Electronics and Electrical Engineering

AIMS Electronics and Electrical Engineering, 4(3): 259–286.
DOI: 10.3934/ElectrEng.2020.3.259
Received: 21 March 2020
Accepted: 08 June 2020
Published: 02 July 2020

http://www.aimspress.com/journal/ElectrEng

Research article

A comparison of three evolved controllers used for robotic navigation

Mark Beckerleg*, Justin Matulich and Philip Wong

Department of Electrical and Electronic Engineering, Auckland University of Technology, Auckland, New Zealand

*** Correspondence:** Email: mark.beckerleg@aut.ac.nz.

Abstract: This paper compares three evolved controllers including, an evolvable hardware controller, an artificial neural network and a lookup table. The comparison made between these controllers looks at relative evolutionary efficiency, controller performance and scalability. The controllers were evolved for three navigational behaviours including light following, obstacle avoidance, and the combined behaviours of light following while avoiding obstacles. Both monolithic and subsumption techniques were used to evolve the combined behaviours to evaluate scalability. It was found that all three evolved controllers performed the assigned tasks equally well. The evolutionary efficiency and scalability of the evolvable hardware and artificial neural network were similar, whereas the lookup table had an acceptable result but was subjective to scalability. The virtual-FPGA can be implemented in a fault tolerant system using a hybrid FGPAs with a hard-core processor for continuous evolution.

Keywords: evolutionary robots; artificial neural network; evolvable hardware; lookup table; light following; obstacle avoidance; genetic algorithms

1. Introduction

There has been detailed research into evolutionary capable navigational controllers including evolvable hardware (EHW), artificial neural networks (ANN), and lookup tables (LUT), but almost no research has been performed on how these evolvable robotic controllers compare to each other. The motivation for this paper is to compare the EHW controller with an ANN and LUT based robotic controller when used for simple navigational tasks, and to suggest the systems in which they could best be used.

The navigational tasks in this study were limited to light following, obstacle avoidance, and the combined tasks of light following while avoiding obstacles. The light following and obstacle avoidance navigational tasks were chosen because they have been widely researched, providing a benchmark for the comparisons. The combination of light following whilst avoiding obstacles provides an excellent method to investigate scalability issues of the controllers. The combined tasks were evolved using two methods: a) monolithic evolution [1], where the combined tasks were evolved simultaneously; and b) subsumption evolution [2,3], where the tasks were evolved separately and then combined using a switching controller, which would override the light following controller if an object was detected. These two methods allowed the problems of scalability of the controllers to be investigated.

A comparison is made between these evolvable controllers in three areas: a) the evolutionary efficiency, determined by how quickly the controllers can evolve to a satisfactory performance; b) the controller performance, determined by both the maximum fitness achieved and physical observation of jitter and curvature in the simulated robotic path; and c) the effects of scalability as the task is increased. A standard feed-forward ANN was chosen as it is the smallest network that can achieve the required tasks, with an associated small search space. A nine by nine LUT was chosen to improve the quantization effects on the input sensors and investigate scalability issues. A twenty-bit input was used for the EHW.

The choice of which is the best evolvable controller to use is dependent on the applications they are used in. The recent incorporation of hardcore ARM processors and FPGA architectures within the same silicon substrate (including the Altera Cyclone V and Xilinx Zynq series devices), allows the use of fault tolerant EHW controllers modified by a continuous evolutionary process running on the ARM processor. This enables a single chip EHW robotic controller to be a viable alternative. The ANN requires a more powerful processor with a floating-point unit allowing multiplication of the weightings in the neurons, thus a 32/64-bit based system is required. The LUT based controllers have known limitations in relationship to scalability, however they require no computationally calculation, reducing the computer processing overheads. This makes them ideal for low processor power devices such as the common 8-bit microcontroller which could be used in a distributed control system.

One of the future practical applications of this research is the use of EHW in fault tolerant robotic controllers. Manufacturers of FPGA devices are now embedding powerful, hardcore ARM processors within the device, allowing the evolutionary process to occur in the same silicon fabrication as the EHW. The fault tolerant EHW combined with a hardcore processor would use continuous evolution, with the best evolved solution updating the active controller. Sensors showing changes or faults in the real world are used to update the simulation model allowing the evolving controller to be updated, in order to replace the active controller when a change in the real-world system occurs (Figure 1).

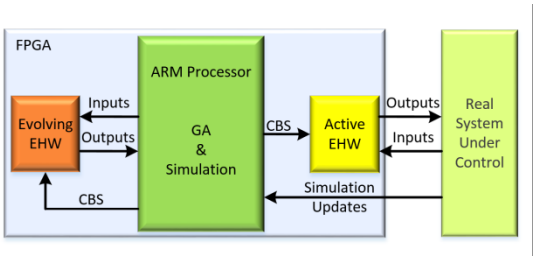

Figure 1. EHW used in an adaptive fault tolerant system.

The genetic algorithm (GA) is an optimization tool that can be used to evolve robotic controllers using a process based on biological evolution. The biological chromosome is replaced by a chromosome that determines how the robot will operate. The GA operates on a population of these chromosomes with three iterative processes: 1) fitness evaluation, where the performance of each chromosome is evaluated; 2) selection, where the chromosomes to be kept are determined; and 3) reproduction, where the selected chromosomes are combined and mutated to produce new chromosomes. The three processes are repeated until a satisfactory solution is found. The chromosome for the three controllers are: the configuration bit-stream (CBS) for the EHW; the weightings within the ANN and the parameters within the LUT.

The controllers were evolved on a simulated robot, based on a robot constructed at Auckland University of Technology (Figure 2). For motion, the robot had two wheels driven by two DC motors providing simplified steering and control. The sensors included: two forward facing light sensors for light following, six infrared distance sensors, three facing forward, two to the side and one at the rear of the robot for obstacle avoidance and five optical-reflective sensors positioned underneath the robot chassis for line following.

Figure 2. The physical two wheeled robot that was designed at the Auckland University of Technology for future research in the reality gap between simulation and real world.

2. Related work

There has been limited research into comparisons between evolvable robotic controller types. One paper was found by Pinter-Bartha et al. [4] comparing two different types of evolvable controllers, the ANN with an evolvable Mealy machine whose task was to move towards a light source. It was found that the ANN performed better than the Mealy machine. No other comparison of these robotic controllers for navigational tasks has been found. The rest of this section researches the use of a genetic algorithm to evolvable robotic controllers for light following and obstacle avoidance tasks.

2.1. Previously evolved robotic controllers

Valentino Braitenburg [5] was an early instigator of autonomous robotics, relating the actions of simple robotic behaviours to that of animal behaviours ranging from foraging (simple light following) to more complex fight or flight behaviours.

2.1.1. Evolvable hardware robotic controllers

An FPGA consists of a two-dimensional array of programmable logic array blocks (LABs) and

programmable interconnections between blocks. Digital circuits are designed using schematics or a hardware description language. These are then compiled into a CBS which is downloaded into the FPGA to configure the LABs and their interconnections. The CBS can be considered as a chromosome and the phenotype of the chromosome is the resultant circuit. Using evolutionary computation, the CBS stream can be modified with a standard genetic algorithm, then downloaded into the FPGA creating an EHW system.

The main requirements of EHW are: a) non-destructive architectures that are resistant to a destructive CBS; b) scalability, the ability to evolve acceptably as the complexity of the problem increases; c) a course-grained architecture to improve the possibility of evolving a high functioning circuit and d) partial reconfigurability, where parts of the FPGA can be reconfigured while other parts are still running.

Original research used the Xilinx XC6216 FPGA, which was suitable for evolution as its architecture was immune to destructive chromosomes [6,7]. However, this FPGA is no longer produced and other EHW methods were developed.

There are three primary approaches for the use of EHW in robotic controllers: a) genetic compilers that are specifically designed to evolve a configuration bit-stream without destructive architectures [8,9]; b) Cartesian genetic programming where the hardware descriptive language is evolved, and then converted into the bit stream [10]; and c) the virtual-FPGA implemented inside a normal FPGA.

The virtual-FPGA consists of a Cartesian based array of LABs comprised of two parts, a multiplexer for input selection, and a logic element (LE) providing a range of selectable logic functions. The operation of the LABs are determined by the CBS, thus altering this bit stream would change the operation of the virtual FPGA (Figure 3).

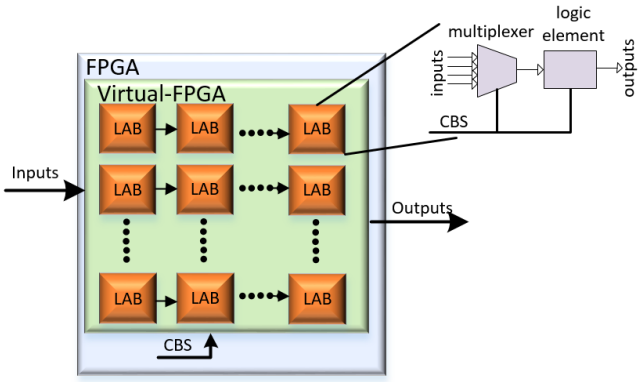

Figure 3. Virtual-FPGA showing LABs placed in Cartesian array with internal multiplexer and logic elements configured by the CBS.

The virtual-FPGA is designed with non-destructive and course-grained architectures suitable for evolution. These circuits have been evolved for: a) image processing using softcore processors [11] and hardcore processors [12,13] for the GA process; b) fault tolerant systems [14,15] and c) pattern recognition [16,17]. It has been investigated as a robot controller for a ball- beam system [18,19].

2.1.2. Evolvable artificial neural network robotic 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, called the activation function, 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, the firing threshold and the activation 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 chromosome's search space is dependent on the ANN's size, which is the number of interconnecting layers and neurons within each layer. ANN have been successfully evolved for light following and obstacle avoidance using: a) a recurrent ANN using variable length genotypes [20]; b) aperiodic firing neural networks using K-sets [21]; c) recurrent time-discrete networks [22]; d) dual networks for obstacle avoidance and target detection [23]; f) combining neural and fuzzy controllers [24]; g) a discrete time recurrent neural network evolved by NEAT (Neuro Evolution of Augmenting Topologies) [25]; h) a three layer recurrent leaky network [26], i) a deep layer neural network [27] and j) a recurrent neural network to move towards coloured objects [28], k) a single layered network evolved by a hardware based particle swarm optimization [29] l) a neurogenetic for obstacle avoidance and path planning [30], and j) a feedforward network cloned into a soft processor in an FPGA [31].

2.1.3. Evolvable lookup table robotic 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. A current example of their use in control systems is the Maxim MAX31760 controller with a programmable LUT stored in non-volatile memory. They have been researched for several control applications including building-environmental control [32], combined with fuzzy logic controllers for underwater vehicles [33], chemical process control [34] and to increase the operational speed of a fuzzy obstacle avoidance controller in a mobile robot [35].

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) a mobile inverted pendulum [36]; b) the walking gait of a hexapod robot [37]; c) a curved ball-balancing beam system [38]; d) a fault tolerant ball plate system [39]; e) robotic navigation [40], and f) robotic behaviours including orbiting, path following, follow the leader and dispersal, implemented on a Kilobot simulation [41].

2.2. Subsumption architecture

The difficulties of scalability as the complexity of the tasks increases is a considerable problem within evolutionary robotics. These symptoms include: a large amount of time for evolution; difficulties in the initial generations due to the poor fitness gradient (bootstrap problem); many local minima (deception); and the difficulties getting the simulation to match the real world (reality gap). This paper uses monolithic and subsumption to evaluate scalability issues within the three controllers under examination.

Subsumption architecture was first conceived by Brooks [2,3], 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. Several researchers have used this method for evolving robotic navigational tasks for light following while avoiding objects using various behaviours including: a) obstacle avoidance with emergency stop [42]; b) light following with object avoidance [43]; avoidance, wandering, exploring and mapping [44]; c) wandering, obstacle avoidance, goal seeking, deadlock recovery, and emergency avoidance [45] and d) a three layer feed-forward network and communication between robots to evolve group behaviours in robots [46].

3. Robot kinematics

The derivation of the robot kinematics have been produced in a previous paper [40] which details the robot motion and sensor activation. The robot motors have been limited to three speeds: full reverse; full forward; and stopped giving nine possible robot motions. The robot has been specifically designed at Auckland University of Technology for future research to investigate the "Reality Gap" between simulation and real-world parameters.

4. Common experimental systems

4.1. Graphical user interface

There are several robot simulator software packages available, including Microsoft Robot Studio, Gazebo, Actin and Webots. However, to have precise control over the software it was decided to create the GUI using Microsoft Visual Studio in the C# language. The GUI provided a means of monitoring and controlling the evolution of the controllers (Figure 4). From right to left the GUI provided: a) a visual representation of the robot and its trajectory, which could be stored; b) control and setup for both the GA, arenas and type of controller; and c) a status window showing current generation, average fitness, maximum fitness and the current elapsed time.

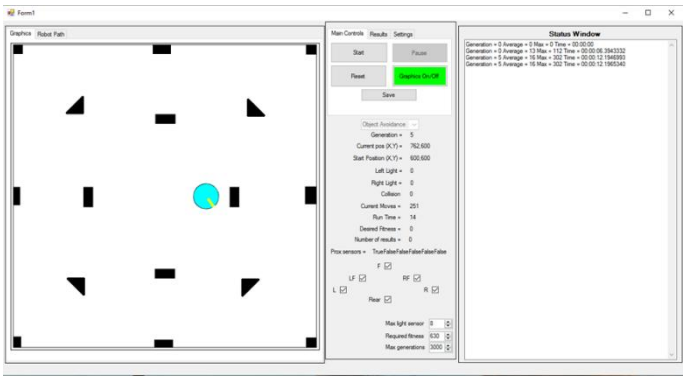

Figure 4. GUI written in Visual Studio.

4.2. Genetic algorithm

The same GA was used across all experiments. The population size was 100, with the initial

population randomized. The selection process used elitist tournament selection with a group size of two, where two random individuals are selected from the population, and only the individual with the best fitness is selected. This process is repeated until there are only 50 remaining individuals that will be used as parents for the reproduction process. The reproduction used two-point crossover with a mutation rate of 3%. The fitness function, chromosome values and structures for the three controllers are described in the following sections.

4.3. Fitness function

4.3.1. Light following fitness function

The task is to evolve the robot to turn towards the light and then move directly to it. The robot simulation had four starting positions, each with two separate headings 180 ° opposed, giving eight test runs for each individual (Figure 5). The fitness function is calculated to encourage the robot to move directly towards the light in a minimum time (1). The maximum fitness is 100, which is averaged over the eight test runs. Note the run time is set by the robot starting position and heading encouraging the robot to take the optimal path.

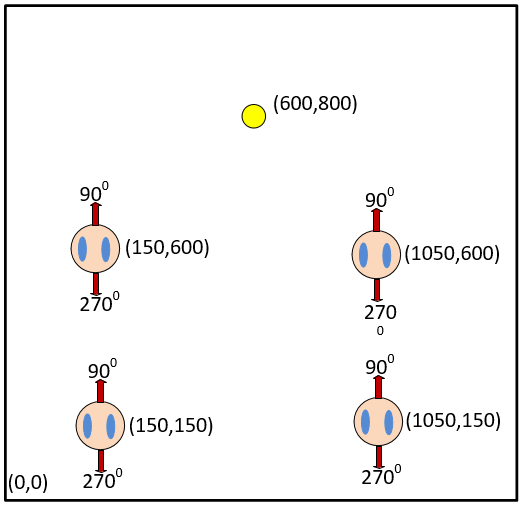

Figure 5. The Cartesian co-ordinates of the starting positions for the light following robot and light source. Each starting position was tested with two orientations of 90^0 and 270^0. (Note the arena size is 1200mm x1200mm).

$$F_{LF} = 100 - \left(\frac{final\ \ distance\ \ from\ \ light}{starting\ \ \ distance\ \ from\ \ light} \times 100\right) \quad (1)$$

4.3.2. Obstacle avoidance fitness function

The task is to evolve a robot controller that will encourage the robot to explore the arena whilst avoiding obstacles. Three arenas are evaluated, with an individual controller evolved for each robot (Figure 6).

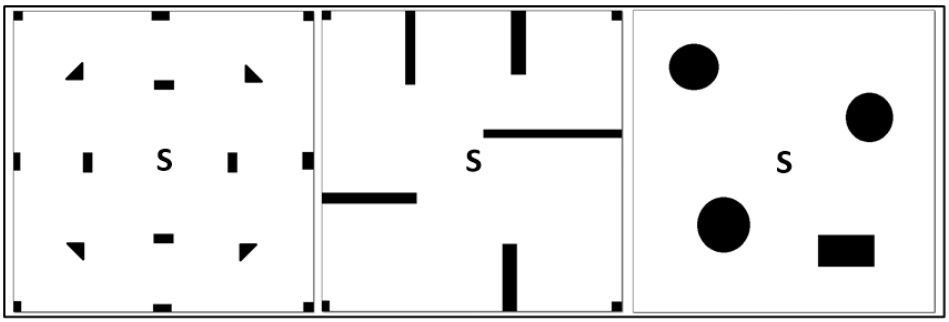

Figure 6. The three arenas used for obstacle avoidance, where S designates the starting position of the robot.

The robot simulation started from the centre of the arena with eight different starting angles, changing in 45 ° steps. The simulation was stopped either when the robot hit a wall or object, or a time period of 20 seconds was reached. A 20 second period was chosen to allow the robot to move for a maximum distance of 2.44 meters, allowing the robot to fully explore the arena and test its object avoidance. The fitness function encouraged the robot to move around the arena avoiding the walls and obstacles. A penalty was given to small circular movements of the robot to encourage it to explore the arena (2). The robot could gain a maximum fitness of 100%.

$$F_{OA} = \left[\frac{run\ \ time}{20} * 50\right] + \left[\frac{20-(circular\ \ \ movement\ \ \ counter\ )}{20} * 50\right] \quad (2)$$

4.3.3. Combined behaviours fitness function

The task was for the robot to move towards the light at the centre of the arena in the shortest time possible whilst avoiding obstacles. Eight starting points and angles were used with obstacles placed directly in the way of the light source at the centre for each starting point (Figure 7).

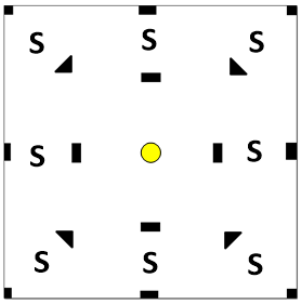

Figure 7. The arena used for the combined behaviours where S designates the starting position of the robot for each of the eight tests.

The robot simulation run time was reduced from twenty seconds to ten seconds as the task was not to explore the arena, but rather to move as directly as possible towards the light. This time period allowed the robot to travel a maximum distance of 1200 mm giving the robot ample time to achieve its goal. The simulation stopped if either the robot hit a wall or obstacle, if the run time was exceeded or if the robot reached the light source. Two separate formulae were used depending on whether the robot reached the light. The first fitness formula encouraged the robot to move towards the light

while avoiding obstacles and has a maximum value of 30 (3). The second formula encouraged the robot to take the shortest path (4). Note the second formula was only used when the robot reached the light, thus 30 from the first formula was added to give a total of 100%. As time was not directly measured, pivoting on the spot to locate the light was not penalized. The maximum fitness was 92%.

$$F_{LF-OA\ [part\ A]} = 30 - \left[\frac{Final\ \ distance\ \ from\ \ the\ light\ source}{Starting\ \ distance\ \ from\ \ the\ light\ source} * 30\right] \quad (3)$$

$$F_{LF-OA\ [part\ B]} = 30 + \left[1 - \frac{distance\ \ travelled - 600}{600} * 70\right] \quad (4)$$

5. System structures and chromosomes

5.1. Evolvable hardware

5.1.1. System overview (EHW)

The system was comprised of two parts: a) the computer running the Altera Quartus IDE, and the Visual studio with the GUI, robot simulation and GA, and b) the FPGA with a NIOS processor interfacing between the virtual-FPGA and computer (Figure 8). The simulation and GA were run on the computer, as its processor speed was almost two orders of magnitude higher than the NIOS processor. A protocol was developed that reduced the need to send data continuously between the simulation and the virtual-FPGA when testing an individual. This protocol stored the response of the virtual-FPGA on the computer for each new input state, thus if the data was already in the computer it would look there, rather than interrogate the virtual-FPGA.

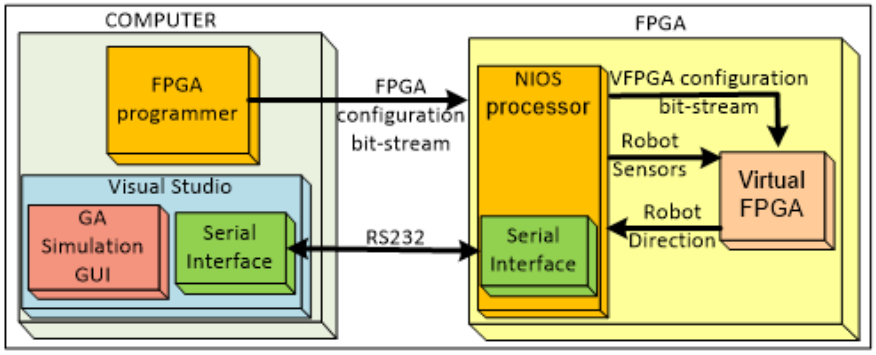

Figure 8. The complete systems running on the computer and FPGA developed to evolve the evolvable hardware controller.

In order to overcome the limitations of a normal FPGA (with a fine-grained architecture, capable of destructive routing), a virtual-FPGA was created. This was based on a Cartesian architecture (non-destructive and coarse-grained, providing a reduced search space) and implemented in the actual FPGA device. The virtual-FPGA was comprised of logic array blocks (LAB) that contain multiplexers for switching inputs, and logic elements (LE) to provide logic manipulation of the selected inputs. The following virtual-FPGA architectures described in this paper are the result of several iterations of testing including reducing and flat layer architectures as well as a variety of logic element configurations.

5.1.2. Motor driver outputs

All the versions of the virtual-FPGA had a coded 3-bit output which was used to control the robot direction (Table 1).

Table 1. The eight robot directions derived from the VEPGA 3-bit output.

Outputs	Direction
000	Forward
001	Forward Right
010	Forward Left
011	Pivot Left
100	Pivot Right
101	Reverse Left
110	Reverse Right
111	Reverse

5.1.3. Light following architecture and chromosome (EHW)

The virtual-FPGA had a twenty-bit input, comprising a maximum of 10 bits from each quantized light level sensor. The bits were encoded so that as the light level increased the bits were left-shifted on the inputs, with a level of zero equal to 0000000001 and a level of five equal to 0000111111. Unused inputs were held at zero. The three output bits were coded to provide eight possible robot motions to match the outputs used in the LUT and ANN controllers.

The virtual-FPGA consisted of three layers of eight LABs and a final stage of three LABs (Figure 9).

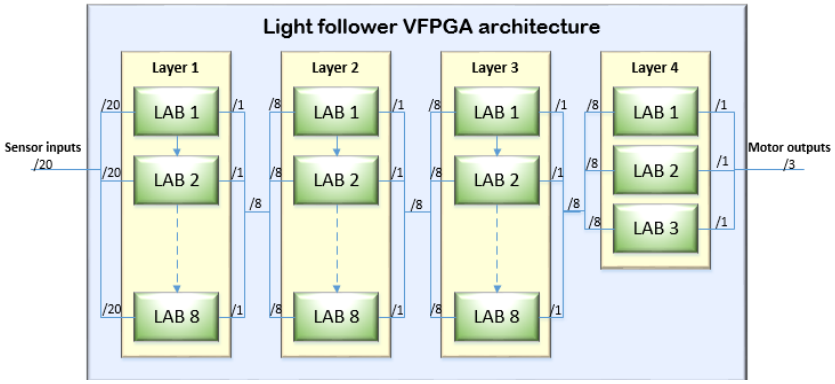

Figure 9. LAB architecture for light following VFPGA.

The LABs in the layer 1 each contained two multiplexers used to select one bit from each input source and a LE with 16 selectable logic expressions applied to the outputs of the multiplexers (Figure 10). The LABs in layers two to four had four multiplexers selecting one bit each from the previous layer, feeding four bits into a LE with 32 logic functions (Figure 11). The chromosome size for the virtual-FPGA can be calculated by the number of bits needed to control the multiplexers and logic elements. The first layer required 112-bits, (each LAB had 14-bits (MUXA 5-bits, MUXB 5-bits LE 4-bits) 8 LABs per layer1 14 * 8 = 112 bits). The second and third layers had 136-bits each,

(each LAB has 17-bits (MUXA 3-bits, MUXB 3-bits, MUXC 3-bits, MUXD 3-bits, LE 5-bits) there were 8 LABs per layer 17 * 8 giving 136 bits). The fourth layer 4 had 51 bits, (it has the same structure as layer 2 and 3, (17-bits per LAB) but only 3 LABs 17 * 3 giving 51 bits). This gave a total chromosome size of 435 bits (Figure 12).

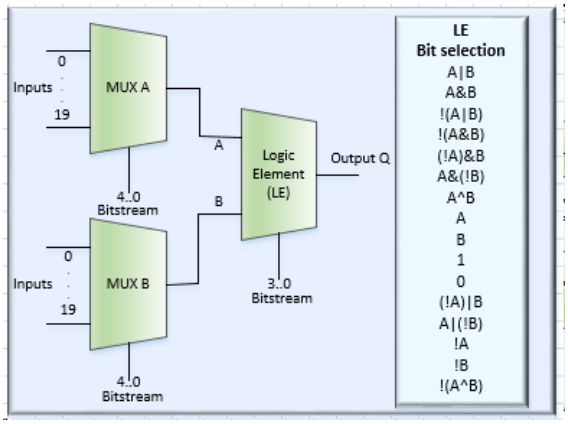

Figure 10. LAB for first layer of light following virtual-FPGA.

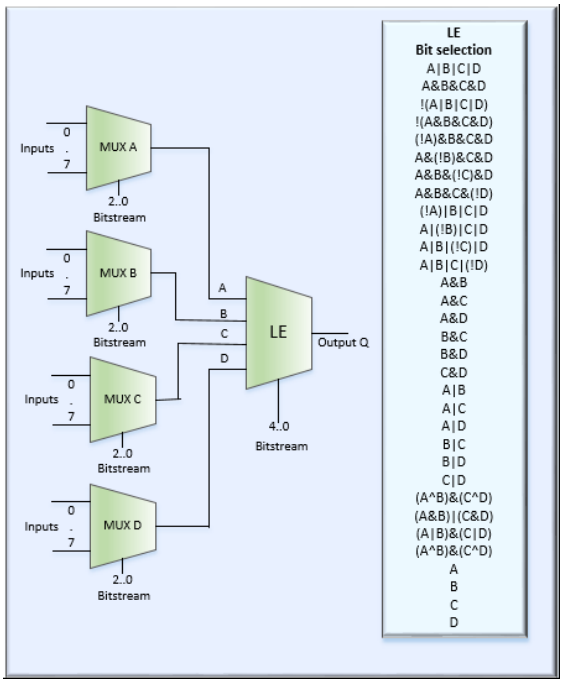

Figure 11. LAB for layers 2-4 of the light following, layers 1-4 of the obstacle avoidance, and layers 1-3 of the combine controller.

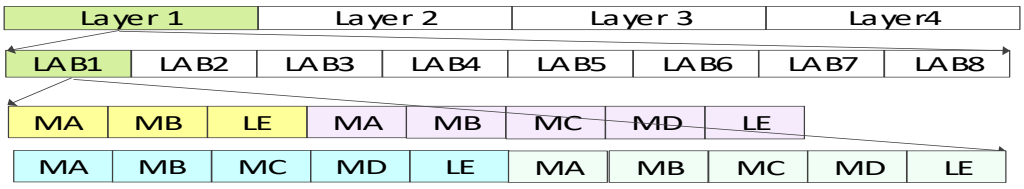

Figure 12. Pictorial representation of the EHW chromosome.

5.1.4. Obstacle avoidance architecture and chromosome (EHW)

The virtual-FPGA had six inputs from the digital obstacle sensors and a three-bit output to give eight robot directions. The virtual-FPGA architecture had three layers, with the first two layers containing six LABs, and the final layer containing three LABs (Figure 13). The multiplexers and logic elements within the LAB were the same as layers 2–4 of the light follower virtual-FPGA (Figure 11). The chromosome size required to control the multiplexers and logic elements are 255 bits. This is derived from the use of 15 LABs each requiring 17 bits, (MUXA 3-bits, MUXB 3-bits, MUXC 3-bits, MUXD 3-bits, LE 5-bits).

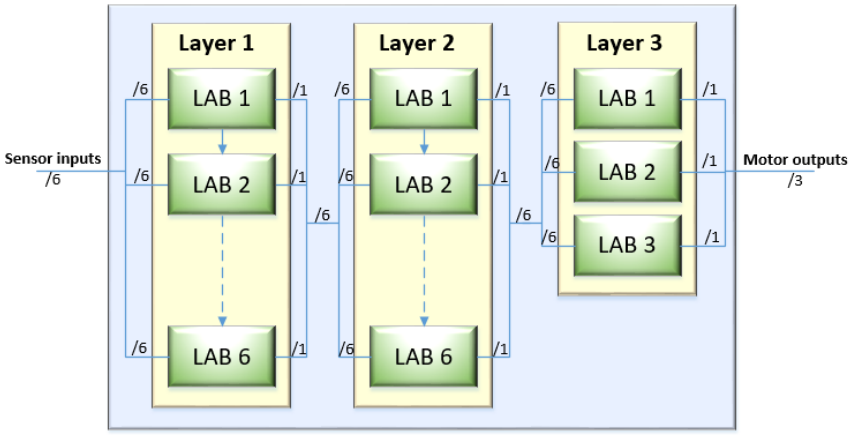

Figure 13. LAB architecture for obstacle avoidance VFPGA.

5.1.5. Combined behaviours architecture and chromosome (EHW)

The virtual-FPGA had a 26-bit input, comprising 10 bits from each light sensor and 6-bits from the proximity sensors. The three output bits were coded to provide eight possible robot motions (Figure 14). Each LAB was similar to that used in the obstacle avoidance EHW (Figure 11).

The chromosome size required to control the multiplexers and logic elements are 459 bits. This is derived from the use of 27 LABs each requiring 17 bits, (MUXA 3-bits, MUXB 3-bits, MUXC 3-bits, MUXD 3-bits, LE 5-bits).

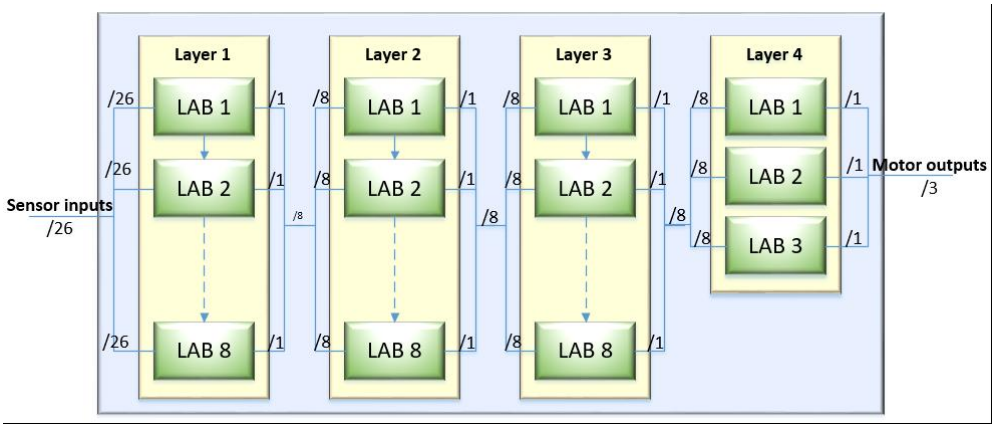

Figure 14. LAB architecture for the combined light following and obstacle avoidance VFPGA.

5.2. Artificial neural network

5.2.1. System overview (ANN)

The completed system programmed in C# contained the ANN controlling the robotic simulation, and the connections to the GA (Figure 15). This setup was similar to the LUT experiment with the LUT replaced by an ANN.

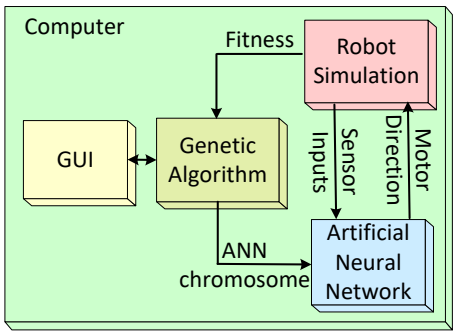

Figure 15. The system developed for evolving an ANN.

A single layered, two-neuron, feed-forward network with an input bias on each neuron was used for all the ANN controllers. The activation output of the neuron used to drive each wheel had three levels, -1, 0 and 1, providing three motor speeds forward reverse and stopped, to match the other controller outputs. The activation set points are shown as -0.5 (A2) and 0.5 (A1); these are the points at which the motor changes from reverse, to stop then forward. (Figure 16).

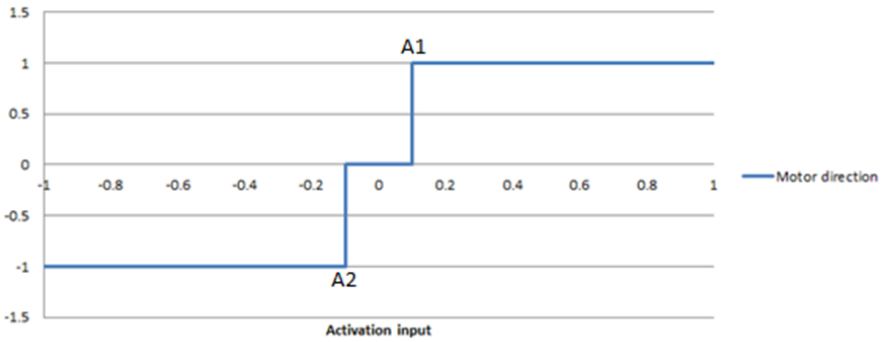

Figure 16. The three-step activation output of the neuron.

The neurons required a bias input to allow them to move when no light was detected. The ANN chromosome was comprised of the neuron's weights, the bias input and the mirrored set point of the activation output. All these parameters ranged from -1 to 1, changing in 0.1 increments.

5.2.2. Light following chromosome (ANN)

The ANN inputs were the two light sensors, which interfaced to each neuron via two weights, while the three state output of each neuron interfaced to the motor (Figure 17). In total, there were four weights, two biases and two activation set points in the chromosome. The activation set points (where the motor switches state) were part of the chromosome and altered by the GA process, ranging from -1 to 1 with a step size of 0.1.

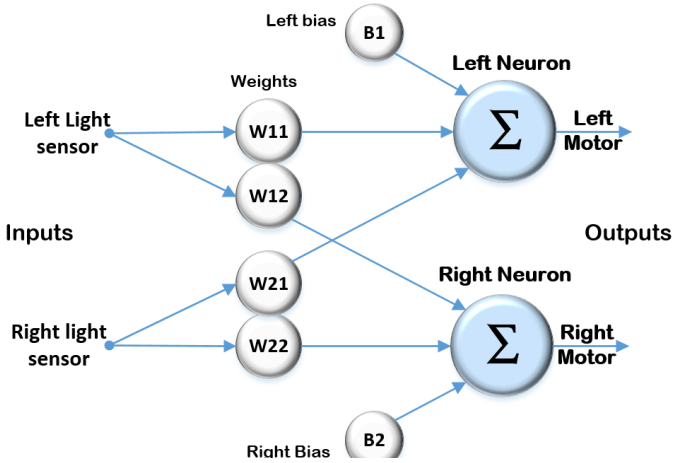

Figure 17. The single layer, two neurons, ANN used for light following.

The chromosome for this network was a combination of four weights (W), two activation thresholds (A) and one bias (B), W11, W12, W21, W22 A1, A2 and B. The weights, activation thresholds and bias values range from -1 to 1 with a resolution of 0.1 providing 21 possible values.

5.2.3. Obstacle avoidance chromosome (ANN)

The structure for the object avoidance ANN was similar to the light following ANN, except the binary output of the six obstacle sensors were interfaced to the individual neuron via six weights, giving sixteen values in the chromosome for the complete ANN (Figure 18). The chromosome for this system was 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.

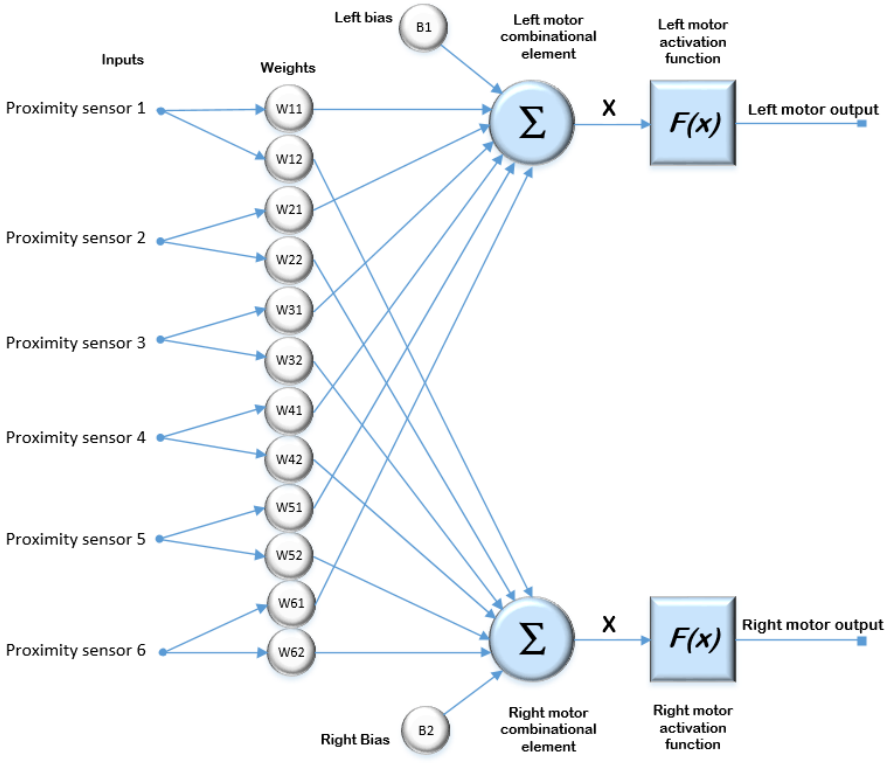

Figure 18. The single layer, two neurons, ANN used for obstacle avoidance.

5.2.4. Combined behaviours chromosome (ANN)

Still keeping with the same single layer two-neuron network, the inputs from the light sensors and obstacle avoidance were interfaced to each neuron via eight weights giving twenty values in the chromosome for the complete ANN (Figure 19). 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.

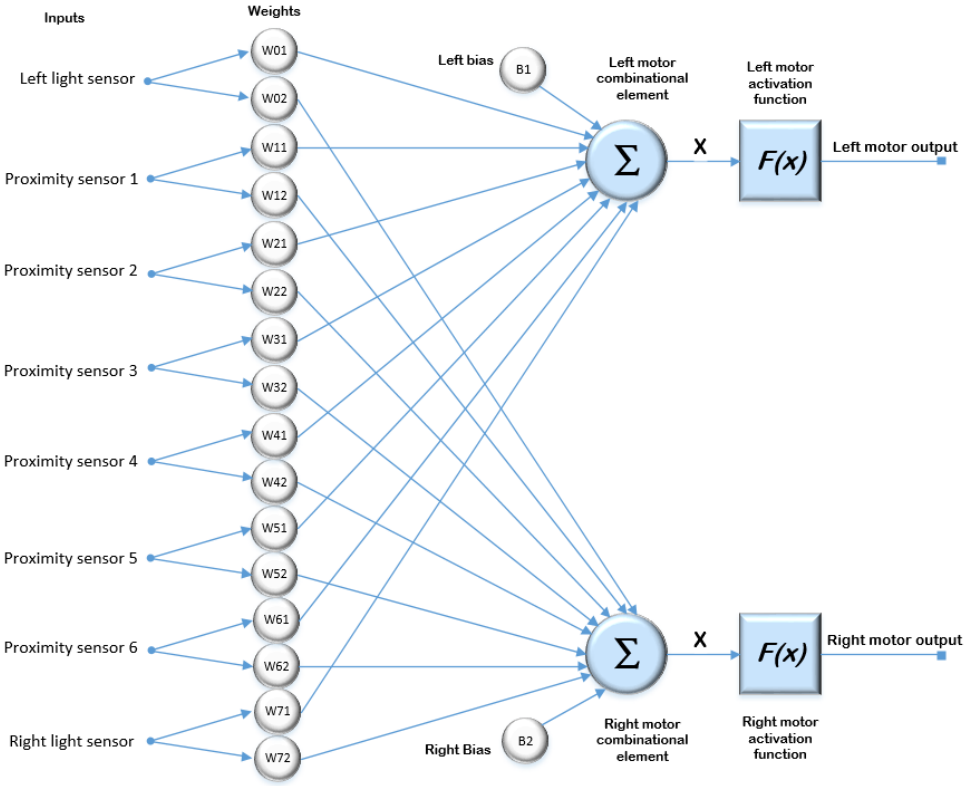

Figure 19. The single layer, two neurons, ANN used for the combined light following and obstacle avoidance.

5.3. Lookup table

5.3.1. System overview (LUT)

The systems used to evolve the LUT were developed in Visual Studio C# containing the GUI, GA, robot simulation and LUT (Figure 20).

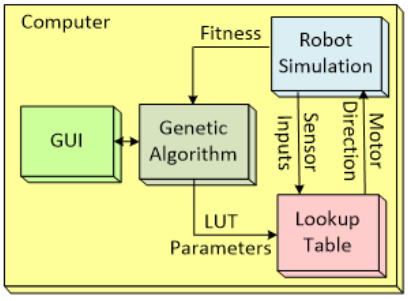

Figure 20. The systems used to evolve the LUT controller.

The LUT mapped the quantized input sensors of the robot to the axis of the array; the parameter at the indexed point was used as an output to drive the robot's actuators. In this case, the parameter contained the three motor speeds for each wheel, including forward (F), stopped (S) and reverse (R), giving eight robot directions (Table 2). The parameter SS (robot stopped) was not used, as it had no purpose in this application.

Table 2. The eight robot directions encoded into the LUT.

Motor	Direction
FF	Forward
FS	Forward Right
SF	Forward Left
RF	Pivot Left
FR	Pivot Right
SR	Reverse Left
RS	Reverse Right
RR	Reverse

5.3.2. Light following chromosome (LUT)

A two-dimensional table was used with the light levels from the left and right sensor quantized to fit the LUT indices (Figure 21). Ranges of LUTs were assessed ranging from 2x2 to 9x9 enabling the effects of quantization on the input sensors to be evaluated. It was found that the 6x6 had the optimum evolutionary efficiency versus controller performance reaching a 95% fitness within 36 generations. However, a 9x9 array was chosen to test scalability issues of the controllers.

	0	1	2	3	4	5	6	7	8
0	RS	SR	FS	RR	RR	FS	FF	RF	RS
1	SS	SR	SR	SR	FS	FF	FR	RF	FS
2	SR	SS	SR	RF	SR	RF	FR	FR	SR
3	FR	FF	SR	SR	FF	FR	RS	RF	FS
4	RF	FF	RR	RF	RR	SR	RR	SF	SF
5	RF	FS	FR	FS	RR	SF	FS	RS	SR
6	RR	RR	SF	SF	SF	FF	FS	SF	FF
7	FR	RF	RR	FS	FF	RR	RR	FR	RS
8	FR	FR	FF	SF	RS	FS	RS	RS	FF

Figure 21. The two-dimensional LUT for light following.

5.3.3. Obstacle avoidance chromosome (LUT)

Each of the six obstacle sensors produced a binary output dependent on whether an object was detected or not. The binary output of each of the six sensors were combined to form a six-bit number

ranging from 0 to 63. This value was used to reference the parameters inside a one-dimensional LUT (Figure 22).

Obstacle Sensor Word

0	1	2	3	...	...	61	62	63
R:S	S:R	F:S	S:F	...	...	S:F	R:F	F:F

Figure 22. The one-dimensional LUT for obstacle avoidance.

5.3.4. Combined behaviours chromosome (LUT)

Both the quantized outputs of the light sensor, and the digital word of the obstacle avoidance sensors were used to link to the axis of a three-dimensional LUT with the left and right sensors linked to the X and Y axis. The proximity sensors were linked to the Z-axis (Figure 23).

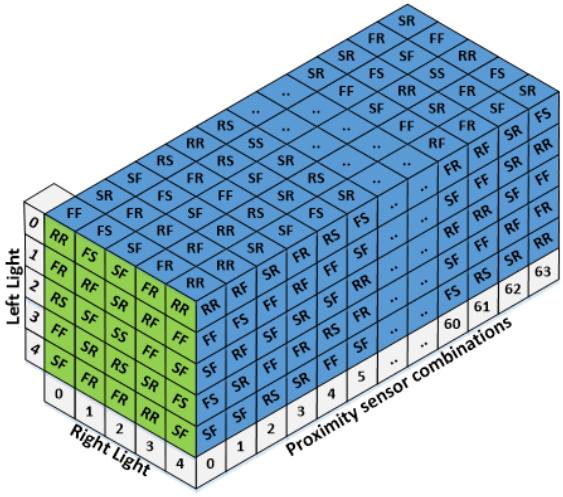

Figure 23. The three-dimensional LUT for combined behaviours.

5.4. Search space

The search space (SS) for the LUT is the number of possible robot directions (eight) to the power of the number of elements in the LUT (5).

$$SS = number\ of\ speeds^{number\ \ of\ table\ \ positions} \tag{5}$$

The search space for the ANN is number of possible steps to the power of the total combination of weights, bias points and activation set point (6). All the parameters in the ANN have been set to range from -1 to 1 giving a total number of 21 steps. The activation set point is mirrored between the plus and minus range giving one activation point per neuron. There is one bias point per neuron.

$$SS = (\ steps)^{(weig\ hts + bias + activation\ \ \ points\ )} \tag{6}$$

The search space for the EHW is two to the power of the number of bits in the configuration bit stream (7).

$$SS = 2^{number\ \ of\ config\ neration\ \ bits} \tag{7}$$

The search space for each controller and evolutionary efficiency is listed in the results section.

6. Results

Extensive evaluations of the three evolved controllers were implemented. A summary of these results is presented in this paper with typical results displayed. Quantizing each light source into nine levels was used in all controllers.

6.1. Light following

It was found that the controller performance over all three controllers was similar. This was determined by their final fitness and observation of their pathways (Figure 24). Investigation of the evolutionary stages show the robot evolving to pivot and progress towards the light, moving more directly as the fitness improved.

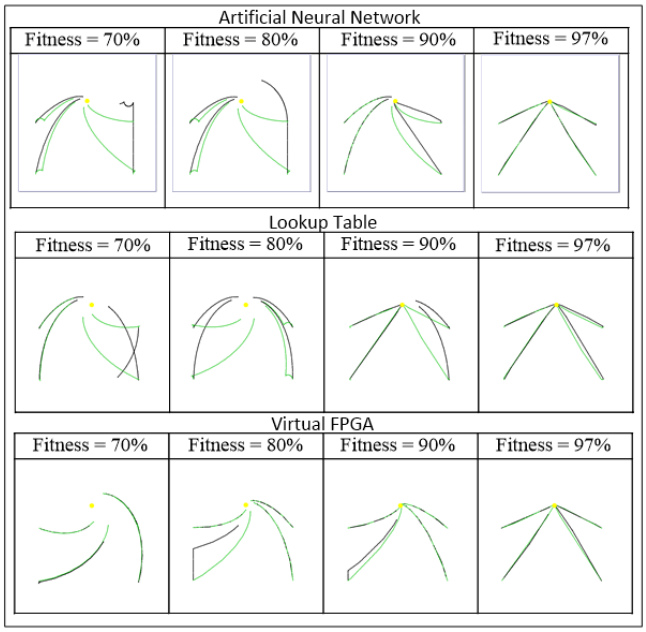

Figure 24. Robot paths at various stages of light following evolution for the ANN, LUT and EHW.

The evolutionary efficiency of each controller was measured in the number of generations it took to evolve to a fitness greater than 95%. A graph of the generation's vs fitness (Figure 25), demonstrate that the ANN and EHW had a similar evolutionary efficiency; however, the LUT took significantly longer to reach the required fitness. Note the starting average fitness was below 10% for all the controllers.

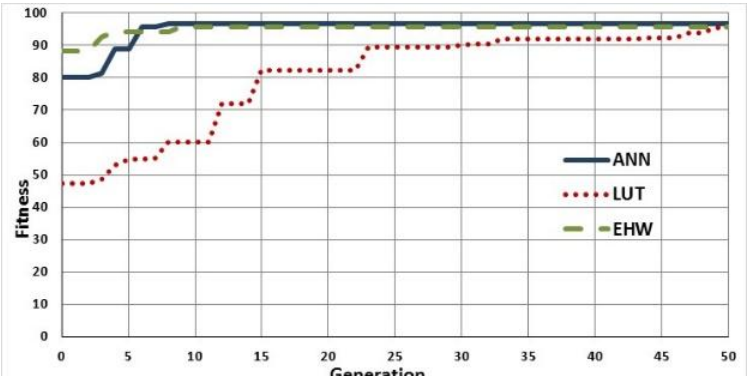

Figure 25. Light following maximum fitness vs generation for ANN, LUT and EHW.

The search space is all the possible combinations that the chromosome contains, (these have been calculated using the Eqs (5–7) shown above). Whereas the fitness landscape is a graphical representation of the individual fitness of each combination of the chromosome in the search space. The search space for each controller was compared with the evolutionary efficiency (Table 3). The variations in search space are due to the variations in chromosome length for each controller which is linked to the sensor quantization which has been matched across controllers. The EHW had an equivalent evolutionary efficiency to the ANN even though its search space was far larger. The LUT had an acceptable result despite its high search space.

Table 3. A comparison of the evolutionary efficiency and search space for light following.

Controller	Generation	Search space
ANN	6	3.8×10^{10}
LUT	49	1.4×10^{81}
EHW	9	8.8×10^{130}

To understand why the LUT had a reasonable performance despite its large search space, a typical cell usage was recorded (Figure 26). The LUT covers combinations of the two sensors that will never occur, such as one sensor at zero and the other at full brightness, giving the 9x9 LUT only 31 valid cells (coloured areas). Recalculating the search space for these cells brings the search space down to $9.9\text{x}10^{37}$. The percentage of time spent in each cell is recorded on the left chart and the motor direction occurrences on the right. The data shows the robot pivoting and moving towards the light with a slight right offset, which is corrected by the occasional left turn.

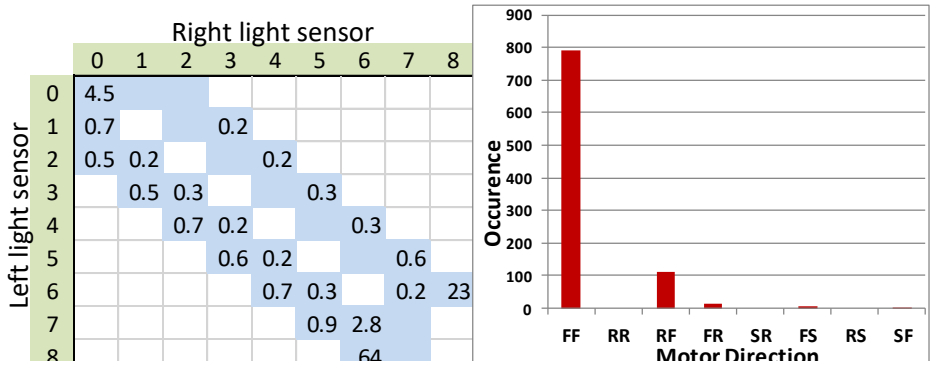

Figure 26. The 9x9 LUT showing the valid (coloured) and invalid (light) cells and the usage of the motor drive parameters.

The EHW evolutionary efficiency was equal to that of the ANN even though its search space was extremely high. An analysis of the evolved circuits showed that the virtual-FPGA was able to produce a multitude of different circuits that performed the same task. This included a range of selections of the multiplexers and logic function. Effectively there are many possible solutions scattered throughout the search space. Three examples of the evolved controller's Boolean logic are shown. There are twenty possible inputs, ten from the left sensor and ten from the right (refer section 4.2.3). The inputs to the Boolean logic are described with the prefixes LS and RS equating to left and right sensors, while the suffix L(0-8) equating to the light level. Thus, LSL0 refers to a logic input from the left sensor light 0, and RSL9 refers to a logic input from the right sensor with a light sensor level of 9. The output to the motor is three bits (Motor[0] to Motor[2]) thus each solution has three Boolean expressions to describe each bit of the motor outputs (Figure 27 and Table 4).

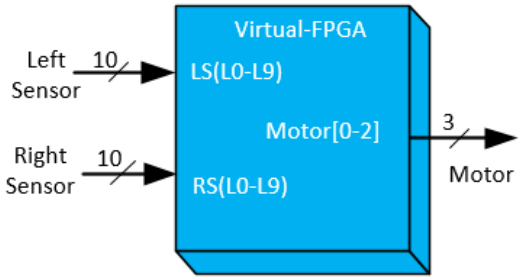

Figure 27. Boolean logic inputs and outputs used in the light following Virtual-FPGA.

Table 4. Light following evolved Boolean expressions.

Circuit One
Motor[0] = (!LSL8 & !LSL9 & LSL7) + (RSL9 & !LSL9 & LSL7) + (!LSL8 & !RSL7 & LSL7) + (RSL9 & !RSL7 & LSL7) Motor[1] = LSL7 + (RSL9 & RSL5) + (!RSL9 & !RSL5) Motor[2] = (!LSL9 & LSL7) + (!RSL7 & LSL7)
Circuit Two
Motor[0] = RSL6 & LSL9 Motor[1] = !RSL4 + !RSL6; Motor[2] = RSL6 & LSL5;
Circuit Three
Motor[0] = !RSL6 & !LSL8 & RSL1 & RSL4; Motor[1] = !RSL6; Motor[2] = RSL4 & RSL1 & !RSL6;

6.2. Obstacle avoidance

The controller performance for all three controllers were similar, with all three able to obtain a fitness greater than 80%. Observation of the paths showed the robot evolving exploration and avoidance behaviours (Figure 28).

The evolutionary efficiency of each controller was measured in the number of generations it took to evolve to a fitness greater than 80%. The stages of the evolutionary process of the controllers in the three arenas showed that the ANN and EHW were similar in performance, whereas the LUT took a significantly longer time to evolve (Figure 29 to Figure 31). Note the starting average fitness of all controllers is below 10%.

The search space for each controller was compared with the evolutionary efficiency (Table 5). The EHW performed well compared to the ANN, even though it had the largest search space. The LUT had the worst result of the three, corresponding to its larger search space.

The EHW for obstacle avoidance required a smaller configuration bit stream, and thus a reduced search space than the one required for light following. However, its search space was still much larger than that of the ANN, but still performed with a similar evolutionary efficiency. Once again, the evolved circuits were analyzed, and it was found that multiple circuits could be evolved to perform the same task, effectively scattering the search space with multiple solutions. The Boolean inputs are named after the position of each obstacle sensor, while the three outputs are combined to

create the 3-bit motor logic Motor[0]-[2] (Figure 32 and Table 6).

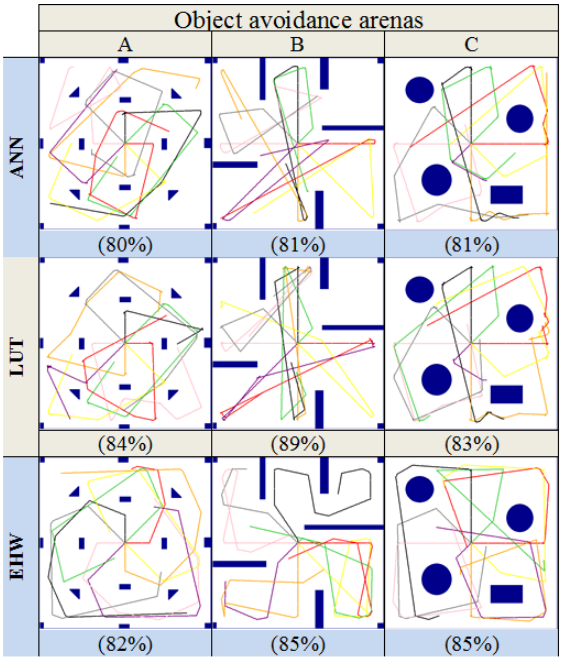

Figure 28. The final evolved paths taken by the robot using three different arenas for the ANN, LUT and EHW.

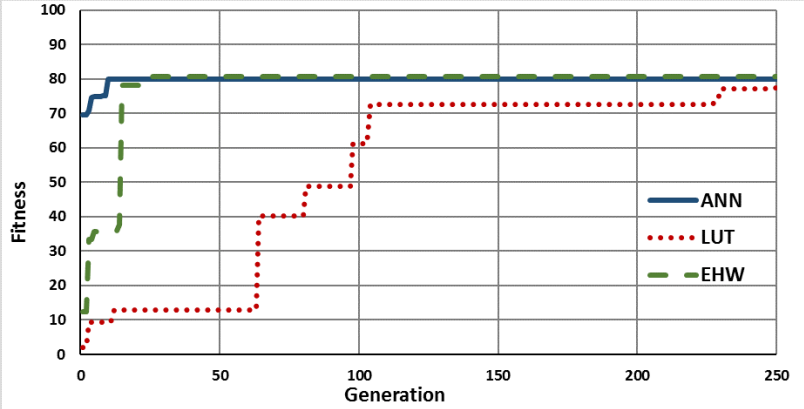

Figure 29. Obstacle avoidance arena-A maximum fitness vs generation for ANN, LUT and EHW.

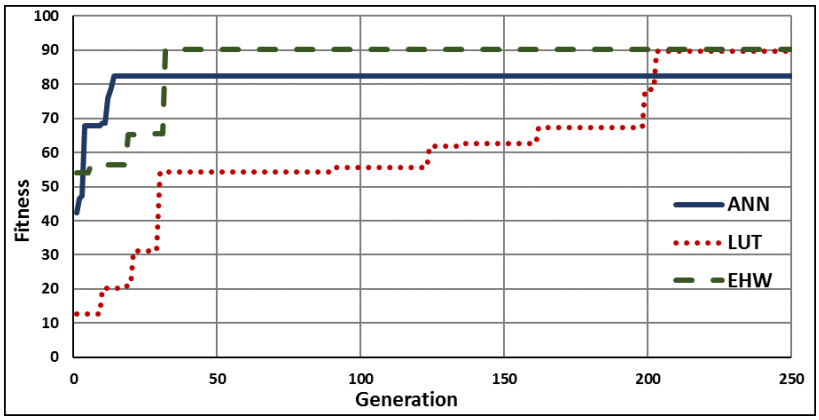

Figure 30. Obstacle avoidance arena-B maximum fitness vs generation for ANN, LUT and EHW.

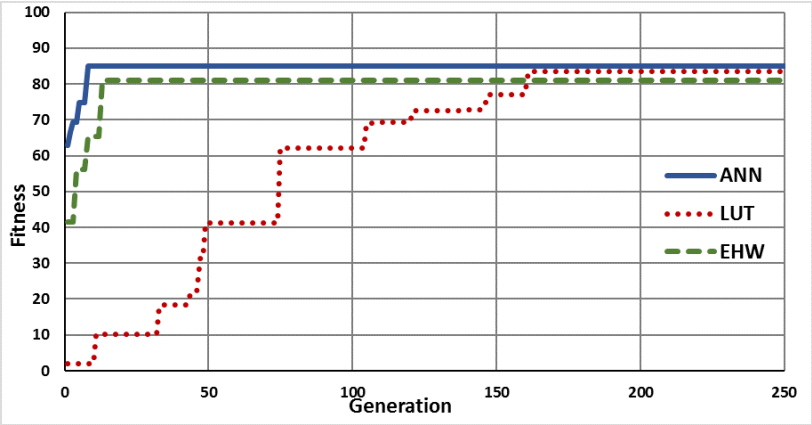

Figure 31. Obstacle avoidance arena-C maximum fitness vs generation for ANN, LUT and EHW.

Table 5. A comparison of the evolutionary efficiency and search space for obstacle avoidance.

Controller	Arena	Generation	Search space
ANN	A	10	$1.4x10^{21}$
	B	14	
	C	8	
LUT	A	262	$6.2x10^{57}$
	B	208	
	C	161	
EHW	A	25	$5.78x10^{76}$
	B	29	
	C	13	

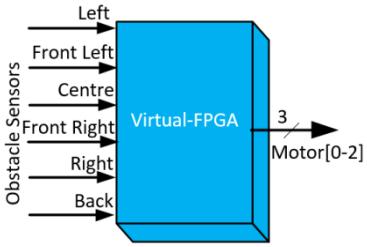

Figure 32. Boolean logic parameters used in the obstacle avoidance Virtual-FPGA.

Table 6. Obstacle avoidance evolved Boolean expressions.

Circuit One
Motor[0] = 0 Motor[1] = Front-Right + Centre + Right + Front-Left + Left Motor[2] = 0
Circuit Two
Motor[0] = 0 Motor[1] = Centre + Right-Centre + Left-Centre + Right Motor[2] = 0
Circuit Three
Motor[0] = Back Motor[1] = Back + Right + Centre + Front-Left + Front-Right Motor[2] = Back + Right

6.3. Combined light following and obstacle avoidance

The controller performance and evolutionary efficiency of each controller is compared for both monolithic and subsumption evolution methods. For monolithic evolution, the GA process was stopped when the fitness was greater than 80%, whereas for subsumption evolution, the independently evolved controller where combined and one fitness test run.

The controller performance for the three controllers for monolithic evolution was comparable with all controllers obtaining a fitness greater than 80% and observation of the paths showing the correct use of the combined behaviours (Figure 33). The controller performance using subsumption evolution showed the ANN and EHW had similar performance. The LUT performed well, although it did not move to the light as directly as the other controllers did.

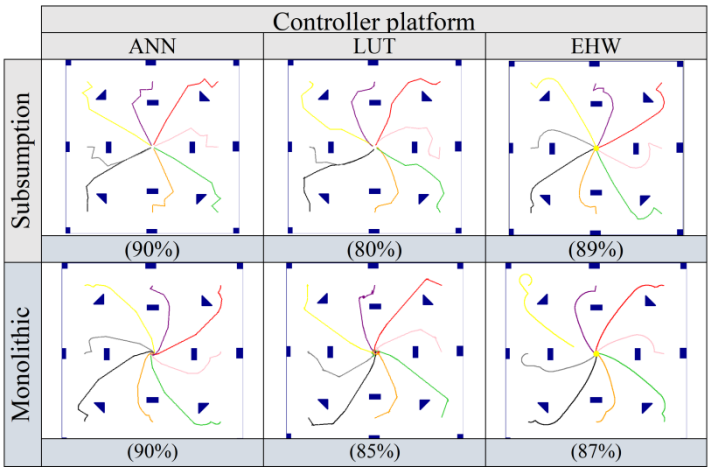

Figure 33. The pathways and fitness for the combined behaviours for the ANN, LUT and EHW comparing monolithic vs subsumption architecture.

The evolutionary efficiency showed that the ANN and EHW had similar behaviours, while the LUT performed poorly in comparison (Figure 34).

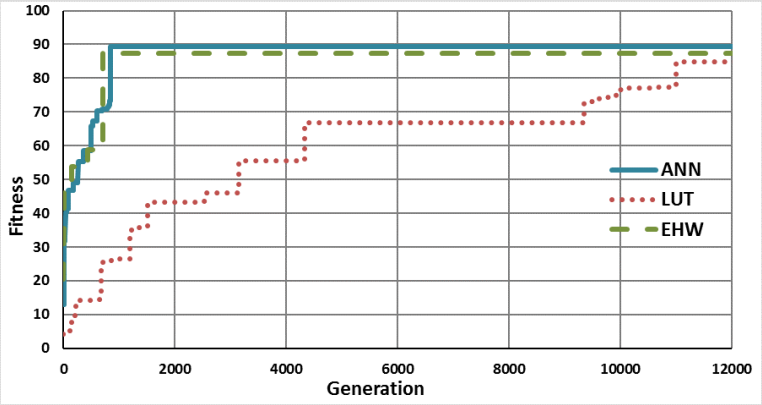

Figure 34. Monolithic light following while avoiding obstacles maximum and average fitness vs generation for ANN, LUT and EHW.

The search space and evolutionary efficiency was compared for the combined behaviours (Table 7). Once again, the EHW had a comparable efficiency to the ANN with the LUT performing poorly. Subsumption has a higher evolutionary efficiency than the monolithic method.

Table 7. A comparison of the evolutionary efficiency and search space for light following white avoiding obstacles.

Controller	Process	Generation	Search Space
ANN	monolithic	850	$2.8x10^{26}$
	light following	6	$3.8x10^{10}$
	object avoidance	10	$1.4x10^{21}$
LUT	monolithic	11,000	$1.0x10^{3699}$
	light following	49	$1.4x10^{81}$
	object avoidance	262	$6.2x10^{57}$
EHW	monolithic	700	$1.4x10^{138}$
	light following	9	$8.8x10^{130}$
	object avoidance	25	$5.78x10^{76}$

The EHW and ANN had a similar number of generations to evolve a suitable solution. The LUT does not scale well, due to its large search space, however a good solution can still be found within a reasonable time. This is due to most of the search space not being used. Only one slice of the LUT is used for light following when no obstacle is detected. When an obstacle is detected, it only needs to adopt a turning motion in the other LUT segments, until the obstacle is avoided and then switch back to the light following. Examples of the evolved EHW controller once again show multiple circuits. The Boolean input and output parameter names are a combination of those used in the light following and obstacle avoidance combining the obstacle avoidance sensors with the light sensors (Figure 35 and Table 8).

Table 8. Combined behaviours evolved Boolean expressions.

Circuit One
Motor[0] = RSL1 + (Back & LSL5) + (Back & !Left) + (LSL5 & Centre) + (!Left & Centre) + (Back & !LSL9) + (Back & RSL2) + (LSL5 & !Centre & RSL6 & LSL9) + (!Left & !Right-Centre & RSL6 & LSL9) + (!Centre & RSL6 & LSL9 & RSL2) Motor[1] = (Right-Centre & LSL5) + (!RSL6 & LSL5) + (Right-Centre & !Left) + (!RSL6 & !Left) + (LSL5 & Right-Centre + (!Left & Centre Motor[2] = Back + RSL1 + (LSL5 & Centre) + (!Left & Centre) + (!Right-Centre & RSL6 & LSL9)
Circuit Two
Motor[0] = !Back + (!RSL6 & !Left) + (!LSL3 & !RSL1) + (RSL6 & RSL4 & Left-Centre) + (!RSL6 & !RSL4 + !Left-Centre); Motor[1] = !Back + (!RSL6 & !Left + (!LSL3 & !RSL1) + (RSL6 & RSL4 & !Left-Centre) + (!RSL6 + !RSL4 +(!Left-Centre) Motor[2] = !Left-Centre & !RSL6 & !Left
Circuit Three
Motor[0] = !LSL5 + !RSL7 + RSL0 Motor[1] = Left + !RSL5 + (RSL2 & !Left-Centre & LSL7) + (RSL2 & Left-Centre & !LSL7) Motor[2] = (!Left & RSL5) + (!Left & Left-Centre) & LSL7) + (!Left & !Left-Centre & !LSL7)

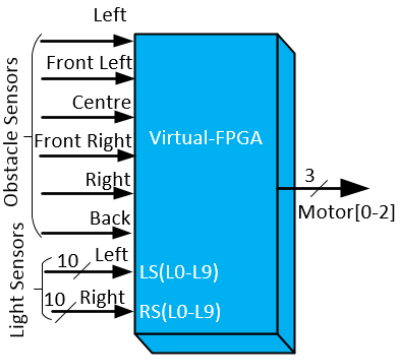

Figure 35. Boolean logic parameters used in the combined behaviours.

6.4. A comparison of results with other researchers

This section compares the results presented in this paper with other researchers evolving controllers for light following or obstacle avoidance in two wheeled robots.

Á. Pintér-Bartha et al. [4] used an ANN to evolve a robot to move towards a light source. A three layer fully meshed recurrent neural network with two hidden nodes was used. The selection method was mixed elitist and roulette wheel with a population size of 50 individuals. The reproduction method was uniform crossover with a mutation rate of 10%. The fitness function was a combination the time taken and the distance from the light. It was found that a satisfactory solution could be found in 200 generations. The single layer ANN used in this manuscript compares favourably against this controller, taking approximately 10 generations to obtain a suitable fitness. This can be accounted for by the differences in the neural network and the number of layers used.

M. Okura et al. [6] evolved a Xilinx XC6216 FPGA to control the motion of a Khepera robot to avoid obstacles. The robot has four front facing sensors that are converted into a binary number. The fitness was calculated from the distance travelled the number of obstacles avoided and the number of changes in direction of the motor. The population size was 10, and the GA was able to achieve a reasonable performance after 20 generations the robot ran for five seconds. The EHW used in this paper compares favourable against this as a controller was evolved in 20 seconds which ran or 20 seconds in a more complex arena.

K. C. Tan et al. [7] evolved a Xilinx XC6216 FPGA to control the motion of a Khepera robot to follow a light. The population size was 121 with a combination of elitist and roulette wheel used for selection. The reproduction method was multi-cut crossover with a mutation rate of one percent. For light following a satisfactory outcome was achieved in 200 generations. The EHW used in this paper once again compares favourable against this.

7. Conclusions

It has been shown that a EWH compares well in comparison with an ANN providing excellent controller performance and good evolutionary efficiency. The scalability of EHW as the complexity of the task increases is mitigated by two factors: 1) the virtual-FPGA's ability to perform multiple tasks without a large increase in hardware; and 2) the architecture having multiple solutions, enabling the GA to find solutions in the search space quickly. A more robust controller is created by evolving the controller in multiple environments. Finally, the resolution of the quantized signal affected only

the top end of the controller performance.

A successful controller for light following can be evolved in less than ten generations. A successful controller for obstacle avoidance can be evolved in less than 10 to 30 generations for the three arenas. The EWH controller would be an excellent choice when used with a hybrid FPGA that incorporates a hardwired ARM CPU internal to the device, such as the Altera Cyclone V series. This could be incorporated into a fault tolerant adaptable controller, with the internal ARM processor running a GA process in the background and updating the virtual-FPGA when required.

The ANN is shown to be a robust controller that is suitable for evolutionary robotics. It does not suffer the scalability issues of the LUT and is well suited for 32 to 64-bit processors.

The LUT can evolve to a good control performance but has issues with scalability, although these are lessened by the reduction in the search space due to large parts of the LUT not being utilized. The evolved controller would be useful in the commonly used 8-bit microcontroller, which has limited processing power, making it unsuitable for the floating-point operations an ANN requires.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. Omara FA and Arafa MM (2009) Genetic Algorithms for Task Scheduling Problem. *J Parallel Distr Com* 70: 13–22.
2. Brooks R (1986) A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation* 2: 14–23.
3. Brooks RA (1989) A robot that walks; emergent behaviors from a carefully evolved network. *Neural Comput* 1: 253–262.
4. Pintér-Bartha A, Sobe A, Elmenreich W (2012) Towards the light — Comparing evolved neural network controllers and Finite State Machine controllers. *10th International Workshop on Intelligent Solutions in Embedded Systems*, 83–87.
5. Braitenberg V (1984) Vehicles: Experiments in synthetic psychology. Cambridge, MIT Press.
6. Okura M, Matsumoto A, Ikeda H, et al. (2003) Artifical evolution of FPGA that controls a Miniature Mobile Robot Khepera. *SICE Annual Conference* 3: 2858–2863.
7. Tan KC, Chew CM, Tan KK, et al. (2002) Autonomous robot navigation via intrinsic evolution. *Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02* 2: 1272–1277.
8. Tyrrell AM, Krohling RA, Zhou Y (2004) Evolutionary algorithm for the promotion of evolvable hardware. *IEE Proceedings-Computers and Digital Techniques* 151: 267–275.
9. Krohling RYZA, Zhou Y, Tyrrell A (2002) Evolving FPGA-based robot controllers using an evolutionary algorithm. *First International Conference on Artificial Immune Systems*.
10. Seok H, Lee K, Joung J, et al. (2000) An On-Line Learning Method for Object-Locating Robots using Genetic Programming on Evolvable Hardware. *International Symposium on Artificial Life and Robotics* 1: 321–324.
11. Rui Y, Yanmei S, Kun H, et al. (2015) Online evolution of image filters based on dynamic partial reconfiguration of FPGA. *2015 11th International Conference on Natural Computation (ICNC)*, 999–1005.

12. Dobai R and Sekanina L (2013) Image filter evolution on the Xilinx Zynq Platform. *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, 164–171.
13. Dobai R and Sekanina L (2013) Towards evolvable systems based on the Xilinx Zynq platform. *2013 IEEE International Conference on Evolvable Systems (ICES)*, 89–95.
14. Srivastava AK, Gupta A, Chaturvedi S, et al. (2014) Design and simulation of virtual reconfigurable circuit for a Fault Tolerant System. *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, 1–4.
15. Kumar PN, Anandhi S, Perinbam JRP (2007) Evolving virtual reconfigurable circuit for a fault tolerant system. *2007 IEEE Congress on Evolutionary Computation*, 1555–1561.
16. Glette K, Torresen J, Hovin M (2009) Intermediate Level FPGA Reconfiguration for an Online EHW Pattern Recognition System. *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, 19–26.
17. Glette K and Kaufmann P (2014) Lookup table partial reconfiguration for an evolvable hardware classifier system. *IEEE Congress on Evolutionary Computation (CEC)*, 1706–1713.
18. Beckerleg M and Collins J (2008) Evolving Electronic Circuits for Robotic Control. *15th International Conference on Mechatronics and Machine Vision in Practice*, 651–656.
19. Beckerleg M and Collins J (2011) Using a Hardware Simulation within a Genetic Algorithm to evolve Robotic Controllers. *International Conference on Intelligent Automation and Robotics (ICIAR'11)*, San Francisco.
20. Abhishek V, Mukerjee A, Karnick H (2003) Artificial ontogenesis of controllers for robotic behavior using VLG GA. *IEEE International Conference on Systems, Man and Cybernetics* 4: 3376–3383.
21. Harter D (2005) Evolving neurodynamic controllers for autonomous robots. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks* 1: 137–142.
22. Elmenreich W and Klingler G (2007) Genetic Evolution of a Neural Network for the Autonomous Control of a Four-Wheeled Robot. *Sixth Mexican International Conference on Artificial Intelligence - Special Session (MICAI)*, 396–406.
23. Wahab W (2009) Autonomous mobile robot navigation using a dual artificial neural network. *TENCON 2009 - 2009 IEEE Region 10 Conference*, 1–6.
24. Mohanty PK, Parhi DR, Jha AK, et al. (2013) Path planning of an autonomous mobile robot using adaptive network based fuzzy controller. *2013 IEEE 3rd International Advance Computing Conference (IACC)*, 651–656.
25. Silva F, Correia L, Christensen AL (2017) Evolutionary online behaviour learning and adaptation in real robots. *Royal Society Open Science* 4: 160938.
26. Gigliotta O (2018) Equal but different: Task allocation in homogeneous communicating robots. *Neurocomputing* 272: 3–9.
27. Harandi FA, Derhami V, Jamshidi F (2019) A new feature selection method based on task environments for controlling robots. *Appl Soft Comput* 85: 105812.
28. Capi G and Doya K (2005) Evolution of recurrent neural controllers using an extended parallel genetic algorithm. *Robot Auton Syst* 52: 148–159.
29. Muñoz DM, Llanos CH, Coelho LDS, et al. (2014) Hardware opposition-based PSO applied to mobile robot controllers. *Eng Appl Artif Intel* 28: 64–77.
30. Bruno DR, Marranghello N, Osório FS, et al. (2018) Neurogenetic algorithm applied to Route Planning for Autonomous Mobile Robots. *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

31. Savage J, Cruz J, Matamoros M, et al. (2016) Configurable Mobile Robot Behaviors Implemented on FPGA Based Architectures. *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 317–322.
32. Coffey B (2011) Using building simulation and optimisatoin to calculate control lookup tables offline. *12th Conference of International Building Performance Simulation Association*.
33. Sobhan PVS, Kumar GVN, Priya MR, et al. (2009) Look Up Table Based Fuzzy Logic Controller for Unmanned Autonomous Underwater Vehicle. *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 497–501.
34. Singh PK, Bhanot S, Mohanta H (2013) Neural Control of Neutralization Process using Fuzzy Inference System based Lookup Table. *International Journal of Computer Applications* 61.
35. Kim J, Kim YG, An J (2011) A Fuzzy Obstacle Avoidance Controller Using a Lookup-Table Sharing Method and Its Applications for Mobile Robots. *International Journal of Advanced Robotic Systems* 8: 62.
36. Beckerleg M and Collins J (2007) An Analysis of the Chromosome Generated by a Genetic Algorithm Used to Create a Controller for a Mobile Inverted Pendulum. *Autonomous Robots and Agents* 76.
37. Currie J, Beckerleg M, Collins J (2008) Software Evolution of a Hexapod Robot Walking Gait. *International journal of intelligent systems technologies and applications* 8: 382–394.
38. Beckerleg M and Collins J (2013) An Analysis of the Genetic Evolution of a Ball-Beam Robotic Controller Based on a Three Dimensional Look up Table Chromosome. *AENG Transactions on Engineering Technologies Lecture Notes in Electrical Engineering* 170: 109–122.
39. Beckerleg M and Hogg R (2016) Evolving a lookup table based motion controller for a ball-plate system with fault tolerant capabilities. *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 257–262.
40. Beckerleg M and Matulich J (2014) Evolving a lookup table based controller for robotic navigation. *2014 IEEE International Conference on Evolvable Systems (ICES)*, 195–202.
41. Beckerleg M and Zhang C (2016) Evolving Individual and Collective Behaviours for the Kilobot Robot. *IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 263–268.
42. Saito H, Amano R, Moriyama N, et al. (2013) Emergency obstacle avoidance module for mobile robots using a laser range finder. *SICE Annual Conference (SICE)*, 348–353.
43. Dasmane VS and Madki MR (2014) Implementation and analysis of real time obstacle avoiding subsumption controlled robot. *International Journal of Advanced Research in Computer and Communication Engineering* 3: 4.
44. Turner JT, Givigi SN, Beaulieu A (2013) Implementation of a subsumption based architecture using model-driven development. *IEEE International Systems Conference (SysCon)*, 331–338.
45. Cheng TT and Mahyuddin MN (2009) Implementation of behaviour-based mobile robot for obstacle avoidance using a single ultrasonic sensor. *Innovative Technologies in Intelligent Systems and Industrial Applications*, 244–248.
46. Martins G, Urbano P, Christensen AL (2018) Using Communication for the Evolution of Scalable Role Allocation in Collective Robotics. *Ibero-American Conference on Artificial Intelligence*, 326–337.