



A comparison of an evolvable hardware controller with an artificial neural network used for evolving the gait of a hexapod robot

Fraser Borrett¹ · Mark Beckerleg¹

Received: 27 September 2022 / Revised: 13 February 2023 / Accepted: 3 March 2023 /

Published online: 28 March 2023

© The Author(s) 2023

Abstract

This paper investigates the implementation of a novel evolvable hardware controller used in evolutionary robotics. The evolvable hardware consists of a Cartesian based array of logic blocks comprised of multiplexers and logic elements. The logic blocks are configured by a bit stream which is evolved using a genetic algorithm. A comparison is performed between an evolvable hardware and an artificial neural network controller evolved using the same genetic algorithm to produce the gait of a hexapod robot. To compare the two controllers, differences in their evolutionary efficiency and robot performance are investigated. The evolutionary efficiency is measured by the required number of generations to achieve an optimal fitness. An optimal hexapod controller allows the robot to walk forward in a straight line maintaining a constant heading and body attitude. It was found that the evolutionary efficiency and performance of the evolvable hardware and artificial neural network were similar, however the EHW was the most evolutionary efficient requiring less generations on average to evolve. Both evolved controllers were evaluated in simulation, and on a physical robot using a softcore processor and custom hardware implemented on a FPGA. The implementation showed that the controllers performed equally well when deployed, allowing the hexapod to meet the optimal gait criteria. These findings have shown that the evolvable hardware controller is a valid option for robotic control of a multilegged robot such as a hexapod as its evolutionary efficiency and deployed performance on a real robot is comparable to that of an artificial neural network. One future application of these evolvable controllers is in fault tolerance where the robot can dynamically adapt to a fault by evolving the controller to adjust to the fault conditions.

Keywords Evolutionary robots · Artificial neural network · Evolvable hardware · Hexapod robotic · Robot gait · Genetic algorithms

✉ Fraser Borrett
fraser.borrett@aut.ac.nz

Extended author information available on the last page of the article

1 Introduction

This paper investigates the implementation of an evolvable hardware (EHW) controller for evolutionary robotic applications. The EHW controller is a “virtual” field programmable gate array (FPGA) that is specifically designed for evolution incorporating the features of: (1) non-destructive architecture to prevent damage during the evolutionary process; (2) coarse-grained structure for an improved evolutionary rate; (3) partial reconfigurability to allow the EHW to change within the FPGA; and (4) scalability allowing successful evolution as the complexity of the problem increases. This EHW architecture is based on the Cartesian genetic programming EHW approach originally introduced by Miller et al. [1, 2]. The EHW controller is evolved to produce the walking gait of a hexapod robot. To study the abilities of the EHW controller a comparison is made between the EHW and an artificial neural network (ANN) controller in two areas: (1) the evolutionary efficiency, determined by how quickly the controllers can evolve to a satisfactory performance; and (2) the evolved controller performance, determined by both the maximum fitness achieved and observation of the gait in the physical robot.

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 the operation of the robot. 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 EHW GA was executed on the Terasic DE10-Nano board which incorporates a Cyclone V FPGA with an onboard hardcore processor (Dual-Core ARM Cortex-A9 MPCore Processor, 925 MHz). The implementation of an EHW controller for evolution (Fig. 1) shows the EHW and the System-on-Chip ARM processor residing in the FPGA. The Arm processor contains the programs for: (a) the genetic algorithm; (b) the robot simulation; and (c) a serial link for data logging of the evolutionary process on a PC.

The ANN GA was performed in MATLAB which ran the GA, robot simulation and the ANN (Fig. 2). A standard 3-layer feed-forward ANN was evolved.

The simulation model of the robot was based on a commercially bought robot chassis (Fig. 3). The electronics of this robot were specifically designed to compare both the evolved EHW and the evolved ANN. The electronics incorporated a RN4677-V Bluetooth module for remote control, and the Terasic DE0-Nano board to interface to the eighteen leg motors. Note the DE0-Nano board used in the robot is different from the DE10-Nano board used to evolve the EHW controller.

A future practical application of EHW is fault tolerant robotic controllers. FPGA devices can incorporate powerful, hardcore ARM processors, allowing

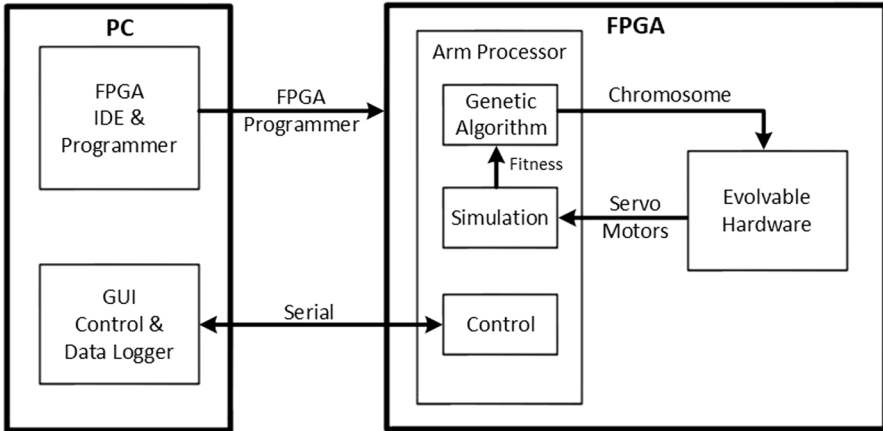


Fig. 1 Overview of the EHW system

Fig. 2 Overview of the ANN system

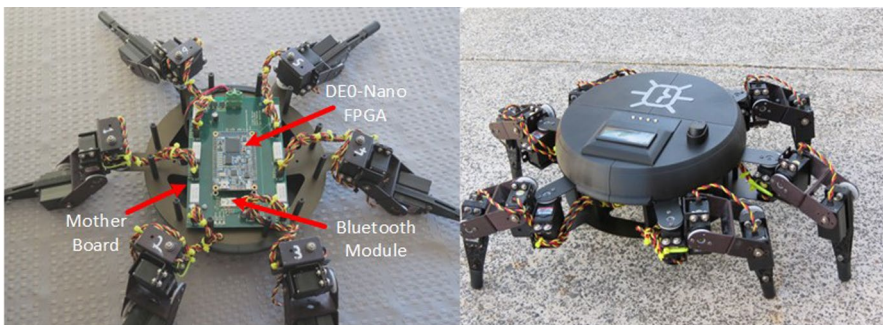
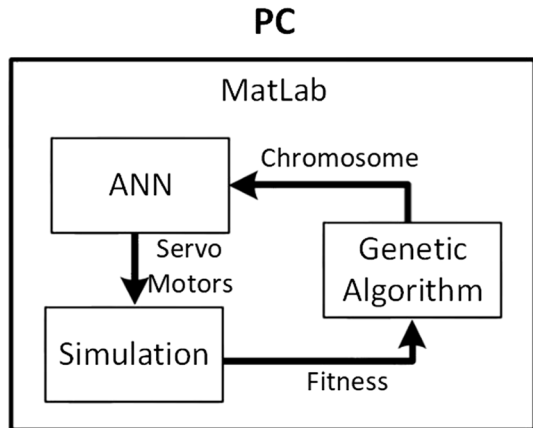


Fig. 3 The physical hexapod robot that was designed at the Auckland University of Technology for evaluation of EHW in real world applications (a) The hexapod controller board with DE0-Nano FPGA and Bluetooth Module (b) Hexapod walking

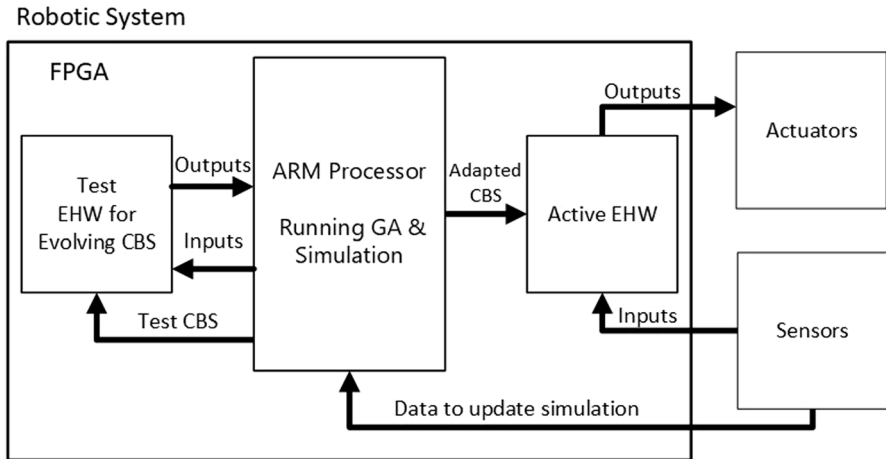


Fig. 4 EHW used in an adaptive fault tolerant system

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, to replace the active controller when a change in the real-world system occurs (Fig. 4).

2 Related work

A comparison of evolvable robotic controller types has been performed by a small number of authors. Pinter-Bartha et al. [3] compared an 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. Beckerleg et al [4]. investigated three robotic controllers, an EHW, an ANN and a lookup table (LUT). The three controllers were evolved for both light following and obstacle avoidance. It was found that the ANN and EHW had a similar evolutionary performance, however the LUT's evolutionary efficiency was much lower for both tasks.

2.1 Evolvable hardware robotic controllers

EHW uses evolutionary algorithms to evolve hardware. Primarily this is performed on FPGAs as their architecture allows the application of reconfigurable digital circuits designed using a hardware description language (HDL) such as Verilog. The FPGA is a two-dimensional array of logic array blocks (LABs) with interconnections between LABs. Normally an integrated development environment (IDE) such as Intel's "Quartus" is used to convert the digital circuit design into a configuration bit stream (CBS) which is then downloaded into the FPGA to configure the LABs

and interconnections. When used in EHW, the CBS can be seen as a chromosome and directly evolved using an evolutionary algorithm to evolve robotic controllers.

The major difficulties of evolving a CBS are: (a) the creation of destructive hardware where outputs are connected to outputs; (b) fine grained architecture where complex circuits are difficult to evolve; and (c) difficulties with scalability. Under normal conditions the IDE would not allow a destructive CBS to be produced, instead producing error messages to the designer. However, evolving the CBS directly allows destructive architectures to be produced. In earlier research the Xilinx XC6216 FPGA was used as its internal architecture did not allow output to output connections [5, 6].

However, this chip is no longer in production and modern FPGAs are capable of destructive architectures, relying on the IDE to prevent this from occurring. Alternative approaches to avoid destructive architectures include: (a) evolutionary algorithms that only manipulate the sections of the CBS corresponding to the LAB's function and not the interconnections [7, 8]; (b) the evolution of the hardware description language itself relying on the compiler to prevent destructive architectures [9]; and (c) virtual FPGAs (VFPGAs) designed in HDL and implemented on FPGAs.

The VFPGA (Fig. 5) consists of a two-dimensional array of LABs. Each LAB contains a multiplexer which selects inputs determining the interconnections between LABs, and logic elements (LEs) which are a LUT of logic functions. These LABs are configured by the CBS. The output of one column of LABs connects to the input of the following column of LABs, preventing destructive output to output connections allowing the hardware to be evolved safely. The LE is comprised of complex logic functions giving the VFPGA a course-grained architecture which is more suitable for evolution.

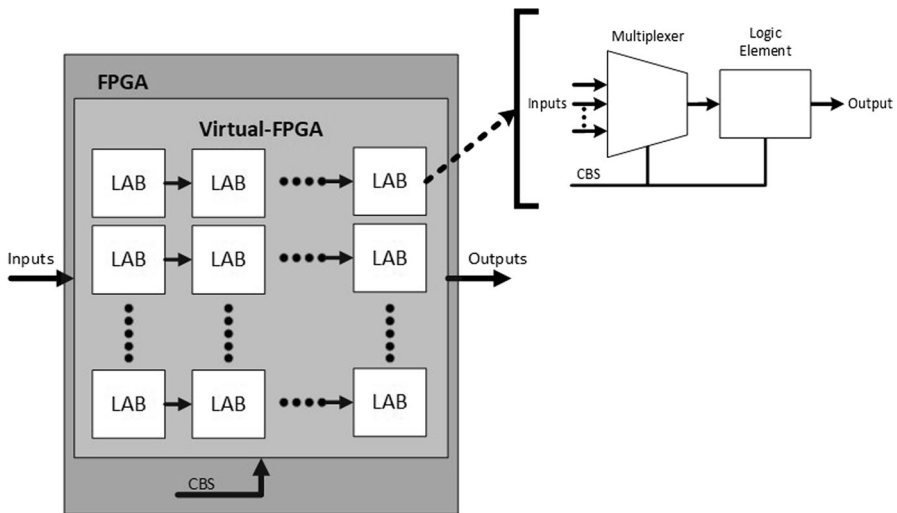


Fig. 5 Virtual-FPGA showing LABs placed in Cartesian array with internal multiplexer and logic elements configured by the CBS

VFPGA have been evolved for a variety of tasks including image processing using [10–12], fault tolerant systems [13, 14], pattern recognition [15–17] and character recognition [18]. Investigation into the use of VFPGAs as robotic controllers has also been conducted looking at a ball-beam system [19, 20] and for object detection/avoidance [4] as previously mentioned.

2.2 Evolvable artificial neural network robotic controllers

ANNs mimic biological neural networks using a layered architecture of interconnected neurons. In the synthetic software form, each input to a neuron has a weighting factor and the sum of these weighted inputs plus a bias is fed into an activation function such as the sigmoid or rectified linear functions. The activation function will determine the output of the neuron. These networks of neurons are then trained using such strategies as supervised, unsupervised or reinforcement learning strategies. A familiar example of ANNs can be seen in modern smart phones where it is used in facial recognition to unlock the phone [21, 22].

Alternatively, to the traditional training methods, evolutionary algorithms can be used to adapt the weights, bias, activation function and even the architecture of the network to develop an optimal ANN for a given task. ANNs have been successfully evolved for light following and obstacle avoidance robots [23–28], solving the inverse kinematics of robotic manipulators [29], hexapod locomotion control [30, 31], and fault recovery of a quadruped where the ANN was implemented in Field Programmable Analog Arrays (FPAA) and evolved using a GA [32].

2.3 Evolved robotic controllers for hexapod locomotion

Currie and Beckerleg [33] evolved a walking gait for the hexapod, where the chromosome was a lookup table containing the angles of the servos at each interval in the gait. The fitness was based on the stability and efficiency of the gait of the robot walking in a straight line. Li et al. [34] evolved an ANN to develop a walking gait for a hexapod robot with 2 degrees of freedom for each leg. The ANN was a six neuron fully recurrent neural network. The ANN was implemented for a single leg, using two inputs that fired when the extremes of the servo positions were reached. The ANN had two outputs to generate the mark value for the servo PWM signal. The fitness was a combination of the forward movement, the number of times the leg was raised, and the drag generated during the gait. It was found that an optimal gait could be generated in approximately 300 generations. Juang et al. [35] evolved fully connected recurrent neural networks (FCRNN) to produce the walking gait of a hexapod with 2DOF legs. The EA used was symbiotic species-based particle swarm optimization (SSPSO). The SSPSO algorithm was successfully used to evolve the controllers to produce a gait where the hexapod walked forward in a straight line. In the paper, SSPSO was compared to different PSO algorithms, and a standard GA. It was shown in the same number of evaluations, SSPSO evolved a better walking gait allowing the robot to walk further compared to the other evolutionary algorithms. Heijnen et al. [36] developed a testbed to evolve the feed forward controllers of a hexapod for a specific mission environment. The paper's

primary objective was to investigate the testbed and how it could be used to evolve the controllers in real time on the real robot. The evolution was done in a two-stage process: (1) using Nondominated Sorting Differential Evolution (NSDE) on a population of possible solution; and (2) selecting a single parent from the stage 1 population based on the mission criteria and using the 1+1-Evolution Strategy (ES) to reach the final solution. Heijnen was able to show the feed forward controllers that produce the desired foot positions could be evolved using the testbed for fitness requirements based on the smoothness, stability, and efficiency of the hexapod’s gait.

The authors have not found any references for evolving EWH for hexapod controllers.

3 Robot kinematics

The following inverse kinematic Eqs. (1–9) are developed from geometric analysis of the leg (Fig. 6). Where H, K, F and O are position vectors defining; the hip, knee, foot and leg origin respectively relative to the robot body. The robot dimensions are shown as the length of the pelvis l_{pelvis} , the length of the femur l_{femur} , and the length of the tibia l_{tibia} . The three angles γ, α and β are the joint angles for the servos positioned at O, H and K respectively.

$$H_x = O_x + l_{pelvis} * \cos(\gamma) \tag{1}$$

$$H_y = O_y + l_{pelvis} * \sin(\gamma) \tag{2}$$

$$H_z = O_z \tag{3}$$

$$K_x = H_x + l_{femur} * \cos(\alpha) * \cos(\gamma) \tag{4}$$

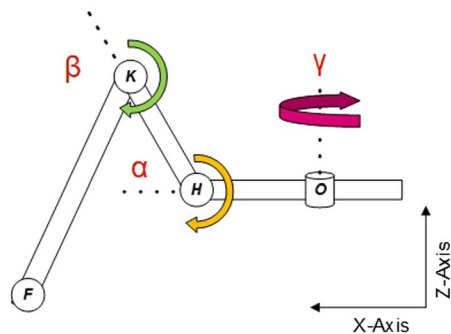
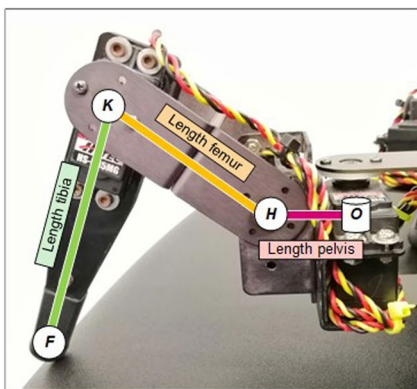


Fig. 6 Hexapod robot leg model

$$K_y = H_y + l_{femur} * \cos(\alpha) * \sin(\gamma) \quad (5)$$

$$K_z = H_z + l_{femur} * \sin(\alpha) \quad (6)$$

$$F_x = K_x + l_{tibia} * \cos(\beta) * \cos(\gamma) \quad (7)$$

$$F_y = K_y + l_{tibia} * \cos(\beta) * \sin(\gamma) \quad (8)$$

$$F_z = K_z + l_{tibia} * \sin(\beta) \quad (9)$$

For both controllers the angular position of the joints is stored in a matrix to be used to form a set of xyz data using the above inverse kinematic equations; the two data sets are used to determine the fitness of the individual's gait.

4 System structures and chromosomes

4.1 Evolvable hardware

The EHW units are digital circuits that are programmed into a FPGA using the Verilog hardware description language. The top-level view of the EHW for one leg (Fig. 7) has three EHW units, one for each servo. There are four inputs which are paralleled to each EHW unit, and the output from each unit is used to drive the leg's

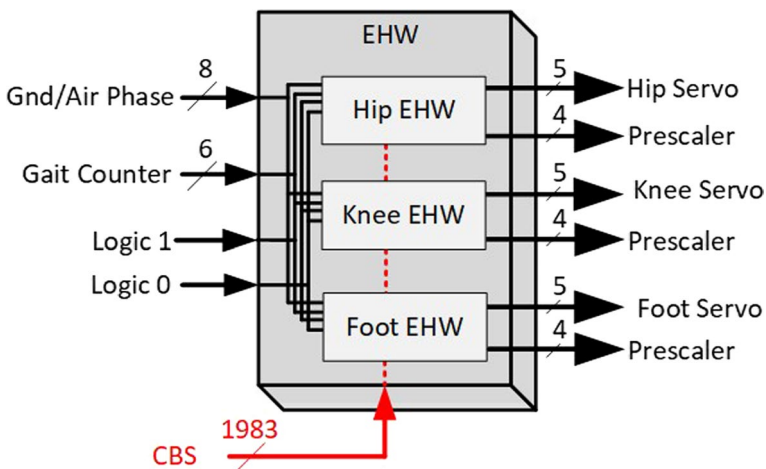


Fig. 7 The EHW structure for one leg showing the inputs and outputs

three servos: hip, knee, and foot. The inputs are broken into: (a) an 8-bit signal to show if the leg is on the ground (ground phase 8'b00001111) or in the air (air phase 8'b11110000); (b) a 6-bit gait counter showing the ten steps of each phase, with the numbers moving up in steps of 6, (6, 12 60); and (c) logic zero and one inputs. The two outputs from each EHW unit are: (a) a signed 5-bit value giving the angular change of the servo motor; and (b) a 4-bit prescaler that scales the angular change. The 4-bit prescaler allows the resolution of the change in servo angle to vary between -15° to $+15^\circ$ down to -0.9° to 0.9° . The inputs and outputs of the EHW are interfaced to a hardcore ARM processor using standard parallel input output ports (PIO). The ARM processor which is running the GA and robot simulation sends the appropriate ground/air phase of the robot and gait counter to the EHW, and then converts the outputs of the EHW into a PWM signal used to drive the simulated hexapod servo motors.

The EHW architecture (Fig. 8) is made up of five layers of interconnected LABs in a Cartesian based architecture, where the data is passed from left to right in a feedforward process until the final output is reached. Each LAB incorporates multiplexers to select inputs, and logic elements in the form of LUTs to provide logic functions. The inputs to the EHW are 16 bits of data made up of the air/ground phase signal, the step counter and logic 0 and 1, while the output is the change in servo motor angle and a prescaler. The EHW are configured using a CBS.

Layer 1 (Fig. 9) consists of a column of 8 LABs with each LAB containing 4 multiplexers and a logic element (LE) which can perform one of 32 selectable logic operations. Each multiplexer is configured by the CBS to select 1 of the 16 input bits to be processed by the LE. Each of the 4 multiplexer outputs are feed into the LE. The CBS fed into the LE determines which of the 32 logic operations will be used on the four inputs by means of a Verilog case statement (Fig. 10). For example, if the LE configuration bit stream is 5'b01000 the logic function for case 8 (A&B&C&D)

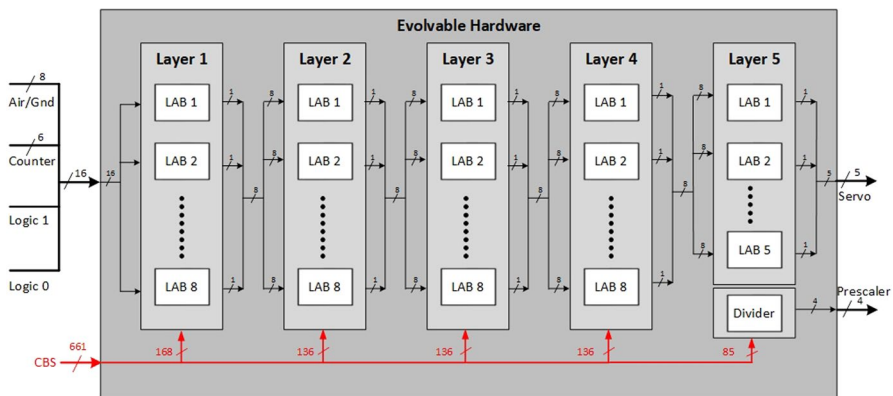


Fig. 8 One of the EHW units used to drive a servo motor, showing the five-layer Cartesian architecture

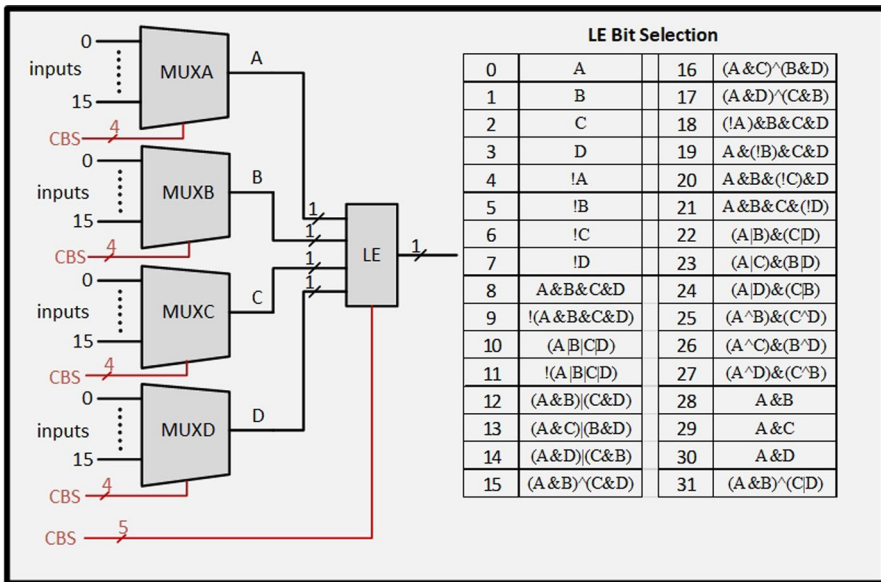


Fig. 9 LAB for layer 1 with 16 inputs, 1 output, and the configuration bit stream required for each component

```

case (config_bits_LUT)
    0:    d_out <= A;
    1:    d_out <= B;
    2:    d_out <= C;
    3:    d_out <= D;
    4:    d_out <= !A;
    5:    d_out <= !B;
    6:    d_out <= !C;
    7:    d_out <= !D;
    8:    d_out <= A&B&C&D;
    9:    d_out <= !(A&B&C&D);
    ....
    31:   d_out <= (A&B)^(C|D)

```

Fig. 10 Verilog case statement for selectable logic operations

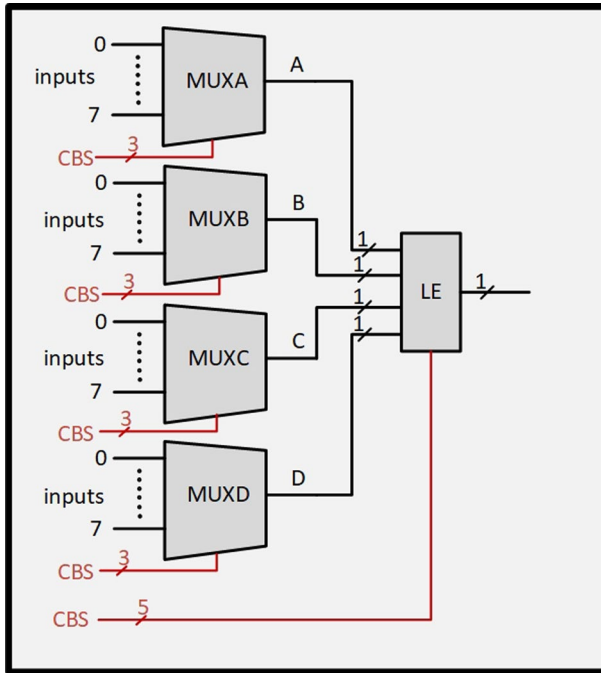


Fig. 11 LAB for layers 2–4 with 8 inputs and 1 output

is used. The outputs from each of the eight LABs are combined to produce a parallel 8-bit output which feeds into the following layer.

Layer 2–4 consists of 8 LABs similar to layer 1 apart from the multiplexers which selects from 8-bits as opposed to 16 (Fig. 11). The outputs from each LAB are combined to produce an 8-bit output for each of the layers 2–4.

Layer 5 is the output layer which has 5 LABs that are identical to those used in layers 2–4. The outputs from each of the 5 LABs are combined to produce a 5-bit signed output that is used to determine the servo motor angular change in degrees for a servo in the hexapod leg.

To calculate the angle change, the 5 bits are separated. Bit 4 the MSB is the sign of the angle change and bits 3-0 are the magnitude. The EHW also has a prescaler output which is used by the ARM processor to scale the magnitude of the servo motor angular change from 1/16th to 1, Eq. (10). This results in a floating-point value used to control the angle of the servo.

$$\Delta position = \frac{sgn(out[4]) * out[3 : 0]}{prescaler} \tag{10}$$

4.1.1 Chromosome

The chromosome for the EHW is its configuration bit stream. The chromosome is stored in the ARM processor as an array of bytes. The bytes are used to configure the virtual FPGA from the ARM processor through addressed PIO ports on the FPGA fabric using the AXI lightweight bridge, which is the interface between the hard processor and the FPGA fabric within the Cyclone V FPGA chip. Each bit of the PIO port is connected to a wire in the multiplexer or LUT within the LABs. The byte arrays are grouped into five arrays, layer 1 has 21 bytes, layer 2–4 has 17 bytes each, and layer 5 has 11 bytes (Fig. 12). Each bit does not define a specific phenotype but contributes to a section of the LAB that will determine a particular output as shown in (Fig. 13).

The genotype–phenotype mapping for the EHW chromosome is shown in Figs. 13 and 14. The genotype is represented by the bits that make up the configuration bit stream, the phenotype is the combined configured operation of the logic array blocks i.e. the selected interconnections between the LABs and the respective logic function the LABs will carry out. The combination of these configured LABs results in a controller that has the necessary characteristics to optimally drive the hexapod.

The Fig. 14 example is a spreadsheet that shows an actual evolved solution. The inputs are the ground/air phase, gait counter and logic I/O. The five layers are shown as columns, the LABs within the column are shown as rows. The first row of the

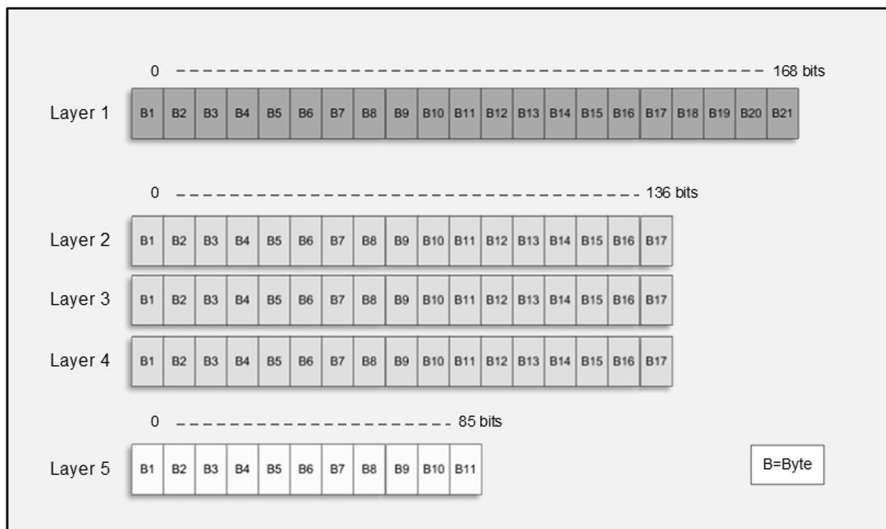


Fig. 12 The five arrays which represent the Chromosome of one EHW unit

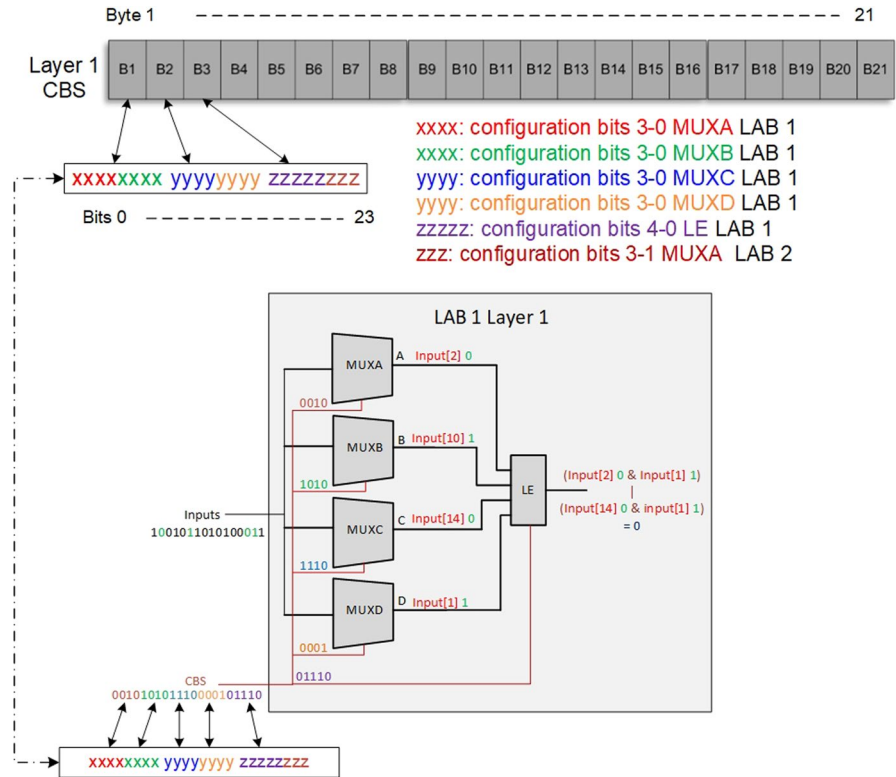


Fig. 13 Visualization of genotype/phenotype map of a LAB, showing the relationship between the individual configuration bits and the programmed operation of a LAB. Example with random input and configuration

LAB shows the multiplexer selection and logic function, the second row shows the CBS for the multiplexers A-D and the LE. The adjacent column shows the logic output of the LAB. The progression from genotype to phenotype is shown as the signal progresses through the EHW until the servo motor angle change is produced at the output.

The crossover and mutation of the EHW chromosome during the reproduction phase of the GA is as follows. The chromosome is divided into layers, with single point crossover performed between the two parents of that layer (Fig. 15). This crossover can be considered to be multipoint when applied to the complete chromosome. One of the bytes in the complete 83-byte chromosome is randomly selected and mutated, giving a mutation rate of approximately 1%. For the prescaler value, the child chromosome will get either parent one or parent two prescaler value at

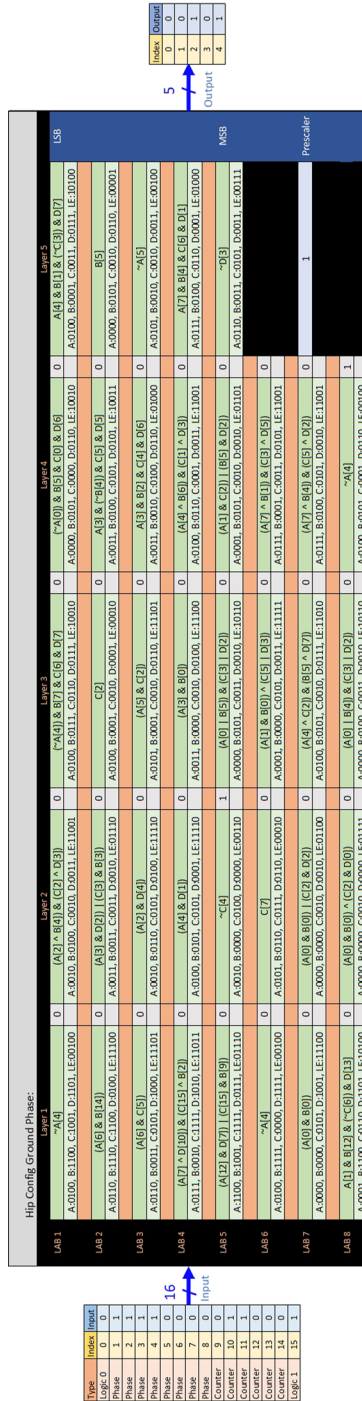


Fig. 14 Evolved configuration hip EHW unit for ground phase. Example input and output, showing how the data is manipulated passing between each layer

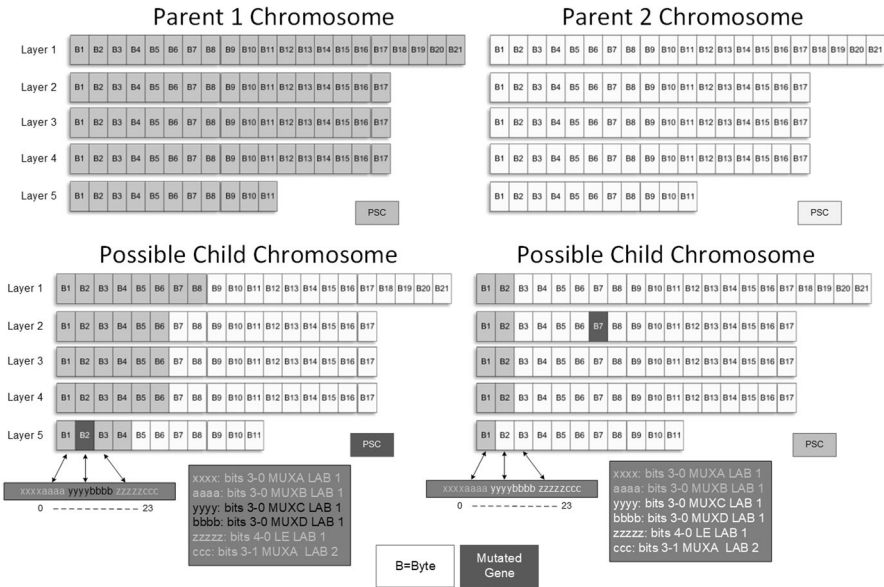


Fig. 15 Representation of the EHW chromosome crossover and mutation operation

an equal probability. The prescaler also has a mutation rate 1 percent. This crossover and mutation process is carried out separately for each of the 3 EHW units that make up an individual leg controller.

4.2 Artificial neural network

The ANN architecture (Fig. 16) is a three-layer feed forward neural network with an input layer, a single hidden layer, and an output layer. The inputs are: (a) air-ground phase (-4.5 when on the ground and $+4.5$ when in the air); and (b) the step count ranging from 1 to 10. The output is the servo angular change using an adapted hyperbolic tan activation function ranging from -15° to $+15^\circ$.

4.2.1 Hidden layer

The weights and bias for the hidden layer neurons (Fig. 17) ranged from -1 to 1 in steps of 0.1 giving an input range from -15.5 to $+15.5$. The activation function is an adapted hyperbolic tangent which is normalized to give an output ranging from -1 to $+1$. The hidden layer neuron is described by Eqs. (11 and 12); where W_i is the weight corresponding to input I_i from the input layer; and B is the bias of the neuron.

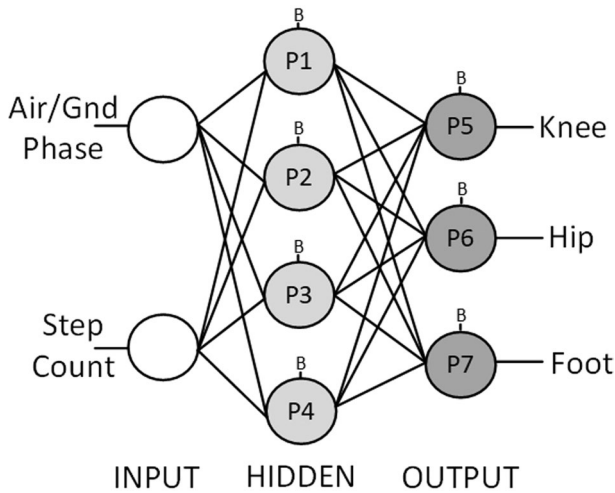


Fig. 16 Three-layer ANN used for each leg

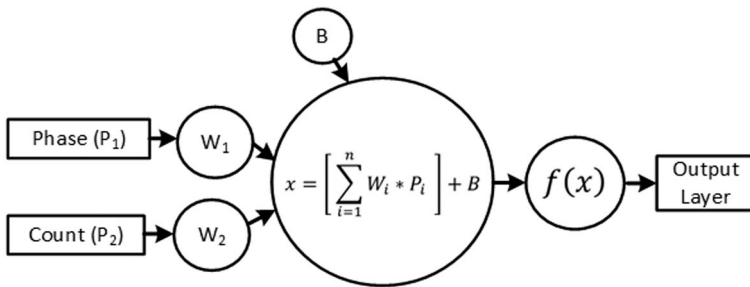


Fig. 17 Hidden layer neurons (P1–P4)

$$x = \left[\sum_{i=1}^n W_i * P_i \right] + B \tag{11}$$

$$f(x) = \frac{1 - e^{-\frac{x}{3}}}{1 + e^{-\frac{x}{3}}} \tag{12}$$

4.2.2 Output layer

The weights and bias for the output layer neurons (Fig. 18) ranged from -1 to 1 in steps of 0.1 giving an input range from -5 to $+5$. The activation function is an

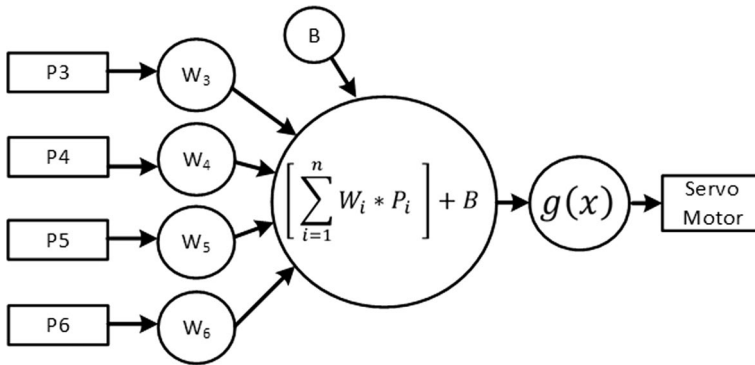


Fig. 18 Output layer neurons (P5–P7)

adapted hyperbolic tangent with a multiplication factor giving an output range from -15 to $+15$.

The neuron function is shown in Eq. (13); where W_i is the weight corresponding to input P_i from the hidden layer neurons. The activation function is shown in Eq. (14) The activation function used is a variation on the commonly used hyperbolic tangent function.

$$x = \left[\sum_{i=1}^n W_i * P_i \right] + B \tag{13}$$

$$g(x) = 22 * \left(\frac{1 - e^{\frac{-x}{3}}}{1 + e^{\frac{-x}{3}}} \right) \tag{14}$$

4.2.3 Chromosome

The chromosome for the ANN is a table with the neuron’s associated weights and bias, with a step size of 0.1 (Table 1).

The crossover and mutation of the ANN chromosome during the reproduction phase of the GA is as follows. The chromosome contains the weights of the 7 neurons, which has been divided into the hidden layer neurons 1–4 and output layers

Table 1 The ANN chromosome

Neuron P1	W_{11}	W_{12}	N/A	N/A	B_1
Neuron P2	W_{21}	W_{22}	N/A	N/A	B_2
Neuron P3	W_{31}	W_{32}	N/A	N/A	B_3
Neuron P4	W_{41}	W_{42}	N/A	N/A	B_4
Neuron P5	W_{51}	W_{52}	W_{53}	W_{54}	B_5
Neuron P6	W_{61}	W_{62}	W_{63}	W_{64}	B_6
Neuron P7	W_{71}	W_{72}	W_{73}	W_{74}	B_7

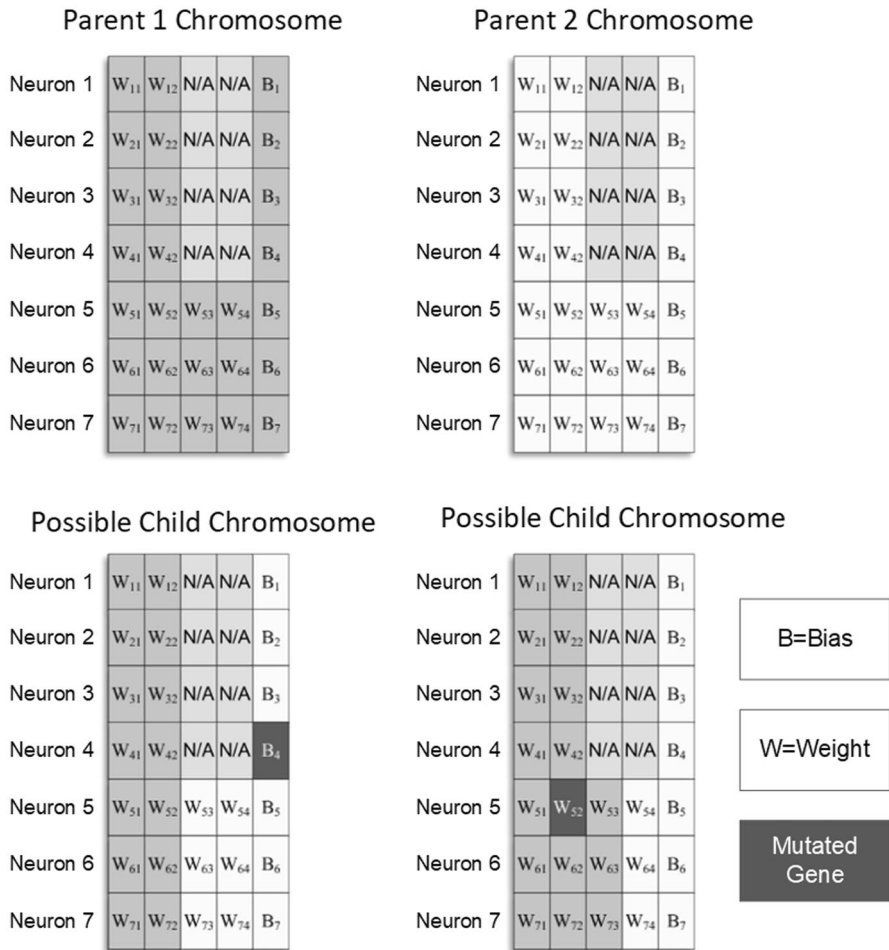


Fig. 19 Representation of the ANN chromosome crossover and mutation operation

neurons 5–7, The chromosome is split into these two layers, with single point crossover performed between the two parents of that layer, this can be considered to be multipoint when applied to the complete chromosome (Fig. 19). A mutation rate of 1% is applied to the chromosome.

5 Genetic algorithm

A GA is used to evolve a walking gait for a hexapod robot which allows the robot to walk one meter in a straight line in a stable manner. The gait is comprised of twenty steps, ten for the ground phase where the leg is on the ground pushing the robot forward, and ten for the air phase where the leg is in the air returning to its ground

starting position. The ground and air phases of the gait are evolved separately using the same GA. The same GA with a population of 100 was applied to both the ANN and the EHW controllers.

Initially all six legs were evolved to develop the gait and performance was determined based on the movement of the robot with each leg. However, it was realized the movement of only a single leg needed to be evolved to develop a suitable walking gait, because the motion for all legs is fundamentally the same. The only difference between the motion of the leg, is the phase of each leg during the walking gait. So, fitness is determined on the motion of a leg rather than all six legs of the hexapod.

5.1 Reproduction and selection

Reproduction uses multi-point crossover with a 1% mutation rate. The selection process uses two stage binary tournament (Fig. 20). In this method, the population is randomly shuffled, then put into the first tournament to compete individual versus individual. The fifty winners of the competition have two outcomes: (1) they are used to reproduce fifty children that compete in the second tournament; and (2) they are directly carried on to the next generation. The losers of the first tournament and new children then compete in a second tournament to select the remaining individuals that will survive for the next generation.

Tournament-based selection was chosen as it is a well-known selection method used in GAs, it is highly effective requiring fewer generations to evolve solutions than other methods such as roulette [37], binary tournament selection is the most common implementation of this strategy, because (a) the simplicity of its implementation and (b) having a larger tournament size increases the chances of loss of diversity [38]. The mutation rate of 1% is chosen at a level that aids the search for an optimal solution but is not overly destructive of the chromosome, this is important as a higher mutation rate will make reaching the optimal solution difficult if useful genes past from parents are being lost due to excessive mutation. The population size of 100 is chosen to ensure a more diverse population particularly early in the evolution

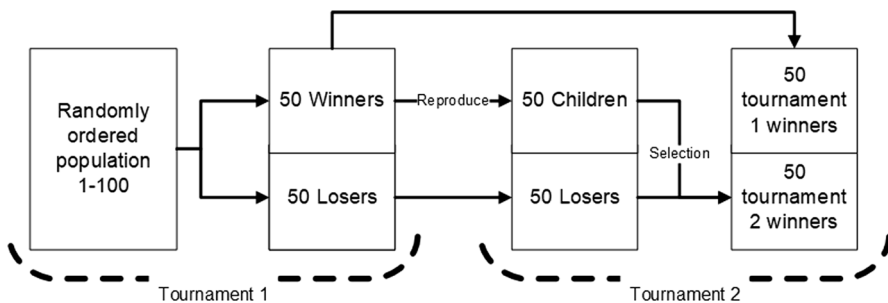


Fig. 20 Two stage binary tournament selection

process. Reduced population size may improve the computation time of the program per generation but can reduce the evolutionary rate.

5.2 Fitness function

To reduce the “reality gap” between the robot simulation and the real robot the fitness functions were linked to the physical robot’s observed behavior including backlash, accuracy, and “play” in the servos. Firstly, it was observed that the robot would sag when a leg was lifted off the ground, therefore a minimum height is required when the leg was lifted, and secondly the robot heading was affected by the trajectory of the leg, with a symmetrical motion giving an improved heading. The following equations are used to determine the fitness with a lower value showing a better fitness.

5.2.1 Ground phase fitness

The factors used to determine the ground phase fitness are: (1) symmetry of the gait; (2) the height is kept constant (smoothness); (3) is the leg contributing to forward motion; and (4) physical limitations, Eq. (15).

$$fitness = fitness_{symmetry} + fitness_{forward\ motion} + fitness_{smoothness} + fitness_{physical\ limits} \quad (15)$$

Symmetry fitness is quantified by checking if the selected start position of the leg is the same as the end position except for the hip angle γ is the opposite sign, implemented using an error squared function. Ideally for example if, Start Position, $\rho = [-80, 25, 20]$ the End Position, $\sigma = [-80, 25, -20]$, Eq. (16).

$$fitness_{symmetry} = (|\rho_\alpha| - |\sigma_\alpha|)^2 + (|\rho_\beta| - |\sigma_\beta|)^2 + (|\rho_\gamma| - |\sigma_\gamma|)^2 \quad (16)$$

Forward motion fitness is quantified by three formulae: (1) Applying an error squared function based on the start height, the average height, and the height at each step in the gait of the foot. The foot must be pushing against the ground to contribute to moving the robot forward ($z = \text{height}$). (2) Applying an error squared function based on the start x position, the average x position, and the x position at each step in the gait. The foot must maintain the same straight heading while pushing against the ground to contribute to moving the robot forward. (3) Penalties are applied if the distance the robot is pushed forward is below a set limit based on experimentation with the hexapod, Eqs. (17–20).

$$fitness_{fm1} = |z_0 - z_{avg}|^2 + |z_0 - z_{10}|^2 + \sum_{i=step1}^{step10} |z_i - z_{avg}|^2 \quad (17)$$

Table 2 Maximum and minimum joint angle limits

Min γ	Max γ	Min α	Max α	Min β	Max β
$-\text{abs}(\gamma_0)$	$\text{abs}(\gamma_0)$	$-\frac{\pi}{4}$	$\frac{\pi}{4}$	$-\frac{3\pi}{4}$	$-\frac{\pi}{4}$

$$fitness_{fm2} = |x_0 - x_{avg}|^2 + |x_0 - x_{10}|^2 + \sum_{i=step1}^{step10} |x_i - x_{avg}|^2 \tag{18}$$

$$fitness_{fm3} = \text{fixed penalty, for distance} < LL \tag{19}$$

$$fitness_{forward\ motion} = fitness_{fm1} + fitness_{fm2} + fitness_{fm3} \tag{20}$$

Smoothness fitness is quantified by applying fixed penalties for uneven jumps between foot positions. In particular when the foot is not moved during a step, this prevents large changes in position occurring to complete the gait (21). F_i is the vector that describes the foot position at a step in the gate i .

$$fitness_{smoothness} = \sum_{i=step1}^{step10} \text{fixed penalty, for } F_i = F_{i-1} \tag{21}$$

Penalties are applied when a step in the gait contains angular positions that are not physically possible as defined by Eq. (22). To improve the evolutionary efficiency the maximum and minimum angles are set to values less than the true physical limits. The specific limits of the joint angles are shown in (Table 2).

$$fitness_{physicallimits} = \begin{cases} \text{fixed penalty, for } \gamma > \text{max or } \gamma < \text{min} \\ \text{fixed penalty, for } \alpha > \text{max or } \alpha < \text{min} \\ \text{fixed penalty, for } \beta > \text{max or } \beta < \text{min} \end{cases} \tag{22}$$

5.2.2 Air phase fitness

The factors used to determine the air phase fitness are: (1) symmetry of the gait; (2) is the foot lifted and put down smoothly; (3) the amount the leg is lifted; and (4) physical limitations.

$$fitness = fitness_{symmetry} + fitness_{lift} + fitness_{smoothness} + fitness_{physical\ limits} \tag{23}$$

Symmetry fitness is quantified by: (1) checking if the end position of the leg is the same as the ground phase starting position, implemented using an error squared function like the ground phase; and (2) ensuring the foot is at its highest position approximately at the halfway point of the gait step n (n can be step 5, 6 or 7), Eqs. (24–26).

$$fitness_{sym1} = (\rho_\alpha - \sigma_\alpha)^2 + (\rho_\beta - \sigma_\beta)^2 + (\rho_\gamma - \sigma_\gamma)^2 \quad (24)$$

$$fitness_{sym2} = \text{fixed penalty, for step } n \text{ not } 5, 6 \text{ or } 7 \quad (25)$$

$$fitness_{symmetry} = fitness_{sym1} + fitness_{sym2} \quad (26)$$

Lift fitness is quantified by: (1) applying a fixed penalty if the middle step in the gait n is below the start height of the foot; and (2) by checking the height of the middle of the gait step n as this should be the highest point so will influence the height of the foot across the gait. A fixed penalty is applied if the highest point of the gait is below a certain limit. The lower limit (LL) was found from experimenting with models that were implemented on the hexapod. The lower limit helps to prevent drag in the real hexapod, Eqs. (27–29).

$$fitness_{lift1} = \text{fixed penalty, for } z_n < z_0 \quad (27)$$

$$fitness_{lift2} = \text{fixed penalty, for } z_n < LL \quad (28)$$

$$fitness_{lift} = fitness_{lift1} + fitness_{lift2} \quad (29)$$

Smoothness fitness is quantified by applying fixed penalties for erratic jumps between foot positions. When the leg is lifted F_i must be lower than F_{i+1} and when the leg is being lowered foot F_i must be higher than F_{i+1} .

$$fitness_{smoothness} = \sum_{i=step0}^{stepn} \text{fixed penalty, for } 0 > |z_i - z_{i-1}| \\ + \sum_{i=stepn+1}^{step10} \text{fixed penalty, for } 0 < |z_i - z_{i-1}| \quad (30)$$

Physical limits fitness is quantified by servo angles which are possible. Fixed penalties are applied when a step in the gait contains angular positions that are not possible Eq. (22).

6 Results

The EHW and ANN controllers were evolved to create a hexapod walking gait allowing the robot to move forward in a straight line using the same GA. The two phases of the leg motion, (leg on the ground, and leg in the air) were evolved separately with each phase broken into ten stages. One hundred solutions for each controller were evolved, with the maximum number of generations limited to 1000 (500 generations for each phase). Both controllers use the same input data and are

evolved to output a change in position of three servos in the leg for each stage of the hexapod's gait.

6.1 Simulation

6.1.1 Evolutionary efficiency

Evolutionary efficiency is determined by how many generations are required to create an optimum walking gait, an optimum walking gait allows the robot to walk forward in a straight line, maintaining a constant heading and body attitude. The EHW has a better evolutionary efficiency, however both controllers can reach a suitable fitness within 100 generations per phase (Fig. 21).

An analysis of the combined ground and air phase results (Fig. 22 and Table 3) shows the EHW median is approximately 45% faster than the ANN with 95% of the ANN results taking longer than the median EHW result. The ANN also had more than double the number of failed evolution attempts, where the desired fitness was not achieved within the max generation limit of 1000 generations. Even though the majority of the EHW results are significantly faster than the ANN results, the EHW does have a small number of outliers (Fig. 22) where the GA gets stuck at a local optima taking a significant number of generations to reach the target fitness, this can be seen in Fig. 21 with the EHW lower part of the range of results. These findings are summarized by Table 3.

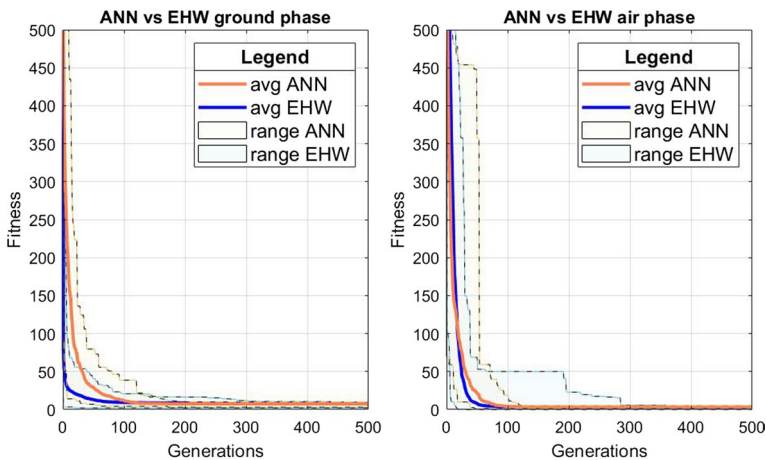


Fig. 21 ANN versus EHW Evolutionary Efficiency

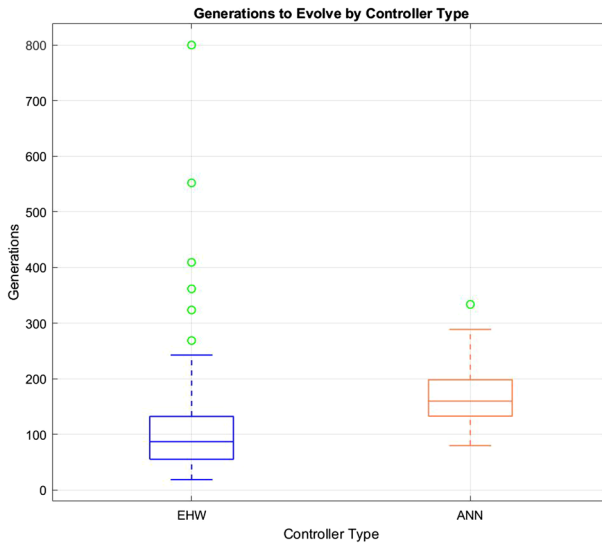


Fig. 22 Box and Whisker analysis of the combined ground and air phase results

Table 3 The combined Air and Ground phase evolutionary efficiency of both controllers; it shows the EHW is faster to evolve than the ANN requiring fewer generations on average to evolve and has fewer failed evolutions where the target fitness was not achieved

Controller	Mean generations	Median generations	95th percentile	Minimum generations to reach desired fitness	Maximum generations to Reach desired fitness	No. of failed evolutions
EHW	116	87	<25	19	800	3
ANN	166	160	<95	70	334	10

6.1.2 Controller performance

The final evolved solutions of both controllers were implemented in a PyBullet [39] simulation of the hexapod (Fig. 23) with an 80 ms control interval, meaning a step in the gait is executed every 80 ms. The controller performance was measured by: (a) the ability to walk a straight line; (b) the ability to maintain a constant height and orientation of the body; and (c) the distance traveled.

The trajectory of the hexapod was observed (Online Resource 1) and plotted (Fig. 24). The plot shows that both evolved controllers had an excellent gait allowing the hexapod to walk in a straight line. It is noted that both controllers had a minor oscillation in their trajectories, but this is negligible and does not affect the heading of the robot.

The motion of the leg gait (Fig. 25) shows that both controllers evolved a stable gait. During the ground phase for each controller the foot of the leg tracks an

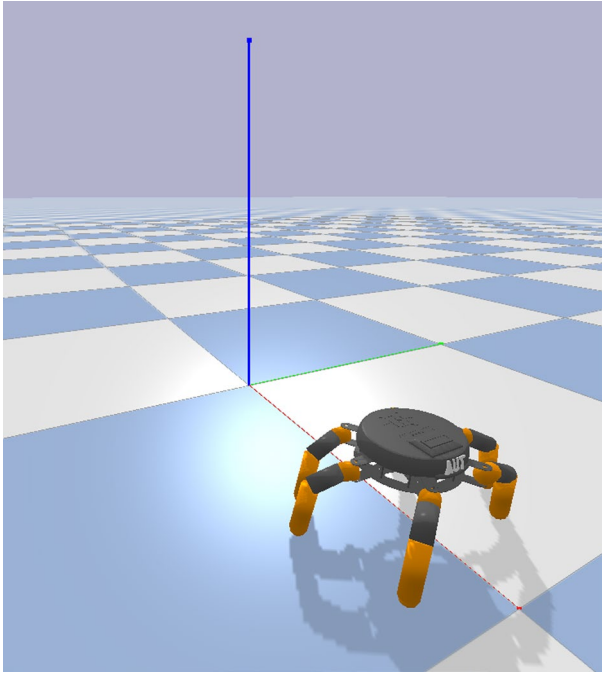


Fig. 23 Hexapod simulation using PyBullet

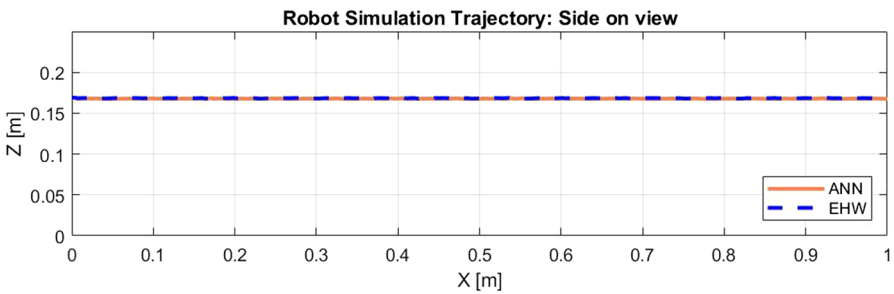
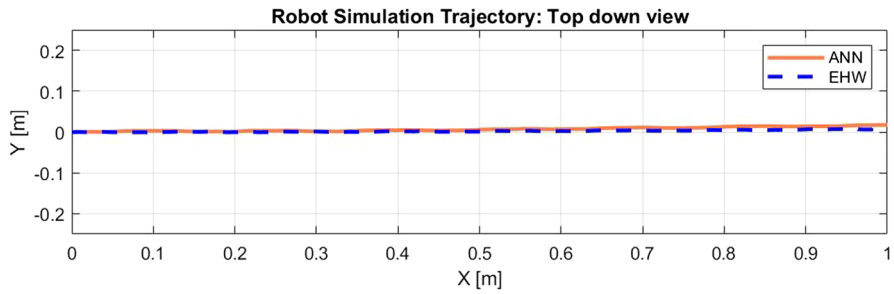


Fig. 24 Trajectory comparison of the ANN and EHW

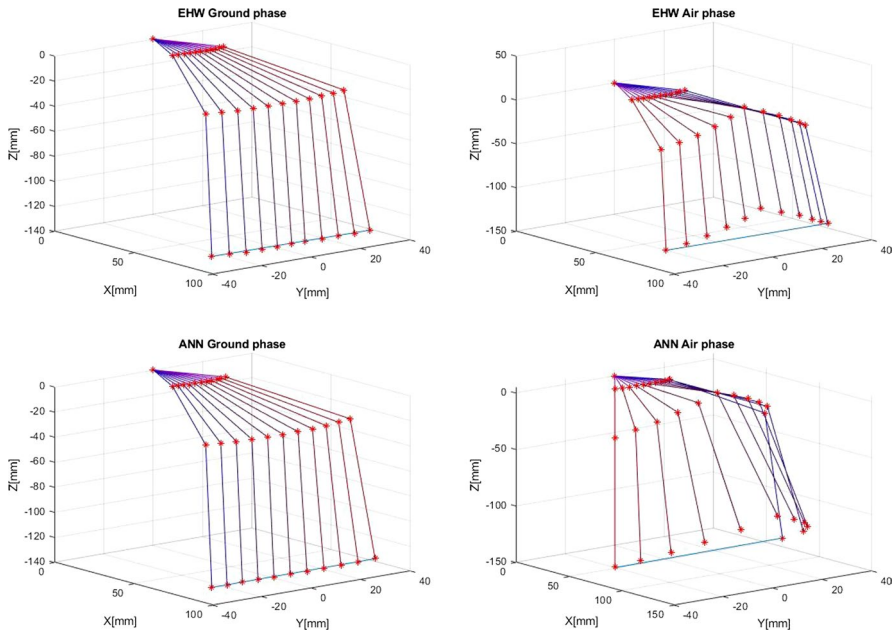


Fig. 25 Evolved ground and air phase leg motion comparison

approximate straight-line path taking even steps which correlates to the hexapod moving forward smoothly in a straight line. This was observed in the PyBullet simulation. During the air phase both controllers lift the foot of the leg high enough to prevent dragging the hexapod’s feet on the real robot due to the backlash in the servo motors and general play in the legs. The EHW results tend to lift the leg more vertically whereas the ANN results tend to combine lifting and swinging the leg out more to get enough separation between the foot and the floor.

6.2 Real robot

After successfully implementing both controllers in simulation, the evolved controllers were implemented on AUT’s physical hexapod robot (Fig. 3). The robot was specifically designed for experimentation, with the servo motors interfacing to a Terasic

Table 4 ANN & EHW implementation resource usage

System component	EHW logic elements	ANN logic elements
Configured NIOS II	13,469	6126
PWM servo controllers	395	395
EHW VFGPA	7027	n/a
NIOS II JTAG interface controller	317	315
Total	21,208	6836

DE0-Nano FPGA development board. The development board contains a 50 MHz clock source, and 32 Mbytes of SRAM. The 50 MHz clock was stabilized using a phase-locked-loop to drive the clock on the SRAM. Both the ANN and EHW produced a 20-bit variable per servo that defined the PWM mark, this was connected to a PWM generator (coded in Verilog), which was used to drive the servo motor PWM inputs. The FPGA was programmed with a NIOS II softcore processor allowing both the ANN and EHW to be implemented and observed. The total amount of logic elements used in the FPGA, was approximately 7000 for the ANN and 21,000 for the EHW (Table 4). The EHW FPGA configuration required more resources for (a) the NIOS processor due to the PIO required to drive the CBS and EHW I/O; and (b) the EHW IP itself.

The board has a Bluetooth interface allowing the robot to be sent commands to walk forward and backward as well as rotating left and right. Walking backwards was achieved by decrementing the step counter input rather than incrementing it. The hexapod could turn left and right by incrementing the step counter for one side of the hexapod and decrementing the opposite. After observation of the hexapod (Online Resource 2), it could be seen that both controllers performed equally well in the real robot, indicating that the simulation was an accurate reflection of the robot.

7 Conclusions

It has been shown that the EHW controller can be implemented as an evolvable robot controller. The EHW matches an equivalent ANN controller in performance, with slightly better evolutionary efficiency. The model-based evolution produces efficient stable gaits for both the ANN and EHW controllers, which was displayed in the results taken from PyBullet simulation of the hexapod robot. Both evolved controllers can be implemented in a real hexapod robot to verify their performance. The ANN and EHW implemented effectively controlled the real robot, allowing it to walk forward, backwards and rotate left or right. In the field of evolutionary robotics EHW could be investigated further looking at its ability to be evolved quickly. In the results of this work, it is shown the EHW could be, in some cases evolved very quickly to produce an effective walking gait. The ability to evolve controllers quickly would be a desirable trait of modern robots allowing them to adapt to changes in their environment. Future work could investigate developing EHW to adapt so the robot can overcome faults in the system or changes in terrain, allowing the robot to traverse a variety of environments.

Supplementary Information The online version contains supplementary material available at <https://doi.org/10.1007/s10710-023-09452-4>.

Author contributions Both authors contributed to the initial study conception and design. FB carried out the final design and development of the Evolvable Hardware and Artificial Neural Network Controllers. FB carried out the software development of the tools used for analysis and comparison of the evolvable robotic controllers in simulation and on the physical robot. FB carried out the evolution and testing of the controllers in simulation and on the physical robot. FB and MB wrote the manuscript text and prepared the figures. Both authors have reviewed the manuscript.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. No funding was received for conducting this study.

Declarations

Conflict of interest All authors have no financial or non-financial interest to declare relevant to the subject matter or materials discussed in the article. The authors confirm that all affiliated organizations or entities have no financial or non-financial interest relevant to the subject matter or materials discussed in the article.

Ethical approval No ethical approval was required for this study.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. J.F. Miller, An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach, in *Proceedings of the Genetic and Evolutionary Computation Conference* vol. 2 (1999), pp. 1135–1142
2. J.F. Miller, D. Job, V.K. Vassilev, Principles in the evolutionary design of digital circuits: part I. *Genet. Program Evol. Mach.* **1**(1), 7–35 (2000)
3. Á. Pintér-Bartha, A. Sobe, W. Elmenreich, Towards the light: comparing evolved neural network controllers and finite state machine controllers. in *Presented at the 10th International Workshop on Intelligent Solutions in Embedded Systems*, (Klagenfurt, Carinthia, 2012), pp. 5–6
4. M. Beckerleg, J. Matulich, P. Wong, A comparison of three evolved controllers used for robotic navigation. *AIMS Electron. Electr. Eng.* **4**(3), 259–286 (2020). <https://doi.org/10.3934/ElectrEng.2020.3.259>
5. M. Okura, H. Matsumoto, A. Ikeda, K. Murase, Artificial evolution of FPGA that controls a miniature mobile robot Khepera. in *SICE Annual Conference in Fukui*, (Fukui University, Japan, 2003)
6. K.C. Tan, C.M. Chew, K.K. Tan, L.F. Wang, Y.J. Chen, Autonomous robot navigation via intrinsic evolution. in *Evolutionary Computation, CEC '02. Proceedings of the 2002 Congress*, vol. 2 (2002), pp. 1272–1277
7. A.M. Tyrrell, R.A. Krohling, Y. Zhou, Evolutionary algorithm for the promotion of evolvable hardware. *Comput. Digit. Tech. IEE Proc.* **151**(4), 267–275 (2004)
8. R. Krohling, Y. Zhou, A. Tyrrell, Evolving FPGA-based robot controllers using an evolutionary algorithm. in *1st International Conference on Artificial Immune Systems, Canterbury* (2002)
9. H. Seok, K. Lee, J. Joung, B. Zhang, An on-line learning method for object-locating robots using genetic programming on evolvable hardware. in *International Symposium on Artificial Life and Robotics* (2000), pp. 321–324, citeseer.ist.psu.edu/456254.html
10. Y. Rui, S. Yanmei, H. Kun, Y. Yang, Online evolution of image filters based on dynamic partial reconfiguration of FPGA. in *2015 11th International Conference on Natural Computation (ICNC)* (2015), pp. 999–1005, <https://doi.org/10.1109/ICNC.2015.7378128>
11. R. Dobai, L. Sekanina, Image filter evolution on the Xilinx Zynq Platform. in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)* (2013), pp. 164–171, <https://doi.org/10.1109/AHS.2013.6604241>
12. R. Dobai, L. Sekanina, Towards evolvable systems based on the Xilinx Zynq platform. in *2013 IEEE International Conference on Evolvable Systems (ICES)* (2013), pp. 89–95, <https://doi.org/10.1109/ICES.2013.6613287>
13. A.K. Srivastava, A. Gupta, S. Chaturvedi, V. Rastogi, Design and simulation of virtual reconfigurable circuit for a Fault Tolerant System. in *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)* (2014), pp. 1–4, <https://doi.org/10.1109/ICRAIE.2014.6909277>

14. P.N. Kumar, S. Anandhi, J.R.P. Perinbam, Evolving virtual reconfigurable circuit for a fault tolerant system. in *2007 IEEE Congress on Evolutionary Computation* (2007), pp. 1555–1561, <https://doi.org/10.1109/CEC.2007.4424658>
15. K. Glette, J. Torresen, M. Hovin, Intermediate Level FPGA Reconfiguration for an Online EHW Pattern Recognition System. in *2009 NASA/ESA Conference on Adaptive Hardware and Systems* (2009), pp. 19–26, <https://doi.org/10.1109/AHS.2009.46>
16. K. Glette, P. Kaufmann, Lookup table partial reconfiguration for an evolvable hardware classifier system. in *2014 IEEE Congress on Evolutionary Computation (CEC)* (2014), pp. 1706–1713, <https://doi.org/10.1109/CEC.2014.6900503>
17. O. Garnica, K. Glette, J. Torresen, Comparing three online evolvable hardware implementations of a classification system. *Genet. Program. Evol. Mach.* **19**(1), 211–234 (2018). <https://doi.org/10.1007/s10710-017-9312-1>
18. J. Wang, C.H. Piao, C.H. Lee, FPGA Implementation of evolvable characters recognizer with self-adaptive mutation rates. in *Adaptive and Natural Computing Algorithms*, ed. by, B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro, (Springer, Berlin, Heidelberg, 2007), pp. 286–295
19. M. Beckerleg, J. Collins, Evolving electronic circuits for robotic control. In *Presented at the 15th International Conference on Mechatronics and Machine Vision in Practice* (Auckland, New Zealand, 2008)
20. M. Beckerleg, J. Collins, Using a hardware simulation within a genetic algorithm to evolve robotic controllers. in *Presented at the International Conference on Intelligent Automation and Robotics (ICIAR'11)* (San Francisco, USA, 2011)
21. S. Lawrence, C.L. Giles, T. Ah Chung, A.D. Back, Face recognition: a convolutional neural-network approach. *IEEE Trans. Neural Netw.* **8**(1), 98–113 (1997). <https://doi.org/10.1109/72.554195>
22. M. Coşkun, A. Uçar, Y. Ö. Y. Demir, Face recognition based on convolutional neural network. in *2017 International Conference on Modern Electrical and Energy Systems (MEES)* (2017), pp. 376–379, <https://doi.org/10.1109/MEES.2017.8248937>
23. J. Matulich, *A Comparison of Three Robotic Controllers for Navigation*, School of Engineering, AUT, 2017.
24. V. Abhishek, A. Mukerjee, H. Karnick, Artificial ontogenesis of controllers for robotic behavior using VLG GA. in *Systems, Man and Cybernetics, IEEE International Conference* vol. 4 (2003), pp. 3376–3383
25. D. Harter, Evolving neurodynamic controllers for autonomous robots. in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on* vol. 1 (2005), pp. 137–142. <https://doi.org/10.1109/ijcnn.2005.1555819>
26. W. Elmenreich, G. Klingler, Genetic evolution of a neural network for the autonomous control of a four-wheeled robot. in *Artificial Intelligence: Special Session, 2007. MICAI 2007. Sixth Mexican International Conference on*, 4–10 Nov. 2007 (2007), pp. 396–406, <https://doi.org/10.1109/micai.2007.13>
27. W. Wahab, Autonomous mobile robot navigation using a dual artificial neural network. in *TENCON 2009–2009 IEEE Region 10 Conference* (2009), pp. 1–6. <https://doi.org/10.1109/tencon.2009.5395892>
28. P.K. Mohanty, D.R. Parhi, A.K. Jha, A. Pandey, Path planning of an autonomous mobile robot using adaptive network based fuzzy controller. in *Advance Computing Conference (IACC), 2013 IEEE 3rd International* (2013), pp. 651–656. <https://doi.org/10.1109/IAdCC.2013.6514303>
29. P. Karlra, N.R. Prakash, A neuro-genetic algorithm approach for solving the inverse kinematics of robotic manipulators. in *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme—System Security and Assurance (Cat. No.03CH37483)*, vol. 2 (2003), pp. 1979–1984. <https://doi.org/10.1109/ICSMC.2003.1244702>
30. G.B. Parker, Z. Lee, Evolving neural networks for hexapod leg controllers. in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* vol. 2, (IEEE, 2003), pp. 1376–1381
31. J.C. Gallagher, An evolvable hardware layer for global and local learning of motor control in a hexapod robot. *Int. J. Artif. Intell. Tools* **14**(06), 999–1017 (2005)
32. D. Berenson, N. Estevez, H. Lipson, Hardware evolution of analog circuits for in-situ robotic fault-recovery, in *2005 NASA/DoD Conference on Evolvable Hardware (EH'05)*, (2005), pp. 12–19, <https://doi.org/10.1109/EH.2005.30>
33. J. Currie, M. Beckerleg, J. Collins, Software evolution of a hexapod robot walking gait. *Int. J. Intell. Syst. Technol. Appl.* **8**(1–4), 382–394 (2010)
34. G.B. Parker, Z. Lee, Evolving neural networks for hexapod leg controllers, in *Presented at the IEEE International Workshop on Intelligent Robots and Systems* (Las Vegas, 2003)

35. C.F. Juang, Y.C. Chang, C.M. Hsiao, Evolving gaits of a hexapod robot by recurrent neural networks with symbiotic species-based particle swarm optimization. *IEEE Trans. Industr. Electron.* **58**(7), 3110–3119 (2011). <https://doi.org/10.1109/TIE.2010.2072892>
36. H. Heijnen, D. Howard, N. Kottege, A testbed that evolves hexapod controllers in hardware. in *2017 IEEE International Conference on Robotics and Automation (ICRA) (2017)*, pp. 1065–1071, <https://doi.org/10.1109/ICRA.2017.7989128>
37. J. Zhong, X. Hu, J. Zhang, M. Gu, Comparison of performance between different selection strategies on simple genetic algorithms. In *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, vol. 2 (IEEE, 2005), pp. 1115–1121
38. N.M. Razali, J. Geraghty, Genetic algorithm performance with different selection strategies in solving TSP, in *Proceedings of the world congress on engineering vol. 2(1)* (International Association of Engineers Hong Kong, China, 2011), pp. 1–6
39. E. Coumans, Y. Bai, PyBullet, a Python module for physics simulation for games, robotics and machine learning. (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Fraser Borrett¹ · Mark Beckerleg¹

¹ Department of Electrical and Electronic Engineering, Auckland University of Technology, Private Bag 92006, Auckland, New Zealand