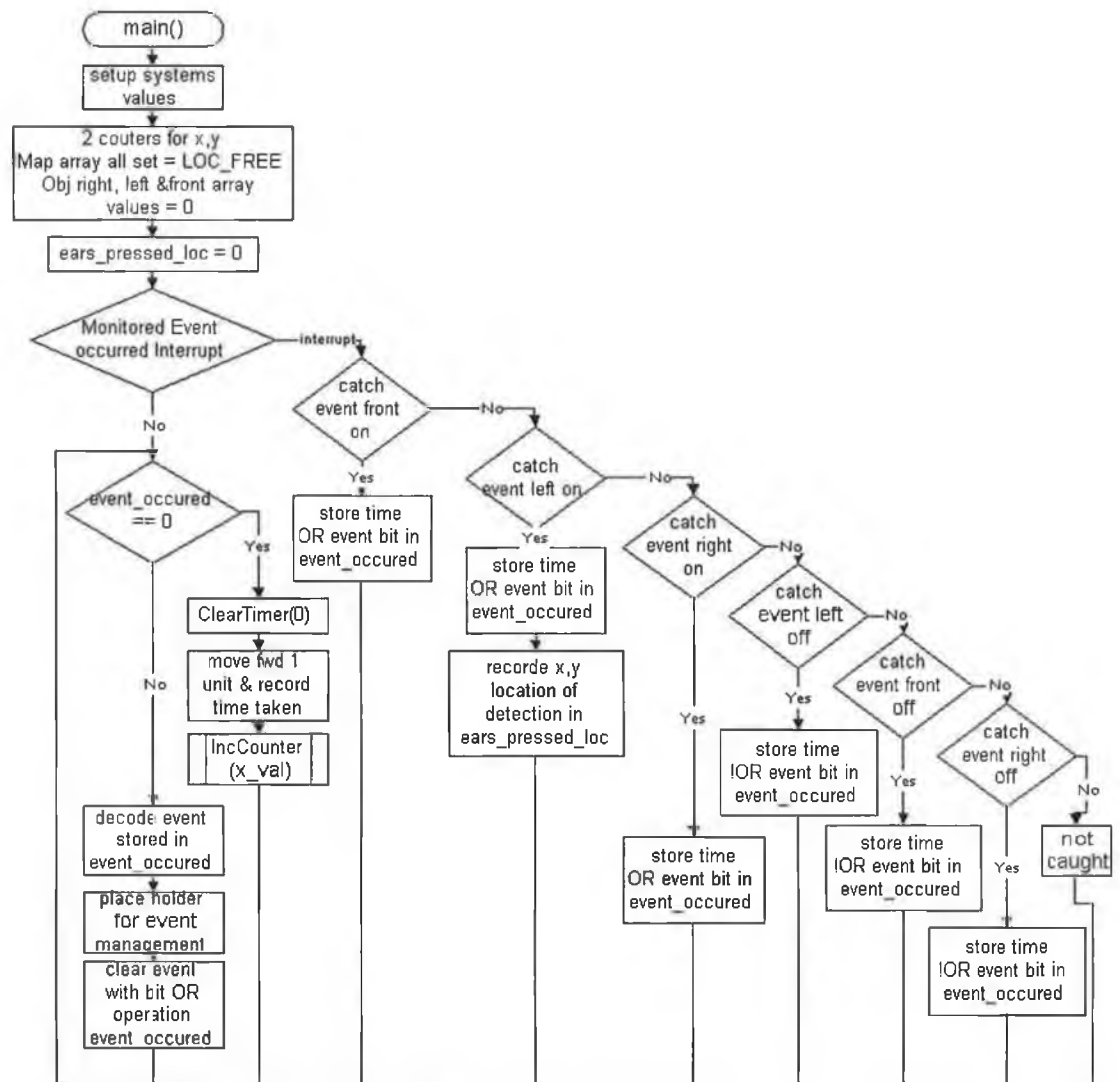


Figure 61 motors\_navigation\_single\_array Partial Flowchart

- `memory_test.nqc`: This programme was used in the testing of memory assignment in the RCX hardware platform using the NQC programming environment. The outputs of which are included in the file "out.txt". In Table 9 the assigned storage locations for each variable is listed, as it is generated by NQC output. The programme tests a number of approaches to memory assignment form using the "#pragma" to reserve 11 storage locations. From storage location 0 to 10. The programme then assigns names to 17 unique Integers Var 34 – Var 47 and Var 11. It also creates an array of Integers of size 19. This is stored in locations 12-31. An integer pointer is also created and stored at location Var 32. It identified that the stated limitation of 32

variables [48] could be somewhat overcome, using arrays and #pragma directive and the use of pointers to access this reserved data.

**Table 9 NQC Variable Storage**

NQC code syntax	Code generates by NQC
#pragma reserve 0 10	
int display	*** Var 11 = display
int i1	*** Var 47 = i1
int i2	*** Var 46 = i2
int i3	*** Var 45 = i3
int i4	*** Var 44 = i4
int i5	*** Var 43 = i5
int i6	*** Var 42 = i6
int i7	*** Var 41 = i7
int i8	*** Var 40 = i8
int i9	*** Var 39 = i9
int i10	*** Var 38 = i10
int i11	*** Var 37 = i11
int i12	*** Var 36 = i12
int i13	*** Var 35 = i13
int i14	*** Var 34 = i14
int map[19]	*** Var 12 = map
int i	*** Var 33 = i
int* ptr	*** Var 32 = ptr

## Appendix B - Simulation Code

### B.1. Olivier Michel's Khepera Simulator - KSim

The following code was produced as part of the investigation in the project.

user.c: This code is a modified version of the example code supplied with the simulator. The simulation environment provides the programmer with a “user info box” [61], shown in Figure 63. This area allows the programmer, to graphically represent information to the user, utilising specific functions provided by the simulation environment. This code was used in the evaluation process of this simulation environment. The edited code's resultant display is shown in Figure 62, where outputs are drawn in the user's info box displaying results. A partial flowchart of the code in Figure 64 illustrates that sensors input information is read in the controller. This value is compared against a predefined level assigned the name “COLLISION\_TH”. If the sensor value is above this level the programme draws a coloured arc indicating that an object is in close proximity to the particular sensor. If the value is less than the predefined limit a grey arc is drawn. The arcs are drawn relative to the physical sensor's location on the robot. This generates a graphical display for the user of the sensory information, as individual sensory inputs become greater than a threshold level. The controller assesses this sensory information and changes the trajectory of the robot accordingly.

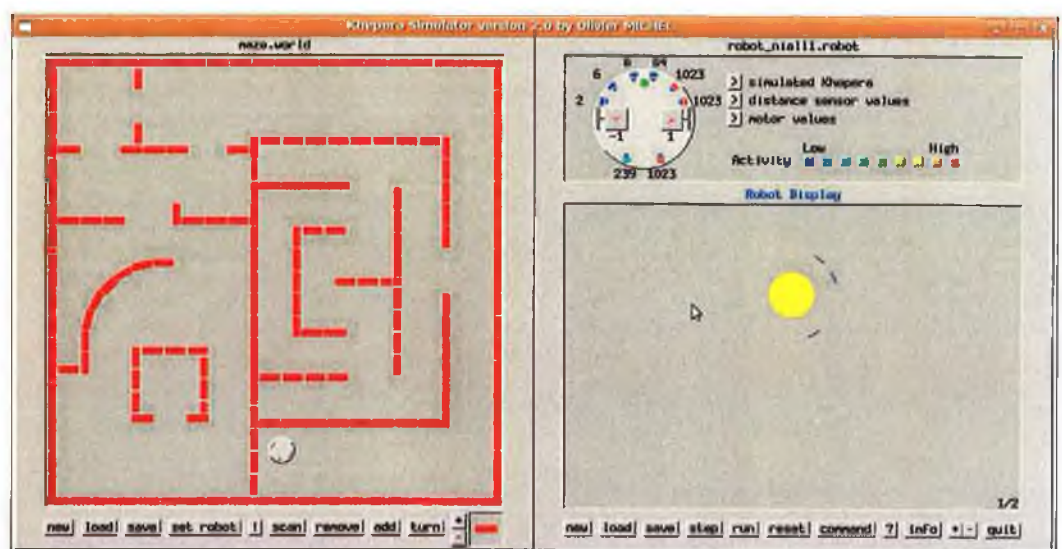


Figure 62 Khepera Simulator Maze environment

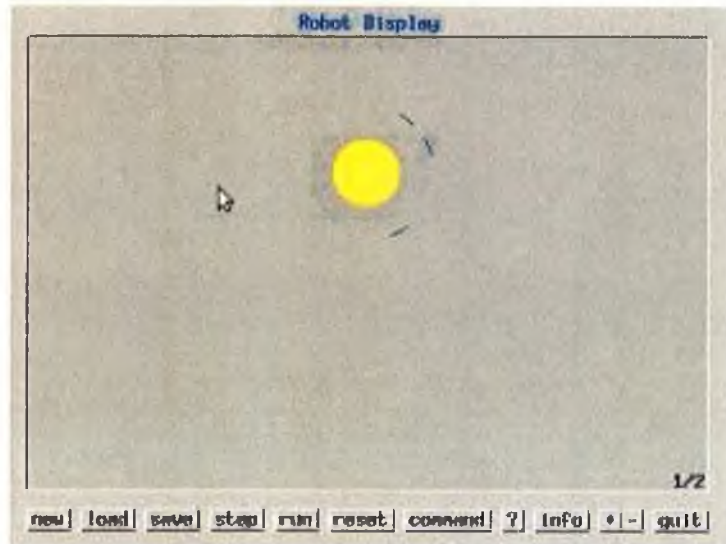


Figure 63 User's info box

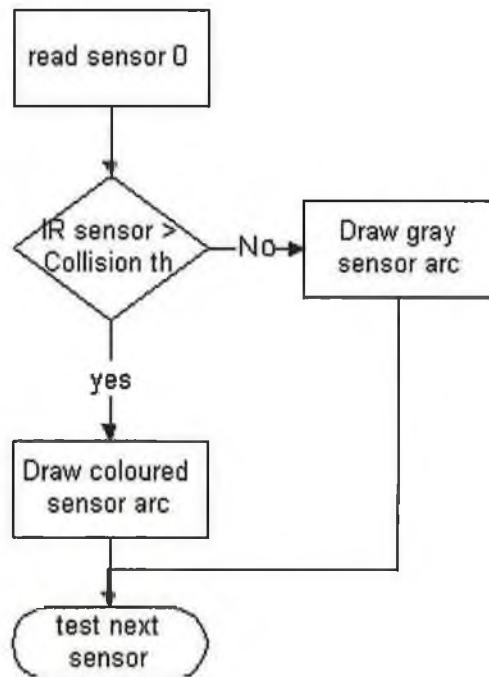


Figure 64 Flow Chart DrawSurrond

## B.2. Webots

The UML diagram for the complete controller is provided in Figure 65 and continued in Figure 66. The individual files for the controller are as follows:

**Piglet.java:** This is the main code of the controller it starts the threads and sensors, it also detects the feeders.



**Planner.java:** This is the thread that plans the path in the grid using harmonic potential fields

**Position.java:** computes the position from the encoders data

**Output.java:** This is the thread that outputs the plans, maps and feeders

**Landmark.java:** The data base of feeders used

**IntPoint.java:** Used in the planning of a path in 2d

**Image.java:** Is used for thread synchronization

**Grid.java:** This is the thread that constructs occupancy grids

**Driver.java:** This deals with the robot state and defines what action will be done.

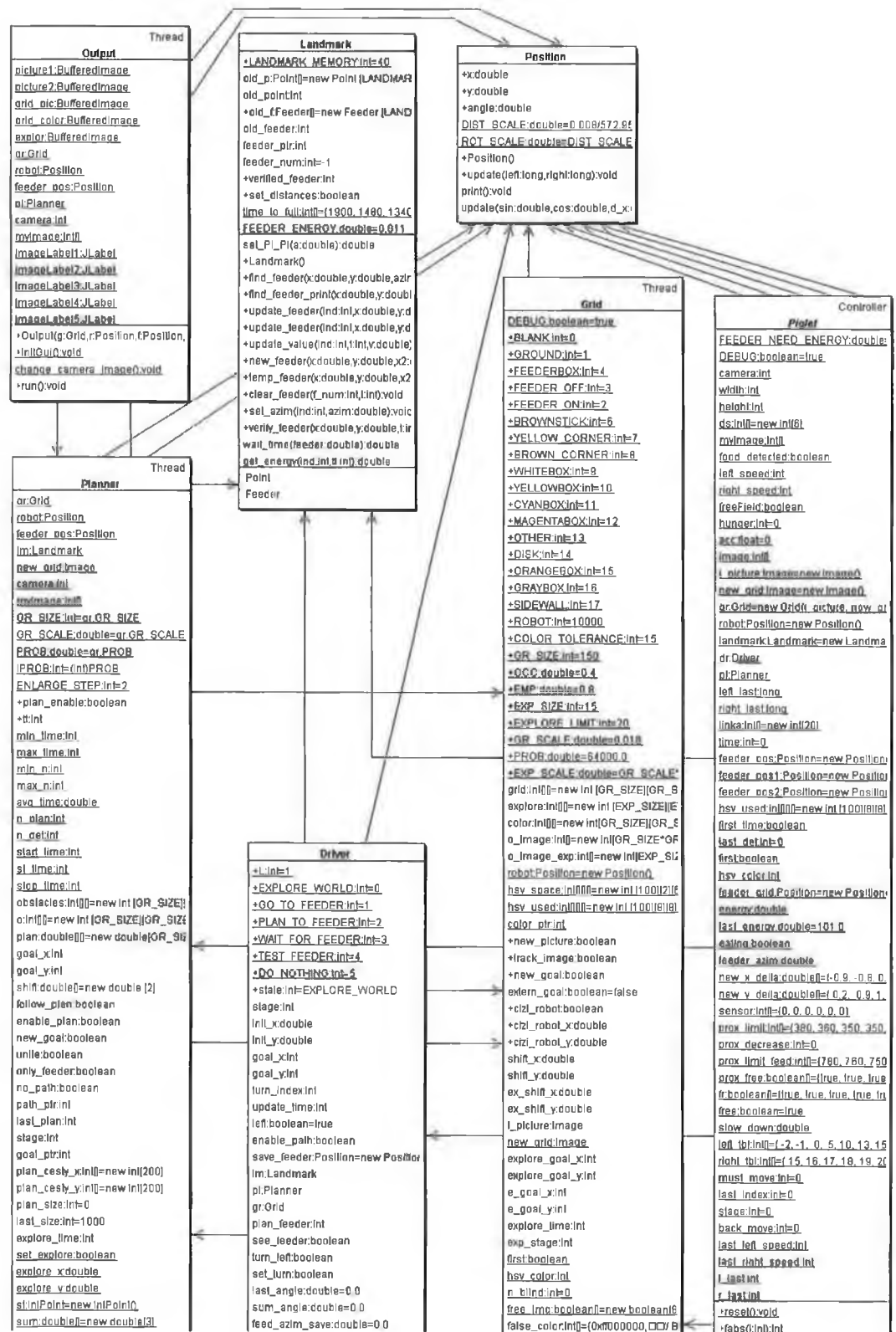


Figure 65 Piglet Controller UML part A



Figure 66 Piglet Controller UML part B



Figure 67 Mapped Environment



Figure 68 Output of Simulated Camera



Figure 69 False Colour Camera

**B.2.1. Webots World File**

Partial world file stub, complete world file included on CD.

“pig\_edit\_v3\_controller\_maze\_low\_consumption.wbt”

```

#VRML_SIM V4.0 utf8
#000000
.....
DEF SURROUNDING_WALL Solid {
  children [
    Shape {
      ....
    }
  ]
  name "wall"
  boundingObject Group {
    ....
  }
}
.....
DEF ROBOT_0 DifferentialWheels {
  ....
  children [
    ....
    Solid {
      ....
      children [
        DEF WHEEL Transform {
          ....
        }
      ]
      name "left wheel"
    }
    Solid {
      ....
      children [
        DEF WHEEL Transform {
          ....
        }
      ]
      name "right wheel"
    }
  ]
  ....
  DistanceSensor {
    ...
    children [
      DEF SFH900 Group {
        ....
      }
    ]
    name "ds0"
    lookupTable [
      0 1023 0
      0.05 0 0.01
    ]
  }
  ...
  Camera {
    ...
    children [
      Shape {
        ....
      }
    ]
    name "camera"
  }
}

```

```

    ****
  }
]
name "robot 0"
boundingObject Transform {
  ****
}
controller "Piglet_edit_v3"
synchronization FALSE
battery [
  92.7796, 100, 10
]
cpuConsumption 0.1
motorConsumption 0.01
axleLength 0.053
wheelRadius 0.008
maxSpeed 20
maxAcceleration 200
speedUnit 1
encoderNoise 0.01
encoderResolution 572.958
}

```

### B.3. Erdos

The author's original code occurs first and then the adapted code:

(simulator) main.py: The code in this file provides the user with a simulated Roomba robot in a maze environment.

Edited code

(simulator) main2.py: This code is adapted for the original code with the addition of the search provided in the Erdos' interface with a real Roomba robot.

### B.4. Pyro

Software implementation for this simulator started with the behavioural procedure called "Avoid.py", from the included sample examples for the Pyro controller.

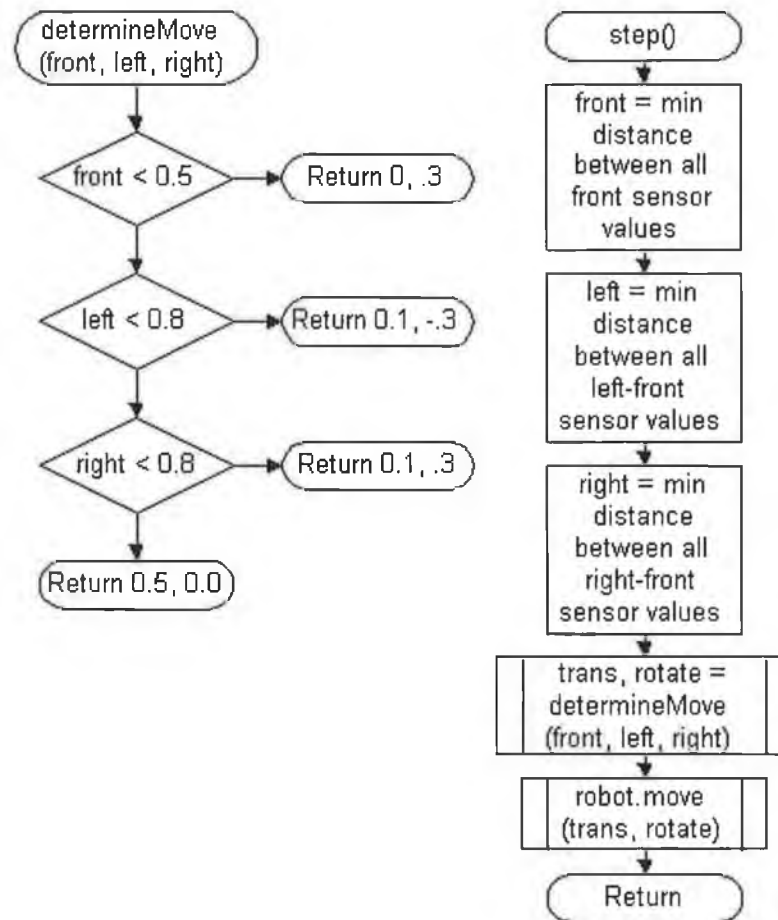


Figure 70 Avoid Flowchart

Avoid.py: Performed the task of requesting all the information for the robots variables front, left and right sensors assigning the smallest value to the appropriate variable Figure 70. The controller then altered its trajectory by the translation and rotation values calculated by its “determineMove()” function.

## Appendix C - Java

### Java Background

Java code is an interpretative language which means it compiles to produce byte code. This byte code requires an interpreter to run on a given system. This enables java code to be written once and run on all different types of system, given that they have an appropriate interpreter written for them. This approach to programming has huge advantage over the standard system, of having the compile a different version of each piece of code written in other languages for each platform are required to run on. The Java byte code requires that the system has one piece of code (the interpreter) running on it to allow the java byte code to run. This piece of code is called the Java Virtual Machine and it interprets the byte code and runs the corresponding local instruction on the host system to perform the required task. E.g. it prints a message to an LCD panel or a Computer Screen which ever is the current system's display.

## Appendix D - Pyro Setup

- The Numeric package has been superseded by the numpy package so this is required to be installed in python directory. Former references to “import Numeric” should then be converted to “import numpy.oldnumeric as Numeric”. This removes the output error “ImportError : No module named Numeric”
- Both \pyrobot\bin\pyrobot and \pyrobot\bin\pyrobot.py should be identical and should be edited in the user intends running Pyro under a windows environment. Here are at required modifications to remove error message about Pythonpath:
  - Line 47 change from “/plugins” to “\plugins”,
  - Line 76 if Python directory is on E:\ drive change the following from “:%s/plugins/simulators” to “;%s\plugins\simulators;e\Python24”
- \pyrobot\gui\\_\_init\_\_.py: must similarly be changed is the user is running python on a windows box with the following modifications:
  - Line 131 137 143 149 159 164 169 174 change “/plugins” to “\plugins\”
  - Line 377 378 change “/” to “\”
  - Line 471 change “%s/plugins/config/%s/” to “%s\plugins\config\%s\”
  - Line 482 487 492 change “/%s/plugins/worlds/%s/” to “\%s\plugins\worlds\%s\”
- Epydoc: Was utilised to construct the UML diagrams for the Python code. It required for error free operation that “getint = int”



be commented out in Line 378 of Lib\Lib-tk\Tkinter.py in python's directory.

## Glossary

**AI:**

Artificial Intelligence

**API:**

Application Programming Interface, a software interface that defines how a software programme requests services from a device.

**Coarse Grained Simulation:**

This is a platform which allows the development of a controller prior the testing on a real robot. The simulated sensory values are a rough approximate of the expected real world values. Further tweaking of the controller will be required when run in the real robot.

**DLLs:**

Dynamic Linked Libraries

**IR:**

Infra-red

**Kidnapped:**

This is where a robot which has localised itself, is picked up and placed in an unknown location, and it is required to localise its self again.

**LAMI:**

Microcomputer Laboratory (LAMI) of Swiss Federal Institute of Technology of Lausanne

**LeJOS:**

Java for LEGO® Mindstorms, Java Language to programme the RCX module

**JDK:**

Java Development Kit

## Glossary

**JVM:**

Java Virtual Machine

**NQC:**

Not Quite C. Programming language designed for programming the RCX module.

**RCX:**

LEGO Programmable brick module, part of the LEGO Robotic Invention System

**ROI:**

Roomba Open Interface

**NQC:**

Not Quite C

**OS:**

Operating System

**SCI:**

Serial Command Interface

**UML:**

Unified Modelling language, an object modelling and specification language used in software engineering

## **Poster Publications**

"Autonomous Robotic Systems: Using entropy theory to minimise redundancy in Autonomous Robotic Systems" Niall McCurry, Dr John Owen-Jones, Paul Dunne. Published in National Symposium of The Irish Research Council for Science, Engineering and Technology, Dublin 3rd November, 2005

"Autonomous Robotic System" Niall McCurry, Dr John Owen-Jones, Paul Dunne. Published in National Symposium of The Irish Research Council for Science, Engineering and Technology, Dublin 2nd November, 2004, pp. 197

## Bibliography

1. DARPA. *The DARPA Grand Challenge Commemorative Program*. 2004. Barstow, California - Primm, Nevada: DARPA. p.
2. DARPA, *2005 DARPA Grand Challenge!* (<http://www.darpa.mil/grandchallenge>) 2005
3. DARPA, *DARPA Announces Third Grand Challenge Urban Challenge Moves to the City:2006*
4. Shannon, C., *Robots can Wear Multiple Hats in the Computer Science Curriculum at Liberal Arts Colleges*. AAAI, 2007
5. Gutmann, J.-S., et al. *Reliable Self-Localization, Multirobot Sensor Integration, Accurate Path-Planning and Basic Soccer Skills: Playing an Effective Game of Robotic Soccer*. in *Ninth International Conference on Advanced Robotics ICAR 99*. 1999: Japan Robot Association. p. 289-296
6. MobileRobots, *Pioneer PatrolBot PowerBot Price List:18 Dec 2009* (<http://www8.cs.umu.se/research/ifor/dl/Product%20info/ActiveMedia/ActiveMedia%20Jan14EduPriceList.pdf>) 2009 MobileRobots Inc
7. MobileRobots, *AmigoBot Price List:18 Dec 2009* (<http://www8.cs.umu.se/research/ifor/dl/Product%20info/ActiveMedia/ActiveMedia%20pricelist%20june%202002.pdf>) 2009 MobileRobots
8. RoadNarrows, *Road Narrows Robotics:28 Dec 2009* (<http://www.roadnarrows.com/robotics/store/khepera-ii.html>) 2009
9. Burhans, D.T., *A Robotics Introduction to Computer Science*. AAAI, 2007
10. Imberman, S., A. Barkan, and E. Sklar, *Extra-curricular Robotics: Entry-level Soccer for Undergraduates*. AAAI, 2007. (<http://www.aaai.org/Papers/Symposia/Spring/2007/SS-07-09/SS07-09-013.pdf>)
11. Juliano, B.A. and R.S. Renner, *An Undergraduate Course in Robotics and Machine Intelligence*. American Association for Artificial Intelligence, 2006. (<http://www.aaai.org/Papers/Symposia/Spring/2007/SS-07-09/SS07-09-016.pdf>)
12. Sklar, E., S. Parsons, and M.Q. Azhar, *Robotics Across the Curriculum*. American Association for Artificial Intelligence, 2006. ([http://www.cs.hmc.edu/roboteducation/papers2007/c46\\_sklar\\_rac.pdf](http://www.cs.hmc.edu/roboteducation/papers2007/c46_sklar_rac.pdf))
13. Sullins, J.P., *Using Robotic Competitions in Undergraduate Philosophy Courses: Studying the Mind Through Simple Robotics*. American Association for Artificial Intelligence, 2002
14. Jacobsen, C.L. and M.C. Jadud. *Towards Concrete Concurrency: occampi on the LEGO Mindstorms*. in *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*. 2005. New York: ACM Press. p. 431-435
15. McKerrow, P.J. *Robotics research and education with LabVIEW*. in *AUC Conference*. 2000. University of Wollongong, New South Wales, Australia. p. ([http://auc.uow.edu.au/conf/conf00/papers/AUC2000\\_McKerrow.pdf](http://auc.uow.edu.au/conf/conf00/papers/AUC2000_McKerrow.pdf))
16. Tripathy, H.K., B.K.Tripathy, and P.K. Das, *A Prospective Fuzzy Logic approach to Knowledge-based Navigation of Mobile LEGO-Robot*. Journal of

- Convergence Information Technology, 2008. **Vol. 3**(No 1): p. 64-70.  
(<http://www.aicit.org/jcit/papers/jcit3-1/8-jcit3-1.pdf>)
17. Vento, A., C. Beltrán, and E. Taniuchi, *A quantitative analysis of robotic languages*. Journal of Computing Sciences in Colleges Volume, 2002. 17(5): p. 72 - 80
  18. BUSCHMANN, C., F. MÜLLER, and S. FISCHER. *Grid-Based Navigation for Autonomous, Mobile Robots*. in *Proceedings of 1st Workshop on Positioning, Navigation and Communication 2004 (WPNC'04)*. 2004: Shaker Publishing. p. ([http://www.ibr.cs.tu-bs.de/papers/Buschmann\\_etal\\_GridBasedNavigation.pdf](http://www.ibr.cs.tu-bs.de/papers/Buschmann_etal_GridBasedNavigation.pdf))
  19. Bosch, S., M. Hendriks, and R. Leemkuil, *F.R.O.P.S Fragile Reconnaissance Object Processing System*. 2004, Universiteit Twente de ondernemende universiteit
  20. Martin, F., *The 6.270 Robot Builder's Guide*. 1992.  
(<http://cherupakha.media.mit.edu>)
  21. Martin, F.G., *The Art of LEGO Design*. The Journal for Robot Builders, 1995. 1(2)
  22. Bennett, J.K., *Introduction to Engineering Design ELEC 201 Course Notes*. 1998
  23. Proudfoot, K., *RCX Internals:Feb 2004*  
(<http://graphics.stanford.edu/~kekoa/rcx/>) 1999
  24. Anders, *RCX Scanner:05 August 2009*  
(<http://www.norgesgade14.dk/scanner.php>) 2009
  25. Brown, J., *Cube Solver:06 August 2009*  
(<http://jpbrown.i8.com/cubesolver.html>) 2001
  26. Juliano, B.A. and R.S. Renner, *LEGO Mindstorms RIS 2.0 Programming: Lego RCX Code*. 2004.  
(<http://www.ecst.csuchico.edu/~juliano/csci224/Slides/04%20-%20Intro%20Lego%20RCX%20Code.pdf>)
  27. iRobot, *Company Milestones:19 August 2009*  
(<http://www.irobot.com/sp.cfm?pageid=203>) 2009 [www.iRobot.com](http://www.irobot.com)
  28. Kurt, T.E., *Hacking Roomba*. 2006: Wiley. 456.
  29. iRobot. *Developers Choose iRobot Platforms To Create Robot Applications*. in *RoboBusiness Conference and Exposition*. 2006. PITTSBURGH. p.  
(<http://www.irobot.com/sp.cfm?pageid=86&id=238>)
  30. iRobot, *iRobot Roomba Open Interface (ROI) Specification:*  
([www.irobot.com](http://www.irobot.com)) 2005
  31. iRobot, *iRobot Roomba Serial Command Interface (SCI) Specification:*  
([www.irobot.com](http://www.irobot.com)) 2005 iRobot
  32. Dickenson, B., et al., *Roomba Pac-Man: Teaching Autonomous Robotics through Embodied Gaming*. American Association for Artificial Intelligence, 2006
  33. Kurt, T.E., *Low-cost On-board Linux, Vision, Wi-Fi and more for the Roomba Robotics Base*. American Association for Artificial Intelligence, 2006.  
(<http://hackingroomba.com/wp-content/uploads/2007/01/fs09kurtt.pdf>)
  34. Matarić, M.J., N. Koenig, and D. Feil-Seifer, *Materials for Enabling Hands-On Robotics and STEM Education*. American Association for Artificial Intelligence, 2006. ([http://cres.usc.edu/pubdb\\_html/files\\_upload/536.pdf](http://cres.usc.edu/pubdb_html/files_upload/536.pdf))

35. Tribelhorn, B. and Z. Dodds, *Envisioning the Roomba as AI Resource: A Classroom and Laboratory Evaluation*. American Association for Artificial Intelligence, 2006
36. Tribelhorn, B. and Z. Dodds. *Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education*. in *2007 IEEE International Conference on Robotics and Automation*. 2007. Roma, Italy. p. 1393-1399. (<http://dx.doi.org/10.1109/ROBOT.2007.363179>)
37. iRobot, *iRobot® Create Owner's guide*:19 August 2009 ([www.irobot.com/support](http://www.irobot.com/support)) 2006 iRobot Corporation
38. iRobot, *iRobot Unveils Programmable Robot*:19 August 2009 (<http://www.irobot.com/sp.cfm?pageid=86&id=282&referrer=28>) 2007
39. iRobot, *iRobot® Create Open Interface*:19 August 2009 ([http://www.irobot.com/filelibrary/create/Create%20Open%20Interface\\_v2.pdf](http://www.irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf)) 2007 iRobot
40. Conbere, M. and Z. Dodds, *Toys and Tools: Accessible Robotics via Laptop Computers*. American Association for Artificial Intelligence, 2007
41. Dodds, Z. and B. Tribelhorn, *Erdos: Cost-effective Peripheral Robotics for AI Education*: (<http://www.cs.hmc.edu/~dodds/erdos/erdos.pdf>) 2006 American Association for Artificial Intelligence [www.aaai.org](http://www.aaai.org)
42. Solórzano, J., *LeJOS 1.0.0 alpha1*. 2000. (<http://sourceforge.net/projects/lejos/files/>)
43. Andrews, P., et al., *LEJOS Java for LEGO Mindstorms*:28 October 2009 (<http://lejos.sourceforge.net/>) 2009
44. Bagnall, B., *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform*. 2002: Prentice Hall PTR.
45. Ferrari, M., G. Ferrari, and R.H.T. Editor, *Building Robots with LEGO MINDSTORMS The ULTIMATE Tool for MINDSTORMS Maniacs*. 2002: Syngress Publishing.
46. Laverde, D., G. Ferrari, and J. Stuber, *Programming Lego Mindstorms with Java*. 1 ed. 2002: Syngress.
47. Irish\_Emigrant\_Publications, *GMIT student designs problem-solver*:05 October 2009 ([http://www.emigrant.ie/index.php?option=com\\_content&task=view&id=17864](http://www.emigrant.ie/index.php?option=com_content&task=view&id=17864)) 2004
48. Baum, D., *NQC Programmer's Guide*, "Ed:2.5a4:05 May 2004 (<http://www.baumfamily.org/nqc>) 2003
49. Harris, W. and D. Arnow, *Remote Shared Access To A Classroom Robotics Lab*. AAAI, 2007. ([http://www.cs.hmc.edu/roboteducation/papers2007/c28\\_HarrisArnow.pdf](http://www.cs.hmc.edu/roboteducation/papers2007/c28_HarrisArnow.pdf))
50. Juliano, B.A. and R.S. Renner, *LEGO Mindstorms RIS 2.0 Programming: NQC Code*. 2004. (<http://www.ecst.csuchico.edu/~juliano/csci224/Slides/05%20-%20Intro%20NQC%20Code.pdf>)
51. Mondada, F., E. Franzi, and A. Guignard, *The Development of Khepera*, in *First International Khepera Workshop*. 1999, HNI-Verlagsschriftenreihe, Heinz Nixdorf Institut: Paderborn. p. 7--14. (<http://infoscience.epfl.ch/getfile.py?recid=89709&mode=best>)
52. Hellström, T., *Khepera*. 2006. (<http://www8.cs.umu.se/kurser/TDBD17/VT07/utdelat/kinematics.pdf>)
53. K-Team, *Khepera User Manual*, "Ed:Version 5.02:1999

54. Lindqvist, F., *Selfsupervised learning for a miniature robot*: (<http://www.cs.umu.se/education/examina/Rapporter/470.pdf>) 2003
55. Rice, K.A.K.a.J.L. *The Khepera Robot as a Teaching Tool*. in *American Society for Engineering Education (ASEE)*. 1999. Charlotte, North Carolina. p.
56. Keeratipranon, N., J. Sitte, and F. Maire, *Beginners Guide to Khepera Robot Soccer*. 2003, Brisbane.
57. Mondada, F., *The Khepera Miniature Mobile Robot*:15 August 2009 (<http://diwww.epfl.ch/lami/robots/K-family/Khepera.html>) 1996
58. K-Team, *The K-Team*. 2005, <http://www.k-team.com/>. (<http://www.k-team.com/>)
59. K-Team, *K-Team USERS & APPLICATIONS*:14 August 2009 (<http://www.innowebtive.com/kteam/resources/users.html>) 2002
60. Trianni, V., *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots*. 1 ed. Studies in Computational Intelligence. 2008, Verlag Berlin and Heidelberg GmbH & Co. KG: Springer. 190.
61. Michel, O., *Khepera Simulator version 2.0 User Manual*:01 November 2004 (<http://diwww.epfl.ch/lami/team/michel/khep-sim/>) 1996
62. K-team, *ROBOT FREEWARES*:november 2004 (<http://www.k-team.com/kteam/index.php?page=158&rub=&site=1>) 2004
63. Perretta, S., et al., *WSU Khepera Robot Simulator*, "Ed:Version 7.1: (<http://ehrg.cs.wright.edu/ksim/downloads/downloads.html>) 2003
64. Nolfi, S., *Evorobot 1.1 User Manual*. 2001. (<http://laral.istc.cnr.it/evorobot/simulator.html>)
65. GNU, *What is Copyleft?*17 August 2009 (<http://www.gnu.org/copyleft/>) 2009
66. Medsker, L. and L.C. Jain, *Recurrent Neural Networks: Design and Applications (International Series on Computational Intelligence)*. 1 ed. 1999: CRC Press. 416.
67. Carlsson, J., *YAKS*. 2000. (<http://r2d2.ida.his.se/>)
68. Skånhaug, S.-R., *Architectural aspects of software for mobile robot systems*, in *FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK*. 2003, NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET. (<http://www.idi.ntnu.no/grupper/su/su-diploma-2003/thesis-skaanhaug.pdf>)
69. Iwe, H., et al., *Khepera Project Dresden*:10 December 2004 (<http://easybot.htw-dresden.de/>) 2000
70. Michel, O., *LightVision3D*. 1999. (<http://easybot.htw-dresden.de/Download/software/genalg.pdf>)
71. Iwe, H., *Easybot*:18 August 2009 (<http://easybot.htw-dresden.de/>) 2007
72. Cyberbotics, *Webots User Guide*, "Ed:4.0.27: ([www.cyberbotics.com](http://www.cyberbotics.com)) 2004 Cyberbotics Ltd.
73. Cyberbotics, *Webots Reference Manual*: ([www.cyberbotics.com](http://www.cyberbotics.com)) 2004
74. Szabó, R. *Navigation of simulated mobile robots in the Webots environment*. in *The Third Conference of PhD Students in Computer Science*. 2002. Szeged, Hungary. p. 149-163. (<http://www.jataka.hu/rics/cscs2002/cscs2002.pdf>)
75. Szabó, R. *Combining metric and topological navigation of simulated robots*. in *The Fourth Conference of PhD Students in Computer Science*. 2004. Szeged, Hungary. p. (<http://www.jataka.hu/rics/cscs2004/cscs2004.pdf>)
76. Cyberbotics, *ALife contest May 1st, 2003*:11 February 2005 (<http://cyberboticspc1.epfl.ch/contest/previous.html>) 2003



77. Blank, D., et al., *The Pyro toolkit for AI and robotics*, in *AI Magazine*. 2005. p. pp 39-50
78. Blank, D., et al., *Advanced Robotics Projects for Undergraduate Students*. American Association for Artificial Intelligence, 2006
79. Blank, D., et al., *Advanced Robotics Projects for Undergraduate Students*. American Association for Artificial Intelligence, 2007
80. Tang, F., *Enhance Students' Hands-On Experience With Robotics*. American Association for Artificial Intelligence, 2006
81. Conroy, K., *Constructopedia*. 2004.  
([http://www.educatec.ch/ressources/Literature/LEGO\\_MINDSTORMS/MINDSTORMSBAUANLEITUNGEN/constructopedia-rcx/file\\_en/](http://www.educatec.ch/ressources/Literature/LEGO_MINDSTORMS/MINDSTORMSBAUANLEITUNGEN/constructopedia-rcx/file_en/))
82. Overmars, M., *Drive-Steer mechanism*:05 May 2004  
(<http://people.cs.uu.nl/markov/lego/tips/Differential1/index.html>) 2004
83. GNU, *GNU GENERAL PUBLIC LICENSE*, "Ed:Version 3:17 August 2009  
(<http://www.gnu.org/copyleft/gpl.html>) 2007
84. Fossum, T. and J. Snow, *How Platform-Independent is Pyro?* American Association for Artificial Intelligence [www.aaii.org](http://www.aaii.org), 2006
85. Klassner, F. and M. McNally, *Demonstrating the Capabilities of MindStorms NXT for the AI Curriculum*. American Association for Artificial Intelligence, 2007
86. Watson, M., *Practical Artificial Intelligence Programming in Java*. 2004.  
([www.markwatson.com](http://www.markwatson.com))