A COMPARISON OF THREE ROBOTIC CONTROLLERS FOR NAVIGATION

Justin Matulich

A thesis submitted to AUT University
in fulfilment of the requirements for the degree of
Masters of Engineering (ME)

2017

School of Engineering
Primary Supervisor: Dr Mark Beckerleg
Secondary Supervisor: Dr John Collins

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institution of higher learning.

Signed [illegible]

Acknowledgement

I would like to acknowledge my supervisor Dr Mark Beckerleg for his continued support and wisdom throughout the duration of this research. Laura Wolken and Katrina Agnew for their very valued contributions. My wife Marise Matulich and family for your unwavering support; I would not have achieved what I have without you.

Abstract

This research provides a comparison of three types of robotic controllers and their suitability in evolutionary robotics. Two novel systems comprised of lookup tables (LUTs) and evolvable hardware (EHW) based controllers are compared against a benchmark single layered artificial neural network (ANN). The controllers have been evolved using a genetic algorithm (GA) to perform the following robotic navigational tasks: light following, object avoidance and the combined behaviour, light following while avoiding obstacles. Five aspects of the evolved robot controllers are evaluated: a) controller performance, assessed both numerically and visually; b) evolutionary efficiency, the number of generations required to obtain a good fitness; c) scalability, based on the controller performance and evolutionary efficiency as the complexity of the task is increased; d) quantization effects as the sampled resolution of the input sensors is varied; and e) operation of the evolved controllers in unknown spaces.

The findings from this research shows that: 1) the evolved controller performance is similar between the LUT, EHW and ANN controllers; 2) the evolutionary efficiency of the ANN and EHW are comparable, whereas the LUT took four times the number of generations to evolve; 3) the scalability of the EHW and ANN controllers were similar with the LUT being the most affected taking twelve times the number of generations to evolve; 4) the quantization effects of the sensors was comparable for all three controller types with a low sensor resolution mostly having an effect as the controller performance was moving towards a maximum; and 5) all the controllers were more robust in unknown environments when evolved in multiple arenas.

Both the EHW and LUT controllers performed far better than the apparent search space would suggest. This was due to the EHW having a large number of possible circuit solutions, effectively allowing solutions to be found quickly, and the LUT requiring only small sections of the LUT to control the robot thereby reducing the GA search space.

The selection of which controller to use is determined by the system that it will be used in. The ANN is suited to a processor that contains a floating-point unit, the EHW is suited to a hybrid field programmable gate array (FPGA) with an ARM-based hard-core processor, whereas the LUT is suited to a low cost 8-bit microcontroller based system.

Table of Contents

Table of Figures

Tables

Chapter 1

Chapter 1: Introduction

Evolutionary robotics has been widely researched for autonomous robots due to its adaptability in unknown environments and fault tolerance. A common evolvable controller that has been extensively studied in this field is the ANN, where the weights in the network are evolved rather than training the network. Other less studied evolvable robotic controllers are the LUT where the chromosome is the parameters in the table and EHW where the chromosome is the configuration bit-stream (CBS), which is used to create the circuit. This research provides a comparison between these controllers and a standard feed-forward single layer ANN. The controllers have been evolved to perform three navigational tasks, light following, obstacle avoidance and a combination of the two, light following while avoiding obstacles. A comparison has been made which investigated five key aspects: 1) the performance of the evolved controller (controller performance); 2) the number of generations required to reach this performance (evolutionary efficiency); 3) the effects of quantization on the input sensors; 4) the effects of scalability as the controller becomes more complex; and 5) the effects of altering the environment after the robot controller has been evolved. The effects of scalability for the three controller platforms are assessed using two methods: monolithic, where complex behaviours are evolved concurrently; and subsumption, where the behaviours are evolved individually, then combined using a switching controller.

1.1 Artificial neural network

Widely used in robotics, the ANN is a system of interconnected neurons (Figure 1-1) which imitates a biological neural network. A simple single layer ANN used in robotics is comprised of three key sections: 1) inputs from the sensors are adjusted via a weighting factor and passed onto the neuron; 2) the neuron which sums the weighted input signals and feeds the combined signals into an activation function; and 3) the output from the neuron is used to drive an actuator. The ANN configuration used in this research is described in section 4.2.1.

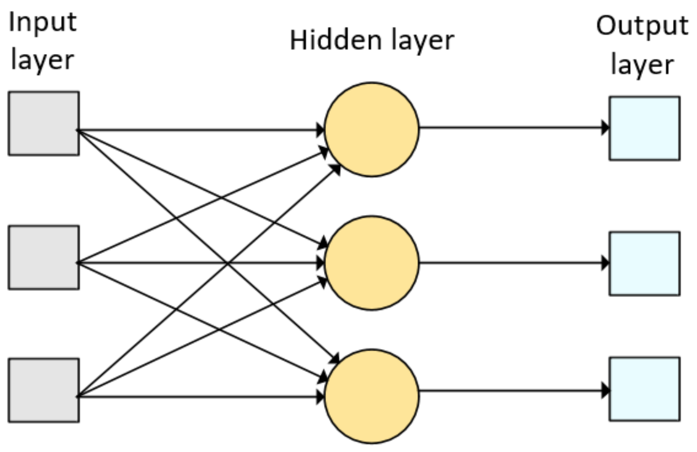


Figure 1-1 Single-layered ANN

1.2 Lookup Table

Used widely in low powered microprocessors, the LUT provides a means of reducing computational runtime. The LUT uses a simple index based system to provide a path to a previously calculated result. LUTs are configured in a variety of formats depending on the complexity of the computational task. Table 1 shows a typical LUT which could be used to calculate the values of sine for a given angle. The inputs to the LUT in this example are the angles (θ) and the outputs (sin(θ)) are contained in the table elements.

Table 1 LUT example (Sine wave)

θ	0	30	60	90	120	150	180	210	240	270	300	330
Sin(θ)	0	0.5	0.87	1	0.87	0.5	0	-0.5	-0.9	-1	-0.9	-0.5

LUTs when used as robotic controllers are restricted by the quantization of the inputs, outputs, and scalability difficulties in complex systems. However, once evolved, the LUT requires far fewer computational resources than that of an ANN and other traditional control systems. This makes it ideal for implementation in processors with low computational power such as the commonly used 8-bit microcontroller. The LUT based controller could be used in a distributed control system. This would have a central processor to perform the evolutionary computation for multiple local LUT based microcontrollers. Continuous evolution of these multiple controllers, allows the system to adapt to fault conditions by updating the LUT in each microcontroller (Figure 1-2)

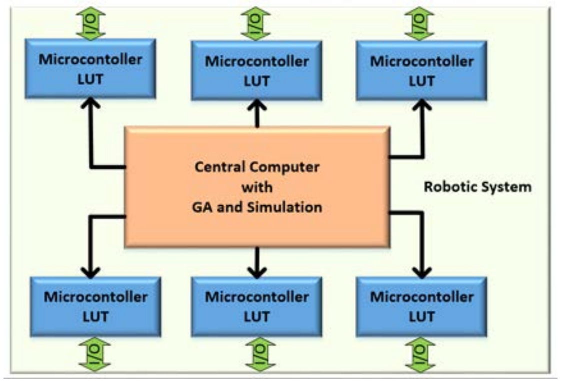


Figure 1-2 An adaptable robotic system with a central processor for the genetic algorithm process and multiple LUT based microcontrollers for control.

1.3 Evolvable hardware

EHW, sometimes referred to as a virtual-FPGA or a virtual reconfigurable circuit (VRC) is comprised of an adjustable interlinked system of logic elements (Figure 1-3). Each logic element performs logic manipulation of its inputs and passes the resulting output to the subsequent connected logic element. The logic element and interconnections is reconfigurable via a CBS which is used to describe the logic hardware.

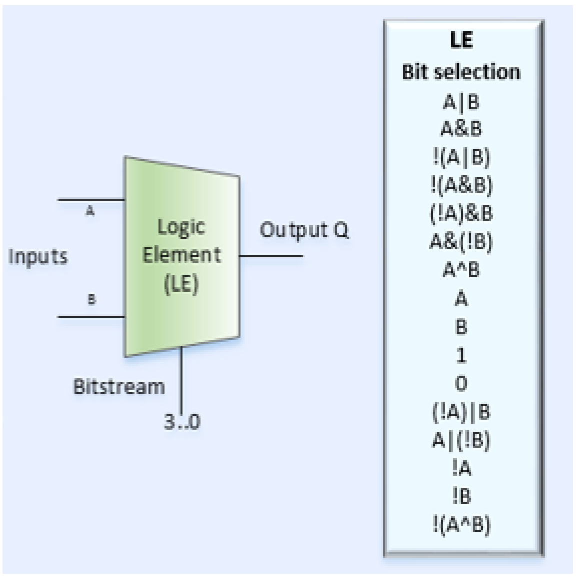


Figure 1-3 EHW logic element configuration

The use of EHW in the form of a virtual-FPGA or VRC has been studied for image processing and fault tolerance, but very little research has been undertaken into their use as robotic controllers. For this research, a flat-layered virtual-FPGA has been developed and provides the platform for the EHW controllers. The evolved chromosome is comprised of a CBS which is used to describe the virtual-FPGA hardware.

1.4 Research Objectives

The aim of this research has been to investigate the effectiveness of using a GA to develop two types of novel robotic controllers for three specific behaviours. The behaviours chosen for this exercise were; light following, object avoidance and light following while avoiding obstacles. The research has focused on developing controllers for each of the described behaviours using LUT and EHW based controller platforms. The controllers have been compared against an ANN controller as a benchmark.

The objectives outlined above produced five research questions.

- Can a LUT be evolved for light following and object avoidance behaviours?
- Can a LUT be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?
- Can a virtual-FPGA (EHW) be evolved for light following and object avoidance?
- Can a virtual-FPGA (EHW) be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?
- How do these controllers compare with an ANN, in regard to controller performance, evolutionary efficiency, resolution of inputs, scalability and performance in unknown environments?

The methods used to answer the above questions are described in the sections 1.4.1 to 1.4.5.

1.4.1 Can a LUT be evolved separately for light following and object avoidance behaviours?

The internal elements of the LUT were configured by means of a GA process and the controller performance was assessed. The GA was designed separately to produce two individual LUTs, one to perform light following behaviours and the other to perform object avoidance behaviours. The axes that define the size of the LUTs contained either a quantized representation of light intensity or a binary representation of the six proximity sensors. The elements within the LUT contained directional information for the robots left and right motors.

The LUT developed for the light following behaviour was a two dimensional array, the \`X\` and \`Y\` axes contain the left and right light sensor values. The light sensor inputs were quantized values indicative of the robots orientation with respect to the light source.

The LUT developed for the object avoidance behaviour was a one dimensional array with the axis representing the six proximity sensors. The six digital proximity sensors were combined in binary format to produce a number form 0-63 ($2^6 = 64$), which is indicative of the proximity sensor states.

The elements in the LUTs described above contain the direction for the robots left and right motors. The direction of each wheel is defined by one of three states, forward, reverse and stopped. This results in eight possible robot motions as shown in Table 2, (Except for both motors stopped, which was excluded).

Table 2 All eight possible robot motions

Wheels(L/R)	Description
F/F	Forwards, travel straight forward
F/S	Turn right forwards, turn about right wheel clockwise (CW)
F/R	Rotate CW, pivot on the spot CW
S/F	Turn Left forwards, turn about the left wheel counter clockwise (CCW)
S/R	Turn right reverse, turn about left wheel CW
R/F	Rotate CCW, pivot on the spot CCW
R/S	Turn left reverse, turn about right wheel CCW
R/R	Reverse, travel straight backwards

The LUTs were evaluated using a computer simulation and the block diagram shown in Figure 1-4 is a representation of the development environment. The simulation and mathematical model pass sensor information to the LUT which returns the robots left and right motor directions. The GA process reconfigures the LUTs and assigns each individual a fitness value. GA progress and robot information is displayed on the graphical user interface (GUI) which also provides user control over the evolution process.

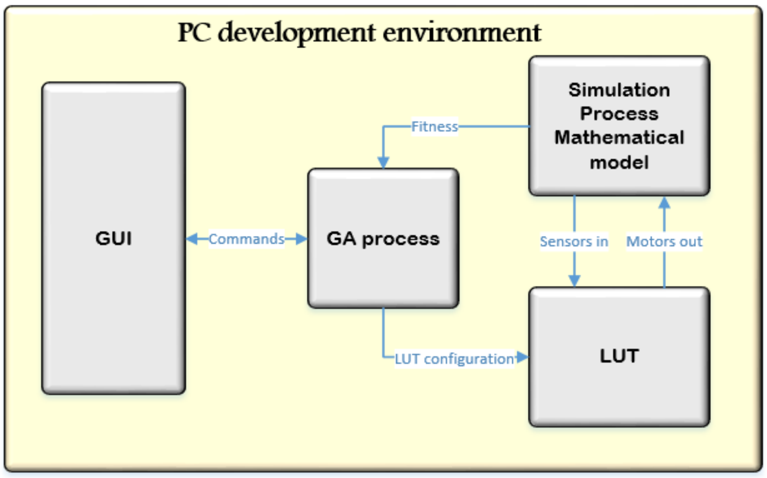


Figure 1-4 Block diagram of LUT simulation

1.4.2 Can a LUT be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?

Two methods were used to explore the issues surrounding controller scalability, and are described as follows.

Monolithic evolution– The LUT developed for this investigation was a three dimensional array. Each axis of the LUT were represented by the following robot sensors, right light sensor (X), left light sensor (Y) and proximity sensors (Z). The elements of the array contain the information that determines the left and right wheel directions.

Subsumption evolution– the previously evolved LUTs for light following and obstacle avoidance were implemented in a hierarchy based system of which the obstacle avoidance LUT was given a higher level of importance.

The LUTs developed for the combined behaviours were evolved using a computer simulation that is represented by the block diagram shown in Figure 1-4.

1.4.3 Can a virtual-FPGA (EHW) be evolved for light following and object avoidance?

A virtual-FPGA has been developed using an Altera FPGA development board. The virtual-FPGA was used to investigate the possibility of evolving a hardware controller for light following and obstacle avoidance behaviours. The controller input was the robot light and object sensors and the output was a three bit combination that was used to determine the resulting robot motor directions, (same directions as the LUT). The virtual-FPGA was configured using a CBS which was evolved using the GA process.

The block diagram in Figure 1-5 shows the interconnections between the virtual-FPGA, the NIOS processor and the PC. The PC contains the GA and robot simulation, the evolved bit-streams and simulated sensor information is passed to the NIOS processor. The NIOS processor configures the virtual-FPGA accordingly and returns the resulting robot motor direction to the PC.

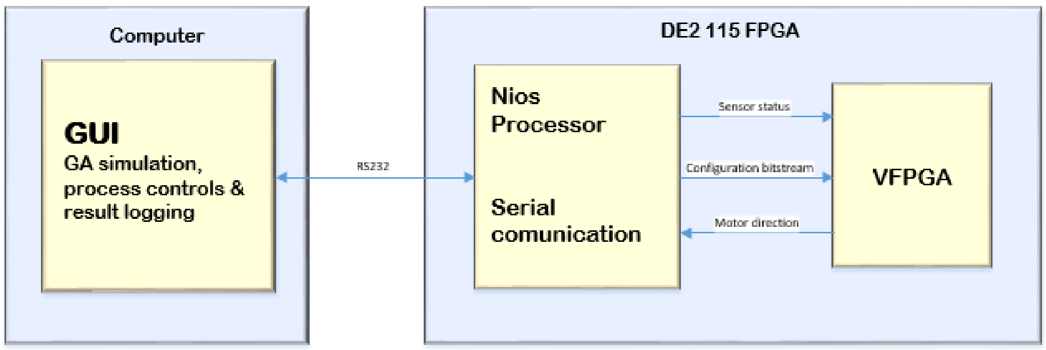


Figure 1-5 Block diagram of EHW simulation

1.4.4 Can a virtual-FPGA (EHW) be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?

Two controllers have been designed to investigate the use of monolithic and subsumption techniques to evolve a virtual-FPGA controller. The controllers are used to perform the combined behaviour of light following while avoiding obstacles.

Monolithic- the monolithic EHW controller combines the light and obstacle sensors into 26 inputs (ten left light sensor bits, ten right light sensor bits and six proximity sensor bits). The controller outputs are the same as described in section 1.4.3.

Subsumption- the previously evolved EHW controllers for light following and obstacle avoidance were implemented in a hierarchy based system of which the obstacle avoidance controller was given a higher level of importance. The block diagram shown in Figure 1-6 is a visual representation of the subsumption architecture. The high level decision maker is able to mask and unmask controllers depending on the state of the sensor inputs.

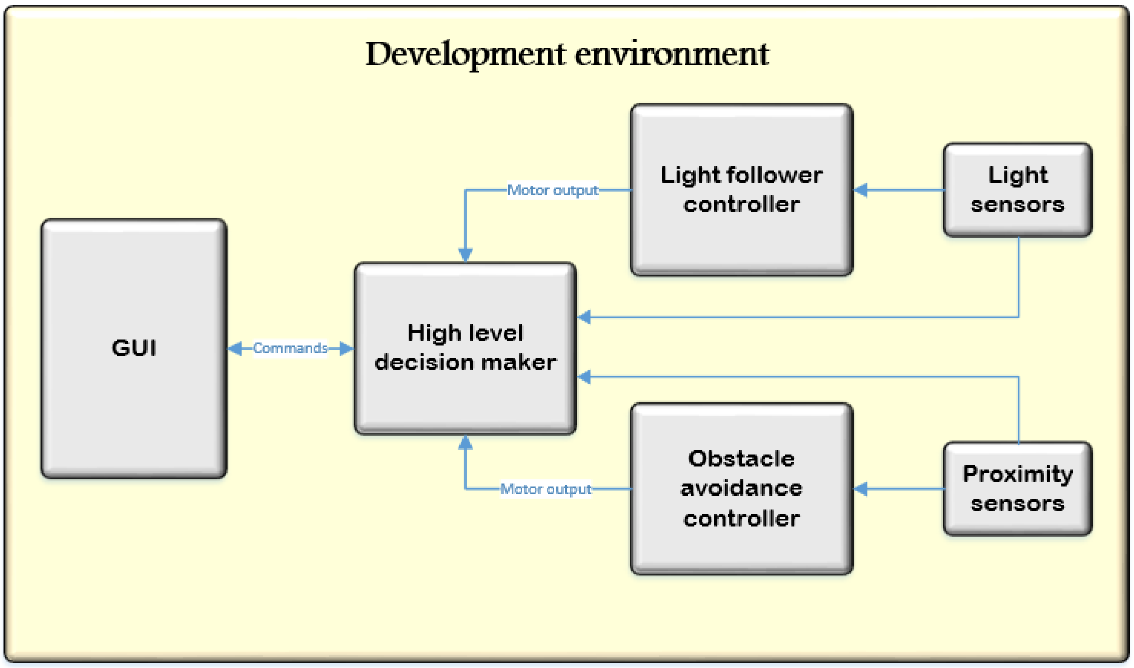


Figure 1-6 Block diagram of subsumption architecture

1.4.5 How do these controllers compare with an ANN?

An ANN was created and then evolved for the behaviours previously described. The three controllers from each platform were assessed, relating to controller performance, evolutionary efficiency, scalability, effects of sensor resolution and performance in unknown environments.

1.5 Publications

Results from this research has been published in an international conference and submitted to journals.

1.5.1 Conference Paper

M. Beckerleg, J Matulich, “Evolving a lookup table based controller for robotic navigation” in Evolvable Systems (ICES), 2014 IEEE International Conference, Orlando, Florida, 2015.

1.5.2 Journal articles submitted

-"A Comparison of Three Evolvable Robotic Navigation Controllers: an Artificial Neural Network, a Lookup Table and Evolvable Hardware" in Journal of Intelligent & Robotic Systems.

-"Evolving Robotic Navigational Behaviours using Evolvable Hardware Controllers" publication in Genetic Programming and Evolvable Machines.

1.6 Thesis Structure

This thesis is structured as described below.

CHAPTER 1**:** This chapter provides an introduction to the topic and outlines the focus of this research.

CHAPTER 2: Contains an overview of the previous work that has been undertaken within this area of research. The areas of interest outlined in this chapter have been divided into three sections, a general overview of evolutionary robotics followed by a review on the uses of EHW, look-up tables and ANN controllers. The final section explores the controllers when demonstrating light following and obstacle avoidance behaviours.

CHAPTER 3**:** This chapter is used to provide a description of the common systems used throughout this thesis. This includes a description of the hardware systems and models that have been developed and a description of the robot and its hardware along with mathematical models which described the robot kinematics and sensor systems.

CHAPTER 4**:** A complete description is provided on the work undertaken during the development of ANN based controllers. The chapter is split into three main sub categories each outlining the work undertaken and results obtained for the three desired behaviours.

CHAPTER 5**:** A complete description is provided on the work undertaken during the development of LUT based controllers. The chapter is split into three main sub categories each outlining the work undertaken and results obtained for the three desired behaviours.

CHAPTER 6: A complete description is provided on the work undertaken during the development of EHW controllers. The chapter is split into three main sub categories each outlining the work undertaken and results obtained for the three desired behaviours.

CHAPTER 7: The content within this chapter provides a comparison between each of the developed controllers for three separate behaviours. The comparison considers five key areas, controller performance, evolution efficiency, scalability, quantization and operation in unknown environments.

CHAPTER 8: This chapter is the final chapter of this research and is used to provide a detailed description and analysis of the results obtained. The chapter also outlines any future research which could be undertaken to further develop the work completed as part of this research.

Chapter 2

Chapter 2: Literature Review

Evolutionary algorithms are heuristic search procedures designed around the natural reproduction processes. The algorithms are designed as search engines that are used to find a solution to a problem of which the solution is not easily calculable. The type of evolutionary algorithm chosen for this research is a GA. The GA has been developed to mimic the natural selection and evolution process which was developed in 1859 by Charles Darwin. The evolution process involves three discrete functions, reproduction, fitness allocation and selection. Reproduction is the process where a new generation of offspring is created from the parent population. The offspring are created using crossover and mutation methods with the goal of producing a superior individual. Fitness allocation is the process where the individuals are tested and their performance assessed. Each individual is put through the testing procedure and a fitness value is assigned to each. The selection process uses this fitness value to determine the individuals that are kept and used for reproduction and the ones that are discarded.

2.1 Review of robotic controllers

Robotic controllers and fully autonomous robots have long been the focus of evolutionary algorithms. The development of heuristic optimisation algorithms has provided a method for robotic controller design that does not rely on human design. This has proven beneficial for several areas of controller design, from simplistic controllers through to complex controllers that would otherwise require a high level of understanding and knowledge.

2.1.1 ANN controllers

An ANN mimics a biological neural network with a structure of layered neurons with each layer interconnected. Each input to the neuron is multiplied by a weighting factor. When the sum of these inputs into the neuron exceeds a firing threshold, the output value of the neuron will change. This output can be of various shapes ranging from a step to a sigmoid function. There are wide ranges of network structures with the simplest being feed-forward. Normally the structure of the network and the neuron transfer function are fixed when it is designed, while the interconnection weightings are adjusted in a training period before the ANN is used.

For evolutionary purposes the weightings, firing thresholds, transfer function and even the network structure can be encoded in a chromosome, which can be evolved using standard evolutionary techniques. The search space of the chromosome is related to the number of layers and neurons in each layer of the network. ANN controllers have become widely used in evolutionary robotics, and applications for their use have been extensively researched, such as; real time object detection and guidance oriented control systems using computer vision in vehicles.

Pomerleau et al. [1] used a rule based method of combining several ANNs. Complex behaviours such as those required for self-driving cars require a controller capable of employing different controller capabilities for differing environments. The author used a rule based method for achieving this controller selection method and developed a high level controller capable of staying on the road and following a set route to a destination.

Kodjabachian et al. [2] used simple geometry oriented cellular encoding (SGOCE) to evolve recurrent neural networks for control of a simulated six legged insect. The resulting robotic controller was capable of gradient following and object avoidance behaviours. On average a solution which was capable of performing the desired task to a satisfactory standard was found in 50 generations.

Glasius et al. [3] Uses a Hopfield type neural network with nonlinear neurons to control a robot for path planning and object avoidance behaviours. Three applications were investigated; 1) A point-robot was tasked with finding a point in a labyrinth; 2) A two joint planar robot was tasked with avoiding obstacles within an arena; and 3) A moving target based environment. The evolved controllers were capable to finding a suitable path between point locations, and were able to avoid both static and moving obstacles while moving between points.

Bartha et al. [4] used a fully meshed ANN to control a light seeking robot, the controller was a three layer recurrent neural network with two hidden nodes. The goal of the controller was to navigate the robot towards a light source, which was placed inside a rectangular arena. The author explored the effects caused by differing fitness functions and chromosome combination methods on the evolution efficiency and provided a comparison between the ANN and a finite state machine (FSM). It was found that the ANN outperformed the FSM and was able to reach a suitable fitness within 500 generations. However, the FSM controller was found to be simpler in design and hence easier to comprehend.

Abhishek et al. [5] developed an ANN controller for obstacle avoidance behaviours, the model was designed around the Khepera robot which used six touch and two light sensors. The authors evolved controllers to a suitable fitness in 50 generations. The controllers were also tested in a real world environment and proved to perform well when tasked with avoiding the wall of the arena.

Wahab [6] trained an ANN controller for a mobile robot which was designed with two types of proximity detection, ultrasonic range and infrared sensors. The controller used two ANNs the first of which provided position control while the second provided the obstacle avoidance control. The final controller was successful at navigating an arena and stopping when reaching a target location.

Harter [7] designed a neural controller which has been used to navigate a Khepera robot about an arena with obstacles. Rather than the standard ANN model the controller was based on the aperiodic K –set neural population model. Each controller chromosome consisted of 10 KA units, four from the left proximity sensors, and four from the right and two for the robot motors. Each unit ranged from -1 to 1 and consisted of 100 increments. Results from the experiments showed that the evolved controllers produced simple but effective object avoidance behaviours.

Elnebreich and Klingler [8] used an ANN controller to navigate a four wheeled robot around an arena. The controller was provided sensory inputs via three dynamically adjustable infrared sensors. The sensors were adjusted to provide an increase detection angle. The controller output provided robot steering and motion control. A comparison was undertaken of the GA evolved ANN vs. a pre-engineered ANN. It was found that after 50 generations the ANN controller was able to outperform the pre-engineered controller. It was also discovered that small ANNs with fewer nodes performed best and those with large node numbers would exhibit population convergence.

2.1.2 Fuzzy logic controllers

The FLC attempts to avoid crisp data, such as a precise temperature, or true/false, instead using fuzzy data that is imprecise or partial. This allows for a robust controller, which is easy to implement. A FLC has three processes, fuzzification, rule evaluation, and defuzzification. Fuzzification converts precise data inputs into imprecise values, by assigning the input to a degree of membership. Rule evaluation acts on the fuzzified inputs by a set of fuzzy rules, normally a sequence of if-then logic statements and fuzzy logic algebra. Finally, the manipulated data is defuzzified to produce a precise output value. The parameters of a fuzzy logic controller including fuzzification of inputs, implementation of fuzzy rules, and defuzzification of outputs, are normally configured by experiment or design. However, a GA can be applied to these parameters to evolve fuzzy logic controllers. Seung et al. [9] developed a sensory motor controller for a mobile Khepera robot which was capable of efficiently navigating different arenas and demonstrating emergent behaviours. Hagras et al. [10] used a modified GA to evolve a fuzzy logic based robot controller. The controller was capable of online self-calibration to automatically adjust the membership functions. This made the controller suitable for a fast changing dynamic environment. Sung et al. [11] developed a self-organized fuzzy controller to provide the means of controlling a wheeled mobile robot. A rule based evolutionary process was used to find solutions to output and input membership functions. It was discovered that optimal solutions were not always found using the rule based membership function.

Mohammad et al. [12] developed a controller for the E-Puck robot, the controller was designed for robot navigation and obstacle avoidance. The author used MATLAB to implement the fuzzy logic rules for which eight infrared sensors provide the inputs while the outputs are used to control the robots two motors. The fuzzy logic was a successful controller for the E-Puck robot and the controller enabled the robot to navigate through an arena to a desired location.

Almasri et al. [13] developed two fuzzy logic robotic controllers, the controllers were designed for the E-Puck robot and implemented two behaviours; line following and object avoidance. The line follower controller used two of the three available line sensors as inputs. This way the controller determines the difference between the two sensor values and inputs the difference into the fuzzy logic controller. Two motor control outputs were generated which determined the robots motion. The object avoidance controller used similar methods however eight proximity inputs were used as inputs to the fuzzy system. The two controllers were combined to produce a robot that was capable of following a line until an object or other robot was encountered, the robot then successfully avoided the object.

2.1.3 LUT controllers

The LUT, also referred to as a “table-based” controller, is used for quantized discrete sensor inputs and actuator outputs where the relationship between the inputs and outputs are mapped in a table. They have their limitations in relationship to scalability; however, they are computationally fast in their execution. A current example of their use in control is the MAX31760 controller with a LUT stored in non-volatile memory. They have been researched for several control applications including building environmental control [14] where LUTs are used to calculate building temperature control set points where dependencies are based on current and predicted weather, occupancy and other conditions; Use in pH neutralization [15] where two dimensional LUTs are used in conjunction with neural networks and fuzzy controllers for applications such as boiler feed water and wastewater treatment; Underwater vehicles [16], where the authors compare the response time, overshoot and steady state error of LUT controllers, fuzzy logic controllers and conventional controllers for use in a six degree of freedom autonomous underwater vehicle; and mobile robot controllers [17], where a LUT controller is designed to enable faster obstacle avoidance in a multi obstacle environment, it was found that the LUT method reduced the operational time by nearly 80% when compared to a positive/negative fuzzy controller.

For evolutionary purposes, the parameters and size of the LUT can be encoded into a chromosome and evolved. Several evolutionary capable robotic LUT controllers have been developed to control a range of robotic systems including: a mobile inverted pendulum [18], the walking gait of a hexapod robot [19], a curved ball-balancing beam system [20], a ball plate and four robotic behaviours [21], orbiting, path following, follow the leader and dispersal, implemented on a Kilobot simulation [22]. A description of the inverted pendulum, curved ball balancing beam and ball and plate system is described below.

A LUT controller for a mobile inverted pendulum was evolved, this controller was a two dimensional LUT, with the input axis linked to the pendulums angle and angular velocity. The internal elements of the LUT contained the motor speed and direction information which were used to maintain the robots upright position. The evolved controller was complete within 200 generations and was capable of maintaining an upright position for more than 200 seconds [18].

A simulation has been developed for a curved ball balancing beam, a three dimensional LUT controller provided the interface between the system inputs (ball position, beam position and ball speed) and the system output (Beam drive motor speed and direction). The author used a GA with two point crossover and tournament selection to evolve the controller and compared the effects on evolutionary efficiency caused by increasing the number of motor speeds in the LUT elements. It was that a controller with only two speeds was able to evolve to the same standard as a controller with 11 speeds in far fewer generations [20].

A novel LUT controller has been developed for a ball and plate system. The system uses two servos to drive the plate about the X and Y axis while a touch screen provides feedback of the ball position to the controller. A separate LUT was evolved for each servo which provided set point selection on the X and Y axis. The author investigated the fault tolerance of the system by introducing a fault condition and determining how quickly the controller would recover. The fault recovery time varied between controllers. However, the majority of controllers recovered in 10 to 30 seconds [21].

2.1.4 EHW controllers

An EHW robotic controller uses evolved logic circuits to control the robot. The electronic circuits coded by a hardware description language and implemented in a FPGA are described by the configuration bit-stream used to program the device. This bit-stream can be used as a chromosome in EHW where the phenotype of the chromosome is the circuit. Original research used the Xilinx XC6216 FPGA, which was suitable for evolution as its architecture was immune to destructive chromosomes, however this FPGA is no longer produced [23, 24]. Currently the three main methods used for hardware evolution of robotic controllers for navigation are: a) genetic compilers that are specifically designed to evolve a CBS without destructive architecture [25, 26]; b) genetic programming where the hardware descriptive language is evolved, and then converted into the bit-stream [27]; and c) the virtual FPGA configured inside a normal FPGA. The virtual FPGA is designed for non-destructive and course-grained architectures suitable for evolution [28, 29]. Previous EHW controllers are described.

Tyrrell et al. [25] developed an EHW controller, which was evolved on a Xilinx FPGA. The controller was designed for object avoidance behaviours and was implemented on the Khepera robot. The controller was comprised of 22 LUTs with eight inputs from the proximity sensors and two outputs to the robots motors. The controllers struggled to reach an average fitness of 1200 in 50 generations; however, when the mutation rate was increased it was found that an average of 1200 was reached in less than 30 generations and the maximum fitness increased from 1200 to 1400.

Keymeulen et al. [30] created an EHW controller which performed object avoidance and mobile navigation tasks. The robot was 250mm in diameter with ten infrared proximity sensors and two cameras which provide information used for colour object tracking. The controller operates as a simple Boolean function control system, the Boolean equivalents of the proximity sensor outputs are binary coded and used as the input to the control system. The controller outputs are used directly to control the robots left and right motors for navigation. The evolved gate level hardware controller was able to take advantage of the numerous correlations of the sensor states. This resulted in a highly robust control system which was not affected by the shape of the obstacles.

Dhanaskekaran et al. [31] used an EHW system to coordinate multiple elements of a smart antenna array. The system was designed to handle faults and extreme unexpected situations which were otherwise incorrectly handled by conventional systems. The control system developed for this was designed around an EHW circuit based in an FPGA and was evolved using a GA. The resulting outcome was a system capable of self-configuration and fault tolerance. It was also found that the EHW system outperformed the conventional systems in terms of evolution efficiency and controller performance.

2.2 Hierarchical evolution

Standard evolutionary techniques use monolithic evolution where a behaviour is evolved in its entirety, however this technique has limitations in the evolutionary process requiring a large amount of time for evolution, difficulties in the initial generations (bootstrap problem), and many local minima. These problems are alleviated using the following techniques, including subsumption architecture and incremental evolution that divide behaviours into modular tasks.

Note, this research does not focus on hierarchical evolution instead hierarchical evolution is used to explore the scalability properties of the controllers.

2.2.1 Subsumption

The concept of subsumption evolution was first conceived by Brooks [32, 33], using a hierarchy layered behavioural approach where the lower level layers directly interfacing to the actuators, perform basic behaviours such as movement, avoidance, and following, whereas the higher layers have complex behaviours such as foraging or fleeing. Each layer works independently but the higher layers can override (subsume) the lower layers using an inhibitor which blocks the output of a lower layer, or a suppressor which replaces the inputs to the lower layer.

Saito et al. [34] used subsumption architecture for emergency obstacle avoidance in a mobile robot. The robot in this experiment consisted of an electronic wheelchair, a laptop and an embedded microcontroller which was used to interface with a laser range finder. The overall task was decomposed into four layers: autonomous, turn left, turn right and emergency stop. The range finder detection zone has been split into three sections and the controller determines which tasks will be suppressed based on the area in the detection zone that an object is detected. The controllers were tested with static and semi-static obstacles with good results.

Dasmane [35] created a subsumption based controller for a robot that moved towards a light while avoiding obstacles. The robot used in the experiments was multi legged and contained real time proximity and light sensing capabilities. Individual controllers were used for light following and obstacle avoidance, however only one controller was used at a time. The light following controller was subsumed whenever an obstacle was detected giving the obstacle avoidance controller a higher level of priority.

Turner et al. [36] used a multi-layer subsumption based robot controller to map its environment while avoiding obstacles. A FSM based controller was used to explore the advantages of subsumption based architectures. A controller platform was developed that used input suppressors and output inhibitors to dynamically manipulate the controller for any given state of sensor input. Much like the work completed by Brooks [33] the controller utilised different function blocks. In this case the blocks required were wander, explore and avoid collisions. The controller was designed to explore and map a robotics lab until all available areas had been mapped. After several test runs and a number of problems encountered, the controller managed to produce a map of the lab which was approximately $790m^2$.

Tiong Cheng and Mahyuddin [37] used subsumption behaviours to move a robot towards a light source while avoiding obstacles. The robot behaviours were split into three sub behaviours, obstacle avoidance, goal seeking (light following) and wandering. The wandering behaviour has the lowest priority and was only implemented when either light or object was detected. The object avoidance controller had the highest level of priority to ensure no object was hit. A trial of 40 produced 35 successful controllers that avoided the obstacles and found the light source.

2.3 Genetic algorithm

The use of a GA as a means of finding a solution to a complex problem was originally explored by John Holland in the 1960s [38]. Holland`s research was directed towards the natural evolution process and the possibility of mimicking this process in manmade systems. The GA provide a method of finding solutions to a problem by using a process very similar to that of which occurs in nature according to the theory of evolution developed by Charles Darwin [39]. The GA process begins by creating a randomized initial population. The population is comprised of several individuals otherwise known as chromosomes. Each of the chromosomes contains a potential solution to the problem for which a solution is required. The initial population is then put through the three stages of evolution as outlined below.

- Fitness evaluation – where the behaviour of each individual chromosome is assessed in relation to the desired behaviour and scored on how well the behaviour was performed.

- Selection – individual chromosomes from the entire population including the offspring are chosen to continue the evolution process depending on the fitness values they have been assigned. Individuals with a high fitness perform the desired task better than individuals with low fitness values and hence are more likely to be chosen for reproduction.
- Reproduction – pairs of adults in the population are used to reproduce a new set of offspring with the goal of passing on desirable traits to the future generations. Amalgamation of the parents to form the offspring is a randomized process designed to mimic that which occurs in nature.

The GA process is a continuously evolving cycle and the steps outlined above are repeated over and over while the population adapts and changes to suit the requirements as outlined by the desired behaviour.

2.3.1 Fitness allocation

The measure of a chromosomes performance (fitness) is calculated by means of a fitness function whose output is a reflection on the chromosomes ability to perform the desired task. The design of the fitness function is important to the success of the GA as it provides the foundation from which the evolved chromosomes are formed. A slightly misguided fitness function can cause the population to converge on a particular undesired behaviour. However, a precisely tuned function can greatly increase the rate at which a desired result is obtained.

The importance of fitness design has previously been explored, Trujillo et al. [40] considered the effects on performance of an unmanned aerial vehicle (UAV) due to variations in fitness algorithms. Teams of three UAVs were required to set off from a starting point and perform a number of defined tasks. The authors discovered that due to limitations in the fitness algorithm some of the UAVs would complete the task much faster than others and return home earlier creating an unbalanced taskforce which was less than desirable. To overcome this, the authors made adjustments to the fitness algorithm to include constraints which would penalize any UAVs not returning home within two minutes. Introducing this constraint in the fitness algorithm proved to balance the task force and group the return times within the allowable time period.

The fitness algorithm for most GAs tends to force the population to perform a particular behaviour. If this fitness algorithm is slightly misguided then the resulting outcome will also tend to be slightly inaccurate. Hence for most genetic evolution problems the actual fitness algorithm will be tuned and adapted over several experimental test runs with the goal of producing the desired result.

2.3.2 Selection schemes

A selection process chooses individuals from the evolving population to be used in the reproduction stage of the evolution. The selection process used can conform to one of the many previously developed selection methods and in some cases contain attributes from more than one method. The goal of the selection process is to select individuals that will provide the best chance of success while maintaining diversity in the population. The actual selection is usually based on the performance of each individual such that better performing individuals are kept over those not performing so well.

Tournament selection

Tournament selection is a technique used in evolutionary algorithms to select chromosomes from the population for reproduction. Tournament selection operates by executing tournaments within the population much like the tournaments of sporting events. The number of individuals in each tournament has a direct effect on the selection pressure applied to each individual. The selection pressure is a measure of the requirement for an individual to perform well, this pressure decreases with small tournament sizes and increases with larger tournament sizes. Miller and Goldberg [41] created a model that predicts the selection pressure based on tournament size. The model was then applied to predict population convergence rates based on the selection pressure.

Fitness proportionate selection

Fitness proportionate selection, otherwise known as roulette wheel selection, is a technique used in evolutionary algorithms for selecting individuals that may prove beneficial for the success of the population. Similar to tournament selection the fitness proportionate selection method requires each individual to have an assigned fitness value. The fitness value is divided by the total fitness of the population and used to determine the probability of the individual being selected for reproduction. This can be visually represented in the form of a roulette wheel as shown in the following example.

Table 3 Fitness proportionate example

Individual number	Individual fitness
1	50
2	10
3	20
4	6
5	1
6	3
7	4
8	9

Based on the information provided in

Table 3 the probability for selection of each individual can be determined; the fitness of each individual is normalised to one by dividing the fitness by the total shown in Equation 1. The resulting normalised values are shown in Table 4

$$\text{Total fitness} = \sum_{k=1}^{8} F_k = (50 + 10 + 20 + 6 + 1 + 3 + 4 + 9) = 103 \qquad \text{Equation 1}$$

Table 4 Fitness proportionate example with normalized fitness

Individual number	Individual fitness	Normalized value
1	50	0.485
2	10	0.097
3	20	0.194
4	6	0.058
5	1	0.001
6	3	0.029
7	4	0.039
8	9	0.087

The roulette wheel shown in Figure 2-1 clearly shows that the chance of individual one being selected is much greater than the others. However the individuals with a low fitness do still have a chance of been selected, meaning that if they do hold valuable chromosomes there still remains a chance that they could be used to reproduce and in turn produce an offspring containing that valuable information.

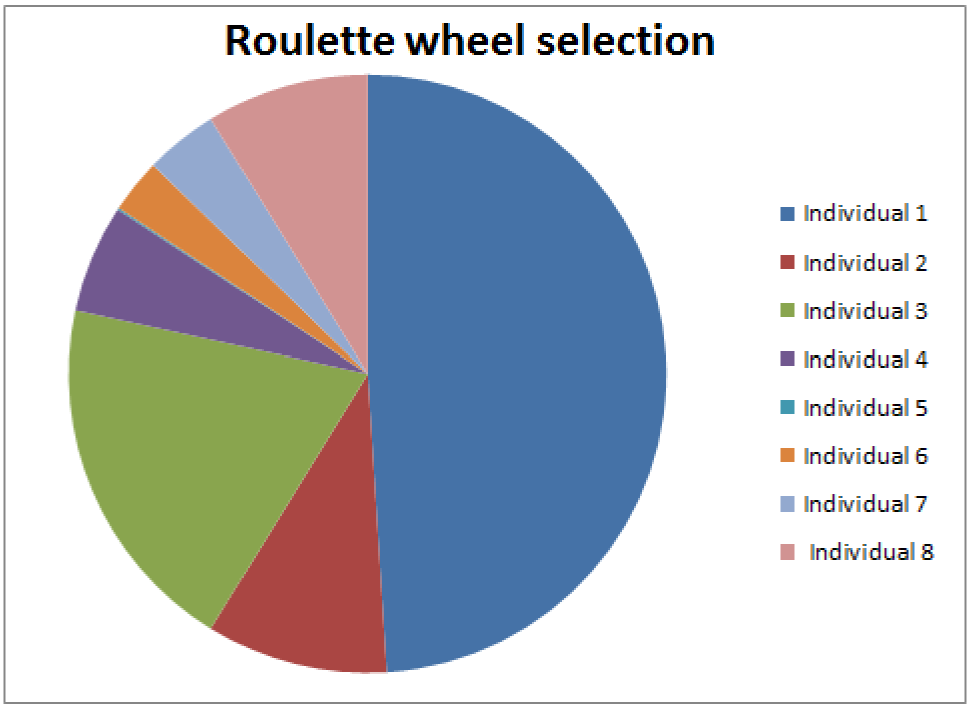


Figure 2-1 Roulette wheel selection scheme

Razali and Geraghty [42] compared rank based roulette wheel and proportional roulette wheel selection strategies with tournament selection. The authors compared the selection performance on the common traveling salesman problem and found that tournament selection outperformed both roulette selection methods. They also discovered that for large problems population convergence became a problem with tournament and rank based roulette wheel selection methods.

2.3.3 Procreation

Single point crossover

Selected at random one point inside the chromosome is chosen, this point provides the separation line which will be used to divide the chromosome. In some cases this is easier than others, for example a two dimensional chromosome like the ones shown in Figure 2-2 can easily be split with one point. However complex chromosomes such as three dimensional LUTs may require a more sophisticated reproduction technique than single point crossover.

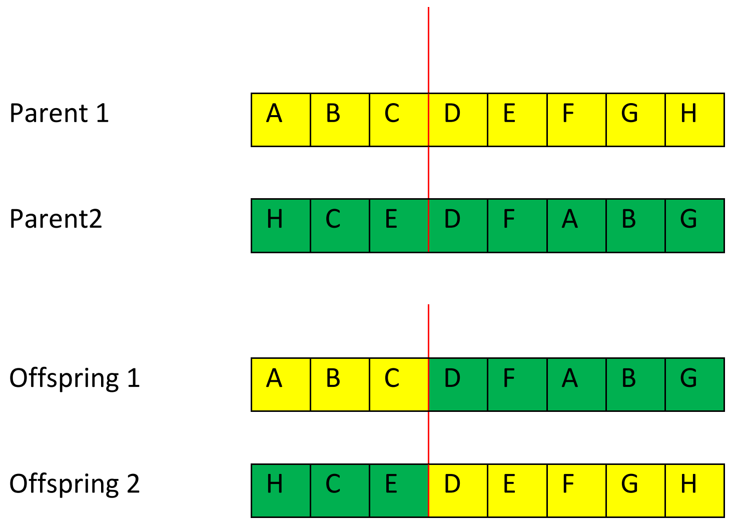


Figure 2-2 Single point cross over example.

In the example shown in Figure 2-2 two parent chromosomes have been selected for reproduction and a single point for crossover has been determined. Splitting the parents along the crossover point and combining the sections with one part from each parent creates two offspring each containing unique properties from their parents.

In some particular situations the offspring may need to be adjusted or repaired after the crossover process. This is necessary when the particular task that the evolutionary algorithm has been designed for requires that the genes in each chromosome cannot be repeated. However not all evolutionary algorithms require a unique combination of genes in each chromosome, in some case it does not matter if a gene was to repeat over and over again. Such is the case for a motor controller where the chromosome may contain motor speed values, in this case there may be several conditions that may require the same speed and hence many genes in the chromosome may be repeated.

If the chromosomes need to be repaired then the evolution process must take care of this before the offspring are released into the population. In Figure 2-3 the genes highlighted in red were damaged genes and have been repaired to complete the chromosome.

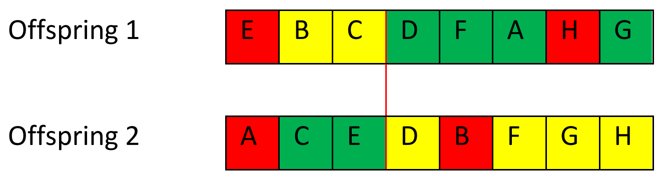


Figure 2-3 Adjusted chromosomes

Multi-point crossover

Multi-point crossover is a similar technique to single point crossover used in the creation of offspring. The difference between single and multi-point is the number of points and hence the number of sections created and combined together for each offspring. Two-point crossover becomes necessary when the evolutionary algorithm is attempting to evolve large chromosomes with complex desired results. The problem with using a single point crossover method for larger chromosomes, is that it becomes difficult and highly unlikely that small sections from the internal sections of the parent chromosomes which may be key to the success of that individual, will be passed on to the future generations.

Figure 2-4 demonstrates how a two-point crossover technique can extract small sections of genes from within the parent chromosome and pass that information on to the offspring. In this example genes C and D are passed from parent one to offspring one and genes E and D are passed from parent two to offspring two. In some cases several points of crossover can be selected providing a larger number of sections used in the creation of offspring.

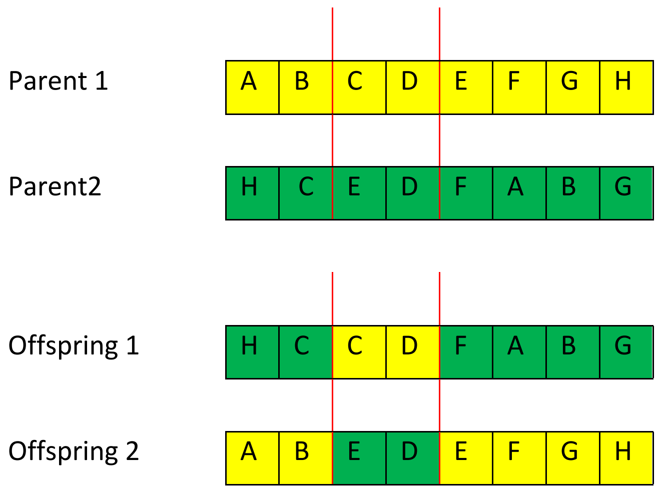


Figure 2-4 Two-point cross over example

2.3.4 Mutation

In nature mutation occurs spontaneously, usually at non periodic intervals with a low mutation rate [43]. In evolutionary algorithms mutation occurs when a gene or chromosome is altered from that which would normally occur during the reproduction process. The rate of mutation is generally set very low in the range of 1-3%, meaning that there is only a very low chance that a mutation will occur and when it does it usually only effects one gene in the chromosome. The gene that is affected is chosen at random and the change that occurs also happens at random. Mutation helps to provide diversity. When the evolution of a population has slowed or even stopped, which may be due to the fitness reaching a local maximum on the fitness landscape, a mutation can provide the required change to move the evolution process to the next peak of the landscape and continue the evolution process.

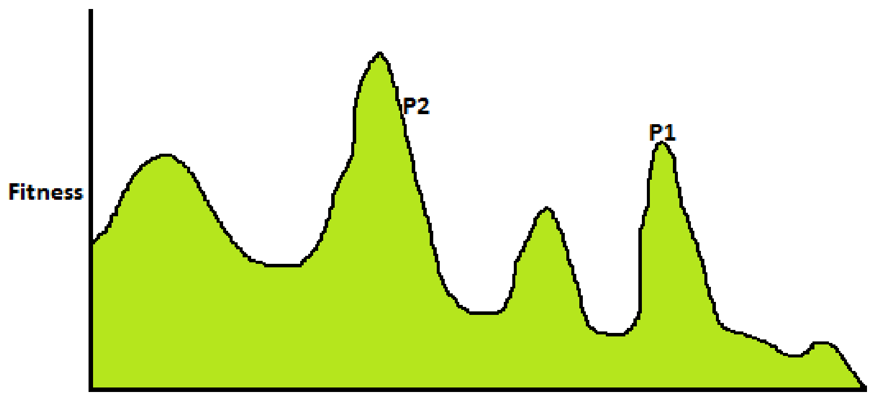


Figure 2-5 Fitness Landscape mutation example

Figure 2-5 shows an example of a fitness landscape with several local maxima if the evolution process were to converge to the point indicated by P1 then the only way the process could continue to find the optimum maxima is by means of a mutation. After which the mutated individual may appear on the landscape at the point indicated by P2 hence allowing the evolution to continue.

Chapter 3

Chapter 3: Common Systems Developed for Experimentation

This chapter describes common systems that have been used in experimentation, and is broken into three main sections. An outline of each section is described below.

- Section 1: Robot overview. This section contains the mathematical models, and descriptions of the hardware systems for the robot from which the simulations have been derived.
- Section 2: Software and simulation design. This section describes the software that performs the GA evolution of the controllers and contains a functional description of the GUI.
- Section 3: GA process. This section contains a description of the processes that have been developed specific to the GA process, such as fitness allocation and selection processes.

Each of the sections outlined above contain an overview followed by a detailed description of the systems that were created for each of the robotic behaviours.

3.1 Hardware overview

The physical robot used in the research for this thesis was specifically designed by the author for research into evolvable robotic controllers for navigation. The robot hardware is comprised of an FPGA development board (Altera’s DE0-Nano), light and obstacle sensors, two wheels with associated hardware drivers, Bluetooth communications, user interface and battery. On board the FPGA, a soft core NIOS processor has been implemented, providing the interface between sensor inputs and the outputs used for motor control. Hardware based modules have been designed in Verilog that relay sensor information to the processor, the sensor information is used to determine how the robot should operate. The processor passes motor speed and directional information to additional hardware modules to drive the robots two DC motors. The DC motors independently drive a wheel located opposite each other near the perimeter of the robot. This gives the robot the ability to rotate on the spot as well as efficiently move in a straight line to any location in the arena. Figure 3-1 shows an actual photo of the robot designed to carry out the research.

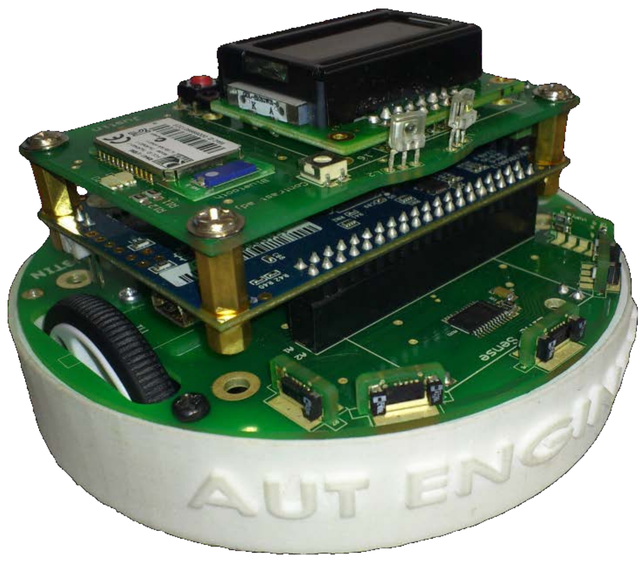


Figure 3-1 Photo of actual robot

3.1.1 Robot kinematics and mathematical model

The robot modelled for this research is a stable two wheeled platform which uses two small guides to provide balance. The mathematical model of the robot is described using three equations. The first equation describes the robots change in heading, the second equation describes the distance travelled along the X axis and the third equation describes the robots distance travelled along the Y axis. Each equation is developed with respect to the left and right motor movements.

The subsequent set of parameters and equations are developed with the following confines applied to the system. Firstly, the simulation is designed to run at 50ms intervals *(dt).* The second restriction applied to the system is the speed control on the motors, to simplify the mathematical model it was decided that the motors should run at one set speed and only have control on the wheel rotational direction (CW or CCW).

Table 5 contains a list of parameters and terms used in the equations and formula in this section.

Table 5 Parameters used in mathematical model of the robot

dt	Simulated time step (ms)
θ	Wheel Diameter (mm)
RPM	Motor revolutions per minute
CW	Clockwise
CCW	Counter clockwise
d	Distance travelled in a straight line in 1 *dt* (mm)
r	Pivot radius (mm)
Wø	Wheel circumference (mm)
$d\theta$	Change in robot heading
dx	Change in x location
dy	Change in y location
R	Turn radius

Because the motor speed has been defined as constant, a finite list of available robot motions can be described; Forward, reverse, turn left forward, turn right forward, turn left reverse, turn right reverse, pivot CW, pivot CCW and stop.

Pivot left and right

When the robot is pivoting CW or CCW the X and Y location of the robot will remain the same with only the heading increasing or decreasing, as shown in Figure 3-2.

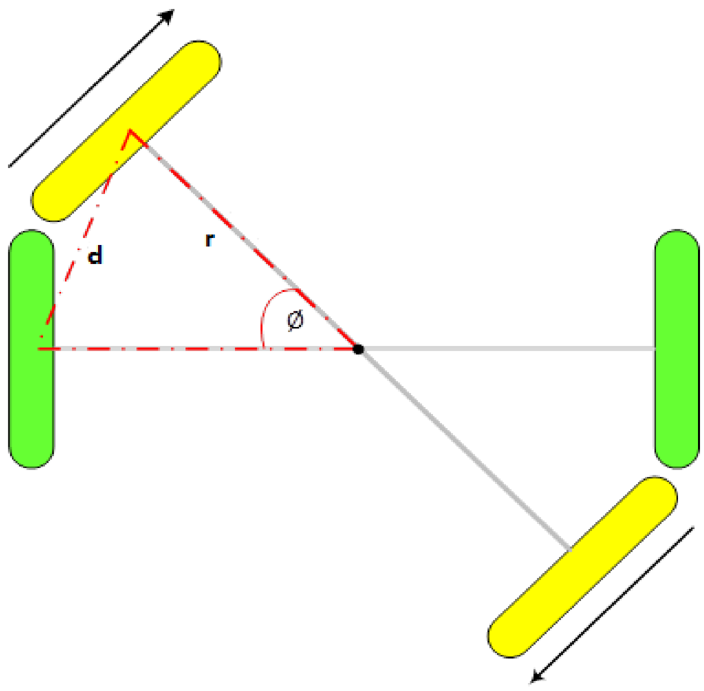


Figure 3-2 Diagram of the robot pivoting clockwise

To determine the change in heading when the robot is pivoting, the distance travelled by a wheel in one time period *(50ms)* needs to be calculated using Equation 2 and Equation 3.

$$W\o = \pi * \theta = \pi * 32 = 100.5mm \qquad \text{Equation 2}$$

$$d = \frac{\frac{RPM}{60}}{dt} * W\o = \frac{\frac{73}{60}}{0.05} * 100.5 = 6.1mm \qquad \text{Equation 3}$$

Where:

Wø is the wheel circumference.

θ is the diameter of the wheel (32mm),

RPM is the selected motor speed, 10% of the maximum (730RPM),

dt is the simulated time period (50ms),

d is the distance travelled in dt

The black dot in the centre of the two wheels in Figure 3-2 represents the centre point of the robot. If both the left and the right motors are running in opposite directions the robot is said to be pivoting and the resulting change in heading can be calculated using Equation 4.

$$d\theta = = \frac{\mathrm{d}}{\mathrm{r}} * \frac{180}{\pi} = \frac{6.1}{40} * \frac{180}{\pi} = \pm 8.7°$$ Equation 4

Forward and reverse

When the left and right motor directions are the same i.e. both forward or both reverse, the robot moves in a straight line. For simulation purposes the speed of the motors are assumed identical.

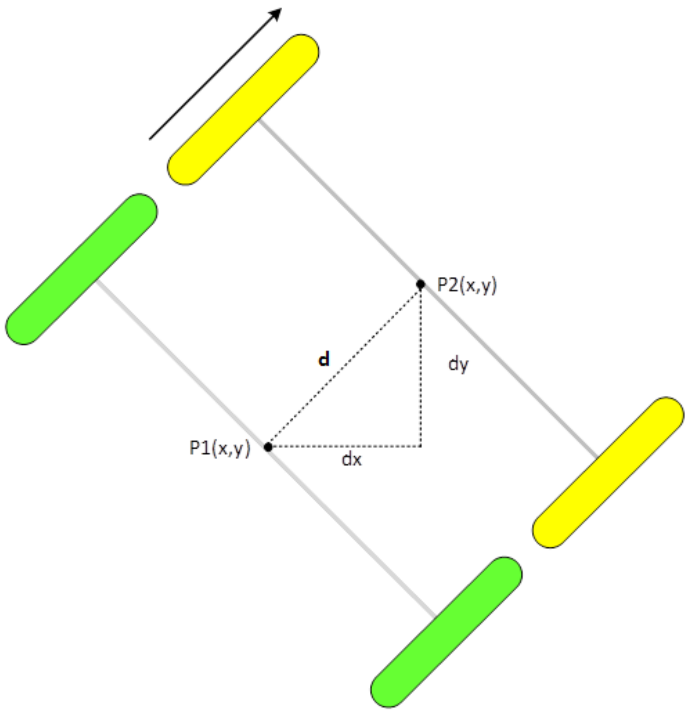


Figure 3-3 Diagram of the robot moving in a straight line

From Figure 3-3 Equation 5 and Equation 6 for the change in X (dx) and change in Y (dy) coordinates can be derived.

$$dx = 6.1mm * \sin(Robot\ heading)$$ Equation 5

$$dy = 6.1mm * \cos(Robot\ heading)$$ Equation 6

Example calculation: If the robot is travelling at a heading of 0 degrees, the expected change in coordinates should be a 6.1mm increase on the X axis and zero change on the Y axis.

$$dx = 6.1mm * \cos(90) = 6.1mm$$

$$dy = 6.1mm * \sin(90) = 0mm$$

As expected the robots position changes by 6.1mm on the X axis and remains the same on the Y axis.

This set of equations holds true as long as the robot is moving in a straight line. If the robot starts to move nonlinearly then a separate set of equations are used to find the new location. These equations are explained in the next section.

Turn Left and right

When one of the robots wheels are moving and the other is stationary, the robot will pivot around the stationary wheel producing a change in the robots heading and position, as shown in Figure 3-4.

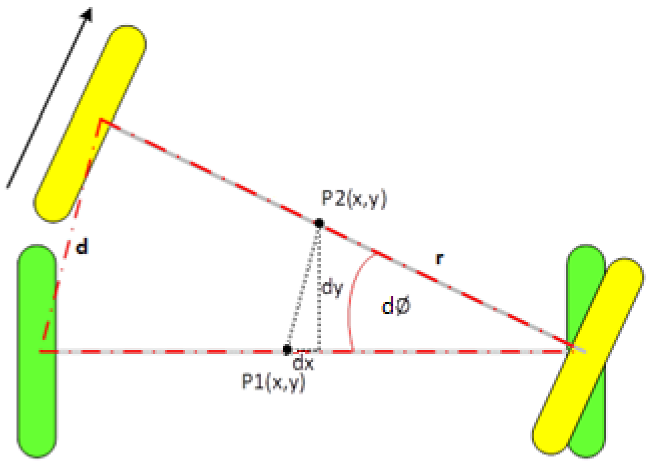


Figure 3-4 Diagram of the robot turning

The new robot position is calculated using Equation 7 and Equation 8.

$$dx = 40 * [cos(Robot\ heading + \ d\theta) - cos(Robot\ heading)] \qquad \text{Equation 7}$$

$$dy = 40 * [sin(Robot\ heading + \ d\theta) - sin(Robot\ heading)] \qquad \text{Equation 8}$$

Where:

$d\theta$ is the change in robot heading while turning which equals half that of the change in heading while rotating ($\pm 8.7°$), therefor $d\theta = \pm 4.35°$.

Example calculation: If the robot heading is 0 degrees and the left wheel moves forward while the right wheel remains stationary. Then it is expected that the robot should move along on the X axis by some value that is slightly less than half of 6.1mm and move down the Y axis by a distance slightly greater than zero.

$$dx = 40 * (\sin(4.35) - \sin(0)) = \ 3.03mm$$

$$dy = 40 * (\cos(4.35) - \cos(0)) \ = -0.12mm$$

The values obtained from the equations above are as expected, a small amount of movement down the Y axis due to the negative value (-0.12mm) and 3.06mm along the X axis.

3.1.2 Robot Sensor Models

3.1.2.1 Obstacle avoidance systems

Proximity hardware

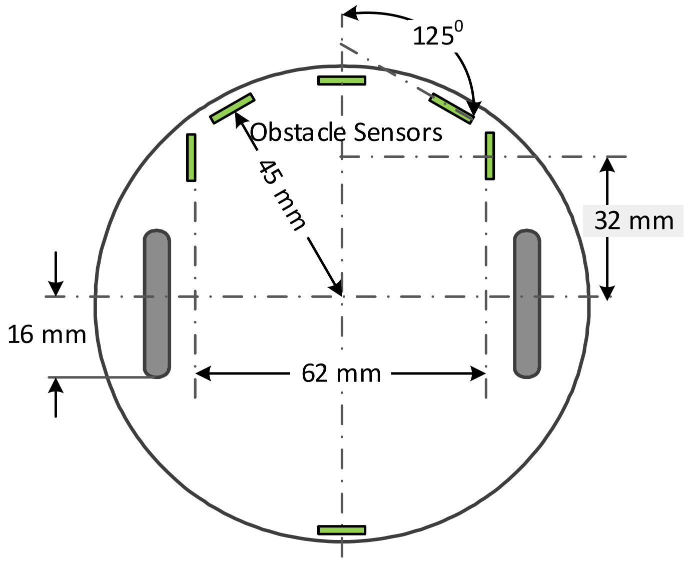


Figure 3-5 Proximity sensors physical layout

Figure 3-5 shows the orientation and positioning of the robots six proximity sensors. Five sensors provide the coverage towards the front of the robot and a sixth sensor provides feedback from the rear. The application notes available for the proximity sensors (VCNL3020) describe a cone shaped detection zone with an angle of ±40º and a maximum range of 200 mm as shown in Figure 3-6.

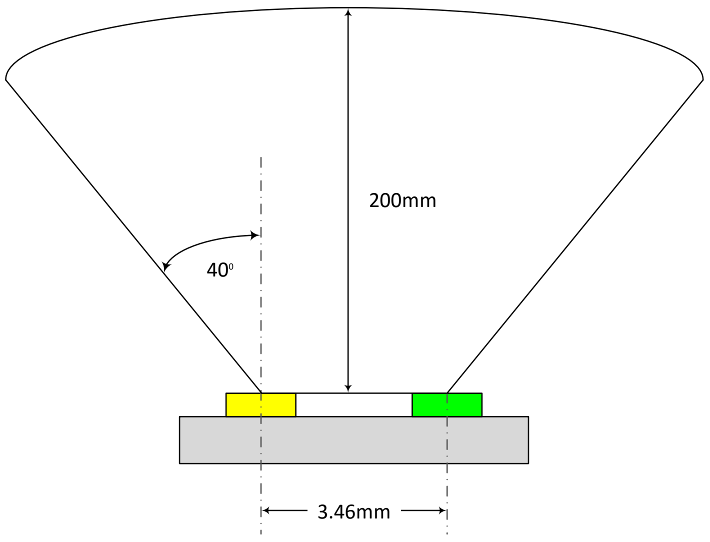


Figure 3-6 Proximity sensor detection view

The sensors have software configurable detection range, so for the simulation this detection range has been set to 50mm as shown in Figure 3-7. This provides ample distance to allow time for the robot to stop before a collision, and also allows the robot to navigate between closer obstacles. The sensors have been configured to provide a digital output.

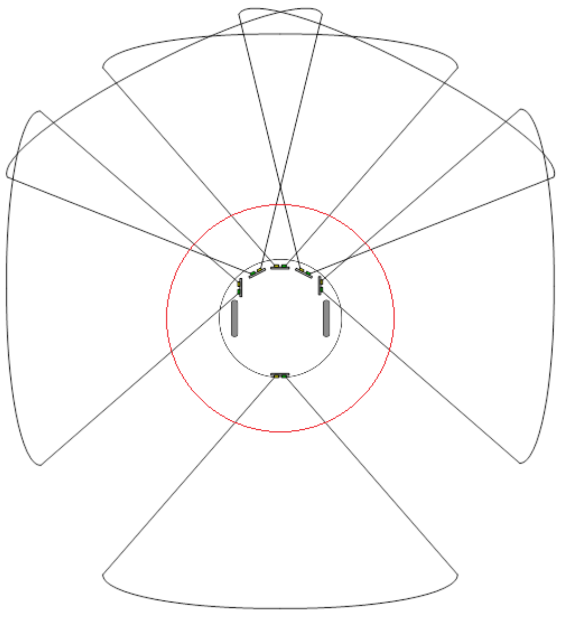


Figure 3-7 Maximum proximity detection area (red circle 50mm sensor range)

Proximity sensor mathematical model

For simulation purposes the sensors have been modelled as a straight line heading directly outwards from the perimeter of the robot in the direction of the sensor as shown in Figure 3-8.

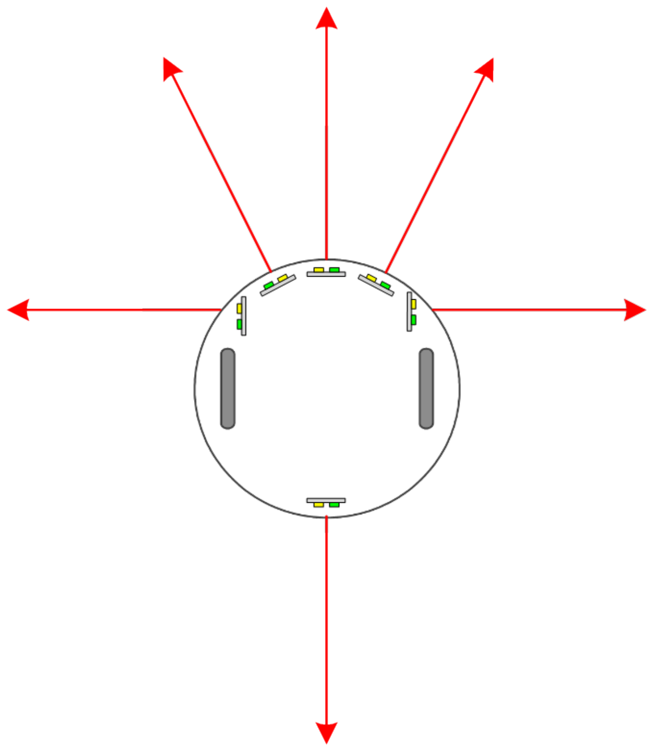


Figure 3-8 Simulated proximity sensor coverage

Reducing the detection area to a single line reduces the processing time required to check each sensor. In a real world operation the proximity detection operates at the speed of light only limited by the time it takes for the sensors to process the data and trigger an interrupt. In simulation the software needs to progressively check the sensor regions one small section at a time. For example if the sensor covers an area of $1000mm^2$. Then because of the scale of the grid in the simulation, the software would have to check 6000 locations. This increases the evolution time dramatically.

A formula has been developed to simulate the checking process of each of the proximity sensors. This formula works by calculating each X and Y coordinate in a straight line out from the proximity sensors to a maximum range of 50mm.

Equation 9 and Equation 10 provide the increments along the X and Y axes which are used when scanning along each sensors object detection path.

$$dx = \sin(Robot\ heading + sensor\ angle) \qquad \text{Equation 9}$$

$$dy = \cos(Robot\ heading + sensor\ angle) \qquad \text{Equation 10}$$

Where:

dx is the increment on the X axis,

dy is the increment on the Y axis,

Robot heading is the current heading of the robot,

Sensor angle is the angle of the sensor in relation to 0 degrees on the robot.

A list of the sensors and their respective angles is shown below in Table 6.

Table 6 Sensor positions and their respective angles

Sensor #	Position	Angle (Front = 0°)
1	Left side	270°
2	Left front	330°
3	Front	0°
4	Right front	30°
5	Right side	90°
6	Rear	180°

The left and right side sensors are not positioned radially round the centre point of the robot; because of this, the starting coordinates of the scan line need to be calculated using Equation 11 and Equation 12. If the robot has a heading of 0 degrees and a location of (600, 600), then the resulting starting coordinates for the scan line will be (-31, 32) and (31, 32) for the left and right sensors respectively.

$$X = robotX + dx(31 * \cos(robot\ heading)) + dy(32 * \sin(robot\ heading)) \qquad \text{Equation 11}$$

$$Y = robotY + dy(31 * \sin(robot\ heading)) + dx(32 * \cos(robot\ heading))$$ Equation 12

Where:

X is the starting location for the scan cycle on the X axis

Y is the starting location for the scan cycle on the Y axis

50mm is the robots radius

8 is the distance the sensor is in from the perimeter

30 is the sensor offset from the Y axis

dx is the increment on the X axis

dy is the increment on the Y axis

The scan cycle for checking each position along the straight line from each sensor, starts at the coordinates (X, Y) calculated using Equation 11 and Equation 12 and scans in 1mm increments. For each step, the simulation cross references the coordinates with the arena information to determine if an object is within the range of the sensor. If an object is detected then the sensor output is true otherwise, it remains false.

3.1.2.2 Light following systems

Light sensing hardware

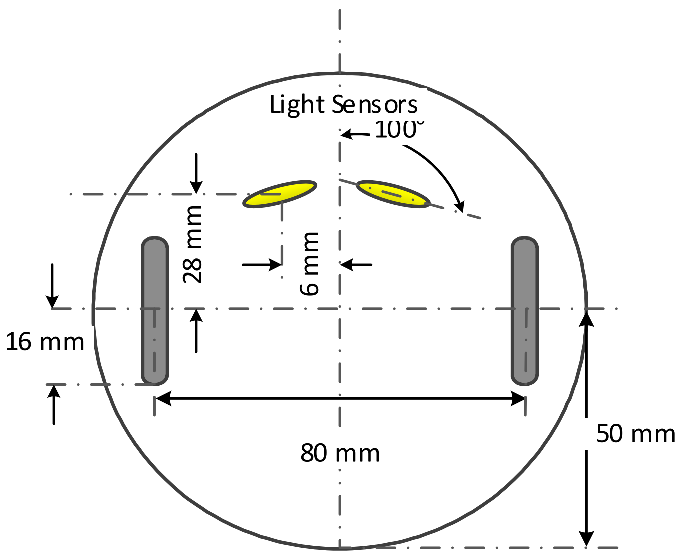


Figure 3-9 Light sensor physical layout

The robot has two light sensors; the sensors are located on the top and near the centre of the robot as shown in Figure 3-9. The sensors and are rotated 10 degrees down from the X axis which provides 20 degrees of separation between the sensors. This physical layout provides amplification between the two light sensor readings, therefore increasing the measurable difference in light intensity on each sensor when looking at a single point light source. If the two sensors were both pointing in the same direction, then the light intensity seen by each sensor would be very similar.

Light sensor mathematical model

The light intensity value measured by the light sensors is dependent on the angle of the sensor in relation to the light source. If the sensor is pointing directly at the source, then the value will be at its maximum. If the sensor is in the same position but pointing in the opposite direction then the value will be at its lowest. As the sensor rotates towards the light source the values obtained from the sensor will gradually increase. This property means that the model for the light sensors only relies on the orientation or heading of the sensors, which can easily be calculated based on the actual heading of the robot.

The following set of equations determines the direction of the light source in relation to the robots heading. Based on this the angles between the light source and the light sensors are determined (left and right sensor error).

$$Light\ direction = \tan^{-1}\left(\frac{Robot_X - Light_X}{Robot_Y - Light_Y}\right) \qquad \text{Equation 13}$$

$$Robot\ heading\ error = ABS(Lightdirection - RobotHeading) \qquad \text{Equation 14}$$

$$Left\ sensor\ error = Heading\ error + 10 \qquad \text{Equation 15}$$

$$Right\ sensor\ error = Heading\ error - 10 \qquad \text{Equation 16}$$

A restriction was put in place to allow for a more realistic simulated light intensity, this was done by implementing a maximum displacement angle from the light source. It has been determine that if the angle between the light sensor and the light source is more than 90 degrees then the calculated intensity would be 0. Using Equation 17 the light intensity for each sensor can be calculated. The resolution of the sensors is determined by the quantization level. A quantization level of nine results in a light intensity ranging from zero to eight.

$$Light\ intensity = (QL - 1) - \left(\frac{ABS(sensor\ error)}{90}\right) * (QL - 1) \qquad \text{Equation 17}$$

Where:

QL is the quantization level, selectable from within the GUI,
Sensor error is the value obtained in Equation 15 and Equation 16,
90 is the maximum sensor error before the output value will return zero

An adjustable quantization level has been implemented. This is used to explore the effects caused by different sensor quantization levels on the evolution process and controller performance.

Example calculations

In Figure 3-10 the robot has a heading of 90 degrees, in this case the light source is directly in front of the robot and the quantization level is set to nine (0-8).

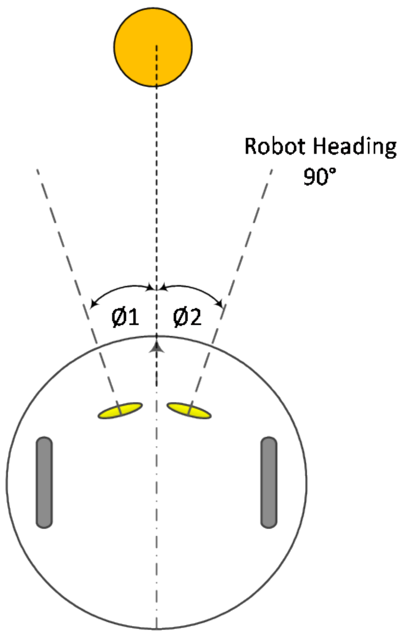


Figure 3-10 Light calculation 90° heading

Left sensor error (Ø1) = 0 + 10 = 10

Right sensor error (Ø2) = 0 – 10 = -10

Left sensor intensity = $8 - \left(\frac{ABS(10)}{90}\right) * 8 = 7$

Right sensor intensity = $8 - \left(\frac{ABS(-10)}{90}\right) * 8 = 7$

As expected both sensors return a light intensity of seven, this is because the robot is facing directly at the light source.

Even though the robot is facing directly at the light source the sensors don’t achieve the maximum light intensity, this is due to the 10 degrees offset of each sensor

Figure 3-11 is an example of what light sensor values are obtained if the robot is not looking directly at the light. In this case the robot has a heading of 80 degrees and again the quantization level is set to nine (0-8).

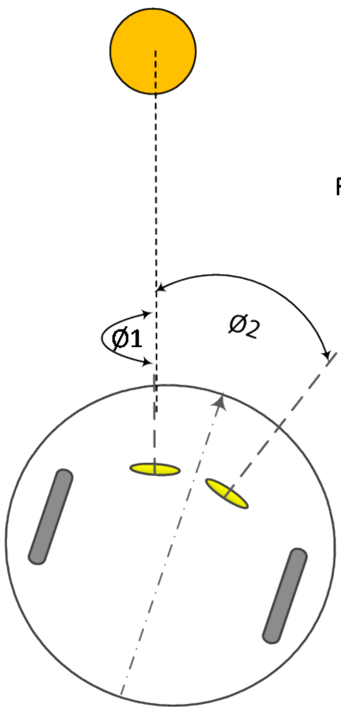


Figure 3-11 Light calculation 80° heading

Left sensor error(Ø1) = -10 + 10 = 0

Right sensor error(Ø2) = -10 – 10 = -20

Left sensor intensity = $8 - \left(\frac{ABS(0)}{90}\right) * 8 = 8$

Right sensor intensity = $8 - \left(\frac{ABS(-20)}{90}\right) * 8 = 6$

As expected the left light sensor returns a value of eight because it is facing directly at the light and the right sensor returns a value of six because it is facing 20 degrees away from the light.

The final example shown in Figure 3-12 shows the robot with a heading of 45 degrees and in this case both light sensors are pointing slightly away from the light source, however the right sensor is pointing further away.

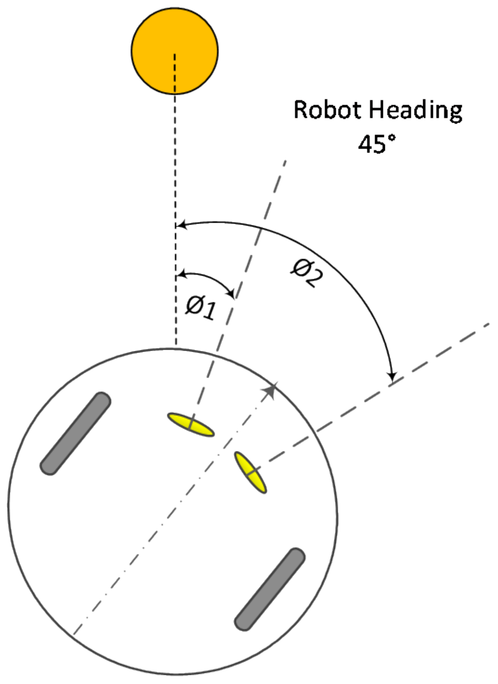


Figure 3-12 Light calculation 45° heading

Left sensor error (Ø1) = -45 + 10 = -35

Right sensor error (Ø2) = -45 – 10 = -55

Left sensor intensity = $8 - \left(\frac{ABS(-35)}{90}\right) * 8$ = 4.5

Right sensor intensity = $8 - \left(\frac{ABS(-55)}{90}\right) * 8$ = 2.5

The left sensor is closer to the light source and hence has a higher light intensity than the right sensor. For use in the simulation the light values are rounded, left = 5 and right = 3.

3.1.3 Arenas

This section contains the scaled models of the six object avoidance arenas used in the object avoidance sections of this research. Arenas A, B and C have been used to evolve the controllers, and arenas 1, 2 and 3 are the unfamiliar arenas which are used to test the adaptability of the evolved controllers. All the arenas in Figure 3-13 are scaled versions of the actual arenas used.

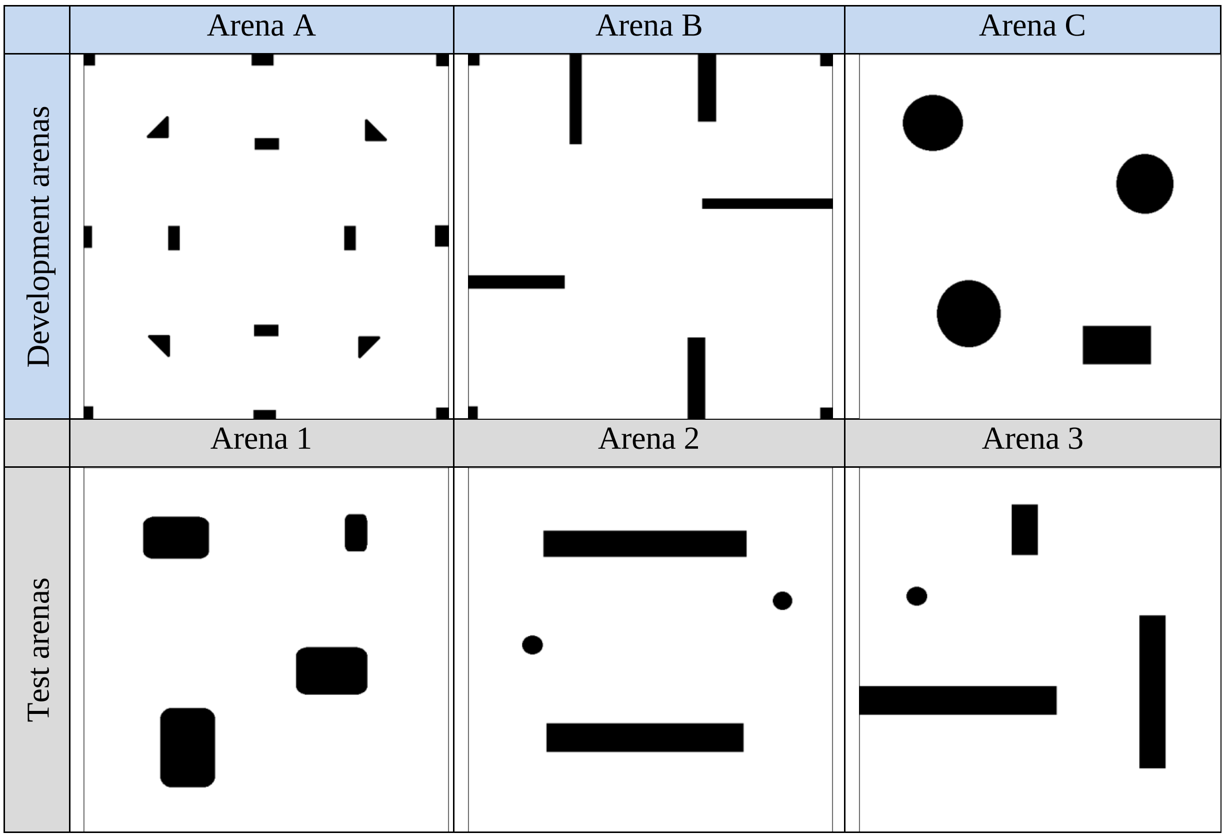


Figure 3-13 Development and test arenas

Genetic algorithm procedures

A general overview of the evolution process for each of the behaviours is outlined below.

- Initial population is created and tested.
- The following is repeated.
    - The parent population is used to create offspring.
    - The offspring are tested.
    - The new parents are selected from the offspring and old parents.

Initial population is created – 100 individuals are randomly generated, a number from 0 to 7 is randomly selected and assigned for every element in the LUT. The value stored in each element is decoded using the information shown below in Table 7 to determine the left and right motor directions and hence the motion of the robot.

Table 7 Robot motions based on value stored in the LUT elements

LUT Element content	Wheels(L/R)	Description
0	F/F	Forwards, robot travels in straight forward
1	F/S	Turn right forwards, robot will turn about right wheel CW
2	F/R	Rotate CW, robot pivots on the spot CW
3	S/F	Turn Left forwards, robot will turn about the left wheel
4	S/R	Turn right reverse, robot will turn about left wheel CW
5	R/F	Rotate CCW, robot pivots on the spot CCW
6	R/S	Turn left reverse, robot will turn about right wheel CCW
7	R/R	Reverse, robot travels straight backwards

Note, S/S has been removed from the table as it is not used in the simulation.

A detailed description of the LUT, ANN and EHW configurations used in each of the three behaviours is described in chapter 4, 5 and 6 of this thesis.

Initial population is tested – The initial population is put through a test process that determines how well each controller performs the desired behaviour. The simulation places the robot at a predefined starting location and uses the mathematical models of the robot to determine the required sensor information. The calculated sensor information is used in conjunction with the controller to determine the robots next motion as outlined in Table 7. The simulation again uses the mathematical models of the robot to determine the robots new position and orientation. The process continues until a predetermined time limit is reached.

On completion of the individual test, the robot performance is assessed. The assessment uses a fitness algorithm to determine a suitable fitness value based on how well the individual performed. Descriptions of the fitness functions for the behaviours are described in section 3.1.4 of this thesis.

The population is used to create offspring – The offspring reproduction method varies slightly depending on the type of chromosome that is being evolved. However, the method of reproduction used is the same for each controller. Individuals are paired up and random points in the chromosome are chosen. These areas provide the point at which combination between chromosomes occurs before the mutation process is undertaken. This creates two new individuals and doubles the population size.

The offspring are tested – The newly created offspring are then put through the same fitness evaluation as the initial population (parents) and suitable fitness values assigned to each.

The entire population is put through a selection process – A selection process is used to determine if the newly created offspring preform the required behaviour better than the parents from which they were created. If the offspring preforms better, then it will replace the parent. The selection process used throughout this research is a version of the well-known tournament based selection process. Using a tournament size of two, one offspring and one parent are compared. If the parent holds a fitness value which is greater than the offspring, then the parent will remain in the population and the offspring will be discarded. However if the offspring holds a fitness value greater than the parent, then the parent will be discarded and replaced by the offspring.

3.1.4 Fitness allocation

Light following

The fitness evaluation method used in the assessment of a controllers light following performance is determined by the distance the robot is from the light source at the end of the simulation A robot close to the light will achieve a high fitness and a faraway robot will achieve a low fitness. This method relies on a minimum runtime being used for each test, which is calculated as the minimum time required for the robot to reach the light if an optimum path is taken.

$$Fitness = 100 - \left[\frac{Fd}{Id} * 100\right] \qquad \text{Equation 18}$$

Where:

Fitness is the value assigned to the light follower controller that is under test
Fd is the final distance from the light source
Id is the initial distance from the light source

The formula works by multiplying the ratio of initial distance vs. final distance by 100. This results in a value ranging from zero for a robot that has reached the light source to 100 for a robot that did not move towards the light at all. This value is then subtracted from 100 to invert the results therefore zero is poor and 100 is good. If the robot turns away from the light, it is possible for the final distance to be greater than the initial distance. In this situation, the fitness it limited to zero and does not go negative.

The initial distance is calculated using the Pythagoras formula in which a right angle triangle can be drawn using two given coordinates within the arena. The first coordinate in the formula is the robots starting location and the second is the location of the light source. Using these two coordinates a straight line can be drawn between the two points whose length is equal to the initial distance. The final distance is calculated using the same method as that used to calculate the initial distance, however the two coordinates used are the robots final position and the light source location. The equations to calculate the initial distance and the final distance are shown in Equation 19 and Equation 20 respectively below

$$Id = \sqrt{(xi - xl)^2 + (yi - yl)^2} \qquad \text{Equation 19}$$

$$Fd = \sqrt{(xf - xl)^2 + (yf - yl)^2} \qquad \text{Equation 20}$$

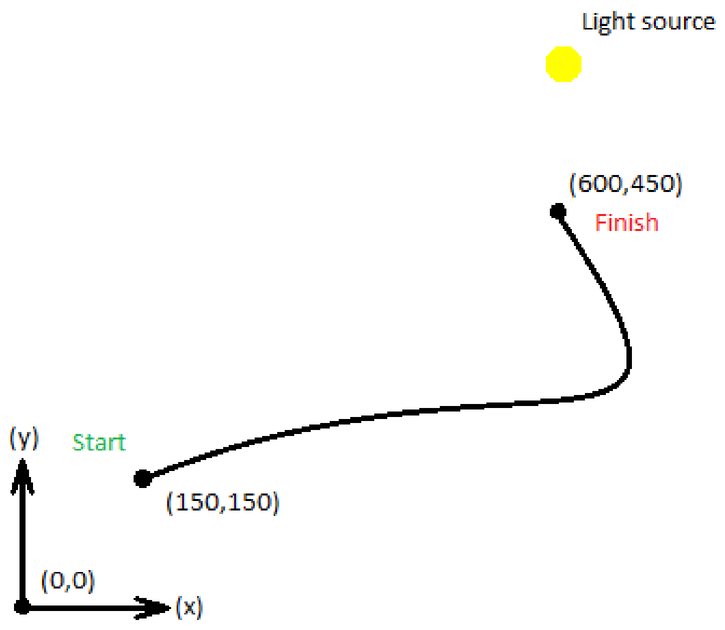


Figure 3-14 Example 1 - light follower fitness

It can be seen from the robot trajectory shown in Figure 3-14, that the robot started at the (X, Y) coordinate (150,150) and that its final location was at (600,450). The location of the light source in this case was at location (600,600).

Using Equation 18, Equation 19 and Equation 20 as described above, the fitness of the above example can be calculated as follows.

$$Id = \sqrt{(150 - 600)^2 + (150 - 600)^2} = 636\text{mm}$$

$$Fd = \sqrt{(600 - 600)^2 + (450 - 600)^2} = 150\text{mm}$$

$$Fitness = 100 - \left[\frac{150}{636} * 100\right] = 76.4$$

Based on the calculations above, an individual that preforms as shown in Figure 3-14 would receive a fitness value of 76.4%.

The next example shown in Figure 3-15 is of an individual that came very close to the light source when it finished, so it is expected that this individual should receive a higher fitness that the previous example in Figure 3-14.

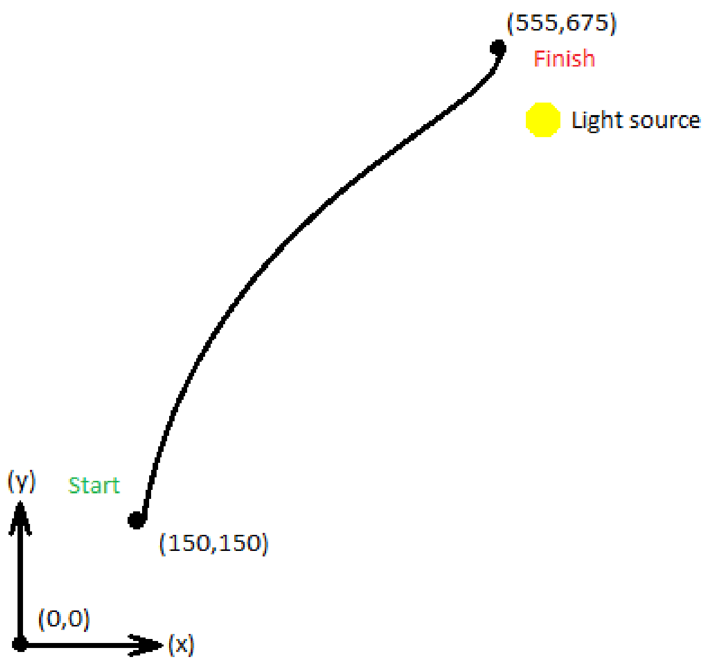


Figure 3-15 Example 2 - light follower fitness

$$Id = \sqrt{(150-600)^2 + (150-600)^2} = 636\text{mm}$$

$$Fd = \sqrt{(555-600)^2 + (675-600)^2} = 87\text{mm}$$

$$Fitness = 100 - \left[\frac{87}{636} * 100\right] = 86.3$$

Obstacle avoidance

A fitness function has been designed specifically for the object avoidance behaviour. This fitness function provides a numerical indication based on how well the robot preformed the required task.

It has been determined that an object avoidance controller is required to self-navigate through an unknown arena for a set period of time without coming into contact with a wall or objects. Based on these requirements the following evolution procedure has been developed.

1. The robots are started in the centre of the arena from eight different starting headings (0°, 45°, 90°, 135°, 180°, 225°, 270° and 315°). This forces the robots to explore different areas within the arena, and hence encounter different objects and situations in an attempt to produce a controller suitable for arenas other than the one which it has been evolved in.
2. The robot would be allocated a set time limit (20 seconds) after which the testing procedure would be halted.
3. If the robot encountered an obstacle then the simulation would be cut short and the resulting runtime was used in Equation 21 to calculate the fitness value.

$$OA\ Fitness \ = \frac{Run\ time\ count}{20} * 100 \qquad \text{Equation 21}$$

Where:

Run time count is the length of time the robot ran before it came in contact with an object.

Object avoidance Fitness is the value allocated to the individual under test ranging from 0 to 100.

From this method it was predicted that a robot with a run time that matched the allowed runtime of 20 seconds, would be adequately preforming obstacle avoidance behaviours. The resulting trajectory of the robot is shown below in Figure 3-16.

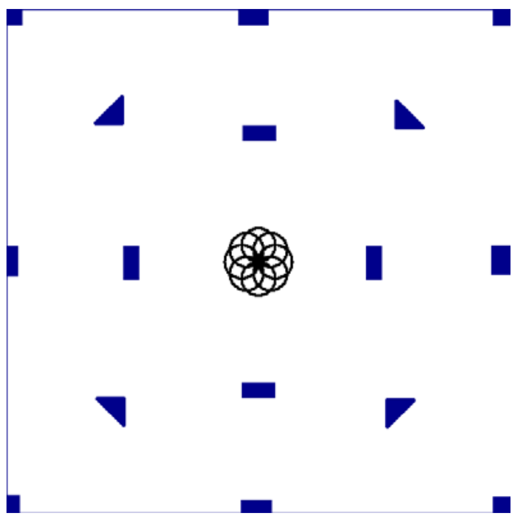


Figure 3-16 Object avoidance trajectory with only runtime fitness

From observations made over several different test runs like the one shown in Figure 3-16, it became clear that a different method was required for determining a more suitable fitness function. The initial requirement was that the robot needed to run for a set length of time without hitting any objects, the problem with this method was that the robot was able to do this and achieve maximum fitness by simply driving in small circles without even coming close to an obstacle.

Based on the conclusions from the previous tests it was decided that the fitness function required an influence to force the robot to explore the arena. So with this in mind the following modifications were made to the initial test procedure.

1. The arena was divided into 16 sections of equal size. Whenever the robot entered one of the 16 previously unexplored sections, that section was marked as explored.
2. Then as part of the fitness calculation after the test had finished, the number of explored sections was used to adjust the fitness value allocated to that individual. An individual with a high exploration count would receive a higher fitness than that of an individual with a low exploration count.

$$Object\ avoidance\ \text{Fitness} = \left[\frac{\text{Run time count}}{20} * 50\right] + \left[\frac{\text{Arena explored count}}{16} * 50\right] \qquad \text{Equation 22}$$

Using Equation 22 as the fitness function, the robots began to demonstrate a higher level of object avoidance behaviours. A typical trajectory can be seen in Figure 3-17.

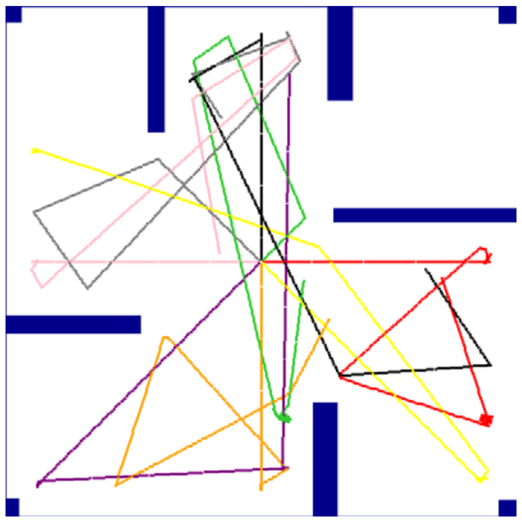


Figure 3-17 Object avoidance trajectory with arena exploration fitness

It was noted that due to the orientation and positioning of the obstacles it was impossible for the robot to explore some areas of the arena. This may not have caused an issue in the current arena however if an arena was chosen for the simulation that the object positioning only allowed the robot to explore 20% of its surroundings then there would be some effects on the achievable fitness.

Due to the previously described issues regarding the maximum fitness limitations, the following method for determining fitness was developed.

During the testing of each individual, a count linked to the robots continued movement was kept. This count was incremented whenever the individual failed to move a predetermined distance within a set time period (one second). This count was then used within the fitness function to determine an appropriate fitness for the individual, an individual with a high movement count would receive a lower fitness than that of an individual with a low movement count. The distance the robot would need to travel so as not to be penalized was determined to be 55% of the maximum distance the robot could travel if it was moving in a straight line. This time period in conjunction with the distance threshold allows the robot enough time to rotate and move about if obstructed by an obstacle without been penalized.

The fitness function developed for object avoidance controllers is outlined below in Equation 23

$$\textit{Object avoidance}\ \text{Fitness} = \left[\frac{\text{Run time count}}{20} * 50\right] + \left[\frac{20-\text{Movement count}}{20} * 50\right] \qquad \text{Equation 23}$$

Light following while avoiding obstacles

A fitness function has been developed to provide a suitable indication of the performance of controllers which have been designed to perform light following while avoiding obstacle behaviours.

Initially it was thought that the fitness function used for the light following behaviours could be used. However unlike the light follower testing procedure which had an easily calculable minimum runtime, the minimum runtime of the combined behaviour test was not easily calculable due to the addition of the obstacles to the robots arena. Because of this an alternative testing procedure was developed.

This testing procedure uses a predefined runtime of 10 seconds and during the runtime three variables are considered; robot path length, destination reached and collision detection.

Robot path length – at half second intervals the simulation calculates the linear distance the robot has travelled, the sum of all the recorded values provides the distance the robot has travelled. The path length is used as part of the fitness function to determine the final fitness.

Destination reached – the length of the robot path can only work as a fitness function if the simulation is able to determine when the robot reaches the light source. This is done by calculating the linear distance the robot is from the light at every simulated step. If the robot is within 20mm of the light source then the simulation will stop and the fitness is calculated.

Collision detection - at every simulated step the simulation checks the robot perimeter to determine if contact has been made with any of the objects in the arena. If contact has been made then the simulation will stop and calculate the fitness value. If a collision occurs then path length is set to the maximum (1200).

$$\text{Fitness} = 30 - \left[\frac{Fd}{Id} * 30\right] + 70 * \left[1 - \frac{Pl - MinPl}{1200}\right] \qquad \text{Equation 24}$$

Where:

Fd is the final distance from the light

Id is the initial distance from the light

Pl is the length of the path the robot took to get to the light source.

$MinPl$ is the straight line distance between the robot and the light

1200 is the maximum possible path length the robot can travel in the allowable run time.

3.1.5 Selection process

The selection process is a mechanism in which individuals are compared against one another and selected based on their assigned fitness values. The selection method that has been used throughout this research is a version of tournament selection. Tournament selection works on a group based section method. Each group has a set number of individuals which is known as the tournament size. The tournament size directly affects the selection pressure. A large tournament size will result in a higher selection pressure whereas a small tournament size will result in low selection pressure [44].

The tournament size for this research is two and is comprised of one parent and one offspring created by that parent. This provides low selection pressure, which allows individuals that are preforming well but may not be preforming as well as others, to remain in the population. A parent is only replaced when an off spring is produced which preforms better than the parent. If a larger tournament size was chosen, the selection pressure on the individuals would increase and individuals not performing as well may be lost, even if the individuals are still valid and contributing to the progression of the population.

The tournament selection process that has been implemented is graphically represented below in Figure 3-18. Two parents ‘A and B’ have created two offspring ‘a and b’, Parent ‘A’ and offspring ‘a’ make up one tournament group, and parent ‘B’ and offspring ‘b’ make up the other. In this example Parent ‘A’ has a fitness of 90 and offspring ‘a’ has a fitness of 50, so parent ‘A’ will remain in the population and offspring ‘a’ will be discarded. However Parent ‘B’ with a fitness of 60 will be replaced by offspring ‘b’ which has a fitness of 95.

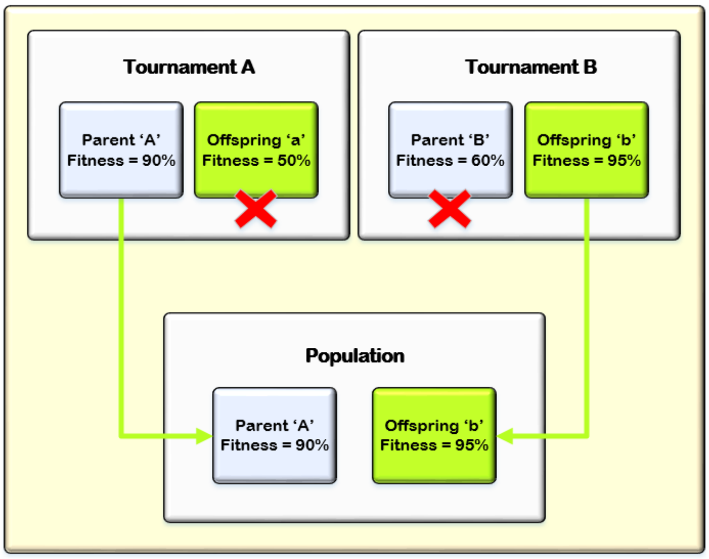


Figure 3-18 Tournament selection block diagram

3.1.6 Simulation procedure

This section describes the procedures used during the simulation of the controllers for the three robot controller behaviours.

Common procedures

The simulation of the three behaviours involves the following common procedures:

1. Check sensors – The simulation uses the mathematical models described in section 3.1.2 to determine the current status of the robots light sensors. The sensor outputs are used in conjunction with the controller under test to determine the resulting robot motion.
2. Calculate position and heading – The simulation uses the equations described in section 3.1.1 to determine the new robot position and heading based on the resulting controller output.

The remaining procedures used during the controller development are dependent on the controller under development and hence they have been outlined in separate subsequent categories.

Light following procedure

Position Initialisation: There are 4 starting locations used when evolving the light follower controllers, these locations are represented graphically below in Figure 3-19. From each position the robot also starts from two headings resulting in eight starting orientations for each individual test.

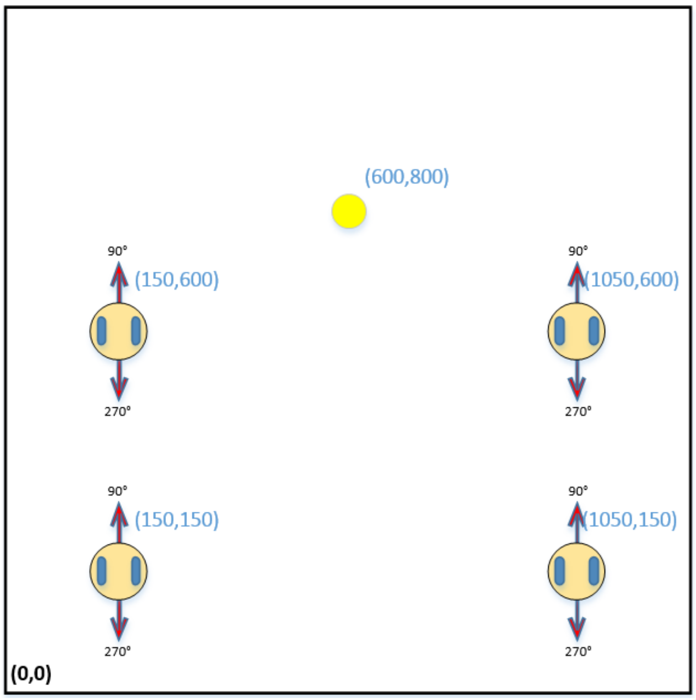


Figure 3-19 Light follower starting locations and headings

Determine runtime: The minimum runtime is calculated by determining the shortest distance between the robots starting position and the light source. This distance is divided by the maximum distance the robot can travel in a time period (6.1mm in 50ms). The result provides the number of simulation steps that the robot needs to reach the light source. An allowance has also been made for the time it takes for the robot to rotate too the light source. This rotation allowance is determined by calculating the difference between the robots starting heading and the light source direction and dividing this angle by the maximum rotation angle of the robot 8.5 degrees.

Object avoidance procedure

Position Initialisation : The robots are started in the centre of the arena from eight different starting headings (0°, 45°, 90°, 135°, 180°, 225°, 270° and 315°) as shown in Figure 3-20.

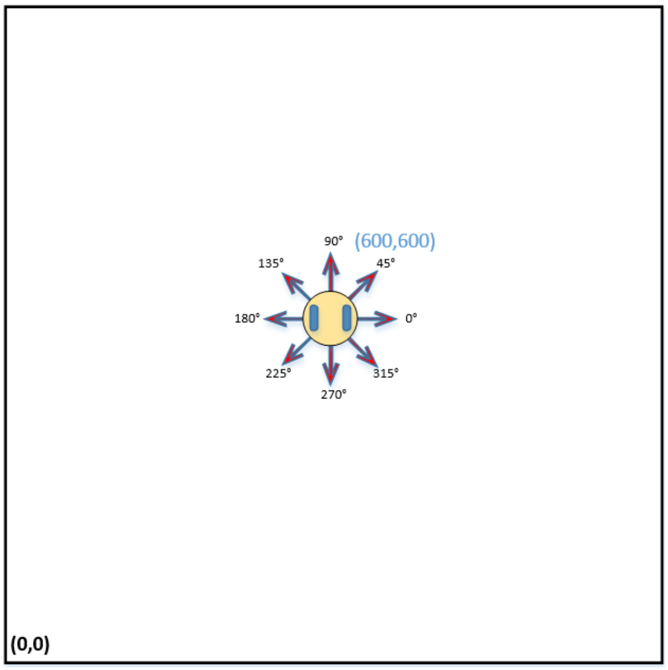


Figure 3-20 Object avoidance starting location and headings

Determine runtime: The eight tests last for 20 seconds and based on the values obtained in 3.1.1 this would allow the robot to travel a maximum of 2.4 metres. 20 seconds allows the robot to explore a large section of the 1.2m x 1.2m arena providing ample opportunity to encounter several different objects and hence thoroughly test the individual.

Collision detection – The simulation checks the perimeter of the robot to determine if a collision has occurred. If a collision has been detected then the simulation will stop, calculate the fitness then begin the test at the next scheduled starting heading. If no collision is detected then the simulation will continue.

Movement threshold – As explained in section 3.1.4 the fitness function for object avoidance controllers requires that the robot moves 67mm every second. Every second the software compares the robots new position to the previous and determines if the threshold has been reached. A count is kept for the number of time the controller fails to reach this threshold.

Light following while avoiding obstacles procedure

Position initialisation: The robots are started around the perimeter of the arena at the eight locations as shown in Figure 3-21 and each location has a unique starting heading.

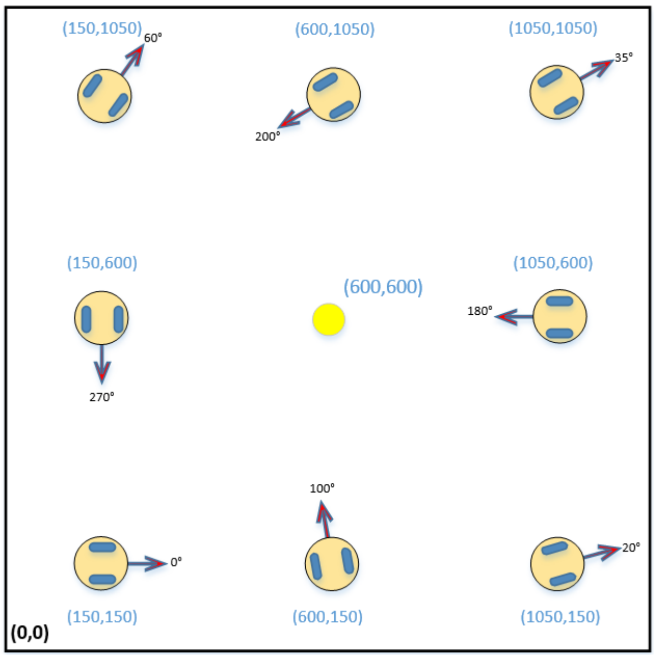


Figure 3-21 Light follower and object avoidance starting locations and headings

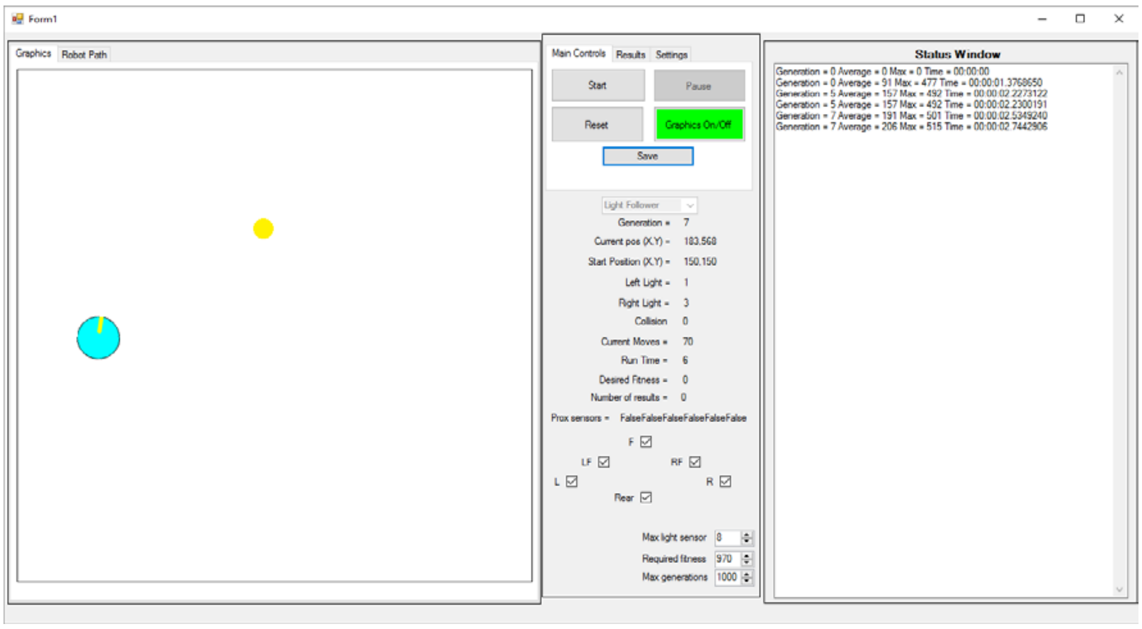
Determine runtime: The eight tests last for 10 seconds and based on the values obtained in 3.1.1 this would allow the robot to travel a maximum of 1.2 metres. This allows the robot the reach the other side of the arena which is more than enough time to navigate around an obstacle and reach the light source.

Collision detection – The simulation checks the perimeter of the robot to determine if a collision has occurred. If a collision has been detected then the simulation will stop, calculate the fitness then begin the test at the next starting location. If no collision is detected then the simulation will continue.

Path length calculation – During the simulation the software measures the distance travelled by the robot every half a second. These values are summed together to give the total path length and used to determine the robot fitness.

3.2 Graphical user interface

The GUI provides a means of monitoring and controlling the evolution of the controller. A status window contains a log of the current generation, average fitness, maximum fitness and the current elapsed time. The status window is shown to the right in Figure 3-22. Next to the status window in the middle of the GUI is the numerical status of the robot. This information contains the robots current position, the starting position and the sensor information.

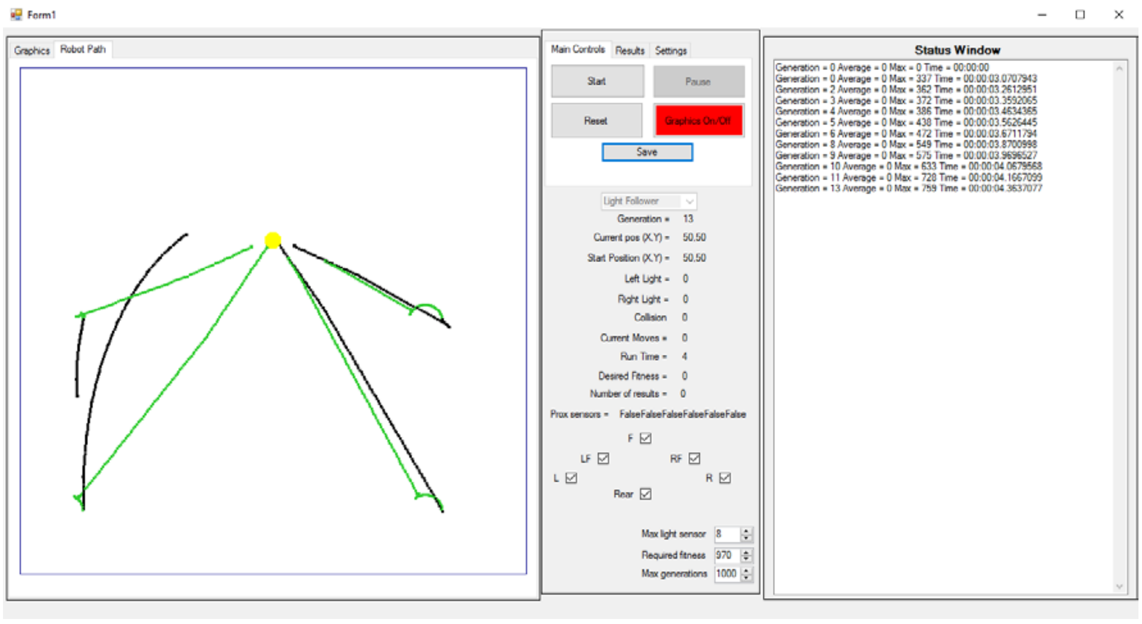


Generation = 7
Current pos (X,Y) = 183,568
Start Position (X,Y) = 150,150
Left Light = 1
Right Light = 3
Collision 0
Current Moves = 70
Run Time = 6
Desired Fitness = 0
Number of results = 0
Prox sensors = FalseFalseFalseFalseFalseFalse
F ☑
LF ☑ RF ☑
L ☑ R ☑
Rear ☑
Max light sensor 8
Required fitness 970
Max generations 1000

```
Generation = 0 Average = 0 Max = 0 Time = 00:00:00
Generation = 0 Average = 91 Max = 477 Time = 00:00:01.3768650
Generation = 5 Average = 157 Max = 492 Time = 00:00:02.2273122
Generation = 5 Average = 157 Max = 492 Time = 00:00:02.2300191
Generation = 7 Average = 191 Max = 501 Time = 00:00:02.5349240
Generation = 7 Average = 206 Max = 515 Time = 00:00:02.7442906
```

Figure 3-22 Graphical user interface

The status of the robot is shown using two methods. The first is a real time scaled model showing the robot in the arena (Figure 3-22). The second is a visual representation of the path taken by the robot (Figure 3-23).



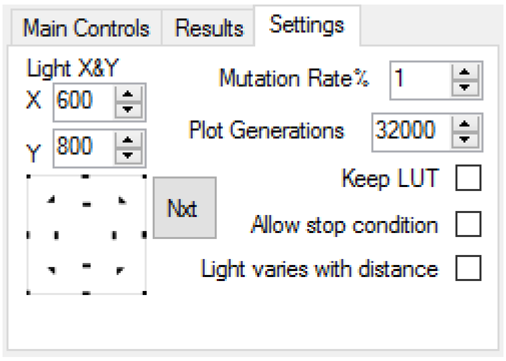
Generation = 13
Current pos (X,Y) = 50,50
Start Position (X,Y) = 50,50
Left Light = 0
Right Light = 0
Collision 0
Current Moves = 0
Run Time = 4
Desired Fitness = 0
Number of results = 0
Prox sensors = FalseFalseFalseFalseFalseFalse
F ☑ LF ☑ RF ☑ L ☑ R ☑ Rear ☑
Max light sensor 8
Required fitness 970
Max generations 1000

```
Generation = 0 Average = 0 Max = 0 Time = 00:00:00
Generation = 0 Average = 0 Max = 337 Time = 00:00:03.0707943
Generation = 2 Average = 0 Max = 362 Time = 00:00:03.2612951
Generation = 3 Average = 0 Max = 372 Time = 00:00:03.3592065
Generation = 4 Average = 0 Max = 386 Time = 00:00:03.4634365
Generation = 5 Average = 0 Max = 438 Time = 00:00:03.5626445
Generation = 6 Average = 0 Max = 472 Time = 00:00:03.6711794
Generation = 8 Average = 0 Max = 549 Time = 00:00:03.8700998
Generation = 9 Average = 0 Max = 575 Time = 00:00:03.9696527
Generation = 10 Average = 0 Max = 633 Time = 00:00:04.0679568
Generation = 11 Average = 0 Max = 728 Time = 00:00:04.1667099
Generation = 13 Average = 0 Max = 759 Time = 00:00:04.3637077
```

Figure 3-23 Trajectory view

Figure 3-23 shows an example trajectory during the evolution of the light follower controller. The robot is seen starting from four locations and two headings (green and black). The light source (yellow circle) is also shown in the location selected.

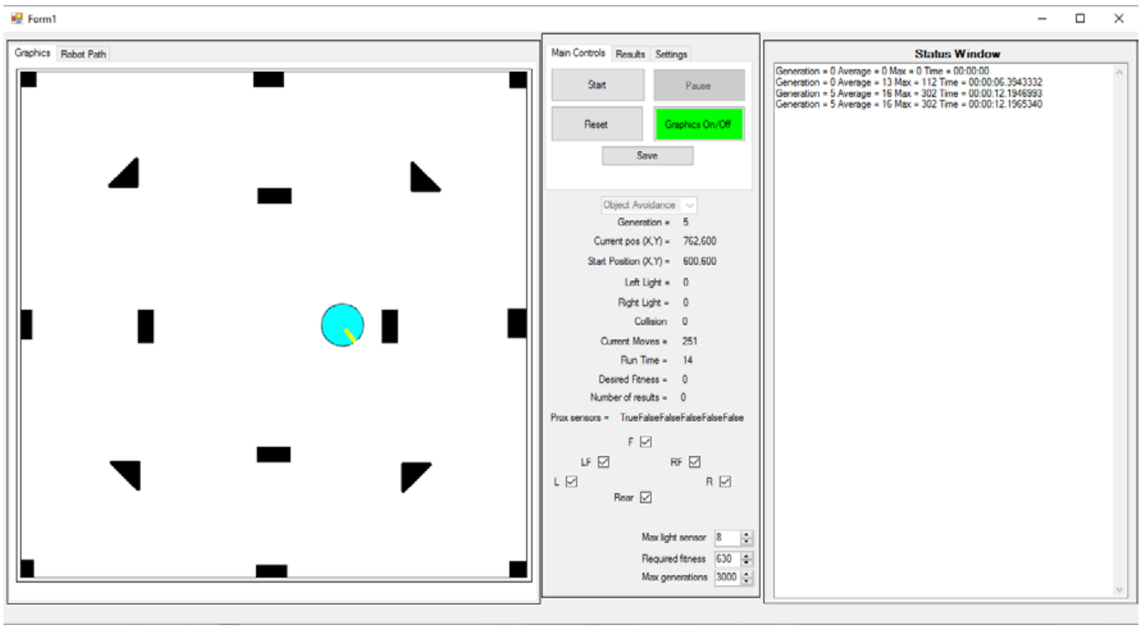
Adjustments to the robots arena can be made from within the settings tab in the centre section of the GUI. From here it is possible to change the location of the light source and select between prebuilt arenas for object avoidance behaviours. An example of the settings tab is shown in Figure 3-24



Light X&Y
X 600
Y 800
Mutation Rate% 1
Plot Generations 32000
Keep LUT ☐
Allow stop condition ☐
Light varies with distance ☐

Figure 3-24 GUI settings tab

If the object avoidance mode is selected, the GUI displays the selected arena in the display window. This provides a scaled visual representation of the robot and the obstacles in the arena.

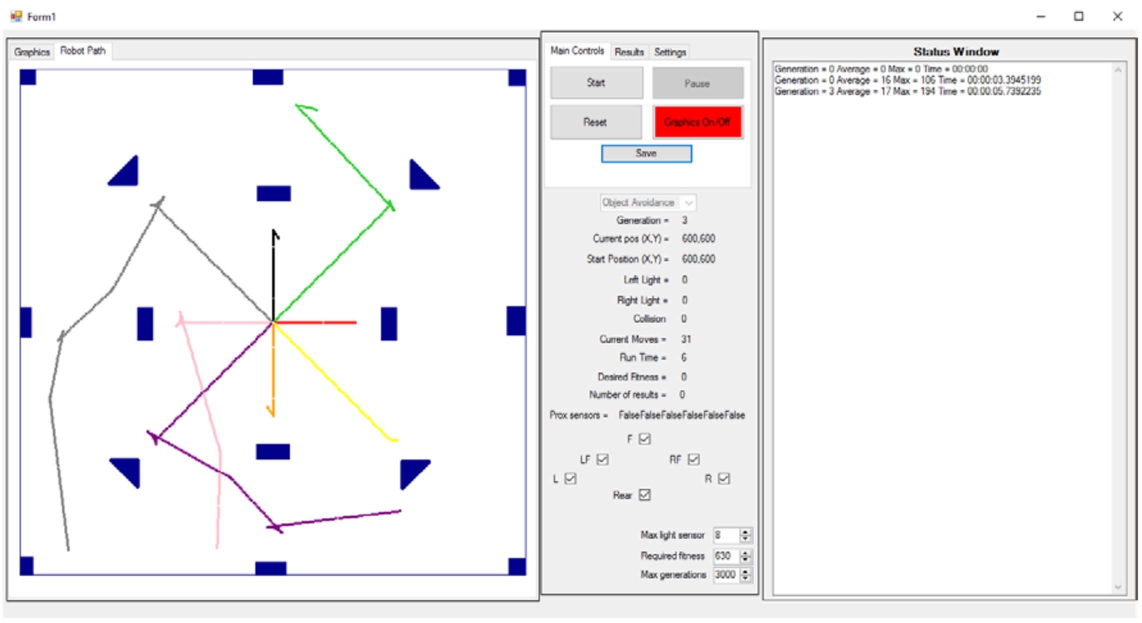


```
Generation = 0 Average = 0 Max = 0 Time = 00:00:00
Generation = 0 Average = 13 Max = 112 Time = 00:00:06.3943332
Generation = 5 Average = 16 Max = 302 Time = 00:00:12.1946993
Generation = 5 Average = 16 Max = 302 Time = 00:00:12.1965340
```

```
Generation = 5
Current pos (X,Y) = 762,600
Start Position (X,Y) = 600,600
Left Light = 0
Right Light = 0
Collision 0
Current Moves = 251
Run Time = 14
Desired Fitness = 0
Number of results = 0
Prox sensors = TrueFalseFalseFalseFalseFalse
Max light sensor 8
Required fitness 630
Max generations 3000
```

Figure 3-25 GUI set up in object avoidance mode

The view from the trajectory tab when operating in object avoidance mode is shown in Figure 3-26. Each starting heading is shown in a different colour and the obstacles are shown in blue.



```
Generation = 0 Average = 0 Max = 0 Time = 00:00:00
Generation = 0 Average = 16 Max = 106 Time = 00:00:03.3945199
Generation = 3 Average = 17 Max = 194 Time = 00:00:05.7392235
```

```
Generation = 3
Current pos (X,Y) = 600,600
Start Position (X,Y) = 600,600
Left Light = 0
Right Light = 0
Collision 0
Current Moves = 31
Run Time = 6
Desired Fitness = 0
Number of results = 0
Prox sensors = FalseFalseFalseFalseFalseFalse
Max light sensor 8
Required fitness 630
Max generations 3000
```

Figure 3-26 Object avoidance trajectory view

Chapter 4

Chapter 4: Artificial Neural Network evolution for use in robotic controllers

This chapter describes the experimental processes and results of the evolution of ANN controllers. Three individual controllers have been evolved to perform these three behaviours, light following, object avoidance and the combination of the two light following while avoiding obstacles.

Each chromosome is comprised of the following ANN parameters: the weights for the ANN inputs, two activation trigger points and two bias values. All of the parameter values range from -1 to 1 in 0.1 increments giving a total of 21 possible parameter values. The evolution methods and the simulation software are common to all the controllers being evolved and are fully described in Chapter 3,

4.1.1 ANN GA and software structure

The software for this research has been developed in Visual Studio using windows forms and C#. It runs the GA process, the robot simulation and data logging and GUI interface.

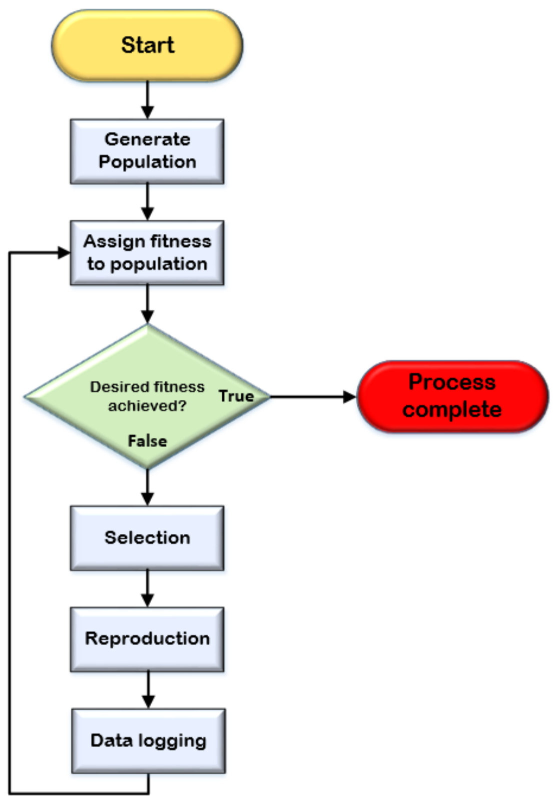


Figure 4-1 Simulation flow chart

Initialisation – The initial steps taken by the software are to setup the system. This involves generating the initial population with the random individuals. The software also generates the virtual arena which provides the proximity and light source information used during the evolution process.

Controller evolution – The core of the software is the evolution process, this is where the controllers are evolved until a solution is found.

- The software uses the mathematical models which are described in section 3.1 to determine the proximity and light sensor values. The sensor values are used in conjunction with the controller under test to determine the resulting robot motion. The robot motion and the mathematical model of the robot provide the new robot location.
- Fitness evaluation: where each individual is tested for a predetermined length of time or until the desired result is achieved. At this point a fitness algorithm is used to assign a fitness value to each of the individuals under test.

- Selection: where the population is put through a selection process which uses an elitist-tournament selection process to determine which individuals to keep or discard.
- Reproduction: where individuals selected to remain in the population are used to produce a new set of offspring. The offspring are then put through the same testing process and the evolution process continues until a satisfactory result is obtained.

Data logging – The information and results gained from the evolution process are stored in three formats to allow for easy analysis.

- The maximum and average fitness values are stored at regular intervals with the current number of generations in a text file.
- The maximum and average fitness values vs. generations are graphed and stored in an Excel file.
- The chromosome of the best individual is stored in a text file for further analysis.

4.2 Light following

4.2.1 Artificial neural network configuration

An ANN has been designed as a controller to perform light following behaviours. The neural network takes two sensory inputs and provides two motor control outputs. The inputs to the system contain the quantized light intensity which is proportional to the robots heading with respect to the light source. The methods and calculations used when determining the light intensity are described in section 3.1. The ANN is a two neuron, single layer network, with inputs to the ANN linked directly to the light sensors. The outputs of the ANN are used to drive the left and right motors. The activation output is a signed multistep function which allows three possible motor control options, forward, reverse and stopped (similar to the LUT controllers).

The illustration shown in Figure 4-2 is a representation of the ANN used for the light follower controller. The inputs from the left and right light sensors can be seen on the left of the diagram, these inputs are multiplied by the synapses weights represented by W11, W12, W21 and W22 then fed into the combinational block. The summation block then uses Equation 25 and Equation 26 to determine the raw output result for the left and right motors respectively.

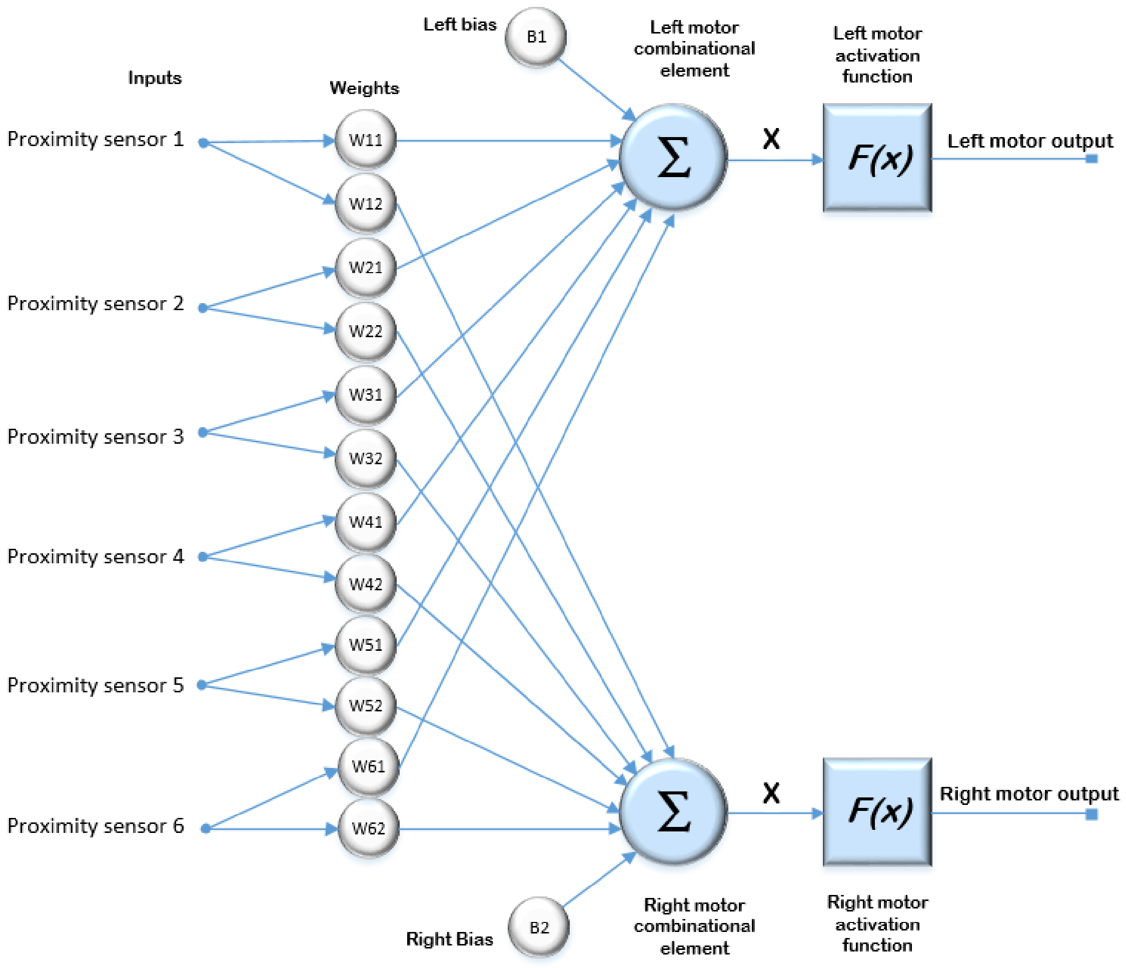


Figure 4-2 ANN configuration for light following

$$\text{Output left neuron} = [\textstyle\sum_{i=1}^{2} W1_i * X_i] + B1 \qquad \text{Equation 25}$$

$$\text{Output right neuron} = [\textstyle\sum_{i=1}^{2} W2_i * X_1] + B2 \qquad \text{Equation 26}$$

Where:

B is the bias input to the system.

W is the respective weight for each branch.

X is the light input value.

The bias to the ANN is required to overcome the initial state of the system when a light intensity value of zero is calculated for both light sensors. Without the bias the result of both motor summing equations would be zero, resulting in both motors being stopped, causing no more changes to the light intensity and causing the controller to fail. The bias overcomes this problem by introducing a constant to the equation that causes the robot to move when no light is encountered. Allowing the robot a chance of finding the light source.

The result from the summation block is fed into the activation function element which determines the output state of the neuron. For the ANNs developed in this research the output function is a signed step function as shown below in Figure 4-3.

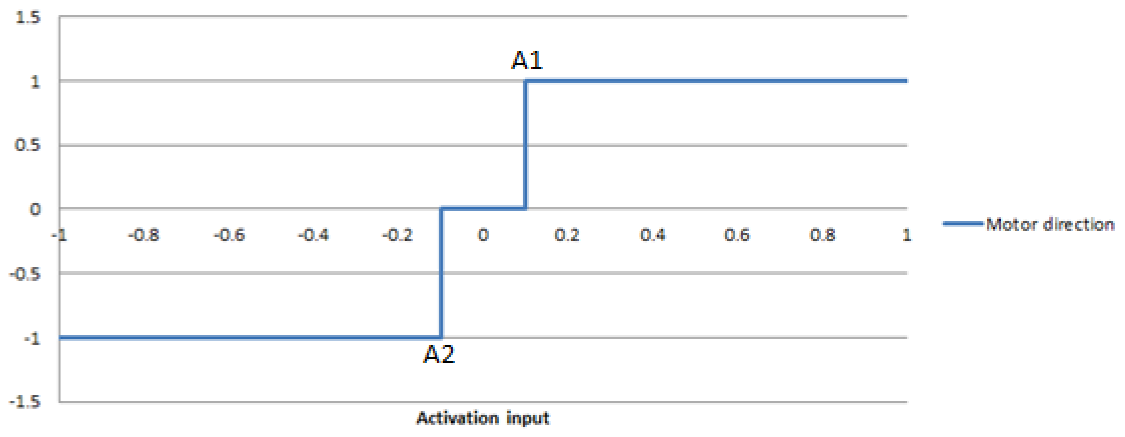


Figure 4-3 Light follower ANN output function

The input to the activation function is represented on the X axis while the resulting motor output is represented on the Y axis. The motor output clearly has three states, forward, reverse and stopped. The points which determine the transition between these states are the variable activation thresholds (A1 and A2).

4.2.2 Light following ANN chromosome

The chromosome for this system is a combination of four weights, two activation thresholds and two biases, W11, W12, W21, W22 A1, A2, B1 and B2. The weights, activation thresholds and bias values range from -1 to 1 with a resolution of 0.1 providing 21 possible values. The values are used directly in Equation 25 and Equation 26 to determine the input value to the activation function.

The GA search space for the ANN chromosome, which provides a value representing the number of possible combinations of weights, bias and threshold values, can be calculated using (Equation 27)

$$\text{Search space} = (\text{W})^{(\text{Synapses}+\text{bias}+\ \text{activationthresholds})} = (21)^{4+2+2} = 3.8 * 10^{10} \qquad \text{Equation 27}$$

Where:

W is the number of available weights (-1 to 1 in increments of 0.1)
Synapses is the number of input branch weights (4)
Bias is the number of bias`s in the system (2).
Activation thresholds are the number of points at which the output function can change (2).

4.2.3 Results

Artificial neural network performance

A set of experiments has been designed to fully evaluate the controller performance under a range of conditions. The experiments used four starting positions with two starting headings per position. The exact positions and headings have been chosen to provide worst and best case scenarios with the goal of providing the best chance of developing a universal controller. For all of the light follower experiments an upper fitness limit of 95% has been set, at this point the GA stops. 95% was chosen, because after this point, the population tends to converge and only very small levels of progress are made.

The performance of the controllers can be assessed by analysing the recorded trajectories of the path taken by the robot during development. These trajectories of the stages of evolution, as shown in Figure 4-4 show how the particular controller performed from each of the starting positions and headings. As the evolution progresses, the robot moves towards the light and the trajectories straighten giving a greater fitness. The trajectories shown below contain two starting headings. The first heading of 90 degrees is shown in black while the second heading of 270 degrees is shown in green.

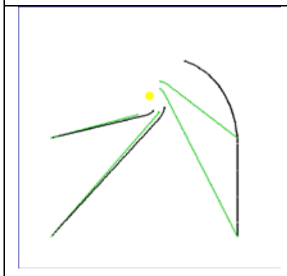
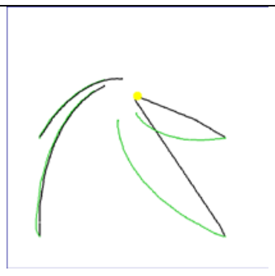
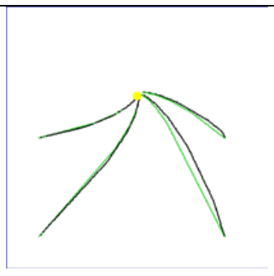
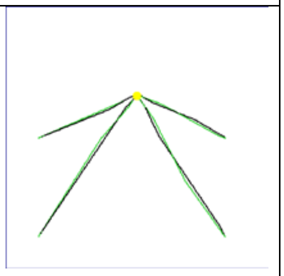
Fitness = 78%	Fitness = 82%	Fitness = 92%	Fitness = 97%
			

Figure 4-4 Example light follower trajectories and fitness percentage

Light level quantization

To analyse the effect of sensor quantization on the controller performance, a range of quantized light levels were evaluated, ranging from 2 to 9. These levels are also used for the light follower LUT and EHW controllers as described in sections 5.1.1 and 6.1.1. These values are compared with a controller that would use a 10 bit ADC with a quantization level of 1024. The resulting trajectories are shown in Figure 4-5 and Figure 4-6

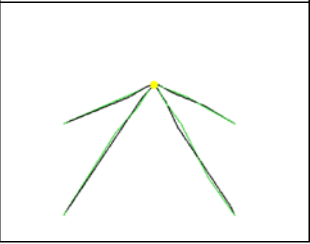
Quantization = 1024, Fitness = 97%


Figure 4-5 ANN raw quantization level trajectory

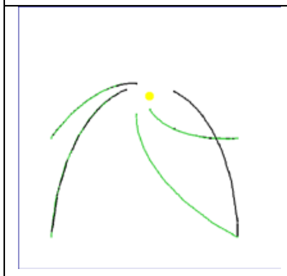
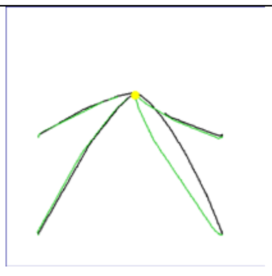
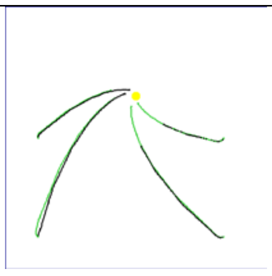
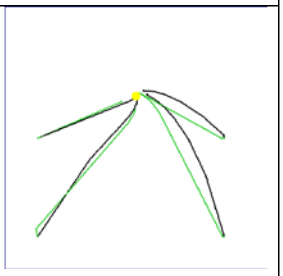
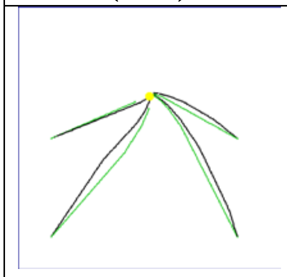
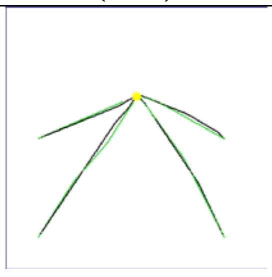
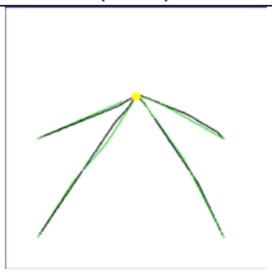
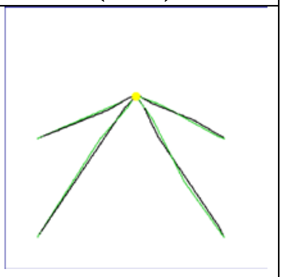
Quantization = 2 (79%)	Quantization = 3 (86%)	Quantization = 4 (82%)	Quantization = 5 (93%)
			
Quantization = 6 (95%)	Quantization = 7 (97%)	Quantization = 8 (97%)	Quantization = 9 (97%)
			

Figure 4-6 ANN Light level quantization

Analysis of the trajectories reveals two key attributes related to the level of sensor quantization. Firstly it can be seen that the controllers with a quantization level of five and below have difficulty in obtaining a high fitness, this is largely due to the reduced resolution from the sensors and operation of the fitness function. Secondly, increasing levels of quantization and hence increasing sensor resolution, the controllers can be seen taking the optimum path to reach the light source. A quantization level of seven and above matches that of a 10-bit ADC (1024 quantization level), indicating that only relatively low levels of quantization are required to create a good controller. The minimum quantization level to reach a fitness of 95% is six.

Evolution efficiency

The experiments in this section have been designed to explore the effects on evolution efficiency caused by a varying light quantization level. The evolutionary efficiency for these results is measured in generations so the points of interest will be analysed with respect to the generation of which they occurred.

Two criteria have been tested, firstly a high level of quantization will require more generations to realize a suitable solution and secondly a lower quantization level may not be able to fully realize a suitable solution. To explore these criteria eight individual ANNs with different light quantization levels were evaluated.

Table 8 contains the results from 8 levels of quantization and the result from the raw “un-quantized” sensor inputs. The controllers all exhibited light following behaviours, however only the controllers with a quantization level of six and above produced a controller capable of achieving the desired fitness (95%). The controllers that failed were unable to reach the light within the allowable runtime and hence achieved lower fitness.

Table 8 ANN Light level quantization results

Quantization level	Initial fitness (%)	Final fitness (%)	Generation at 95%
1024 (Raw)	74	97	24
2	76	79	---
3	75	86	---
4	74	82	---
5	75	94	---
6	79	95	348
7	79	96	19
8	79	97	18
9	80	97	7

Controllers with lower levels of quantization are unable to reach a fitness of 95% as they cannot recognize small changes in light level and are unable to navigate directly towards the light, while a quantization level of 6 takes 348 generations to find a solution. However, for quantization levels much higher than those chosen (Raw 1024), the generations required to reach 95% also begins to increase. The added sensor resolution adds a level of complexity to the controller which means the robot is able to pivot at a very small change in light intensity even if a pivot may not have been required. This reduces the chance of the robot reaching the light source and increases the number of required generations.

Comparison of maximum fitness for different quantization levels

A comparison of the maximum fitness trends for the eight quantization levels (Figure 4-7) reveals two interesting attributes. Firstly the final fitness is proportional to quantization level (Controllers with high quantization levels have higher sensor resolution and are more capable of reaching the light in the allowed time). Secondly, it can be seen that high levels of quantization decrease the initial fitness.

Based on the comparisons described above, it was decided that the optimum quantization level for this set of experiments was nine. At this level the controller was able to reach the desired fitness in only 7 generations and perform as required.

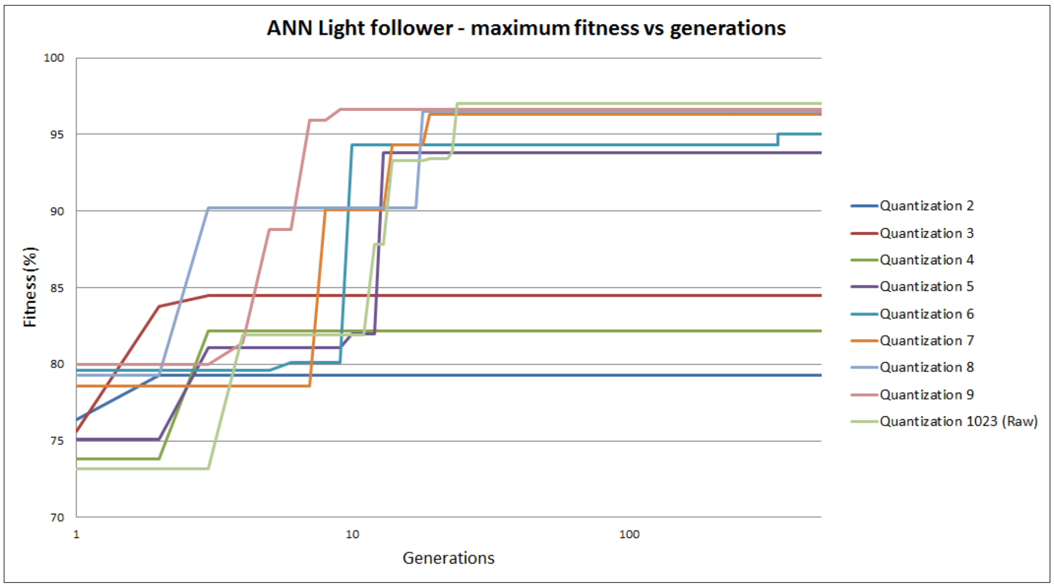


Figure 4-7 ANN Quantization comparison vs. generations

4.3 Obstacle avoidance

4.3.1 Artificial neural network configuration

An ANN has been evolved to perform obstacle avoidance behaviours. The input to the network is provided via six proximity sensors each of which provides a digital input, true when an object is detected and false when not. The modelling and simulation for these sensors is described in section 3.1. The outputs of the network are two signed multistep functions which drive the robot’s left and right motors. A visual representation of the ANN system used for this controller is above in Figure 4-8. On the left of the diagram the six proximity inputs to the system can be seen.

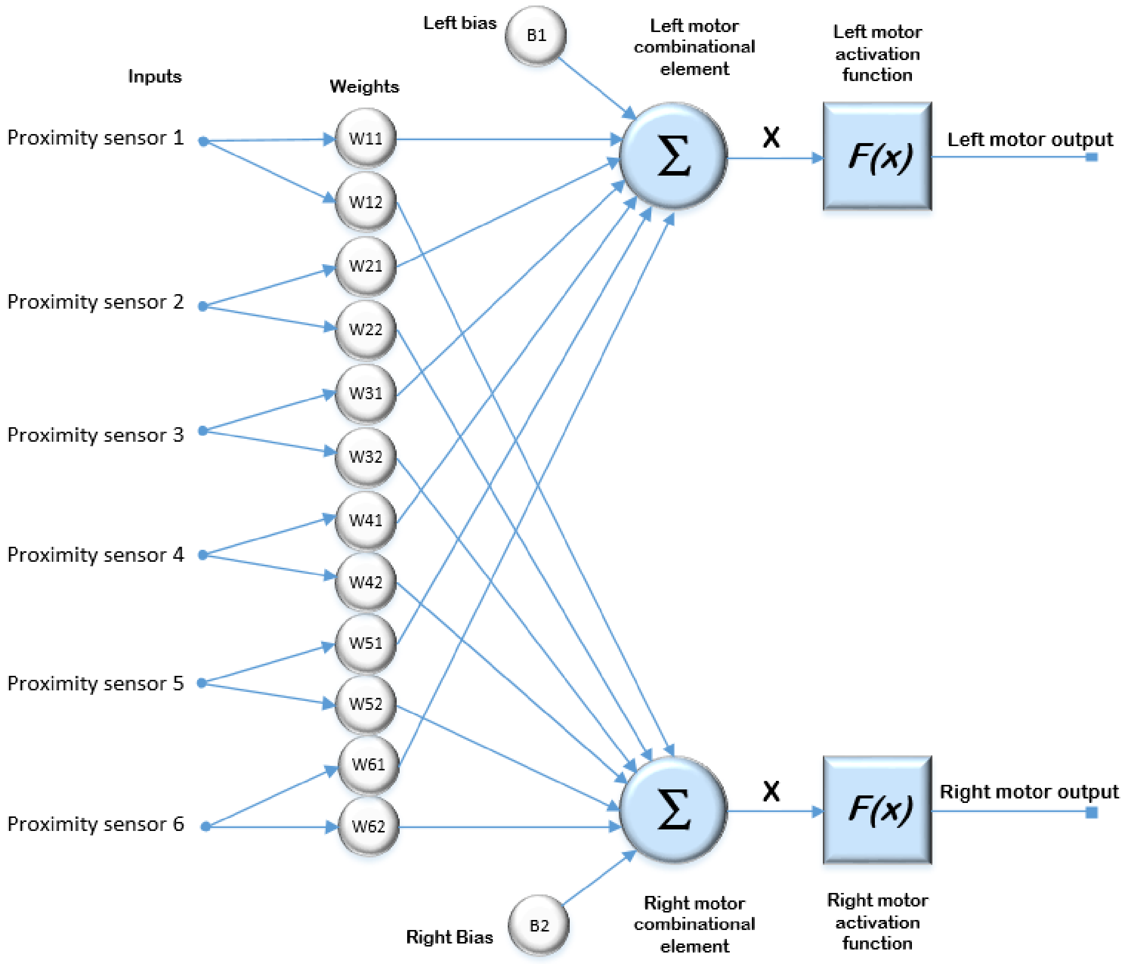


Figure 4-8 ANN configuration for obstacle avoidance

The outputs of these sensors (True or False) are multiplied by their respective synapses weights and passed to the summation element. Inside the summation elements Equation 28 and Equation 29 are used to determine the left and right raw output values.

$$\text{Output left neuron} = \left[\sum_{i=1}^{6} W1_i * X_i\right] + B \qquad \text{Equation 28}$$

$$\text{Output right neuron} = \left[\sum_{i=1}^{6} W2_i * X_i\right] + B \qquad \text{Equation 29}$$

Where:

X is the input from the proximity sensor one or a zero.
W is the branch weighting negative one to one.
B is the bias value.

The bias in this case is used to overcome the state of the system when all of the sensors are inactive. This would cause the output of the equations to be zero and because of the way the firing thresholds have been configured this would result in the robot not moving. The bias overcomes this problem by introducing a value to the equation that under these conditions would cause the robot to move and achieve a higher fitness value.

The summation block in the neuron is feed into the activation function element. This element serves as the decision maker and determines which state the output will be based on the input value. For the ANNs developed in this research the output function is a signed step function (Figure 4-3).

4.3.2 Obstacle avoidance ANN chromosome

The chromosome for this system is a combination of 12 weights, two activation thresholds and one bias, W11, W12, W13, W14, W15, W16, W21, W22, W23, W24, W25, W26 A1, A2 and B. The weights range from -1 to 1 with a resolution of 0.1 providing 21 possible weights per synapses. The weights are multiplied by their respective input and added together to create the input for the activation function.

The GA search space for the ANN chromosome, which provides a value representing the number of possible combinations of weights, bias and threshold values, can be calculated using (Equation 27)

$$\text{Search space} = (\text{W})^{(\text{Synapses}+\text{bias}+\ \text{activationthresholds})} = (21)^{12+2+2} \\ = 1.4 * 10^{21}$$ Equation 30

Where:

W is the number of available weights (-1 to 1 in increments of 0.1)
Synapses is the number of input branch weights (12)
Bias is the number of bias`s in the system (2).
Activation thresholds are the number of points at which the output function can change (2).

4.3.3 Results

A set of experiments has been designed to evaluate the use of an ANN as an object avoidance controller and the performance of the GA used to evolve it. The results obtained from these experiments are outlined in the two sections below, the first section focuses on the ANNs performance as a controller while the second section focuses on the GA performance.

The fitness function used to evolve the obstacle avoidance controller considers two variables, the robot's total runtime and the movement count. Both values are converted to a ratio with their respective maximums and used to calculate the fitness value. For all of the object avoidance experiments an upper fitness limit of 80% has been set. Due to the operation of the fitness function, a fitness value much above this point is unachievable. (A complete description of the fitness function can be found in section 3.1.4)

Artificial neural network performance

A set of experiments has been developed to evaluate the ANNs ability to perform in a range of different environments. The controllers are evolved from eight starting headings in three arenas with differing object configurations. The controllers from each of the experiments are then put into other arenas and the performance evaluated.

The first controller has been evolved in arena A and the progressing trajectories are shown in Figure 4-9. (A diagram of each arena used in these experiments can be found in section 3.1.2.1)

Note, the trajectories shown below contain eight starting headings.

0° = RED, 45° = Green, 90° = Black, 135° = Grey, 180° = Pink, 225° = Purple, 270° = Orange, 315° = Yellow

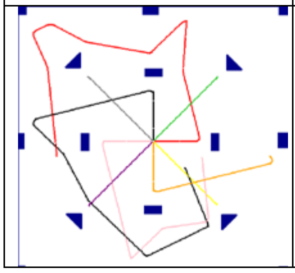
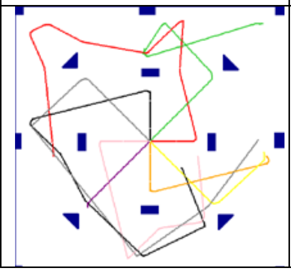
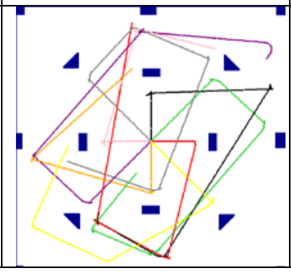
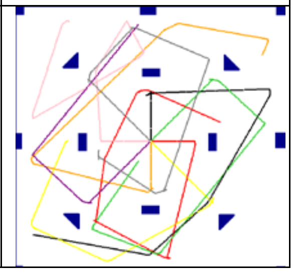
Fitness = 40%	Fitness = 58%	Fitness = 75%	Fitness = 80%
			

Figure 4-9 Arena A stages of evolution trajectories and the fitness obtained (ANN)

The set of images shown in Figure 4-9 are common trajectories achieved during the evolution cycle. As seen in the first two trajectories basic object avoidance techniques are evolved early on. However these techniques are limited to when the robot is approaching at a heading perpendicular to the obstacle. The ability to avoid obstacles at other angles is developed in the final stages of the evolution cycle. This process of evolution is largely due to the configuration and object placement of arena A, which has objects perpendicular to the robots starting headings meaning that this technique needs to be evolved first.

To compare the effect of arena selection during the evolution process on the performance of the ANN, two extra arenas have been used to develop two additional controllers. The trajectories for arena B and arena C are shown in Figure 4-10 and Figure 4-11

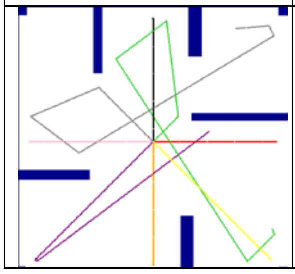
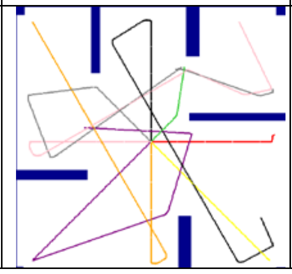
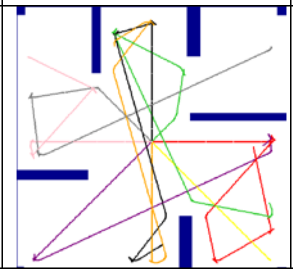
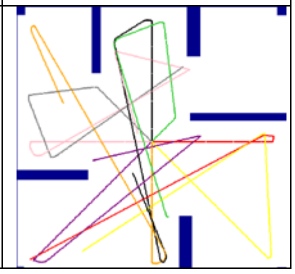
Fitness = 45%	Fitness = 62%	Fitness = 74%	Fitness = 81%
			

Figure 4-10 Arena B stages of evolution trajectories and the fitness obtained (ANN)

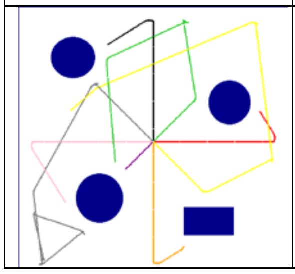
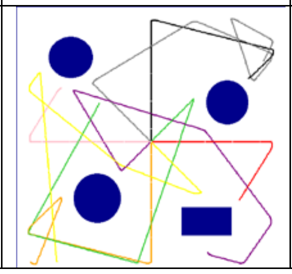
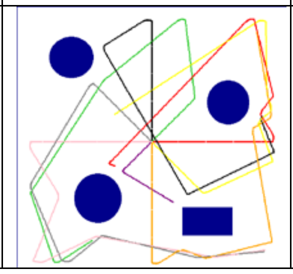
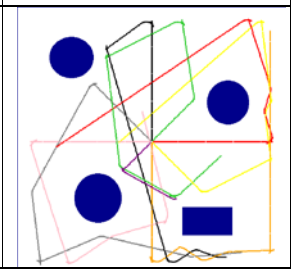
Fitness = 33%	Fitness = 58%	Fitness = 75%	Fitness = 81%
			

Figure 4-11 Arena C stages of evolution trajectories and the fitness obtained (ANN)

Similar evolution patterns to those found in the trajectories from arena A can be seen in the trajectories from arena B and arena C. In the initial stages of evolution the controller develops basic object avoidance behaviours which are fine-tuned with the trajectories becoming more refined in the later stages of the evolutionary cycle.

The evolved controllers from arena A, B and C have been evaluated using three unknown arenas. The trajectories from each controller in the unknown arenas are shown in Figure 4-12. It can clearly be seen from the trajectories shown that the controllers in most of the tests exhibit some object avoidance behaviours, however, these controllers are not performing to the same level of fitness as achieved in the original arenas.

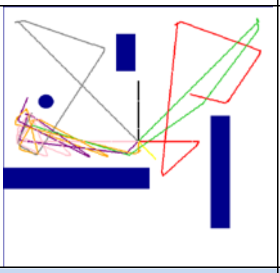
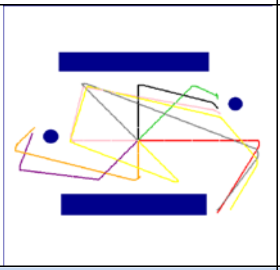
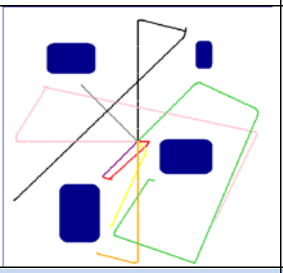
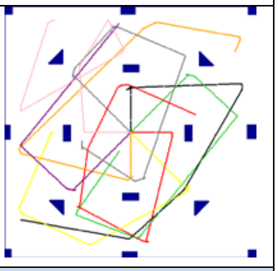
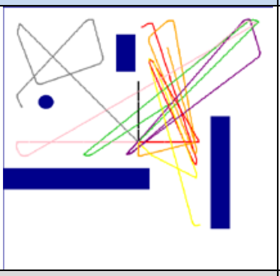
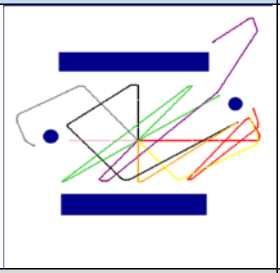
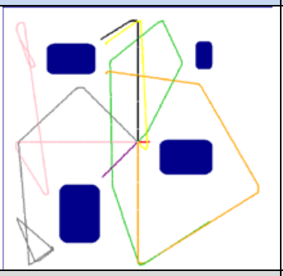
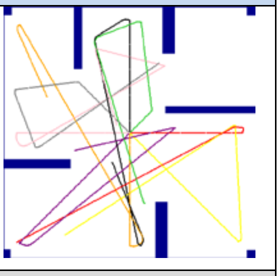
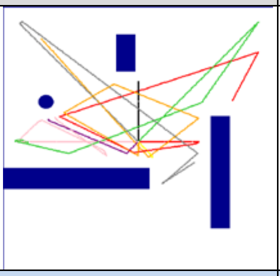
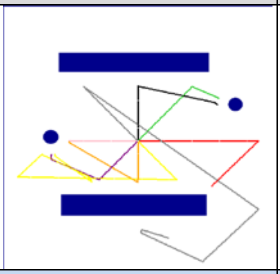
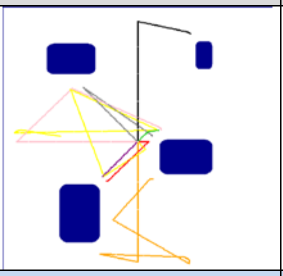
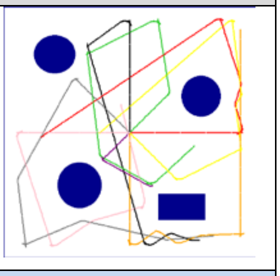
	Arena 1	**Arena 2**	**Arena 3**	**Original arena**
Controller A				
	Fitness = 64%	Fitness = 47%	Fitness = 44%	Fitness = 80%
Controller B				
	Fitness =77%	Fitness = 51%	Fitness = 56%	Fitness = 81%
Controller C				
	Fitness =57%	Fitness = 39%	Fitness = 40%	Fitness = 81%

Figure 4-12 Controller testing in unfamiliar arenas (ANN)

Multiple arena evolution

To overcome the incomplete evolution problem, a change has been made to the evolution process. For this test a controller (called ABC) was evolved in all three known arenas A, B and C. The average fitness from the combined arenas was used to determine the fitness of the individual. The evolved controller achieved a fitness of at least 80% in each known arena.

The resulting trajectories from the combined controller in the unfamiliar arenas are shown in Figure 4-13. When compared to the trajectories in Figure 4-12, a significant performance increase can be seen, evident in the longer runtimes and smoother trajectories, while the fitness in all arenas was above 80%

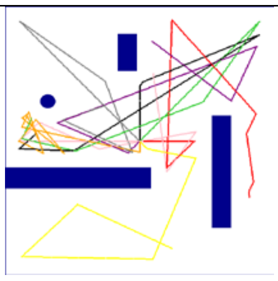
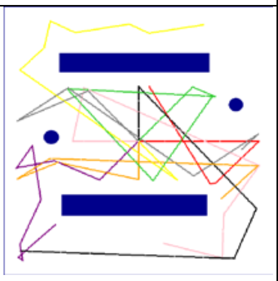
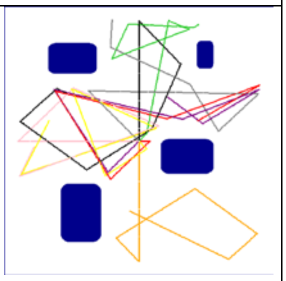
	Arena 1	**Arena 2**	**Arena 3**
Controller ABC			
	Fitness = 85%	Fitness = 81%	Fitness = 81%

Figure 4-13 Combined arena evolution test in unfamiliar arenas (ANN)

Evolutionary efficiency

This section focuses on the evolution efficiency in particular the number of generations required for a suitable result to be obtained. Controllers have been evolved in arenas A, B and C separately and in arenas A, B and C simultaneously. The results are shown in Table 9.

Table 9 Arena evolution results (ANN)

Arena	Initial fitness (%)	Final fitness (%)	Generation at 70%	Generation at 80%	Arena Complexity Less to more(1-4)
A	70	80	3	10	3
B	42	82	8	8	2
C	38	81	8	8	1
ABC	49	89	8	20	4

All four controllers achieve the desired fitness of 80%, with three of them exceeding it. A difference in complexity level can be seen between the arenas, where arena ABC is seen to be the most complex due to the number of generations required achieving 80%.

Maximum individual fitness trends for each arena configuration.

Maximum fitness trends shown in Figure 4-14, show that the arena complexity is directly related to the number of generations required to reach a suitable fitness. This also confirms the original order of arena complicities been C, B, A, ABC (from least complex to most) and a clear complexity difference can be seen for arena ABC which takes the longest to evolve.

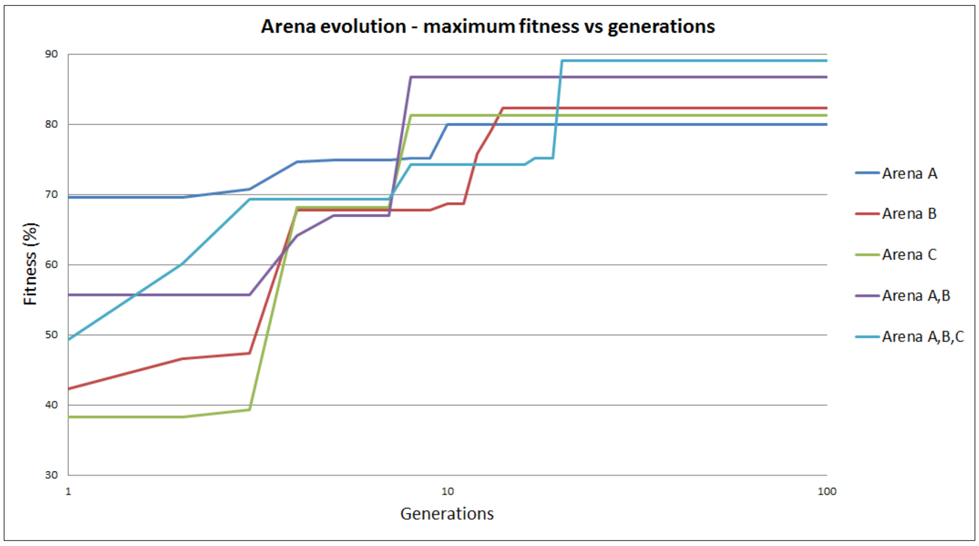


Figure 4-14 Maximum fitness comparison for each arena configuration (ANN)

4.4 Light following while avoiding obstacles

The focus for this section is to investigate scaling issues with an ANN as the complexity of the problem is increased. This is performed by comparing the results of monolithic and subsumption evolution.

4.4.1 Artificial neural network configuration

An ANN has been created to perform the combined task of light following while avoiding obstacles using monolithic evolution. The ANN contains eight inputs and two outputs. The inputs have been quantized as they were for the separate light following and object avoidance controllers and the outputs are configured as signed multistep functions to provide valid comparisons with the LUT and EHW controllers. Figure 4-15 is a representation of the ANN created for the combined behaviour controller. The system contains 16 individual synaptic weights which provide variable amplitude adjustment of the system inputs.

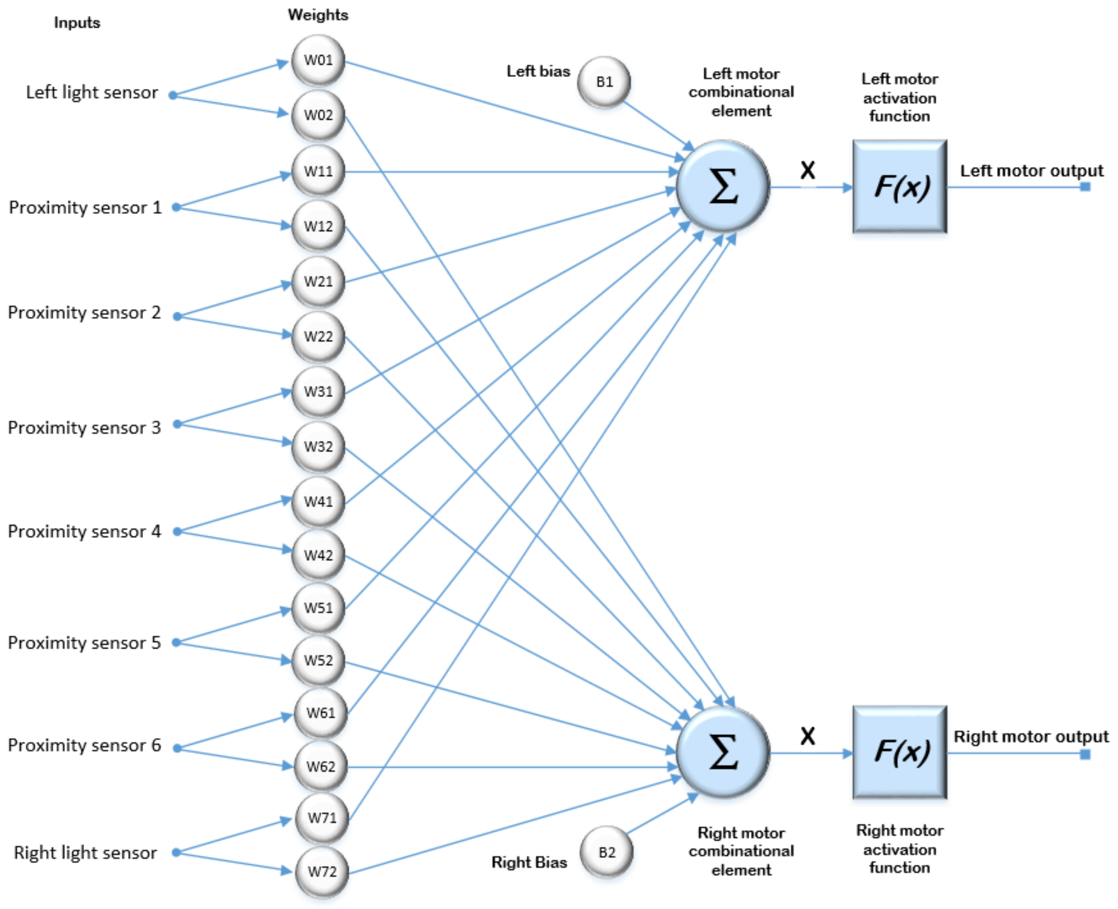


Figure 4-15 ANN configuration for light following while avoiding obstacles

Using Equation 31 and Equation 32 the weighted inputs are summed together and the result used in the activation function to determine the system output.

$$\text{Activation function left motor} = \left[\sum_{i=1}^{8} W1_i * X_i\right] + B \qquad \text{Equation 31}$$

$$\text{Activation function right motor} = \left[\sum_{i=1}^{8} W2_i * X_i\right] + B \qquad \text{Equation 32}$$

Where:

X is the input from the proximity sensors and the light sensors

W is the branch weighting (-1 to 1)

B is the bias value.

The bias values for this configuration are used to overcome the no light or no obstacle condition which results in a zero from both summation equations. The output from the activation function has been configured as a signed multistep function (Figure 4-3).

4.4.2 Light following while avoiding obstacles ANN chromosome

The chromosome for this system is a combination of 16 weights, two activation thresholds and one bias, W11, W12, W13, W14, W15, W16, W17, W18, W21, W22, W23, W24, W25, W26, W27, W28, A1, A2 and B. The weights range from -1 to 1 with a resolution of 0.1 providing 21 possible weights. The weights are multiplied by their respective input and summed together to create the input for the activation function.

The GA search space for the ANN chromosome, which provides a value representing the number of possible combinations of weights, bias and threshold values, can be calculated using (Equation 33)

$$\begin{aligned}\text{Search space} &= (\mathrm{W})^{(\text{Synapses+bias+ activationthresholds})} = (21)^{16+2+2} \\ &= 2.8 * 10^{26}\end{aligned} \qquad \text{Equation 33}$$

Where:

W is the number of available weights (-1 to 1 in increments of 0.1)
Synapses is the number of input branch weights (16)
Bias is the number of biases in the system (2).
Activation thresholds are the number of points at which the output function can change (2).

4.4.3 Artificial neural network subsumption evolution

Two ANN controllers are evolved separately and a switching controller employed to perform the combined task of light following while avoiding obstacles. This controller has been developed using subsumption evolution techniques where the complex task is broken down into individual less complicated tasks (light following and object avoidance) and evolved separately as described in 4.2 and 4.3. The evolved controllers are then combined using an algorithm which decides based on the sensory inputs, which controller is used and which is masked.

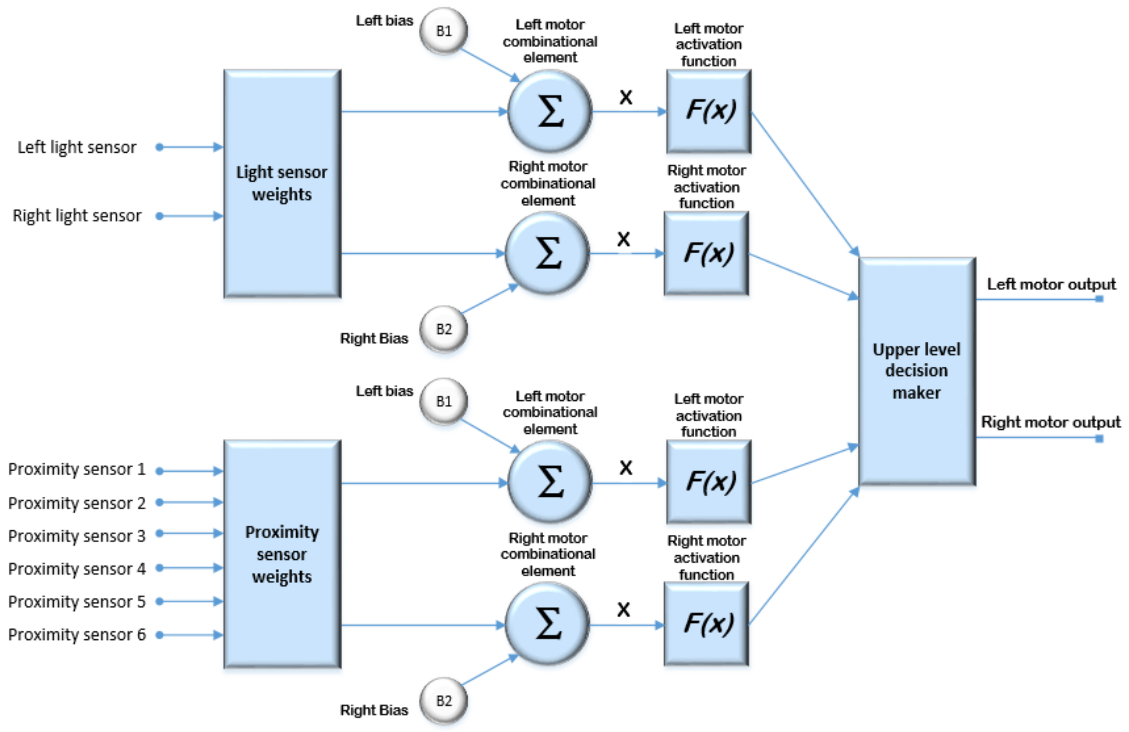


Figure 4-16 ANN subsumption diagram

Figure 4-16 illustrates the system design for the subsumption controller. The decision making block has highest level of control and has the ability to mask and unmask the inputs from the individual controllers. If an obstacle is detected then the obstacle avoidance controller is unmasked until the robot is clear of any obstacles, at which point after a small delay the obstacle avoidance controller is masked and the light follower controller is unmasked.

4.4.4 Results

A set of experiments have been designed to evaluate the ANN and the GA performance when controllers are evolved for light following while avoiding obstacles. Each individual is placed in an arena with obstacles and a light source. Each individual is tested in eight starting locations each with a unique heading. Eight controllers have been evolved, four using the monolithic and four using subsumption techniques. The evolved controllers performance and evolutionary efficiency are compared.

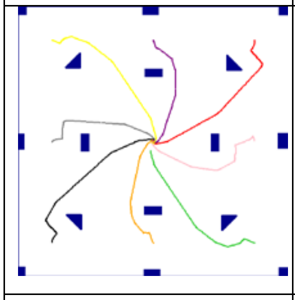
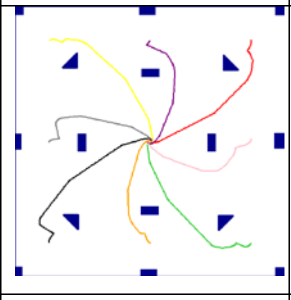
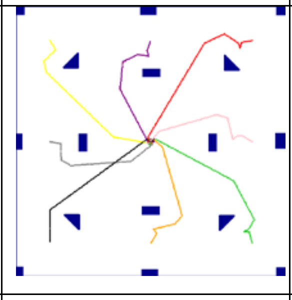
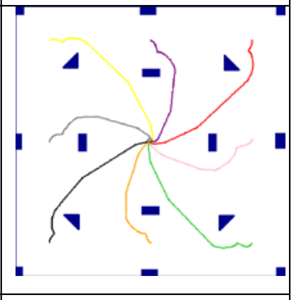
The fitness function used to evolve the ANNs for the combined behaviours considers two variables; the robots distance from the light at the end of the test and the length of the path taken. (A complete description can be found in section 3.1.4).

Note, all of the experiments are undertaken using arena A and an upper fitness limit of 80% or 3000 generations has been set for the monolithic evolution. A controller receiving a fitness of 80% is considered to be a good controller due to the way the fitness function operates.

Monolithic evolution results

The four controllers shown in Figure 4-17 perform to a very high standard and are fully developed in the allocated generation limit of 3000. The controllers are seen to be moving towards the light source and navigating around any objects encountered. In doing so, the robot is able to reach the light source and maintain an optimum path length. The final fitness achieved for each controller is indicative of a good result due to the operation of the fitness function.

As shown in Figure 4-7, the average number of generations required to reach a desirable result was 1120, substantially more than when the behaviours were evolved separately. The evolution process steadily increased until a fitness of about 35% was reached and after this the GA progress reduced noticeably. The reduction in progress was found to be the point where the controller had managed to reach the light source. However in order to progress past this point it was required that the path to the light be optimized.

Controller 1	Controller 2	Controller 3	Controller 4
			

Fitness = 83%
 	
Generations = 1025
	Fitness = 90%
	
	Generations = 850
		Fitness = 80%
		
		Generations = 495
			Fitness = 82%
			
			Generations = 2116

Figure 4-17 Monolithic controller trajectories (ANN)

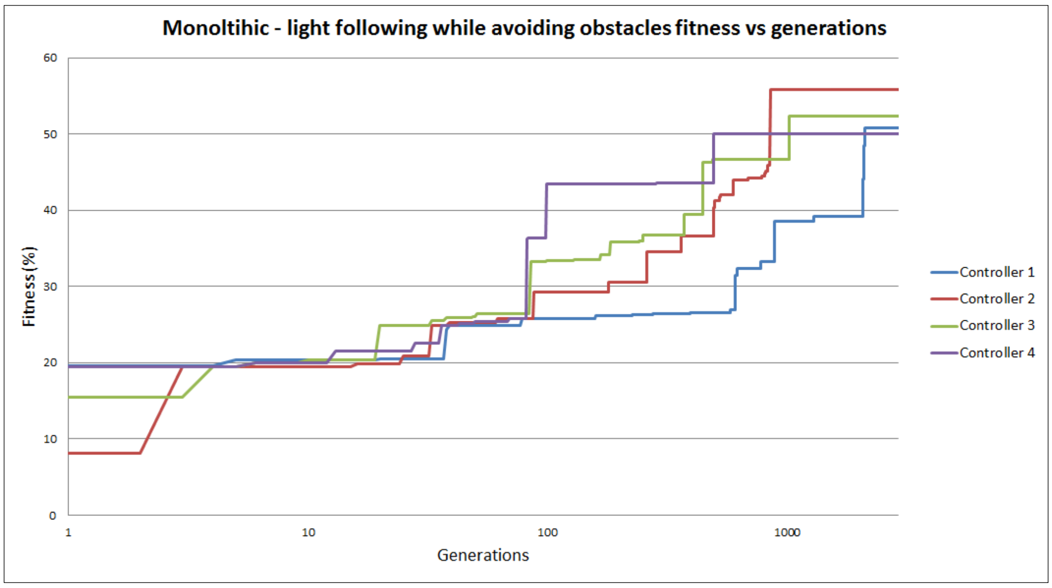
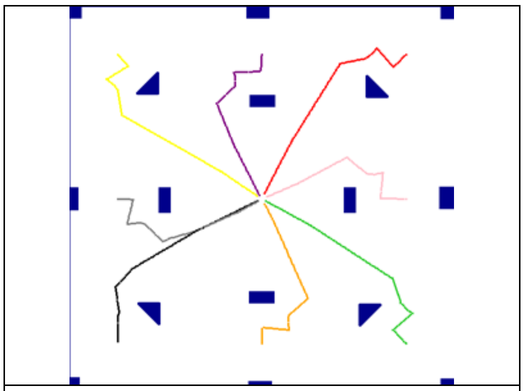


Figure 4-18 Maximum fitness trends for monolithic controller evolution

Subsumption evolution results

Four previously evolved light follower and object avoidance ANN controllers were chosen for the creation of the subsumption controllers. The controllers were paired up with a selection algorithm to create four controllers. The results are shown in Figure 4-19.

Note, the previously evolved controllers have been selected and paired randomly with no bias towards certain controllers. The trajectory shown below is a result from one of these pairs.



Subsumption Fitness = 94%

Generations for light follower ANN = 7

Generations for object avoidance ANN = 10

Total generations = 17

Figure 4-19 Subsumption controller trajectories (ANN)

The trajectories of the subsumption controllers show excellent light following and obstacle avoidance properties. Each of the four controllers clearly navigates towards the light source and only deviates from course when an object is detected. The same test was applied to several other controller combinations with very similar results obtained.

The advantages of subsumption evolution are clear when compared with monolithic techniques. The average 17 generations were required to evolve the controllers with subsumption techniques, where as to evolve a controller of the same performance level with monolithic techniques requires 1120 generations.

4.5 Overall Conclusions

- Light following – a range of ANNs were successfully evolved, it was found that quantization level of nine was the optimum size.
- Obstacle avoidance – an ANN has been evolved which was fully capable of collision avoidance.
- Combined behaviours – controllers have been successfully evolved using monolithic and subsumption evolution techniques. It has been found that controllers developed using subsumption methods outperform those developed using monolithic methods. The combined behaviours only require 17 generations to reach a desired result, where as those controllers evolved using monolithic techniques required on average 1120 generations.

Chapter 5

Chapter 5: Evolvable Lookup Tables for use in robotic controllers

This chapter outlines the experimental outcomes that have arisen as a result of evolving LUTs for use in robotic controllers. Experimentation has been undertaken with the goal of producing three evolved controllers for three different robotic behaviours.
Light following, obstacle avoidance and the combined behaviour of light following while avoiding obstacles.
The controllers for the behaviours outlined above were designed in a PC based simulated environment. The software containing the simulation and GA process has been described in Chapter 3:

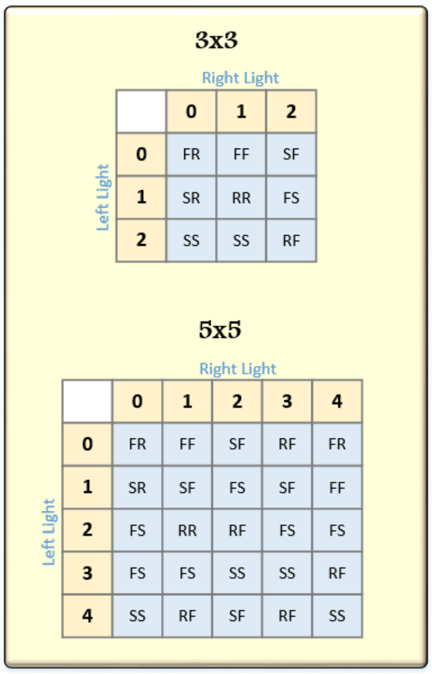
5.1.1 LUT GA and software structure

The software structure for the LUT controller GA and simulation is very similar to the software used to develop the ANN controller as described in section 4.1.1. Only changes to the chromosome format were made.

5.2 Light Following

5.2.1 LUT Chromosome

The LUTs were required to enable the robot to locate and move directly towards a light source. Thus the input to the controller was two quantized light levels received from the left and right senor, and the output is the required robot direction. A two dimensional LUT was used with the two axis linked to the left and right light sensor inputs. The LUT elements are used to control the left and right motors shown in Figure 5-1.



3x3

Left Light \ Right Light	0	1	2
0	FR	FF	SF
1	SR	RR	FS
2	SS	SS	RF

5x5

Left Light \ Right Light	0	1	2	3	4
0	FR	FF	SF	RF	FR
1	SR	SF	FS	SF	FF
2	FS	RR	RF	FS	FS
3	FS	FS	SS	SS	RF
4	SS	RF	SF	RF	SS

Figure 5-1 Light following LUT sizes examples 3x3 and 5x5

Two important questions arose while designing the LUT configuration for light following.

- What size LUT was required, relating to the quantization of the inputs and outputs?
- What sensor and actuator values should the LUT contain?

What size LUT is required, relating to the quantization of the inputs and outputs?
The elements of the LUT contain left and right motor control information, however what remained unclear was the level of control information that was required to achieve a satisfactory result. It was decided that the elements of the LUT would contain directional information only and that the motors would be both running at the same set speed. Using only the directional control versus speed control to drive the motors reduces the number of possible combinations that are available for each of the LUT elements, hence keeping the search space to a minimum. From the information presented above in Figure 5-1 it can be seen that limiting the control to directional control there are nine possible combinations of motor directions.

The search space for a LUT with a quantization light level of six:

- Quantized light level (Q) = 6

- Possible element outputs (E) = 9

$$\text{Search space} = E^{(Q)^2} = 9^{6^2} = 2.2 * 10^{34} \qquad \text{Equation 34}$$

To see the change in search space size, one can consider the option of implementing speed adjustment. If only three speed increments are used, such that the motors have three reverse speeds and three forward speeds, then it can be seen that the search space will be greatly increased.

- Quantized light level (Q) = 6
- Possible element outputs (E) = $(3\, forward + 3\, reverse + 1\, stop)^{\,2} = 49$

$$\text{Search space} = E^{(Q)^2} = 49^{6^2} = 7 * 10^{60} \qquad \text{Equation 35}$$

The search space of the LUT with speed control would be 2.5*10^{18} times larger than the LUT that uses only directional information.

In an effort to minimise the search space further it was decide that the stop condition in the list of available output motions was unnecessary and it could be removed. This would reduce the number of combinations to eight and reduce the search space as shown below.

- Quantized light level (Q) = 6
- Possible element outputs (E) = 8

$$\text{Search space(no stop)} = E^{(Q)^2} = 8^{6^2} = 3.2x10^{32} \qquad \text{Equation 36}$$

Removing the stop condition provides a reduction in the search space by a factor of 67.

A number of experiments were undertaken to further investigate the effect of search space on the GA process and the LUT performance, these tests include testing quantization levels from two to nine. The results can be found in section 5.2.3.

What sensor values should the LUT contain?

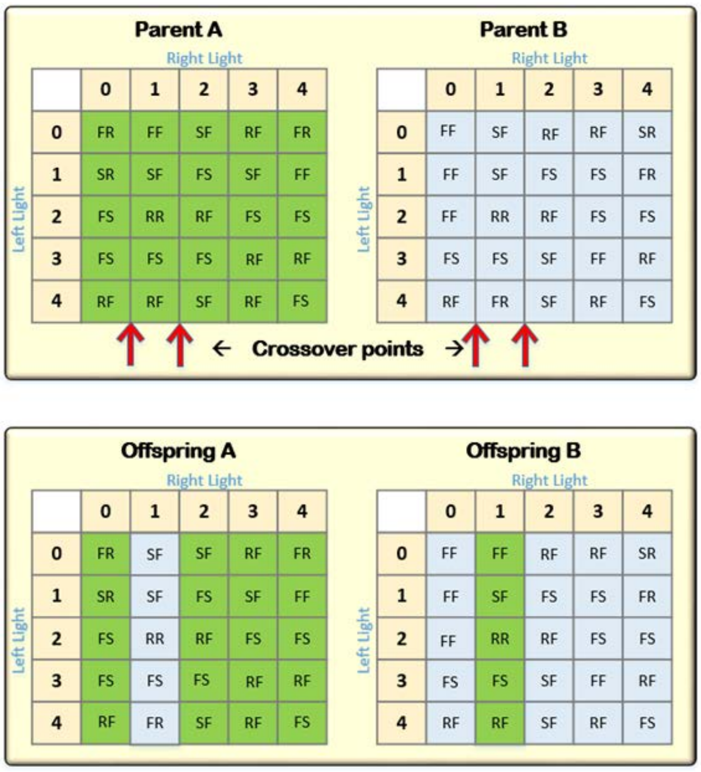
The size of the LUT is directly related to two important factors, the robot behaviour, (a smoother path requires a finer resolution of the light sensors) and secondly the GA search space (a finer resolution of the light sensors would create a larger LUT and a large search space). The light sensor outputs have been converted to integer format and quantized, with the level of quantization determining the size of the LUT and the search space for the GA.

Several quantization levels were investigated to determine the optimum motion of the robot and efficiency of the GA process, these ranged from 2x2 to 9x9. Figure 5-1 shows example 3x3 and 5x5 LUTs with respective search spaces of $1.3x10^{8}$ and $3.8x10^{22}$

The LUTs shown in Figure 5-1 are possible solutions that could be obtained from experimentation if the quantization levels were configured to be three or five. The elements within the LUTs contain the left and right motor directions and for this example each motor direction is represented as Forward (F), Reverse (R) and Stopped (S).

5.2.2 Offspring creation

A two-point crossover technique has been implemented. This technique involved selecting two random points along the X axis then sections separated by the selected points are swapped over to create the new offspring. Using a two-point cross over method for reproduction allows smaller sections of the LUT to be altered without making changes to other areas of the LUT. An example of the described method of reproduction is shown below in Figure 5-2.



Parent A

	0	1	2	3	4
0	FR	FF	SF	RF	FR
1	SR	SF	FS	SF	FF
2	FS	RR	RF	FS	FS
3	FS	FS	FS	RF	RF
4	RF	RF	SF	RF	FS

Parent B

	0	1	2	3	4
0	FF	SF	RF	RF	SR
1	FF	SF	FS	FS	FR
2	FF	RR	RF	FS	FS
3	FS	FS	SF	FF	RF
4	RF	FR	SF	RF	FS

Offspring A

	0	1	2	3	4
0	FR	SF	SF	RF	FR
1	SR	SF	FS	SF	FF
2	FS	RR	RF	FS	FS
3	FS	FS	FS	RF	RF
4	RF	FR	SF	RF	FS

Offspring B

	0	1	2	3	4
0	FF	FF	RF	RF	SR
1	FF	SF	FS	FS	FR
2	FF	RR	RF	FS	FS
3	FS	FS	SF	FF	RF
4	RF	RF	SF	RF	FS

Figure 5-2 Light follower LUT two-point crossover

5.2.3 Results

A set of experiments has been designed to test the performance of the LUT controllers and to analyse the efficiency of the GA process for light following behaviours. The results are divided into two main sections. The first section demonstrates the performance of the evolved controllers, and the second section explores the GA performance.

The fitness function used to evolve the light follower LUT calculates the robots distance from the light source after the allowable simulation time. This distance is used to determine a suitable fitness value. (A complete description of the fitness function can be found in section 3.1.4). Note, for all of the light follower experiments an upper fitness limit of 95% has been set, at this point the GA stops. 95% was chosen, because after this point, the population tends to converge and only very small levels of progress are made.

Look up table controller performance

The experiments start the robot in four starting positions and two starting headings per test, the exact positions and headings have been chosen to provide worst and best case scenarios with the goal of providing the best chance of developing a universal controller. One of the starting headings is pointing away from the light source while the other is pointing almost directly at the light and two of the starting positions are close to the light source while the others are much further away.

The performance of the controllers can be assessed by analysing the recorded trajectories taken by the robot, and the fitness during evolution. These trajectories as shown in Figure 5-3 demonstrate how the particular controller performed from each of the starting positions and headings during the evolutionary process. Note, the trajectories shown below contain two starting headings. The first heading of 90 degrees is shown in black while the second heading of 270 degrees is shown in green.

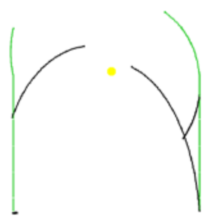
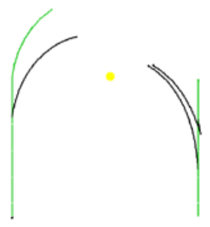
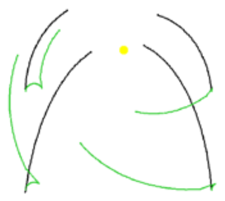
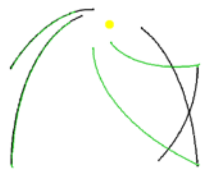
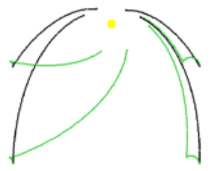
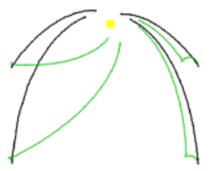
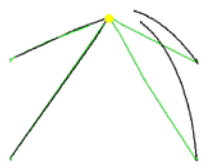
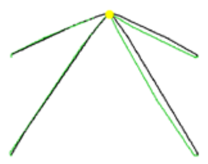
Fitness = 33%	Fitness = 44%	Fitness = 50%	Fitness = 68%
			
Fitness = 78%	Fitness = 83%	Fitness = 90%	Fitness = 97%
			

Figure 5-3 Example light follower trajectories and fitness percentage

In Figure 5-3 the stages of evolution can be seen. The initial stages of evolution seem to focus on developing the ability of the controller to steer towards the light and in most cases the path taken is a curve. Although traveling in an arc is not the most direct route, it still allows the robot to get fairly close to the light source and hence receive a reasonable fitness value.

Separate stages of the LUT trajectories can be seen evolving at different periods during the evolution of the controller. This is due the controller evolving different behaviours at separate times. For instance, the first resemblance of light following behaviours occurs when the robot is started in the general direction of the light source. In this situation a high fitness can be achieved by simply directing the robot to go forward causing the robot to end up nearer to the light source. Controllers that perform in this way tend to achieve a fitness of approximately 30-40%, and hence usually out perform any of the initial population which usually start out with an average fitness of 10-15%.

The next characteristic that tends to develop is the tightening of the arc shaped pathways creating a more direct route to the light, achieving a higher fitness value, usually 60-80%.

The final stage of evolution involves the rotation of the robot when no light source is detected. This occurs at the beginning of the testing procedure when the robot is initially pointing away from the light source. There are two main reasons this occurs only in the final stages of evolution. The first is due to complexity of the problem itself, the robot is required to rotate then move towards the light. If the robot does not stop rotating the rotate function serves no benefit and would achieve a fitness of zero. This rotate and move function requires more than just one evolved area of the LUT and it is highly unlikely that this would be found in the early stages of the evolution cycle. The second reason the rotate function tends to be evolved last is due to the way the fitness is allocated. An individual can achieve a high enough fitness to succeed in the tournament selections without the ability to rotate. This requirement only becomes a necessity to succeed in the final stages of evolution due to increasing levels of the average population fitness.

Light level quantization

Eight levels of quantization have been selected for controller performance analysis. These levels are shown with their respective LUT dimensions in Table 10

Table 10 Light levels chosen for analysis

Quantization	LUT size	Elements	Search space
2	2x2	4	4096
3	3x3	9	$1.3 * 10^{8}$
4	4x4	16	$2.8 * 10^{14}$
5	5x5	25	$3.8 * 10^{23}$
6	6x6	36	$3.2 * 10^{32}$
7	7x7	49	$1.8 * 10^{44}$
8	8x8	64	$6.2 * 10^{57}$
9	9x9	81	$1.4 * 10^{74}$

The resulting trajectories of the fully evolved controllers are shown in Figure 5-4. Note, not all the trajectories reached the light in the required time, even though the path was more direct. This was due to minor oscillations of the robot's movement caused by the quantization of the light sensors.

The final trajectories shown (Figure 5-4) revealed two noticeable characteristics that can be attributed to the level of quantization used. The first is the limited ability for the controllers with low quantization levels to reach the light source in the time limit set for each test. This is more noticeable in the trajectory for the 2x2, 3x3 and 4x4 controllers. The fitness measurement used for the evolution of these controllers uses an algorithm to determine the run time for each test. The algorithm assumed that the robot will rotate on the spot until facing the light then travel in a straight line until the light source is reached. However if the light sensor resolution is limited as is in the smaller LUTs, the fine heading adjustments required to move directly towards the light are not possible. This can cause the controllers to oscillate when moving towards the light. The oscillations waste run time resulting in the robot coming up short at the end of the time period.

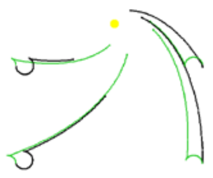
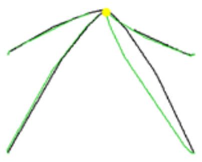
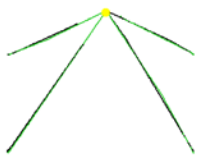
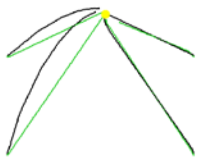
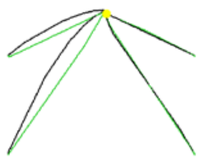
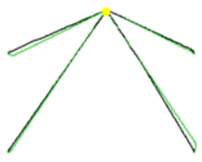
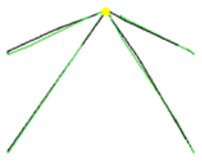
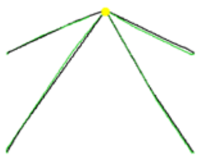
2x2 Fitness = 78%	3x3 Fitness = 91%	4x4 Fitness = 91%	5x5 Fitness = 95%
			
6x6 Fitness = 96%	7x7 Fitness = 97%	8x8 Fitness = 97%	9x9 Fitness = 97%
			

Figure 5-4 Resulting light follower trajectories

The second characteristic is the path in which the robot takes to get to the light source. Ideally the robot should begin the test by rotating on the spot until the sensors detect that the light source is directly in front. The robot should then move in a straight line towards the light. To do this the controllers require a sensor input resolution fine enough to accurately determine when the light is directly in front. The smaller LUTs have a disadvantage in the way that the sensor inputs have been quantized, such that the range of headings where the sensors would detect a light source directly in front would be greater than that of the larger LUTs. This causes the controllers to drive the robot forward assuming that the light source is directly in front however in reality the robot could be 10 degrees off course. When the robot gets close enough to the light the sensor inputs change and only then can the controller correct the course, this action results in an arc-like pathway towards the light source. As expected the radius of the arc shaped pathway tends to increase straightening out the path as the quantization level increases.

Evolutionary efficiency

The experiments in this section have been designed to explore the effects on evolutionary efficiency caused by varying search space. The evolutionary efficiency for these results is measured in generations and hence the points of interest will be analysed with respect to the generation in which they occurred.

Two criteria are tested: firstly will a high level of quantization require more generations to realise a suitable solution; and secondly will a lower quantization level be able to fully realise a suitable solution. To explore these criteria eight individual LUTs with different light quantization levels were evaluated.

The results obtained from the effects of quantization on the evolutionary efficiency are shown in Table 11.

Table 11 Light level quantization results

LUT Size	Initial fitness (%)	Final fitness (%)	Generation at 95%	Search space
2x2	77	80	---	4096
3x3	70	92	---	$1.3* 10^{8}$
4x4	68	91	---	$2.8* 10^{14}$
5x5	55	95	37	$3.8* 10^{23}$
6x6	53	95	33	$3.2* 10^{32}$
7x7	52	95	33	$1.8* 10^{44}$
8x8	48	97	43	$6.2* 10^{57}$
9x9	47	97	50	$1.4 * 10^{74}$

The lower levels of quantization (2x2 to 4x4) were unable to meet the required 95% performance. The optimum evolutionary efficiency occurred at quantization levels of 6x6 and 7x7. It is clear from the results that increasing quantization levels result in increased evolution times. The fitness of controllers with low quantization levels plateaued within 100 generations whereas those with higher levels plateaued much later at 200-300 generations.

The initial fitness obtained is directly related to the quantization level. A high quantization level provides a large search space which makes finding an adequate solution at the beginning of the GA process unlikely, and this reduces the initial fitness. However with a low quantization level the search space is much smaller and the chance of finding an adequate random solution is much more likely, and this causes the initial fitness to be higher.

Comparison of maximum fitness for different quantization levels

A comparison between the maximum fitness trends from each of the eight quantization (Figure 5-5) levels yields two interesting results. Firstly, as described above, the initial fitness can be seen to be increasing as the quantization level decreases and secondly, the generation at which the fitness begins to plateau increases as the quantization level increases.

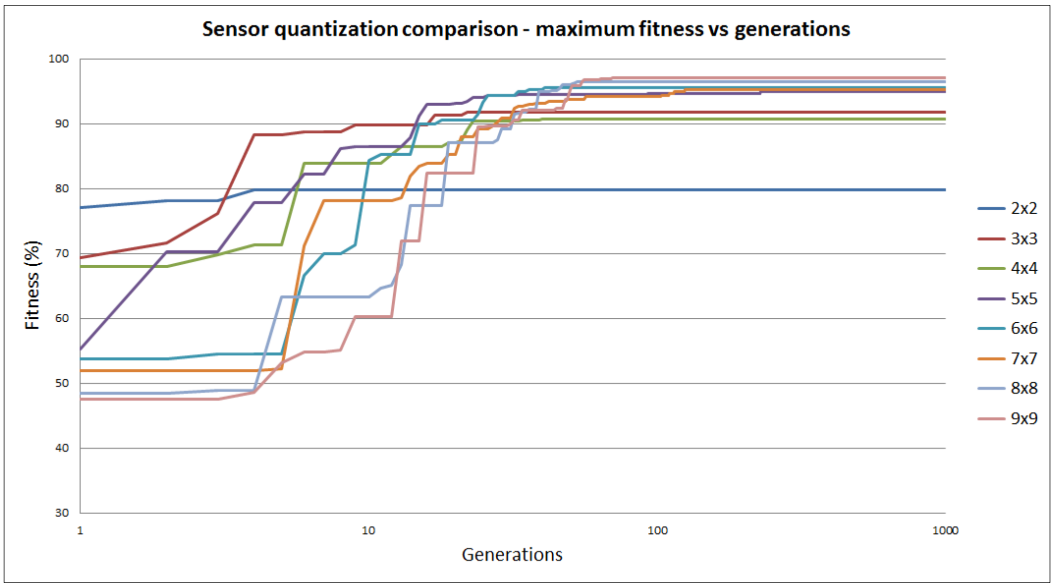


Figure 5-5 Quantization comparison - maximum fitness

Comparison of average fitness for different quantization levels

The general shape and positioning of the fitness curves for the population averages (Figure 5-6) over all the quantization levels remain very similar with the only noticeable difference being that of the 2x2 LUT.

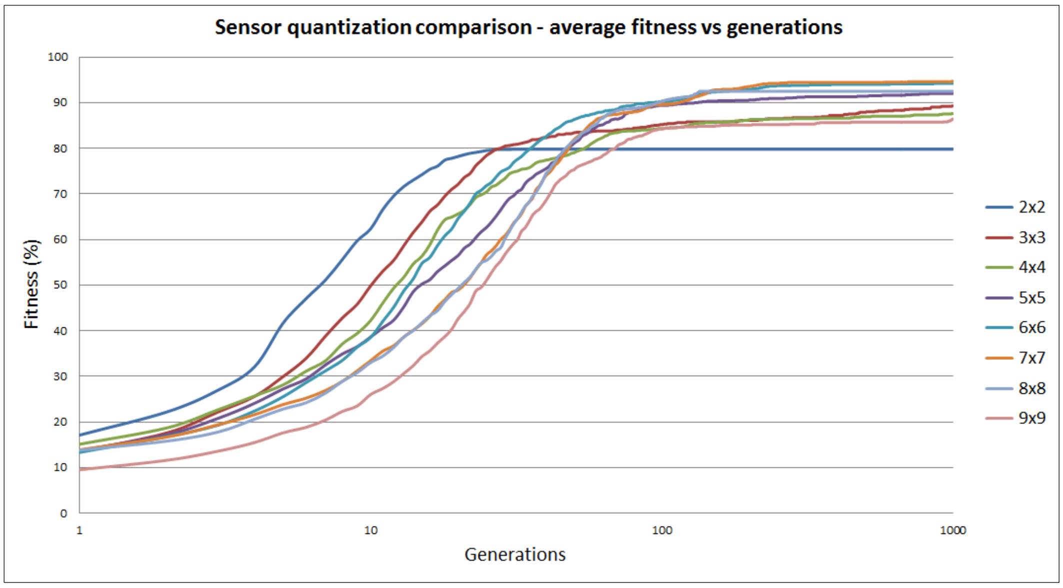


Figure 5-6 Quantization comparison - average fitness

Based on the comparisons described above it was determined that the optimal quantization level for LUT controllers in this simulation was eight (8x8 LUT). This was chosen because of the high level of fitness (95%) achieved and only requiring 33 generations vs. 50 generations for the quantization level of nine which also reached the high fitness level.

5.3 Obstacle avoidance

5.3.1 Object avoidance LUT chromosome

LUTs have been evolved to perform object avoidance behaviours. The LUTs are configured as one dimension arrays with the axis connected to the proximity sensors and the parameters providing the output motor controls.

The six analogue proximity sensors mounted on the robot have been quantized to provide a digital signal. The quantization of the outputs from analogue to digital reduces the GA search space and the overall complexity of the problem. The quantization level is such that the sensors return a logic one when an object is detected within range (50mm) and a logic zero when no object is detected.

The quantized (digital) output from the sensors is combined in a binary format, which provides a number from 0-63. The binary configuration of the sensors is used as the indexing value for the LUT. Equation 37 describes how the maximum decimal value is obtained from a varying number of object sensors.

$$\text{Object sensor bit combinations} = 2^n = 2^6 = 64 \qquad \text{Equation 37}$$

Where:

n is the total number of object sensors.

As shown in Figure 5-7 the proximity sensors have been allocated a bit number from 0-5, this provides the order for which the sensor outputs are combined. In this example, sensors one and four are active which combines to the binary format 010010 which is equivalent to the decimal value 18. Hence under these conditions the contents of element 18 in the LUT would be retrieved and used to provide the motor control.

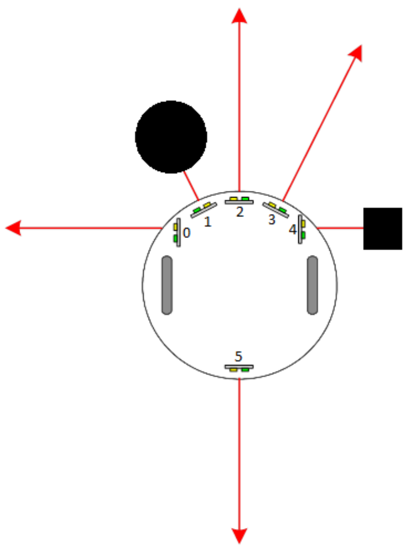
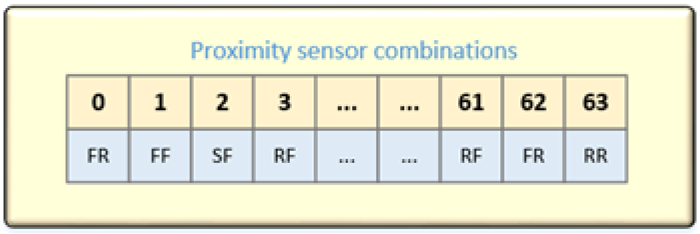


Figure 5-7 Proximity sensor combination example

A visual representation of the LUT used can be seen in Figure 5-8, the index values range from 0-63 and the elements contain the left and right motor directions.



Proximity sensor combinations

0	1	2	3	...	...	61	62	63
FR	FF	SF	RF	...	...	RF	FR	RR

Figure 5-8 Object avoidance LUT

Search space – The search space of the designed LUT can be calculated as follows.

$$\text{Object avoidance Search space} = E^{2^n} = 8^{2^6} = 6.2 * 10^{57} \qquad \text{Equation 38}$$

Where:

> E is the number of possible element combinations
> n is the number of sensors

5.3.2 Offspring creation

The reproduction method uses four-point crossover with a mutation rate of three per cent. This has proven to be successful and provides enough amalgamation to produce desirable offspring.

Figure 5-9 shows the creation of two offspring by means of a four point crossover, the four points are chosen at random and are used to determine which parts of the parent's chromosomes are used to create the offspring.

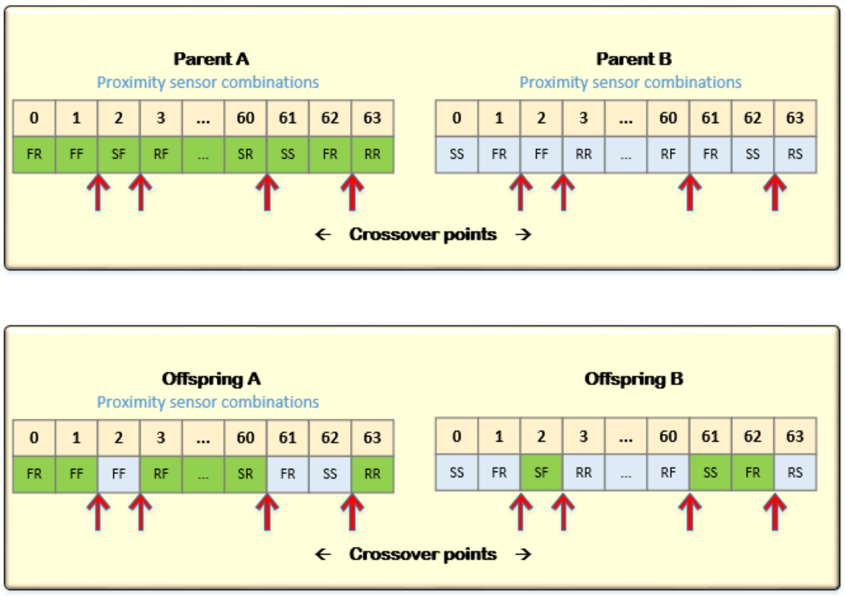

Parent A
Proximity sensor combinations
0 1 2 3 ... 60 61 62 63
FR FF SF RF ... SR SS FR RR
Parent B
Proximity sensor combinations
0 1 2 3 ... 60 61 62 63
SS FR FF RR ... RF FR SS RS
← Crossover points →
Offspring A
Proximity sensor combinations
0 1 2 3 ... 60 61 62 63
FR FF FF RF ... SR FR SS RR
Offspring B
0 1 2 3 ... 60 61 62 63
SS FR SF RR ... RF SS FR RS
← Crossover points →

Figure 5-9 Offspring creation with four crossover points

5.3.3 Results

A set of experiments has been designed to evaluate the controller performance of a LUT used as an object avoidance controller and the evolutionary efficiency of the GA used to evolve it. The results obtained from these experiments are outlined in the two sections below, the first section focuses on the LUTs performance as a controller and the second section focuses on the GA efficiency.

The fitness function used to evolve the obstacle avoidance LUT considers two variables, the robot's total runtime and the movement count. Both values are converted to a ratio with their respective maximums and used to calculate the fitness value. Note, for all of the object avoidance experiments an upper fitness limit of 80% has been set. Due to the operation of the fitness function, a fitness value much above this point is unachievable. A complete description of the fitness function can be found in section 3.1.4.

LUT controller performance

The experiments developed are used to evaluate the LUTs ability to perform in a range of different environments. The controllers are evolved from eight starting headings in three arenas with differing object configurations. The controllers from each of the experiments are then put into other arenas and the performance evaluated. The first controller has been evolved in arena A and the progressing trajectories are shown in Figure 5-10. A diagram of each arena used in these experiments can be found in section 3.1.3.

Note the trajectories shown below contain eight starting headings.
0° = RED, 45° = Green, 90° = Black, 135° = Grey, 180° = Pink, 225° = Purple,
270° = Orange, 315° = Yellow.

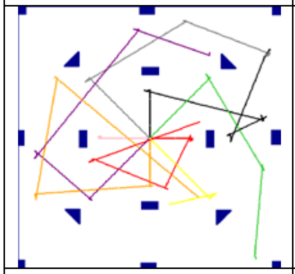
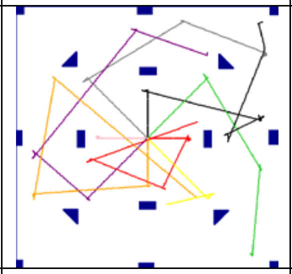
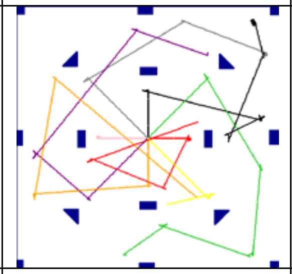
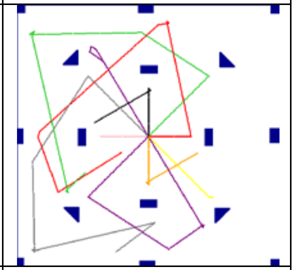
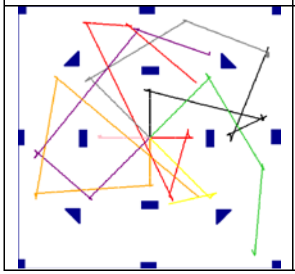
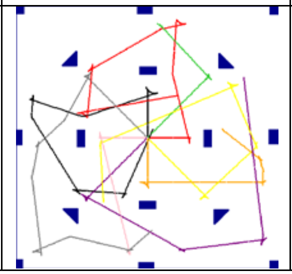
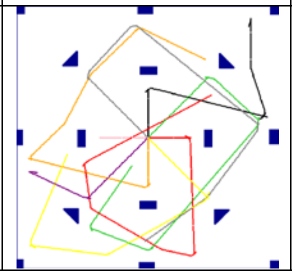
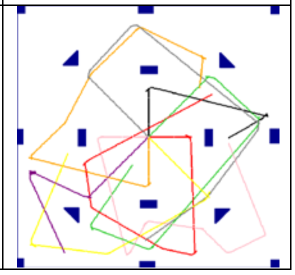
Fitness = 11%	Fitness = 30%	Fitness = 40%	Fitness = 45%
			
Fitness = 50%	Fitness = 54%	Fitness = 60%	Fitness = 84%
			

Figure 5-10 Arena 'A' evolutionary stages, showing the trajectories and fitness obtained

The total fitness for each test is comprised of an average from the eight starting headings. Because of this an individual can achieve a low fitness even though it may appear that it performed well. This is the case with the individual achieving 11% (shown in Figure 5-10), where clearly some of the starting headings are performing well but others are not. As the evolution process progresses it can be seen that other starting headings begin to perform and hence the fitness begins to increase.

It can also be seen that the individuals moving about the arena with a smoother trajectory achieve a higher fitness as is the case with the individuals with 54% and 60%. This is due to the properties of the fitness function which has been designed to force the robot to move about the arena. This is achieved by keeping track of the robots movement at periodic intervals. Therefore, when the robot fails to move a set distance the fitness is reduced and this can be seen in the trajectories where the path appears to get stuck when an object is encountered.

To compare the effect of arena selection during the evolution process on the performance of the LUT, two more arenas have been used to develop two additional controllers. The trajectories for arena B and arena C are shown in Figure 5-11 and Figure 5-12.

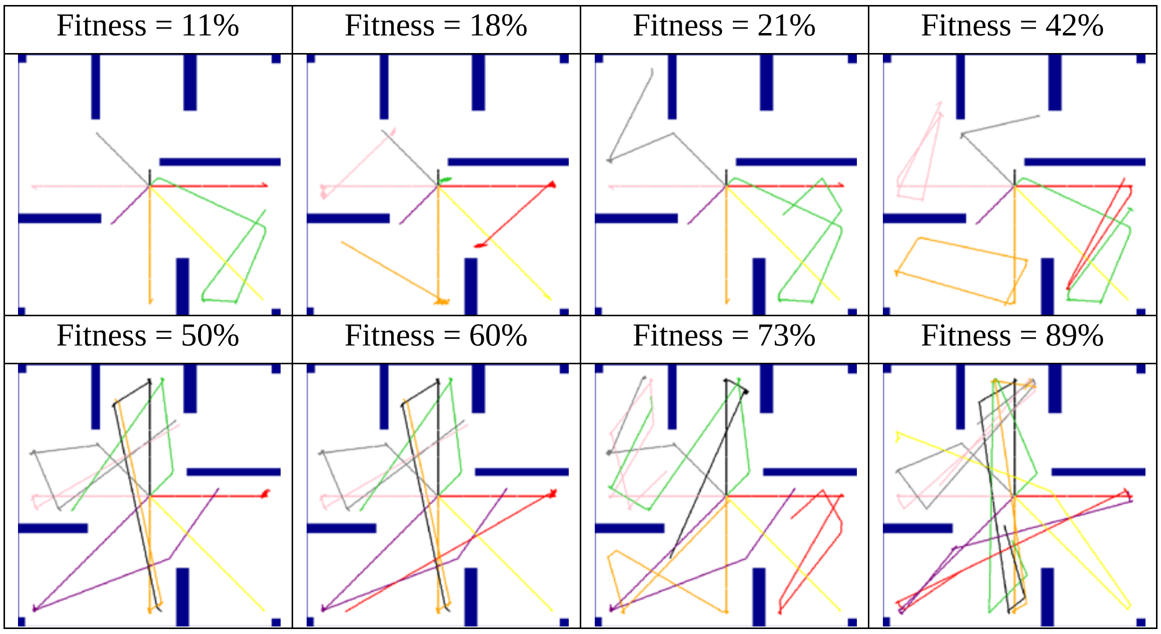


Figure 5-11 Arena B evolutionary stages, showing the trajectories and fitness obtained

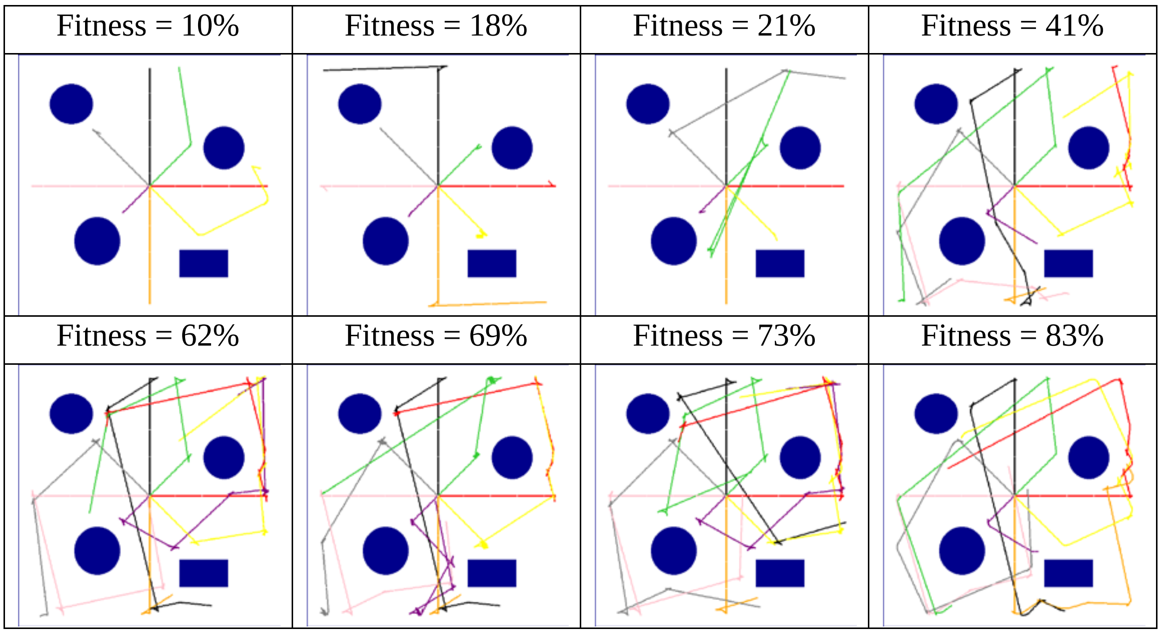


Figure 5-12 Arena 'C' evolutionary stages, showing the trajectories and fitness obtained

Similar evolution patterns to those found in the trajectories from arena A can be seen in the trajectories from arena B and arena C. The initial stages of evolution cause the controller to roughly move about the arena with the overall path becoming smoother in the later stages of the evolutionary cycle.

The evolved controllers from arena A, B and C have been evaluated using three unfamiliar arenas. The trajectories from each controller in the arenas are shown below in Figure 5-13.

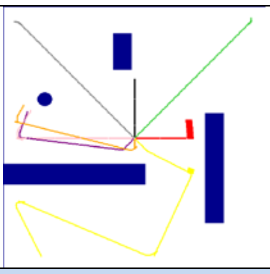
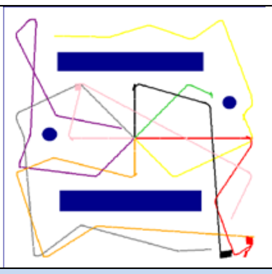
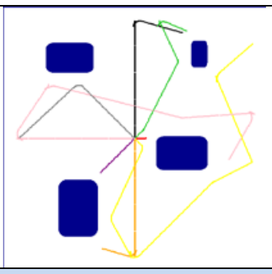
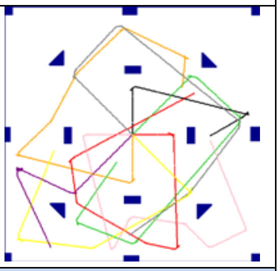
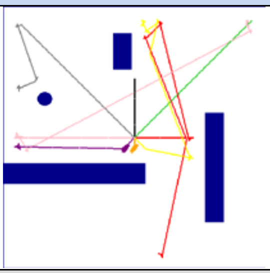
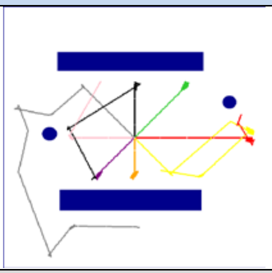
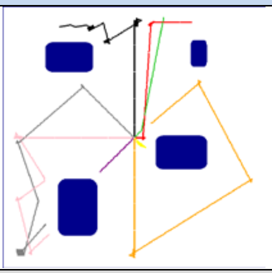
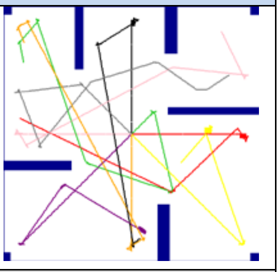
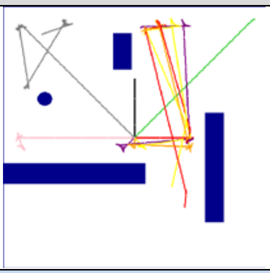
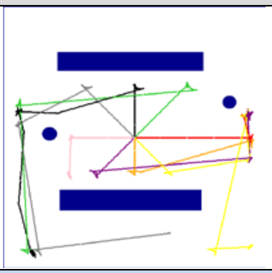
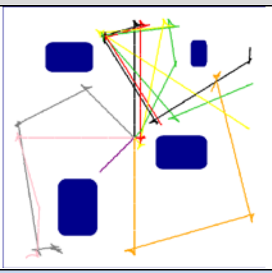
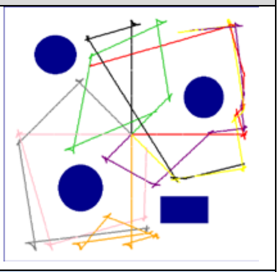
	Arena 1	**Arena 2**	**Arena 3**	**Original arena**
Controller A				
	Fitness = 11%	Fitness = 75%	Fitness = 22%	Fitness = 84%
Controller B				
	Fitness =40%	Fitness = 20%	Fitness = 29%	Fitness = 85%
Controller C				
	Fitness =30%	Fitness = 51%	Fitness = 51%	Fitness = 83%

Figure 5-13 Evolved controllers evaluated in unfamiliar arenas

It can clearly be seen from the trajectories shown above that the controllers in most of the tests exhibit some object avoidance behaviours. However when placed into an unfamiliar arena, these controllers are not capable of performing to the same standard as achieved in the original arenas. This inability to perform as a universal controller is due to the incomplete evolution of the LUT. Not all of the possible sensor combinations are encountered during the evolution process and the LUT does not fully evolve.

Multiple arena evolution

To overcome the incomplete evolution problem, a change has been made to the evolution process. For this test the controllers have been evolved using a combination of arenas simultaneously. The combination is comprised of arenas A, B and C. The average fitness from the combined arenas is used to determine the fitness of the individual. Note, the combined evolved controller achieved a fitness of 81%

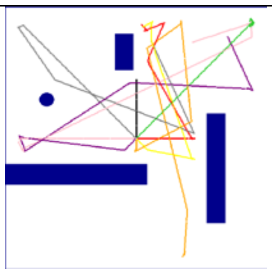
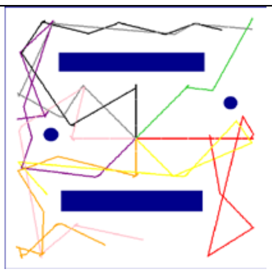
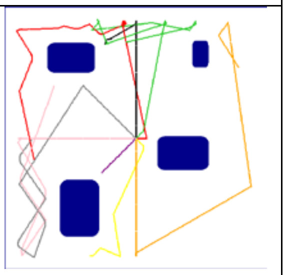
	Arena 1	Arena 2	Arena 3
Controller ABC			
	Fitness = 63%	Fitness = 78%	Fitness = 55%

Figure 5-14 Combined LUT evolution test in unfamiliar arenas

The resulting trajectories from the combined controller in the unfamiliar arenas are shown in Figure 5-14. When compared to the trajectories in Figure 5-13, a significant performance increase can be seen. The controller runs for much longer during each run and also travels via a much smoother path.

Evolutionary efficiency

This section focuses on the efficiency of the GA process in particular the number of generations required to obtain a suitable result. Controllers have been evolved in arenas A, B and C separately and arenas A, B and C simultaneously. The results are shown in Table 12.

Table 12 Arena evolution results (LUT).

Arena	Initial fitness (%)	Final fitness (%)	Generation at 70%	Generation at 80%	Arena Complexity Less to more(1-4)
A	2	80	104	262	3
B	19	85	101	208	2
C	2	84	121	161	1
ABC	12	81	213	347	4

Due to the complexity of the problem, very low initial fitness's are achieved (2-19%) and any differences in arena complexities are not obvious when comparing these values. However when comparing the generations at which the fitness reaches 70 and 80%, a clear difference can be seen. Based on the number of generations required to reach a suitable solution, the order of complexity for the arenas from highest to lowest would be ABC, A, B, C.

Maximum individual fitness trends for each arena configuration.

Analysis of the maximum fitness plots in Figure 5-15 reveals two areas of interest. Firstly the number of generations required to evolve the controllers is directly related to the complexity of the arena. The least complex arena C can be seen evolving at a steady rate and reaching a fitness of 80% in fewer generations than the other arenas. The second area of interest is the difference in evolution progress between the single arenas (A, B and C) and the combined arena (ABC). The single arenas tend to have large increases in fitness whereas the combined arena has small increases. This is attributed to the arena environment in which the controllers are evolved. A large increment in fitness can occur in a single arena when a controller evolves to navigate one particular obstacle in that arena. However, the same evolved behaviour in a combined arena environment will not exhibit the same result due to the large variety of obstacles in the combined arenas.

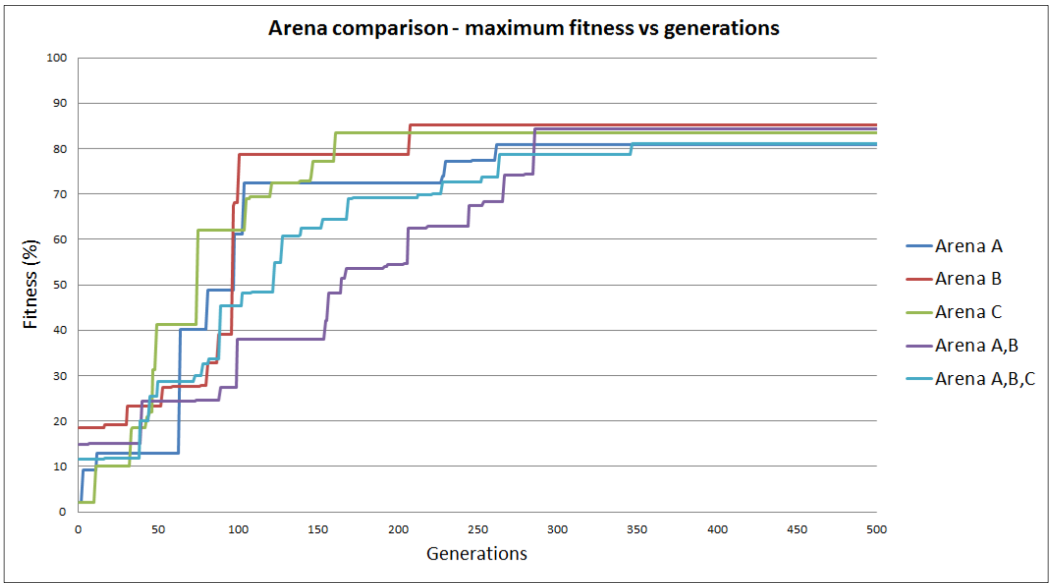


Figure 5-15 Maximum fitness comparison for each arena configuration

5.4 Light following while avoiding obstacles

The focus for this section of the research has been to evaluate the GA efficiency with an increase in the problems' complexity, using monolithic and subsumption evolution techniques to evolve LUT controllers.

5.4.1 Light following while avoiding obstacles LUT chromosome

LUTs have been evolved to perform the combined tasks of light following while avoiding obstacles. A three dimensional LUT has been configured to integrate the light and proximity sensor inputs. The outputs from the left and right light sensors provide the x and y index values while the proximity sensors provide the z index. The LUT elements contain directional control for the robot's left and right motors.

Figure 5-16 is a visual representation of the LUT configured for light following and object avoidance. The front face of the LUT is the same as the dedicated light follower LUT with the additional sections behind providing the necessary proximity detection control.

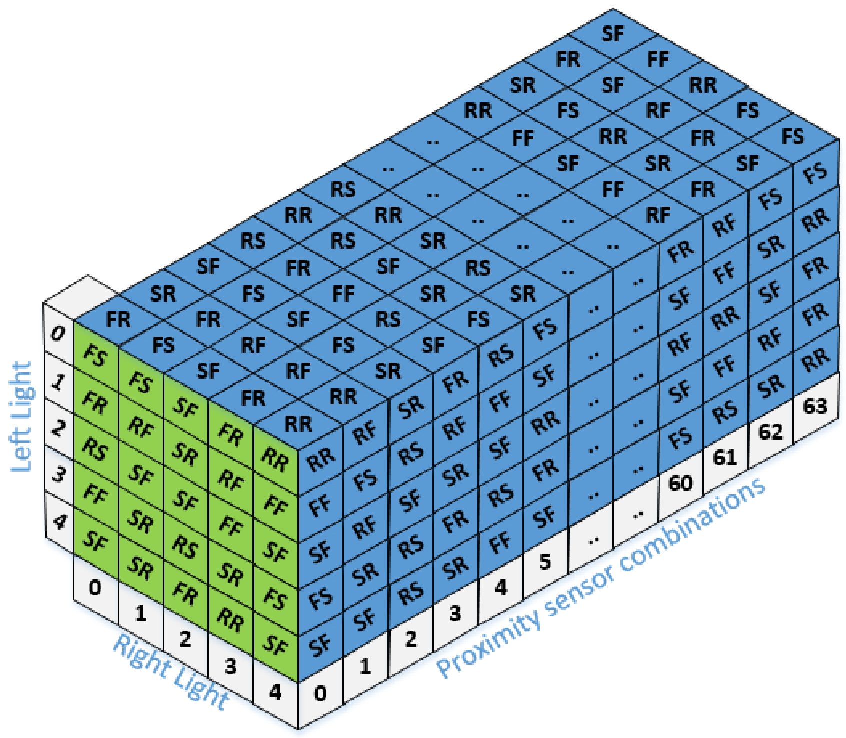


Figure 5-16 Light follower and object avoidance LUT configuration

As with the dedicated object avoidance LUT the number of elements available is directly related to the number of proximity sensors. In this case with six proximity sensors there are 2^6 possible elements, this combined with a light follower LUT with a light quantization level of 8 would provide a LUT size of 8x8x64 resulting in a total of 4096 elements in the LUT and a search space of 8^{4096}.

5.4.2 Offspring creation

The reproduction process chosen for the light follower and object avoidance LUT differs slightly from the reproduction of the individual behaviours. During reproduction the LUT is treated as a single dimension array of length determined by the number of elements in the LUT. Four points are chosen at random along the array and used as the crossover points to produce the offspring. This method provides the necessary level of genome amalgamation and has proven to produce suitable offspring.

5.4.3 LUT subsumption evolution

Subsumption evolution involves taking a complex behaviour and splitting it into smaller less complicated behaviours. In this case the combined behaviour was split into two behaviours, light following and object avoidance and evolved separately. The two evolved solutions were then combined using a switching controller such that the two controllers operate as one and perform the desired complex behaviour. Figure 5-17 below contains an example block representation of a subsumption controller vs. a monolithic controller.

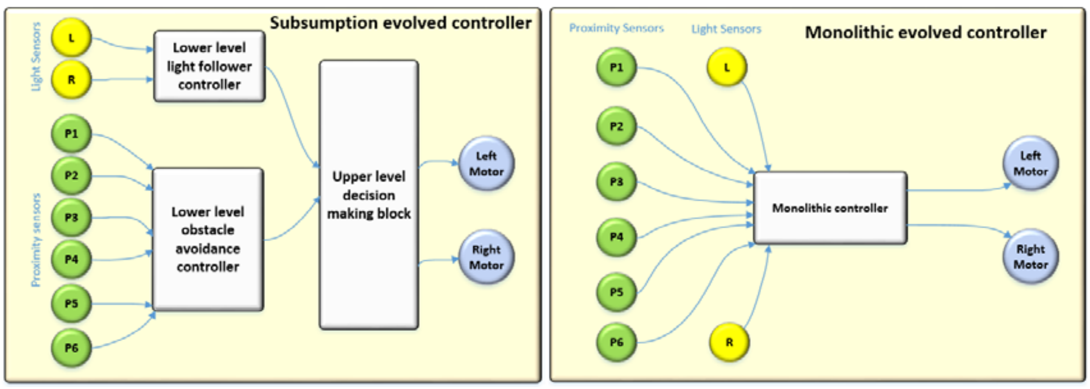

Subsumption evolved controller
Light Sensors
L
R
Lower level light follower controller
Proximity sensors
P1
P2
P3
P4
P5
P6
Lower level obstacle avoidance controller
Upper level decision making block
Left Motor
Right Motor
Proximity Sensors
Light Sensors
Monolithic evolved controller
P1
P2
P3
P4
P5
P6
L
R
Monolithic controller
Left Motor
Right Motor

Figure 5-17 Subsumption controller vs. monolithic example

Implementation of the subsumption controller involved taking the previously developed controllers for the separate behaviours and switching between them by considering the current status of the robot’s sensors. If the proximity sensors are inactive, then the controller will use the LUT evolved to perform light following behaviours. However when an object is present and one or more of the proximity sensors are active, the LUT evolved for the object avoidance behaviours will be used.

The desired outcome would be a robot that would turn directly to the light source and move towards it in a straight line. If the robot encountered an object then the robot would switch controllers and move away from the object until the object was not detected, then switch back to the light follower controller.

5.4.4 Results

A set of experiments have been designed to evaluate the LUT performance and evolutionary efficiency when controllers are developed using monolithic and subsumption techniques.

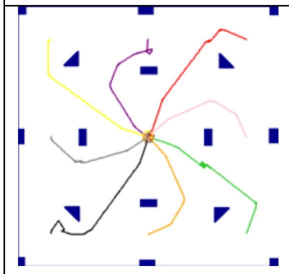
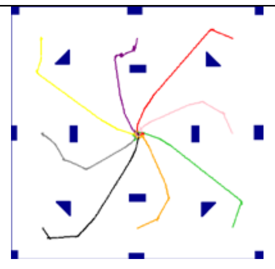
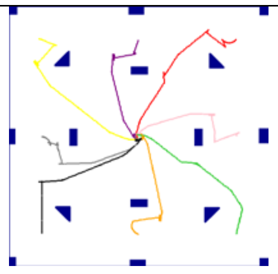
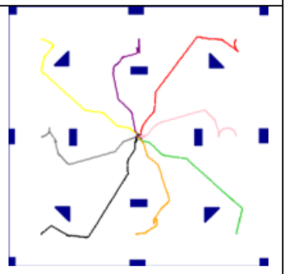
The fitness function used to evolve the LUTs for the combined behaviours considers two variables; the robots distance from the light at the end of the test and the length of the path taken.
A complete description of the fitness function can be found in section 3.1.4

The method of evolution developed for these experiments starts each individual from eight different starting locations each with a unique heading. Four controllers have been evolved using the monolithic and subsumption techniques. These controllers have been compared in two ways, firstly the performance of the controller is analysed and secondly the GA process itself is analysed.

Note, all of the experiments are undertaken using arena A and an upper fitness limit of 80% or 20,000 generations has been set for the monolithic evolution. A controller receiving a fitness of 80% is considered to be a good controller due to the way the fitness function operates.

Monolithic evolution results

The four controllers shown in Figure 5-18 clearly exhibit both light following and object avoidance behaviours. All of the controllers perform to a very high standard and have been fully evolved before the limit of 20,000 generations. The evolved controllers can be seen navigating around obstacles and when a clear path is found heading directly towards the light and hence achieving the maximum allowable fitness. The reason these controllers do not reach fitness much higher than 80% is due to the operation of fitness function. A path length equal to the minimum straight line distance between the starting position and the light would be required to reach a fitness of 100% and this is not possible.

Controller 1	Controller 2	Controller 3	Controller 4
			

Fitness = 81%
Generations = 8227
	Fitness = 85%
	Generations = 10998
		Fitness = 84%
		Generations = 16126
			Fitness = 86%
			Generations = 11041

Figure 5-18 Monolithic controller trajectories

The disadvantage of using this method of evolution for complex behaviours is the number of generations required to achieve a desirable fitness as shown in Figure 5-19. The evolution progress was very slow with most of the controllers taking over 10,000 generations to evolve. This is largely due to the large search space created by the three dimensional LUT. In summary the LUT is useful for evolving controllers for simple behaviours, however it suffers from scalability issues as the complexity of the controller increases.

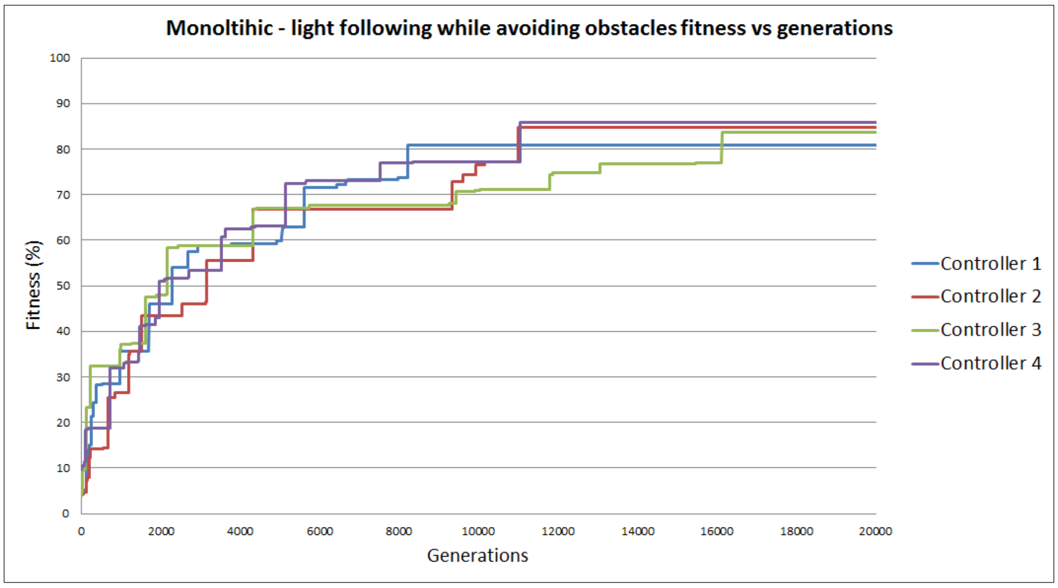
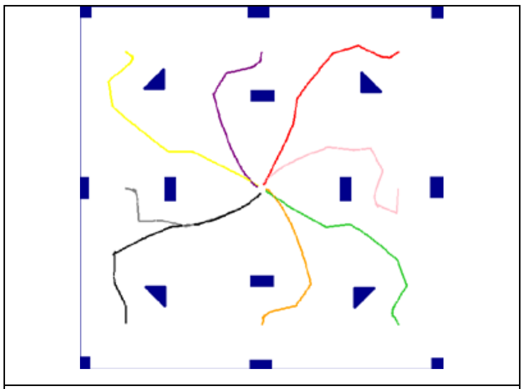


Figure 5-19 Maximum fitness trends for monolithic controller evolution

Subsumption evolution results

Four previously evolved light follower and object avoidance LUTs were chosen for the creation of the subsumption controllers. The LUTs were paired up with a selection algorithm to create four controllers of which the results are shown in Figure 5-20.

Note, the previously evolved controllers have been selected and paired randomly with no bias. The trajectory shown is a result from one of these pairs.



Subsumption Fitness = 80%

Generations for light follower LUT = 50

Generations for object avoidance LUT = 298

Total generations = 348

Figure 5-20 Subsumption controller trajectories

The trajectories of the subsumption controllers show excellent light following and obstacle avoidance properties. Each of the four controllers clearly navigates towards the light source and only deviates from course when an object is detected.

Evolving the controllers using subsumption techniques has clear advantages, the combined generations required to evolve both of the sub-behaviours are far fewer than if the controller were to be evolved using monolithic techniques as one complete controller.

5.5 Conclusion of the experiments with a LUT based controller

- Light following – a range of LUTs were successfully evolved, it was found that the 8x8 LUT was the optimum size.
- Obstacle avoidance – a LUT has been evolved which was fully capable of collision avoidance.
- Combined behaviours – Controllers have been successfully evolved using both monolithic and subsumption evolution techniques and it has been found that controllers developed using subsumption methods outperform those developed using monolithic methods.
    - Monolithic – A singular three dimensional LUT has been evolved and is capable of performing light following tasks while avoiding obstacles.
    - Subsumption – Two LUTs have been separately evolved and combined using a high level decision making algorithm to perform light following tasks while avoiding obstacles.

Chapter 6

Chapter 6: Evolvable Hardware for use in robotic controllers

This chapter describes the use of EHW controllers to create robotic controllers. These controllers have been evolved to perform three behaviours -light following, object avoidance and light following while avoiding obstacles. In order to overcome the limitations of a normal FPGA with a fine-grained architecture, capable of destructive routing, a reconfigurable FPGA platform known as a virtual-FPGA was created. This was based on a Cartesian architecture, non-destructive and course grained, providing a reduced search space and implemented in the actual FPGA device. The virtual-FPGA was comprised of logic array blocks (LABs) that contain multiplexers for switching inputs, and logic elements to provide logic manipulation of the selected inputs. The following virtual-FPGA architectures described here are the result of several iterations of testing including reducing and flat layer architectures as well as a variety of logic element configurations. The virtual-FPGA is configured via a CBS. The bit-stream provides control over the internal LABs of the virtual-FPGA and is the chromosome evolved via the GA.

The GA and the software simulation are run on a PC and the configuration bit-stream is transmitted via a serial link to a NIOS soft-core processor and the virtual-FPGA that has been implemented on Altera's DE2-115 application board (Figure 6-1). The NIOS was constructed with a UART enabling serial communications to the PC; a SDRAM driver for memory; fourteen 32-bit I/O ports for parallel transfer of the CBS and two I/O ports for the virtual-FPGA inputs and outputs sent from the robot simulation on the computer.

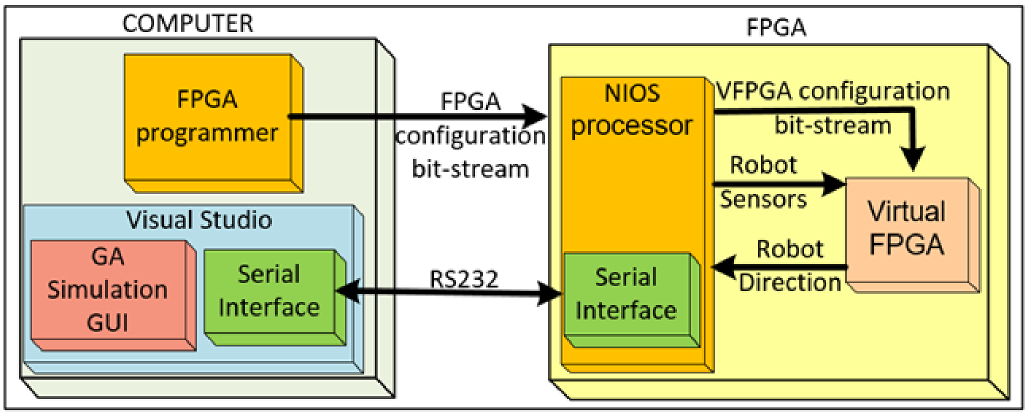


Figure 6-1 EHW System

6.1.1 EHW GA and software structure

The software used to evolve the EHW controllers is similar to the software used for the LUT and ANN controllers however an additional piece of software has been developed which runs on an NIOS processor on the FPGA development board. The NIOS receives bit-stream information from the PC and configures the virtual-FPGA accordingly. Note, as with the LUT and ANN controllers the chromosome for the EHW controller is evolved on the PC.

The structure of the software on the NIOS processor is described below.

- Communications received – The main function of the NIOS processor is to receive and interpret the communications from the PC. The information sent from the PC contains the bit-stream from the individual under test as well as the robot’s light and proximity sensor information.
- Virtual-FPGA configuration – The received bit-stream information is used to configure the virtual-FPGA for the individual under test.
- Using the virtual-FPGA – New sensor information received from the PC is input to the virtual-FPGA and the resulting outputs are read and sent back to the PC.

The structure of the software on the PC is described below.

- Initialisation – The Initialisation takes place once at the beginning of every evolution cycle. This involves generating the randomised population of bit-streams. Each bit-stream conforms to the format that has been developed to configure the specific virtual-FPGA for the desired behaviour.

- Communication – For each new test, the software begins by sending the bit-stream information via the RS232 interface to the NIOS processor. This configures the virtual-FPGA. The software then transmits the current sensor status to the NIOS processor and receives back the resulting output to determine the robot motion.
- Bit-stream evolution – the bit-streams are put through the testing, selection and reproduction processes until a satisfactory result is achieved.
- Data logging – The results obtained from the simulated evolution process are stored in text files which contain the maximum fitness, average fitness and the generations. These results are also plotted in an Excel spread sheet for ease of result analysis. The software also takes the bit-stream from the best performing individual and converts this into three logic expressions, the output of each expression represents one of the three bits required to determine the robot's motion.
- The computer clock speed was two orders of magnitude faster than the clock on the NIOS processor, thus the robot simulation and the GA were executed on the computer. This however required a large amount of data to be continually sent on the serial interface to the NIOS, slowing the evolution down. In order to reduce the data transfer, a protocol was developed that stored the response of the EHW on the computer for each new input state request. Thus if the data had already been retrieved from a previous request, the computer would use the previously stored data rather than interrogate the EHW.

Figure 6-2 is a visual representation of the sequences taken by the software running the GA evolution and the simulation for the evolution of a generic VFGPA.

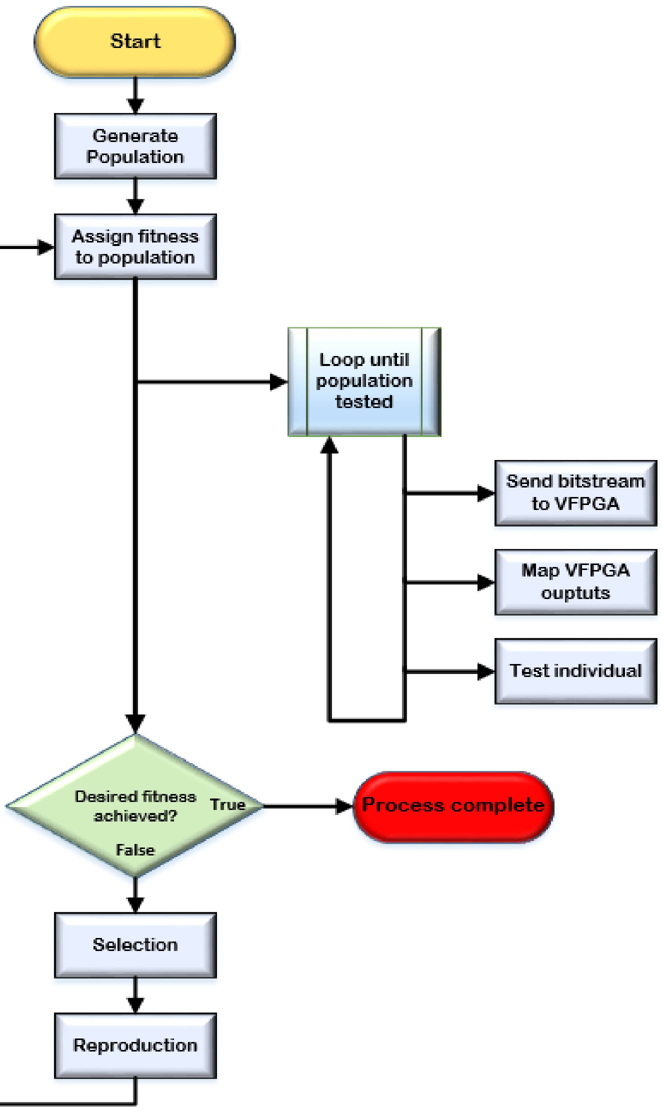


Figure 6-2 Simulation software flow chart for EHW evolution

6.1.2 Offspring creation

The reproduction method applied to the CBS, is similar to the method used to evolve the LUTs used for object avoidance. This uses a multiple point crossover method and a 3% mutation rate. The number of points used during the reproduction is chosen at random and ranges from 2 to 20.

6.2 EHW evolution for light following

This section describes the methods used to evolve the virtual-FPGA controllers designed to perform light following behaviours.

6.2.1 Virtual FPGA configuration

The virtual-FPGA has four layers of LABs. The inputs and outputs of the system consist of 20 digital inputs and 3 digital outputs. The digital inputs are split into two 10-bit sections which are used to provide the light level input to the system (Figure 6-3). For valid comparisons between controller platforms, the three output bits have been configured to provide eight possible robot motions as used in the LUT and ANN controllers.

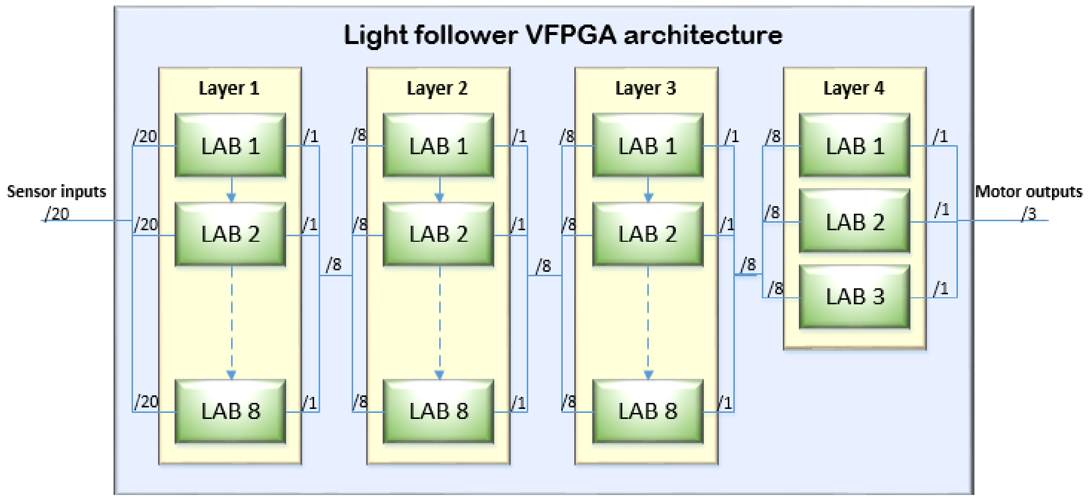


Figure 6-3 Light following virtual-FPGA architecture

The LAB contains two sections. The first section (MUX A and MUX B) is used to make two selections (A and B), from the available inputs. The second part (LE) is used to combine the selected inputs (A and B) using one of 16 selectable logic expressions (Figure 6-4).

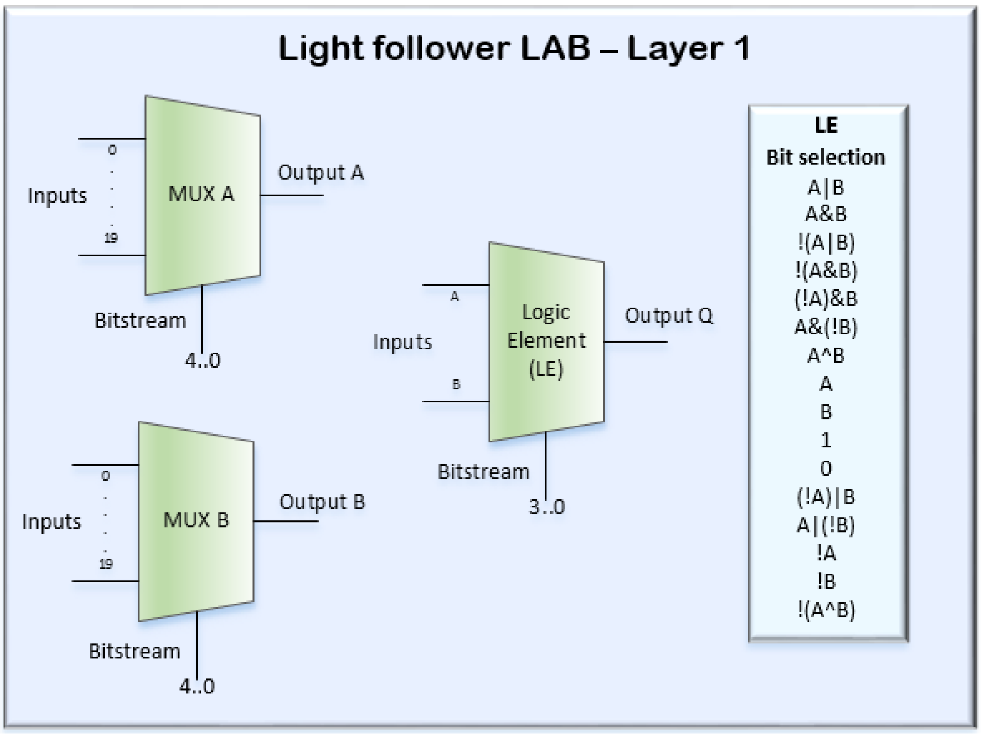


Figure 6-4 Light following layer 1 LAB

The input bits 0-9 contain the light intensity for the left sensor while inputs 10-19 contain the intensity from the right sensor. The bits are encoded so that as the light level increases the bits are left-shifted on the inputs, with a level of zero equal to 0000000001 and a level of five equal to 0000111111. Unused inputs are held at zero. A light intensity of five and three for the left and right light sensors respectively would create the combination of inputs shown in Table 13

Table 13 Light sensor input example for light intensities left = 5 and right = 3

(L,R)	Bit#	Bit
L	0	1
L	1	1
L	2	1
L	3	1
L	4	1
L	5	1
L	6	0
L	7	0
L	8	0
L	9	0
R	10	1
R	11	1
R	12	1
R	13	1
R	14	0
R	15	0
R	16	0
R	17	0
R	18	0
R	19	0

The number of bits required to configure each LAB in the first layer is comprised of five bits to control each multiplexer and four bits from the LE selection, resulting in a total of 14 bits per LAB. To control all eight LABs in the first layer of the virtual-FPGA requires (14*8 = 112) bits.

The configuration of the LABs used in the second, third and fourth layers differ quite considerably from those used in the first layer. Each comprises of eight inputs and one output (Figure 6-5).

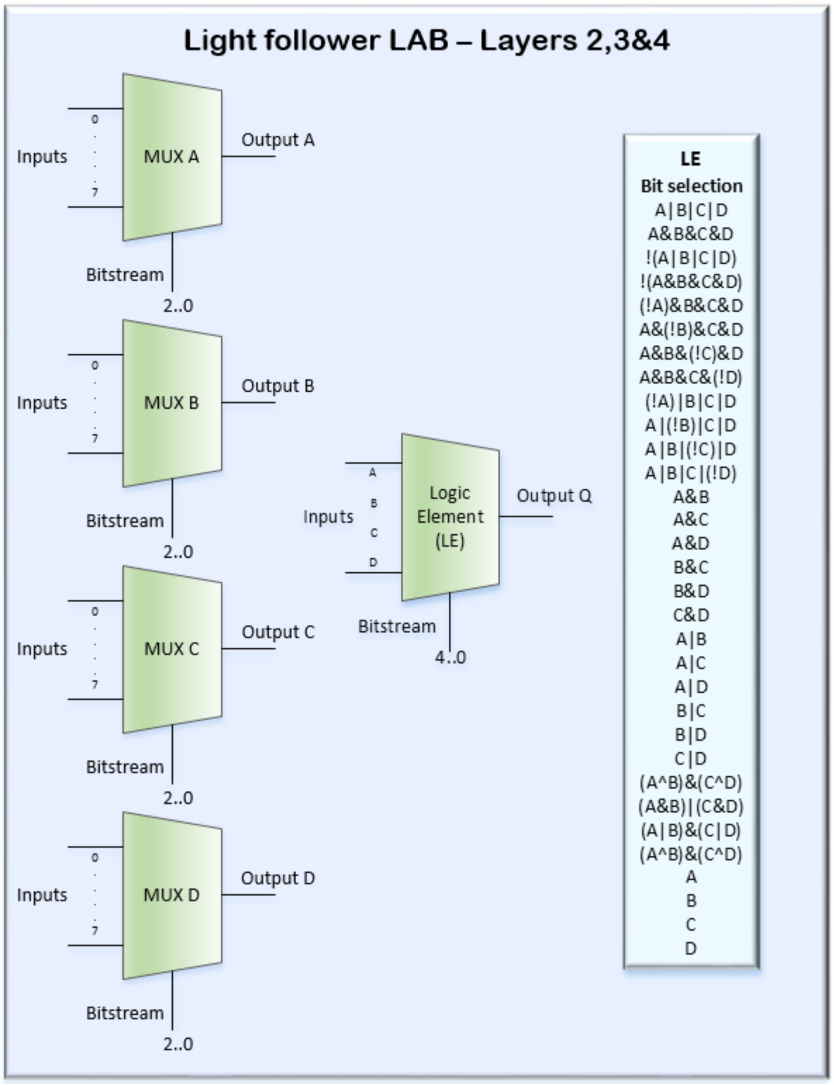


Figure 6-5 Light following layers 2, 3 & 4 LABs

Each input multiplexer selects one of the eight inputs and passes them on to the LE which performs one of 32 available logic expressions on the inputs (A, B, C and D). The output is a single bit.

Each LAB in these layers requires 17 bits to configure the multiplexers, three for each of the input multiplexers and five for output providing a total of 435 bits to configure the entire virtual-FPGA (112 bits in layer one and 323 bits in layers two, three and four). The search space for this virtual-FPGA is 8.8x10^{130}.

6.2.2 Results

A set of experiments has been designed to test the effectiveness of the EHW controller designed for light following behaviours (controller performance) and to analyse the efficiency of the GA process used to develop them, (evolutionary efficiency). To clearly outline the results obtained from this area of research the following section is divided into two main sections. The first section demonstrates simulated controller performance of the evolved controllers, while the second section explores the evolutionary efficiency.

The fitness function used to evolve the light follower EHW calculates the robots distance from the light source after the allowable simulation time. This distance is used to determine a suitable fitness value. A complete description of the fitness function can be found in section 3.1.4. Note, for all of the light follower experiments an upper fitness limit of 95% has been set, at this point the GA stops. 95% was chosen, because after this point, the population tends to converge and only very small levels of progress are made.

EHW controller Performance

The experiments described in this section use four starting positions and two starting headings per position. With the intention of developing a robust controller, one of the starting headings has been chosen so that the robot is pointing away from the light source while the other is pointing almost directly at the light. The starting positions have been chosen so that two are close to the light source and two are much further away.

The performance of the controllers is assessed by analysing the recorded paths taken by the robot during development. These paths as shown in Figure 6-6 show how a typical evolution cycle of a light follower controller. Note, the paths shown contain two starting headings. The first heading of 90 degrees is shown in black while the second heading of 270 degrees is shown in green.

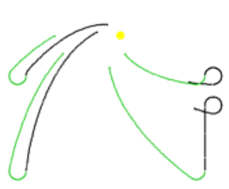
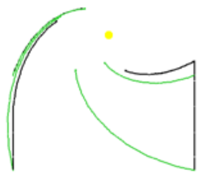
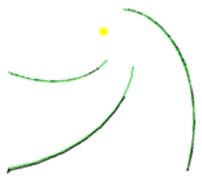
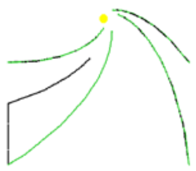
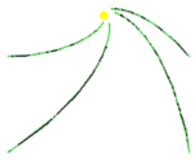
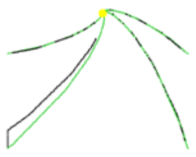
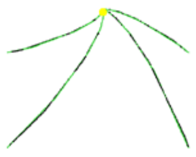
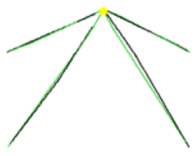
Fitness = 58%	Fitness = 65%	Fitness = 67%	Fitness = 85%
			
Fitness = 88%	**Fitness = 92%**	**Fitness = 94%**	**Fitness = 97%**
			

Figure 6-6 Example light follower trajectories and fitness percentage (quantization 9)

The controller trajectories clearly show an increase in controller performance as the evolution cycle continues. As seen with the previous light follower controllers the initial stages of evolution seem to focus on developing the ability of the controller to steer towards the light and in most cases the path taken is usually a curve and it is not until the later stages of evolution the trajectories become more direct.

Light level quantization

Eight levels of quantization have been selected for performance analysis. These levels and the trajectories of the fully evolved controllers are shown in Figure 6-7. Note, not all the trajectories reached the light within the required timeframe, even though the path was more direct, due to minor oscillations of the robot’s movement caused by the quantization of the light sensors.

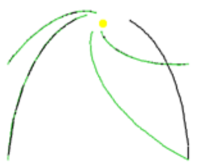
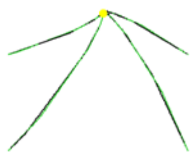
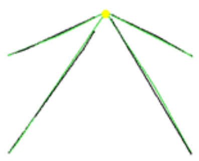
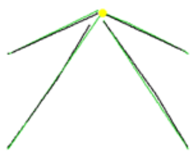
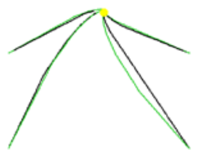
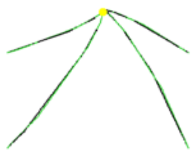
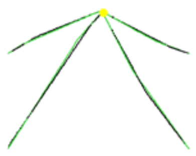
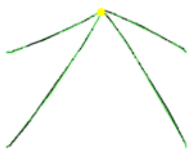
Quantization = 2 (82%)	Quantization = 3 (91%)	Quantization = 4 (91%)	Quantization = 5 (92%)
			
Quantization = 6 (94%)	Quantization = 7 (95%)	Quantization = 8 (95%)	Quantization = 9 (95%)
			

Figure 6-7 Resulting light follower trajectories (EHW)

The final trajectories revealed two noticeable characteristics that can be attributed to the level of quantization used. Firstly for the controllers with quantization levels greater than five there are only slight differences in performance. However the lower levels of quantization are unable to perform in an optimal manner. This is due to the loss of light sensor resolution caused by the level of quantization. Note, the fitness for the quantization levels four and five was reduced because the robot did not pivot in the shortest rotation before moving towards the light.

Evolutionary Efficiency

The experiments in this section have been designed to explore the effects on evolutionary efficiency caused by varying the level of light quantization. The evolutionary efficiency for these results is measured in generations and hence the points of interest will be analysed with respect to the generation of which they occurred. To explore these criteria eight individual EHW controllers with different light quantization levels were evaluated.

The table of results (Table 14) show that low levels of quantization affects the controller performance. This is not related to the search space, as in the case of the LUT, rather the ability to more accurately observe the direction of the light will affect the controller performance.

Table 14 Light level quantization results (EHW)

Quantization level	Initial fitness (%)	Final fitness (%)	Generation at 95%
2	45	82	---
3	50	91	---
4	82	91	---
5	72	92	---
6	61	94	---
7	73	95	15
8	70	95	15
9	79	95	10

The effects of quantization on the inputs can be seen graphically (Figure 6-8). With the exception of the controller with a quantization level of two, the remaining controllers all reach a fitness of 90% in a similar number of generations, however only the controllers with a quantization level of seven and above can reach the required 95% fitness.

In general, the evolution progresses up to a fitness of 90 per cent relatively quickly, with the majority of controllers requiring less than 10 generations. The final stages of evolution where the controller refines the path to the light can be seen evolving around the 80th generation point. This refinement is fairly difficult to achieve and only results in a small increase in fitness and this is why so many generations are required to achieve this.

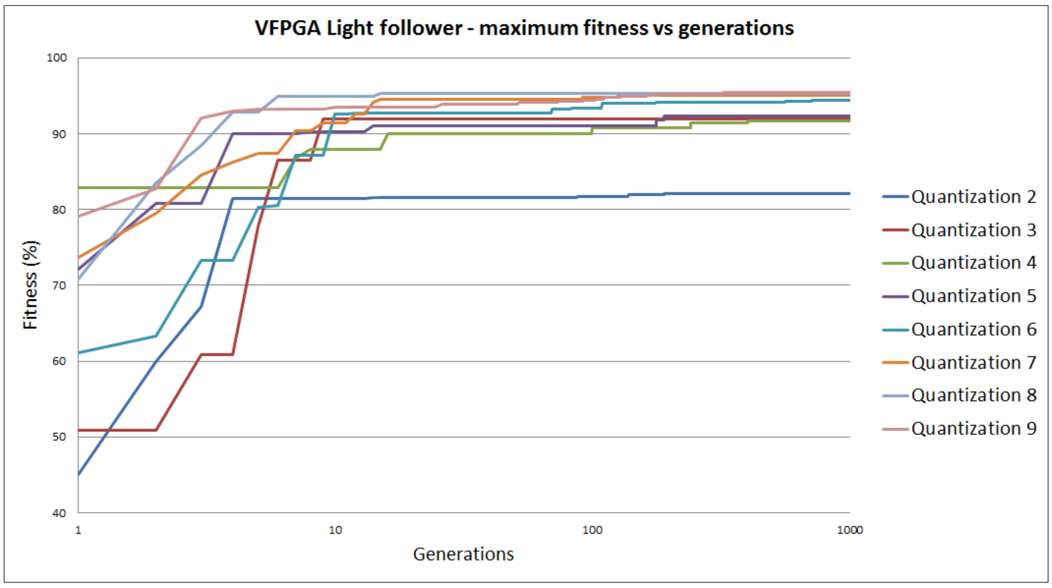


Figure 6-8 Quantization comparison - maximum fitness (EHW)

6.3 EHW evolution for obstacle avoidance

This section contains a description of the virtual-FPGA architecture used in the evolution of the hardware controller developed to perform object avoidance behaviours.

6.3.1 Virtual FPGA configuration

A controller has been evolved to perform object avoidance behaviours and is comprised of a three layer virtual-FPGA. The first two layers contain six LABs and the third layer has three LABs reducing the six input bits to three output bits. The inputs to the controller are six proximity sensors which have been digitised, the sensor model can be found in section 3.1.2. The output bits have been combined to provide eight combinations which provide the eight robot motions.

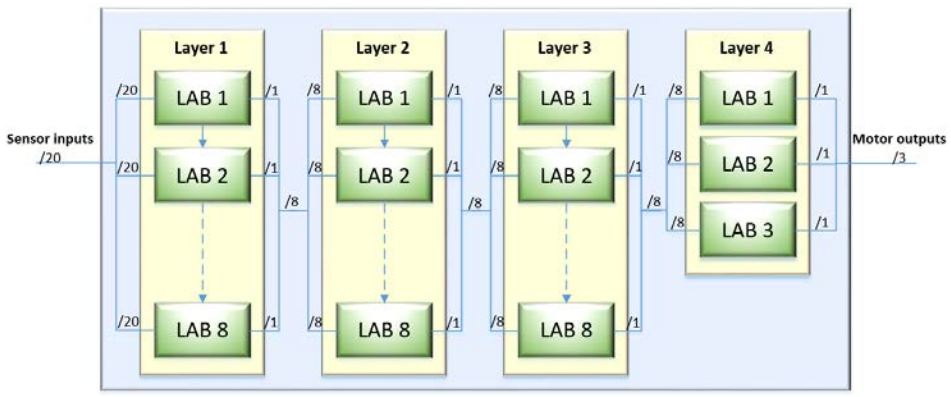


Figure 6-9 Object avoidance virtual-FPGA

The evolved controller contains a total of 27 LABs. Although unlike the virtual-FPGA evolved for light following, these LABs remain the same for each layer of the virtual-FPGA. The complexity of the input layer and the LABs has been reduced due to the reduction of input bits from the robot sensors. The overall reduction in complexity of the systems means that the required bit-stream length is also reduced.

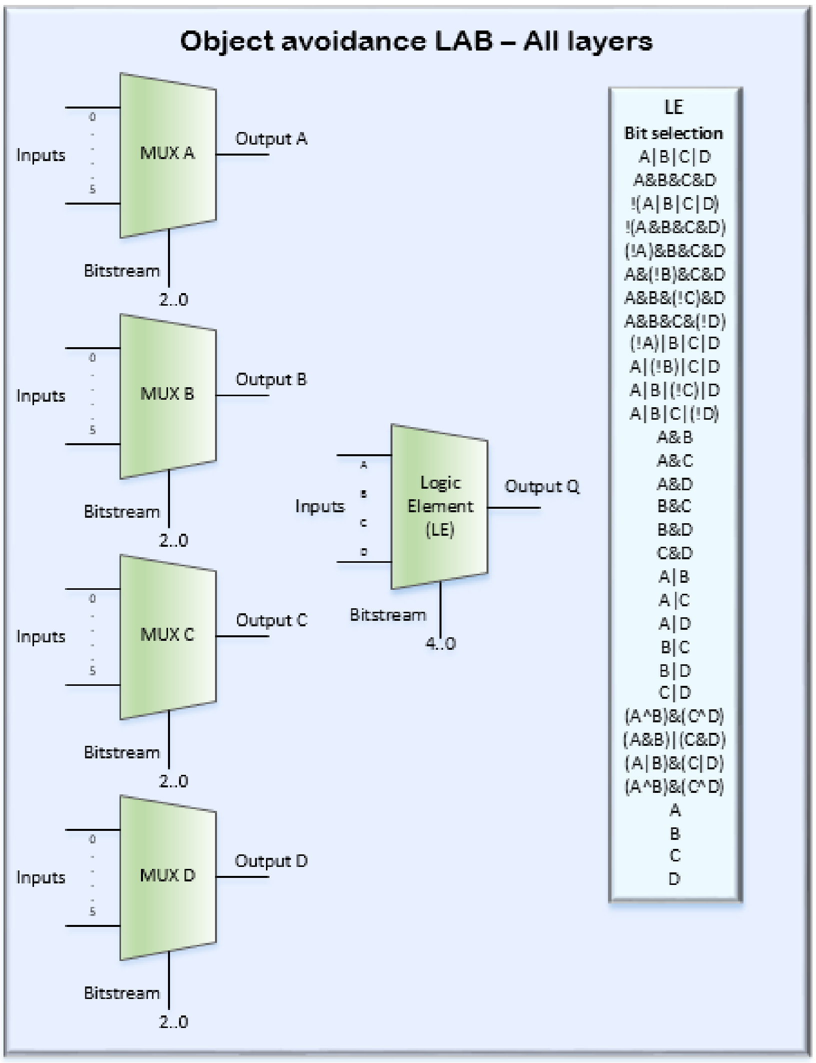


Figure 6-10 Object avoidance LAB configuration

Each LAB is comprised of two stages. The first stage selects between the six inputs and passes the selected inputs to the second stage. The second stage performs one of the thirty two available logic operations on the selected inputs and passes the result on to the next stage of the virtual-FPGA. The virtual-FPGA is fully configurable with a total of 459 bits comprised of 27 LABs each requiring 17 bits to configure, giving a search space of 1.5x10^{138}.

6.3.2 Results

A set of experiments has been designed to evaluate an EHW controller for object avoidance behaviours and the efficiency of the GA used to evolve it. The results obtained from these experiments are outlined in the following two sections. The first section focuses on the controller performance while the second section focuses on the evolutionary efficiency.

The fitness function used to evolve the obstacle avoidance controller considers two variables, the robot's total runtime and the movement count. Both values are converted to a ratio with their respective maximums and used to calculate the fitness value. Note, for all of the object avoidance experiments an upper fitness limit of 80% has been set as due to the operation of the fitness function, a fitness value much above this point is unachievable. A complete description of the fitness function can be found in section 3.1.4.

EHW controller performance

A set of experiments has been developed to evaluate the EHWs ability to perform in a range of different environments. The controllers are evolved from eight starting headings in three arenas with differing object configurations. The controllers from each of the experiments are then put into unknown arenas and their performance evaluated.

The first controller has been evolved in arena A. The set of images shown in Figure 6-11 are common trajectories achieved during the evolution cycle. Note, the trajectories shown below contain eight starting headings.
0° = RED, 45° = Green, 90° = Black, 135° = Grey, 180° = Pink, 225° = Purple,
270° = Orange, 315° = Yellow

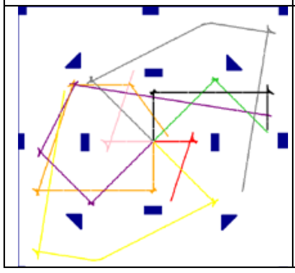
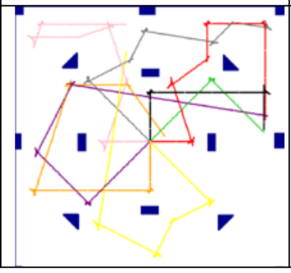
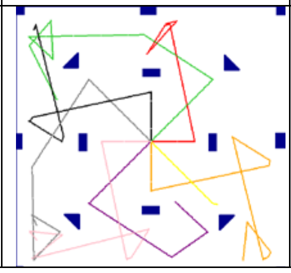
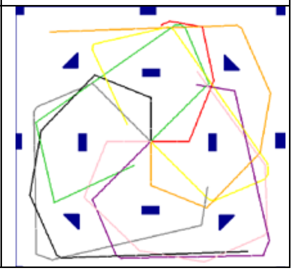
Fitness = 43%	Fitness = 51%	Fitness = 71%	Fitness = 82%
			

Figure 6-11 Arena A stages of evolution and the fitness obtained (EHW)

The paths show that several fail dependant on the angle that the robot moves towards the obstacle, however over time the fitness improves as the controller evolves the ability to avoid an obstacle when approached from several angles. The final stage of evolution is avoidance refinement, where the controller achieves a smoother path.

To compare the effect of arena selection during the evolution process on the performance of the EHW, two extra arenas have been used to develop two additional controllers. The trajectories for arena B and arena C are shown in Figure 6-12 and Figure 6-13.

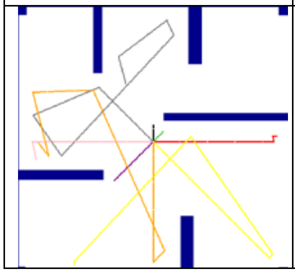
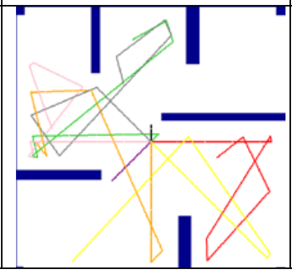
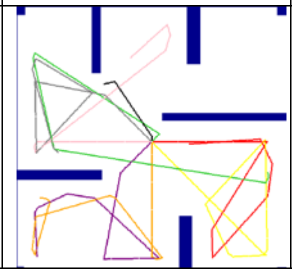
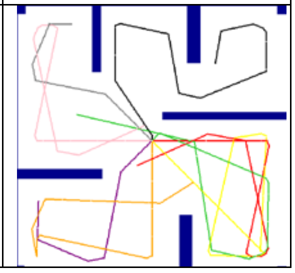
Fitness = 23%	Fitness = 66%	Fitness = 76%	Fitness = 85%
			

Figure 6-12 Arena 'B' stages of evolution and the fitness obtained (EHW)

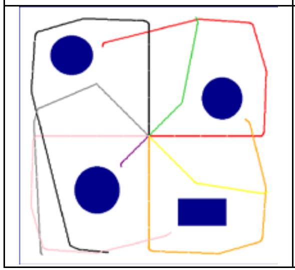
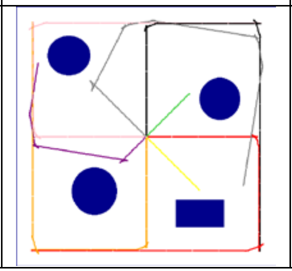
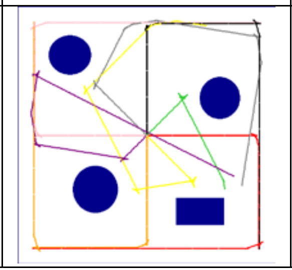
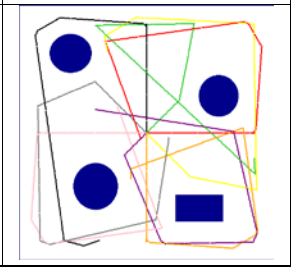
Fitness = 12%	Fitness = 56%	Fitness = 77%	Fitness = 85%
			

Figure 6-13 Arena 'C' stages of evolution and the fitness obtained (EHW)

Similar evolution patterns to those found in the trajectories from arena A can be seen in the trajectories from arena B and arena C. In the initial stages of evolution the controller develops basic object avoidance behaviours which are fine-tuned with the trajectories becoming more refined in the later stages of the evolutionary cycle.

The evolved controllers from arena A, B and C have been evaluated using three new arenas. The trajectories from each controller in the arenas are shown in Figure 6-14. It can clearly be seen from the trajectories shown in Figure 6-13 that the controllers in most of the tests exhibit some object avoidance behaviours. However when placed into an unfamiliar arena, these controllers do not perform to the same standard as achieved in the original arenas.

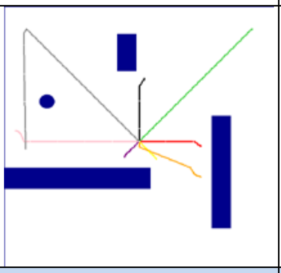
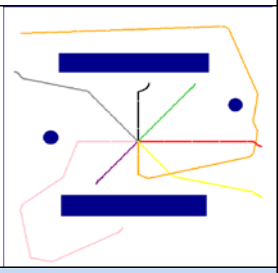
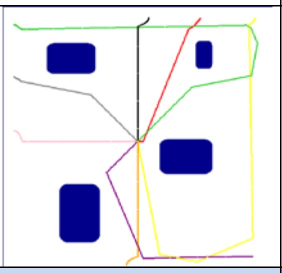
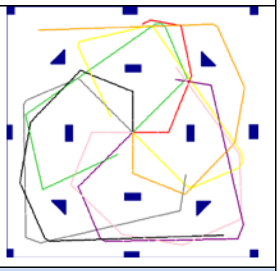
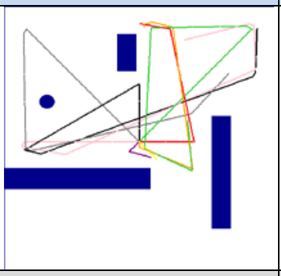
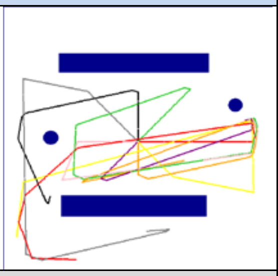
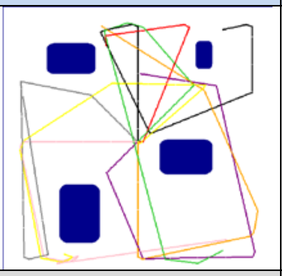
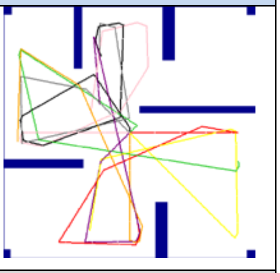
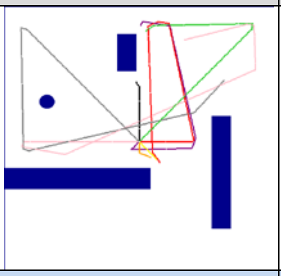
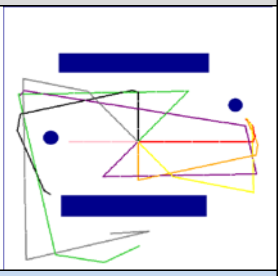
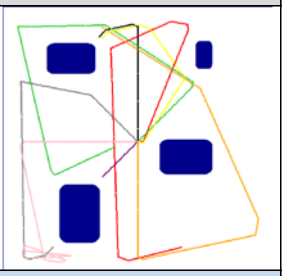
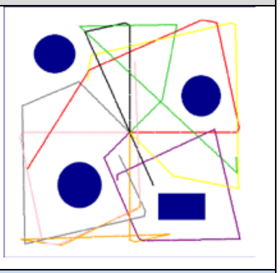
	Arena 1	**Arena 2**	**Arena 3**	**Original arena**
Controller A				
	Fitness = 10%	Fitness = 12%	Fitness = 9%	Fitness = 82%
Controller B				
	Fitness =46%	Fitness = 57%	Fitness = 70%	Fitness = 81%
Controller C				
	Fitness =23%	Fitness = 35%	Fitness = 45%	Fitness = 81%

Figure 6-14 Controller testing in unfamiliar arenas (EHW)

Multiple arena evolution

To overcome the incomplete evolution problem, a change has been made to the evolution process. For this test the controllers have been evolved in arenas A, B and C simultaneously. The average fitness from the combined arenas is used to determine the fitness for the individual. Note, the evolved controllers both achieved fitness's of at least 80%.

The resulting trajectories from the combined controllers under test in the unfamiliar arenas are shown in Figure 6-15. When compared to the trajectories in Figure 6-14, a significant performance increase can be seen. Controller ABC has a fitness greater than 80% in all the unknown arenas. This is a substantial performance increase and is linked to the longer runtimes and smoother trajectories.

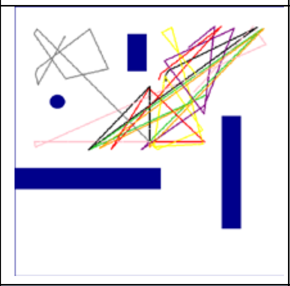
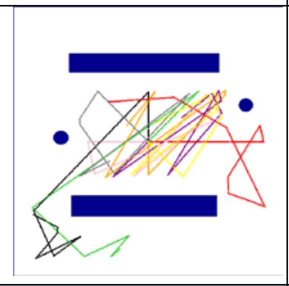
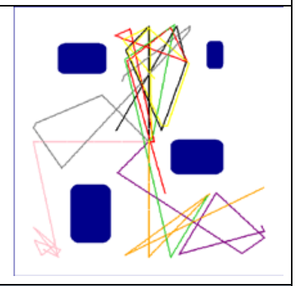
	Arena 1	Arena 2	Arena 3
Controller ABC			
	Fitness = 88%	Fitness = 88%	Fitness = 88%

Figure 6-15 Combined arena evolution test in unfamiliar arenas (EHW)

Evolutionary Efficiency

This section focuses on the efficiency of the GA process in particular the number of generations required for a suitable result to be obtained. Controllers have been evolved in arenas A, B and C separately and A, B and C simultaneously. The results are shown in Table 15.

Table 15 Arena evolution results (EHW)

Arena	Initial fitness (%)	Final fitness (%)	Generation at 80%	Arena Complexity Less to more(1-4)
C	61	86	13	1
B	58	85	24	2
A	48	83	25	3
ABC	22	88	27	4

All five controllers exceeded the desired fitness of 80%. A difference in complexity level can be seen between the arenas. Arena ABC is seen to be the most complex as it was the combination of A, B and C.

Maximum individual fitness trends for each arena configuration.

The maximum fitness plots shown in Figure 6-16 can be seen evolving at rates related to the complexity of the arena in which the controller is evolved. The fitness increments are larger for the controllers evolved in single arenas compared to the combined arena controller. The arena complexity is also evident in the number of generations required for the controllers to reach 80%, where arena ABC which is the most complex, is seen to require more generations than arena C which is the least complex. Overall the number of generations required to reach a solution is low, giving this controller a high evolutionary efficiency.

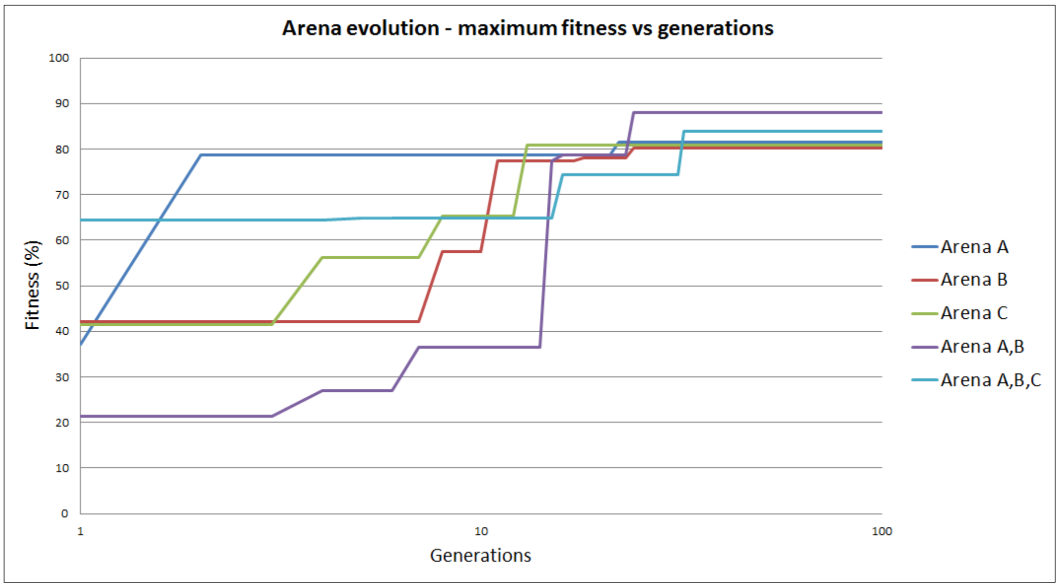


Figure 6-16 Maximum fitness comparison for each arena configuration (EHW)

6.4 EHW for light following while avoiding obstacles

This section describes the virtual-FPGA configuration used and the results obtained in the evolution of the controllers evolved to perform light following while avoiding obstacles behaviours.

Two techniques have been used to evolve the controllers. The first technique uses monolithic evolution to evolve the controller as a complete unit. The second uses subsumption evolution, where two separate controllers are evolved for individual behaviours and combined in such a way to perform the desired overall behaviour. This way scalability issues can be observed.

6.4.1 Monolithic Virtual FPGA configuration

The virtual-FPGA architecture used for the light follower and object avoidance controller is a combination of the previous controllers used for the individual behaviours. Much like the input stage used in the light follower controller there are eight LABs. However to cater for the addition of the proximity sensors to the system, the number of inputs to each of the LABs has been increased from twenty to twenty-six. Twenty of the inputs are used to provide the required information used to indicate light intensity and the remaining six provide the proximity information to the system.

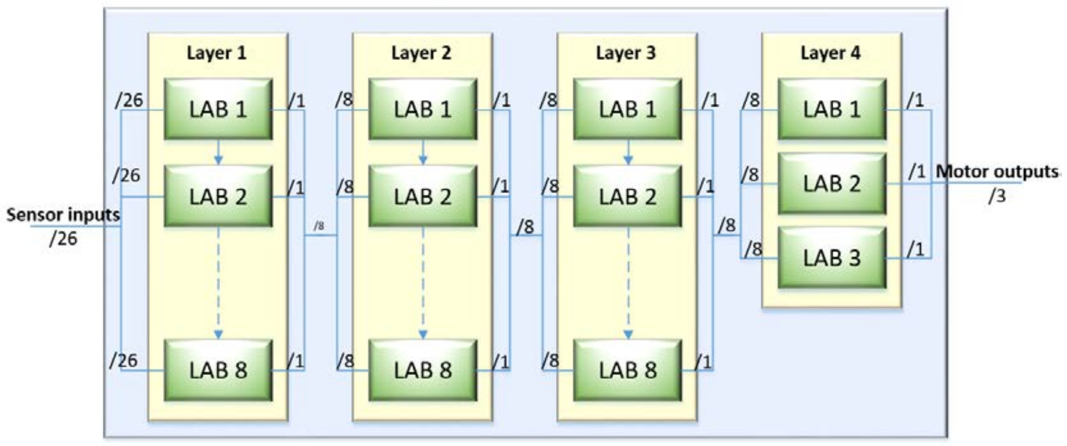


Figure 6-17, Light follower and object avoidance virtual-FPGA architecture

With the exception of layer one all of the LABs are the same and contain eight inputs and one output. The LAB in layer one provides the interface between the sensors and the virtual-FPGA system and handles a total of 26 inputs as shown in Figure 6-18.

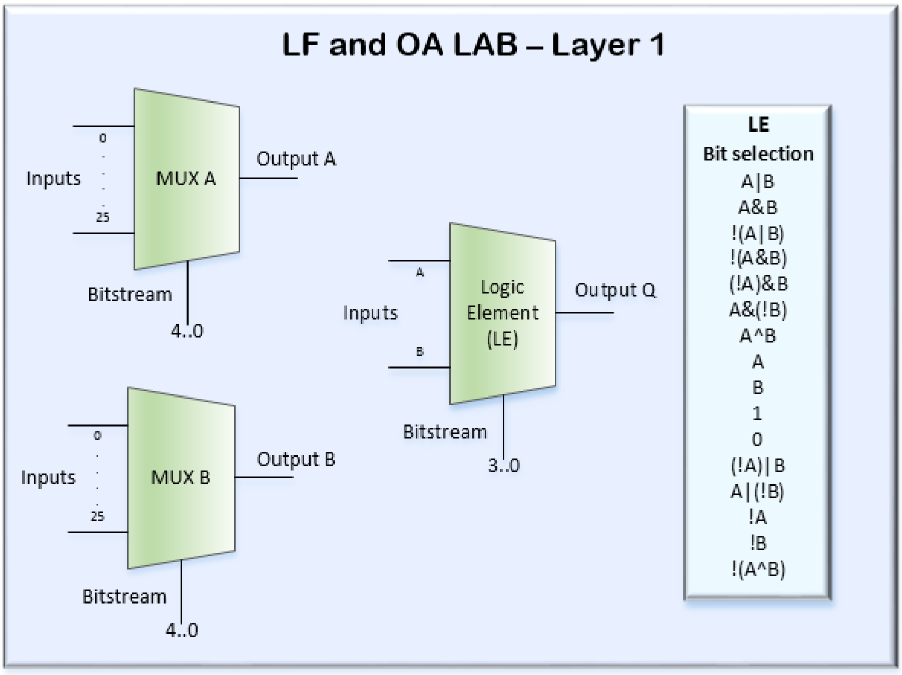


Figure 6-18 LABs in layer one of the light follower and object avoidance virtual-FPGA

Unlike the LABs used in the virtual-FPGAs for the previous behaviours, these LABs contain only two multiplexers in the input stage. This reduces the length of the bit-stream required to configure the system and reduces the overall complexity of the system. The outputs from the two input multiplexers are fed through to the logic element and put through a selection of logic expressions with the output passed on to the next layer in the system.

The remaining layers in the virtual-FPGA contain LABs with the configuration as shown in Figure 6-19.

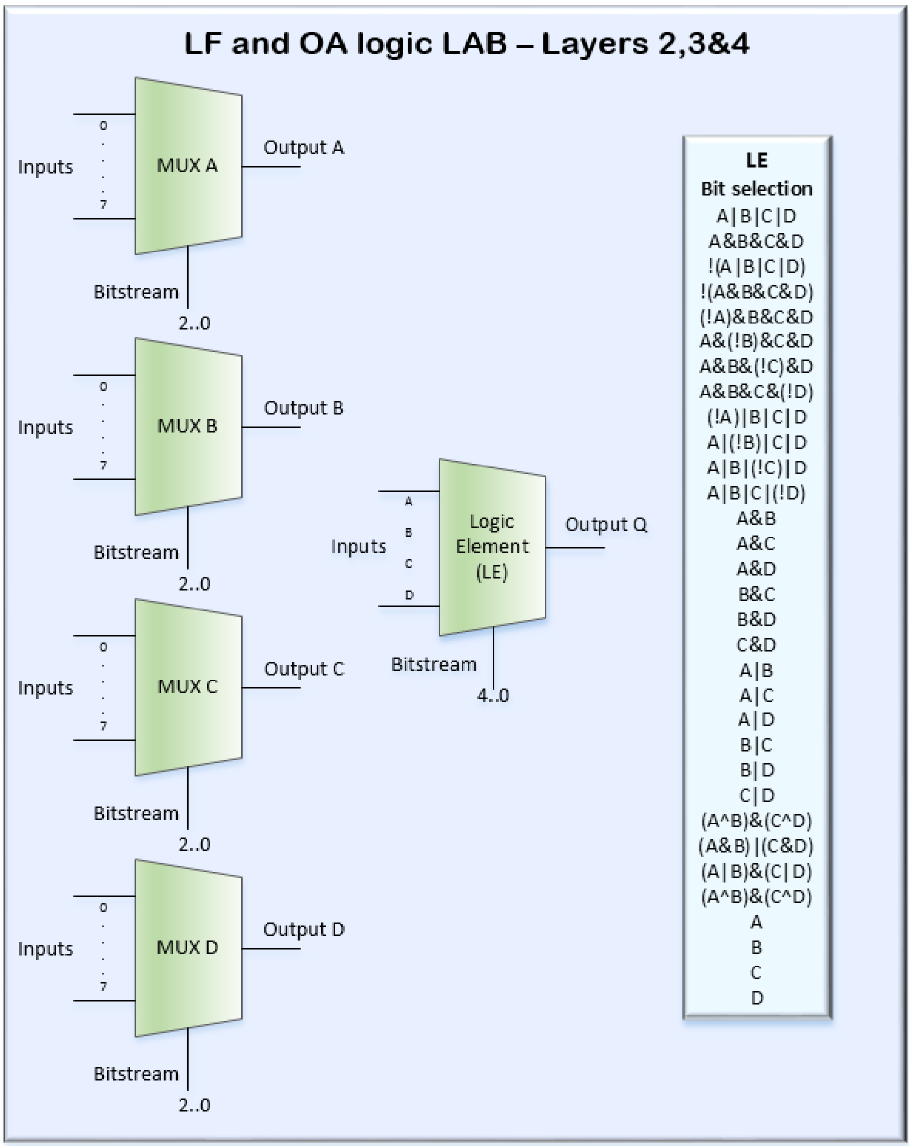


Figure 6-19 LABs in the layers 2-4 of the light follower and object avoidance virtual-FPGA

Each LAB in layers 2, 3 and 4 is comprised of an array of multiplexers designed to provide the interface to the LAB. The input multiplexers are configured to select one of the eight inputs and transfer the selected input to the LE. The LE puts the four selected inputs through one of 32 logic expressions and outputs the result to the next layer. The total system is configured using a bit-stream length of 435 bits comprised of 19 LABs in layers 2, 3 and 4 each requiring 17 bits and 8 LABs from layer 1 each requiring 14 bits. The search space for this is 8.9x10^{130}, similar to the light following virtual-FPGA.

6.4.2 EHW Subsumption evolution

The focus of this section is on the evolution of an EHW controller created to perform the combined task of light following while avoiding obstacles. This controller has been developed using subsumption evolution techniques where the complex task is broken down into individual less complicated tasks (light following and object avoidance) and evolved separately as described in sections 6.1.1 and 6.3. The evolved controllers are then combined using a switching controller which decides which controller is used and which is masked based on the sensory inputs.

Figure 6-20 is a visual representation of the subsumption based hardware controller architecture. The individual controllers for light following and object avoidance are switched by the upper level decision maker, which determines the status of the input sensors and selects the appropriate controller accordingly. If an obstacle is detected then the obstacle avoidance controller is used until the robot is clear of any, at which point the light follower controller is used.

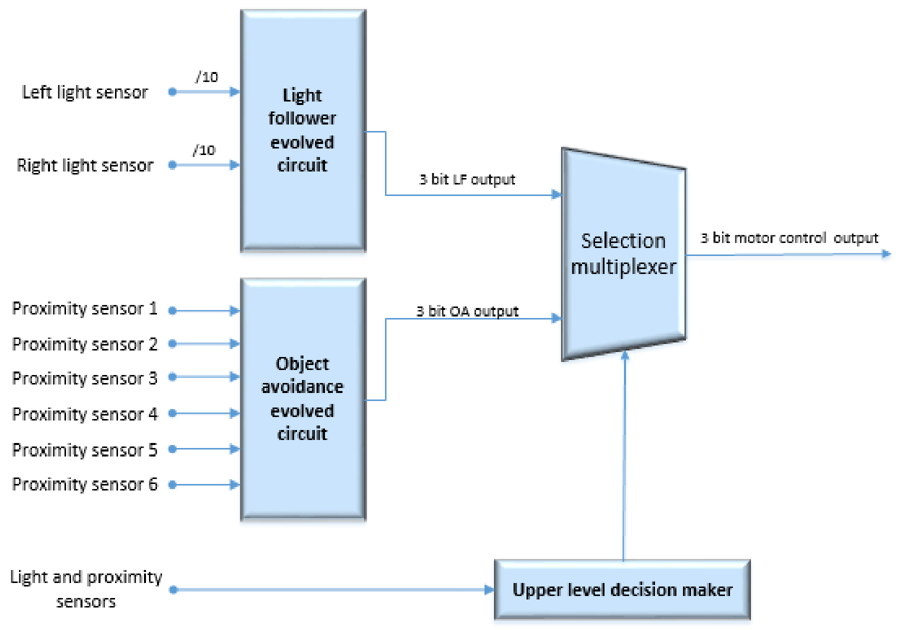


Figure 6-20 EHW subsumption diagram

6.4.3 Results

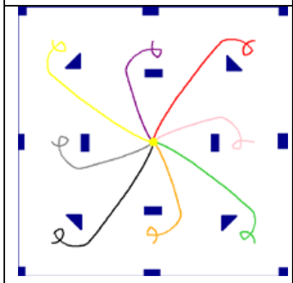
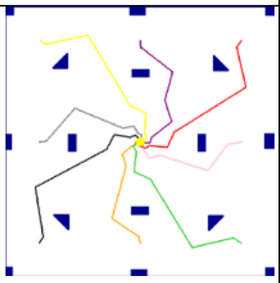
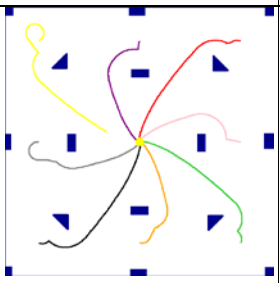
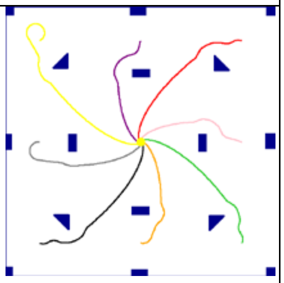
A set of experiments has been designed to evaluate the EHW controller performance for both monolithic and subsumption techniques. The method of evolution developed for these experiments starts each individual from eight different starting locations each with a unique heading. Four controllers have been evolved using the monolithic and subsumption techniques and these controllers have been compared in two ways. Firstly the controller performance is analysed and secondly the evolutionary efficiency itself is analysed.

The fitness function used to evolve the EHW controllers for the combined behaviours considers two variables; the robots distance from the light at the end of the test and the length of the path taken. A complete description of the fitness function can be found in section 3.1.4.

Note, all of the experiments are undertaken using arena A and an upper fitness limit of 80% or 3000 generations has been set for the monolithic evolution. A controller receiving a fitness of 80% is considered to be a good controller due to the way the fitness function operates.

Monolithic evolution results

The four controllers shown in Figure 6-21 perform to a very high standard and are fully developed much faster than the allocated 3000 generation limit. Each controller can be seen actively seeking out the light source and navigating around any objects encountered. However different methods of avoiding obstacles can also be seen, such that some controllers are seen navigating smoothly around obstacles while others roughly avoid them. This variety of controller performance is due to the large number of possible controller configurations providing several solutions to the problem.

Controller 1	Controller 2	Controller 3	Controller 4
			

Fitness = 83%
 	
Generations = 868
	Fitness = 86%
	
	Generations = 1026
		Fitness = 87%
		
		Generations = 702
			Fitness = 94%
			
			Generations = 574

Figure 6-21 Monolithic controller trajectories (EHW)

On average the number of generations required to reach a desirable result was 800, substantially more than when the behaviours were evolved separately (Figure 6-22). Staggered progress is clear during the evolution of the controllers, in the early generations small increases in fitness can be seen which is usually attributed to one of the starting positions reaching or getting close to the light source. It is not until much later on in the evolution cycle where larger gains in fitness can be seen, these large increases in fitness occur when the majority of the starting positions manage to reach the light source.

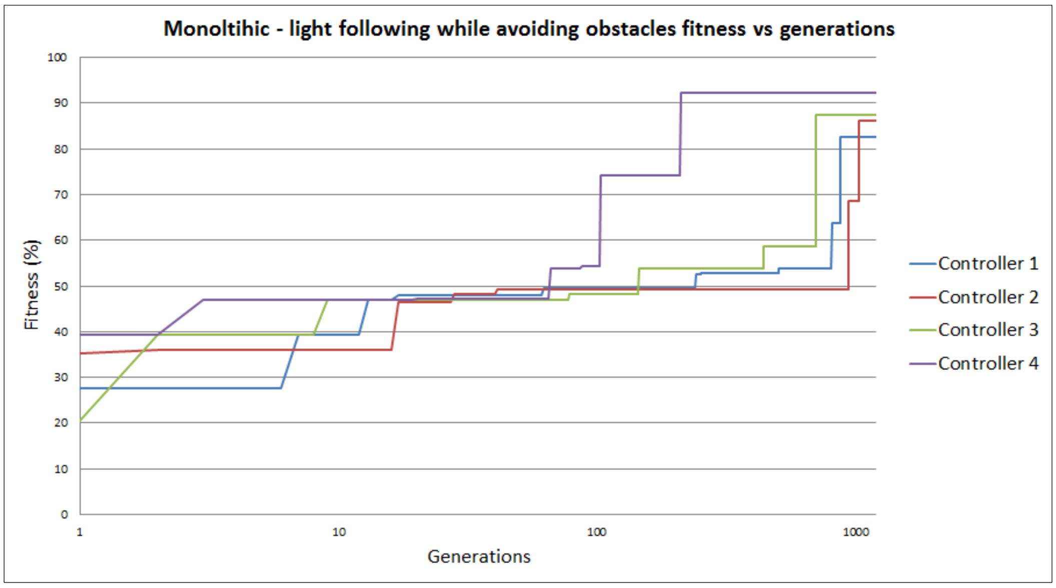
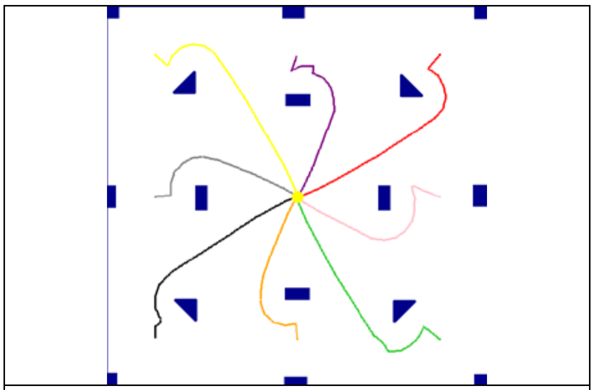


Figure 6-22 Maximum fitness trends for monolithic controller evolution

Subsumption evolution results

A previously evolved light follower and object avoidance hardware controllers was randomly chosen for the creation of the subsumption controller. The controller was run in the simulation to observe the path and corresponding fitness (Figure 6-23).



Subsumption Fitness = 89%

Generations for light follower EHW = 111

Generations for object avoidance EHW = 25

Total generations = 136

Figure 6-23 Subsumption controller trajectories (EHW)

The trajectories of the subsumption controllers show good light following and obstacle avoidance properties. For all eight starting positions the robot can be seen navigating around the obstacles and moving directly toward the light source when the path is clear.

Chapter 7

Chapter 7: ANN, LUT and EHW evolved controllers comparison

This chapter compares the three evolvable controllers, LUTs ANNs and EHWs for the navigational tasks, light following, object avoidance and light following while avoiding obstacles. The comparisons made are in evolution efficiency, controller performance, scalability and sensor quantization. The results obtained in Chapters Four, Five and Six provide the comparable data from which the conclusions have been made.

7.1 Light follower controllers

This section compares the three controllers used for light following. The comparisons are made on the effects of sensor quantization; robot trajectories; evolutionary efficiency; controller performance and controller motor direction statistics.

Note, the fitness function used to evolve the controllers is a function of the robots initial and final distance from the light source and the simulation is stopped when a fitness value greater than 95% is achieved

Quantization trajectory comparison

Figure 7-1 illustrates the final light following trajectories for the ANN, LUT and EHW controllers for five levels of sensor quantization (2-3-5-7-9). It can be seen that all the controllers have a similar controller performance. The controller performance is adequate (above 90%) for a quantization level of three and above. A quantization level of seven and above produces excellent performance.

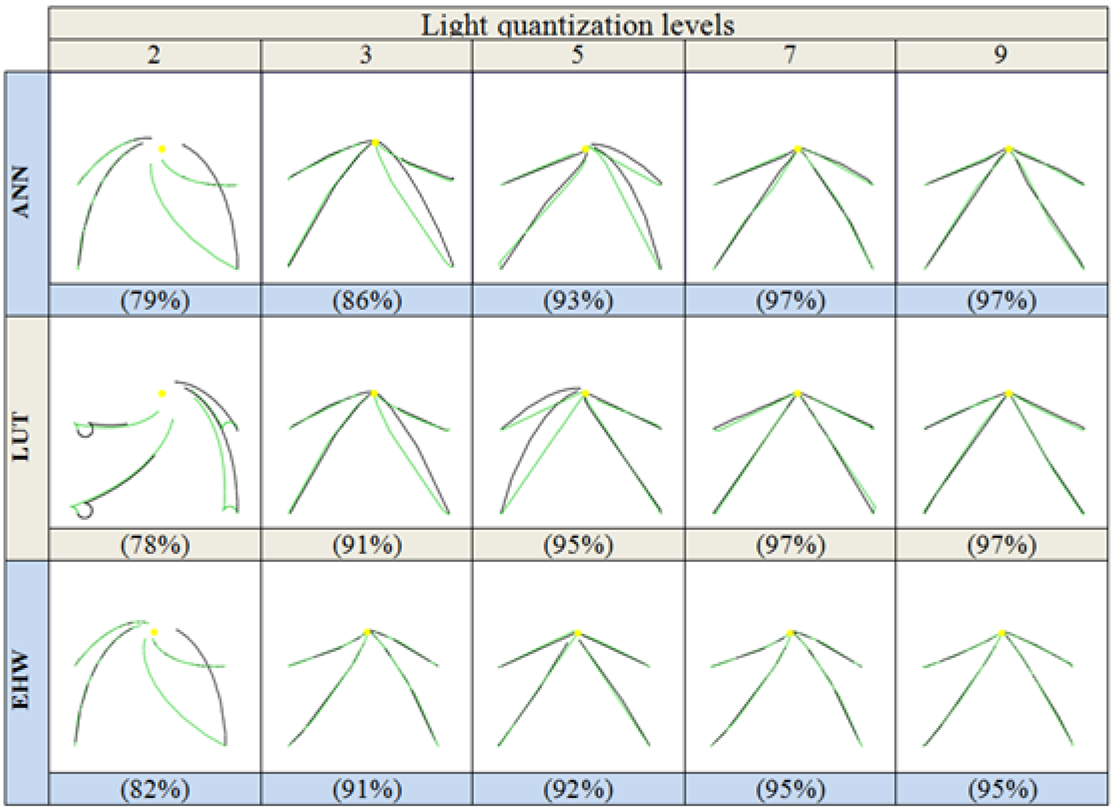


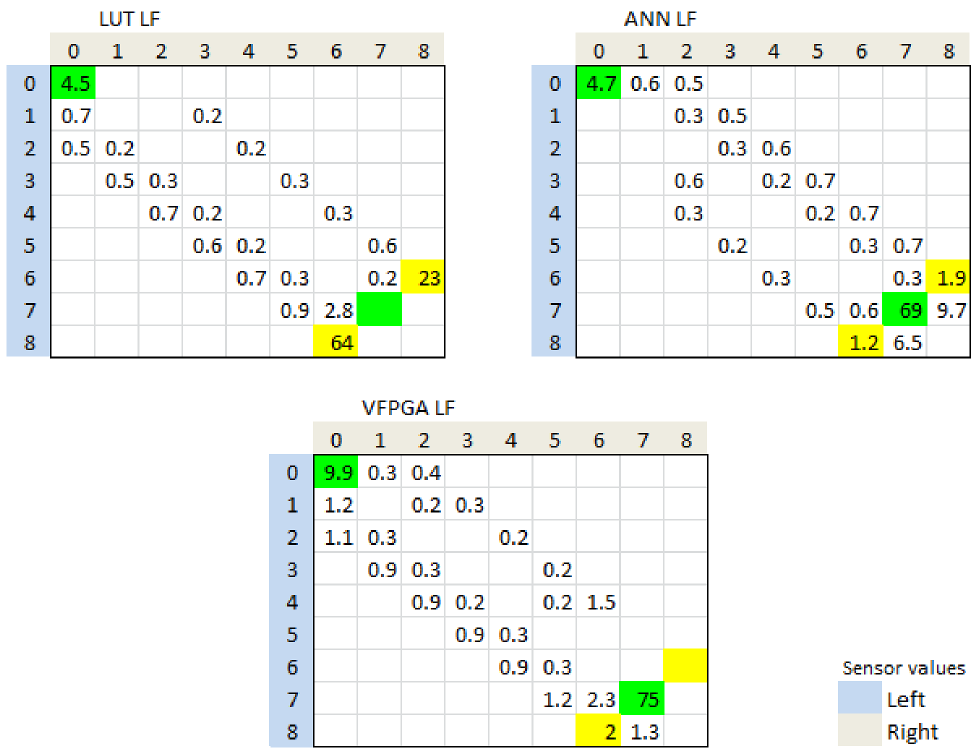
Figure 7-1 Light quantization comparison of fully evolved ANN, LUT and EHW

Controller input light levels and output motor direction statistics

A statistical analysis of the input light levels and output motor directions for all three light following controllers has been undertaken. The results are recorded during one test run of the fully evolved controllers with a light quantization level of nine. The light sensor activation points shown in Figure 7-2 represents an overall percentage of occurrences from 0-100% (0% not shown).

All of the light sensor points occur diagonally down the centre of each table. This is due to the light sensors being intentionally misaligned by 20 degrees. The light intensity is based on robot heading the majority of sensor combinations are unobtainable. For example it is impossible for the left sensor to see a maximum light level and the right sensor to see a minimum level. Two key areas in the occurrence tables have been identified as (0,0) and (7,7), the former when the robot is facing away from the light and the latter, when the robot is facing directly towards the light source. These points (highlighted in green) indicate areas where the robot is required to pivot when no light is detected and move directly towards the light when the light value is the same for both sensors.

The LUT controller is slightly different from the other controllers. Instead of the majority of sensor values occurring at (7,7) as with the ANN and EHW controllers, the majority occurs at (8,6) and (6,8) (Highlighted in yellow). From observation it was noticed that the LUT trajectory was slightly offset from the light, and it made minor adjustments as it moved towards the light. This offset was small and had little effect on the maximum fitness.



LUT LF

	0	1	2	3	4	5	6	7	8
0	4.5								
1	0.7			0.2					
2	0.5	0.2			0.2				
3		0.5	0.3			0.3			
4			0.7	0.2			0.3		
5				0.6	0.2			0.6	
6					0.7	0.3		0.2	23
7						0.9	2.8	🟩	
8							64		

ANN LF

	0	1	2	3	4	5	6	7	8
0	4.7	0.6	0.5						
1			0.3	0.5					
2				0.3	0.6				
3			0.6		0.2	0.7			
4			0.3			0.2	0.7		
5				0.2			0.3	0.7	
6					0.3			0.3	1.9
7						0.5	0.6	69	9.7
8							1.2	6.5	

VFPGA LF

	0	1	2	3	4	5	6	7	8
0	9.9	0.3	0.4						
1	1.2		0.2	0.3					
2	1.1	0.3			0.2				
3		0.9	0.3			0.2			
4			0.9	0.2		0.2	1.5		
5				0.9	0.3				
6					0.9	0.3			🟨
7						1.2	2.3	75	
8							2	1.3	

Sensor values

▨	Left
▨	Right

Figure 7-2 Light sensor activation occurrences

The output directions of the controller have been analysed and a graph representing the percentage of each occurrence during one test run is shown in Figure 7-3. As expected, the controllers are moving forward for the majority of the time during each test run, with pivoting taking up the most of the remaining time, as well as some small course corrections.

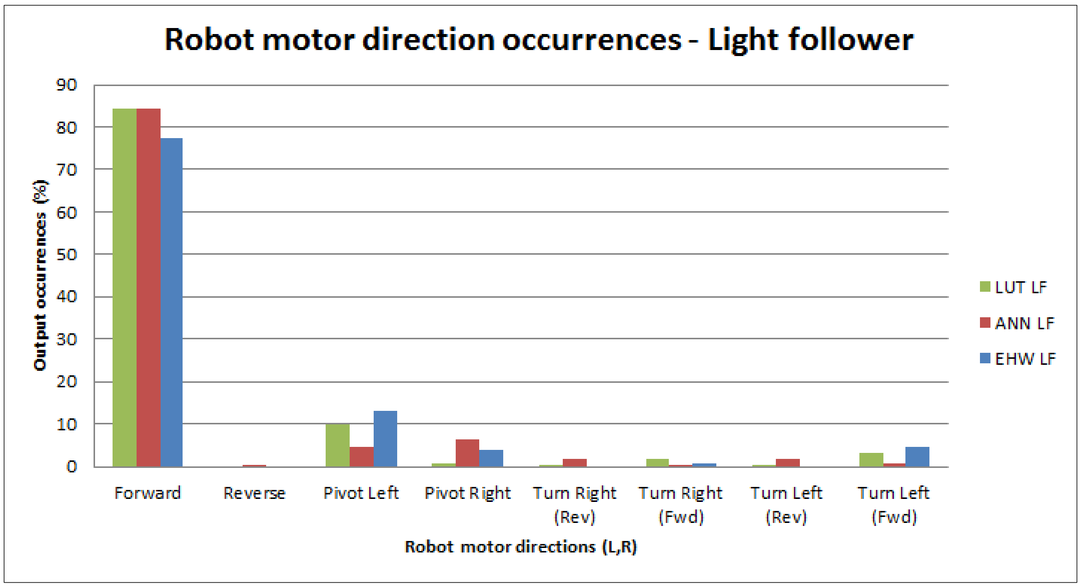


Figure 7-3 Robot motor direction occurrences for the light follower controllers

Evolutionary efficiency and search space

The plots shown in Figure 7-4 represent a typical test run for each controller platform. The evolution efficiency for the EHW and ANN controllers are much the same, both start at similar fitness levels and progress to 95% very quickly. The LUT evolution efficiency is significantly lower and hence takes much longer to evolve than the EHW and ANN.

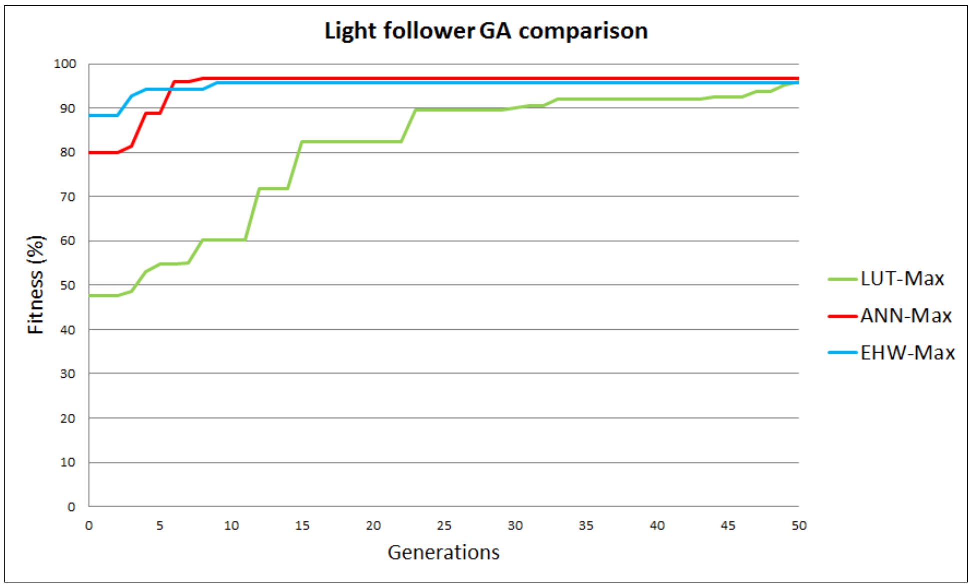


Figure 7-4 Light follower progress comparison

An initial assumption was that the evolution efficiency is related to the search space of the chromosome. However on analysing Figure 7-4 and Table 16 it was found that the controller with the largest search space (EHW) evolved in the minimum number of generations. On further investigation it was found that the EHW controllers have many possible configurations capable of performing the desired task. This means that a solution is more easily found for EHW controllers than the other controllers (ANN and LUT). In cases where a controller may be configured several ways and still perform the desired task the search space alone is not an accurate method to determine evolution rates.

Table 16 Search space vs. required generations for light follower controllers

Controller	Generation at 95%	Search space
ANN	6	$1.2 * 10^{18}$
LUT	50	$1.4 * 10^{74}$
EHW	15	$8.9 * 10^{130}$

Light follower comparison summary

In summary all the controllers evolved to an excellent level of controller performance, however the evolutionary efficiency of the LUT was less than the ANN and EHW. For applications requiring simplistic controllers, such as those running eight bit processors; the simplistic properties and minimal computing requirements of a LUT may outweigh the reduced evolution efficiency found for the LUT controllers.

7.2 Object avoidance controller comparison

This section of the thesis investigates and compares the three controller platforms and their capacity to perform as object avoidance controllers. Comparisons are made on: unfamiliar environment testing; robot trajectories; Evolution efficiency; controller performance and controller motor direction statistics.

Note, the fitness function used to evolve the obstacle avoidance controllers considers two variables, the robot's total runtime and the movement count. Both values are converted to a ratio with their respective maximums and used to calculate the fitness value and the simulation is stopped when a fitness of 80% is achieved.

Controller performance

The trajectories shown in Figure 7-5 are the results of the controllers evolved in three arenas. Clearly each controller is more than capable of performing the desired task with all controllers exceeding the required fitness of 80%.

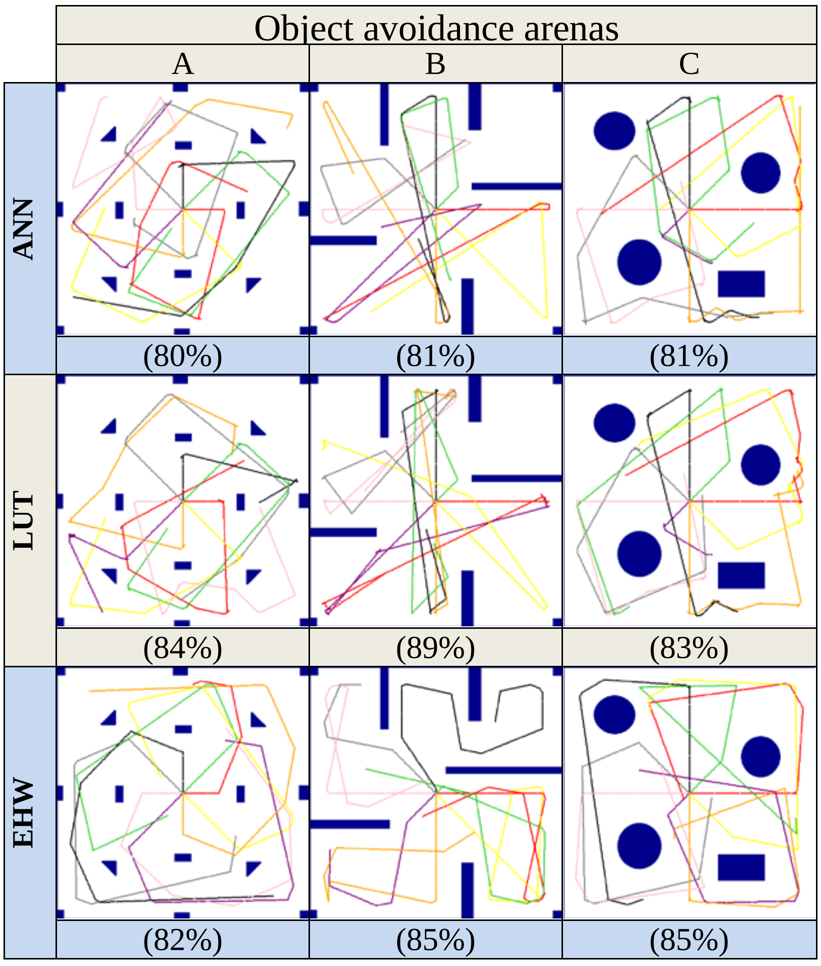


Figure 7-5 Object avoidance trajectory comparison (ANN, LUT and EHW)

Unfamiliar environment testing

To assess the obstacle avoidance controller’s ability to operate in an unknown environment, the controllers evolved in arena A have been tested in three unfamiliar arenas and the resulting trajectories compared in Figure 7-6. All of the results showed noticeable reductions in controller performance. The ANN performs better than the EHW and LUT controllers more tolerant to changes in environment. This is due to the digital circuits and LUT parameters which confine the controller’s ability whereas the ANN`s “Analogue” controller architecture allows for a more adaptable controller.

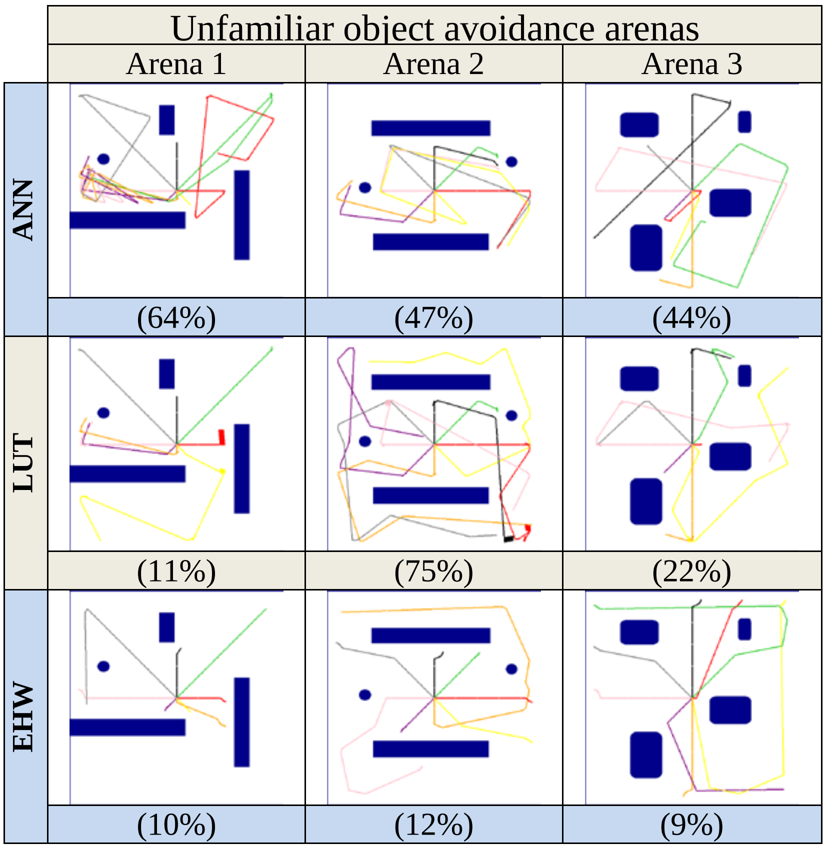


Figure 7-6 Unfamiliar arena test for controllers evolved in arena A

The reduction in controller performance (shown in Figure 7-6) is caused by incomplete evolution of the controllers. To improve the adaptability of the controllers, each has been evolved in arenas A, B and C simultaneously then retested in the unfamiliar arenas. The resulting trajectories are shown in Figure 7-7.

The test results clearly show that each controller exhibits improved performance with all of the ANN and EHW tests reaching the desired fitness level. The increased performance is related to the controller's level of evolution. A simple test environment (one arena) creates a poorly evolved controller, whereas a controller which has evolved in multiple arenas evolves and adapts to a larger range of sensor input combinations which creates a more adaptable controller.

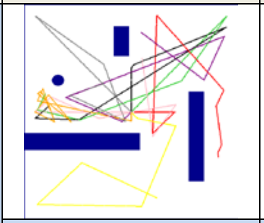
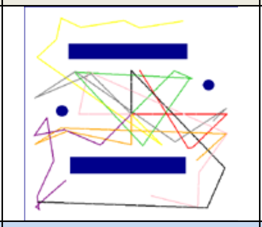
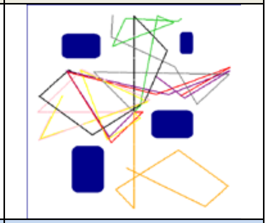
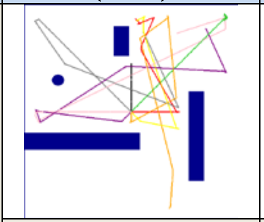
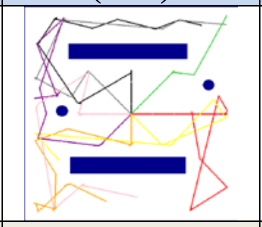
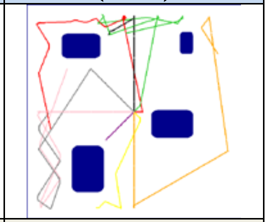
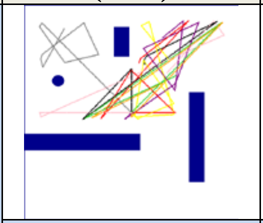
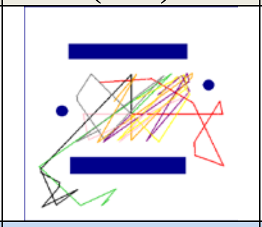
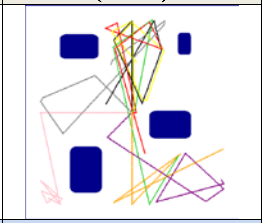
	Unfamiliar object avoidance arenas		
	Arena 1	Arena 2	Arena 3
ANN			
	(85%)	(81%)	(81%)
LUT			
	(63%)	(78%)	(55%)
EHW			
	(88%)	(88%)	(88%)

Figure 7-7 Unfamiliar test for controllers evolved in arena A, B & C simultaneously

Controller input sensor combinations and output motor direction statistics

A statistical analysis of the input object sensor combinations and output motor directions for all three object avoidance controllers has been undertaken. This focuses on the percentage of occurrences for the sensor combinations and robot motor directions. The results are recorded during one test run of the fully evolved controllers (Using the controllers evolved simultaneously in arenas A, B and C). The object sensor occurrences shown in Figure 7-8 represents an overall percentage of occurrences from 0-100%.

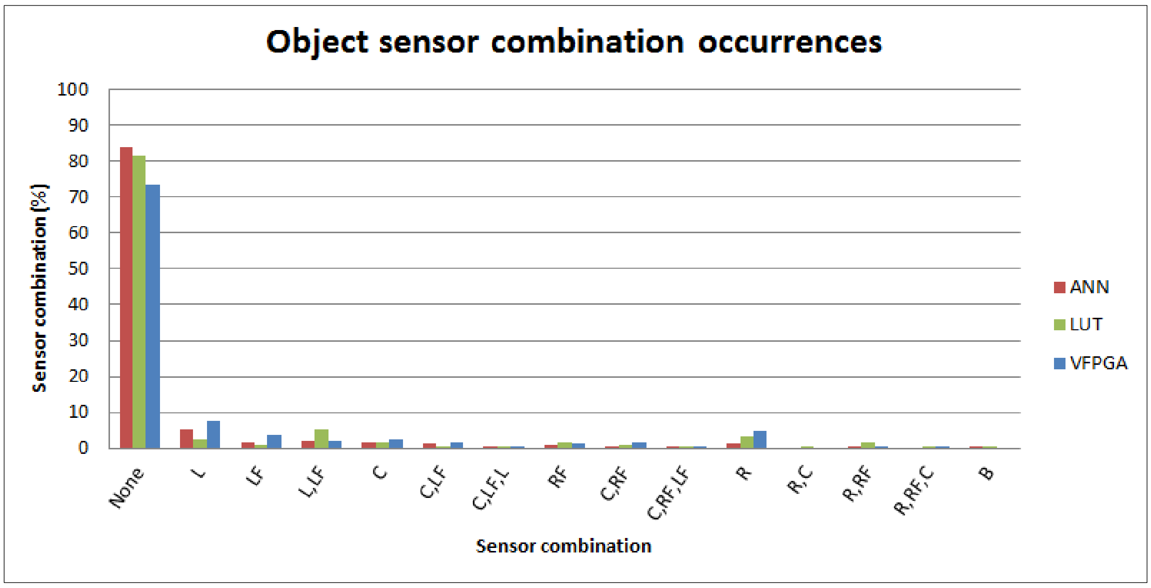


Figure 7-8 Object sensor input combinations

As expected the majority of time the robot sensors are not active (the robot is not near any objects), as the robot will alter its course when an object is detected.

The motor direction outputs for each controller have been assessed and are shown in Figure 7-9. The data has been recorded during one test run of the fully evolved controller (ABC) in arena A.

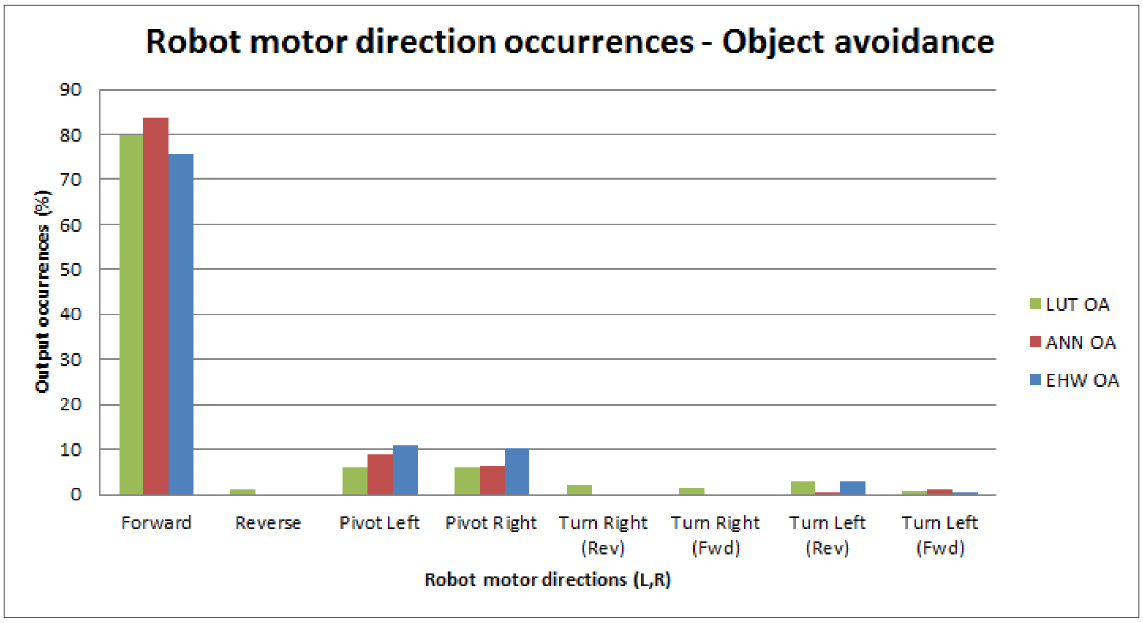


Figure 7-9 Robot motor direction occurrences for the object avoidance controllers

The majority of motor directions are forward (F, F). Pivoting left (RF) and pivoting right (FR) are the next most common directions. For the controller to be successful it needs to move straight when no objects are detected, this forces the robot to move about the arena instead of stopping or going in circles. When an object is detected the robot will pivot until the object is not seen then move forward again.

Evolutionary efficiency and search space

Figure 7-10 represents the evolutionary progression from a typical solution for each ABC controller platform. The maximum values for the ANN and EHW controllers are seen progressing at similar rates reaching 80% in a similar number of generations. The LUT controller has a poor evolutionary efficiency in comparison.

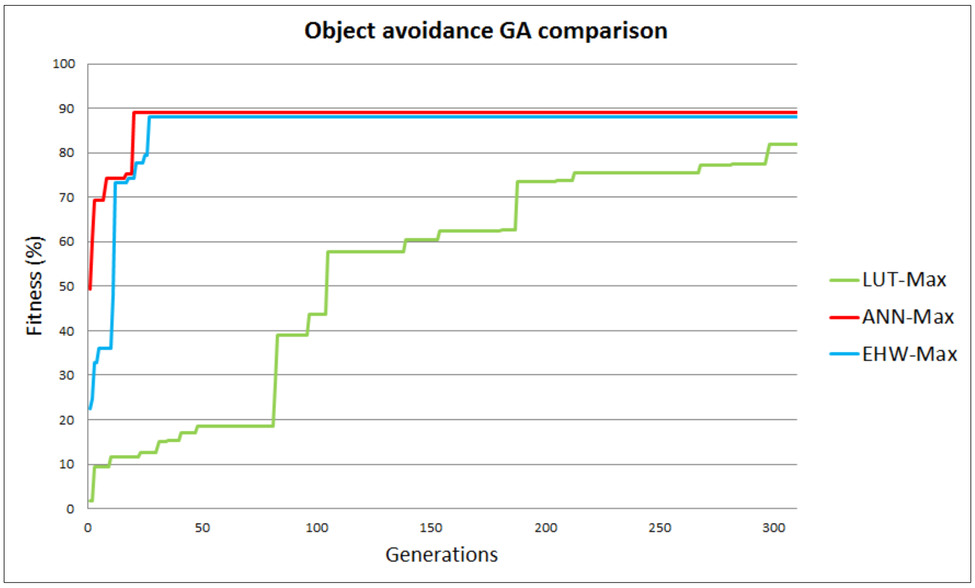


Figure 7-10 Object avoidance progress comparison

The search space (Table 17) of the ANN is much smaller than the EHW search space, however as discovered with the light follower controllers, there is no difference in the evolution rate. This is because for the EHW controller there are multiple solutions available which creates multiple maxima in the fitness landscape and hence increases the rate of evolution and the chance of finding a successful solution.

Table 17 Search space vs. required generations for object avoidance controllers

Controller	Generation at 80%	Search space
ANN	20	$1.2 * 10^{24}$
LUT	298	$6.2 * 10^{57}$
EHW	27	$5.8 * 10^{76}$

Multiple evolved circuits for the EHW controllers have been analysed and none of the circuits are the same which indicates there are multiple circuit configurations that can be created to produce the required output. Below are two examples of the logic expressions evolved for object avoidance controller. Both controllers perform object avoidance behaviours to the required standard but contain different logic.

```
BIT[0] = BACK
BIT[1] = BACK + RIGHT + CENTRE + FRONT_LEFT + FRONT_RIGHT
BIT[2] = BACK + RIGHT

BIT[0] = Always zero
BIT[1] = FRONT_RIGHT + CENTRE + RIGHT + FRONT_LEFT + LEFT
BIT[2] = Always zero
```

Figure 7-11 Two example EHW circuits for object avoidance

Object avoidance comparison summary

In summary, all the controllers have evolved to an excellent level of controller performance. The EHW and ANN controllers both have comparable evolutionary efficiencies whereas the LUT exhibits a reduced evolutionary efficiency.

7.3 Light following while avoiding obstacles comparison

This section investigates any scalability issues within the three controller types. This is tested using monolithic and subsumption evolution techniques. The fitness function used to evolve the controllers for light following and object avoidance considers two variables, the robots final distance from the light and the length of the path taken to reach the light. The simulation is stopped after a fitness of 80% is achieved.

7.3.1 Monolithic

Controller performance

Three controllers have been evolved for each of the controller platforms (ANN, LUT and EHW) and the resulting final trajectories are shown in Figure 7-12. All of the controllers have successfully evolved and are capable of avoiding obstacles and seeking the light. The ANN and EHW controllers both exhibit smooth pathways; however on average the EHW controller exhibits shorter path lengths resulting in a higher fitness.

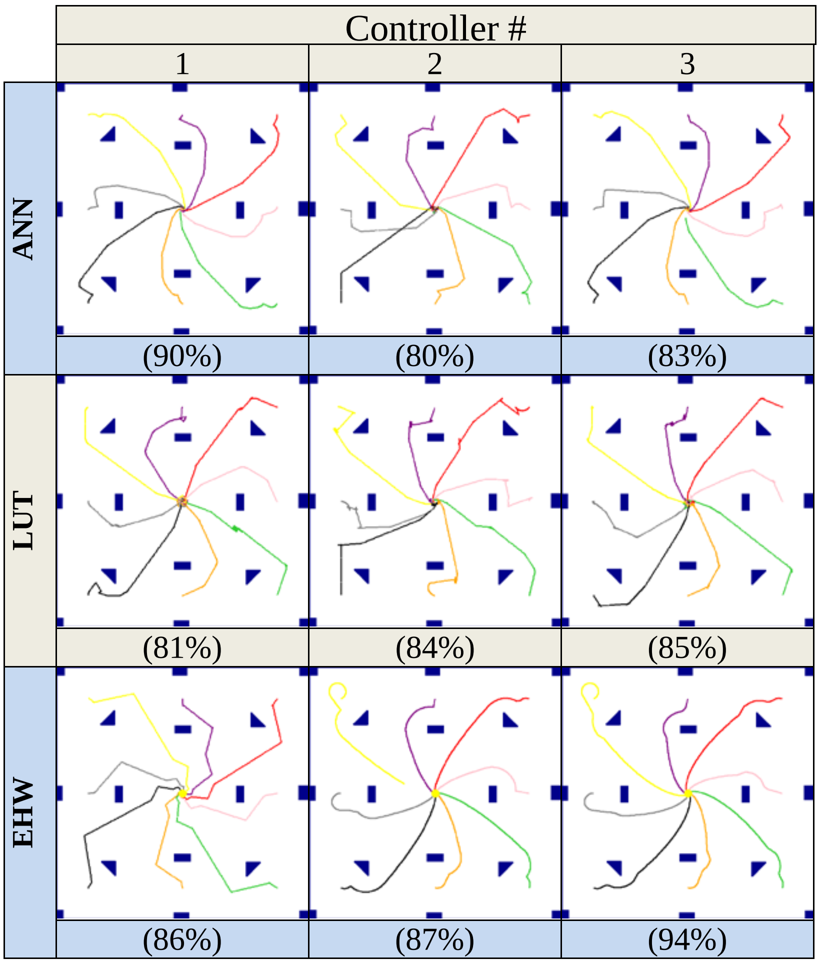


Figure 7-12 Light following and object avoidance comparison (ANN, LUT and EHW)

Evolutionary efficiency

The plots shown in Figure 7-13 represent a typical evolution cycle for each controller platform. The slowest to evolve is the LUT controller which takes more than 9000 generations. The evolution progress for the ANN and EHW controllers are very similar; however taking 850 and 702 generations respectively.

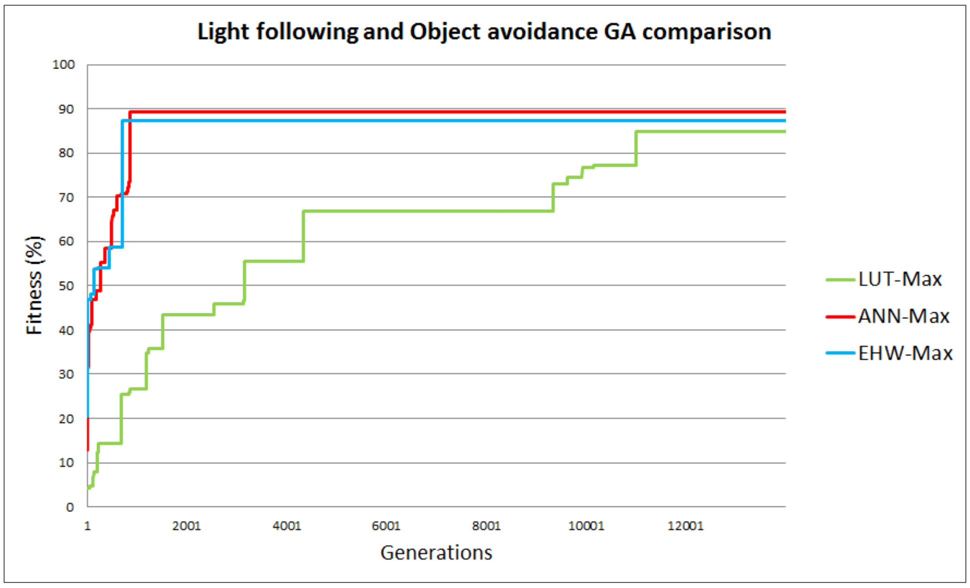


Figure 7-13 Light following and Object avoidance comparison

As discovered with the light following controllers, the search space (Table 18) for the EHW controller does not directly relate to the evolution time. In this case the EHW search space is much larger than the ANN yet the required generations to evolve are actually fewer. This is due to the large number of available hardware configurations that can produce the same required result. The LUT had an extremely large search space and yet managed to evolve a good controller in a reasonable number of generations. This is because only a part of the LUT is required to control the robot, effectively reducing the search space.

Table 18 Search space comparison (light following and object avoidance controller)

Controller	Generation at 80%	Search space
ANN	850	$1.0 * 10^{26}$
LUT	10998	$1.1 * 10^{3699}$
EHW	702	$8.9 * 10^{130}$

Controller output motor direction statistics

A statistical analysis of the output motor directions for all three light following and object avoidance controllers has been undertaken. This focuses on the percentage of motor direction occurrences. The results are recorded during one test run of the fully evolved controllers. The output directions shown in Figure 7-14 represent an overall percentage of occurrences from 0-100%.

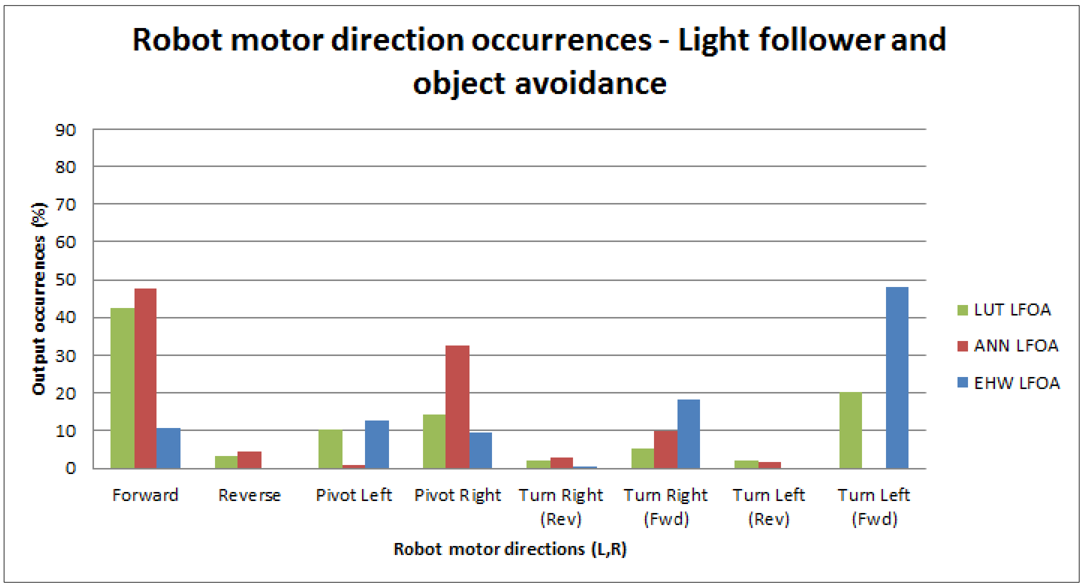


Figure 7-14 Motor direction occurrences for the light follower and object avoidance controllers

The three controllers use different approaches to find the light and avoid obstacles; the LUT moves forward for the majority of the time and pivots left (R, F) and right (F, R). The ANN also moves forward for the majority of the time but is more likely to pivot right (F, R). The EHW uses an approach that sees the robot moving in a straight line only 10% of the time. To move forward the EHW alternates between turn left (S, F) and turn right (F, S) causing a zigzag action.

7.3.2 Subsumption

The trajectories and generation statistics shown in Figure 7-15 represent subsumption controllers for the three controller platforms (ANN, LUT and EHW). The number of generations required to create the controller is a summation of the required generations from the individual controllers. The technique used to determine the controller fitness is the same as that used for the monolithic controllers.

Each controller is capable of avoiding obstacles and reaching the light source; however the controllers do not perform to the same standard. The LUT controller performs the worst, only achieving a fitness of 80%. This low fitness is caused by the forward and reverse motion used to avoid the obstacles which results in an excessive path length. The EHW and ANN controllers both perform well and achieve the desired fitness; however the path taken by the ANN controller is shorter than the EHW path so it achieves a higher fitness.

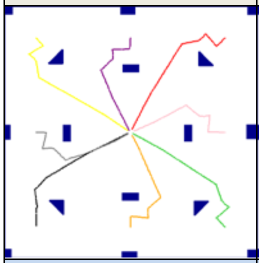
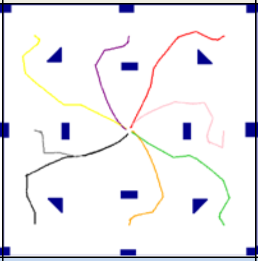
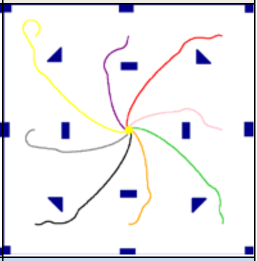
Controller platform		
ANN	LUT	EHW
		
(94%)	(80%)	(94%)
Generations(LF) = 7 Generations (OA) = 10 **Total generations = 17**	Generations(LF) = 50 Generations (OA) = 298 **Total generations = 348**	Generations(LF) = 15 Generations (OA) = 27 **Total generations = 42**

Figure 7-15 Subsumption trajectory controller comparison

7.3.3 Scalability

The trajectories shown in Figure 7-16 are typical results obtained from light following and object avoidance controllers evolved using monolithic and subsumption techniques. It can be seen that the ANN and EHW perform well with increases in complexity, however the LUT is more affected, taking ten times the number of generations to evolve compared to the others.

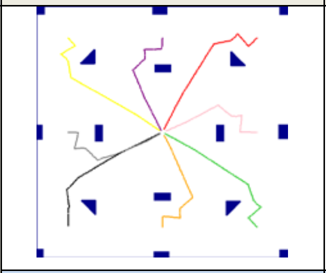
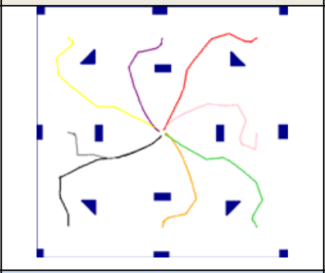
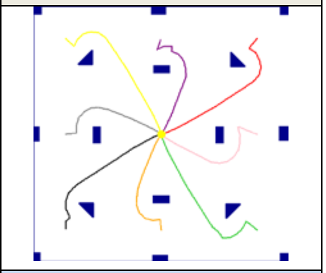
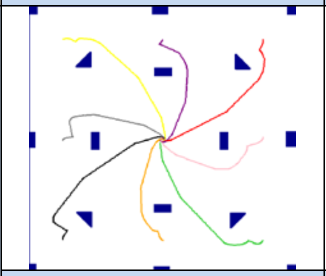
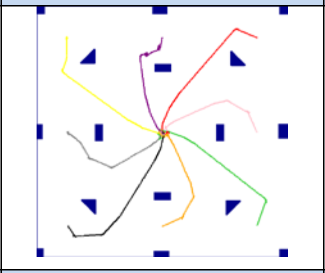
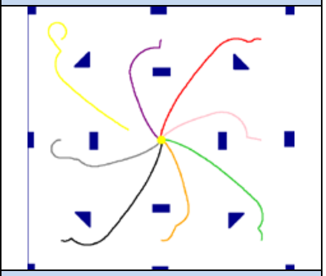
	Controller platform		
	ANN	LUT	EHW
Subsumption			
	(94%)	(80%)	(89%)
	Generations = 17	Generation = 348	Generations = 42
Monolithic			
	(90%)	(85%)	(87%)
	Generations = 850	Generations = 10998	Generations = 702

Figure 7-16 Subsumption vs. monolithic controller trajectories

7.3.4 Light following and object avoidance comparison summary

In summary, all controllers evolved were capable of seeking the light and avoiding obstacles achieving similar controller performance. The ANN and EHW had a similar evolutionary performance whereas the LUT performed the worst. The effects of scalability were most noticeable with the LUT, with both the ANN and EHW performing well with an increase in complexity.

Chapter 8

Chapter 8: Conclusions and Future Research

8.1 Summary

To summarise, this research has investigated two novel forms of robotic controller and their suitability in the field of evolutionary robotics. LUT and EHW controllers have been developed and comparisons made against benchmark ANN controllers. The controllers have been designed to perform light following, obstacle avoidance and light following while avoiding obstacle behaviours. The controllers have been evolved using a GA which used tournament selection and two-point crossover reproduction. Comparisons were made on the generations required for the controller to evolve and the final fitness achieved. The results from the comparisons has revealed that the EHW controllers outperform the LUT controllers for all comparisons and the ANN and EHW controllers both perform very well achieving high fitness values in a minimal number of generations.

8.1.1 Question responses

Can a LUT be evolved separately for light following and object avoidance behaviours?

Two LUTs have been designed to investigate the suitability of LUTs as evolvable robotic controllers for light following and object avoidance behaviours. The LUTs elements contained the robots left and right motor directions, while the table axes were linked to the sensory inputs (left and right light sensors and proximity sensors).

Light follower controller: The light following controller was designed as a two dimensional LUT and the X and Y axes of the LUT were linked to the left and right light sensor values. To investigate the effect of GA search space on the evolution time eight levels of sensor quantization were used (two, three, four, five, six, seven, eight and nine). This resulted in LUTs sizes from (2x2) to (9x9) and search spaces ranging from 4096 to $1.4 * 10^{74}$. It was found that the controllers with lower search spaces would reach their maximum fitness earlier than those with large search spaces however due to a reduction in sensor resolution the maximum fitness achievable was reduced. All of the sensor quantization levels exhibited light following behaviours however; it was found that the best light quantization level was eight.

Object avoidance controller: The object avoidance controller was designed as a singular dimensional LUT and the X axis was linked to the bit combination of the six proximity sensors. The object avoidance controllers have been assessed on two outcomes, the first is the controller's ability to avoid obstacles and the second is on the controller's ability to avoid obstacles in unfamiliar environments. It was found that the evolved controllers were fully capable of avoiding obstacles in the arena which they were evolved however; when the controllers were tested in unfamiliar arenas a substantial reduction in performance observed. To create a more adaptable controller a new controller was evolved using three arenas simultaneously. The resulting controller tested very well in unfamiliar arenas showing a considerable increase in the overall performance.

Can a LUT be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?

Two methods of creating a light following and obstacle avoidance LUT controller have been investigated. The first method, monolithic, evolves a three dimensional LUT with light sensor values linked to the X and Y axis and the bit combination of six proximity sensors linked to the Z axis. The second method, subsumption, evolves two LUT controllers separately for the individual behaviours and combines them using a selection algorithm.

Monolithic controller: The monolithic controller is 9x9x64 three dimensional LUT, each element in the LUT contains one of eight possible motor direction combinations. Multiple controllers were evolved and all of the completed controllers we able to seek out the light source while avoiding obstacles. However, it was found that the controllers required a large number of generations to evolve. This was found to be caused by the LUTs large search space and limited number of solutions available. When compared to the ANN monolithic controller the generations required were about ten times more.

Subsumption controller: The design of the subsumption controller involved a two part process; the first was to evolve the two individual controllers separately, the second was to develop a selection algorithm that would choose which controller was used under differing input sensor combinations. The selection algorithm selected the light follower LUT when no objects were detected and the object avoidance LUT when objects were detected. This resulted in a LUT controller with a high performance level that required a minimal number of generations to produce (348 vs. 10098 for the monolithic). In comparison to the ANN subsumption controller it was found that the required generations were still about ten times more for the LUT.

Can a virtual FPGA (EHW) be evolved for light following and object avoidance?

Two EHW controllers have been designed to explore the possibilities of evolving a virtual-FPGA for light following and object avoidance behaviours.

Light following: The light following EHW controller had 20 inputs and three outputs. The 20 inputs were comprised of ten left light sensor bits and ten right light sensor bits. The three output bits were used in a bit combination to determine which of the eight motor direction combinations were selected. Eight controllers were tested using eight levels of quantization. All eight controllers showed light following abilities, however it was found that with a reduction in sensor resolution, the controllers with low levels of quantization performed poorly. The ideal sensor quantization level was nine, at this level the controllers performed very well and achieved the maximum fitness in only three generations vs. 50 for the LUT and six for the ANN.

Object avoidance: The object avoidance EHW controller had six inputs and three outputs. The six inputs contained the digital input from the six proximity sensors and the three output bits were used in a bit combination to determine which of the eight motor direction combinations were selected. The evolved controllers performed well in the arenas which they were evolved in however; when tested in the unfamiliar arenas the EHW controller performed poorly. A second controller was evolved in three different arenas simultaneously and this controller had a major performance increase surpassing the required 80% fitness.

Can a virtual FPGA (EHW) be evolved to follow a light source while avoiding obstacles using monolithic and subsumption methods?

Two EHW controllers have been designed to investigate the possibility of using a virtual-FPGA as a controller for the combined behaviour of light following while avoiding obstacles. The first is a complete controller which has been evolved using monolithic techniques. The second controller uses subsumption techniques, where smaller sub-behaviours are evolved separately and combined using a selection algorithm to perform the desired task.

Monolithic controller: The monolithic controller is a flat three layered virtual-FPGA with a fourth layer providing a three bit reduced output. The input to the VPFGA contained 26 bits left light sensor (ten), right light sensor (ten) and proximity sensors (six). The three bit output provided the bit combination used to determine the resulting motor directions. Multiple controllers were developed using monolithic techniques and all were successful in seeking out the light while avoiding obstacles and achieved and average fitness of 89%. When compared to the controllers developed on the ANN and LUT platforms it was found the EHW controller outperformed both controllers.

Subsumption controller: The subsumption controller is made up of two individual controllers previously evolved for the sub behaviours of light following and object avoidance. A selection algorithm is used to select which of the controllers is used for a given input sensor combination. The controllers created for this section of the research proved to be very efficient at avoiding obstacles and seeking out the light source.

Which of the controller platforms perform better (LUT or EHW)?

To determine which if better controller is LUT or EHW, comparisons have been made in four categories (The results are shown in Table 19). The categories are light following, object avoidance, light following while avoiding obstacles- monolithic and light following while avoiding obstacles- subsumption. In all four comparisons the EHW controllers require about 10% of the generations required by the LUT controllers to fully evolve. The EHW and LUT controllers achieve the same maximum fitness for all comparisons except for the object avoidance and subsumption. In these cases the EHW controller was superior to the LUT.

Table 19 LUT, ANN and EHW analysis

Light follower (Quantization level of nine)			
Controller type	EHW	LUT	ANN
Final fitness (%)	97	97	97
Generations (95%)	3	50	6
Object avoidance evolved in arena (A,B & C)			
Controller type	EHW	LUT	ANN
Final fitness (%)	87	83	88
Generations (80%)	27	298	20
Light following while avoiding obstacles (Monolithic)			
Controller type	EHW	LUT	ANN
Final fitness (%)	86	85	89
Generations (80%)	702	10998	850
Light following while avoiding obstacles (Subsumption)			
Controller type	EHW	LUT	ANN
Final fitness (%)	89	80	94
Generations (80%)	54	348	17

The results found in this research show that the EHW controller is an ideal development platform for evolutionary robotics. A large number of solutions exist within the EHW GA search space, this results in a controller type that exhibits very good scalability performance and is largely unaffected by an increasing GA search space. The LUT exhibits good controller performance; however, has poor evolution efficiency in comparison to that seen with the EHW and ANN. Based on these discoveries it was decided that the EHW controller is better suited for evolutionary robotics than the LUT controller.

8.2 Future Research

Real world environment testing

- The configuration and architecture of the EHW circuits could be further explored with experiments designed to investigate changes in LAB configuration and the resulting changes to controller performance.
- Speed options could be applied to the outputs of the controllers rather than only using forward, reverse and stop. This would enable complex control of the robot motions, but would greatly increase the complexity of the developed controllers.
- Investigation into fault tolerance where faults could be introduced into the controller models and the controller's ability to adapt to the faults could be assessed.
- An analysis of simulated controllers in a real world environment could be undertaken using the controllers developed in this research. Previous work undertaken has resulted in the design and construction of a two wheeled robot with light sensing and object avoidance capabilities. This robot was designed to be used as a platform for research into the field of evolutionary robotics. The robot has been designed using an Altera FPGA which is suitable for implementing hardware and software controllers. All three controller platforms will be implemented on this robot and the resulting controller performance analysed. It is expected that further adjustments will need to be made to the simulation in which the controllers are evolved to allow for environmental factors such as external noise and non-ideal robot characteristics.

References

[1] D. A. Pomerleau, J. Gowdy, and C. E. Thorpe, "Combining artificial neural networks and symbolic processing for autonomous robot guidance," *Engineering Applications of Artificial Intelligence,* vol. 4, pp. 279-285, 1991.

[2] J. Kodjabachian and J. A. Meyer, "Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects," *IEEE Transactions on Neural Networks,* vol. 9, pp. 796-812, 1998.

[3] R. Glasius, A. Komoda, and S. C. Gielen, "Neural network dynamics for path planning and obstacle avoidance," *Neural Networks,* vol. 8, pp. 125-133, 1995.

[4] A. Bartha, A. Sobe, and W. Elmenreich, "Towards the light Comparing evolved neural network controllers and Finite State Machine controllers," in *Intelligent Solutions in Embedded Systems (WISES), 2012 Proceedings of the Tenth Workshop on*, 2012, pp. 83-87.

[5] V. Abhishek, A. Mukerjee, and H. Karnick, "Artificial ontogenesis of controllers for robotic behavior using VLG GA," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 2003, pp. 3376-3383 vol.4.

[6] W. Wahab, "Autonomous mobile robot navigation using a dual artificial neural network," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, 2009, pp. 1-6.

[7] D. Harter, "Evolving neurodynamic controllers for autonomous robots," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005, pp. 137-142 vol. 1.

[8] W. Elmenreich and G. Klingler, "Genetic Evolution of a Neural Network for the Autonomous Control of a Four-Wheeled Robot," in *2007 Sixth Mexican International Conference on Artificial Intelligence, Special Session (MICAI)*, 2007, pp. 396-406.

[9] L. Seung-Ik and C. Sung-Bae, "Emergent behaviors of a fuzzy sensory-motor controller evolved by genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 31, pp. 919-929, 2001.

[10] H. Hagras, V. Callaghan, and M. Colley, "Online learning of the sensors fuzzy membership functions in autonomous mobile robots," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2000, pp. 3233-3238 vol.4.

[11] K. Sung Hoe, P. Chongkug, and F. Harashima, "A self-organized fuzzy controller for wheeled mobile robot using an evolutionary algorithm," *IEEE Transactions on Industrial Electronics,* vol. 48, pp. 467-474, 2001.

[12] S. H. A. Mohammad, M. A. Jeffril, and N. Sariff, "Mobile robot obstacle avoidance by using Fuzzy Logic technique," in *2013 IEEE 3rd International Conference on System Engineering and Technology*, 2013, pp. 331-335.

[13] M. M. Almasri, K. M. Elleithy, and A. M. Alajlan, "Development of efficient obstacle avoidance and line following mobile robot with the integration of fuzzy logic system in static and dynamic environments," in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016, pp. 1-6.

[14] B. Coffey, "Using building simulation and optimisation to calculate control lookup tables offline," in *12th Conference of International Building Performance Simulation Association*, 2011.

[15] P. K. Singh, S. Bhanot, and H. Mohanta, "Neural Control of Neutralization Process using Fuzzy Inference System based Lookup Table," *International Journal of Computer Applications,* vol. 61, 2013.

[16] P. V. S. Sobhan, G. V. N. Kumar, M. R. Priya, and B. V. Rao, "Look Up Table Based Fuzzy Logic Controller for Unmanned Autonomous Underwater Vehicle," in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2009, pp. 497-501.

[17] J. Kim, Y. G. Kim, Y. D. Kim, W. S. Kang, and J. An, "Design Based on a Shared Lookup-Table for an Obstacle Avoidance Fuzzy Controller for Mobile Robots," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, 2009, pp. 731-736.

[18] M. Beckerleg and J. Collins, "An analysis of the chromosome generated by a genetic algorithm used to create a controller for a mobile inverted pendulum," in *Autonomous Robots and Agents,* ed: Springer, 2007, pp. 145-152.

[19] J. Currie, M. Beckerleg, and J. Collins, "Software Evolution Of A Hexapod Robot Walking Gait," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 305-310.

[20] M. Beckerleg and J. Collins, "An analysis of the genetic evolution of a ball-beam robotic controller based on a three dimensional look up table chromosome," in *IAENG Transactions on Engineering Technologies,* ed: Springer, 2013, pp. 109-122.

[21] M. Beckerleg and R. Hogg, "Evolving a lookup table based motion controller for a ball-plate system with fault tolerant capabilites," in *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 2016, pp. 257-262.

[22] M. Beckerleg and Z. Chan, "Evolving individual and collective behaviours for the Kilobot robot," in *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 2016, pp. 263-268.

[23] M. Okura, A. Matsumoto, H. Ikeda, and K. Murase, "Artificial evolution of FPGA that controls a miniature mobile robot Khepera," in *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*, 2003, pp. 2858-2863 Vol.3.

[24] K. C. Tan, C. M. Chew, K. K. Tan, L. F. Wang, and Y. J. Chen, "Autonomous robot navigation via intrinsic evolution," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, 2002, pp. 1272-1277.

[25] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, pp. 267-275, 2004.

[26] R. Krohling, Y. Zhou, and A. Tyrrell, "Evolving FPGA-based robot controllers using an evolutionary algorithm," in *1st international conference on Artificial Immune Systems, Canterbury*, 2002.

[27] H.-S. Seok, K.-J. Lee, and B.-T. Zhang, "An on-line learning method for objectlocating robots using genetic programming on evolvable hardware," in *Proceedings of the Fifth International Symposium on Artificial Life and Robotics*, 2000, pp. 321-324.

[28] M. Beckerleg and J. Collins, "Evolving Electronic Circuits For Robotic Control," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 651-656.

[29] M. Beckerleg and J. Collins, "Using a hardware simulation within a genetic algorithm to evolve robotic controllers," in *Proceedings of the World Congress on Engineering and Computer Science WCWCS*, 2011.

[30] D. Keymeulen, M. Iwata, K. Konaka, Y. Kuniyoshi, and T. Higuchi, "Evolvable hardware: A robot navigation system testbed," *New Generation Computing*, vol. 16, pp. 97-122, 1998.

[31] D. Dhanasekaran and K. B. Bagan, "Fault Tolerant Dynamic Antenna Array in Smart Antenna System Using Evolved Virtual Reconfigurable Circuit," in *21st International Conference on VLSI Design (VLSID 2008)*, 2008, pp. 77-83.

[32] R. A. Brooks, "A robot that walks; emergent behaviors from a carefully evolved network," in *Proceedings, 1989 International Conference on Robotics and Automation*, 1989, pp. 692-4+2 vol.2.

[33] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal on Robotics and Automation*, vol. 2, pp. 14-23, 1986.

[34] H. Saito, R. Amano, N. Moriyama, K. Kobayashi, and K. Watanabe, "Emergency obstacle avoidance module for mobile robots using a laser range finder," in *The SICE Annual Conference 2013,* 2013, pp. 348-353.

[35] V. S. Dasmane and M. R. M. , "Implementation and analysis of real time obstacle avoiding subsumption controlled robot," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 3, p. 4, 2014.

[36] J. T. Turner, S. N. Givigi, and A. Beaulieu, "Implementation of a subsumption based architecture using model-driven development," in *2013 IEEE International Systems Conference (SysCon)*, 2013, pp. 331-338.

[37] C. Tan Tiong and M. N. Mahyuddin, "Implementation of behaviour-based mobile robot for obstacle avoidance using a single ultrasonic sensor," in *2009 Innovative Technologies in Intelligent Systems and Industrial Applications*, 2009, pp. 244-248.

[38] J. Holland, *Adaptation in Natural and Artificial Systems* MIT Press, 1975.

[39] M. Ruse, "Charles Darwin's theory of evolution: an analysis," *Journal of the History of Biology,* vol. 8, pp. 219-241, 1975.

[40] M. M. Trujillo, K. Duling, M. Darrah, E. Fuller, and M. Wathen, "Fitness function changes to improve performance in a GA used for multi-UAV tasking," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, 2015, pp. 211-218.

[41] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems,* vol. 9, pp. 193-212, 1995.

[42] M. R. Noraini and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," 2011.

[43] P. D. Sniegowski, P. J. Gerrish, and R. E. Lenski, "Evolution of high mutation rates in experimental populations of E. coli," *Nature,* vol. 387, pp. 703-705, 1997.

[44] T. Blickle and L. Thiele, "A Mathematical Analysis of Tournament Selection," in *ICGA*, 1995, pp. 9-16.

Appendix A : Acronyms

Lookup table – LUT

Evolvable hardware – EHW

Artificial neural network – ANN

Genetic algorithm – GA

Configuration bit stream – CBS

Virtual reconfigurable circuit – VRC

Graphical user interface – GUI

Finite state machine – FSM

Clockwise – CW

Counter clockwise – CCW

Logic array block – LAB

Logic element – LE

Light follower – LF

Object avoidance – OA

Field programmable gate array – FPGA

Appendix B Alteras DE2115 Development Board used for experiments