The Application of a Virtual Programmable Logic Device for Robotic Control and Pattern Recognition

Fraser Alexander Cornforth Borrett

A thesis submitted to AUT University
in fulfilment of the requirements for the degree of
Doctor of Philosophy (PhD)

2025

School of Engineering
Supervisors:
Mark Beckerleg & Martin Stommel

Attestation of Authorship

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

Signed……...

Dedication

I dedicate this thesis to all my family and friends, without whom this epic journey would have not been possible.

Acknowledgement

I would like to express my gratitude to my supervisors. First, to Dr. Mark Beckerleg, who has been with me from the very beginning of this journey, offering support, invaluable guidance, and constant encouragement - for which I will be forever grateful. I also wish to thank Dr. Martin Stommel, who joined partway through this journey but whose expert insight and mentorship have greatly contributed to the success of this thesis.

I would also like to acknowledge my parents and my brother, for their endless support throughout my pursuit of higher education. Without their belief in me and their unwavering encouragement and support, I would not have been able to achieve what I have, thank you.

Abstract

This research derives and evaluates a novel machine learning architecture called the Virtual Programmable Logic Device (VPLD), and if the VPLD can become a competitor to the Artificial Neural Network (ANN) when evolved for applications in robotic control and pattern recognition. The VPLD is based on the architecture of a programmable logic device but is coded in software rather than in hardware. This allows the VPLD to be run on CPU based platforms including standard PCs, mobile phones and ARM based embedded systems such as the Raspberry Pi. The operation of the VPLD can be configured and optimised using evolutionary algorithms. The VPLD is inspired by previous Evolvable Hardware architectures evolved for applications such as robotic control. In the Evolvable Hardware domain electronic circuits are evolved on programmable logic devices such as the field programmable gate array (FPGA).

The VPLD is investigated in two fields: 1) evolutionary robotics where the gait control of a hexapod robot and the autonomous navigation of a two-wheel drive mobile robot is examined; and 2) in pattern recognition where character recognition, and melanoma classification are evaluated. In both application domains two types of VPLD are investigated, the first is the digital VPLD (D-VPLD) which mimics the PLD binary variables and digital logic, the second is the floating-point VPLD (F-VPLD) which uses floating-point variables and mathematical functions. The floating-point variables are complex to implement in hardware on a FPGA.

In the gait control of a hexapod, a evolvable hardware implementation is designed to validate the VPLD architecture. In this validation it is demonstrated that the VPLD compared to the evolvable hardware for robotic control is faster to evolve, as well as simpler and cheaper to implement.

To assess the VPLDs controller and classifier performance it is benchmarked against an ANN. The VPLD and ANN are evolved for the robotic control and pattern recognition problems using the same evolutionary algorithm. The results of the experiments show the VPLD is a viable alternative to the ANN in both robotic control and pattern recognition applications as the VPLD could achieve the same, and in some cases better performance than the ANN.

Table of Contents

Abbreviations and Acronyms

VPLD	Virtual Programmable Logic Device
EHW	Evolvable Hardware
ANN	Artificial Neural Network
D-VPLD	Digital Virtual Programmable Logic Device
F-VPLD	Floating-point Virtual Programmable Logic Device
FAB	Function Array Block
FE	Function Element
EA	Evolutionary Algorithm
GA	Genetic Algorithm
ES	Evolutionary Strategies
DE	Differential Evolution
PSO	Particle Swarm Optimisation
CPU	Central Processing Unit
PC	Personal Computer
IDE	Integrated Development Environments
SoC	System on Chip
PLD	Programmable Logic Device
FPGA	Field Programable Gate Array
FPAA	Field Programable Analogue Array
CBS	Configuration Bitstream
HDL	Hardware Description Language
LAB	Logic Array Block
LE	Logic Element
LUT	Lookup Table
MSB	Most significant bit
IR	Infrared
LDR	Light Dependant Resistor
2WD	Two-wheel drive
OA	Obstacle Avoidance
LF	Light Following
OA-LF	Obstacle Avoidance and Light Following
RPM	Revolutions Per Minute
DOF	Degrees of Freedom
OCR	Optical character recognition
ABCD	Asymmetry, Boarder, Colour, and Diameter
MRI	Magnetic Resonance Imaging
CT	computed tomography
CLAHE	contrast limited adaptive histogram equalization
CV	Computer Vision
AUT	Auckland University of Technology

Table of Figures

Tables

Chapter 1

Chapter 1: Introduction

This chapter provides an overview of the thesis, explaining what a virtual programmable logic device is and how it can be used in robotic control and pattern recognition. The research questions for this thesis are listed and finally an overview of the chapters is listed.

1.1. Overview

The focus of this research is the development of a novel machine learning architecture called the Virtual Programmable Logic Device (VPLD), and to determine if the VPLD can become a competitor to the Artificial Neural Network (ANN) when evolved for applications in robotic control and pattern recognition. The VPLD is based on a programmable logic device (PLD) which uses programmable function elements and routing between the function elements. However instead of using hardware reconfigurable digital circuits, the VPLD is the software equivalent of the PLD that can be executed on a central processing unit (CPU). The aim of the VPLD is to improve on the speed of optimization for reconfigurable hardware architectures implemented on field programmable gate arrays (FPGA), while also reducing the complexity and cost of implementation associated with such architectures. These reconfigurable hardware architectures are called evolvable hardware (EHW). The EHW and the VPLD share a similar architecture, but the VPLD is run in software on a processor rather than the EHW which is implemented as an electronic circuit running on a FPGA. As the EHW is implemented on a FPGA, this limits the devices it can be deployed on. The VPLD on the other hand can run on a variety of platforms because it is processor driven, including 1) a desktop PC, 2) an ARM based embedded system like a Raspberry Pi, and 3) a 32-bit microcontroller like the ESP32. The function of the VPLD is dynamically configured using common machine learning optimization algorithms such as genetic algorithms (GA) from the field of evolutionary computation. As a result of this thesis, a foundation for the VPLD has been developed, including 1) the implementation of a virtual PLD architecture in software, 2) methodology for the optimization of the VPLD function, 3) a mathematical model of the VPLD, and 4) the definition of the VPLD hyperparameters.

The VPLD is investigated in two fields, the first is in evolutionary robotics where the gait control of a hexapod robot and the autonomous navigation of a two-wheel drive mobile robot is examined. The second field is in pattern recognition where binary character recognition and melanoma detection are evaluated. A comparison of the VPLD and an ANN is conducted for each of these tasks. It was found that for the robotic control and pattern recognition experiments performed in this research, the performance of the VPLD is comparable to an ANN.

1.2. Background Information

The EHW are evolvable electronic circuits commonly implemented on a FPGA, but they have also been implemented on analogue devices using Field Programmable Analogue Array (FPAA) [1, 2]. The EHW controllers have been evolved for various tasks using evolutionary algorithms (EA) such as the GA or evolutionary strategies (ES)[1, 2]. The goal of these evolutionary algorithms is to optimize the configuration bitstream (CBS) used to program the FPGA device[1-3]. Early approaches used FPGA devices where the CBS could be directly manipulated without the risk of generating configurations with output-to-output connections that can destroy the FPGA device[4, 5]. This approach is not possible in modern FPGA devices. Therefore, EHW architectures using fixed input/output connections are implemented on the FPGA fabric[6-9]. EHW robotic controllers[10-14] have been evolved for tasks such as locomotion control, autonomous navigation, and robotic manipulators. Outside of robotics EHW architectures have been evolved for tasks in pattern recognition[15], image processing[16] and fault tolerant systems[14, 17, 18]. The EHW has several challenges including, 1) the requirement of expensive custom hardware[15]; 2) the complex hardware descriptive languages and the multiple development tools required for configuration; and 3) the process of developing and compiling the EHW designs on these platforms is slow, making design refactoring a time-consuming practice, slowing down research. The EHW also has limitations on the complexity of the problems it can solve in robotics, where the simulation environments used for training have become more sophisticated over time reducing the ability of on-chip training. For example, systems such as the DE10 Nano that have embedded ARM processors do not have the computation power required to efficiently run (if at all) complex simulations. An alternative would be serial communication with a PC to transfer the trial CBS and return the EHWs outputs [10].However, the speed of communication between an FPGA system and a PC is drastically slower compared to an ANN running natively on a PC for example.

Evolutionary algorithms are a series of nature inspired optimization algorithms based on Charles Darwin's theories on natural evolution[19], specifically using the concepts of reproduction whereby new individuals inherit beneficial characteristics from previous generations, and survival of the fittest ensures individuals with the best genetic characteristics for the environment survive to reproduce. Evolutionary algorithms such as the GA, ES, differential evolution (DE), and particle swarm optimisation (PSO) have been used to optimize ANNs and EHW for a variety of applications.

1.3. Research Objectives

The objective of this research is to: a) develop a new architecture in software as an improved alternative to the EHW architectures used in evolutionary robotics and pattern recognition research, focusing on reducing system complexity and cost of implementation, as well as the speed of evolution in robotics; b) demonstrate the performance of the new architecture in comparison to evolvable hardware; and c) demonstrate the performance of the new architecture in comparison to artificial neural networks. From the outlined challenges in the evolvable hardware field, the following list of research questions are investigated.

Main Question:

- Can a Virtual Programmable Logic Device be created that could run on a processor instead of reconfigurable hardware, and how does it compare to an Artificial Neural Network when evolved for robotic control and pattern recognition?

To address the main question a set of specific questions are asked:

1. How can a Programmable Logic Device be virtualised in software?
2. How do Virtual Programmable Logic Devices and Evolvable Hardware compare when evolved for robotic control?
3. How do Virtual Programmable Logic Devices and Artificial Neural Networks compare when evolved for robotic control?
4. How do Virtual Programmable Logic Devices and Artificial Neural Networks compare when evolved for pattern recognition?
5. What are the important hyperparameters for the Virtual Programmable Logic Device?

These questions and the experiments used to evaluate them are detailed in the following sections.

1.3.1 How can a PLD be virtualised in software?

The VPLD implementation will be investigated in Python. However, most programming languages including C and C++ could have been used. The overall function of the VPLD is configured using a configuration bitstream, that defines routing and functions used(detailed information can be found in Chapter 3). Important parameters are how quickly the VPLD can evolve, the computer resources required, and propagation delay (time to feedforward the data from input to output). The propagation delay is important particularly in robotics as the VPLD has to meet the timing constraints of the robot, this also impacts pattern recognition tasks that require low latency. The propagation delay is also important for the optimisation of the VPLD configuration because the VPLD is executed thousands of times during the fitness assessment, reducing the execution time of the VPLD, even by a minor amount will accumulate to a decrease in duration of the optimisation process. The implementation of the VPLD is discussed in Chapter 3, and the propagation delay of the VPLD is investigated in Chapter 4.

1.3.2 How do VPLDs and EHW compare when evolved for robotic control?

In order to assess the VPLD and EHW in robotic control (Chapter 4), both are evolved for a robotic locomotion problem, the gait control of a hexapod. This required the development of a hexapod model, simulation, and a genetic algorithm that could be used to evolve the VPLD and EHW. The robotic platform uses an off the shelf robot chassis combined with electronics developed for research in the EHW and evolutionary robotics fields. Two robotic simulations are developed, the first is a simulation of the hexapods leg that could run both on a standard PC and an embedded ARM processor in a Cyclone V FPGA. The same simulation has to be able to run on both platforms to make a fair comparison. The hexapod leg simulation is used to evolve the VPLD and EHW, a walking gait can then be produced using the evolved leg gaits utilizing alternate phases for each of the six legs. The second simulation also modelled on the hexapod platform, is a physics based simulation developed using PyBullet[20] which employs the Bullet physics engine. The PyBullet simulation is used to validate and compare the VPLD and EHW controller performance prior to deployment on the real robot. The VPLD and EHW are compared based on: 1) evolutionary efficiency; (how many generations and the computation time required to reach a solution); 2) the controller performance, (the walking ability of the deployed controllers, i.e. robot trajectory and body attitude); and 3) the propagation delay between the input and output of the controller.

For evolutionary efficiency, it is expected that the VPLD and EHW will take approximately the same number of generations to evolve, this is because the VPLD and EHW share the same architecture and are evolved using the same GA with the same parameters and crossover/mutation operators. They only differ in their implementation, one in software the other hardware.

VPLD Evolution

The VPLD is implemented on a PC which is running the robot simulation, the GA, and the VPLD. For the fitness assessment, the VPLD is configured by the GA with a trial chromosome, the robot simulation provides input to the VPLD, and the VPLD then returns the control signals for the given time step, this process continues until the end of the simulation at which point the fitness is calculated and returned to the GA (Fig. 1-1).

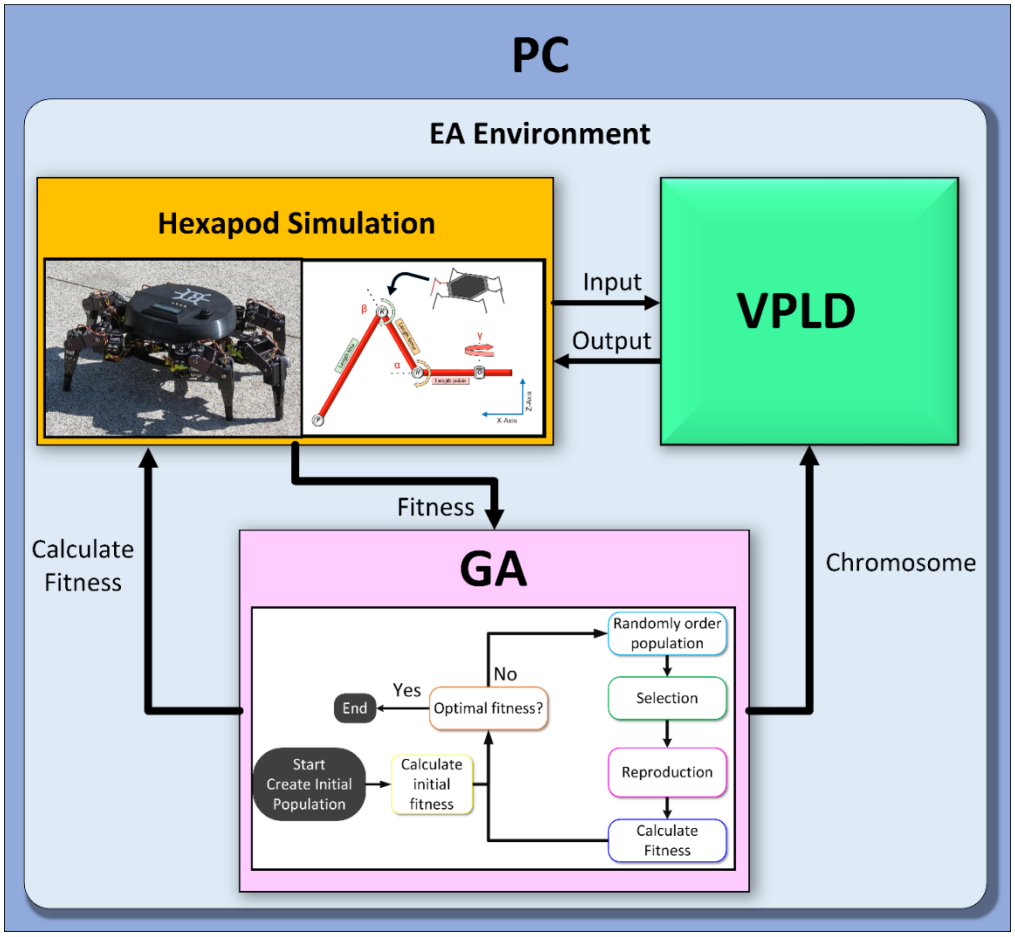

Fig. 1-1 The VPLD evolutionary algorithm environment for hexapod gait evolution

EHW Evolution

The EHW is implemented on an Altera Cyclone V FPGA (Fig. 1-2), which has an ARM hard processor. The GA and robot simulation run on the embedded ARM processor, while the EHW is running in the FPGA fabric. The interface between the ARM processor and the EHW on the FPGA fabric is the AXI lightweight bridge. This means the trial chromosome from the GA, and the inputs and outputs from the robotic simulation need to be transferred via the AXI lightweight bridge.

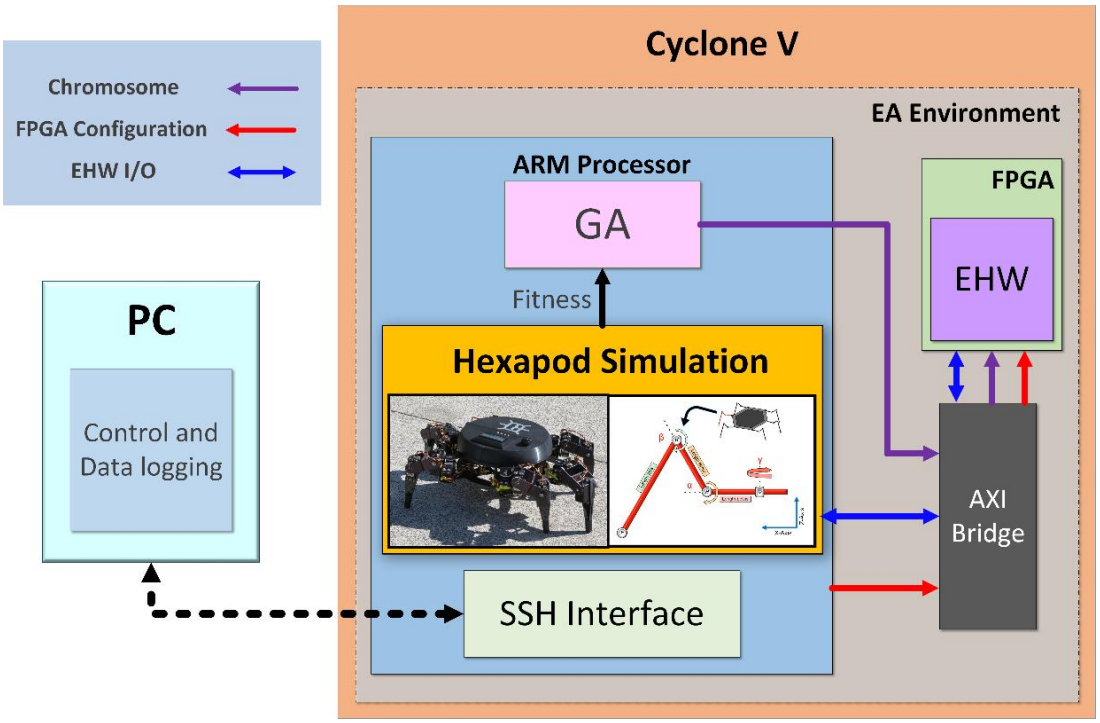

Fig. 1-2 The EHW evolutionary algorithm environment for hexapod gait evolution

1.3.3 How do VPLDs and ANNs compare when evolved for robotic control?

ANNs are used frequently in modern robotics applications. Because of this the ANN is used in this research as a benchmark to assess the performance of the VPLD. Two robotic applications are investigated, the first is robotic locomotion where the task is to evolve the gait of a hexapod robot as described above. The second is to evolve the robotic controllers for the navigation of a two wheeled mobile robot, for three separate tasks, obstacle avoidance (OA), light following (LF) and combined obstacle avoidance and light following (OA-LF). The ANNs weights and biases are evolved using the same GA that is used to evolve the VPLDs configuration bitstream.

To investigate the robotic navigation problems, the development of a simulation environment is required that could be used to evolve the VPLD and ANN controllers for the three tasks. The simulation (Fig. 1-3) is modelled on the dynamics of a physical robot platform developed at AUT for evolutionary robotics research. The simulation also required the modelling of the robot's sensors, six IR proximity sensors, and two light intensity sensors. Using the simulation a set of arenas are created to evolve the robotic controllers, and a second set are created to test the evolved controllers. A genetic algorithm is created to evolve the VPLD and ANN. As with the hexapod problem, the VPLD and ANN are compared based on evolutionary efficiency, and the controller performance.

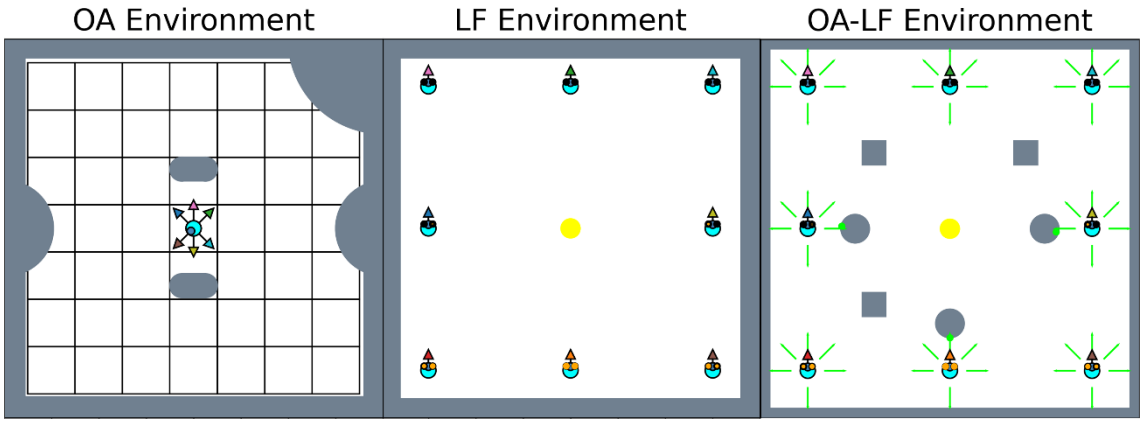

Fig. 1-3 2WD mobile robot simulation environments used for evolution of the VPLD and ANN robotic controllers, the figure shows the starting positions and orientation used to evaluate the controllers

The evolution environment used to evolve the VPLD and ANN for the 2WD autonomous navigation is shown in Fig. 1-4. The GA is used to optimize the VPLD configuration and the weights/biases of the ANN. The VPLD and ANN controllers are configured by the GA with an individual's chromosome. The robot simulation provides input to the controllers, then the controllers return the control signals for the given time step, this process continues until the end of the simulation period or if the robot crashes at which point the fitness is calculated and returned to the GA.

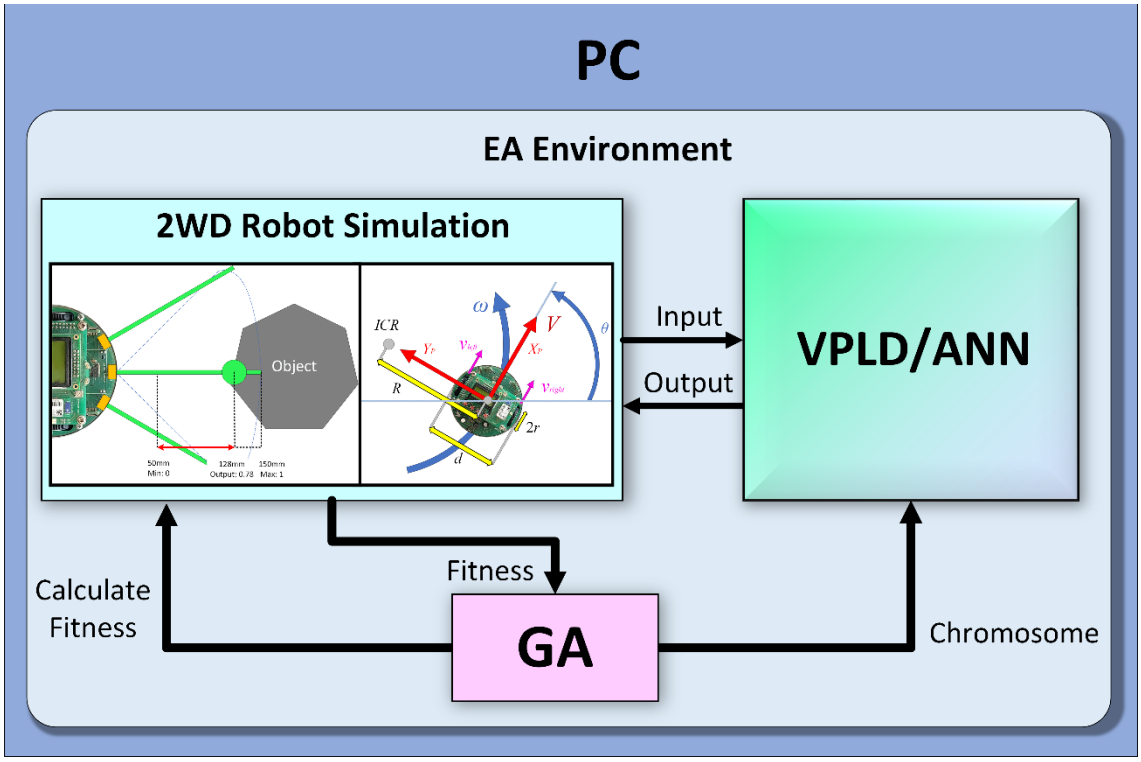

Fig. 1-4 The EA environment used for autonomous navigation to evolve the VPLD and ANN

1.3.4 How do VPLDs and ANNs compare when evolved for pattern recognition?

The VPLD is investigated in pattern recognition to see if its ability to evolve for robotic control tasks is transferable to a different domain. The VPLD is assessed in comparison to an evolved ANN for two pattern recognition tasks. The first task is evolving the VPLD and ANN for a multiclass classification of a set of 7x5 binary characters (A-Z). The second task is the evolution of VPLD and ANN classifiers for melanoma classification, this is a binary classification of skin lesions as either cancerous or non-cancerous.

Character recognition

The first task evolves the VPLD and ANN for character recognition of a set of 7x5 binary characters (A-Z). This required the development and investigation of VPLD and ANN architectures suitable for a multiclass classification problem. A GA with an appropriate fitness function is also developed for the evolution of the two architectures. Each architecture has 26 outputs, one for each character in the alphabet, for a given input image the output with the highest activation determines the prediction.

The EA environment used to evolve the VPLD and ANN for the character recognition problem is shown below (Fig. 1-5). The VPLD and ANN classifiers are configured by the GA with an individual's chromosome (CBS for the VPLD, weights & biases for the ANN). The character recognition program sequentially provides the input images, for each input image the VPLD/ANN classifiers return a prediction (A-Z). The predictions are used to calculate the fitness which is returned to the GA.

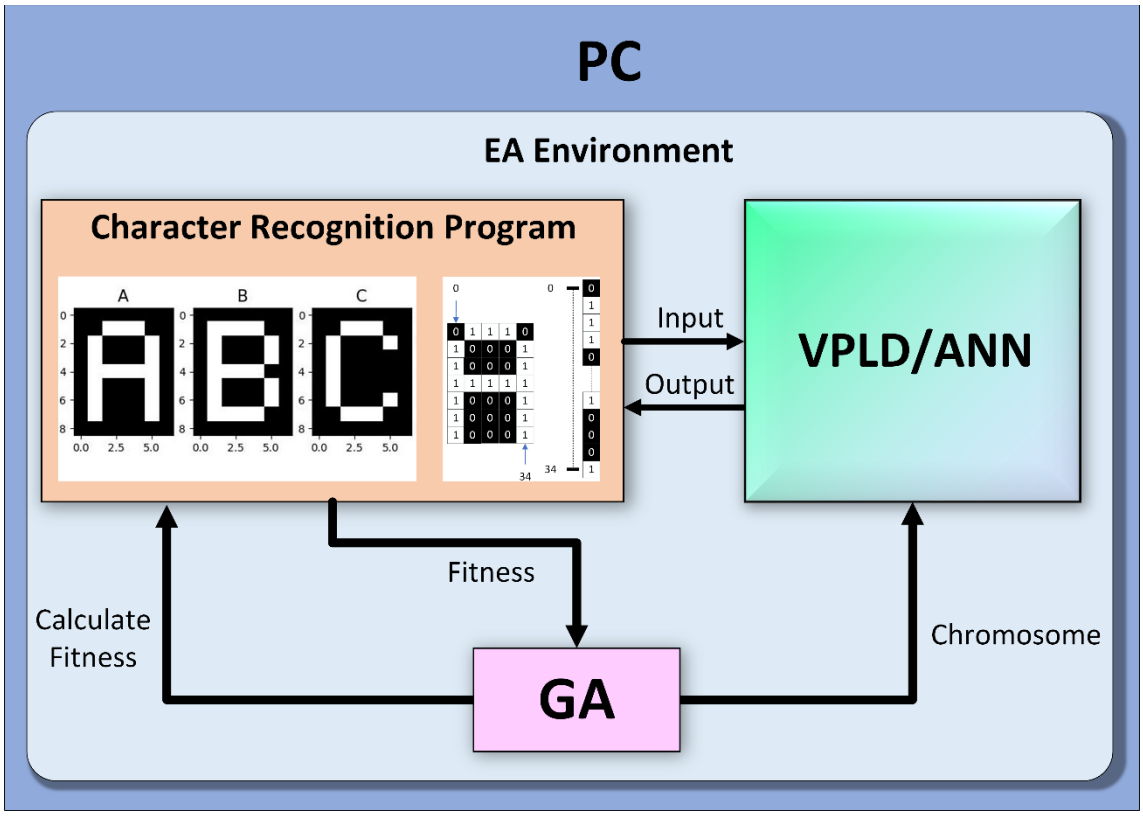

Fig. 1-5 The EA environment used for the character recognition problem to evolve the VPLD and ANN

Melanoma Detection.

The second pattern recognition task is melanoma detection, which is a binary classification task. The dataset used is the PH2 dataset[21], which is a set of 768x560 pixel dermoscopic images, with images of common nevi (non-cancerous), atypical nevi (non-cancerous), and melanomas (cancerous). To generate inputs for the VPLD and ANN a series of features are extracted from the skin lesion images. This required the development of a feature extraction algorithm which had 3 main steps; 1) preprocessing, consisting of illumination correction and a hair removal algorithm, 2) segmentation, using k-means clustering and Otsu thresholding; and 3) the feature extraction, firstly of ABCD features (Asymmetry, Boarder, Colour, and Diameter), and Harlick features using the Grey Level Co-Occurrence Matrix.

Along with the feature extraction method VPLD and ANN architectures are designed which are suitable for the binary classification of melanoma. A GA and associated fitness function (accuracy of the VPLD and ANN classifiers) are also developed. The EA environment used to evolve the VPLD and ANN for the melanoma detection is shown below (Fig. 1-6). The VPLD and ANN classifiers are configured by the GA with an individual's chromosome. The melanoma detection program then sequentially provides a set of features extracted from the training images, for each image in the training data the VPLD/ANN classifiers return a prediction (cancerous melanoma or non-cancerous typical/atypical nevi). The predictions are used to calculate the fitness which is returned to the GA.

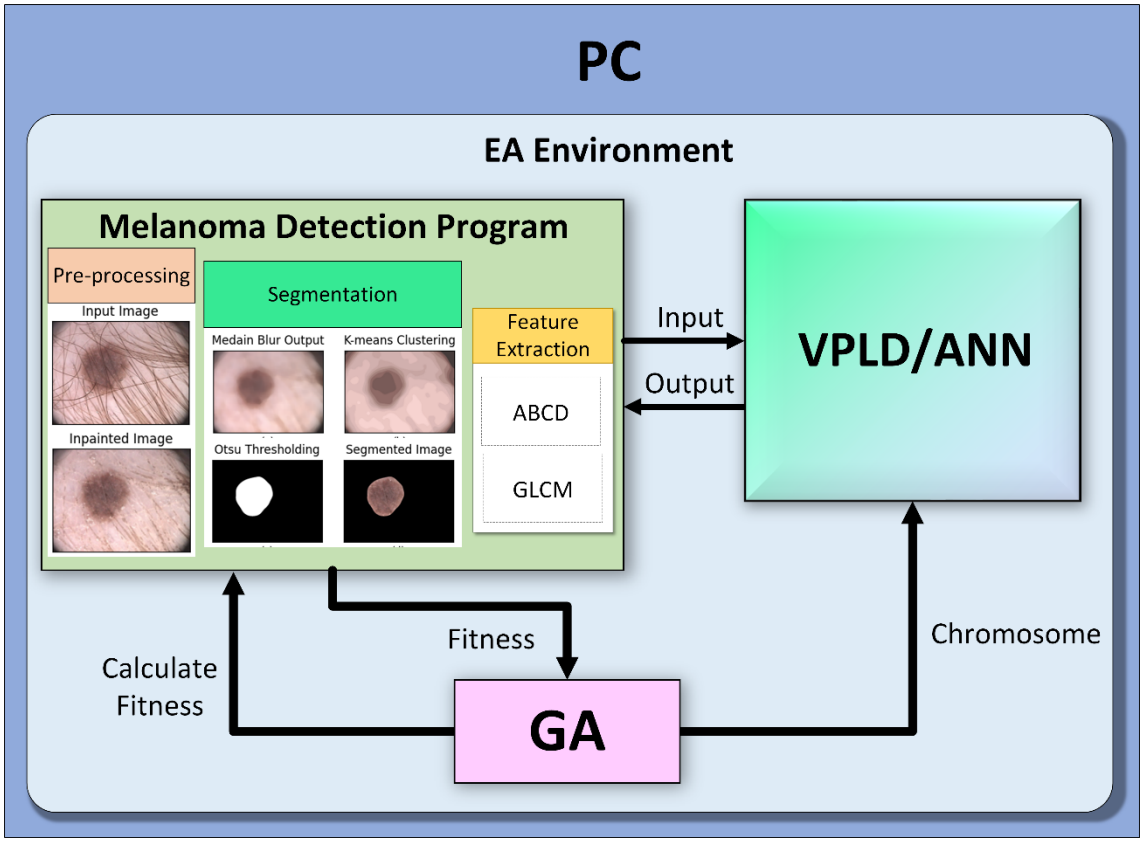

Fig. 1-6 The EA environment used for the melanoma detection to evolve the VPLD and ANN

For both the character recognition and melanoma classification problems, the VPLD and ANN are compared based on evolutionary efficiency, and the prediction accuracy of the evolved classifiers.

1.3.5 What are the important hyperparameters for the VPLD?

Hyperparameters are parameters set by the engineer/developer prior to optimisation of the VPLDs configuration bitstream. The hyperparameters for the VPLD are: 1) the number of layers; 2) the number of FABs; and 3) the data type of FAB. For each problem investigated, gait control of a hexapod, 2WD robot autonomous navigation, character recognition, and melanoma classification, a set of experiments are conducted to evaluate multiple architectures. These experiments are used to find the optimal architecture to use in the comparison between the VPLD and EHW, and the VPLD and ANN. Different hyperparameter experiments are conducted for each problem, but collectively the effects of altering, number of layers, number of FABs and types of FAB are investigated. The effect of these hyperparameters on the ability of the VPLD to control robots or classifier data correctly is investigated. The effect of the hyperparameters on both the evolutionary efficiency and the controller/classifier performance are analysed.

1.4. Publications

As a result of the research conducted in this thesis several articles have been accepted for publication in peer reviewed international conferences and Journals.

1.4.1 Journal Articles

- F. Borrett and M. Beckerleg, "A comparison of an evolvable hardware controller with an artificial neural network used for evolving the gait of a hexapod robot", *Genetic Programming and Evolvable Machines* **24**, 5 (2023)
    - Related chapters: 2 & 5.

1.4.2 Conference Papers

- F. Borrett and M. Beckerleg, "The Virtual Programmable Logic Device, a Novel Machine Learning Architecture," *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, pp. 1-8
    - Related chapters; 2, 3, & 5.

- F. Borrett and M. Beckerleg, "A Comparison of a Digital and Floating-Point Virtual Programmable Logic Device and an Artificial Neural Network Evolved for Robotic Navigation," *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, pp. 1-8
    - Related chapters; 2, 3, & 5.

1.5. Thesis Structure

This thesis is organised as follows:

Chapter 1: This chapter explains the research objectives for the thesis, and the motivation for the research question. The experiments and methods used to address these questions are also introduced.

Chapter 2: This is the literature review chapter, covering key areas of the current body of knowledge on EHW, specifically its implementation, and application in robotic control and pattern recognition. Additionally, literature on evolvable ANNs is also investigated, again covering applications in robotic control and pattern recognition. Comparisons between EHW and evolved ANNs are also discussed.

Chapter 3: In this chapter the methodology of the VPLD is described in detail, covering all areas including, the VPLD model, hyperparameters and operation, as well as its implementation in comparison to EHW. The optimization of the VPLD is also discussed, introducing how nature inspired optimization algorithms such as EAs can be used to optimize the VPLDs configuration bitstream for a given task.

Chapter 4: This chapter includes an overview of the robotic platforms used and the experimentation of the VPLD in the robotic control domain. Where the VPLD is validated in comparison to a EHW system for the gait control of a hexapod. The VPLD is then compared with an ANN for the hexapod problem and for controlling a 2WD mobile robot in autonomous navigation tasks. For each task a detailed description is provided for the VPLD, EHW and ANN architectures, as well as the parameters of the GA used to evolve the three architectures. A set of results for the experiments conducted are presented and discussed.

Chapter 5: This chapter covers the experimentation of the VPLD in the pattern recognition domain. The VPLD is compared with the ANN for a binary character recognition problem and for melanoma classification. For each task a detailed description is provided for the VPLD and ANN architectures, as well as the parameters of the GA used to evolve the architectures for pattern recognition. The results of the VPLD and ANN in the pattern recognition tasks are discussed, and a comparison is made on the performance of each architecture.

Chapter 6: In this chapter the effect of altering the VPLD hyperparameters on performance is investigated, this is done by comparing various VPLD architectures that are evaluated in the robotic control and pattern recognition problems.

Chapter 7: The research questions introduced in chapter 1 are discussed and conclusions are made based on the results of the experiments conducted in chapter 5 & 6. Potential areas of future research regarding both the VPLD architecture and its applications are examined.

Appendix A: Describes the different ANN architectures that are evaluated to find optimal architectures for each experiment. These optimal ANN architectures are then used in the main text of this thesis for comparison to the VPLD for applications in robotics and pattern recognition.

Appendix B: Includes links to the supplementary resources. The supplementary resources are videos demonstrating the evolved hexapod robotic controllers, both in simulation and on the real robot.

Chapter 2

Chapter 2: Literature Review

Evolutionary algorithms are a series of nature inspired optimization algorithms based on Charles Darwin's theories on natural evolution [19], specifically using the concepts of reproduction whereby new individuals inherit beneficial characteristics from previous generations, and survival of the fittest ensures individuals with the best genetic characteristics for the environment survive to reproduce. These concepts have been used to generate a number of metaheuristics for solving complex optimization tasks, such as genetic algorithms, evolutionary strategies, differential evolution and particle swarm optimization. In the field of evolvable hardware, evolutionary algorithms have been used to evolve digital and analogue electronic circuits for various tasks, ranging from digital adder and multiplier circuits to more complex architectures used for robotic control, pattern recognition and image processing. In this chapter a review of the literature on evolvable hardware and the evolutionary algorithms used to optimize them is conducted. Also, evolved ANNs are reviewed as in the literature as they are used as a point of comparison and a benchmark to EHW, particularly in the evolutionary robotics field. In this research evolved ANNs also serve as a benchmark comparison for the novel VPLD architecture for evolutionary robotics and pattern recognition tasks.

2.1. Evolvable Hardware and their application

Evolvable hardware is the study of using evolutionary algorithms to optimize digital and electronic circuits [1, 2]. EHW architectures are normally implemented and/or evolved directly on reconfigurable hardware such as Field Programmable Gate Arrays and Field Programmable Analogue Arrays. The applications of these EHW architectures varies from the optimization of adder and multiplier logic circuits, through to complex tasks in robotics, such as locomotion, autonomous navigation, manipulators and fault tolerance. In the field of pattern recognition, EHW have been developed for many applications including character recognition and facial recognition. A variety of EAs have been used for the optimization of EHW, primarily genetic algorithms and evolutionary strategies are used, however other methods such as DE and PSO have been investigated.

2.1.1 Reconfigurable Hardware Architectures

Reconfigurable hardware refers to electronic integrated circuits that feature reconfigurable routing and logic elements, along with additional peripherals. These characteristics make them adaptable to various applications. These architectures incorporate both digital and analog hardware, typically configured by a bitstream stored in a flash-based configuration memory device. The most common type of digital reconfigurable hardware is the FPGA.

Note: In the literature, the internal architectural components of field programmable gate arrays are described using terminology that varies between manufacturers and publications. Generic terms such as Configurable Logic Block (CLB) are commonly used to describe the fundamental logic resources within an FPGA. However, FPGA vendors often employ device-specific terminology in their documentation.

In this thesis, the term Logic Array Block is used in place of CLB when describing FPGA architectures. This terminology follows the conventions used by Altera in their device documentation and development tools, which form the basis of the evolvable hardware platforms referenced throughout this work. While the naming differs, LABs and CLBs represent functionally equivalent architectural concepts within FPGA fabric.

The use of vendor-specific terminology is intended to maintain consistency with the hardware platforms, documentation, and development environments employed in this research, and does not imply a conceptual deviation from standard FPGA architectures described elsewhere in the literature.

FPGA Architecture

FPGAs are a common type of programmable logic device. These devices are reconfigurable integrated circuits, making them highly adaptable for many applications. The FPGA architecture consist of a two-dimensional array of logic array blocks (LABs) with interconnections between LABs (Fig. 2-1).

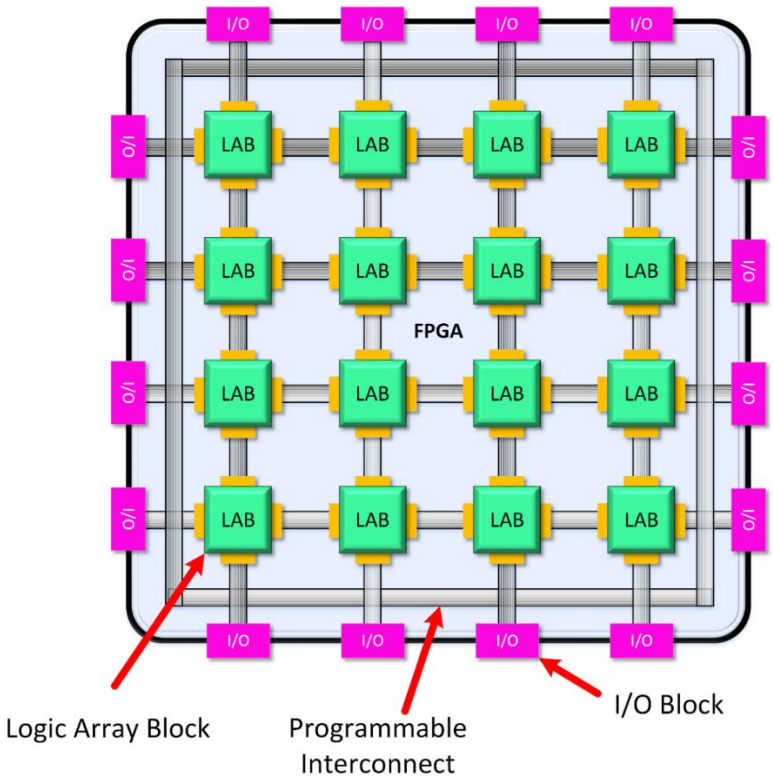

Fig. 2-1 Field Programmable Gate Array Architecture

Each LAB (Fig. 2-2) contains several interconnected logic elements (LE) which can be configured to perform a variety of logic functions. Logic elements typically have four inputs and one output, the four inputs are fed to a RAM-based lookup table which can be configured by the user to perform any combinatorial logic function on the four inputs (i.e. A&B&C&D). Logic elements also typically contain a programmable register (Fig. 2-3).

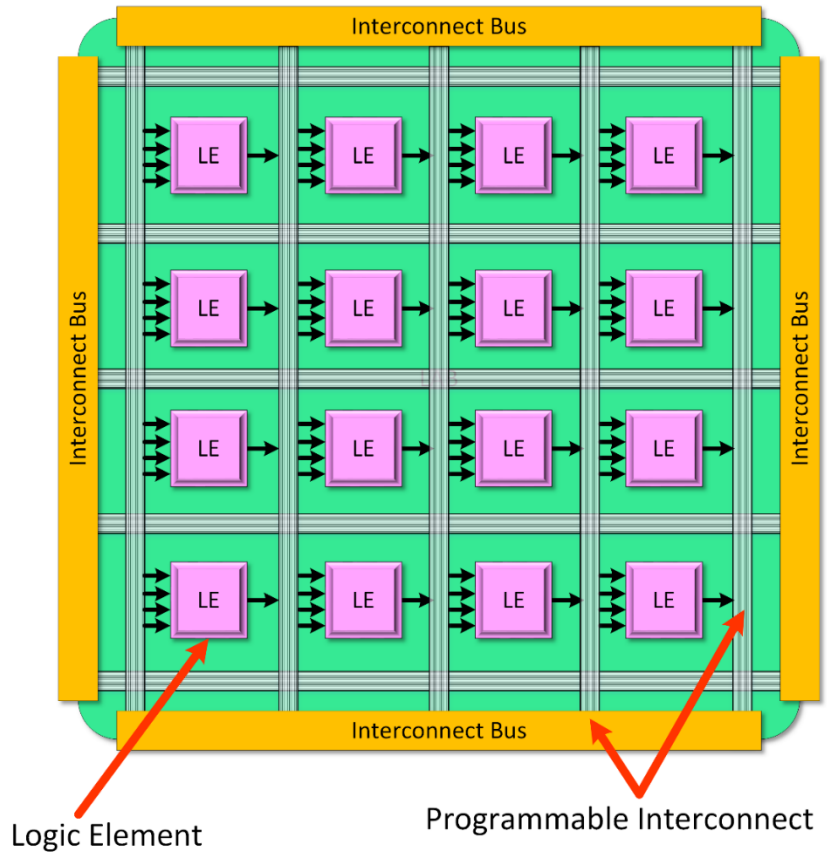

Fig. 2-2 Simplified Logic Array Block Architecture used in Altera FPGAs

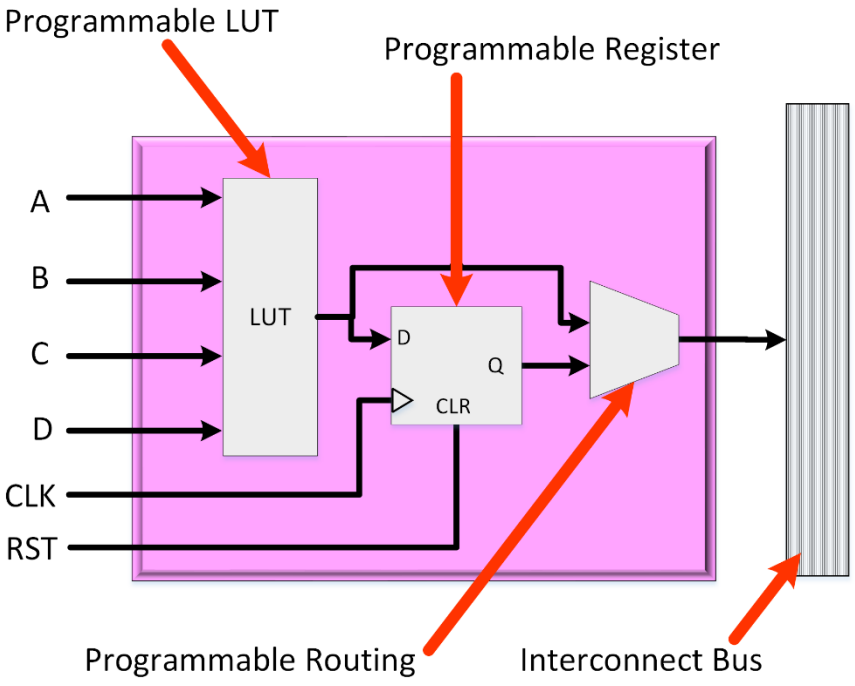

Fig. 2-3 Basic Logic Element Architecture with programmable LUT, register and routing

An FPGA device is programmed using a hardware description language such as Verilog and VHDL. Manufactures like Altera and Xilinx have their own Integrated Development Environments (IDE)s for the programming and compilation of a design to be implemented on their respective FPGA devices. An IDE such as Altera's Quartus is required for compilation of the HDL design to generate a configuration bitstream, which is used to program the FPGA device for the required operations. The routing and function of the LABs/LEs can be configured to perform simple multiplier and adder operation through to complex logic circuits used for display drivers or high-speed network communication. To aid in these complex applications modern FPGAs make use of multipliers, memory blocks, digital signal processing blocks as well as embedded hard-core processors. Having such features with a reconfigurable architecture makes FPGAs extremely useful in embedded and control systems for a variety of fields, including telecommunications, aerospace, and robotics.

2.1.2 Digital Evolvable Hardware

Digital EHW is the study of the optimization of digital circuits using evolutionary algorithms. 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 such as Altera's Quartus is used to convert the digital circuit design into a configuration bitstream 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 produce a variety of circuits from adders[3] and multipliers through to robotic controllers[10, 11, 13, 22] and classifiers[15] for pattern recognition.

The major difficulties of evolving a FPGA 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, directly evolving the CBS 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 [4, 5]. 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 [23, 24]; b) genetic programming of HDLs, evolving the hardware description language itself relying on the compiler to prevent destructive architectures [25]; and c) virtual FPGAs designed in HDL and implemented on FPGAs. A variety of virtual FPGA architectures have been proposed however most consists of a two-dimensional architecture of units that perform configurable logic functions.

The S-block (Fig. 2-4) is one such virtual FPGA architecture originally introduced by Haddow and Tufte [6, 7]. Each S-block contains a configurable LUT and data register, the S-block has four inputs and outputs (north, south, east, and west). This architecture can be synthesised on the FPGA fabric and is configured using a bitstream, the bitstream can be evolved to optimize the functions selected by the LUT and the routing between blocks. A destructive CBS is not possible as only output to input connections are possible.

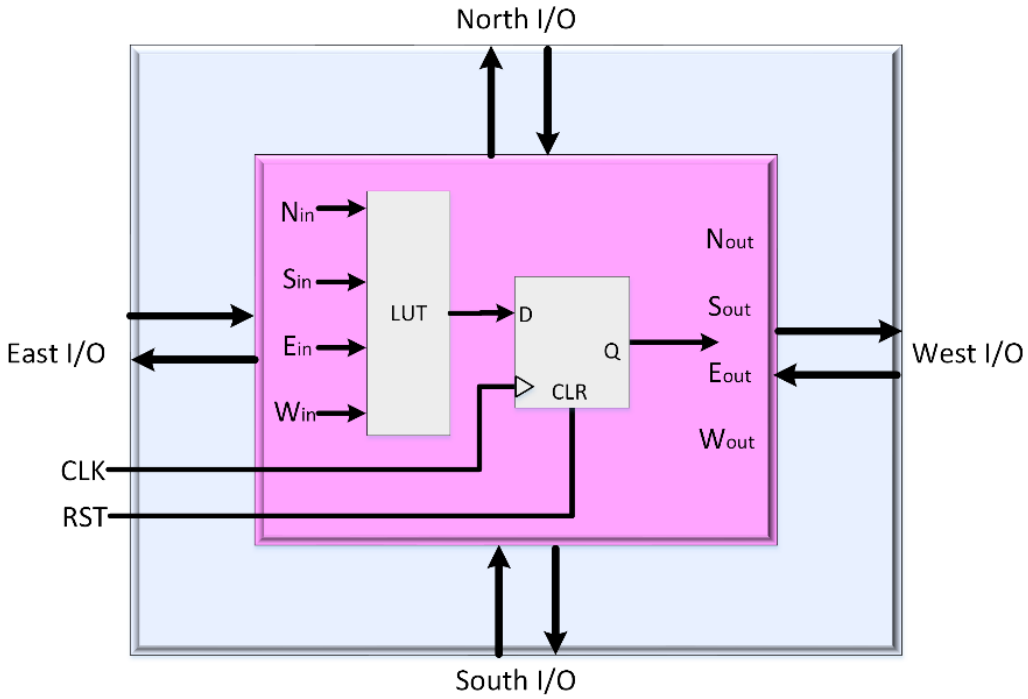

Fig. 2-4 S-Block architecture introduced by Haddow and Tufte

The limitation of the S-block was that it had a fine-grained architecture making it difficult to evolve complex circuits. Alternative to the S-Block approach, are virtual FPGAs [8, 9] which use logic units equivalent to the FPGA LAB, each of these logic units contains a multiplexer which selects inputs determining the interconnections between logic units, and logic elements. The logic elements contain a LUT with complex logic functions giving these virtual FPGAs a course-grained architecture which is more suitable for evolution. These logic units are configured by a configuration bitstream. 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.

2.1.3 Examples in Robotics

In the domain of evolutionary robotics, EHW architectures have been designed for many applications including autonomous navigation [10, 11], ball-beam system [12, 13], locomotion [14], and fault tolerance [14, 17, 18].

Beckerleg and Collins [12] evolved a cartesian EHW architecture on a FPGA to control a ball-beam system. A hardware GA was used to evolve the configuration bitstream of the digital EHW, this GA used only mutation and no crossover operation. It was found the EHW architecture could be evolved successfully to balance the ball in the beam for over five minutes. The initial evolution was slow until the EHW controllers could balance the ball for approximately ten to twenty seconds at which point the fitness (balance time) rapidly increased to beyond the goal of five minutes. On average the EHW was evolved in under 40,000 generations.

Beckerleg, Matulich and Wong [10], investigated a digital EHW robotic controller for a 2WD mobile robot. The EHW system was compared with an ANN and a lookup table. The three controllers evolved using a GA for obstacle avoidance and light following tasks, and combined light following while avoiding obstacles. The EHW used for the light following while avoiding obstacle task, was a four-layer architecture implemented on a FPGA. The outputs for each motor were forward, reverse and stop. It was found that the EHW and ANN controllers had a comparable evolutionary performance, with the combined obstacle avoidance and light following problem evolved in 700 generations by the EHW and 850 generations by the ANN controller. However, the ANN reached a slightly higher fitness level.

In a similar robotic navigation task, Tyrrell et al.[11] evolved an obstacle avoidance controller for the 2WD Khepera robot. The robot had eight proximity sensors for inputs and two binary outputs allowing the robot to move forward, backward, left, and right. The fitness was based on the distance and time travelled without a collision. Fault tolerance was investigated using faulty sensors, and it was found that some sensors had minimal or no impact on performance allowing the evolutionary process to evolve more quickly under such fault conditions.

Some researchers have chosen alternatively to use FPAAs; like their digital counterpart the FPGA these devices are reconfigurable but implement analogue hardware. Berenson et al[14], used an FPAA EHW approach with artificial neural networks approximated as analogue hardware. Two ANN controllers were developed and evolved using a GA, one for the locomotion of a biped robot and a second used in restoring the ability of a quadruped robot to move after several faults were introduced. For the quadruped experiments, two fault scenarios were investigated. The first scenario looked at achieving motion with only a single leg left controllable, the others were locked in a fully extended position. The second scenario involved simulating partially removing the shank of one of the legs by attaching a weight to where the shank connects. It was found that using their chosen methods the quadruped was not able to recover in extreme friction conditions. But on a plywood surface it was able to achieve some forward motion. It was noted in their work that the online evolution is time intensive having to test each candidate solution each generation and required resetting of the robots after each test.

Combining EWH and redundancy, Hereford [18] developed a fault tolerance strategy using FPAAs to recover from sensor failures. In his experiments, three temperature sensors were connected to the inputs of a FPAA, the multiple sensor inputs are combined by the FPAA for a single output value. When faults are detected using a Kalman filter such as open or short circuits, the FPAA had to be adapted using a GA to accurately produce the correct value. A standard GA was used with roulette wheel selection, 70% crossover probability and a 6.25% mutation rate. During the experimentation the reconfiguration allowed the system to recover and output values within 2% of the correct value for all fault cases.

2.1.4 Examples in Pattern Recognition

In the pattern recognition domain evolvable hardware has been used as a classifier for a variety of tasks. In the related field of image processing, evolvable hardware has also been investigated for tasks such as noise removal from corrupt images [16].

EHW architectures have been evolved for classification of sonar signals as a means of target detection by Garnica, Glette, and Torrsen [15]. In this study a comparison was made between three EHW implementations using a Xilinx XC7-Z020 FPGA, 1) a full hardware solution on FPGA fabric, 2) EHW on FPGA fabric and EA implementation on soft-core processor, and 3) EHW on FPGA fabric and EA implementation on an embedded ARM processor.

The configuration of the EHW is small only requiring 90-bits, this configuration was evolved using a (1+8) evolutionary strategy. The approach using full hardware and the approach using the ARM processor (on max clock speed 667MHz) could run the same number of generations in a comparable time, but the approach using the softcore processor was slower due to the lower clock speed. Note: this study did not assess the required generations to evolve an optimal classifier and the level of accuracy at the end of the evolution. A previous study by Glette and Kaufmann [26] also investigated the same sonar classification problem using EHW classifiers, and in this work an accuracy of 86.3% was achieved. However, this was not the goal of the research so it is noted that better accuracy may have been achieved with alternative architectures, instead the paper was focused on reducing the resource utilization of the EHW by using lookup table partial reconfiguration.

For the task of character recognition using EHW as is investigated in this thesis, Wang, Piao and Lee[9] designed a four layer EHW cartesian architecture for the classification of 5x6 pixel images of characters. Specifically, the architecture was evolved using a (1+4) evolutionary strategy with a self-adaptive mutation rate, to classify the characters A to P. The 4-layer architecture consisted of 64 function blocks, 16 per layer. To classify the characters correctly the EHW had to evolve to give a high signal (logic 1) for a given character, the other 15 outputs had to be low (logic 0). The complete evolutionary process was implemented on a FPGA SoC. The EHW architecture successfully evolved to classify all the characters.

A second approach to the character recognition problem was investigated by Wang et al.[27] where multiple EHW circuits acted in parallel with the outputs of each combined via a counter meaning the output of the EHW architecture was no longer binary but an integer value ranging from 0 to n, where n is the number of EHW circuits used in parallel. Classification of the characters A-P was then determined by the comparison of the magnitude of each output. It was concluded that the new architecture had an improved accuracy over the original with the presence of input noise (1-5 bits altered in the image).

2.2. Challenges

Although EHW architectures have been successfully evolved for robotics and pattern recognition applications, there are number of challenges with the EHW and their implementation. In particular 1) a high degree of complexity in the system design; 2) high cost with the need of expensive development boards, as well as custom hardware for robotics applications; 3) limited resources of FPGA and FPAA devices which reduces scalability; and 4) low processor performance on FPGA SoCs, which limits the robotic simulation complexity.

2.2.1 System Design Complexity and High Cost

The EHW system design is very complex, requiring multiple elements for both the software and hardware components. The software components that run on a PC or the FPGA SoC processor are: 1) the evolutionary algorithm; and 2) the robotic simulation or data processing for pattern recognition applications. Other software components may also be needed to transfer data and trial configurations between the SoC processor and the FPGA fabric. The hardware components are the EHW IP itself and the system design which may include embedded ARM processors or softcore processors, along with peripheral components such as parallel ports, communication interfaces, memory, phase locked loops etc to be implemented on the FPGA fabric. Having many components in the complete evolution system spread across different platforms poses a challenge when a researcher wants to make any alterations to the design, it may require changes to the system at multiple levels. The complexity also poses a risk of introducing errors into the system with multiple different components all required to carry out a single operation, including: a) the custom IP defining the EHW; b) the program interface the processor and FPGA; c) the GA code managing the chromosomes of the individuals; and d) the system architecture.

In the investigation of EHW for the control of a 2WD mobile robot described above [10], the EA implementation has a high degree of complexity. The system required a PC to program the FPGA device and to run the robotic simulation and GA. The simulation I/O and trial chromosomes were transferred to the FPGA from the PC using a RS232 serial link to a NIOS softcore processor running on the FPGA. The NIOS processor then configures the EHW and gets the output control signal to send back to the PC to drive the simulation (Fig. 2-5). This research utilised an Altera DE2-115 which has a cost of $423USD (for academic institutions), and a custom robotic platform utilizing a FPGA IC was also required.

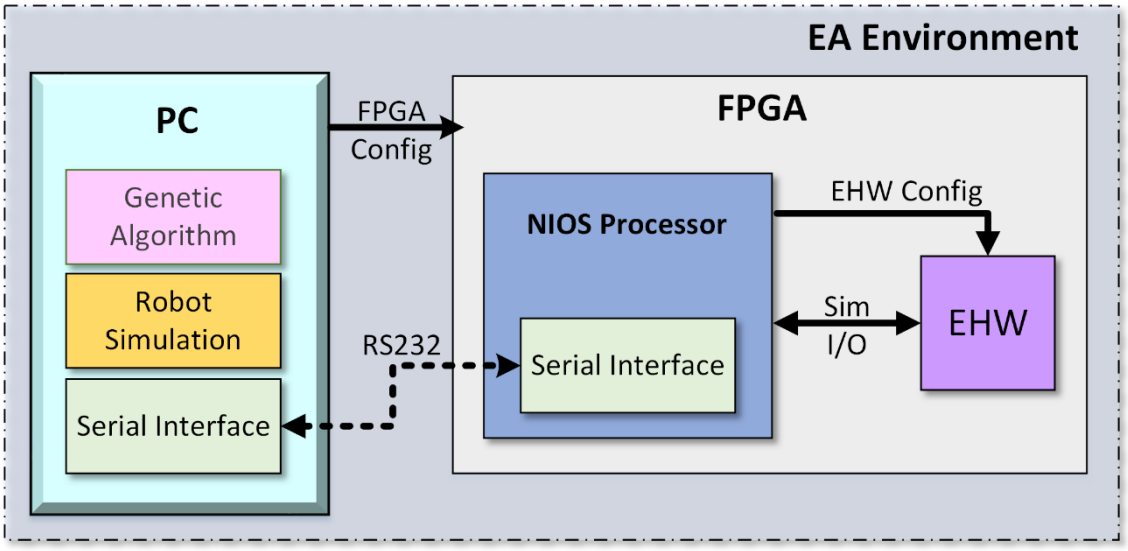

Fig. 2-5 EA environment used by Beckerleg, Matulich and Wong [10] to evolve EHW controllers for a 2WD mobile robot

In the sonar pattern recognition problem[15], has a system design of similar complexity to the 2WD robotic control[10]. However, they have removed the need of a PC to FPGA interface by having the software managing the training data and the EA run on an processor apart of the FPGA SoC. As stated previously separate implementations utilizing an embedded ARM processor and also a softcore processor running on the FPGA fabric were investigated (Fig. 2-6). The FPGA board used in this research was the Xilinx ZedBoard Zynq-7000 which costs $589USD.

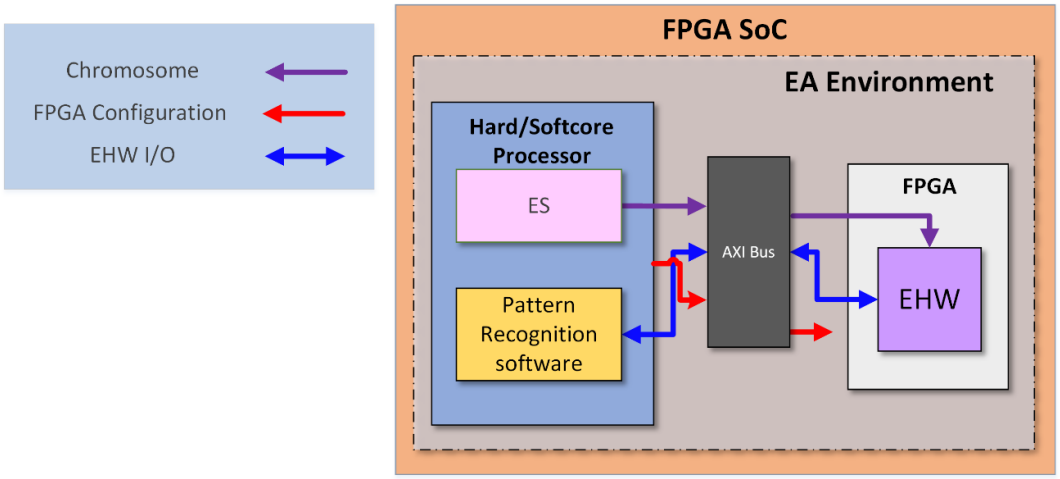

Fig. 2-6 EA environment used by Garnica, Glette, and Torrsen [15] to evolve EHW classifiers on FPGA SoCs

2.2.2 Limited Resources and Low Processor Performance

Another challenge is in the resources required to implement the EHW. Analogue EHW utilising FPAAs have been limited by the low resource count of such devices, often requiring multiple FPAAs connected together[28]. Even with this approach the architectures implemented have been relatively small. For larger architectures to be implemented in the hope of solving more complex tasks, custom FPAA platforms would be required. Because of these limitations the research into and success of analogue EHW has been restricted compared to digital EHW. The digital EHW utilising FPGAs can also have issues with limited resources. When using the EHW IP combined with softcore processors and other IP such as communication interfaces, the system quickly reaches the max number of logic elements. However a larger FPGA with more logic resources can be purchased, but FPGA SoCs with high logic resources and the necessary peripherals are expensive.

In regards to processor performance, it is shown in the literature that embedded ARM processor as part of the FPGA can execute the EA faster than softcore processors synthesized on the FPGA fabric, this is because the ARM processor have a higher clock speed[15]. However, these processors are not capable of running efficiently (if at all) modern robotic simulations which have become more complex and have a high computational load but have a lower simulation to reality gap. As in the case of the EHW used for 2WD robotic control[10], the simulation can run on a PC with the GA, and transfer simulation I/O and trial configurations to a FPGA SoC but the communication is a bottleneck in the system increasing the time to evolve. Apposed to an ANN implementation for example, where the simulation, EA and the ANN itself operate in the same environment which doesn't require communication between different devices.

2.3. Contribution of this Thesis

The primary contribution of this research is the Virtual Programmable Logic Device, inspired by both the PLD and EHW architectures implemented on FPGA devices. The VPLD is based on the architecture of a PLD which uses programmable function elements and routing between the function elements. However instead of using hardwired digital circuits, the VPLD is the software equivalent of the PLD written in a standard programming language that can be executed on a CPU. The author has not found references of such an architecture in the current body of knowledge.

The VPLD has a simpler system design than EHW, the complete VPLD evolution process can be implemented in a single programming environment to run on a processor akin to an equivalent implementation for the optimisation of ANN. This means multiple devices and development tools are not needed, and the VPLD can be implemented with any programming language and the associated IDE. This implementation has several benefits including, 1) it is faster to develop and compile designs, and 2) it is simpler to debug and refactor designs. These factors will make the process of conducting research using the VPLD easier and faster than when using EHW.

The VPLD can be optimised on standard desktop PCs and laptops which have large amounts of memory and multicore processors with clock speeds much higher than the processors on the FPGA SoCs. Meaning the VPLD does not have the same resource limitations of EHW. The VPLD will be faster to evolve for modern robotics applications as the VPLD is running in the same native environment as the simulation on a PC. These simulations cannot run on FPGA SoCs efficiently, and the alternative PC-FPGA system is bottlenecked by the communication between the PC and the FPGA device.

The VPLD has a reduced cost compared to the EHW, as it doesn't require expensive FPGA development boards, instead devices like the Raspberry Pi can be utilised which can be brought for $35USD. Also, in robotics research, expensive platforms with custom FPGA hardware are not required. Instead off the shelf robotics platforms which often utilize the Raspberry Pi as the main controller can be brought for significantly less cost.

2.4. Other Relevant Work

This section describes other research areas that are relevant to this thesis including, 1) the optimisation of ANNs using evolutionary algorithms, and 2) the evolutionary algorithms used for the optimisation of EHW. The research areas discussed in this section do not directly concern the primary research contribution being the VPLD architecture and the challenges outlined in the EHW field which the VPLD aims to address. However, as the evolved ANN is used as a benchmark for the VPLD performance, and an evolutionary algorithm is used to optimize the configuration of the three architectures (ANN, VPLD, and EHW), a review of the literature in these domains is relevant to this thesis.

2.4.1 Evolved Artificial Neural Networks and their application

ANNs aim to replicate biological neural networks using a layered architecture of interconnected neurons (Fig. 2-7) in software. Each neuron has a series of weighted inputs, and the sum of these weighted inputs plus a bias is fed into an activation function such as the sigmoid or rectified linear functions (Fig. 2-8). The activation function will determine the final output of the neuron. A familiar example of ANNs can be seen in modern smart phones where it is used in facial recognition to unlock the phone [29, 30].

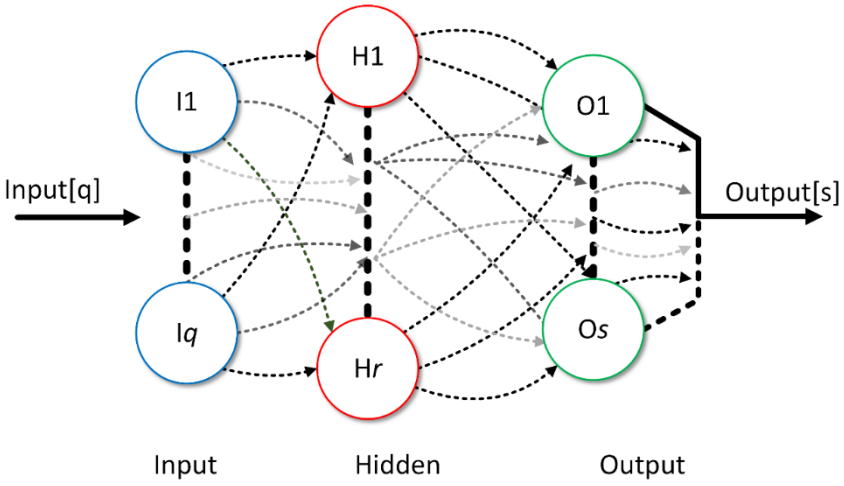

Fig. 2-7 Feedforward fully connected ANN

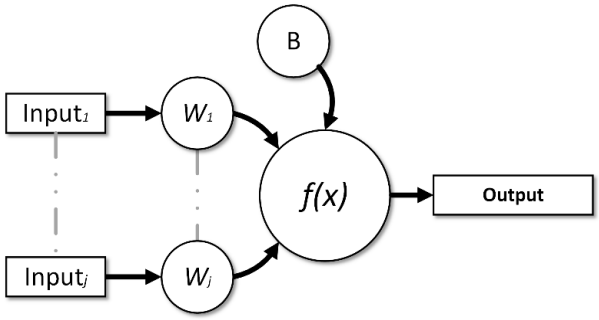

Fig. 2-8 ANN neuron with weighted inputs, bias and activation function

These networks of neurons are typically trained (weights and bias value optimisation) using a gradient based optimisation algorithm. However, other non-gradient approaches can be used, including nature inspired methods like evolutionary algorithms. The evolutionary algorithms have been used for 1) optimisation of ANN hyperparameters[31-34], and 2) the optimisation of the ANNs weights and biases for a given application[32], such as robotic control and pattern recognition. This thesis is concerned with the latter optimisation of the ANNs configuration (weights and biases), as the ANN is used to benchmark the performance of the VPLD. The VPLD and ANN configurations will be optimised using the same EA for a fair comparison.

The evolution of ANNs weights and biases, and applications of the evolved ANNs in robotics and pattern recognition are described below.

Evolution of Artificial Neural Networks

Evolutionary algorithms including the GA, ES, and PSO have been used to optimise the weights and bias of ANNs for various applications in the robotics and pattern recognition domains. The interest of the ML community in the use of evolutionary algorithms for the training of ANNs is twofold, 1) evolutionary algorithms can be used for non-differentiable problems, and 2) evolutionary algorithms such as the GA have good global search characteristics compared to gradient based methods.

Examples in Robotics

In robotics ANNs have been successfully evolved for tasks such as controlling mobile robots [35-39], solving the inverse kinematics of robotic manipulators [40], and hexapod locomotion control[41, 42].

Beckerleg, Matulich and Wong [10], investigated a comparison of three different robotic controllers for a 2WD mobile robot. One of the controllers was a feed forward ANN, a GA was used to evolve the weights and biases of the ANN for different navigation tasks. It was found the ANN took more generations to evolve compared to the EHW, but the ANN reached a slightly higher fitness level.

Juang, Chang and Hsiao [43] 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 and a Webots simulation of the robot was successfully used to optimize the neural

network to produce a gait where the hexapod walked forward in a straight line. The evolved neural network could be transferred to control the real robot, however it is noted the distance travelled by the real robot was less than that observed in the simulation. The reduced distance was caused by the slower execution of the neural network on the embedded hardware.

As an alternative to reinforcement learning, Salimans et al[44] investigated the use of evolutionary strategies for the optimisation of the weights and biases of a neural network. The neural network was a feed forward multilayer fully connected ANN with two hidden layers each with 64 neurons, all layers use the hyperbolic tangent activation function. The ANN was evolved using ES for multiple tasks in the OpenAI Gym[45] which utilises the MuJoCo simulator[46]. The tasks evolved for include, both single and double inverted pendulums, as well as controlling the gait of a humanoid in 3D cartesian space. The combination of the ES and ANN was able to solve the humanoid problem in ten minutes, which was faster than the traditional gradient based reinforcement learning approach.

Examples in Pattern Recognition

ANNs have been investigated extensively in the pattern recognition domain with great success, and have been used for multiple pattern recognition applications, including medical diagnosis and imaging. Medical diagnosis is an important application of ANNs. Specifically neural networks have been used for applications such as lung disease classification from x-rays[47], other examples include the use of photos, CT scans, and MRI scans for detection of various cancers[48, 49]. Another useful application of ANNs in this domain is segmentation, architectures such as U-net and its many variants are used to segment tumours from medical images for example[50, 51], which aids a medical practitioners diagnosis.

One such task in this domain which is investigated in this thesis is skin lesion classification. Chakraborty et al [52] investigated the classification of three skin lesions, angioma, basal cell carcinoma, and lentigo simplex using a ANN. The ANNs input was a set of 13 extracted features, including Harlick features extracted from a Grey Level Co-Occurrence Matrix [53]. Genetic algorithms are compared with PSO for the optimization of the ANNs weights and biases. It was shown that a non-dominated sorting GA, achieved better accuracy than the PSO implementation, at approximately 88% and 82% respectively.

For the classification of melanoma as benign or cancerous, Aswin, Jaleel and Salim [54] evolved the weights and biases of a fully connected feedforward ANN using the sigmoid activation function. A set of features were extracted using images from a dermatoscope. The extracted features were four Harlick features from a GLCM, and three colour features being the average red, green and blue values. Prior to feature extraction the images underwent preprocessing and segmentation stages. The preprocessing involved filtering to reduce noise, and a dull razor algorithm to remove hair from the image. For segmentation of the skin lesions the Otsu Colour Threshold method is used. It was shown that the GA could evolve the ANN to reach an accuracy of 88% for the binary classification of melanoma.

2.4.2 Evolutionary Algorithms used for Hardware Evolution

Evolutionary algorithms are a series of nature inspired optimization algorithms based on Charles Darwin's theories on natural evolution[19], specifically using the concepts of reproduction whereby new individuals inherit beneficial characteristics from previous generations, and survival of the fittest ensures individuals with the best genetic characteristics for the environment survive to reproduce. These concepts have been used to generate a number of metaheuristics for solving complex optimization tasks, such as ES[55-57], DE[58], GAs[59-61] and PSO[62]. Evolutionary algorithms which have been used to optimize EHW for a variety of problems, generally use an iterative approach of fitness assessment, selection of the fittest individuals, and reproduction. The EA is run until the maximum fitness in the population converges to an optimal level. Other than for the optimization of EHW, EAs have been used in many different domains for optimization problems, including for tasks such as organising airline flight plans, and shipping manifests to maximize efficiency. EAs have also been used to optimize ANNs for robotics and pattern recognition tasks.

Fitness

Fitness is an important component of every evolutionary algorithm as it determines which individuals are optimal or closest to optimal in the population. The fitness of an individual is evaluated using a fitness function. Each fitness function has an associated fitness landscape, which is the entire solution space containing the fitness for all the possible chromosome configurations (Fig. 2-9). Depending on the specific problem and fitness function an EA maybe searching for either a minima or maxima as the optimal fitness. For instance, in the traveling salesman problem where the fitness maybe set to

minimize the path of the salesman, the EAs goal would be to find the global minima (point A). On the other hand, for an engineering problem such as the generative design of a heat sink, the EA may be searching for the global maxima which is the configuration of the heat sink with the highest efficiency (point B).

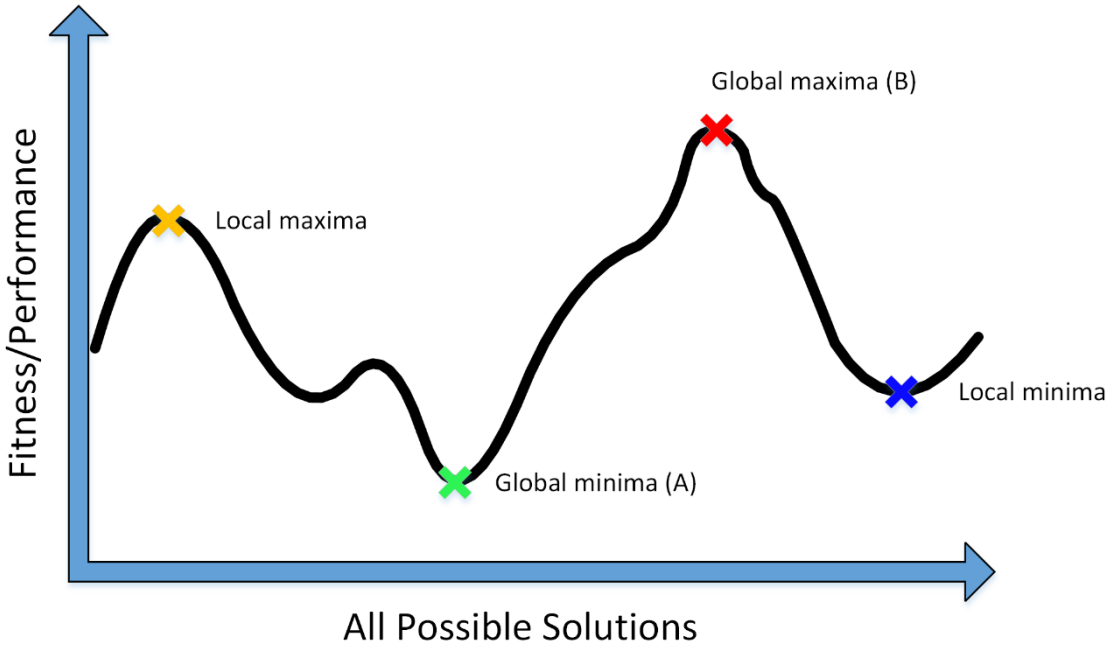

Fig. 2-9 Fitness Landscape, showing local and global maxima/minima

The success of the EA in searching the fitness landscape depends on a variety of factors including, the size of the population, mutation and crossover operators. These factors will influence the diversity in the population, high genome diversity particular in early generations is important as it aids in global search of the fitness landscape, however diversity will generally decrease as the EA progress and the population converges to an optimal solution. Which is why the mutation rate needs to be high enough to not be overly destructive of offspring chromosomes, but also aid in maintaining diversity so the population can break away from a local optima to move closer to the best global solution.

For the fitness assessment in robotic control application, typically trial controller configurations are assessed in a simulation of the robot, as the robot can be assessed in faster than real time, meaning a population of 50 or 100 robots can be assessed for many generations faster in simulation than trying to assess fitness of a population on the real robot. However, simulations suffer from the sim to reality gap, but modern simulators have greatly improved in this area.

Examples in the literature on the fitness assessment of robotic controllers on the real robot include Heijnen, Howard and Kottege's[63] work on a testbed for the evolution of hexapod controllers. The paper's primary objective was to investigate the testbed and how it could be used to evolve the controllers on the real robot. The evolution was done in a two-stage process: 1) using a DE on a population of possible solutions; and 2) selecting a single parent from the stage 1 population based on the mission criteria and using the (1+1) ES to reach the final solution. They were able to show that 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. However, the evolution process took nine hours to complete. In simulation equivalent controllers could have been evolved in the realm of seconds or in the worst-case minutes.

In the pattern recognition domain fitness assessment is relatively straight forward in comparison to robotics, whereby classifiers can make a series of predictions on a set of training data, and the fitness can simply be the accuracy or error of the classifier. This was seen in the examples of EHW classifier evolution discussed[9, 26].

Genetic Algorithms

The GA is one of the many metaheuristics to come from Darwin's theories on natural evolution. Such metaheuristics were originally introduced in the 1960s but the GA was later introduced and made popular in the 1970's from research conducted by John Holland [59-61]. The GA employs the evolutionary concepts of reproduction, crossover, mutation, and natural selection to manipulate a population of individuals. GAs are an iterative process of fitness calculations followed by selection and reproduction; iteration stops when the population converges towards the optimal solution for the given fitness landscape (Fig. 2-10). The reproduction stage of the GA involves the crossover of parent chromosomes to produce offspring, which ideally have taken beneficial genes from each parent to result in a new individual with a higher fitness than those in the previous generation. Mutation operators are also used during the reproduction stage to, 1) introduce new genes to the population, and 2) maintain diversity of the population, which is important for the success of the GA in escaping local optima to reach the global optima. Genetic algorithms have been used for a variety of applications, one application is the GA can be used to optimise EHW configurations for robotic controller, and pattern recognition tasks. Examples discussed were robotic controllers for mobile robot navigation[10], and a ball-balancing system[12].

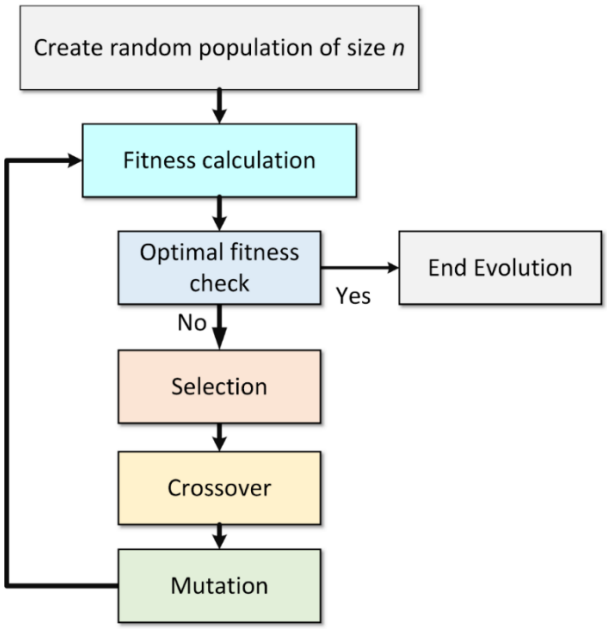

Fig. 2-10 Standard Genetic Algorithm

Crossover Operators

Crossover operators are the methods used to combine parent chromosomes to generate new child chromosomes. Various techniques have been used to generate offspring, some standard operators are described below, outside of the standard methods several approaches have been used including some problem specific operators [64].

Single point crossover (Fig. 2-11) selects a point randomly to slice each parents' chromosome. The first offspring chromosomes will have the genes from the left side of crossover point of parent α, combined with the right side of the crossover point of parent γ. The second offspring will have the opposite combination of the parent genes. The disadvantage of using a single point crossover is that useful genes at either end of the chromosome in one parent can't all be passed to the offspring, as the child will either have the beginning or the end of the parent's chromosome. However, single point benefits are that 1) it's simple to implement, and 2) larger sections of the parents genes are maintained, which is useful at the end of evolution where other operators are more destructive of the parent chromosomes.

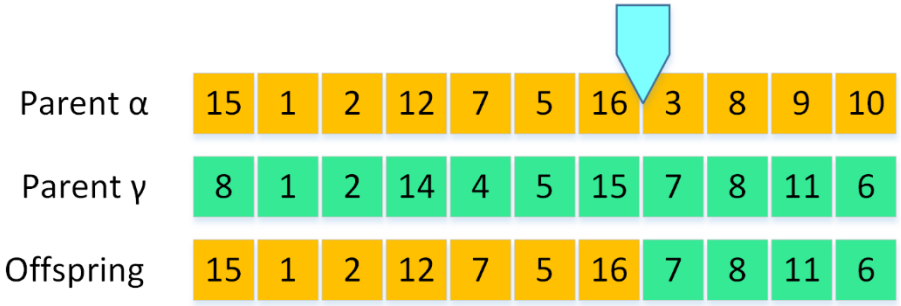

Fig. 2-11 Single point crossover operator example

Multi-point crossover (Fig. 2-12), also known as n-point or k-point crossover, uses a method similar to single-point crossover but with n>1 crossover points. As with single point the crossover is completely random, and the offspring chromosome is a combination of two parents' chromosomes. Multi-point can result in better exploration of the search space compared to single point. However, this operator can be disadvantaged with a higher number of crossover points, because the higher the crossover points the more destructive the operation is. Meaning parts of the chromosome (specifically combinations of genes) which may be optimally configured can be lost during the crossover.

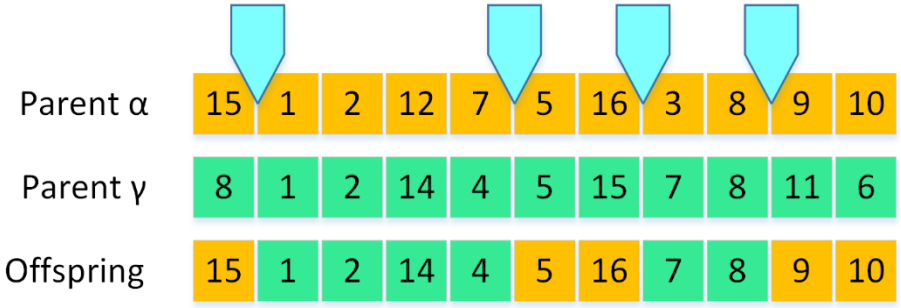

Fig. 2-12 Multi-point crossover operator example

Uniform crossover (Fig. 2-13) is another common method. The uniform crossover operator selects genes from each parent randomly, but each parents' genes have a 50% chance of being passed to the offspring chromosome. This method like multi-point has good exploration capabilities but also like multi-point can be destructive. However, this is not always the case as the child chromosome may end up with 95% of good genes intact from one parent, and only 5% of genes from the second parent for example.

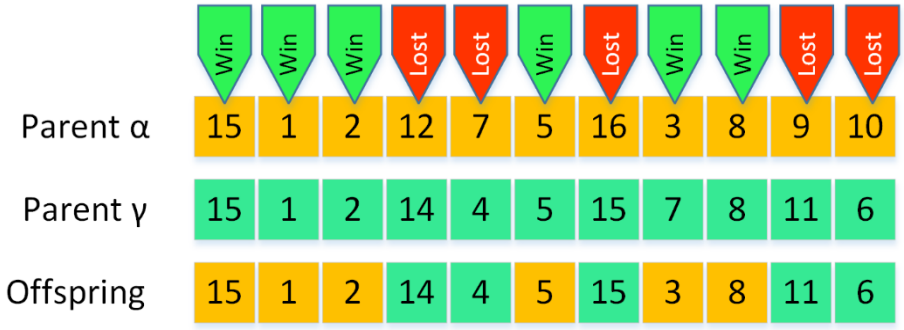

Fig. 2-13 Uniform crossover operator example

Average crossover (Fig. 2-14) operator generates children using the average of both parents' chromosome. This crossover method is only suitable for chromosomes, that use integer or floating point values to express the genes.

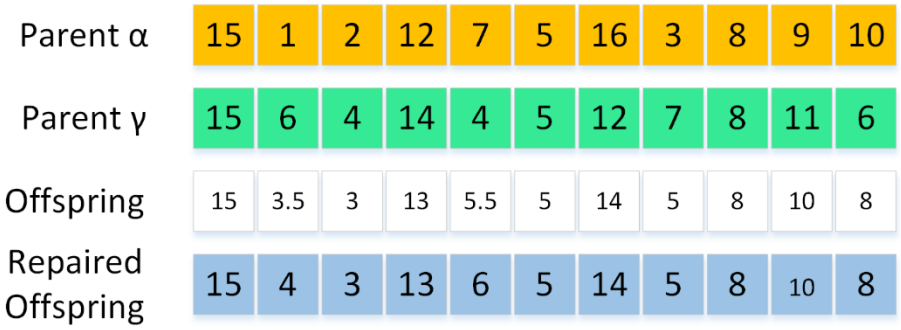

Fig. 2-14 Average crossover operator example

Mutation Operators

After a new offspring chromosome is generated, it has a probability of being mutated. In nature the probability of mutation is very low, the human genome for example has an estimated probability of approximately 10^{-8} of being mutated[65]. In GAs a low mutation rate is typically used in the range of 0.1% to 3%. A low mutation rate aids the search for an optimal solution, an excessive mutation rate (5% or greater) would be overly destructive of the chromosome, resulting in useful genes from past parents being lost. Because of this the probability at the end of evolution that mutation improves the fitness of a given chromosome is low. Like crossover there are a few standard mutation operators[66].

Insertion mutation (Fig. 2-15) selects one or multiple points in the chromosome being mutated and replaces them with random genes. If the operator is setup to mutate more than one point, the number of points is kept very low, not higher than a few percent.

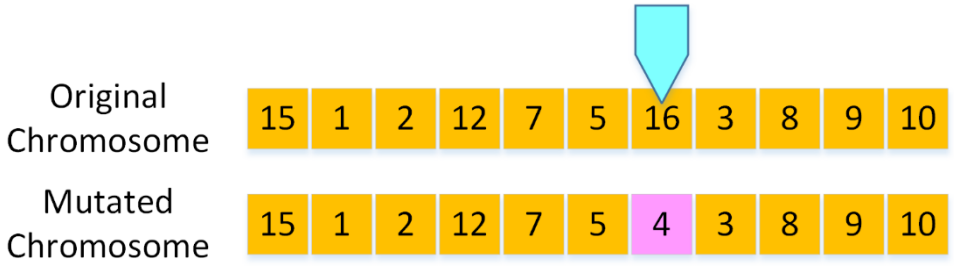

Fig. 2-15 Insertion mutation operator example

Inversion mutation (Fig. 2-16) as the name suggest involves an inversion of the genes within in the chromosome. The inversion operation requires two points in the chromosome be selected randomly, the genes within the two points are then inverted. A similar form of mutation does occur naturally in biological chromosomes.

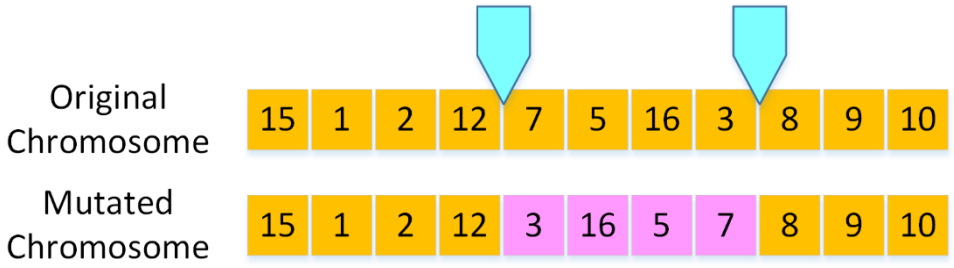

Fig. 2-16 Inversion mutation operator example

Displacement mutation (Fig. 2-17) works by first selecting two random points like the inversion operator, but instead of inverting them, the selected region of the chromosome is moved (displaced) to a new point in the chromosome, the genes in that position are shifted to the right to make room for the displaced genes.

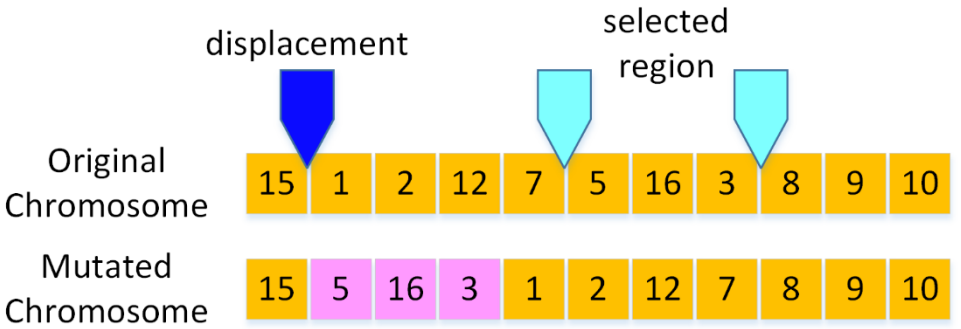

Fig. 2-17 Displacement mutation operator example

Exchange mutation (Fig. 2-18) swaps genes in the chromosome being mutated. To perform the operation two random genes are selected to exchange places in the chromosome. Multiple genes can be exchange but similar to insertion the number of operations is kept very low.

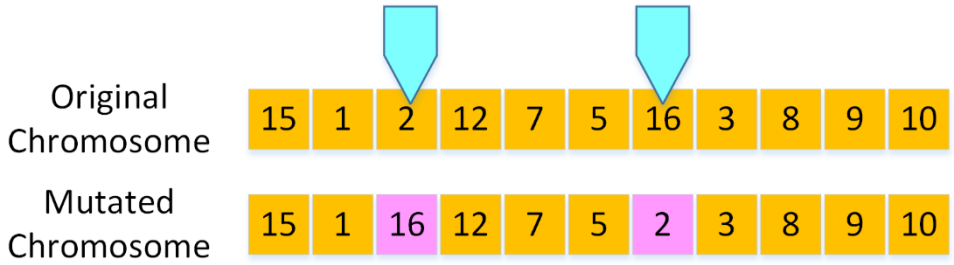

Fig. 2-18 Exchange mutation operator example

Selection Methods

The selection method is used by the GA to choose the parents to be used in reproduction and which individuals will carry on to the next generation. Typically, the selection criteria is based on the fitness of the individuals but other factors may be used, for example generational selection which only retains offspring, the "older" parent individuals are not retained for the next generation. There are several factors that influence the success of a selection method, including selection pressure. The selection pressure is how likely the best individuals in the population are to survive the next generation. Selection methods with a high selection pressure suffer from early convergence to a local optimum. This is because a high selection pressure results in diversity in the population decreasing too quickly. Again, like the crossover and mutation operators, there are a few standard selection methods.

Elitist selection is a relatively simple selection process, whereby the elite individuals in the top *x* percent of the population are used as parents and kept for the next generation, while the remainder of the population is discarded (Fig. 2-19). Elitist has a high selection pressure, and with only a small percentage of elite individuals kept it has a high probability of premature convergence. The percentage of individuals kept can be increased to counteract this but it will still likely loose diversity faster than other selection methods.

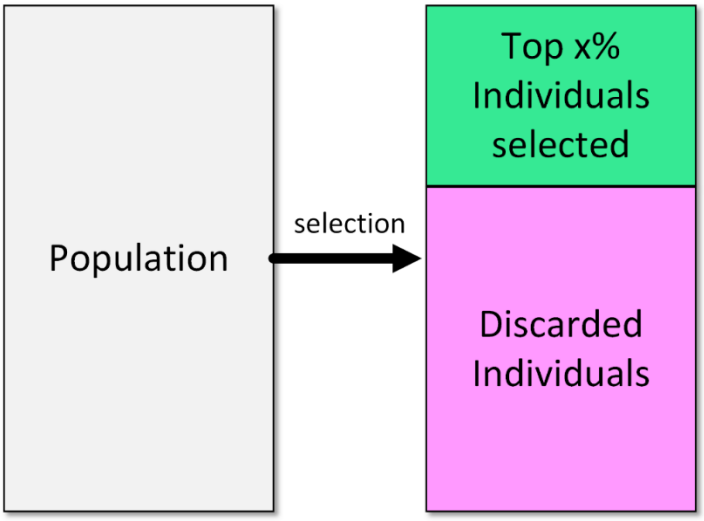

Fig. 2-19 Elitist selection method

Roulette wheel selection works by assigning a probability (portion of the roulette wheel) which determines the likelihood of a given individual of being selected (Fig. 2-2). The probability is generally calculated by the ratio of an individual's fitness to the average population fitness. After the probabilities are assigned, the wheel is "spun" (a random value is generated), and the region in which the value lies determines the selected individual. The "wheel is spun" repeatedly until the required number of parents is selected. This method means that individuals with high fitness have a high chance of selection, but the low fitness individuals also have a chance of selection unlike in elitist. When using roulette wheel selection diversity can be maintained as the population isn't forced to converge to the high fitness individuals.

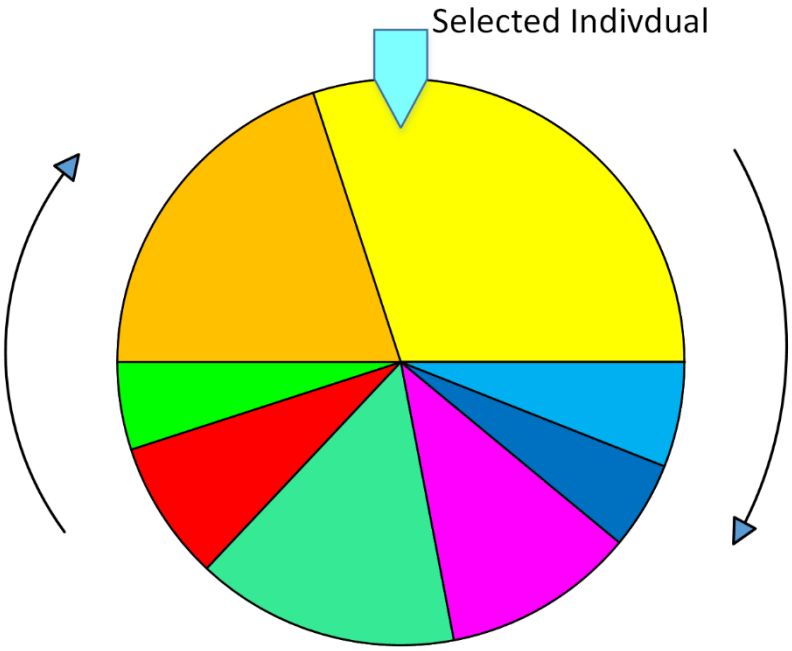

Fig. 2-20 Roulette wheel selection method

Rank-based selection is a very similar alternative to roulette wheel selection. Like roulette wheel selection each individual has a probability of selection, but the probability is determined by the individuals rank in the population rather than the magnitude of its fitness. Each individual rank is unique, meaning that individuals with the same fitness can't have the same rank. When the population is ordered by fitness the rank of the individuals with the same fitness will come down to how they were shuffled by the sorting algorithm employed.

Tournament selection uses a format similar to a sports tournament, where tournaments are executed between the individuals of the population. The tournament process works by dividing the population into small groups, each of these small groups undergoes a mini tournament until one individual is left. The winners are combined again to form the parents to reproduce and carry on to the next generation. The number of individuals in each of the groups that compete is called the tournament size, this can be two (binary tournament) or more individuals. Binary tournament selection (Fig. 2-21) is the most common implementation of this strategy, because a) the simplicity of its implementation and b) having a larger tournament size increases the probability that diversity is lost [67].

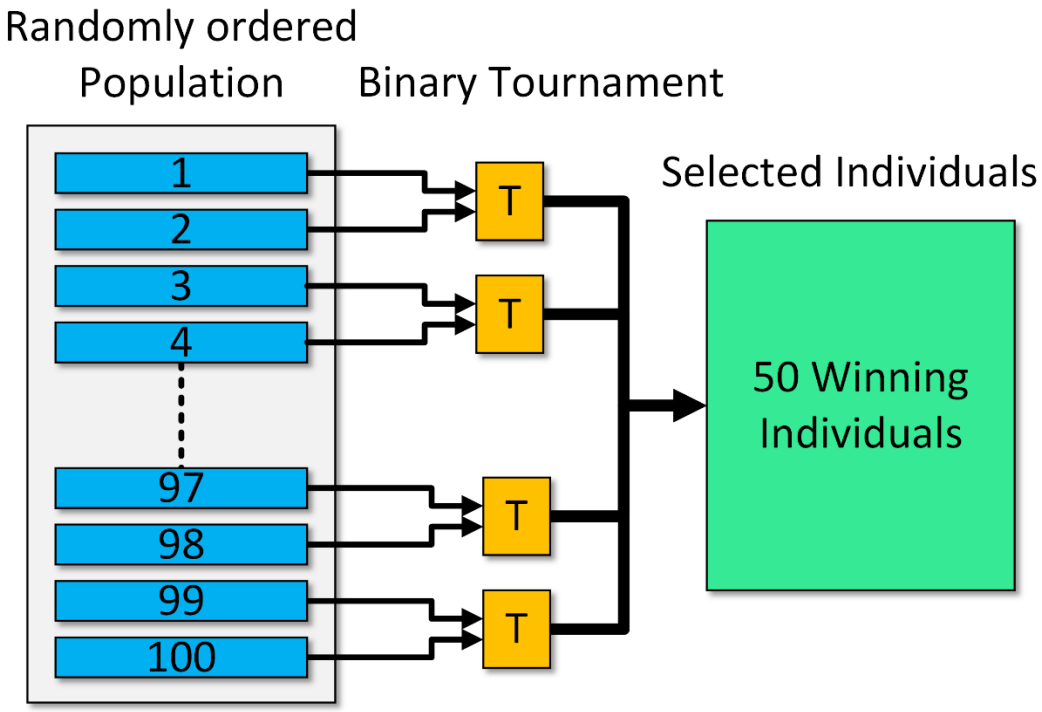

Fig. 2-21 Tournament selection method, showing a binary tournament example with 100 individuals

Outside of these standard selection methods there are several other approaches, including scaling, steady state, hierarchical and fitness uniform selection.

Evolutionary Strategies

Evolutionary strategies were developed in the 1960s and 1970s from research conducted by Rechenberg[55] and Schwefel[56]. Their research investigated evolving the shape of drag bodies in a wind tunnel. ES like the GA are based on the Darwinian concepts, and as such have similar stages for reproduction and the selection processes. ES work by performing crossover and mutation operators using μ parents to create λ offspring [57]. There are two main forms of ES which have distinct selection processes, (μ, λ) and (μ + λ).

- (μ, λ): works by selecting μ individuals from the offspring to be the parents of the next generation.
- (μ + λ): works by selecting μ individuals to be the parents of the next generation, the individuals that can be selected from are both the offspring and the parents of the current generation (Fig. 2-22).

Not all ES implementation utilize crossover, they mainly rely on mutation of parent chromosomes using values from a gaussian distribution. Because of this ES typically utilize an adaptive mutation rate, based on the rate of fitness improvement.

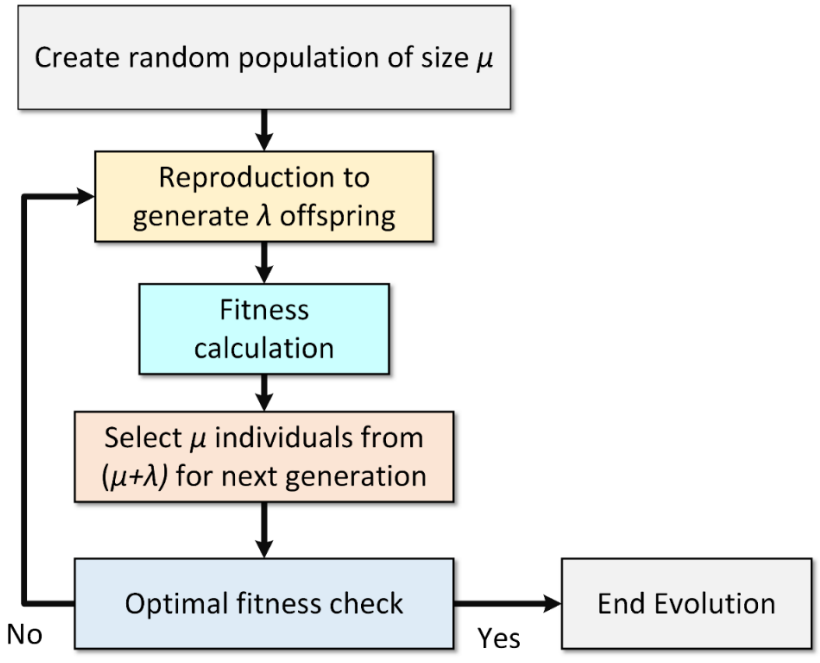

Fig. 2-22 Standard process for (μ + λ) Evolutionary Strategies

ES can be used for applications with discrete, combinatorial and real valued search spaces, but they are best suited to unconstrained real valued search spaces. However, ES have been used in the EHW field for the evolution of the EHW bitstream[15, 26, 27] which is a combinatorial search space with discrete values.

Differential Evolution

The GA and ES are predominantly used in the EHW field for the optimisation of the EHW bitstream. However there has been research investigating the use of other EAs such as differential evolution [68]. DE proposed by Storn & Price [58] follows some of the same evolution theories as genetic algorithms but takes a different approach using operations better suited for real valued vector solutions, opposed to GAs which uses the concepts of genes and chromosomes that define a solution in a particular search space. DE is considered to be simple to implement compared to other EA[69]. Like the genetic algorithm, DE is an iterative process trying to improve individuals in a population of size N but instead of following the idea of reproduction and crossover of parent's genes to develop children (new trial solutions) it first works on generating a donor vector Vi for each individual Xi in the population based on a difference weight F and the combination of other individuals. Crossover then occurs between the individual Xi and its donor Vi to generate a trial solution Ui based on the crossover probability CR, if the fitness f(Ui)>=f(Xi) then Ui is kept and replaces Xi in the population[70].

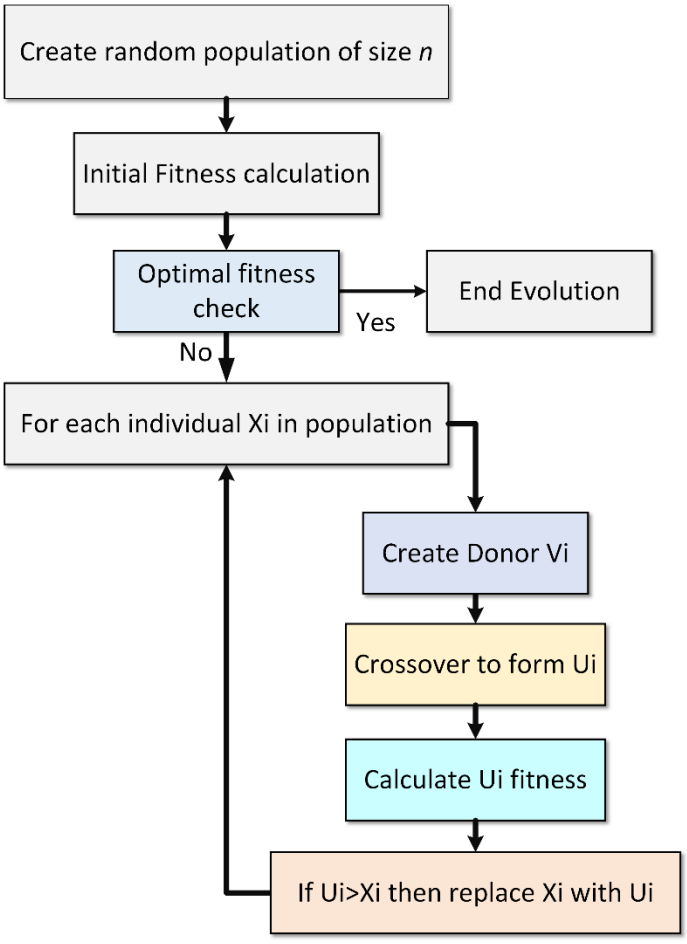

Fig. 2-23 Differential Evolution algorithm

Particle Swarm Optimization

PSO is a nature inspired optimization algorithm that loosely mimics the social behaviour of a flock of birds. However, it does have similar process to EAs such as the GA, ES and DE discussed, whereby a population is updated in a generational manner using techniques akin to crossover and mutation. The algorithm was originally introduced by Kennedy & Eberthart [62]. Like other EAs, PSO is an iterative process with a random population that is manipulated and updated continuously until the algorithm converges to an optimal solution. In the PSO algorithm the population represents a set of candidate solution thought of as particles with a position in the search space along with a velocity describing the particles movement through the search space. At each iteration of the algorithm the position of the "particles" is updated based on a velocity calculation which is formulated from the particles personal best position, the populations global best position and the adaptive inertia[71](weighted influence of previous velocity) of the particle.

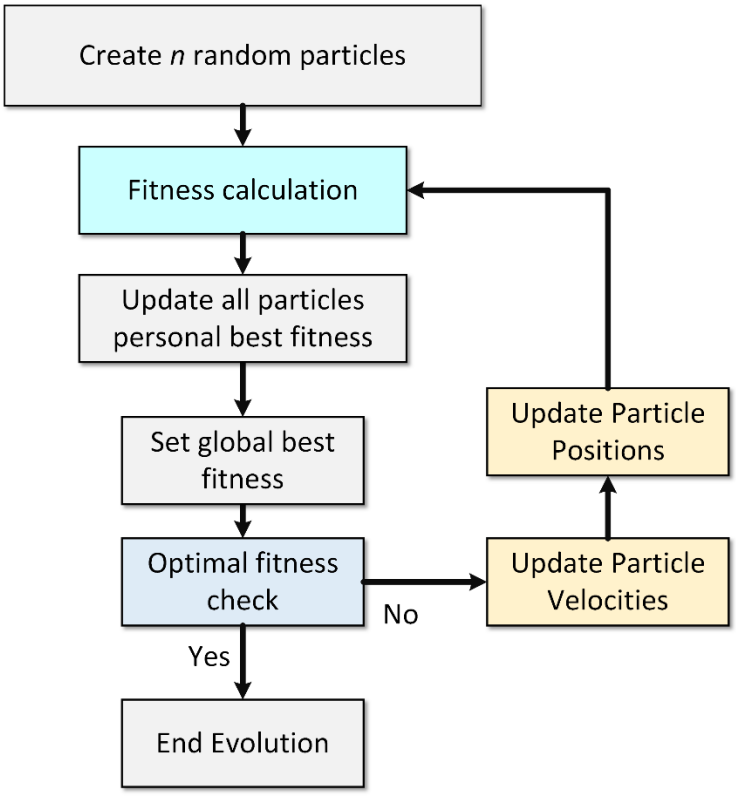

Fig. 2-24 Particle Swarm Optimisation algorithm

PSO has been used in the EHW field for the evolution of digital circuits[68, 72], but like DE it is not used to the extent of the GA and ES.

Chapter 3

Chapter 3: The Virtual Programable Logic Device

This chapter introduces the VPLD. In subsequent chapters the VPLD is evaluated by simulations and experiments for applications in robotic control and pattern recognition.

3.1. Overview

The aim of the VPLD is to move from an FPGA based hardware solution used in EHW to a software solution that can run on a standard CPU providing: 1) improved speed of optimization in robotic control applications; 2) reduced system complexity and 3) high cost associated with the development of EHW. To achieve this, the VPLD is to be based on the structure of a PLD but implemented in software rather than hardware. A PLD is an integrated circuit (common families are the CPLD and FPGA) which use configurable routing and logic elements comprised of a lookup table to create digital electronic circuits. The function of the PLD is determined by a configuration bitstream (CBS) which is used to configure the internal routing and LUTs. These reconfigurable ICs are used in the field of EHW to evolve electronic circuits for a variety of applications, including robotic control and pattern recognition. Like the PLD based EHW, a VPLD is required to be a reconfigurable architecture, that takes an array of inputs and produces an array of outputs that could be used either for the control of a robot, or to classifier data into separate categories for example. The VPLD configuration is to be optimised using machine learning algorithms such as EAs used to optimize EHW.

By using a software implementation, the VPLD system complexity will be reduced compared to the EHW, as the multiple programming environments with the need for both software and hardware languages can be replaced with a single programming environment and any standard programming language. Also, the software implementation of the VPLD will have a reduced cost compared to the EHW, as it can be implemented on any off the shelf platform and doesn't require expensive custom hardware. These proposed advantages are discussed further in section 3.7.

3.2. VPLD Architecture Design

The VPLD (Fig. 3-1) is a software architecture comprising of a two-dimensional feed-forward array of functional array blocks (FABs) grouped into vertical columns called layers. This is based on the hardware architecture of the PLD, which utilizes a two-dimensional array of configurable logic elements. Each FAB contains; 1) a multiplexer, and 2) a function element (FE), which is a table of selectable functions. The multiplexer selects inputs from the preceding layer, while the function element performs the selected operation on the data coming from the multiplexer. The outputs of one column of FABs connects to the inputs of the following column of FABs. The operation of the VPLD is determined by a configuration bitstream which configures the multiplexers routing and FEs of each FAB. The VPLD can be configured using machine learning optimization algorithms such as those used in evolutionary computation, i.e. genetic algorithms, differential evolution, particle swarm optimization etc.

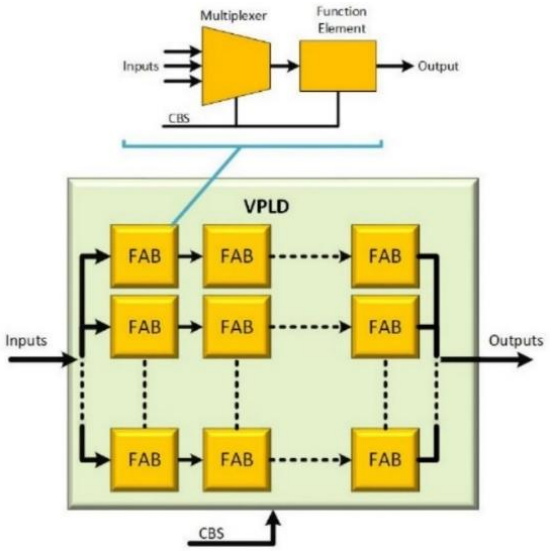

Fig. 3-1 Virtual Programmable Logic Device showing FABs placed in Cartesian array with internal multiplexer and function elements configured by the CBS

The VPLD FE like a PLD uses binary data and digital logic functions utilizing standard logic operators (NOT !, AND &, OR |, and XOR ^). Alternatively, the FE can also use floating point data and functions utilizing the standard mathematical operators such as addition, subtraction, and multiplication. Floating point arithmetic and mathematical operators can be implemented in hardware, but the implementation is very complex and uses a lot of resources on the FPGA. In software using floating point data and mathematical operators is trivial. In this research when comparing a VPLD using digital logic, and a VPLD using floating point arithmetic, they are denoted as a digital VPLD (D-VPLD) and floating point VPLD (F-VPLD) respectively.

3.3. VPLD Architecture Model

The VPLD consists of a two-dimensional array of function array blocks arranged in a series of layers to form a feed forward architecture, mathematically the VPLD denoted by the function ν with m layers can be described as the composition of a series of vector functions as shown below. Let $\boldsymbol{l}_m$ denote a layer, and f_{n} denote a function array block.

$$\nu(\boldsymbol{x}, \boldsymbol{S}) = \boldsymbol{l}_m \circ \ldots . \circ \boldsymbol{l}_2 \circ \boldsymbol{l}_1 \tag{3.1}$$

$$\nu(\boldsymbol{x}, \boldsymbol{S}) = \boldsymbol{l}_m(\ldots . \boldsymbol{l}_2(\boldsymbol{l}_1(\boldsymbol{x}, \boldsymbol{R}_1), \boldsymbol{R}_2), \boldsymbol{R}_m) = \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} \tag{3.2}$$

where $\nu(\boldsymbol{x}, \boldsymbol{S})$ is the function describing the VPLD with two inputs, 1) data vector $\boldsymbol{x}$**,** and 2) configuration $\boldsymbol{S} = [\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots . \boldsymbol{R}_{\mathrm{m}}]$ which is a set of matrices (one for each layer) used to store the configuration bitstream. The VPLD outputs the vector $\boldsymbol{y}$ with size p, which is equal to the number of FABs in the final layer.

The vector of functions that describes a layer with n FABs,

$$\boldsymbol{l}_m(\boldsymbol{x}, \boldsymbol{R}_{\mathrm{m}}) = \begin{bmatrix} f_1(\boldsymbol{x}, \boldsymbol{R}_{m,1}) \\ \vdots \\ f_n(\boldsymbol{x}, \boldsymbol{R}_{m,n}) \end{bmatrix} \tag{3.3}$$

has input vector $\boldsymbol{x}$ size q for the first layer, for the proceeding layers the size of the vector $\boldsymbol{x}$ is equal to the number of FABs n in the previous layer $(m-1)$.

Each FAB can be defined as a multivariate piece-wise function when using digital logic (D-VPLD),

$$f_n(\boldsymbol{x}, \boldsymbol{R}_{m,n} = [\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \lambda]) = \begin{cases} (x_a \,\&\, x_{\mathrm{b}}) \,\hat{}\, (x_{\mathrm{c}} | x_{\mathrm{d}}), & \lambda = 0 \\ (x_a \,\&\, x_{\mathrm{b}} \,\&\, x_c \,\&\, x_d), & \lambda = 1 \\ \vdots & \vdots \\ (x_a \,\hat{}\, x_{\mathrm{b}}) \,\&\, (x_{\mathrm{c}} \,\hat{}\, x_{\mathrm{d}}), & \lambda = 20 \end{cases} \tag{3.4}$$

and floating-point arithmetic (F-VPLD),

$$f_n(\boldsymbol{x}, \boldsymbol{R}_{m,n} = [\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \lambda]) = \begin{cases} (x_a + x_{\mathrm{b}}) \cdot (x_{\mathrm{c}} - x_{\mathrm{d}}), & \lambda = 0 \\ (x_a + x_{\mathrm{b}} + x_c + x_d), & \lambda = 1 \\ \vdots & \vdots \\ (x_a \cdot x_{\mathrm{b}}) - (x_{\mathrm{c}} \cdot x_{\mathrm{d}}), & \lambda = 20 \end{cases} \tag{3.5}$$

where a, b, c, and d are the four multiplexer configurations for the respective FAB n in layer m, and λ is the function element configuration for the respective FAB n in layer m. The parameters a, b, c, d, and λ make up the configuration of the respective FAB.

From the VPLD model, the three main hyperparameters of the VPLD architecture are a) number of layers m, b) number of FABs within each layer n, and c) type of function in the array block, and whether it is digital or floating-point. The functions in the function element LUT are required to use the datatype of the architecture (digital or floating point) but the number of functions and the number of variables in each function can vary, making these also hyperparameters of the VPLD. The number of variables used in the function element sets the number of multiplexers required for each FAB, functions that use four variables such as A+B+C+D require four multiplexers, functions such as A+B-C would only require three. In this research experiments are conducted investigating the major hyperparameters a) number of layers, b) number of FABs, c) types of functions used in a function array block (Chapter 6).

3.4. VPLD Implementation

An example VPLD diagram is shown below (Fig. 3-2), with 2-layers, 8 FABs in layer one and 5 FABs in layer two. Each FAB has a FE with 21 digital functions (Table 3-1) and requires four multiplexers (Fig. 3-3). This VPLD has a 16-bit input and a 5-bit output. For this architecture, the hardware equivalent would require a 253-bit CBS, 21-bits for each layer one FAB, and 17-bits for each layer two FAB.

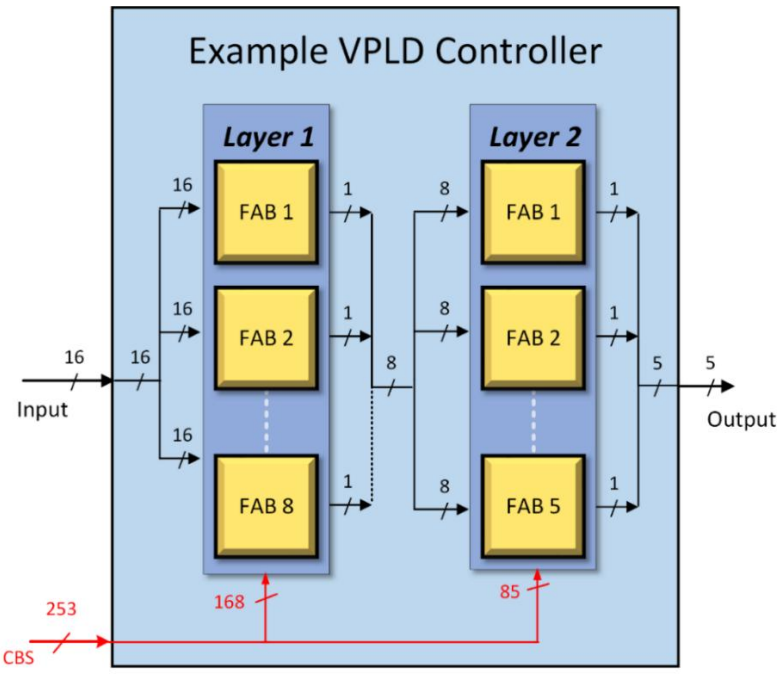

Fig. 3-2 Example VPLD Architecture

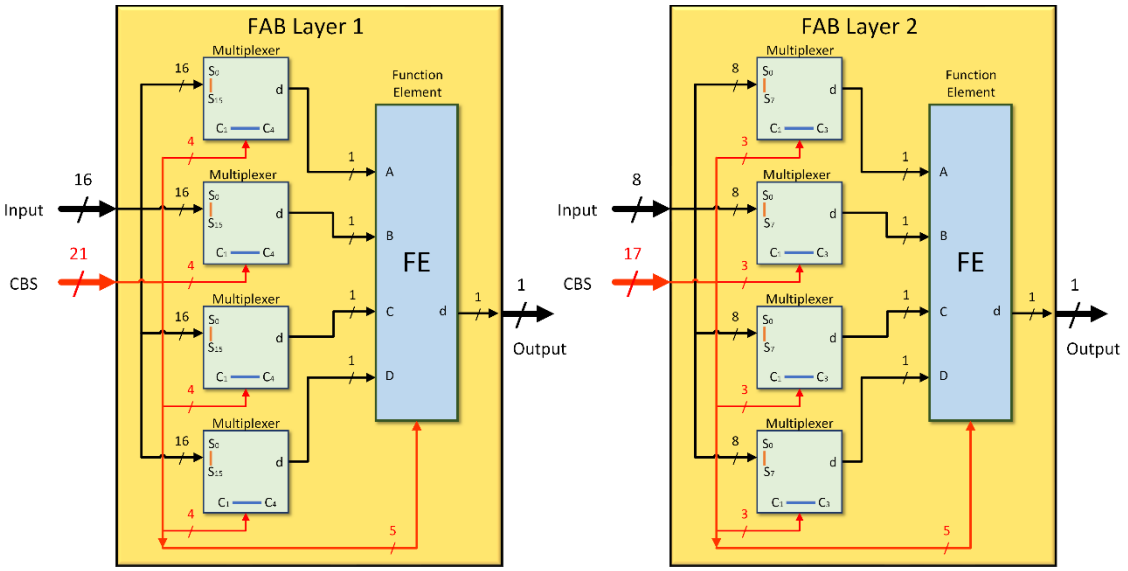

Fig. 3-3 Example FAB Architectures

Table 3-1 VPLD Function Element LUT example

No.	Function	No.	Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

The EHW equivalent of the above architecture would be created using a HDL program and implemented on a FPGA. In software, the VPLD can be implemented as a module with a set of layers that can be executed sequentially for a given input and configuration bitstream to generate an output. Each layer contains a set of FABs.

In this research the VPLD is implemented using functional programming in python, the VPLD is executed as a set of functions (Fig. 3-4 - Fig. 3-6) where the input and configuration is passed as inputs. A functional programming approach was chosen over an object-oriented approach because of the way python manages instances of class objects in memory, when using the object-oriented programming paradigm in python it slows down the execution of functions when looking up class parameters.

The downside to the VPLD implementation is when running on a CPU each FAB is executed sequentially, opposed to the EHW where the logic elements in each layer execute in parallel each clock cycle. However, this is outweighed by the speed gained when running the VPLD in the same native environment as the robotic simulations.

Algorithm 1: Function for the VPLD execution

Function: `feedForwardVPLD(input_data, cbs)`

input_data - data to use for control or classification, i.e. IR sensors, encoders, image data etc.

cbs – configuration of the VPLD architecture

```
1.  for layer_cbs in cbs:
2.      output_data=feedForwardLayer(input_data,layer_cbs)
3.      input_data= output_data
4.  end for
5.  return output_data
```

Fig. 3-4 Pseudo Code of the VPLD feedforward function

Algorithm 2: Function for a VPLD Layer execution

Function: `feedForwardLayer(input_data, layer_cbs)`

input_data – the original input data if the first layer, or the output data from the previous layer, passed from `feedForwardVPLD()`

layer_cbs – configuration of the respective layer

```
1.  for fab_cbs in layer_cbs:
2.      output_data=feedForwardFAB(input_data,layer_cbs)
3.      input_data= output_data
4.  end for
5.  return output_data
```

Fig. 3-5 Pseudo Code of the VPLD layer feedforward function

Algorithm 3: Function for a FAB execution

Function: `feedForwardFAB(input_data, fab_cbs)`

input_data – the original input data if a first layer FAB, or the output data from the previous layer, passed from `feedForwardLayer()`

fab_cbs – configuration of the respective FAB

```
1.  mux_cbs, fe_cbs = fab_cbs
2.  selected_data=multiplexer(input_data, mux_cbs)
3.  output_data =functionElement (selected_data, fe_cbs)
4.  return output_data
```

Fig. 3-6 Pseudo Code of the Function Array Block feedforward function

The FABs function element operation can be defined as a set of case statements where the state is defined by the FABs function element configuration (Fig. 3-10).

Algorithm 4: Function Element Lookup Table

Function: `functionElement(selected_data, fe_cbs)`

selected_data – the data selected by the multiplexer, passed from `feedForwardFAB()`

fe_cbs – configuration of the respective function element

```
1.  A, B, C, D = selected_data
2.  switch fe_cbs:
3.      case 0:
4.          output = (A&B)^(C|D)
5.      case 1:
6.          output = A&B&C&D
:       :
:       :
7.      case 19:
8.          output = (A^D)&(C^B)
9.  end switch
```

Fig. 3-7 Pseudo Code of the VPLD layer feedforward function

The VPLD configuration bitstream is stored in sets of two-dimensional integer arrays, one two-dimensional array of integers is used to store the configuration of a layer (Fig. 3-8). The CBS for the example architecture (Fig. 3-2) is shown in Fig. 3-9. Each row contains the configuration for one FAB, being the configuration of the multiplexers and the function element (Fig. 3-10).

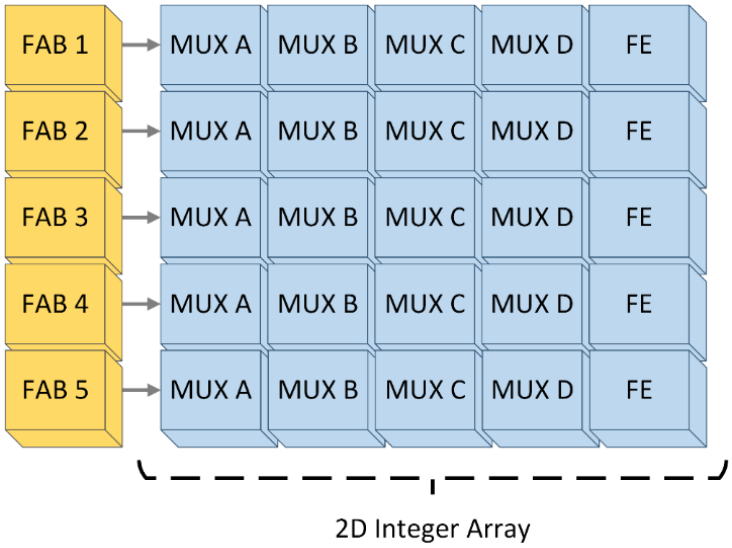

Fig. 3-8 CBS for a VPLD layer, 2D Integer Array

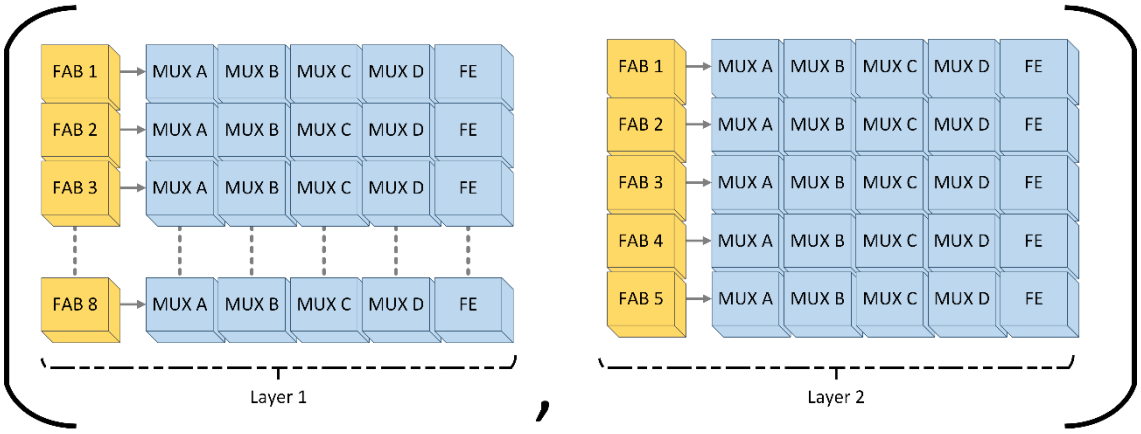

Fig. 3-9 Complete CBS for example VPLD showing the configuration for the *8-5* architecture

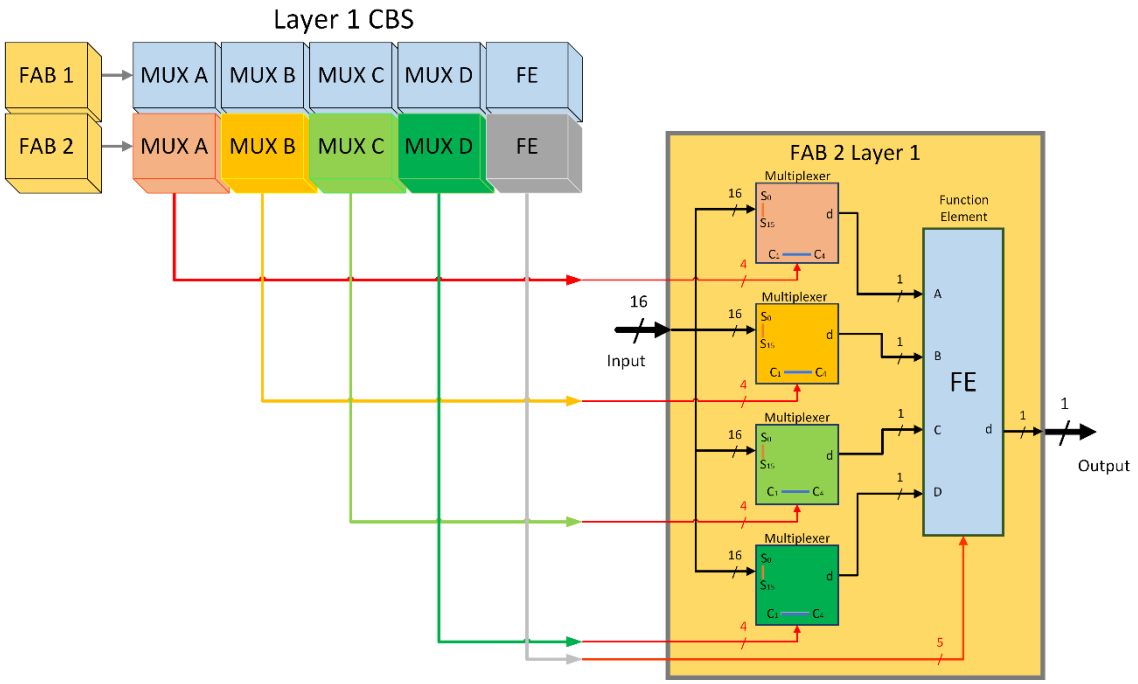

Fig. 3-10 CBS-FAB relationship

3.5. VPLD Optimization

The VPLD like other machine learning architectures, such as an ANN, use optimization algorithms to generate configurations such that the VPLD can be configured to optimally perform the given task. The VPLD is not differentiable as it is not continuous, therefore gradient descent approaches used to train ANNs such as stochastic gradient descent and other adaptations such as 'Adam'[73] cannot be used. However, nature inspired optimization algorithms such as evolutionary algorithms can be used to optimize the VPLDs CBS.

Primarily GAs and ES have been used in optimizing EHW in robotics and other domains. In this study GAs are chosen to optimize the VPLD as GAs are better suited to the optimization of discrete combinatorial chromosomes such as the VPLD CBS, whereas ES are primarily used in real valued unconstrained optimization. In future work a comparison of GAs, ES and other evolutionary algorithms such as differential evolution and particle swarm optimization could be investigated.

The GA originally introduced by Holland through his work on adaptation in natural and artificial systems[59-61], is one of many evolutionary algorithms inspired by Charles Darwin's theories on natural evolution [19]. The genetic algorithm has three main stages; 1) fitness assessment, where in the case of robotics trial individuals are evaluated (typically in simulation) for the given task; 2) selection, where the fittest individuals are chosen to survive to the next generation and/or to participate in the final step; 3) reproduction, where typically the fittest individuals chromosomes undergo crossover and mutation operators to generate new offspring for the next generation (Fig. 3-11).

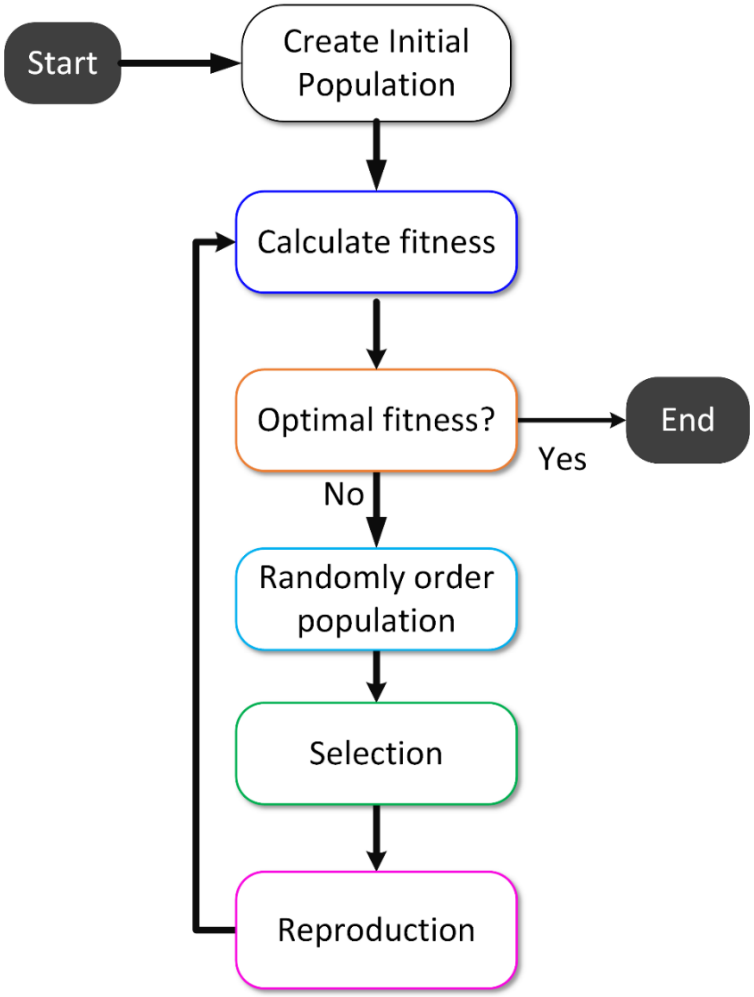

Fig. 3-11 Genetic Algorithm showing the iterative process of fitness assessment, selection, and reproduction.

The selection method used can have a large impact on the evolutionary efficiency (generations and computation time to evolve) of the EA system. Many selection methods have been devised including elitist, roulette, rank and tournament selection. In this research, tournament-based selection methods are used, these are discussed in more detail in subsequent chapters.

Tournament-based selection is 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 [74]. Also, 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 selection pressure, this means the chance of the population losing diversity is increased[67].

The reproduction stage of the GA requires the crossover of parent chromosomes with mutation to generate offspring. As with the selection method, there are a variety of crossover methods, including one-point, two-point, multi-point and uniform crossover. In this research multipoint crossover is used, the advantage of multipoint crossover compared to other crossover methods is it allows for more of the search space to be explored, this is because a higher fidelity crossover will make new combinations/strings of genes within the chromosome. However, this can also be destructive at the end of the evolution when useful strings/combinations of genes are lost during crossover. The purpose of mutation in the reproduction stage is to maintain the diversity of the population, also due to the first generation individuals being randomly generated, it may add genes that are not present in the initial population. A low mutation rate is typically chosen in the range of 0.1% to 3%, a low mutation rate aids the search for an optimal solution. An excessive mutation rate would be overly destructive of the chromosome, resulting in useful genes from past parents being lost.

3.6. VPLD EA Implementation

The VPLD EA implementation (Fig. 3-12) requires a single programming environment. All the code is written in Python running on Visual Studio Code. Three software scripts are required; 1) the GA program to perform the optimization of the VPLD configuration bitstream; 2) the robot simulation used to test the fitness of individuals; and 3) the VPLD controller. These python scripts are optimized with the use of NumPy (a high-level mathematical library) and Numba (a JIT compiler that is used to translate some python methods into machine code using LLVM). To test the fitness of an individual, its chromosome is used to configure the VPLD, and the simulation is run to determine the individual's fitness. The simulation provided the inputs to the robot and the VPLD provided the outputs back to the simulation to determine the next state of the robot, this cycle is repeated until the simulation period ends. At the end of the simulation the fitness of the individual is calculated based on its performance.

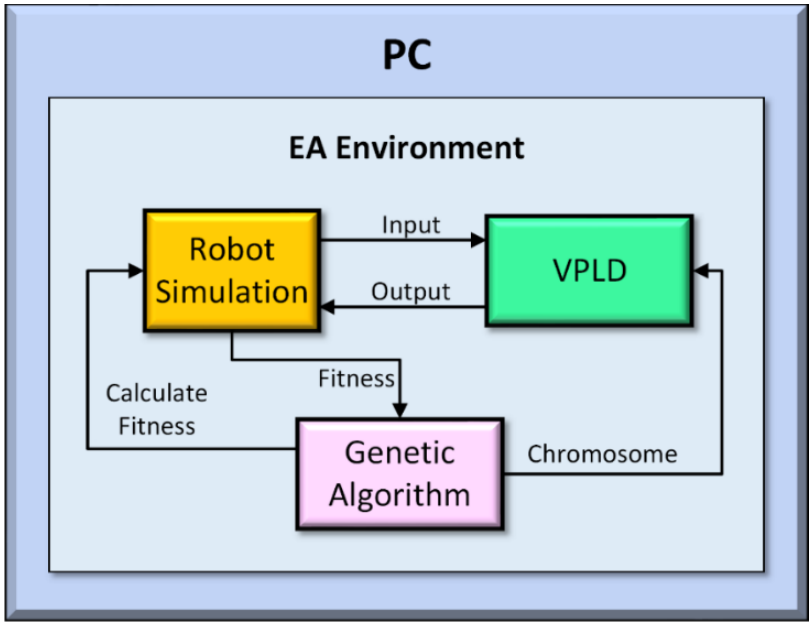

Fig. 3-12 VPLD EA Implementation for robotic control

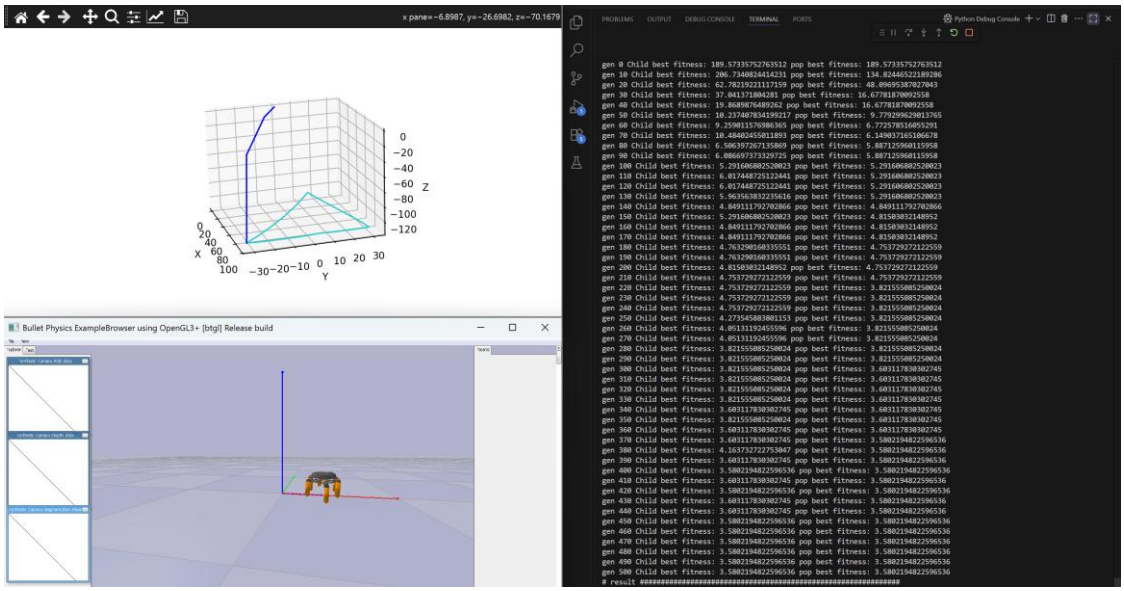

Fig. 3-13 EA Implementation for hexapod gait evolution. Top left) evolved gait, bottom left) trialling evolved gate in PyBullet, and right) GA running evolution of VPLD CBS for hexapod gait control.

This VPLD EA implementation can be compared to the equivalent EHW implementation (Fig. 3-14). The EHW implementation is more complicated compared to the VPLD, requiring more resources and uses both software and hardware programming languages. The hardware used in this research is the DE10 Nano board incorporating a Cyclone V FPGA with a SoC ARM processor. The GA and robot simulation are Python scripts similar to those used by the VPLD and are executed on the SoC ARM processor.

The EHW IP is written in Verilog using the Altera Quartus IDE, also requiring Platform designer to design the architecture of the system including the HPS, clocks, PIO ports and the associated interconnections. An interface between the SoC processor and EHW is written in software to use the AXI lightweight bridge, which is the interface between the hard processor and the FPGA fabric within the Cyclone V chip. To control this interface an additional set of methods on the ARM processor is required.

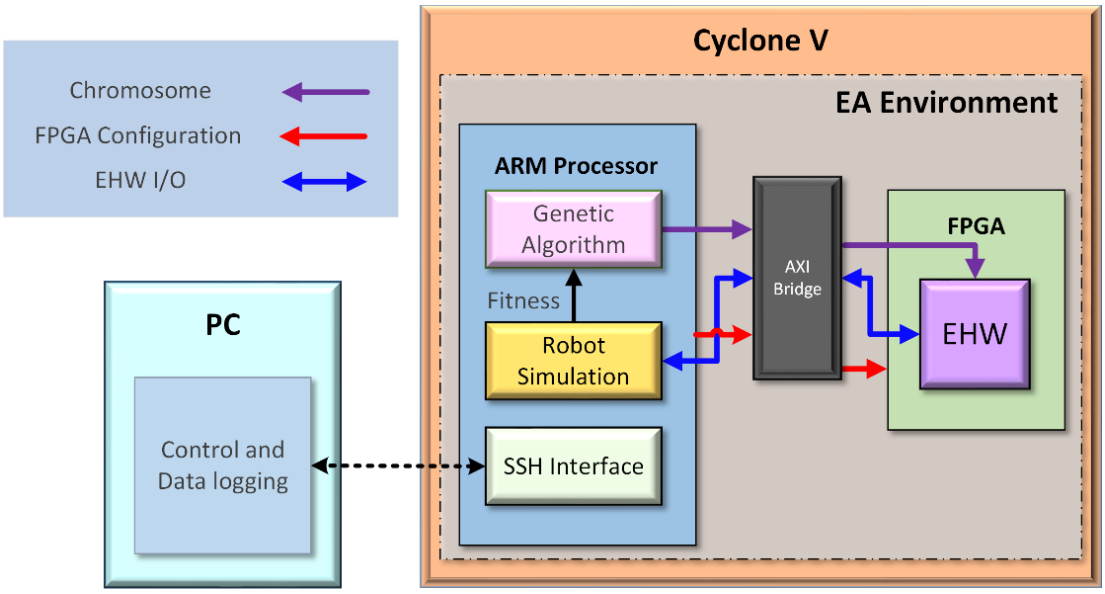

Fig. 3-14 EHW EA Implementation for robotic control

As can be seen the EWH system is significantly more complex than the VPLD system, when a researcher wants to make any alterations to the design, it may require changes to the EHW system at multiple levels; i.e. changes to the EHW architecture will require changes to the system design, the interface between the FPGA fabric and ARM processor, and possibly the GA. Making such alterations to conduct new experiments is a time-consuming process slowing down research. Also, compiling the HDL used to configure the FPGA takes several minutes. The complexity also poses a risk of introducing errors into the system with multiple different components all required to carry out a single operation, including a) the custom IP defining the EHW, b) the program used to interface the processor and FPGA, c) the GA code managing the chromosomes of the individuals, and d) the system architecture defined in platform designer.

3.7. Discussion

Chapter 3 has introduced the VPLD architecture, detailed its implementation, and also covered its optimization using evolutionary algorithms. From this chapter potential advantages of the VPLD over the EHW have been identified as well as some possible disadvantages. A minor disadvantage of the VPLD is because it runs on a CPU each FAB is executed sequentially, opposed to the EHW where the logic elements in each layer execute in parallel each clock cycle. Meaning the process of feeding forward data from input to output in hardware on the FPGA fabric will be faster than in software. When only taking into account the execution of the EHW and VPLD architectures, typically the EHW on a FPGA will have a response time in the realm of a few nano seconds (40ns for a two layer architecture not including pre and post processing of I/O data), and the VPLD will have a response time of a few micro seconds (27us using python programming, including pre and post processing of input data). In most applications where the evolved controllers are deployed the slower response time of the VPLD is irrelevant. The difference in response time is also irrelevant during optimisation as the VPLD interfaces directly with the software simulation, whereas an EHW controller on the FPGA fabric must transfer I/O data to the simulation and receive trial configurations from the EA, both of which will be running externally either on an embedded arm processor or on a PC.

Even though the propagation delay of the VPLD may be slower than EHW, having the VPLD in software does give rise to a number of potential advantages including, a) reduced system complexity, b) a faster evolutionary time, c) the use of floating-point variables and mathematical functions, d) reduced cost, and e) a large range of processor based devices for deployment. Each of these hypothesised advantages are detailed further below. The advantages and disadvantages of the VPLD will be evaluated in subsequent experimental chapters.

3.7.1 Reduced system complexity

The VPLD EA system is much simpler than the EHW implementation, as the GA, simulation, and the VPLD itself can be implemented on a PC (or any other processor based device) using one integrated development environment and one software language. Whereas the EWH EA system is implemented on a FPGA requiring several elements including the EHW IP written in a hardware descriptive language, system design to be implemented on the FPGA, and software for the on-board SoC processor to run the simulation and GA.

3.7.2 Speed of Evolution

The VPLD can run natively on a PC for training which has significantly higher computing performance compared to EHW. The EHW training takes place typically on a hardcore processor embedded within the FPGA or softcore processor running on the FPGA. However robotic simulations are slow to run within the FPGA environment due to the lower computer performance. Alternatively, the simulation could be run on a PC and the chromosome downloaded to the FPGA. But this results in a substantial bottleneck in the evolution process. This problem is exacerbated with more complex EHW architectures requiring a larger configuration bitstream resulting in a very slow optimization process.

Having the VPLD running in the same environment as the simulation, (as done with ANNs) it is expected to improve the speed of evolution. This is investigated in the experiment conducted to validate the VPLD architecture in comparison to the EHW (Chapter 4). It is expected that the VPLD is faster to evolve for the robotic control application due to the simulation, GA and VPLD running on a single processor that has more cores, and higher clock speeds than that of a FPGA running a SoC processor and requiring to externally interface to the EHW implemented on the FPGA fabric.

3.7.3 Floating-Point variables and Algorithms

The VPLD can easily use floating point variables and perform arithmetic operations enabling the function elements to be expanded from simple logic to include more complex mathematical functions which could improve scalability of such an architecture for common machine learning problems in robotics and pattern recognition. Although the use of floating-point arithmetic is possible in the FPGA fabric, it consumes a lot of resources on the FPGA due to the implementation of a floating-point unit for every mathematical operation. This quickly becomes a problem as the architecture scales. The VPLD using floating-point arithmetic is investigated and compared with the use of digital logic in subsequent chapters.

3.7.4 Reduced Cost and Increased Portability

The VPLD unlike EHW doesn't require a complex FPGA development board or a custom motherboard to be developed both of which are costly resources. Instead, it has the ability to be deployed on a wide range of platforms including PCs, mobile phones, ARM based embedded systems such as the Raspberry Pi, or low-cost microcontrollers such as the ESP32. A cost comparison of some development boards that can be used to implement the VPLD and EHW are shown below (Table 3-2 and Table 3-3).

Table 3-2 Price comparison of FPGA devices used in EHW research

Item	Cost (USD)
Terasic DE10 Nano – ARM-A9 dual core 800MHz (used in thesis)	$225
Terasic DE10 Standard – ARM-A9 dual core 925MHz	$500
Digilent Xilinx Cora – ARM-A9 single core 667MHz	$166
Digilent Xilinx Zynq – ARM-A9 dual core 667MHz	$589

Table 3-3 Price comparison of CPU based devices used for the VPLD Research

Item	Cost (USD)
Raspberry Pi 4 – ARM-A72 quad core 1.8GHz(used in thesis)	$35
Raspberry Pi 5 – ARM-A76 quad core 2.4GHz	$89
Raspberry Pico ARM-M33 dual core 150MHz	$6
ESP32 RISC-V single core 160 MHz,	$9

The Raspberry Pi is often the main controller in low cost ready to run robotic platforms such as the Turtlebot[75] robot series of mobile robots (Fig. 3-15). Research projects looking at low-cost robotics platforms for research and education have used the Raspberry Pi as the main controller, including for quadrupeds[76-78] and mobile robot platforms[79-83]. The VPLD can also be implemented on extremely low-cost microcontroller driven robots that use platforms such as the ESP32.

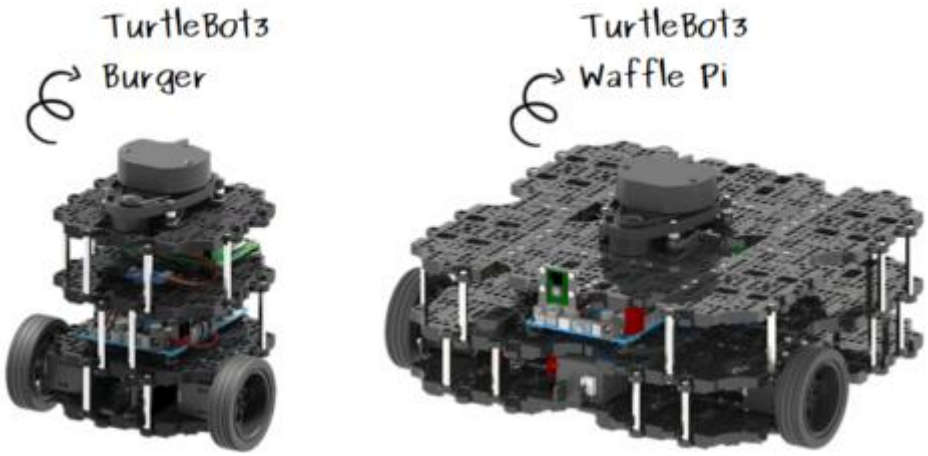

Fig. 3-15 Turtlebot3, developed for use in research and education. Adapted from[75]

3.7.5 System Overhead

As the VPLD is a software implementation of a hardware-based architecture, a degree of overhead is introduced through the virtualisation of logic, routing, and configuration mechanisms. In the VPLD, function elements, interconnections, and configuration bitstreams are represented and evaluated in software, requiring sequential execution on a processor rather than parallel execution in dedicated hardware. This results in a increase in computational workload during the feedforward operation when compared to fixed software controllers or highly optimised artificial neural network implementations. In particular, the evaluation of function element look-up tables and the traversal of the VPLD layers contribute to a increase in propagation delay. The VPLD is executed in the order of microseconds, whereas the EHW on the FPGA fabric executes in the order of nanoseconds, but for the EHW this does not consider the integration with other systems.

However, this overhead must be considered in the context of the overall system design and the alternatives it replaces. Compared to evolvable hardware systems implemented on FPGAs, the VPLD avoids the substantial overhead associated with hardware description, synthesis, compilation, and configuration, as well as the communication latency between a processor and the FPGA fabric. By executing entirely within the processor domain, the VPLD removes these bottlenecks and enables rapid reconfiguration during optimisation. While additional memory is required to store the VPLD configuration bitstream and architectural parameters, this requirement is minimal on modern CPU-based platforms and does not impose a practical limitation for the applications investigated in this research.

Importantly, the system-level overheads introduced by the VPLD do not adversely affect the response time requirements of the target application domains. For robotic control tasks, controller execution should occur on the order of milliseconds, which is well within the timing constraints of the platforms used. Similarly, for pattern recognition applications such as melanoma classification, inference operates on timescales of seconds, where the additional computational overhead introduced by the VPLD will be negligible relative to the overall processing pipeline. As such, the overheads associated with the virtualisation of the programmable logic architecture will not limit the practical deployment of the VPLD for the robotic control and pattern recognition problems considered in this thesis.

3.7.6 Power Consumption

Power consumption is an important consideration in many embedded and hardware-based systems, particularly where devices are energy constrained or battery powered. In the context of evolvable hardware, low instantaneous power consumption is often cited as a benefit of FPGA-based implementations due to their ability to exploit hardware parallelism. However, power consumption must be considered alongside execution time, system complexity, and architectural flexibility to provide a meaningful assessment of overall efficiency.

The VPLD is designed as a software-based implementation of a programmable logic device, intended to operate on general-purpose processors across a wide range of platforms. As a result, the VPLD does not primarily target minimum instantaneous power consumption, but instead focuses on reducing architectural complexity, improving portability, and enabling fast reconfiguration during optimisation. While processor-based platforms typically exhibit higher peak power consumption than FPGA-based systems, they benefit from significantly higher clock frequencies, mature compiler toolchains, and efficient execution of complex control and optimisation algorithms. From an architectural perspective, the VPLD can be advantageous in terms of energy efficiency when execution time is considered. By executing entirely within the processor domain, the VPLD avoids the additional overhead associated with hardware configuration, inter-device communication, and synchronisation between processors and reconfigurable fabric. This enables faster evaluation of candidate solutions during evolution, which in turn reduces the total runtime of optimisation processes. As a consequence, although instantaneous power consumption may be higher than that of hardware-based evolvable systems, the overall energy required to complete an evolutionary task can be reduced.

In addition, for robotic platforms, the power consumption of the control system represents only a small fraction of the total system power budget. In mobile robots, the dominant contributors to power consumption are typically the actuators rather than the onboard computation. In the case of the hexapod robot used in this work, the system employs eighteen motors, each of which consumes power on the order of watts during operation. In comparison, the low power consumption of the processor executing the VPLD is negligible in the context of the overall robot power draw.

Chapter 4

Chapter 4: Virtual Programmable Logic Device for Robotic Control

This chapter investigates how the VPLD performs in comparison to ANNs evolved for robotic control. The chapter is broken up into two sections: 1) the gait control of a hexapod robot; and 2) the autonomous navigation of a two-wheel drive mobile robot for obstacle avoidance, light following and light following while avoiding obstacles. The first section on evolving the gait of a hexapod robot is then broken up into two parts: 1) a comparison of the VPLD with EHW for validation of the VPLD; and 2) a comparison of the VPLD with an ANN. The second section on autonomous navigation of a two wheeled robot performs only a comparison of the VPLD with an ANN. In the 2WD robot section the expansion of the VPLD function element is investigated, specifically using floating-point data and mathematical arithmetic in its functional elements.

The evaluation of the three controllers for the respective control problems is measured by: 1) the evolutionary efficiency determined by the number of generations and time required to reach an optimal solution; 2) the controller performance, how well the controller performs the evolved task. The same GA is used for the evolution of the three systems to create a fair comparison.

4.1. Hexapod Robotic Control

This section describes in detail the system architectures and respective chromosomes of the VPLD, EHW and ANN used to solve the hexapod control problem. The VPLD, EHW and ANN are evolved using the same GA for the optimal gait of the hexapod, which allows the robot to walk forward in a straight line maintaining a constant heading and body attitude. The results of the controller's evolution are analysed and used to draw conclusions on the comparison of the VPLD with EHW, and the VPLD with an ANN for robotic control. The performance of the evolved controllers is validated firstly in a computer simulation of the robot, followed by practical implementation in the real robot.

As stated the first area of investigation in this chapter is the comparison between the VPLD and EHW. The VPLD and EHW share the same architecture, meaning that the operation of the VPLD and EHW are the same for the same chromosome, the only difference being one architecture is executed in software and the other in hardware. Therefore, the results of the experiments conducted should show that the VPLD and the EHW have a similar controller performance and number of generations to evolve to a solution. However, it is not expected that the results are identical due to the random probability of the crossover and mutation of individuals chromosomes during the GA process. The propagation delay of the VPLD and EHW is also investigated for this problem, propagation delay is important in robotic control problems because the VPLD propagation delay has to meet the timing constraint of practical applications.

The second subject investigated in this chapter is how the VPLD performs in comparison to an ANN. The VPLD and ANN are compared in two different applications in robotic control. For the hexapod control problem, the VPLD using digital logic functions is compared to a feedforward fully connected ANN evolved using the same GA. This experiment is used to investigate if the VPLD can match the performance of an ANN for a robot locomotion problem.

4.1.1 Hexapod Robot

The hexapod robot is a legged robot locomotion problem, where the evolved systems are required to optimally control the gait of the robot. An optimal hexapod controller allows the robot to walk forward in a straight line maintaining a constant heading and body attitude. A simulation of the hexapod robot is required for the evolution of the VPLD, EHW and ANN controllers to evaluate individuals during the fitness assessment process. The robot simulation is based on a physical hexapod robot (Fig. 4-1). This physical robot is used to validate the evolved simulation controller's performance.

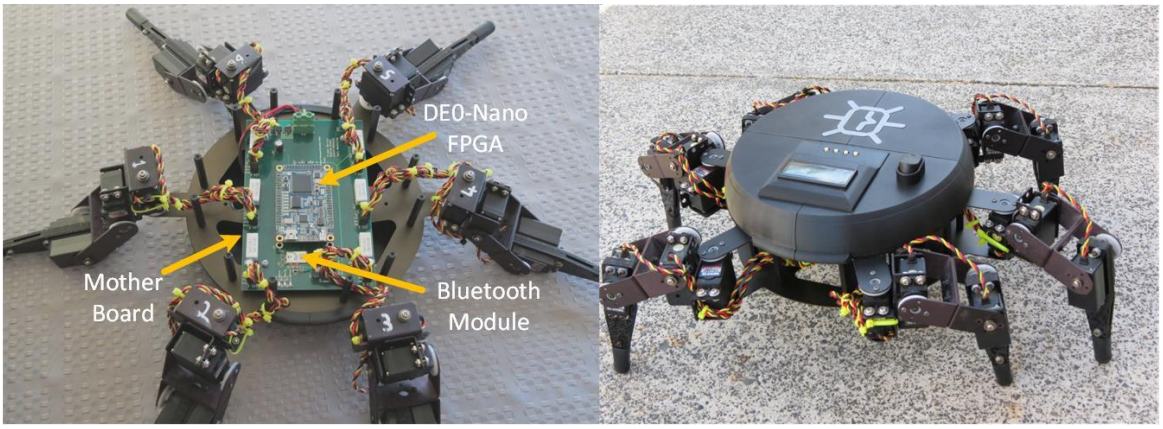

Fig. 4-1 The physical hexapod robot that was designed at the Auckland University of Technology for evaluation of EHW in real world applications

Robotic Platform

The physical robot is the Lynxmotion CH3-R Hexapod with custom electronics that were designed by the author for the Auckland University of Technology. The intended purpose of the hexapod is for the evaluation of EHW in real world applications. The electronics incorporate a RN4677-V Bluetooth module for linking to a remote controller, and the Terasic DE0-Nano board to control the servo motors. The Cyclone IV FPGA on the DE0-Nano allows for all three controller types (VPLD, EHW and ANN) to be compared on the same robot (Fig. 4-2). The EHW is implemented as custom IP on the FPGA, and the VPLD and ANN are implemented in C on a NIOS-II softcore processor running at 50MHz.

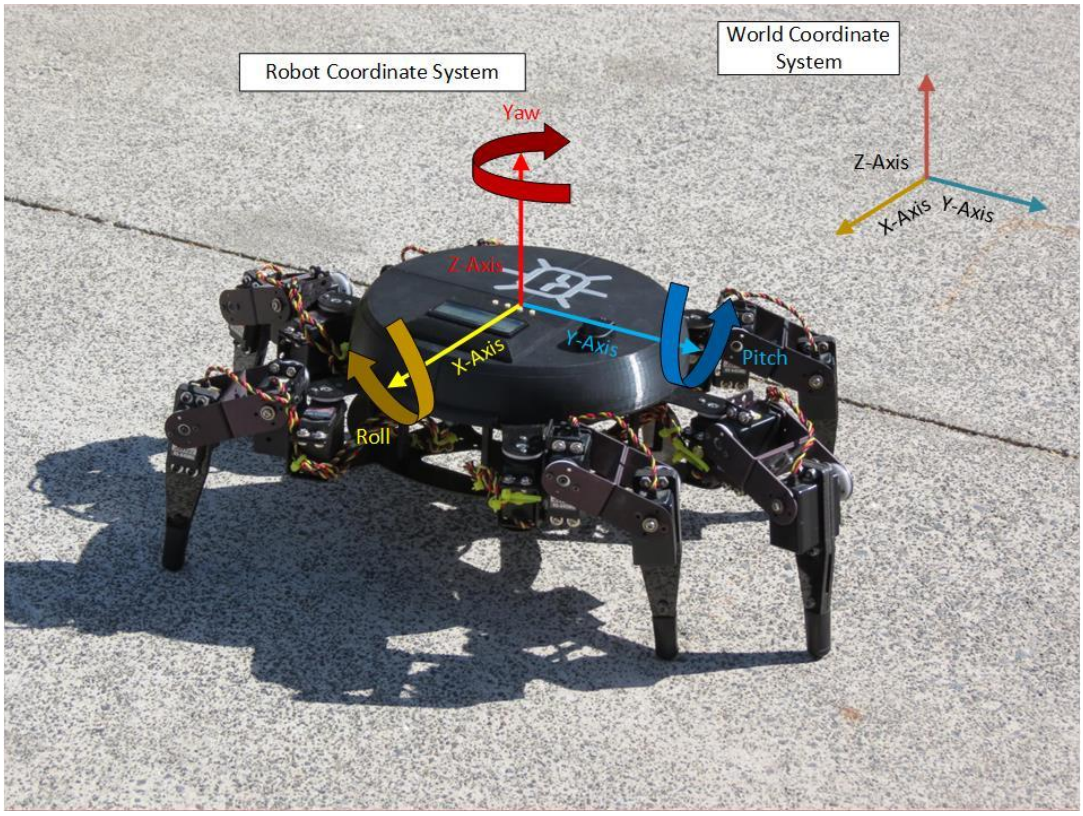

Fig. 4-2 Robot coordinate system

Mathematical Model

A kinematic model of the hexapod robot is required for the fitness evaluation of the controller chromosomes. The kinematic model of the leg is developed from geometric analysis of the Lynxmotion hexapod robot (Fig. 4-1 & Fig. 4-2) which uses legs with 3DOF.

The equations (4.1) to (4.9) define the hexapod leg model where, positions $\boldsymbol{H}$, $\boldsymbol{K}$, $\boldsymbol{F}$ and $\boldsymbol{O}$ are the vectors defining, the hip, knee, foot and leg origin respectively relative to the origin of the robots body (Fig. 4-2). 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 $\gamma, \alpha \; and \; \beta$ are the joint angles for the servos positioned at $\boldsymbol{O}$, $\boldsymbol{H}$ and $\boldsymbol{K}$ respectively (Fig. 4-3).

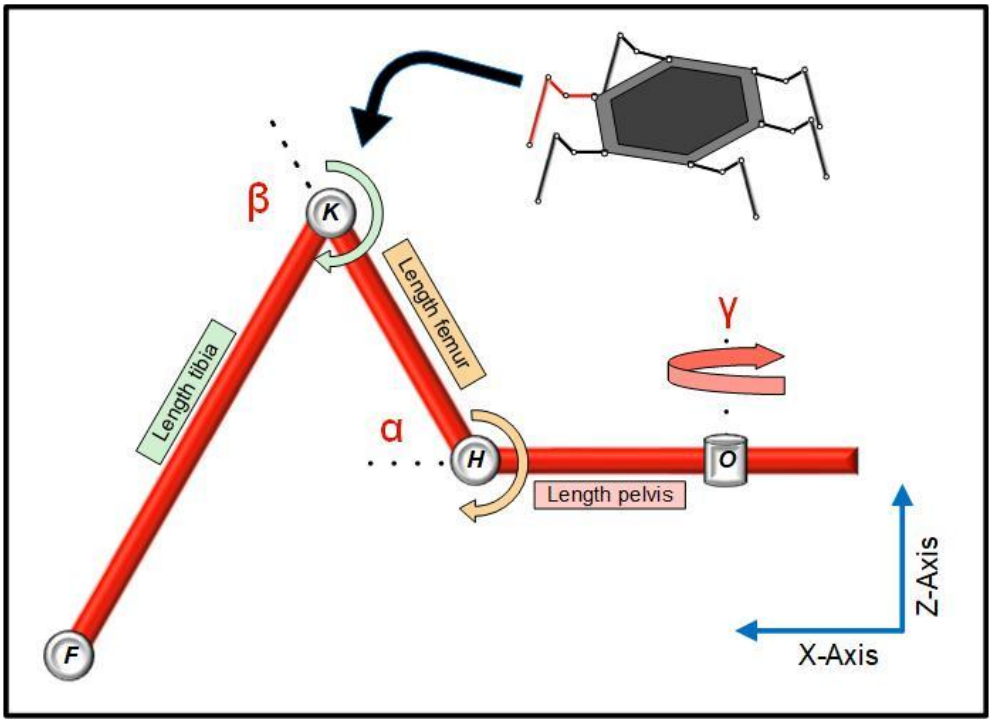

Fig. 4-3 Hexapod Robot Leg model

$$H_x = O_x + l_{pelvis} * \cos(\gamma) \tag{4.1}$$

$$H_y = O_y + l_{pelvis} * \sin(\gamma) \tag{4.2}$$

$$H_z = O_z \tag{4.3}$$

$$K_x = H_x + l_{femur} * \cos(\alpha) * \cos(\gamma) \tag{4.4}$$

$$K_y = H_y + l_{femur} * \cos(\alpha) * \sin(\gamma) \tag{4.5}$$

$$K_z = H_z + l_{femur} * \sin(\alpha) \tag{4.6}$$

$$F_x = K_x + l_{tibia} * \cos(\beta) * \cos(\gamma) \tag{4.7}$$

$$F_y = K_y + l_{tibia} * \cos(\beta) * \sin(\gamma) \tag{4.8}$$

$$F_z = K_z + l_{tibia} * \sin(\beta) \tag{4.9}$$

Simulation

Two simulations are created: 1) a single leg simulation; and 2) a full robot simulation. To reduce the evolutionary computation time, a single leg simulation (rather than the full simulation) is used for calculating the fitness for the controllers during the evolutionary process. The robots walking gait can then be produced using the evolved leg gait utilizing alternate phases for each of the six legs. When testing an individual, the angular position of the leg joints are stored in a matrix during the simulation step of the GA. The kinematic equations are used to form a set of *3D* cartesian coordinates to determine the trajectory of the hexapod's legs throughout the gait (Fig. 4-4). The angular and spatial position data is used in determining the fitness of an individual's gait.

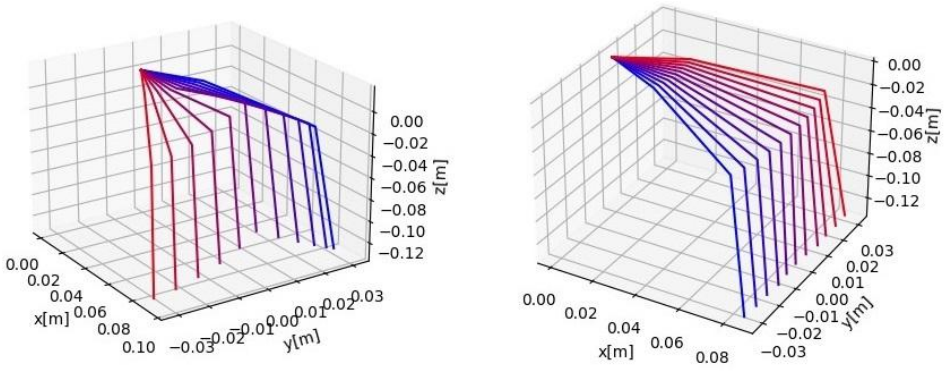

Fig. 4-4 Leg simulation left) air phase, right) ground phase

The full simulation of the hexapod (Fig. 4-5) is made in PyBullet [20] and is modelled on the AUT physical robot (Fig. 4-1). This simulation is used after the evolution process is complete, this is so the evolved controllers for the VPLD, EHW and ANN can be validated in software before being deployed on the real robot.

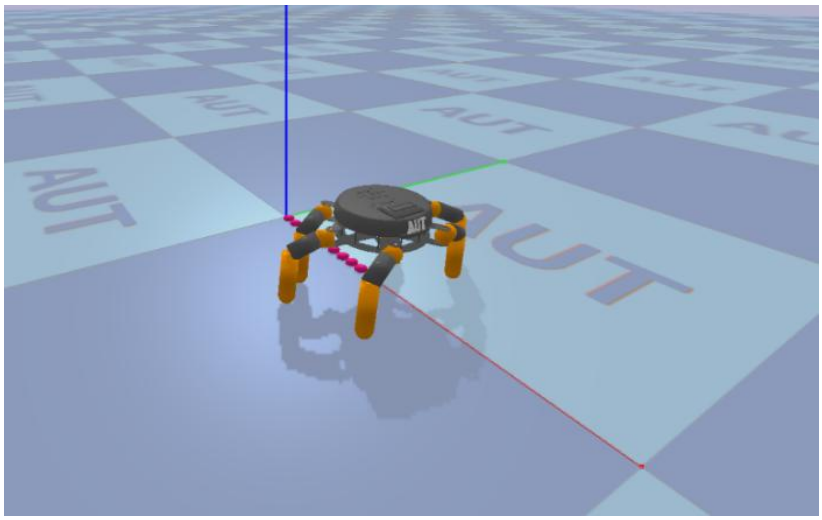

Fig. 4-5 PyBullet hexapod full simulation, modelled on the physical hexapod

A practical application of EHW is fault tolerance, in future work the VPLD could be investigated for use in fault tolerant robotic control. This physics simulation could be used in future research as a training environment for the VPLD to adapt to mechanical and sensor faults.

4.1.2 VPLD and EHW Control System Architecture

The top-level architecture for the VPLD and EHW control systems (Fig. 4-6) consists of three identical units, each controlling one of the three servo motors in the hexapod leg. The VPLD has three inputs to show the current state of the leg motion and three outputs to control each of the three leg servo motors. The first input is the ground/air phase signal; this is an 8-bit value that shows whether the leg is on the ground moving the robot forward (0b00001111), or the leg is in the air moving back to repeat the cycle (0b11110000). The second input is a counter for the ten steps; this is a 6-bit value where the counter increments in steps of 6 i.e. (6,12,18….60). The third group of inputs are logic zero and one (1-bit each).

The controller has two outputs for each servo motor. The first output is a signed 5-bit value that gives the angular change of the joint for the given step. The second output is a 4-bit prescaler value that is used to scale the magnitude of the angular change. The combination of these two outputs gives possible outputs in degrees of -15 to +15 (prescaler=0) down to -0.9 to 0.9 (prescaler=15).

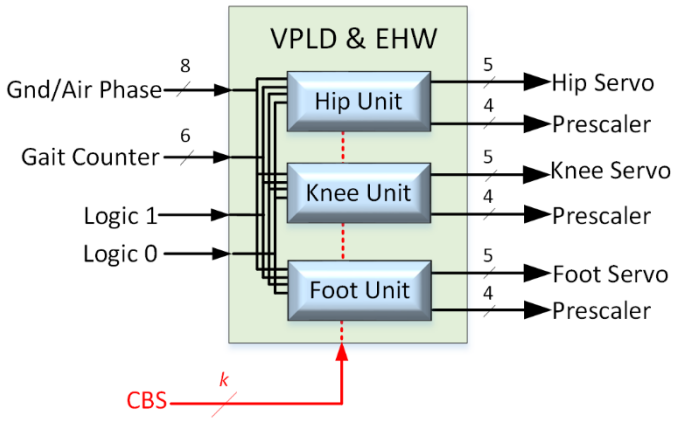

Fig. 4-6 Top-level view of the VPLD & EHW leg controller showing the architectures I/O

Several variations of the VPLD and EHW are investigated consisting of a) a five-layer 8-8-8-8-5 architecture and b) three, two layered architectures 20-5, 10-5 and 5-5 (Fig. 4-7). It is found that the 5-5 architecture is the best, therefore this architecture is discussed in detail below. A comparison of all architectures evaluated can be found in section 6.1.

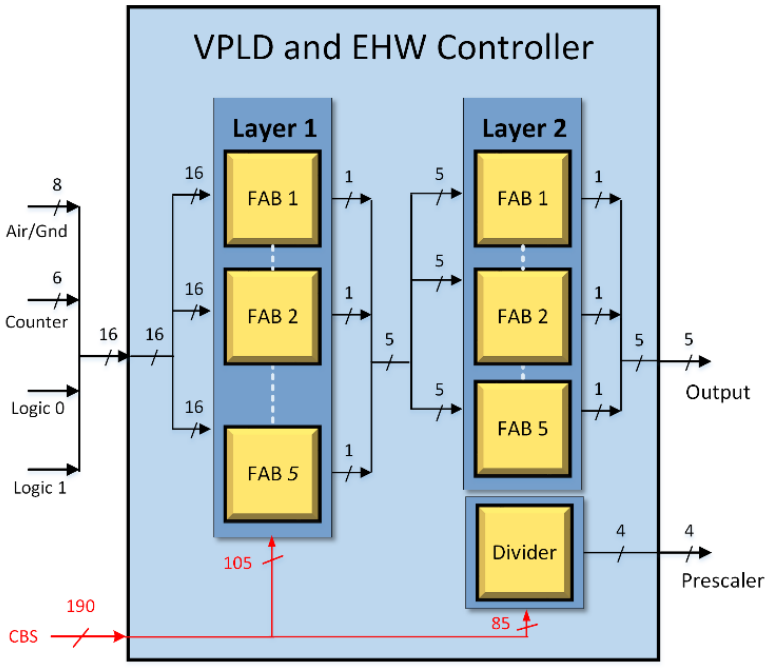

Fig. 4-7 Two-layer (5 by 5) cartesian architecture. Layer one and two both contain 5 FABs

The architecture uses a FAB in the first layer, which contains four 16x1 multiplexers and a function element. Each multiplexer selects 1-bit from the inputs that then gets fed into the function element, the function element is a lookup table (Table 4-1) of 32 logic functions whose 1-bit output is fed into the following layer. Each FAB requires a 21-bit CBS, 16-bits for the multiplexers, and 5-bits for the functional element (Fig. 4-8).

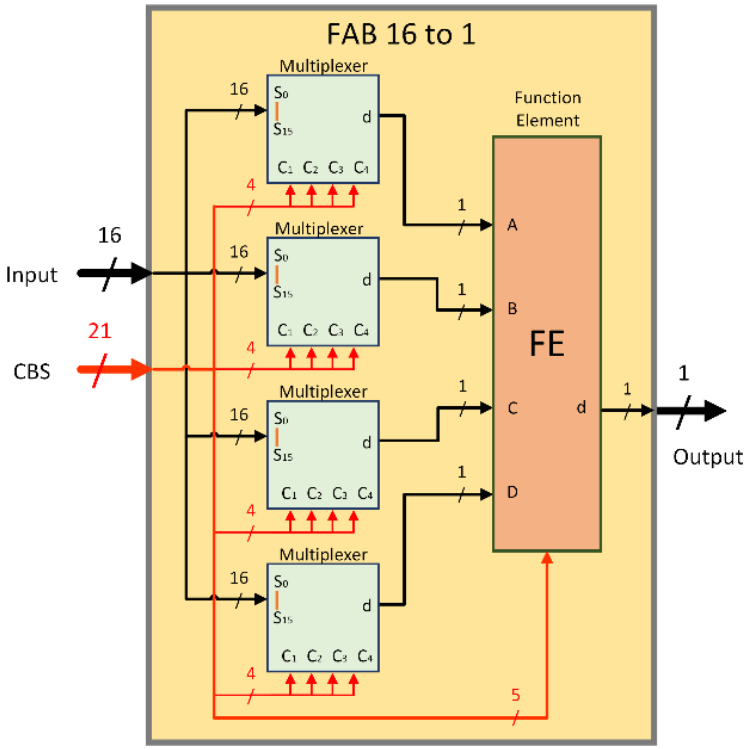

Fig. 4-8 The 16x1 FAB with a 16-bit input, a 1-bit output requiring 21 configuration bits

The second layer FAB uses a 5x1 multiplexer to select one of the 5 outputs from the first layer. The function element is the same as the first layer. The 5x1 FAB (Fig. 4-9) requires 17-bits for configuration, 12-bits for the multiplexers and 5-bits for the functional element. The complete CBS for the 5-5 architecture is 190-bits.

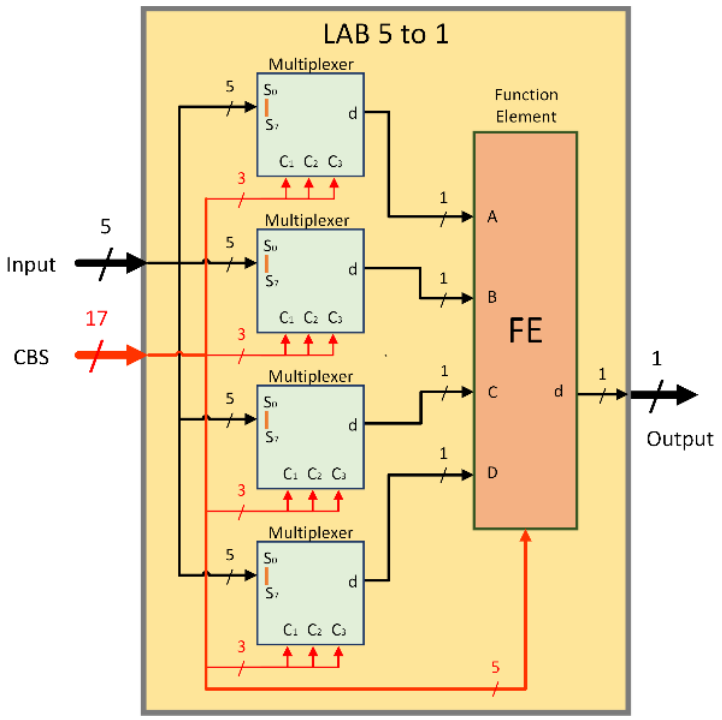

Fig. 4-9 The 5x1 FAB with a 5-bit input, a 1-bit output requiring 17 configuration bits

Table 4-1 VPLD and EHW Function Element LUT

No.	Function	No.	Function
0	A	16	(A&C)^(B&D)
1	B	17	(A&D)^(C&B)
2	C	18	(!A)&B&C&D
3	D	19	A&(!B)&C&D
4	!A	20	A&B&(!C)&D
5	!B	21	A&B&C&(!D)
6	!C	22	(A|B)&(C|D)
7	!D	23	(A|C)&(B|D)
8	A&B&C&D	24	(A|D)&(C|B)
9	!(A&B&C&D)	25	(A^B)&(C^D)
10	A|B|C|D	26	(A^C)&(B^D)
11	!(A|B|C|D)	27	(A^D)&(C^B)
12	(A&B)|(C&D)	28	A&B
13	(A&C)|(B&D)	29	A&C
14	(A&D)|(C&B)	30	A&D
15	(A&B)^(C&D)	31	(A&B)^(C|D)

The output is a 5-bit signed value which is the combination of the five FABs outputs in the final layer. This value is used to determine the angular change in position of a servo motor. In this 5-bit value the MSB represents the sign and bits 3-0 are the magnitude of the angular change. The magnitude of this value is first manipulated by a prescaler ranging from 1/16 to 1 before the change in position is applied to the joint (4.10).

$$\Delta position = \frac{sgn(out[4]) * out[3:0]}{prescaler} \tag{4.10}$$

Chromosome

The VPLD and EHW share the same architecture, thus the chromosomes for both are the same. The chromosome of the VPLD and the EHW is the configuration bitstream. A two-dimensional array of integers is used to store the chromosome of each layer (Fig. 4-10). Each row in the array contains the CBS for one FAB, detailing the configuration of the four multiplexers and the function element. Two of these two-dimensional arrays, (one for each layer) plus a prescaler value make up the complete CBS for each VPLD unit controlling one leg joint. (Fig. 4-11).

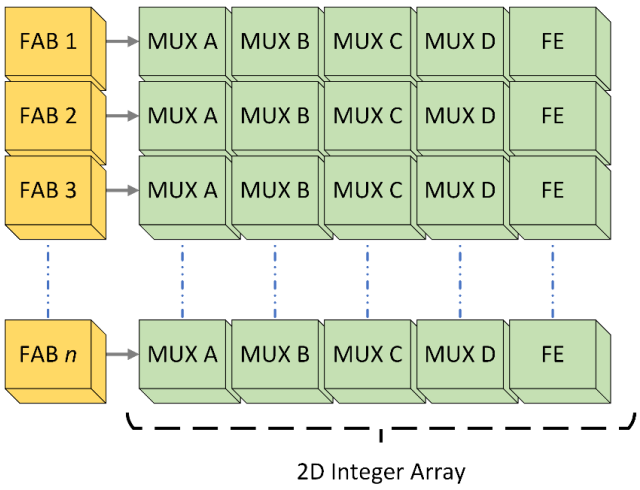

Fig. 4-10 Chromosome of VPLD & EHW layer, 2D Integer Array

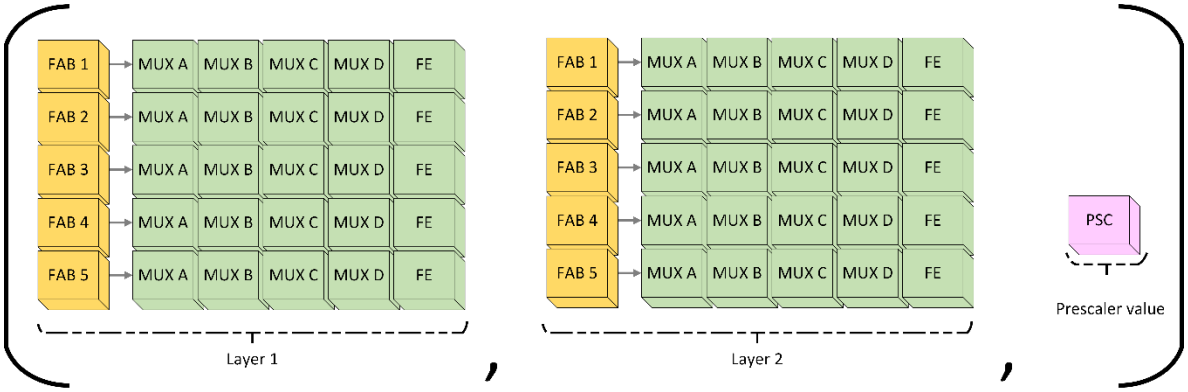

Fig. 4-11 Complete chromosome for one joint in the VPLD & EHW units, showing the chromosome for the *5-5* architecture

For the EHW its CBS is transferred from the GA running on the SoC processor to the EHW through addressed PIO ports. Each bit of the PIO port is connected to a wire in the multiplexer or FE within the FABs. Note the prescaler part of the chromosome is kept inside the SoC processor.

The search space (4.11) for the VPLD and EHW unit CBS using the 5-5 architecture, with each FAB containing 4 multiplexers and a FE with 32 functions, is $1.29 \cdot 10^{53}$.

$$S = (L1_{inputs}{}^{L1_{MUXs}})(L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(L2_{functions}{}^{L2_{FEs}}) \quad (4.11)$$

$$S = (16^{20})(32^{5})(5^{20})(32^{5})$$

$$S = 1.29 \cdot 10^{53}$$

Crossover

Single point crossover is used on each layer in the CBS; this can be considered multipoint crossover when applied to each of the two-dimensional integer arrays that make up the complete chromosome of a VPLD unit (Fig. 4-12). Across the three VPLD units that make up a leg controller the crossover point is independent. A one percent mutation rate is applied to the child chromosomes. For the prescaler value crossover, the child chromosome will get the prescaler value of either parent one or parent two at an equal probability. The prescaler also has a mutation rate of one percent. The crossover and mutation algorithms are constrained to each joint, i.e. the configuration of the knee joint of parent alpha can only be crossed over with the knee configuration of parent gamma.

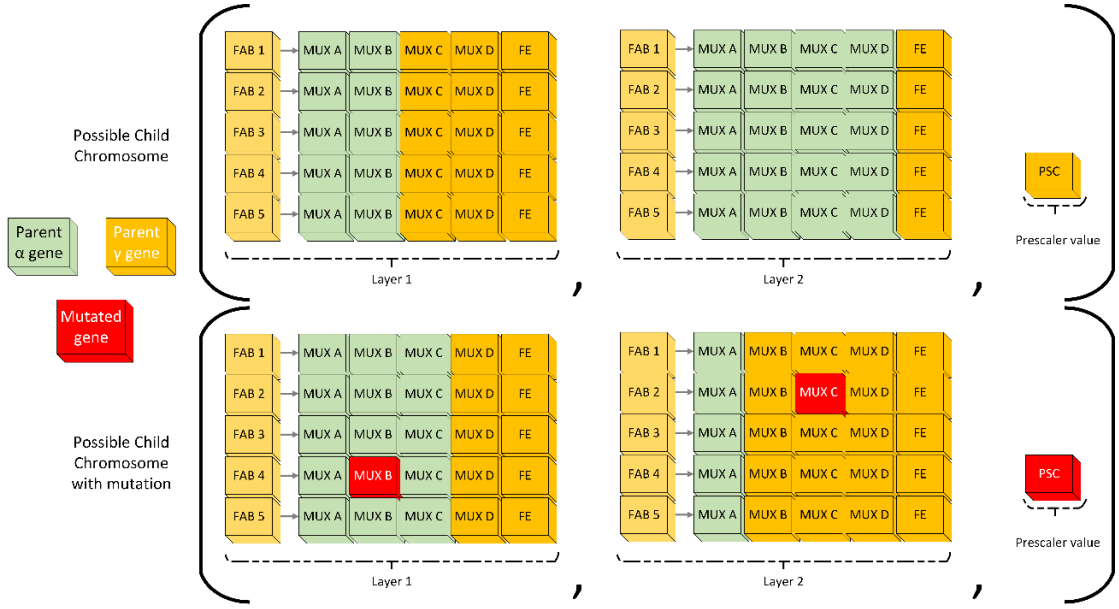

Fig. 4-12 Crossover of VPLD & EHW layer and prescaler, example showing possible children and mutation

4.1.3 Genetic algorithm

The GA (Fig. 4-13) is used to evolve the controllers for the walking gait of the hexapod robot. The gait is comprised of twenty steps split into two phases, 1) a ground phase where the leg is supporting the weight of the robot and moving the body forward maintaining the robots heading; 2) an air phase where the leg is lifted off the ground and returns to the ground phase starting position without affecting the overall motion of the robot. For each phase the servo motors are moved in ten steps. The ground and air phases of the gait are evolved separately using the same GA. The same GA with a population of 100 individuals is applied to all the controllers VPLD, EHW and ANN. The population size of 100 is chosen to ensure a more diverse population particularly early in the evolution process. Reduced population size may improve the computation time of the program per generation but can reduce the evolutionary rate.

Initially all six legs are evolved to develop the gait and performance is 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. The GA process is run until the controllers that have reached an optimal fitness allowing the robot to walk forward in a straight line while maintaining a constant heading and body attitude.

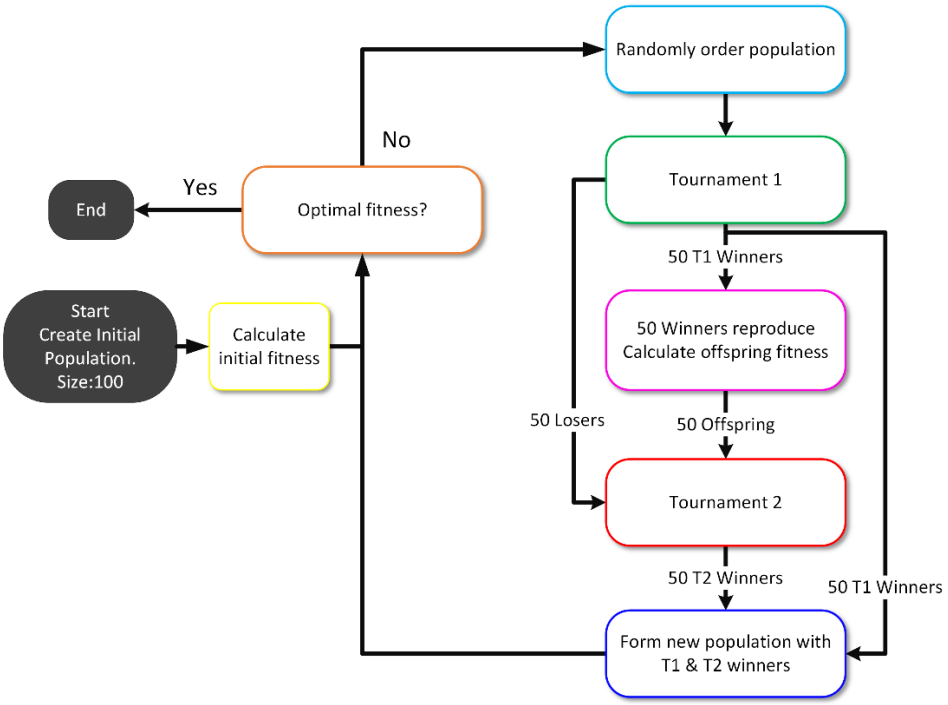

Fig. 4-13 Full Genetic Algorithm using two stage binary tournament selection

Reproduction and Selection

Reproduction uses multi-point crossover with a 1% mutation rate. The selection process uses two stage binary tournament (Fig. 4-14). 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.

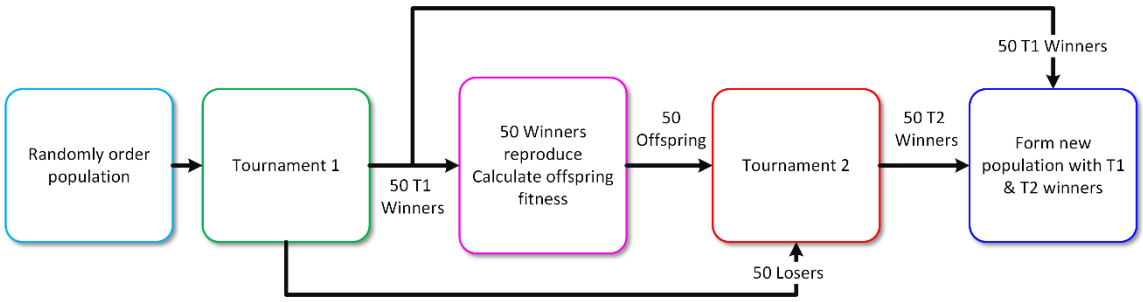

Fig. 4-14 Two stage Binary tournament selection method

Tournament-based selection is 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 [74]. A 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 [67]. 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.

4.1.4 Fitness Evaluation

To reduce the "reality gap" between the robot simulation and the real robot, the fitness functions are linked to the physical robot's observed behaviour including servo motor back-lash, accuracy, and "play". 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.

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) the leg contributing to forward motion; and 4) physical limitations. This is shown in equation (4.12).

$$F_{ground} = F_{symmetry} + F_{forward\,motion} + F_{smoothnesss} + F_{physical\,limits} \tag{4.12}$$

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 which is the opposite sign, implemented using an error squared function. Ideally for example if, Start Position, $\rho = [-80, 25, 20]$ the End Position, $\sigma = [-80, 25, -20]$, equation (4.13).

$$F_{symmetry} = (|\rho_\alpha| - |\sigma_\alpha|)^2 + \left(|\rho_\beta| - |\sigma_\beta|\right)^2 + \left(|\rho_\gamma| - |\sigma_\gamma|\right)^2 \tag{4.13}$$

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 = 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, equations (4.14)-(4.17).

$$F_{forward\,motion} = F_{fm\,1} + F_{fm\,2} + F_{fm\,3} \tag{4.14}$$

$$F_{fm\,1} = |z_0 - z_{avg}|^2 + |z_0 - z_{10}|^2 + \sum_{i=step\,1}^{step\,10} |z_i - z_{avg}|^2 \tag{4.15}$$

$$F_{fm\,2} = |x_0 - x_{avg}|^2 + |x_0 - x_{10}|^2 + \sum_{i=step\,1}^{step\,10} |x_i - x_{avg}|^2 \tag{4.16}$$

$$F_{fm\,3} = \text{fixed penalty, for distance} < LL \tag{4.17}$$

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 (4.18). The vector $\boldsymbol{Foot_i}$ describes the foot position at a step in the gate *i.*

$$F_{smoothness} = \sum_{i=step\ 1}^{step\ 10} \text{fixed penalty, for } \boldsymbol{Foot_i} = \boldsymbol{Foot_{i-1}} \tag{4.18}$$

Penalties are applied when a step in the gait contains angular positions that are not physically possible as defined by equation (4.19). 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 4-2).

$$F_{Physical\ limits} = \begin{cases} \text{fixed penalty, for } \gamma > \max \text{ or } \gamma < \min \\ \text{fixed penalty, for } \alpha > \max \text{ or } \alpha < \min \\ \text{fixed penalty, for } \beta > \max \text{ or } \beta < \min \end{cases} \tag{4.19}$$

Table 4-2 Maximum and minimum joint angle limits

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

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.

$$F_{air} = F_{symmetry} + F_{lift} + F_{smoothnesss} + F_{physical\ limits} \tag{4.20}$$

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), equations (4.21)-(4.23).

$$F_{symmetry} = F_{sym\ 1} + F_{sym\ 2} \tag{4.21}$$

$$F_{sym\,1} = (\rho_\alpha - \sigma_\alpha)^2 + \left(\rho_\beta - \sigma_\beta\right)^2 + \left(\rho_\gamma - \sigma_\gamma\right)^2 \tag{4.22}$$

$$F_{sym\,2} = \text{fixed penalty, for step } n \text{ not } 5, 6 \text{ or } 7 \tag{4.23}$$

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, equations (4.24)-(4.26).

$$F_{lift} = F_{lift\,1} + F_{lift\,1} \tag{4.24}$$

$$F_{lift\,1} = \text{fixed penality, for } z_n < z_0 \tag{4.25}$$

$$F_{lift\,2} = \text{fixed penalty, for } z_n < LL \tag{4.26}$$

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

$$F_{smoothness} = F_{\text{sm }1} + F_{\text{sm }2} \tag{4.27}$$

$$F_{sm\,1} = \sum_{i=step\,0}^{step\,n} \text{fixed penality, for } 0 > |z_i - z_{i-1}| \tag{4.28}$$

$$F_{sm\,2} = \sum_{i=step\,n+1}^{step\,10} \text{fixed penality, for } 0 < |z_i - z_{i-1}| \tag{4.29}$$

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 (4.19).

4.1.5 VPLD and EHW Implementation

The implementation of the EA for the VPLD and EHW follows the outline as described in Chapter 3. The VPLD requires a much simpler EA environment, where all components reside within the same system, the EHW on the other hand relies on multiple systems for the implementation of its EA.

VPLD Implementation

The VPLD implementation (Fig. 4-15) requires a single programming environment with all the code written in Python. Three software scripts are required: 1) the GA program to perform the evolutionary process; 2) the hexapod simulation used to test the fitness of individuals; and 3) the VPLD controller. To test the fitness of an individual, its chromosome is used to configure the VPLD, the VPLD is then used to control the hexapod in the simulation, the motion of the robot is recorded in the simulation to determine the individual's fitness.

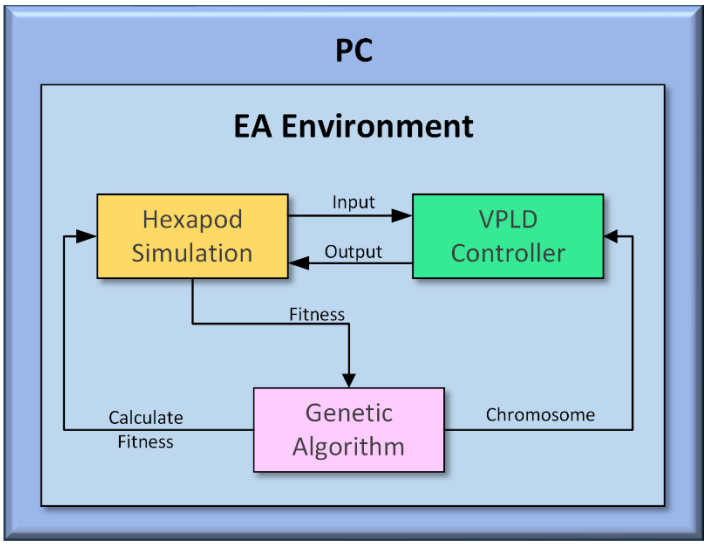

Fig. 4-15 VPLD EA Implementation for the Hexapod control problem

EHW Implementation

The EHW implementation (Fig. 4-16) is more complicated compared to the VPLD, requiring more resources and the knowledge of both software and hardware programming languages. The whole EA process is conducted on a DE10 Nano board which incorporates a Cyclone V FPGA with a SoC ARM processor. Embedded Linux is used on the SoC processor. The GA and hexapod simulation are Python scripts similar to those used by the VPLD and are executed on the SoC processor. The EHW IP is written in Verilog using Altera's Quartus IDE, also requiring Platform designer to design the architecture of the system including the HPS, clocks, PIO ports and the associated interconnections.

An interface between the SoC processor and EHW is written in software to use the AXI lightweight bridge, which is the interface between the hard processor and the FPGA fabric within the Cyclone V chip. SSH interface is used for monitoring the EA process and recording the results.

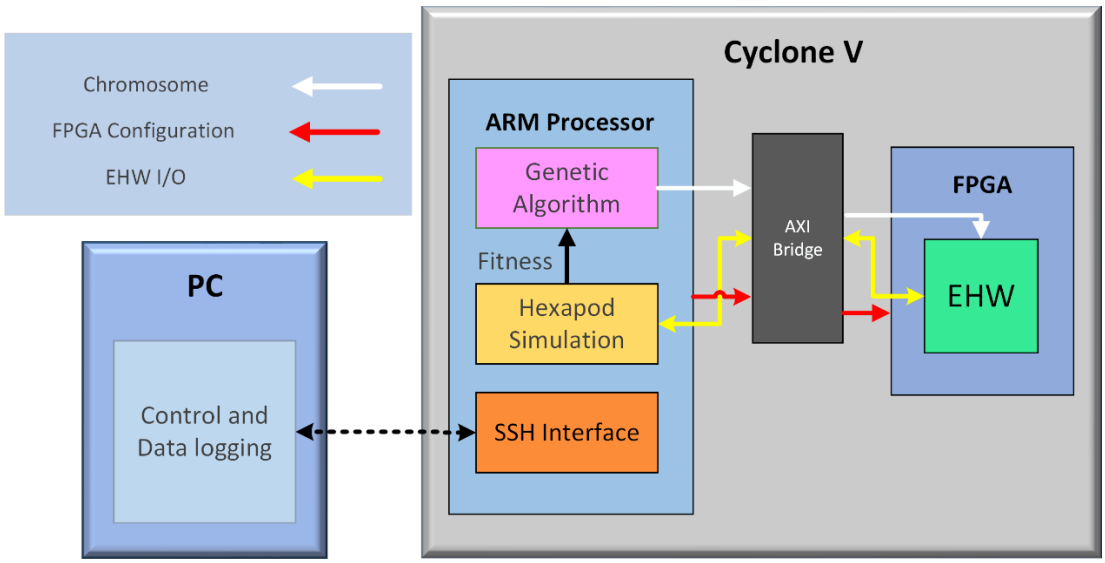

Fig. 4-16 EHW EA Implementation for the Hexapod control problem

As discussed previously, due to the system complexity, its challenging for the researcher to make alterations to the EA environment because a change could require alterations to the system at multiple levels. The complexity poses a risk of introducing errors into the system with multiple different components all required to carry out a single operation, including: a) the custom IP defining the EHW; b) the program interface the processor and FPGA; c) the GA code managing the chromosomes of the individuals; and d) the system architecture defined in platform designer. If there are errors in any of these elements the EHW will not be configured correctly and may require extensive debugging and investigation into all the elements involved to find the error.

4.1.6 Results: VPLD and EHW Comparison

The experiments performed to evaluate the controllers are split into two parts; the first is to determine which architecture for the individual VPLD and EHW controllers are the best; and the second is to compare the performance between the best evolved controllers. In this section the best controller architectures are compared, the results of the other architectures evaluated can be found in section 6.1.

The controllers are evolved to produce an optimal walking gait for a hexapod robot using the same GA. The walking gait allowed the robot to walk forward in a straight line, maintaining a constant heading and body attitude. The leg motion is divided into two phases; the leg on the ground moving the robot forward; and the leg in the air returning the leg to the starting position. Each phase is evolved separately, with the leg motion of each phase broken into ten steps. Three hundred solutions for each controller are evolved, with the maximum number of generations limited to 500 for each phase. All 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.

The parameters used to evaluate the controllers are: 1) evolutionary efficiency, (how many generations and the computation time required to reach a solution); 2) the controller performance, (the ability of the deployed controllers); and 3) the propagation delay between the input and output of the controller.

Three platforms are used to implement the controllers (Table 4-3).

Table 4-3 Development platforms processor properties

Platform	Controllers Tested on Platform	Hardware Description
Terrassic DE10 Nano	EHW	Cyclone V FPGA Dual-Core Cortex-A9 SoC hardcore processor (925 MHz).
Desktop PC	VPLD	Intel i7-11700k Octa-Core processor (3.6 GHz)
Raspberry Pi 4 Model B	VPLD	Quad-Core Cortex-A72 64-bit SoC (1.5GHz).

Evolutionary Efficiency

The graph of the fitness per generation (Fig. 4-17) for the VPLD and EHW shows the improvement of controller performance (fitness) per generation and the evolutionary efficiency (number of generations to reach a solution). The EHW and VPLD results are very similar, as was previously stated this is because the GA, controller architecture and chromosome are identical. Any differences in the results are due to the random initial population, crossover, and mutation.

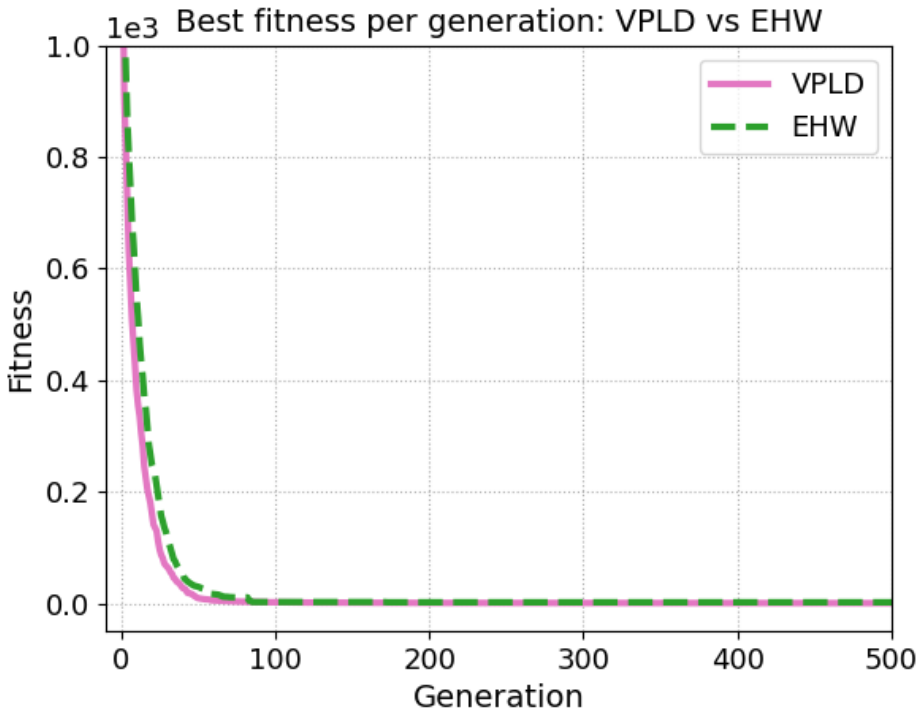

Fig. 4-17 Comparison of VPLD and EHW best fitness per generation averaged over 300 runs

A table showing the number of generations (Table 4-4), shows that both the controllers could obtain the required fitness in a small number of generations. Both the VPLD and EHW required less than 80 generations to evolve on average.

Table 4-4 VPLD and EHW controller number of generations. Average, Median and Coefficient of variance

Controller	Mean	Median	CV
VPLD	68.63	61	0.43
EHW	79.04	71	0.42

The time taken to evolve solutions is shown in detail as a violin plot analysis (Fig. 4-18). The evolution time of the EHW is two orders of magnitude slower than the VPLD (Table 4-5). This time difference is due to the different platforms the GA must run on; with the VPLD running on a PC and the EHW running on the SoC ARM processor on the Cyclone V FPGA. Although it is possible to run the GA and leg simulation on a PC and download the CBS to the EHW using a serial link, it can be shown that a serial link running at 115k, will produce a bottle neck, requiring 1.6ms for the CBS to be transferred to the EHW, (1.6s to test a population of 100, or 130 seconds for a typical number of 80 generations to evolve). Therefore, running the GA on the PC and communicating via a serial-link would not improve the time required to evolve a solution.

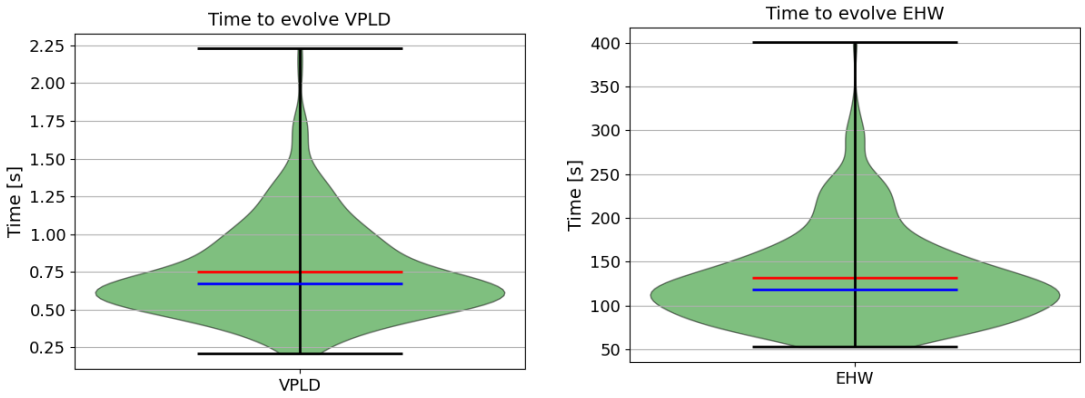

Fig. 4-18 Violin Plot of Average Time to Evolve. Shows distribution of results, Maximum/Minimum time, median(blue line) and mean(red line)

Table 4-5 VPLD and EHW time to evolve. Average, Median and Coefficient of variance

Controller	Mean[s]	Median[s]	CV
VPLD 5-5	0.75	0.67	0.41
EHW 5-5	131.28	118.13	0.41

The GA for the VPLD is also evaluated on the Raspberry Pi. This platform is more comparable to a real-world robotic system where the online adaptation of the controllers could be used for fault recovery of deployed robots. The VPLD could be evolved on the Raspberry Pi in 6.95s on average, which is slower than on the PC but is significantly faster than the EHW.

Controller Performance

The controller performance is judged on the ability of the robot to walk forward in a straight line while maintaining a constant heading and body attitude. To observe the performance of the evolved controllers, first they are run on the PyBullet full simulation of the hexapod. To assess the controllers in the full simulation they are run for a set time with the trajectories and height of each recorded. A video of the controllers running in full simulation can be observed in Supplementary Resource 1. The trajectory of the robot in the world coordinate frame, (Fig. 4-19) shows that both controllers provide an excellent gait, moving the robot forward in a straight line while maintaining a constant height. It can be seen (Table 4-6) that the controller performance of both the evolved controllers are very similar.

Table 4-6 VPLD and EHW simulation performance. Looking at distance travelled, deviation from a straight path, average body height, and average body attitude

Controller	Distance [m]	Path Deviation [cm]	Height [cm]	Roll [°]	Pitch [°]	Yaw [°]
VPLD	1.01	0.03	16.8	0.01	0.19	0.17
EHW	1.00	0.25	16.8	0.01	0.15	0.73

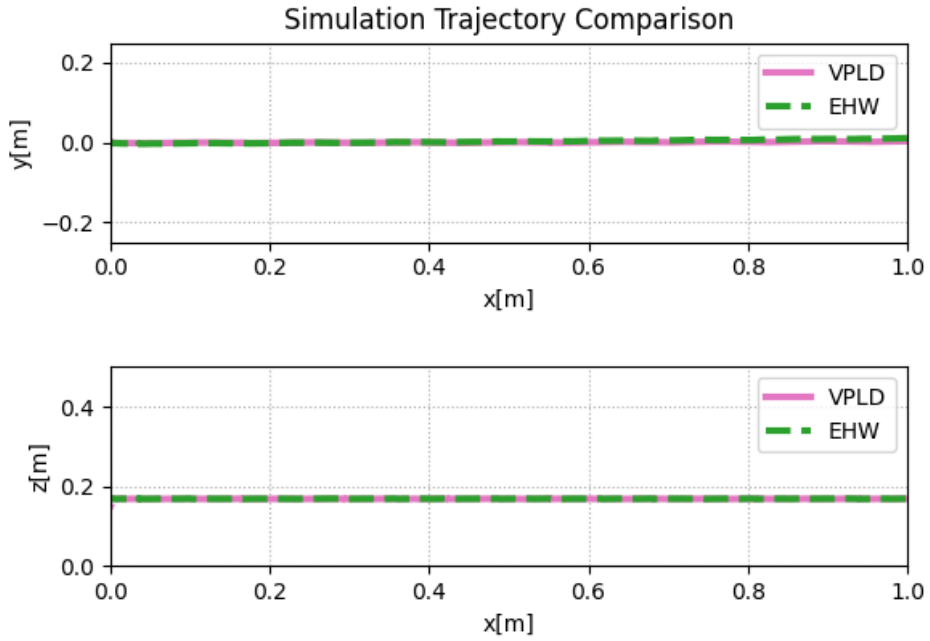

Fig. 4-19 Robot trajectory in full simulation, world coordinate system

The attitude of the hexapod in reference to the robots coordinate frame is shown in Fig. 4-20. The hexapod maintains a constant orientation with negligible oscillation that has no impact on the performance of the robot's gait.

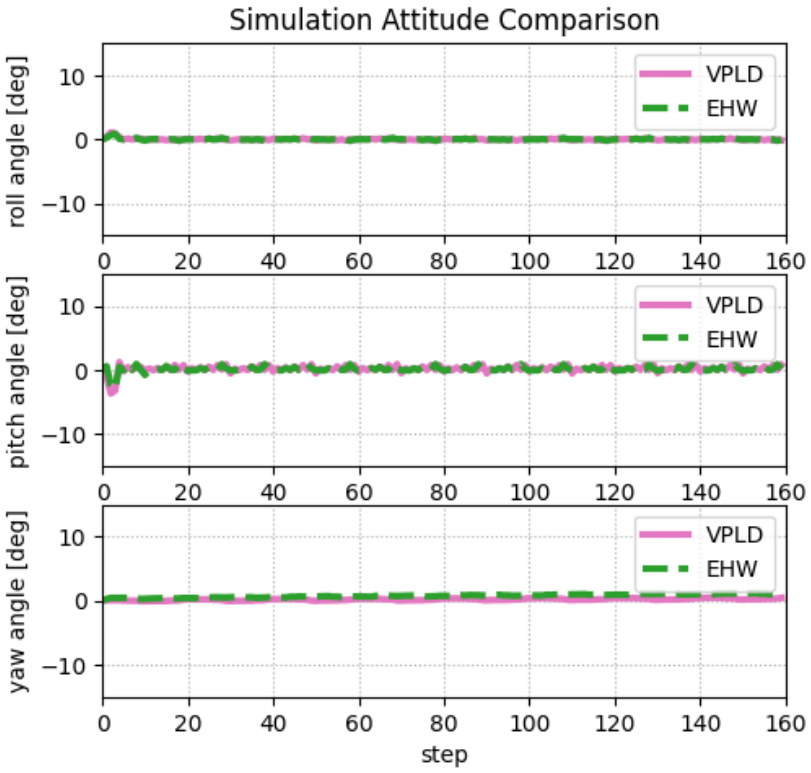

Fig. 4-20 Robot attitude in full simulation

Finally, the controller performance is observed on the real robot as described in section 4.1.1. The robot used is an off the shelf chassis with custom electronics designed by the authors at AUT for experimentation in the field of evolvable robotic controllers. The hardware included a Terasic DE0-Nano FPGA development board which has 32MB of SRAM and a 50MHz clock. A NIOS II softcore processor is created within the FPGA, which is used to directly run the VPLD controllers, and to download the CBS to the EHW controller. A hardware PWM generator (coded in Verilog) is interfaced to the NIOS II processor via a standard PIO port, and this is connected to the servo motors. The VPLD FPGA configurations required approximately 7,000 total logic elements, while the EHW required approximately 20,000 total logic elements. The EHW implementation required more logic elements for a) the NIOS processor due to the extra PIO required to drive the CBS and EHW I/O; and b) the EHW IP itself.

Table 4-7 VPLD and EHW Implementation Resource usage

System Component	VPLD Logic Elements	EHW Logic Elements
Configured NIOS II	6122	12799
PWM servo controllers	395	395
EHW EHW	N/A	5967
NIOS II JTAG interface controller	313	314
Total	6830	19475

The performance of the controllers implemented on the physical robot (Fig. 4-21) can be observed in Supplementary Resource 2. The physical implementation has the same traits and performance as is seen in the full simulation, again meeting the optimal gait criteria.

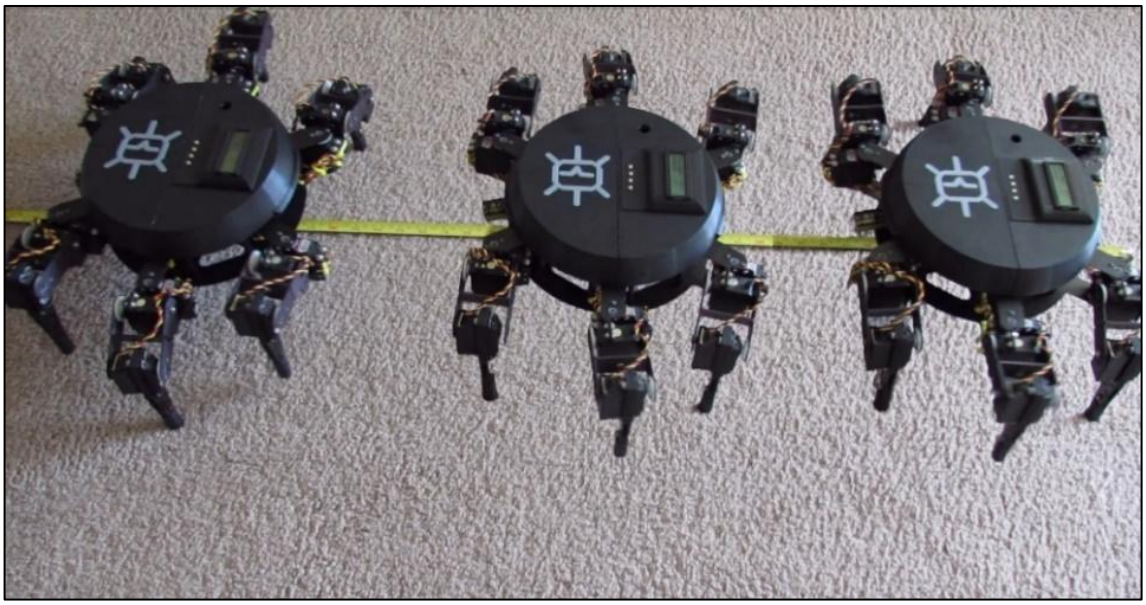

Fig. 4-21 Real Hexapod Controller Implementation

The board has a Bluetooth interface allowing the robot to receive commands to walk forward and backward as well as rotating left and right. The input signals of the evolved controllers can be adjusted to achieve the different motions; walking backwards is achieved by decrementing the step counter; and rotation can be achieved by incrementing the counter for the legs on one side and decrementing the counter for the legs on the opposite side.

Controller Propagation Delay

The propagation delay of these controllers must be within the timing constraints of the servo motors on the physical hexapod robot, which has been set at an 80ms time interval between each step in the gait. In this experiment the EHW is tested on the DE10 Nano, the VPLD is tested on both a PC and a Raspberry Pi 4B. The propagation delay of both the controllers (Table 4-8) is well within the range of the robot requirements.

Table 4-8 Propagation delay with the VPLD running on a PC and Raspberry Pi, and the EHW running on the DE10.

Controller	Time[us]
VPLD (PC)	27.2
EHW	473
VPLD (Pi)	156

The EHW has a longer propagation delay due to the time required to interface the I/O from the ports on the SoC processor to the EHW on the FPGA. This could be rapidly decreased by replacing the SoC processor with a hardware implementation of: a) the gait driver, including a step counter, and air/ground phase generator and b) the PWM generator including the prescaler. The EHW hardware itself only requires two clock cycles(40ns) to execute an input, given the minimal extra clock cycles required for the operation of the gait driver and PWM generator the complete hardware implementation would be significantly faster in the order of magnitude of nanoseconds opposed to microseconds.

Power Consumption

Although power consumption was not a primary design objective of the VPLD, the experimental results obtained in this chapter allow a practical comparison of energy usage between processor-based VPLD implementations and FPGA-based evolvable hardware. For a fair comparison, both instantaneous power consumption and total execution time must be considered, as the latter has a dominant influence on overall energy expenditure during evolutionary optimisation.

When executed on a Raspberry Pi platform operating at 1.8 GHz under high computational load, the processor consumes approximately 12.3 W. The average time required to complete an evolutionary run in this configuration is 6.95 s, resulting in an energy consumption of approximately 85.5 J per run. In contrast, the DE10-Nano platform, incorporating an embedded ARM processor operating at 800 MHz and an

FPGA-based evolvable hardware implementation, consumes approximately 1.3 W in total. This includes the processor, FPGA fabric, peripheral I/O, clocking resources, and a utilisation of approximately 20,000 logic elements. However, the significantly longer evolution time of 131.28 s results in an energy consumption of approximately 170 J per run.

A similar trend is observed for the desktop processor used. An Intel i7-11700K processor, operating within a typical power range of 125 W to 190 W, completes equivalent evolutionary runs in approximately 0.75 s. This corresponds to an energy consumption in the range of approximately 93 J to 142 J per run. Despite the higher peak power consumption, the substantially reduced execution time results in energy usage comparable to, or lower than, that of the FPGA-based evolvable hardware platform.

These results demonstrate that, for the evolutionary robotics experiments investigated in this chapter, total energy consumption is dominated by execution time rather than instantaneous power draw. While FPGA-based systems offer lower power operation, their extended optimisation times lead to higher overall energy expenditure. Conversely, the VPLD benefits from faster execution on processor-based platforms, resulting in reduced or comparable energy consumption per evolutionary run. This supports the conclusion that, for the applications considered in this thesis, the VPLD provides an effective balance between performance, flexibility, and energy efficiency without explicitly optimising for low-power operation.

4.1.7 Conclusions: VPLD and EHW Comparison

This section has demonstrated the VPLD for use as a hexapod robotic controller. The VPLD performance and its benefits to the evolution processes are evaluated in comparison to the EHW from which it is derived. The VPLD and EHW architectures were evolved to produce the gait of a hexapod robot and a comparison was made between each controller. The controller ability was determined by: 1) evolutionary efficiency, 2) the controller performance, and 3) the propagation delay. As expected, the number of generations required to evolve the VPLD and EHW are very similar as they share the same architecture, chromosome and used the same GA. However, the EHW was two orders of magnitude slower to evolve than the VPLD, due to running the GA on the SoC processor. It was shown that even if the GA was run on a PC, the communication between the PC and FPGA would be a bottleneck and have a similar time to evolve. It was also demonstrated that the implementation of the EA environment for the VPLD was much simpler than the EHW implementation. For the VPLD, all components reside within the same system, the EHW on the other hand relies on multiple systems for the implementation of its EA, this increases the complexity and adds greater challenges for debugging the system.

When implementing the evolved controllers in the PyBullet simulation, it was found that they produced an excellent robot gait that met the optimal gait criteria. The evolved controllers are then implemented on the real robot, where it was found the controllers operated in the same manner as observed in the PyBullet simulation.

The main benefits of the VPLD shown by the experiments conducted in this section are: 1) the VPLD was significantly faster to evolve compared to the EHW; and 2) the VPLD could be executed on a standard PC, an ARM based embedded system such as a Raspberry Pi, or microcontrollers such as the ESP32, and 3) required less resources for the genetic algorithm implementation.

4.1.8 ANN Control System Architecture

The ANN uses the same inputs as the VPLD; which are, the air/ground phase and counter signals. The air/ground phase signal has two states 1 for ground and -1 for air. The counter has ten steps for each phase of the gait changing in value from 0 to 0.9 in steps of 0.1. The output of the ANN is the angular change in position of the three joints in the leg of a hexapod, ranging from -15 to +15. A three layer ANN with 5 neurons is evaluated (2 in the hidden layer, 3 in the output layer). The results of other architectures that are tested can be found in Appendix A.

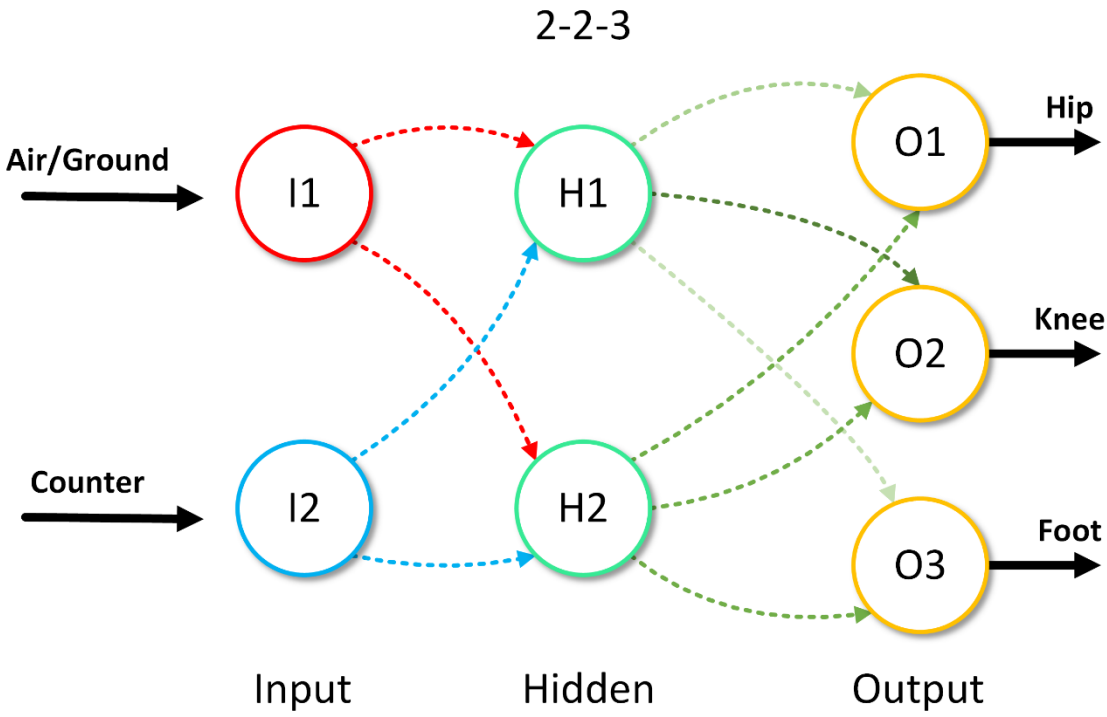

Fig. 4-22 ANN *2-2-3*, two inputs, two hidden layer neurons, three output neurons

The neurons in the hidden layer (Fig. 4-23) use a hyperbolic tangent activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01, giving an input range into the activation function of ±2.9. Equations (4.30) & (4.31) describes the hidden layer neuron functionality.

$$x = \left[\sum_{i=1}^{n} W_i * I_i \right] + B \tag{4.30}$$

$$f(x) = \left(\frac{1 - e^{-2x}}{1 + e^{-2x}} \right) \tag{4.31}$$

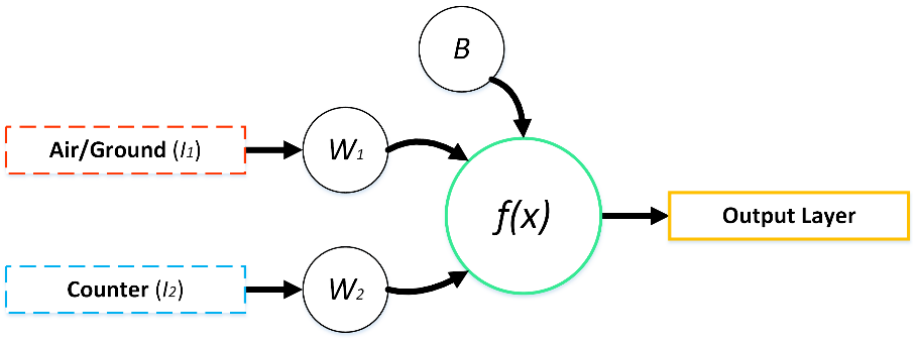

Fig. 4-23 Hidden layer neuron

The output layer neurons (Fig. 4-24) use the same weights and biases as the hidden layer ranging from -1 to +1 in 0.01 steps, this gives an input range of ±3. The activation function of these neurons is the hyperbolic tangent function multiplied by fifteen, which results in the ±15 output range. Equations (4.32) & (4.33) describes the output layer neuron functionality.

$$y = \left[\sum_{i=1}^{n} W_i * H_i \right] + B \tag{4.32}$$

$$g(y) = 15\left(\frac{1 - e^{-2y}}{1 + e^{-2y}}\right) \tag{4.33}$$

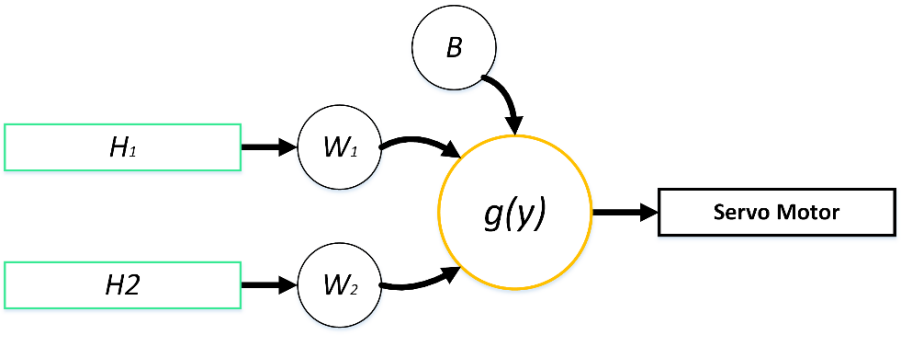

Fig. 4-24 Output Layer neuron

Chromosome

The chromosome of the ANN is the weights and biases. A two-dimensional floating-point array is used to store the chromosome of one layer (Fig. 4-25). A set of these arrays make up the complete configuration for the neural network (Fig. 4-26). As is shown in Fig. 4-25 each row contains the complete configuration for a single neuron in that layer, columns are the configurations of the different elements that make up a neuron, being the weights for each input from 1 to *m* inputs and the bias.

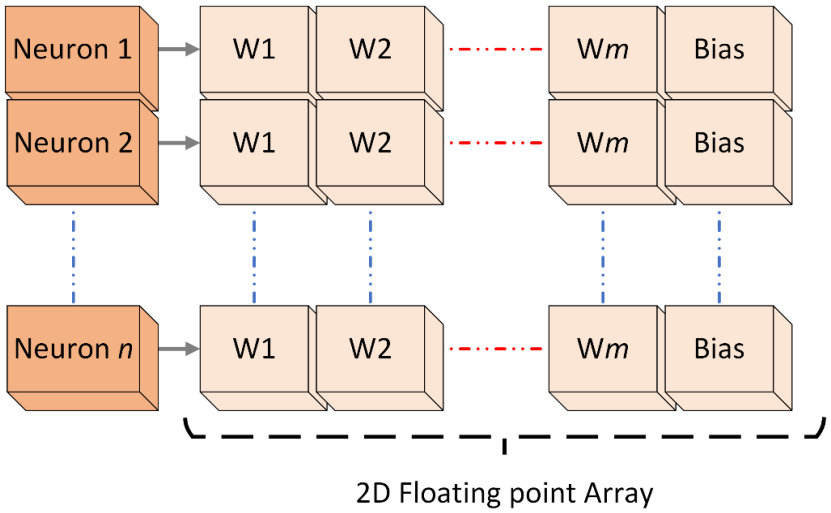

Fig. 4-25 Chromosome of ANN layer, 2D Floating point Array

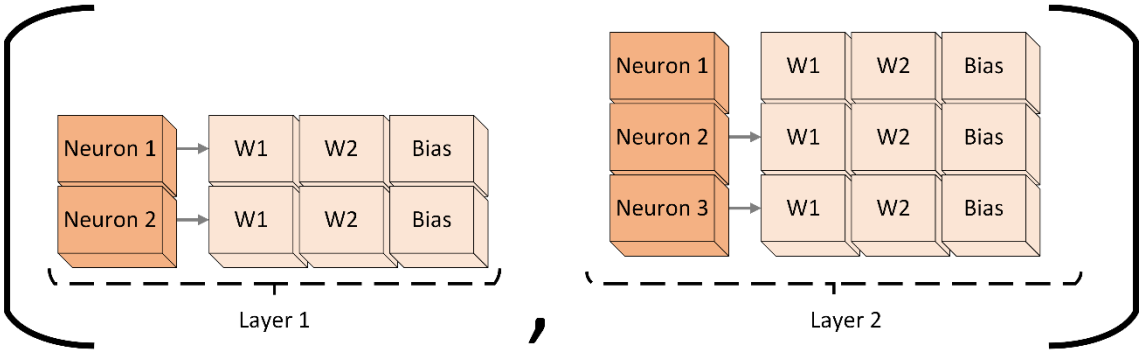

Fig. 4-26 Complete chromosome for an ANN controller

The search space (4.34) for the ANN can be calculated as follows. The hidden layer component of the chromosome is an array of size 2x3, and for the second layer an array of size 3x3; this gives a search space of $3.53 \cdot 10^{34}$.

$$S = (no.of\ values^{H_{weights+bias}+O_{weights+bias}}) \tag{4.34}$$

$$no.of\ values = 1 + \frac{max - min}{increment\ size} = 1 + \frac{1 - -1}{0.01} = 201$$

$$S = 201^{6+9}$$

$$S = 3.53 \cdot 10^{34}$$

Crossover

The crossover of the ANN chromosomes during the reproduction phase of the GA is a single point crossover of each layer's weights and biases, this can be considered a multipoint crossover when applied to each of the two-dimensional floating-point arrays that make up the complete chromosome (Fig. 4-27). For the mutation of the chromosome, a mutation rate of one percent is applied to the child chromosomes.

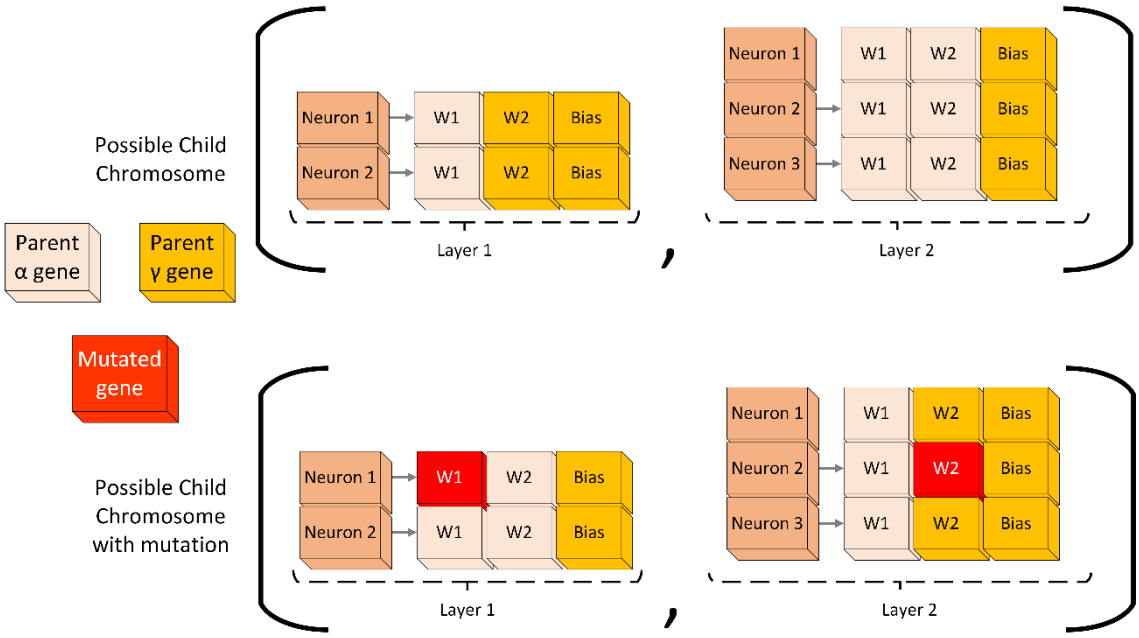

Fig. 4-27 Crossover of an ANN chromosome. Example showing possible children and mutation

4.1.9 ANN Implementation

The ANN implementation (Fig. 4-28) is implemented in software using Python. Three scripts are used: 1) the genetic algorithm program to perform the evolutionary process; 2) the robot leg simulation; and 3) the ANN controller. The ANN and VPLD implementation is essentially identical, except for the ANN and VPLD controller software.

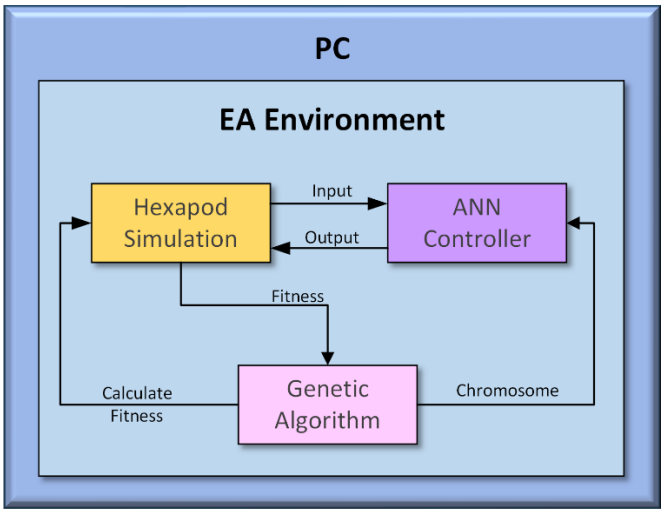

Fig. 4-28 Hexapod ANN EA Environment

4.1.10 Results: VPLD and ANN Comparison

A comparison is made between the VPLD architecture from section 4.1.1 and the ANN architecture. Three hundred solutions for each controller are evolved, with the maximum number of generations limited to 500 for each phase. All 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. The same parameters from section 4.1.6 are used to evaluate the controllers: 1) the controller performance; 2) evolutionary efficiency and 3) the propagation delay between the input and output of the controller. Two platforms; a desktop PC and a Raspberry Pi-4 are used to implement the controllers (Table 4-9).

Table 4-9 VPLD and ANN Development platforms processor properties

Platform	Controllers Tested on Platform	Hardware Description
Desktop PC	VPLD and ANN	Intel i7-11700k Octa-Core processor (3.6 GHz)
Raspberry Pi 4 Model B	VPLD and ANN	Quad-Core Cortex-A72 64-bit SoC (1.5GHz).

Evolutionary Efficiency Comparison

The evolutionary efficiency of each controller is compared. The graph of the fitness per generation (Fig. 4-29), and a table showing the number of generations (Table 4-10), show that the ANN like the VPLD could obtain the required fitness in a small number of generations, but the VPLD had a higher initial evolutionary rate and required less total generations than the ANN to evolve.

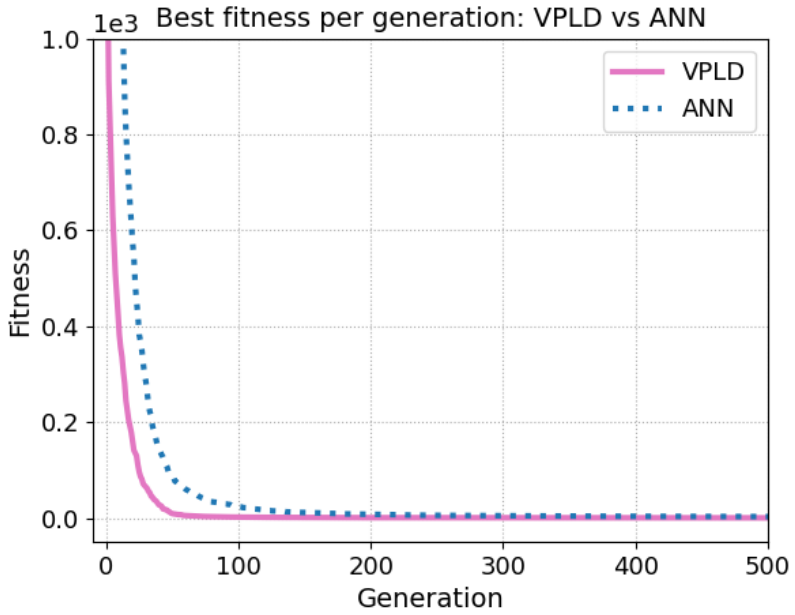

Fig. 4-29 VPLD and ANN controller best fitness per generation averaged from 300 evolution attempts

Table 4-10 VPLD and ANN controller number of generations. Average, median and coefficient of variance

Controller	Mean	Median	CV
VPLD	68.63	61	0.43
ANN	132.97	117	0.5

The ANN and VPLD have a comparable evolution time (Table 4-11). Even though the ANN takes more generations to evolve than the VPLD it has a comparable evolution time because the size of the ANN chromosome is smaller, requiring less computation time to process the reproduction stage of a GA generation. A violin plot analysis of the time to evolve is shown in Fig. 4-30, this shows the distribution of the 300 results for each controller. It can be seen that the VPLD approach is slightly more consistent than the ANN, this is shown quantitatively by the CV value for both controllers.

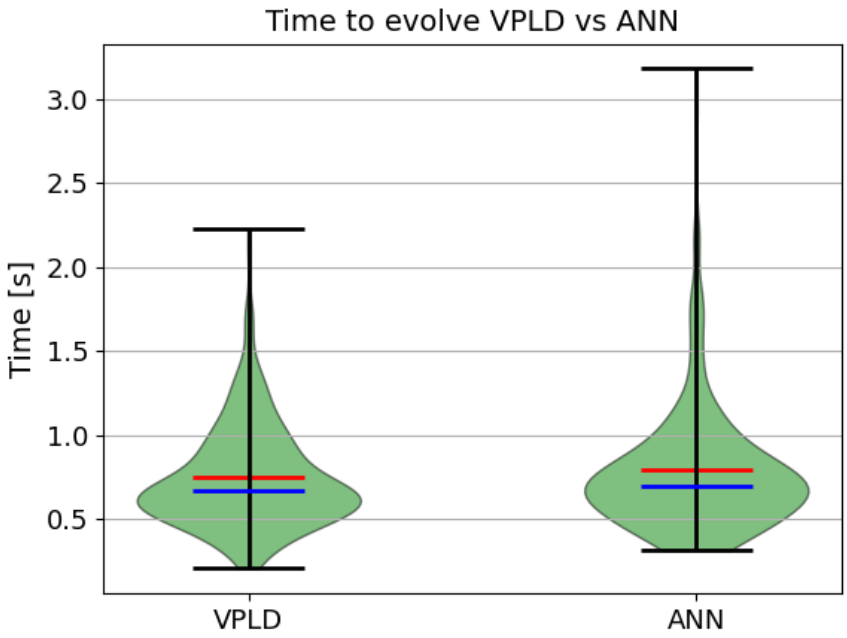

Fig. 4-30 Violin Plot of Average Time to Evolve. Shows distribution of results, Maximum/Minimum time, median(blue line) and mean(red line)

Table 4-11 VPLD and ANN time to evolve. Average, median and coefficient of variance

Controller	Mean[s]	Median[s]	CV
VPLD	0.75	0.67	0.41
ANN	0.79	0.7	0.49

The GA for the ANN is also evaluated on the Raspberry Pi as previously tested with the VPLD. This platform is more comparable to a real-world robotic system where the adaptation of the controllers could be used in fault tolerant controllers. Both architectures have similar performance.

Table 4-12 Average time to evolve on a Raspberry Pi for the VPLD and ANN

Controller	Mean[s]
VPLD	6.95
ANN	7.82

Simulation Robot Deployment

The controller performance is judged on the ability of the robot to walk forward in a straight line while maintaining a constant heading and body attitude. As a measure of the controller performance between the VPLD and ANN, the ANN controller is implemented in the PyBullet simulation, both the controllers walk in a straight line, maintaining a constant heading and body attitude (Fig. 4-31 & Fig. 4-32). There is negligible difference in the ANN and VPLD performance (Table 4-13).

Table 4-13 VPLD and ANN simulation performance. Looking at deviation from a straight path, average body height, and average body attitude

Controller	Distance [m]	Path Deviation [cm]	Height [cm]	Roll [°]	Pitch [°]	Yaw [°]
VPLD	1.01	0.03	16.8	0.01	0.19	0.17
ANN	0.96	0.63	16.8	0.00	0.15	0.57

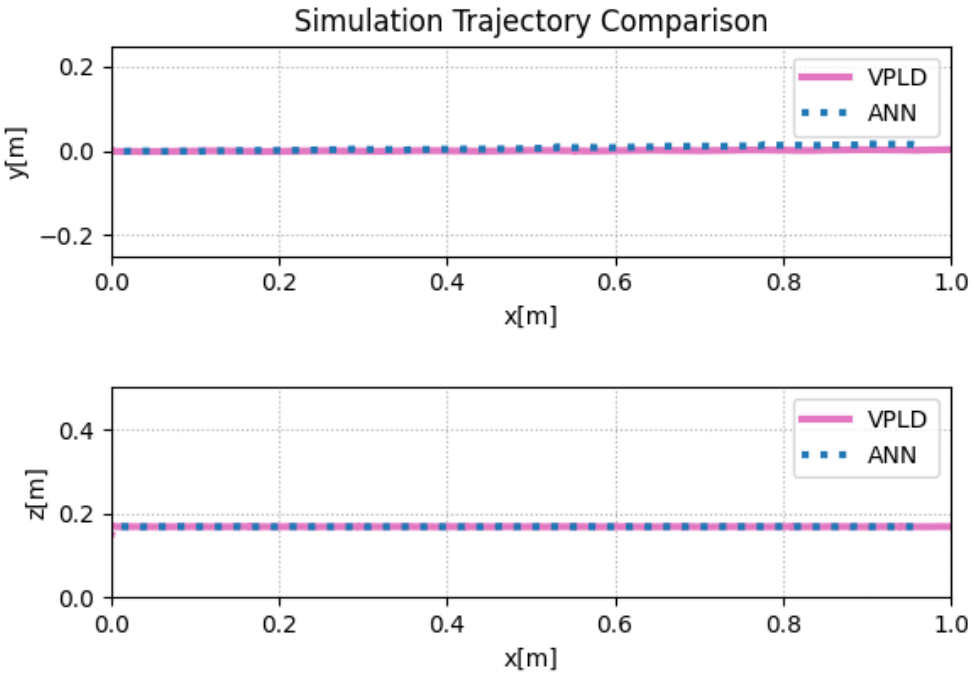

Fig. 4-31 Robot trajectory in full simulation, world coordinate system

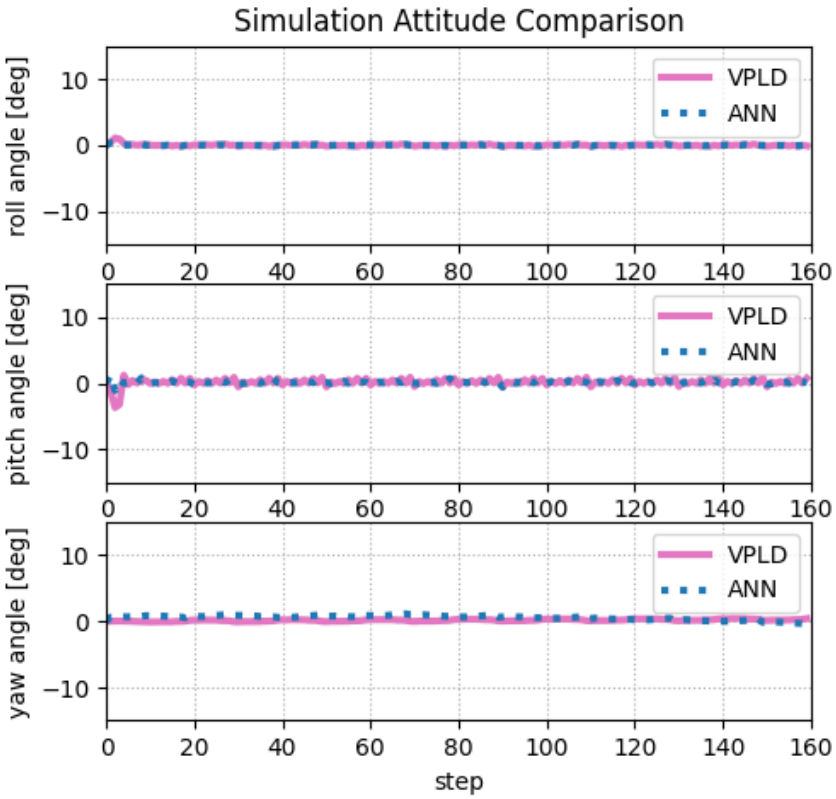

Fig. 4-32 Robot attitude in full simulation

Finally, the controller performance of the ANN and VPLD are implemented on the real robot, running on a NIOS II softcore processor. A video of the performance of the ANN and VPLD controllers implemented on the physical robot can be observed in Supplementary Resource 2. The physical implementation of the ANN and VPLD perform equally well, and the deployed controllers have the same traits and performance as was seen in the full simulation.

Fig. 4-33 ANN Implemented on Hexapod Robot

Controller Propagation Delay

The propagation delay of these controllers must be within the timing constraints of the servo motors on the physical hexapod robot, which has been set at an 80ms time interval between each step in the gait. In this experiment the VPLD and ANN controllers are tested on both a PC and a Raspberry Pi 4B. The propagation delay of both the controllers (Table 4-14) is under 1ms which is well within range of the robot requirements. The ANN controller is faster than the VPLD, which can be contributed to three factors: 1) the reduced size of the ANN architecture; 2) the reduced complexity of the operations required to process a neuron versus a FAB; and 3) the lack of pre-processing required to generate binary values for the input and post-processing to convert the binary data to values suitable to drive the servos.

Table 4-14 Propagation delay of the ANN & VPLD running on a PC and the Raspberry Pi.

Controller	Time[ms]
VPLD (PC)	0.0272
VPLD (Pi)	0.1156
ANN (PC)	0.0065
ANN (Pi)	0.0381

4.1.11 Conclusions: VPLD and ANN Comparison

This section has compared the VPLD and ANN for use as a hexapod robotic controller. The VPLD and ANN architectures are evolved to produce the gait of a hexapod robot and a comparison was made between each controller. The controller's ability was determined by 1) evolutionary efficiency, 2) the controller performance, and 3) the propagation delay.

The ANN required approximately 60 generations more on average to evolve compared to the VPLD. However the time to evolve the two architectures was similar, this was because the size of the ANN chromosome is smaller, requiring less computation time to process the reproduction stage of a GA generation, also as was seen the ANN propagation delay was less meaning the time to run the simulation for ANN is less, which affects the overall time to evolve. Both Controllers could be evolved in a short amount of time (less than 8 seconds) on a Raspberry Pi. Platforms such as the Raspberry Pi or Nvidia Jetson Nano Boards are used commonly in robotics research, being able to run the GA on these platforms in a short amount of time would be useful in fault tolerance application, where robots deployed in harsh environments could adapt to faults such as limb damage or sensor faults.

When implementing the evolved controllers in the PyBullet simulation, it was found that they the VPLD and ANN produced gaits with a comparable performance. The evolved controllers are also implemented on the real robot, where it was found the controllers operated in the same manner as observed in the simulation.

4.2. 2WD Mobile Robot Control

The two-wheel differential drive mobile robot platform is used to assess the VPLD against an ANN when used for robotic navigation. VPLD and ANN robotic controllers are evolved to optimally control the robot for three tasks: 1) obstacle avoidance, 2) light following, and 3) light following while avoiding obstacles using a monolithic control architecture. For the VPLD both a digital logic-based architecture (D-VPLD) and a floating-point architecture that uses mathematical functions (F-VPLD) is investigated. All the controllers are evolved using the same GA. An evaluation is made on the evolutionary efficiency and controller performance of the architectures detailed in the following sections.

The three robotic controllers (D-VPLD, F-VPLD and ANN) each require two units to control the robot, one for each of the two wheels (Fig. 4-34). The inputs are six IR proximity sensors, two light intensity sensors, and two logic inputs (0 and 1). The sensor inputs are a floating-point values ranging from 0 to 1. The outputs are two signed RPM values used to control the speed of each wheel. The inputs to the three controllers are different for the three robotic navigation tasks: a) obstacle avoidance uses only the six proximity sensors and two logic values; b) light following uses only the two light sensors and two logic values; and c) the combined task uses the six proximity sensors, two light sensors and the two logic values.

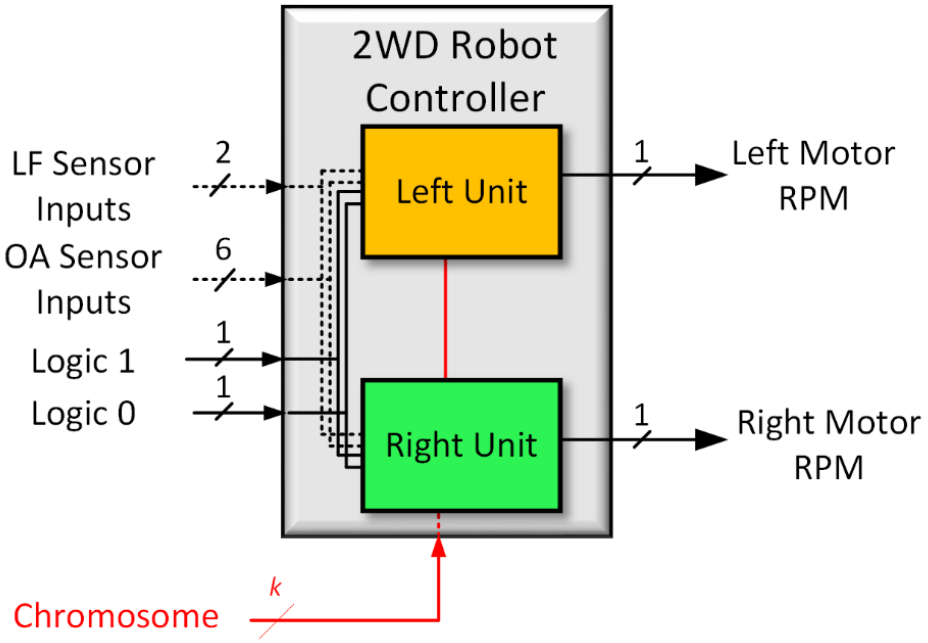

Fig. 4-34 Top Level Overview of 2WD Robot Control System

4.2.1 2WD Mobile Robot

The two-wheel differential drive mobile robot platform is used to assess the VPLD and ANN when used for a robotic navigation problem. The robotic controllers are evolved to optimally control the robot for three tasks; 1) obstacle avoidance, 2) light following, and 3) light following while avoiding obstacles. Two environments for each task are created; the first is the environment that the controllers are evolved in; the second is the test environment to investigate how the evolved controllers respond to a new environment that they were not directly evolved in. The primary aim in this experiment is to evolve high performance robotic controllers quickly. The navigation tasks demonstrates that more complex simulations which are slow to run on a FPGA SoC or impossible to run on the FPGA fabric, can be easily run on a PC along with the optimization algorithm used to evolve the controller configuration. The robotic platform and simulation environment used to evolve the controllers is detailed in the following sections.

Robotic Platform

The 2WD robot was developed at AUT for research in the field of evolutionary robotics. The hardware consists of a DE0 Nano development board (uses a Cyclone IV FPGA) for implementation of the control systems, six IR proximity sensors used for obstacle avoidance, and two light intensity sensors used for light following (Fig. 4-35). The robotic platform also includes other peripherals, a Bluetooth module for remote control and optical encoders used for the management of the motors speed.

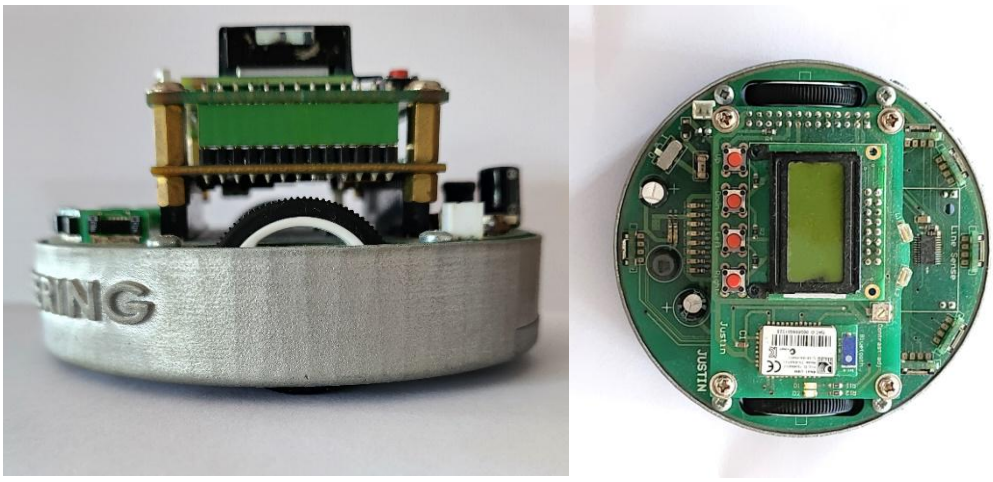

Fig. 4-35 Robotic platform developed at AUT for research in Evolutionary Robotics

Mathematical Model

The robot platform used is a two-wheel differential drive robot, the velocity V and angular velocity ω of the robot is determined by the combined angular velocity of each wheel (ω_L and ω_R). Through analysis of the robotic platform (Fig. 4-36) the forward kinematics of the robot can be determined.

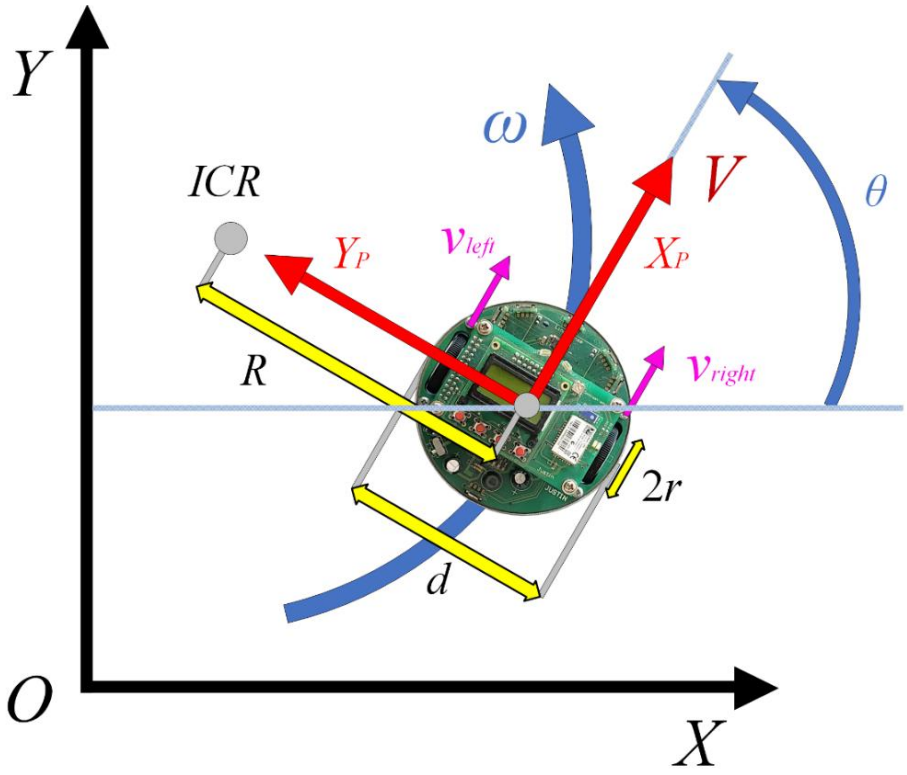

Fig. 4-36 Physical Robot, with kinematic model parameters

Table 4-15 2WD Robot Kinematic Parameters

Symbol	Parameter
V	Robot Velocity
ω	Robot Angular Velocity
θ	Robot Heading
v_{left}	Left Wheel Velocity
v_{right}	Right Wheel Velocity
ω_{left}	Left Wheel Angular Velocity
ω_{right}	Right Wheel Angular Velocity
ICR	Instantaneous center of rotation
R	Distance of the robot from the ICR
d	Diameter of the robot body
r	Radius of the robot's wheel

Using the equations of angular velocity, the velocity at each wheel can be determined with respect to the motion of the robot rotating around the ICR as well as with respect to the angular velocity each wheel is driven at by the control system.

$$V = \omega R \tag{4.35}$$

$$v_R = \omega\left(R - \frac{d}{2}\right) = r\omega_R \tag{4.36}$$

$$v_L = \omega\left(R + \frac{d}{2}\right) = r\omega_L \tag{4.37}$$

Solving equations (4.35), (4.36) and (4.37), for V, ω and R, in terms of the control parameters ω_L, ω_R and the robot properties r and d.

$$\omega = \frac{r(\omega_R - \omega_L)}{d} \tag{4.38}$$

$$R = \frac{d(\omega_R + \omega_L)}{2(\omega_R - \omega_L)} \tag{4.39}$$

$$V = \frac{r(\omega_R + \omega_L)}{2} \tag{4.40}$$

Using equations (4.38), (4.39) and (4.40), the kinematics of the robot relative to the robot coordinate system are found as follows.

$$\begin{bmatrix} V \\ 0 \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{x}_P \\ \dot{y}_P \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{4.41}$$

To obtain the kinematics of the robot in the world coordinate system, a rotational transform is applied based on the orientation of the robot relative to the world coordinate system.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R_z(\theta) \begin{bmatrix} V \\ 0 \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \tag{4.42}$$

Simulation

A simulation of the robot is required for calculating the fitness of the controllers during the evolution process. The simulation is based on the robot model as described and uses the derived forward kinematics of the 2WD robot. It models the robot behaviour, as well as the obstacle avoidance and light following sensors. The robot has a 50ms control interval and is driven by the wheel velocities determined by the evolved controllers. Three environments are used in the evolution of robotic controllers: 1) obstacle avoidance; 2) light following; and 3) combined obstacle avoidance and light following. Three additional environments (one for each task) are used to test the ability of the controllers to generalise the evolved behaviour in new environments.

Sensor Modelling

The six IR proximity sensors used for obstacle avoidance have a programmable cone detection radius, however in order to minimize the computation time, they are modelled as a single beam perpendicular to the sensor position (Fig. 4-37). The sensor distance measurement (Fig. 4-38) ranges from 0 (minimum distance 5cm) to 1 (maximum distance 15cm). The distance approximated by the sensor ρ is used to calculate the input P_n for the robotic controllers (4.43). Random noise μ at a level of 5% is applied to the IR sensor output.

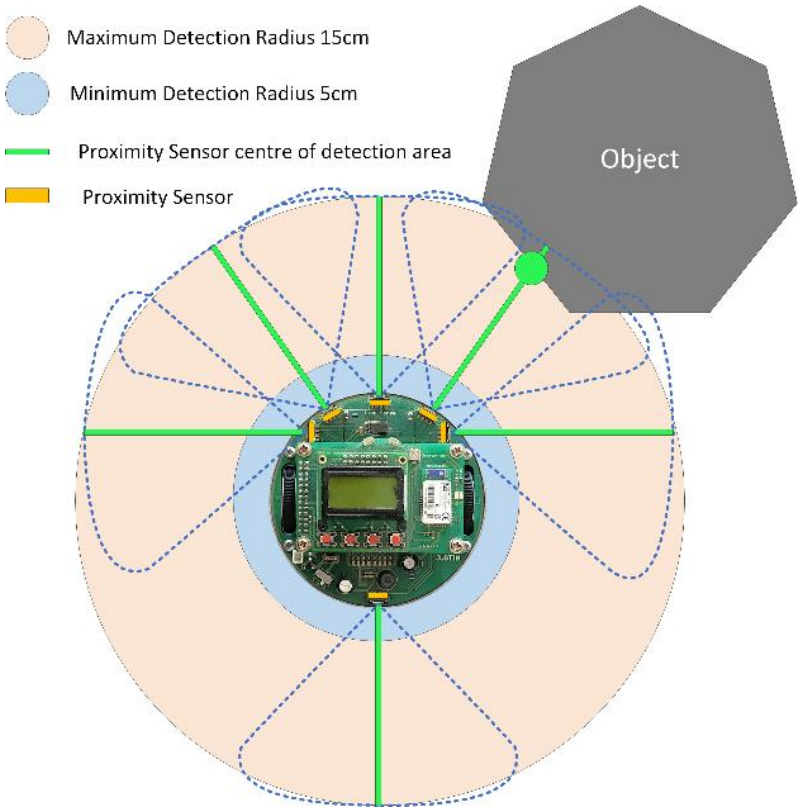

Fig. 4-37 IR Proximity Sensor Positions

$$P_n = \frac{(\rho_n - 5)}{(15 - 5)} + \mu \tag{4.43}$$

An example of the IR proximity sensor model is shown in Fig. 4-38, the object is found to be 12.8cm away from the sensor, this results in a sensor value of 0.78. For example, if there is no object within the maximum distance the sensor output would be 1, alternatively if an object is closer than the minimum distance the sensor output would be 0.

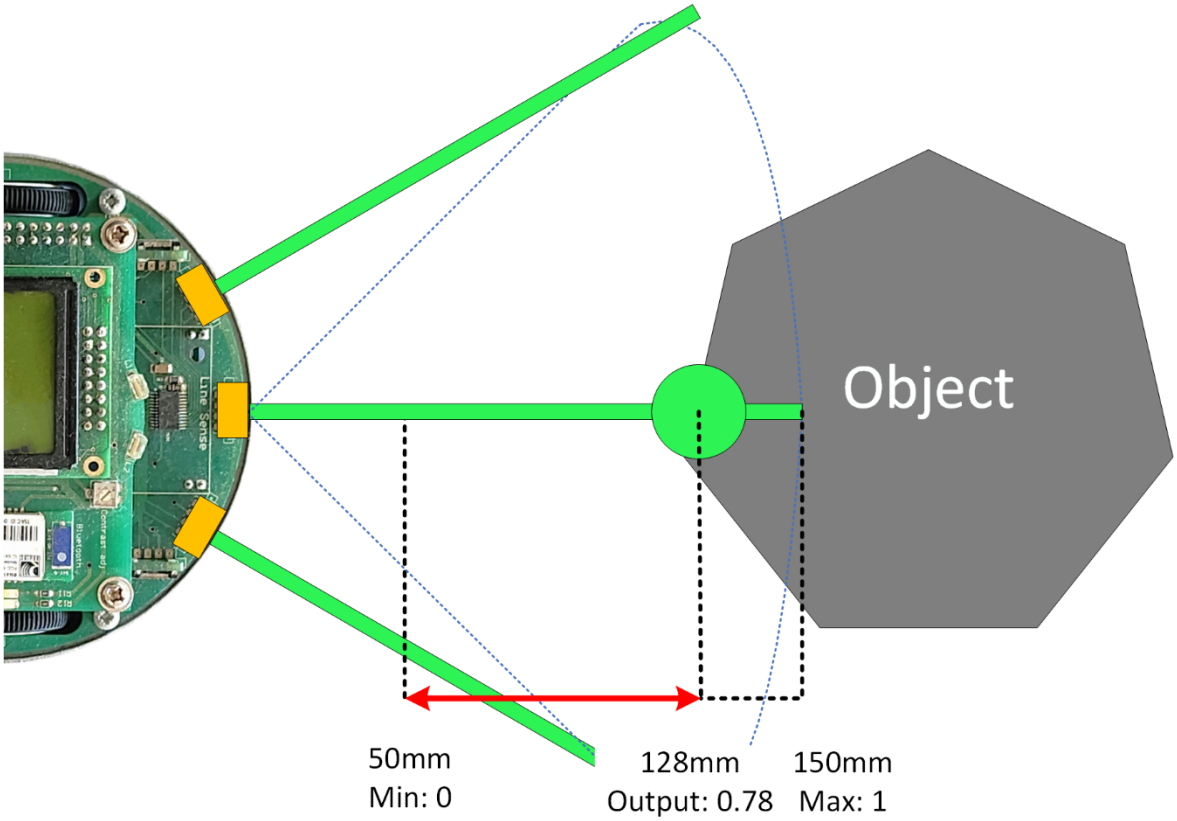

Fig. 4-38 IR Proximity Sensor Measurement Example

Example:

$$\rho_n = 12.8\ cm, \mu = 0.0$$

$$P_n = \frac{(\rho_n - 5)}{(15 - 5)} + \mu$$

$$P_n = \frac{(12.8 - 5)}{(15 - 5)} + 0.0$$

$$P_n = \frac{(12.8 - 5)}{(15 - 5)} = 0.78$$

The two light sensors on the robot used to measure light intensity are physically offset by +/- 10 degrees from the forward heading of the robot resulting in 20 degrees of separation in the field of view of the two sensors. The light intensity of each sensor is determined by the position of the light relative to the robot heading. The difference in the forward direction of an LDR sensor and the position of the light relative to the robot must be less than +/-90 degrees to register any level of intensity. The sensor is at its maximum output of 1 when the light source is 90 degrees (perpendicular) to the forward-facing direction of the sensor, the sensor value decreases linearly to a value of 0 as the position of the light moves away +/- 90 degrees relative to the forward-facing direction of the given sensor.

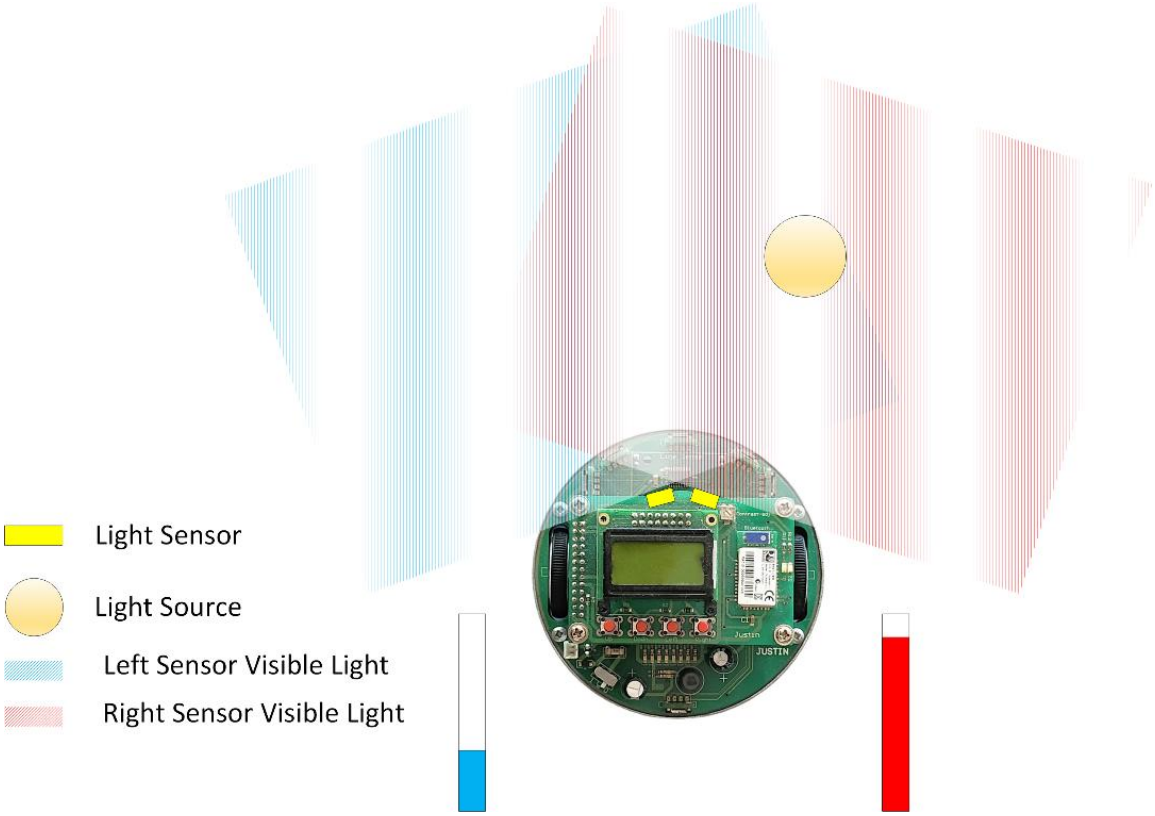

Fig. 4-39 Light Sensor Positions

When calculating the light sensor intensities, h the heading of the light from the robot relative to the robots orientation is calculated (4.45), (γ the heading of the light from the robot, θ orientation of the robot). Then using the piece wise functions (4.46) & (4.47) the individual light sensor intensities can be calculated. Random noise μ at a level of 5% is applied to the IR sensor output.

$$\gamma = tan^{-1}\left(\frac{x_{light} - x_{robot}}{y_{light} - y_{robot}}\right) \tag{4.44}$$

$$h = \gamma - \theta \tag{4.45}$$

$$I_L = \begin{cases} 1 - abs\left(\frac{h-10}{90}\right) + \mu\,, & -90 \leq (h-10) \leq 90 \\ 0\,, & (h-10) > 90 \\ 0\,, & (h-10) < -90 \end{cases} \tag{4.46}$$

$$I_R = \begin{cases} 1 - abs\left(\frac{h+10}{90}\right) + \mu\,, & -90 \leq (h+10) \leq 90 \\ 0\,, & (h+10) > 90 \\ 0\,, & (h+10) < -90 \end{cases} \tag{4.47}$$

A set of three examples are shown in Fig. 4-40. For the first example, light source A is directly in front of the robot, this would result in an equal output for both sensors, being 10 degrees away from the forward-facing direction of the sensors the result of the calculation is a sensor output of 0.88 for both the left and right sensor. The second example using light source B would give the max output for the right light intensity sensor, this is because light source B is perpendicular to that sensor. The left sensor light intensity will be less and can be calculated as 0.78. The final example using light source C would result in both the left and right light sensor registering zero intensity. The heading of Light source C from the robot is greater than +/- 90 degrees relative to the forward-facing direction of each sensor, meaning the light source is outside the field of view of both sensors.

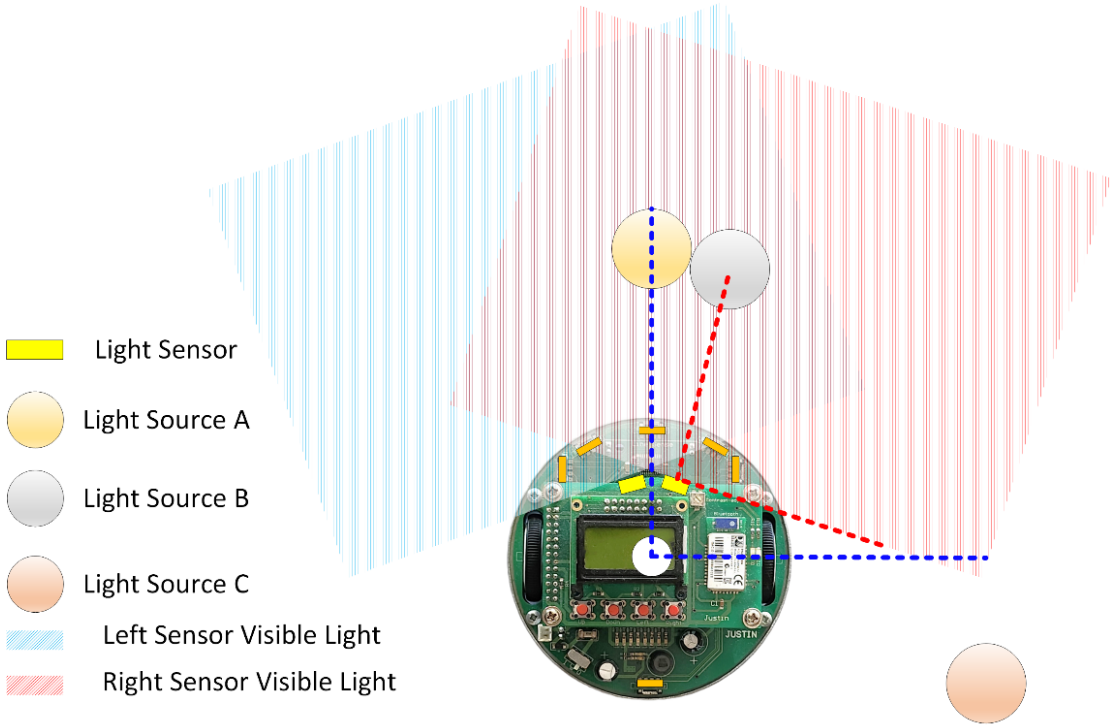

Fig. 4-40 Light Sensor Measurement Example

Example A:

$$h = 0,$$

$$\mu = 0,$$

$$I_L = 1 - abs\left(\frac{h-10}{90}\right) + \mu = 1 - abs\left(\frac{-10}{90}\right) = 0.88$$

$$I_R = 1 - abs\left(\frac{h+10}{90}\right) + \mu = 1 - abs\left(\frac{10}{90}\right) = 0.88$$

Example B:

$$h = -10,$$

$$\mu = 0,$$

$$I_L = 1 - abs\left(\frac{h-10}{90}\right) + \mu = 1 - abs\left(\frac{-20}{90}\right) = 0.78$$

$$I_R = 1 - abs\left(\frac{h+10}{90}\right) + \mu = 1 - abs\left(\frac{0}{90}\right) = 1.00$$

Obstacle Avoidance

For the OA task the robot is evolved from six starting orientations. All the start positions are at the centre of the environment but use different orientations (0, 180, 45, -45, 90, and -90 degrees) (Fig. 4-41). The OA simulation is run for 40s for each starting orientation The test environment for the OA task uses obstacles placed in different locations than in the evolved environment. For the test environment the robot still starts in the centre of the environment (Fig. 4-42).

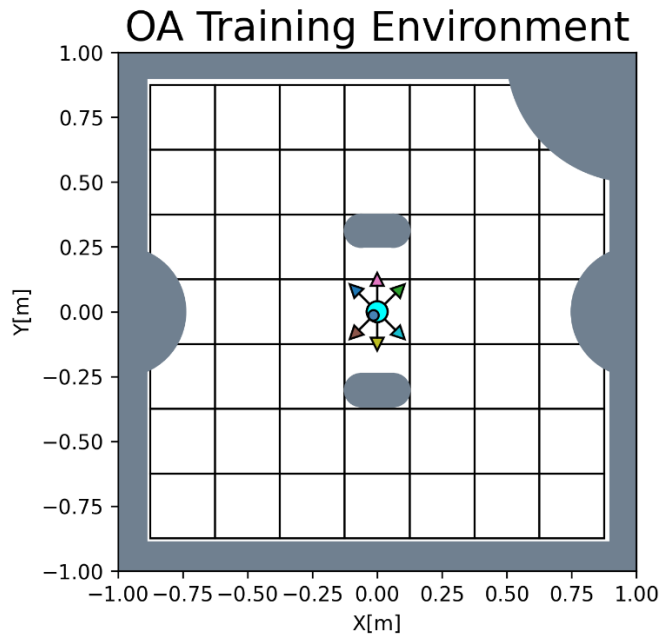

Fig. 4-41 2WD robot obstacle avoidance environment used to evolve the robotic controllers

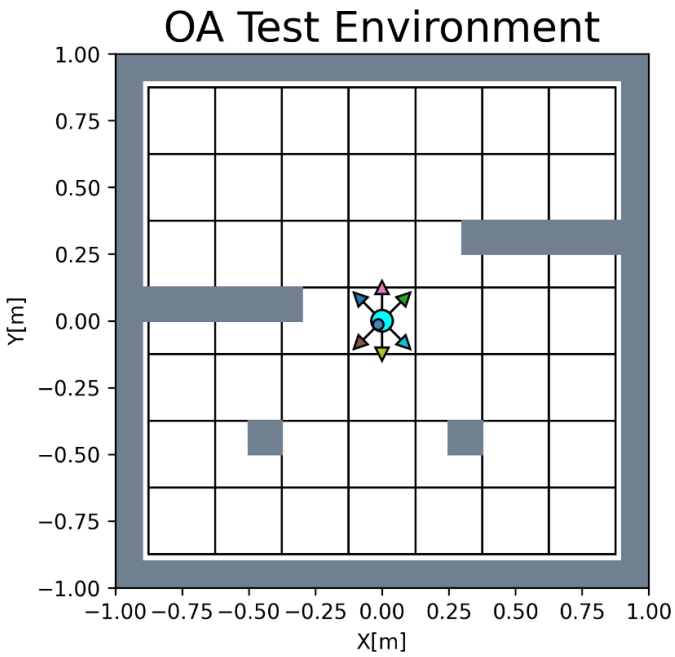

Fig. 4-42 2WD robot obstacle avoidance environment used to test the evolved robotic controllers

Light Following

For the LF task, the robot is evolved using eight starting positions spread around the environment. Each starting position has the same orientation so that the robot is evolved with the light source starting at a different angle relative to the heading of the robot. The LF simulation is run for 10s for each starting position (Fig. 4-43).

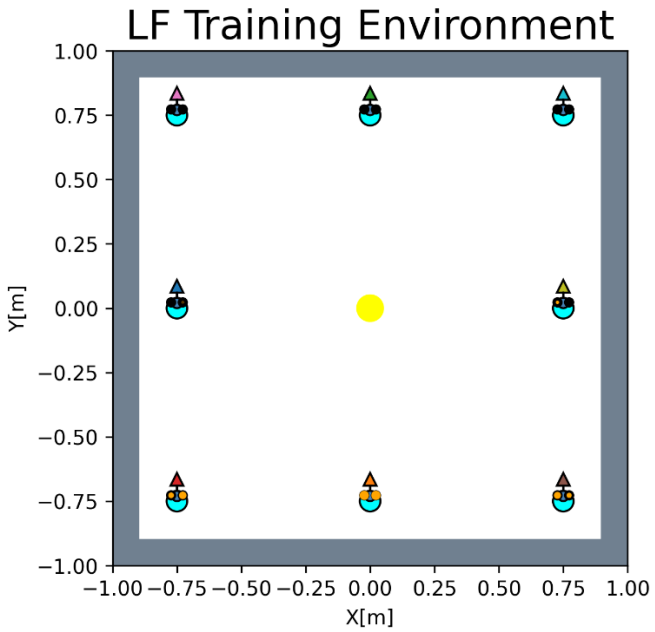

Fig. 4-43 2WD robot light following environment used to evolve the robotic controllers

For the test environment, the light is moved to a different position and the start position of the robots is altered (Fig. 4-44).

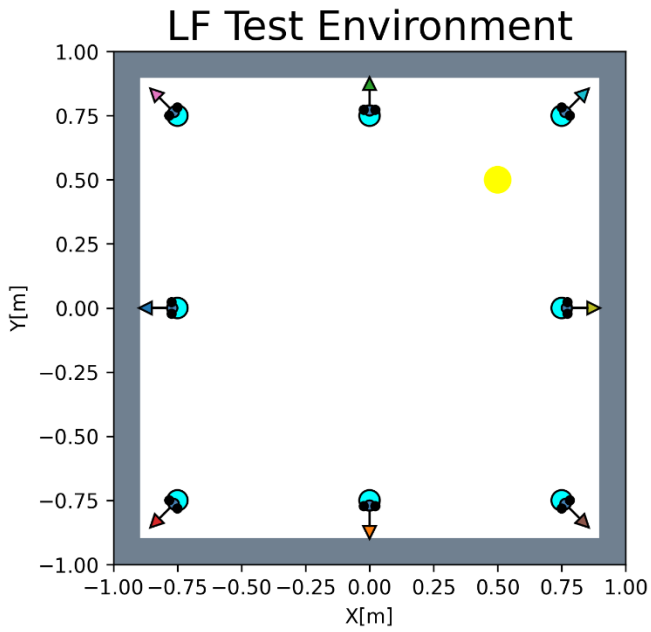

Fig. 4-44 2WD robot light following environment used to test the evolved robotic controllers

Light following while avoiding obstacles

The OA-LF task uses the same starting positions as the LF task, but the evolved environment has obstacles placed in between the robot start positions and the light source. Two of the start position do not have obstacles directly in the way of the light source, this is done to promote controllers that can move directly towards the light source in a straight line when no obstacle is present (Fig. 4-45).

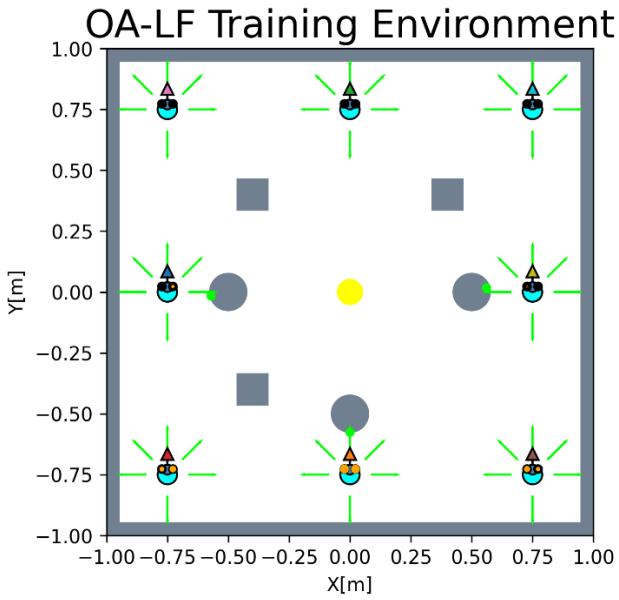

Fig. 4-45 2WD robot light following while avoiding obstacles environment used to evolve the robotic controllers

For the test environment, the starting positions are changed, the light source is moved, and different obstacles are placed in the environment (Fig. 4-46).

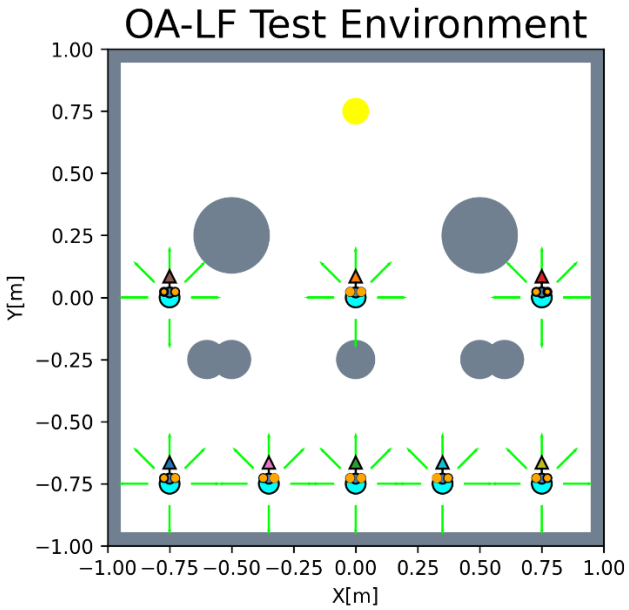

Fig. 4-46 2WD robot light following while avoiding obstacles environment used to test the evolved robotic controllers

4.2.2 D-VPLD Control System Architecture

The D-VPLD variable type is pseudo bits, limiting an integer to two states, 0 or 1 to imitate a digital circuit, as seen in the hexapod robot problem. The floating-point sensor inputs are quantized into 4 bits using a threshold encoding technique similar to "one-hot" encoding. This means the D-VPLD has, 26 bits for the OA task (2 logic and 6x4-bits proximity sensors), 10 bits for the LF task (2 logic and 2x4-bits light sensors), and 34 bits for the combined OA-LF task (2 logic, 6x4bits proximity and 2x4-bits light sensors). The architecture (Fig. 4-47) has two layers with sixteen FABs in the first layer and four FABs in the output layer. The FABs in both layers have four multiplexers and one FE. The inputs to the multiplexers are from the previous layer, the selected inputs are then fed into the FE to produce the FABs output. The FE uses a LUT (Table 4-17) comprised of 21 combinational logic functions.

Table 4-16 D-VPLD Input encoding technique

Value	4bit Encoding
$x = 0$	0000
$0.25 \cdot Sensor_{max} \geq x > 0$	0001
$0.50 \cdot Sensor_{max} \geq x > 0.25 \cdot Sensor_{max}$	0011
$0.75 \cdot Sensor_{max} \geq x > 0.50 \cdot Sensor_{max}$	0111
$1.00 \cdot Sensor_{max} \geq x > 0.75 \cdot Sensor_{max}$	1111

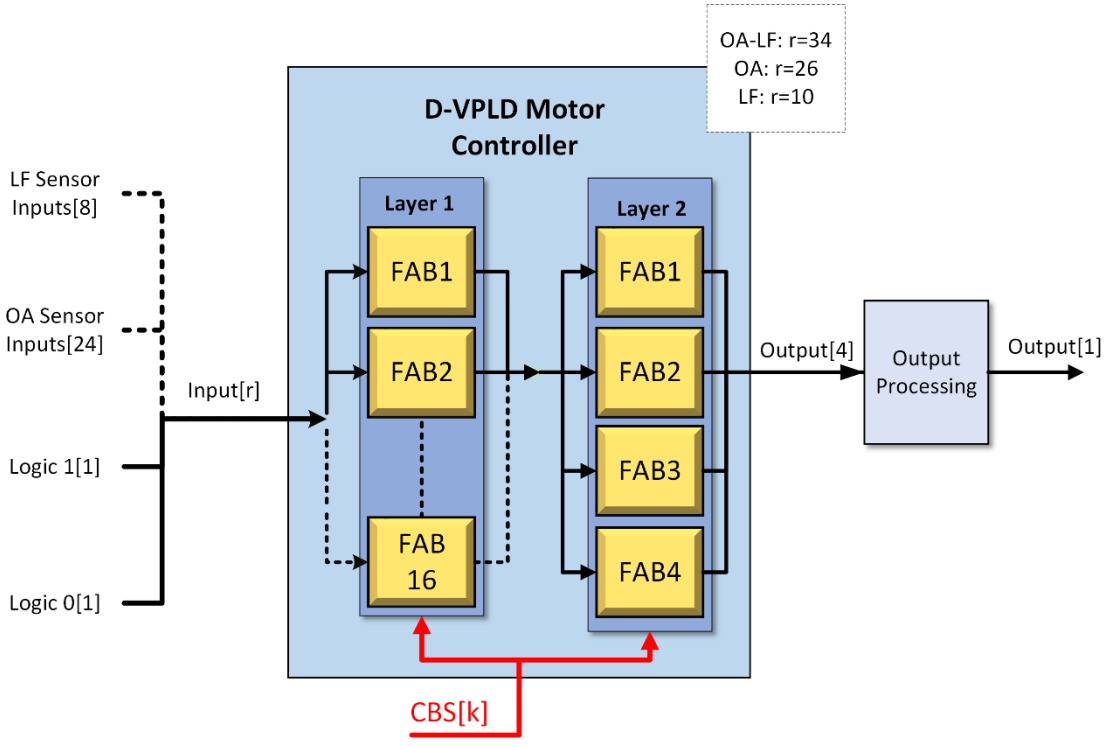

Fig. 4-47 D-VPLD Unit Architecture for 2WD robot problem

Table 4-17 D-VPLD LUT for 2WD robot problem

No.	Logic Function	No.	Logic Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

The output of each D-VPLD unit is a 4-bit signed value which is the combination of the four FABs outputs in the final layer. This value is used to calculate the desired RPM of the wheel the D-VPLD is controlling. The MSB determines the sign, the least significant bits determine the magnitude (4.48).

$$RPM = RPM_{max} * \frac{sgn(output[3]) * output[2:0]}{2^3 - 1} \tag{4.48}$$

Chromosome

The chromosome of the D-VPLD is its configuration bitstream. As seen previously, a two-dimensional array of integers is used to store the chromosome of one layer. For the 2WD robotic control problem a pair of these two-dimensional arrays completes the whole CBS. The search space for the D-VPLD CBS varies for each task (OA, LF, OA-LF), this is because the number of inputs is different for each task. The search space is calculated (using equation 4.11) as follows,

$$S = (L1_{inputs}{}^{L1_{MUXs}})(\,L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(\,L2_{functions}{}^{L2_{FEs}})$$

$$S_{OA} = (26^{4*16})(\,21^{16})(16^{4*4})(\,21^{4}) = 1.85 \cdot 10^{136}$$

$$S_{LF} = (10^{4*16})(\,21^{16})(16^{4*4})(\,21^{4}) = 5.13 \cdot 10^{109}$$

$$S_{OA-LF} = (34^{4*16})(\,21^{16})(16^{4*4})(\,21^{4}) = 5.31 \cdot 10^{143}.$$

Crossover

For the 2WD problem, a different crossover method is used instead of the method used in the hexapod problem. In this problem, the D-VPLD uses a two-part crossover method on each FAB in the CBS; this can be considered full multipoint crossover when applied to each FAB in the two-dimensional integer arrays that make up the complete chromosome of a D-VPLD unit (Fig. 4-48). The first part of the crossover of a FAB configuration (row in the two-dimensional integer arrays) is a point that slices the MUX configurations between genes from either parent alpha or gamma. The second part of the crossover operation is whether the child FAB configuration gets parent alphas FE config or parent gammas FE config. The child chromosomes are mutated based on the mutation rate. The crossover and mutation algorithms are constrained to each motor unit, i.e. the configuration of the left unit of parent alpha can only be crossed over with the left unit configuration of parent gamma.

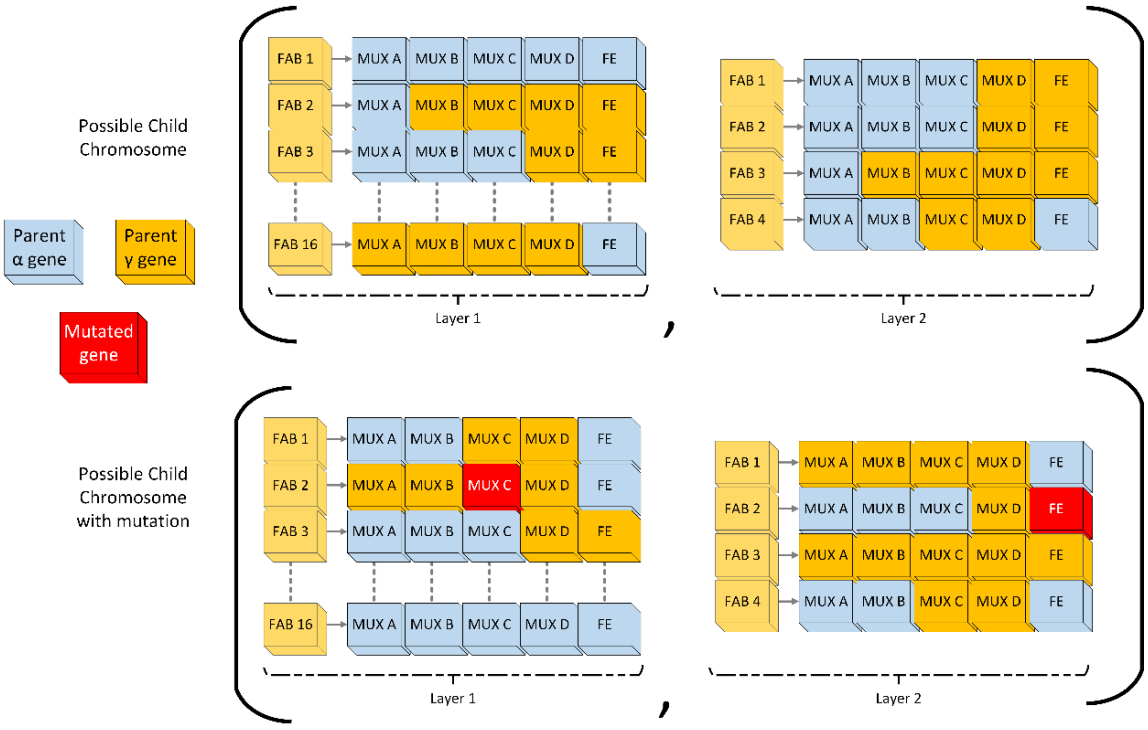

Fig. 4-48 D-VPLD Chromosome Crossover for 2WD problem

4.2.3 F-VPLD Control System Architecture

The F-VPLD architecture (Fig. 4-49) uses a floating-point variable, thus no quantization of the normalized sensor inputs is required. The F-VPLD has, eight inputs for the OA task (two logic and six proximity), four inputs for the LF task (two logic and two light), and ten inputs for the combined OA-LF task (two logic, six proximity and two light). The F-VPLD has a two-layer architecture with sixteen FABs in the first layer and one FAB in the output layer. As with the D-VPLD the F-VPLD FABs have four multiplexers and one FE. The FE contains a LUT (Table 4-18) consisting of 21 arithmetic functions using addition, subtraction, and multiplication operators. The function f is a piece wise linear function which limits the output of the arithmetic operation from -2 to +2 (4.49).

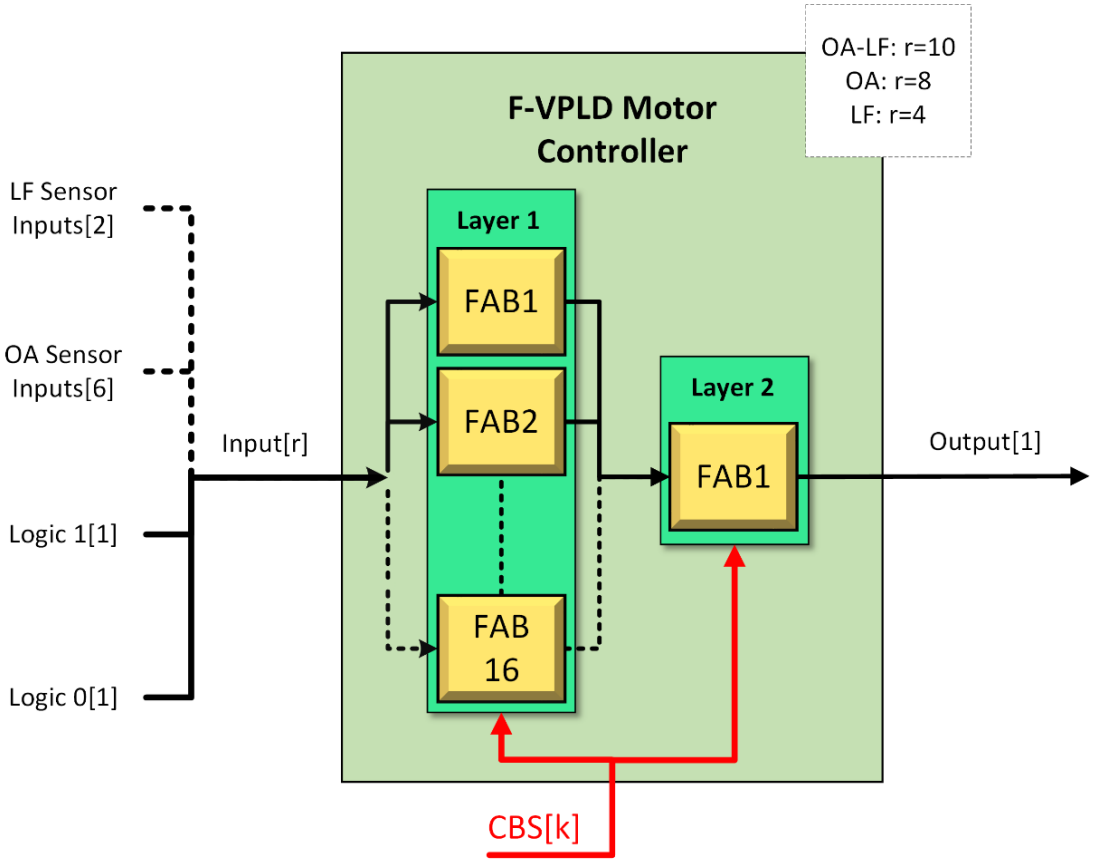

Fig. 4-49 F-VPLD Unit Architecture for 2WD robot problem

$$f(x) = \begin{cases} 2, & x \geq 2 \\ x, & 2 > x > -2 \\ -2, & x \leq -2 \end{cases} \tag{4.49}$$

Table 4-18 F-VPLD LUT for 2WD robot problem

No.	Mathematical Function	No.	Mathematical Function
0	f((A+B)*(C-D))	11	f((-A)+B+C+D)
1	f(A+B+C+D)	12	f(A+(-B)+C+D)
2	f(-(A+B+C+D))	13	f(A+B+(-C)+D)
3	f(A*B*C*D)	14	f(A+B+C+(-D))
4	f(-(A*B*C*D))	15	f((A*B)+(C*D))
5	f((A+B)*(C+D))	16	f((A*C)+(B*D))
6	f((A+C)*(B+D))	17	f((A*D)+(B*C))
7	f((A+D)*(B+C))	18	f((A*B)-(C*D))
8	f((A+B)-(C+D))	19	f((A*C)-(B*D))
9	f((A+C)-(B+D))	20	f((A*D)-(B*C))
10	f((A+D)-(B+C))		

The F-VPLD FABs use the same basic architecture as the D-VPLD but has different functions, data types and number of inputs. The FABs in layer one requires either a 8x1, a 4x1 or a 10x1 multiplexers (depending on whether the task is obstacle avoidance, light following or a combination of both). Each of the four multiplexers selects one input from the sensors and feeds the input into the FE. The CBS is the F-VPLD chromosome and is used to configure the multiplexer and FE.

The D-VPLD required 4 output FABs but the F-VPLD only requires 1 as it produces a signed floating-point output suitable for use in calculating the desired RPM equation (4.50).

$$RPM = RPM_{max} * \frac{output}{2} \tag{4.50}$$

Chromosome and Crossover

The F-VPLD chromosome format and crossover operation is the same as the D-VPLD. The chromosome is stored in a pair of two-dimensional integer arrays, one for each layer. The F-VPLD has a slightly smaller chromosome compared to the D-VPLD, as it has less FABs. As with the D-VPLD, the search space for the F-VPLD CBS varies for each task (OA, LF, OA-LF) because the number of inputs is different for each task. The search space is calculated (using equation 4.11) as follows,

$$S = (L1_{inputs}{}^{L1_{MUXs}})(\,L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(\,L2_{functions}{}^{L2_{FEs}})$$

$$S_{OA} = (8^{4*16})(\,21^{16})(16^{4})(\,21^{1})$$

$$S_{OA} = 1.24 \cdot 10^{85}$$

$$S_{LF} = (4^{4*16})(\,21^{16})(16^{4})(\,21^{1})$$

$$S_{LF} = 6.69 \cdot 10^{65}$$

$$S_{OA-LF} = (10^{4*16})(\,21^{16})(16^{4})(\,21^{1})$$

$$S_{OA-LF} = 1.97 \cdot 10^{91}.$$

4.2.4 ANN Control System Architecture

As with the D-VPLD and F-VPLD the ANN control architecture has two units one for each motor, each ANN unit is a two-layer fully connected feedforward architecture with sixteen neurons in the hidden layer and one in the output layer (Fig. 4-50). The ANN uses the same floating-point inputs as the F-VPLD, eight inputs for the OA task (two logic and six proximity), four inputs for the LF task (two logic and two light), and ten inputs for the combined OA-LF task (two logic, six proximity and two light). The neurons in both layers use the hyperbolic tangent activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01.

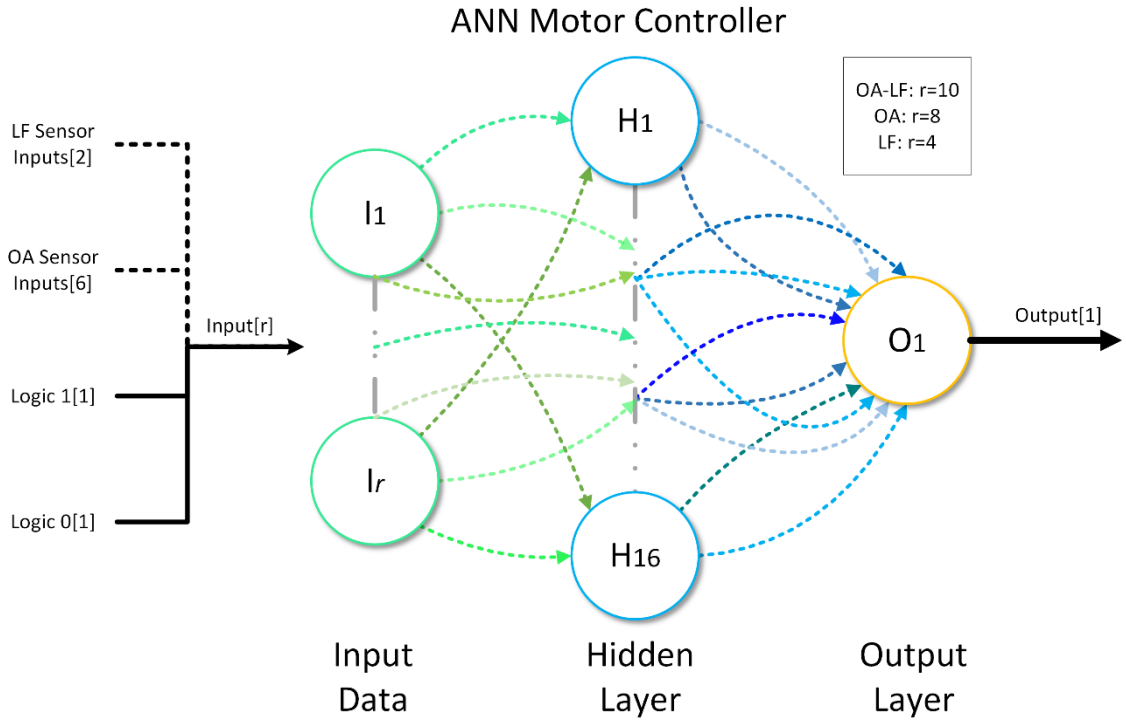

Fig. 4-50 ANN Controller Architecture for 2WD problem

The ANN only requires one output neuron as it produces a floating-point output ranging from -1 to +1 which is suitable for use in calculating the desired RPM equation (4.51).

$$RPM = RPM_{max} * output \tag{4.51}$$

Chromosome

The ANNs chromosome is its weights and biases. The format for the chromosome used in the hexapod control problem is also used in this experiment. This set of two-dimensional floating-point arrays which each store the configuration of a layer. The hidden layer component of the chromosome is an array of size 16xr (r=8,4, or 10), and for the second layer an array of size 1x17; this gives a search space of $3.53 \cdot 10^{34}$. The search space for the ANN chromosome can be calculated (using equation 4.34) as follows.

$$S = (no.of\ values^{H_{weights+bias}+O_{weights+bias}})$$

$$no.of\ values = 1 + \frac{max - min}{increment\ size} = 1 + \frac{1-1}{0.01} = 201$$

$$S_{OA} = 201^{(16*8)+(1*17)} = 9.19 \cdot 10^{333}$$

$$S_{LF} = 201^{(16*4)+(1*17)} = 3.64 \cdot 10^{186}$$

$$S_{OA-LF} = 201^{(16*10)+(1*17)} = 4.63 \cdot 10^{407}$$

Crossover

The crossover of the ANN chromosomes during the reproduction phase of the GA is a two-part crossover of each neurons weights and biases, this can be considered a full multipoint crossover when applied to each neuron in the two-dimensional floating-point arrays that make up the complete chromosome(Fig. 4-51). The first part of the crossover of a neuron's configuration (row in the two-dimensional arrays) is a point that slices the weight values between genes from either parent alpha or gamma. The second part of the crossover is whether the child neuron gets parent alphas bias value or parent gammas bias value. The child chromosomes are mutated based on the mutation rate. The crossover and mutation algorithms are constrained to each motor unit, i.e. the configuration of the right unit of parent alpha can only be crossed over with the right unit configuration of parent gamma.

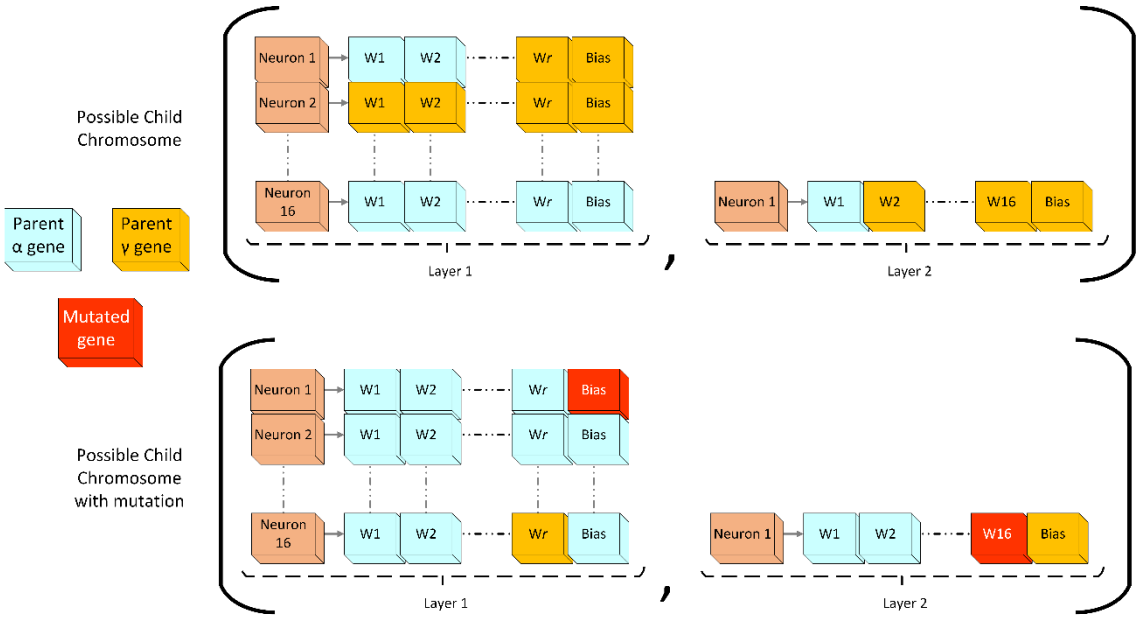

Fig. 4-51 ANN Chromosome Crossover for 2WD problem

4.2.5 Genetic algorithm

A GA is applied using the same parameters (Table 4-19) for all the controllers being evolved, the algorithm follows the process of fitness evaluation, selection and reproduction in an iterative process until an optimal system configuration is evolved. Optimal control system configurations allow the robot to successfully complete one of the three tasks obstacle avoidance, light following, and light following while avoiding obstacles. The genetic algorithm used for evolving the robotic controllers in this problem is the same as the algorithm used in the hexapod problem, but with different GA parameters. The population size of 100 is chosen again to insure diversity during the evolution process. A reduced population size may improve the computation time of the program per generation but can reduce the evolutionary rate.

Table 4-19 GA Parameters used for 2WD controller evolution

Parameter	Value
Population Size	100
Mutation Rate OA	2%
Mutation Rate LF	1%
Mutation Rate OA-LF	2%
Selection	Two stage binary tournament
Crossover	Multipoint

4.2.6 Fitness Evaluation

The fitness functions used to evolve the robotic controllers are designed to result in optimal configurations. An optimal configuration will allow the robot to successfully complete the desired task: 1) obstacle avoidance; 2) light following; and 3) combined obstacle avoidance and light following. The robotic controllers are evolved in the simulation environment (Fig. 4-52) described in section 0.

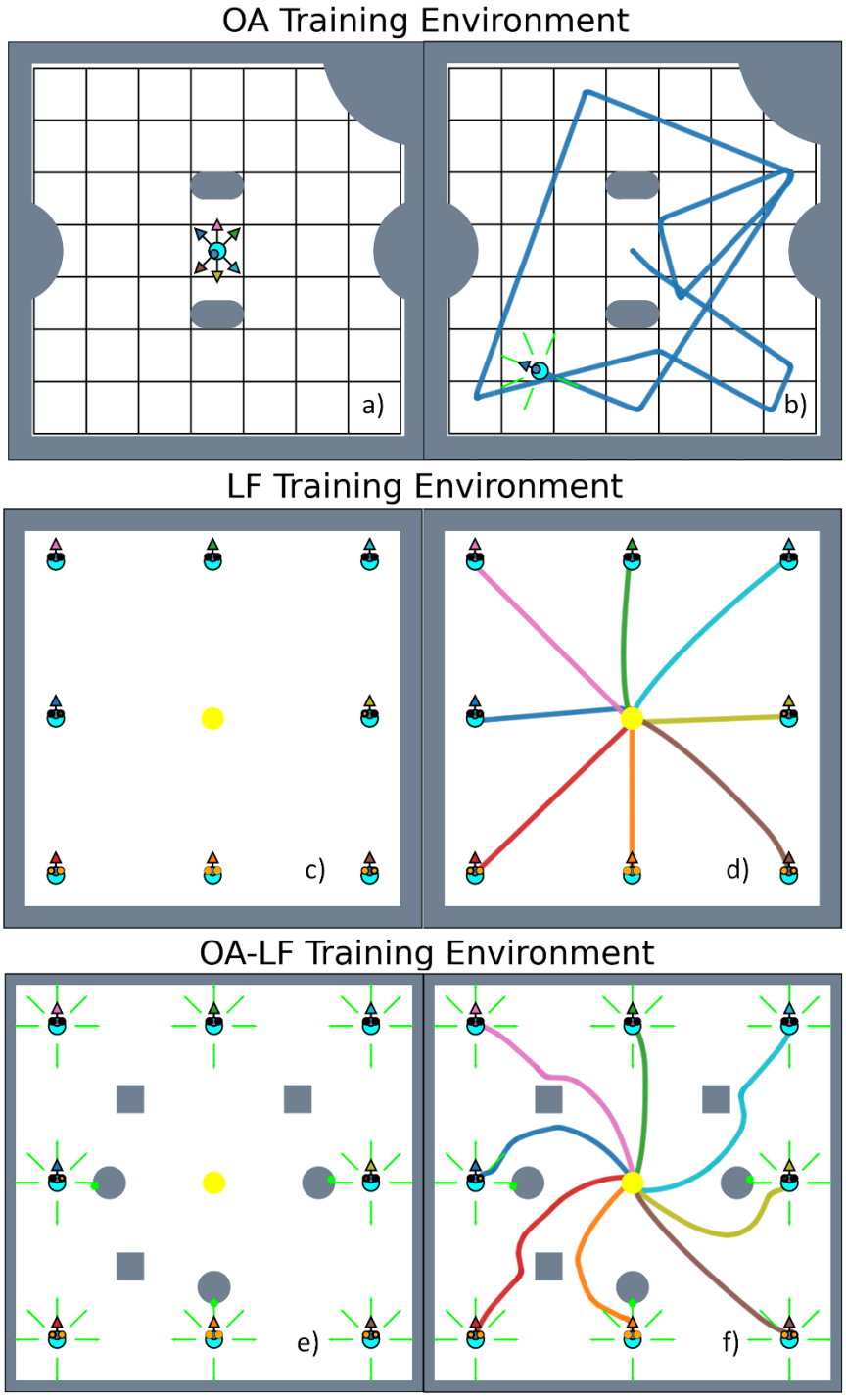

Fig. 4-52 Simulation Environments used to evolve the robotic controllers. Left) shows starting positions, right) shows example trajectories

Obstacle Avoidance

The fitness function for the OA problem is designed to promote controllers that can travel the complete test period without crashing while exploring a large area within the environment. The fitness F_{OA} (4.52) of the OA controllers is based on five factors: 1) the percentage of area explored (value 0-1), the environment is broken down into 44 25x25cm explorable sectors; 2) avoidance, (value 0-1) which is the percentage of the test period time the robot travelled without crashing; 3) straight motion, the percentage of time the robot is traveling straight while it isn't avoiding obstacles; 4) forward motion, the percentage of time the robot is traveling forward while not avoiding obstacles; and 5) speed, the average speed of the robot as a percentage of the maximum speed possible. The weightings of the fitness components are designed to encourage the exploration of the environment as quickly as possible whilst avoiding obstacles.

$$F_{OA} = 30 * F_{Area} + 30 * F_{Avoidance} + 10 * F_{Straight} + 10 * F_{Forward} + 20 * F_{Speed} \quad (4.52)$$

Where:

$$F_{Area} = \frac{sectors\ explored}{total\ sectors} \quad (4.53)$$

$$F_{Avoidance} = \frac{time\ travled\ without\ crashng}{max\ test\ time} \quad (4.54)$$

$$F_{Straight} = \frac{time\ travling\ straight}{time\ not\ avoiding\ obstacles} \quad (4.55)$$

$$F_{Forward} = \frac{time\ travling\ forward}{time\ not\ avoiding\ obstacles} \quad (4.56)$$

$$F_{Speed} = \frac{average\ speed}{\max speed} \quad (4.57)$$

Light Following

An optimal LF robotic controller will allow the robot to orientate itself in the direction of a light source and move towards the light source as quickly as possible. The fitness F_{LF} (4.58) is based on four factors: 1) the distance from the robot and light source at the end of the simulation run (value 0-1); 2) the path of the robot(value 0-1), used to minimize the distance travelled so the robot doesn't move away from the light or take a long arcing path ; 3) the percentage of time travelled without moving out of the environment; and 4) forward motion, the percentage of time traveling forward.

$$F_{LF} = 100 * F_{Reached\ Light} * F_{Path} * F_{Avoidance} * F_{Forward} \tag{4.58}$$

Where:

$$F_{Reached\ Light} = 1 - \frac{final\ distance}{intial\ distance} \tag{4.59}$$

$$F_{Path} = \sum_{i=0}^{n} \frac{initial\ distance - current\ distance_i}{n * intial\ distance} \tag{4.60}$$

$$F_{Avoidance} = \frac{time\ travled\ without\ crashing}{max\ test\ time} \tag{4.61}$$

$$F_{Forward} = \frac{time\ travling\ forward}{total\ time} \tag{4.62}$$

Obstacle Avoidance while Light Following

For the combined obstacle avoidance and light following task, the robot should orientate itself in the direction of a light source and move towards the light source as quickly as possible which is the same as for the light following task, except it must do so whilst avoiding any obstacles in its path. Therefore, the fitness (F_{OA-LF}) is the same as for the light following task.

$$F_{OA-LF} = F_{LF} \tag{4.63}$$

4.2.7 Results: Obstacle Avoidance

For the obstacle avoidance task, all three controllers, D-VPLD, F-VPLD and ANN are evolved using the same GA. The GA is run for 250 generations, with the fitness at each generation and the time taken to complete each generation recorded. One hundred solutions for each controller are evolved. The simulation time used to evaluate the fitness of an individual is 40 seconds, all controllers use the same input data from the simulation and are evolved to output the required wheel velocities to drive the robot. An optimal controller for the obstacle avoidance task allows the robot to explore the environment without crashing into any obstacles within the environment. The simulation time of 40 seconds is chosen to provide sufficient time for the robot to explore its environment.

The parameters used to evaluate the controllers are: a) evolutionary efficiency, (how many generations and the computation time required to reach a solution); and b) the controller performance (the maximum fitness achieved by the controller). As a second test of the controller performance the fully evolved controller is then tested in an environment that differs from the training environment.

Evolutionary efficiency

All three controllers had a comparable evolutionary efficiency in the obstacle avoidance task, all achieving fitness scores above 90%. On average across the 100 results, the F-VPLD and ANN has a faster initial rate of fitness improvement per generation compared to the D-VPLD, but all reach a similar level of fitness (Fig. 4-53 & Table 4-20). When looking at the number of generations to converge to an optimal fitness level, all controllers can do so in under 120 generations, with the ANN the fastest to evolve on average. The median generations to evolve of all controllers is comparable. When observing the evolution, it is noted that in the initial generations the D-VPLD can get stuck in a local maxima where it adequately avoids obstacles but is doing so at a slow pace (approximately half max speed) meaning it loses fitness points on the speed component and the area explored component as it cannot travel as far. Eventually the D-VPLD can break out of the local maxima to reach a high fitness level.

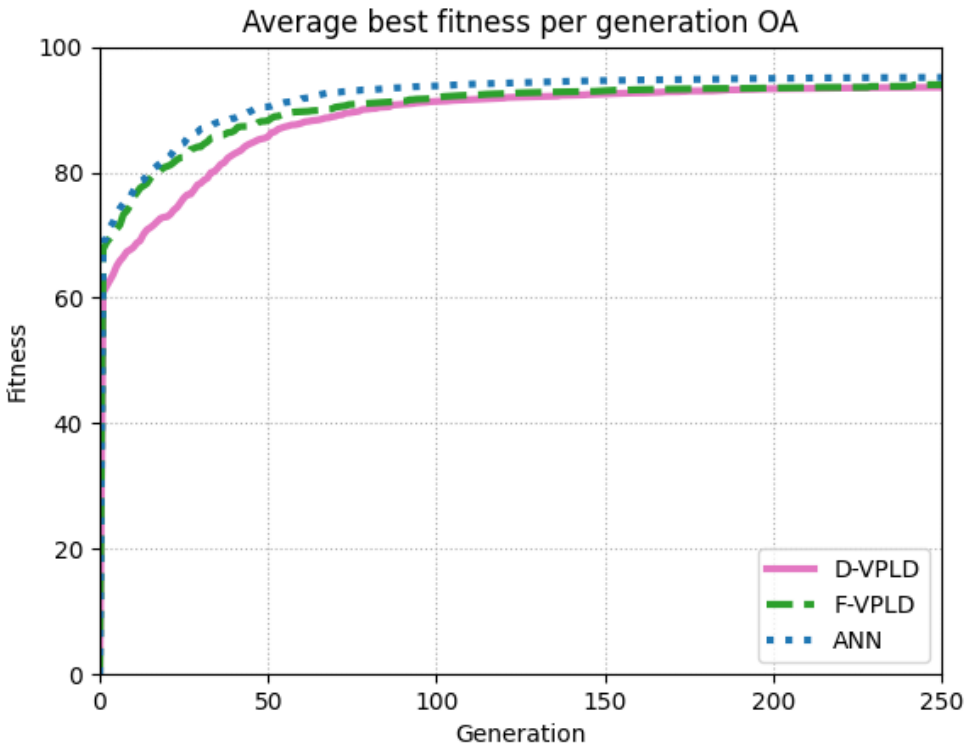

Fig. 4-53 A comparison of the OA fitness per generation of the three controllers

Table 4-20 The Number of Generations to Evolve OA Task

Controller	Mean	Median	CV
D-VPLD	115.58	104	0.41
F-VPLD	109.6	102.5	0.48
ANN	101.77	101.5	0.43

With respect to computation time to evolve, the ANN required the least time to evolve. Based on the median value the D-VPLD controller required approximately 38s more computation time than the ANN, the F-VPLD required approximately 20s more computation time to evolve (Table 4-21). The ANN is faster to evolve even with a comparable median number of generations because of its time to execute at each step in the simulation. The D-VPLD also requires extra computation to convert the floating-point input data from the simulation to a binary format, and then convert the outputs back from binary to floating point values to drive the simulation. The distribution of the time to evolve solutions is shown in Fig. 4-54.

Table 4-21 Computation Time Required to Evolve in Seconds OA Task

Controller	Mean[s]	Median[s]	CV
D-VPLD	173.84	150.72	0.44
F-VPLD	146.02	132.67	0.54
ANN	119.47	113.04	0.48

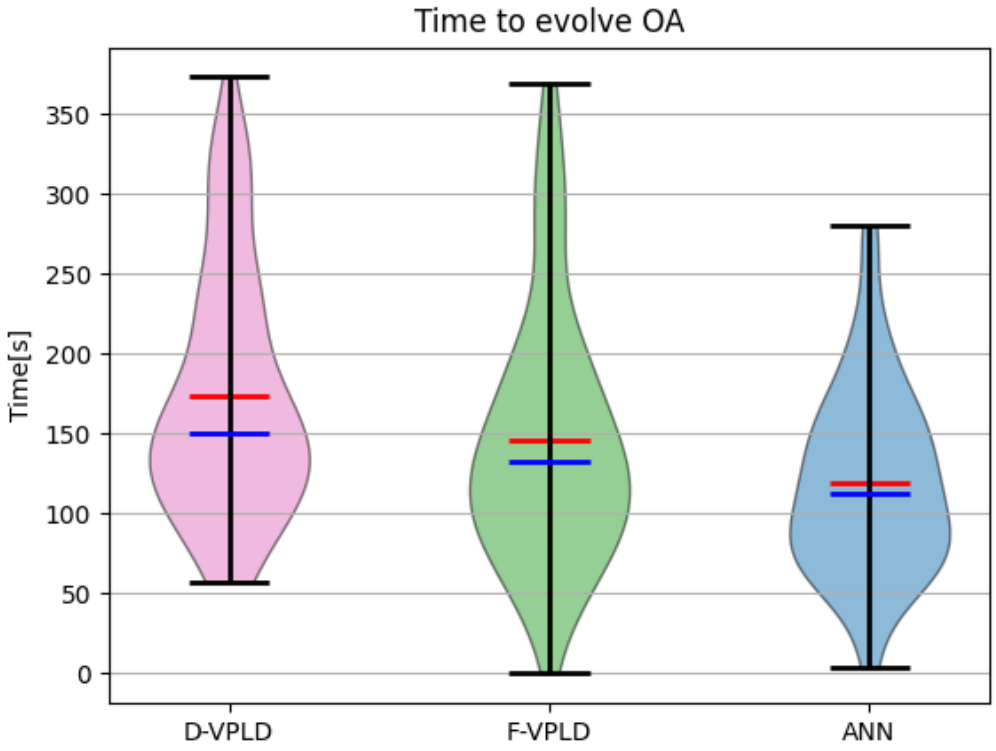

Fig. 4-54 Violin Plot of Time to Evolve for each run for OA results. Shows distribution of results, Maximum/Minimum time, median(blue line) and mean(red line)

Controller Performance

The controller performance is similar across all three controllers with the median fitness for all controllers at approximately 95% (Table 4-22). The ANN reaches the highest average level of fitness, but this is only 1-2% greater than the VPLD controllers (Table 4-22). For all three controllers, the level of fitness achieved for all one hundred results is very consistent, with coefficient of variance of, 0.04, 0.06, and 0.02 for the D-VPLD, F-VPLD and ANN respectively. The distribution of the fitness level in the results is shown in Fig. 4-55.

Table 4-22 The Fitness reached for the OA Task

Controller	Max	Mean	Median	CV
D-VPLD	97	93.54	94.46	0.04
F-VPLD	97.54	93.99	95.58	0.06
ANN	98.06	95.12	95.4	0.02

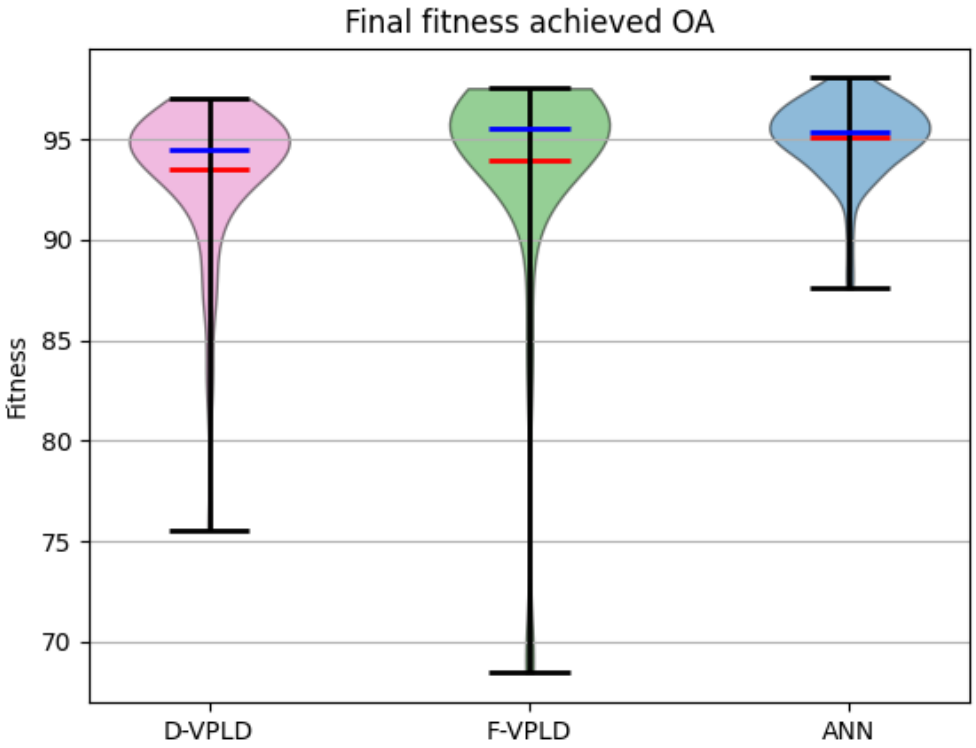

Fig. 4-55 Violin Plot of Final Fitness for each run for OA results. Shows distribution of results, Maximum/Minimum fitness, median(blue line) and mean(red line)

Examples of the stages of evolution for the control systems are shown below. All controllers at the beginning of the evolution process initially rotate on the spot or in a small circle (column one of Fig. 4-56). As the evolution continues the controllers begin slowly trying to explore the environment (column two of Fig. 4-56), in the case of the D-VPLD and F-VPLD the robots are traveling slowly and cannot explore a large area but are avoiding obstacles; specifically in the case of the F-VPLD example, the robot gets confused by the corner and can't decide whether to turn left or right so becomes stuck. For the ANN in this in-between stage, has evolved to avoid the obstacles in its path which can be seen because it is turning before a collision can occur, however its default behaviour is a large circular trajectory. Once the evolution process is complete, all three controllers can explore a large percentage of the total area and navigate around obstacles in the path of the robot to avoid a collision (column three of Fig. 4-56). Given further time the robots will eventually explore the entire environment.

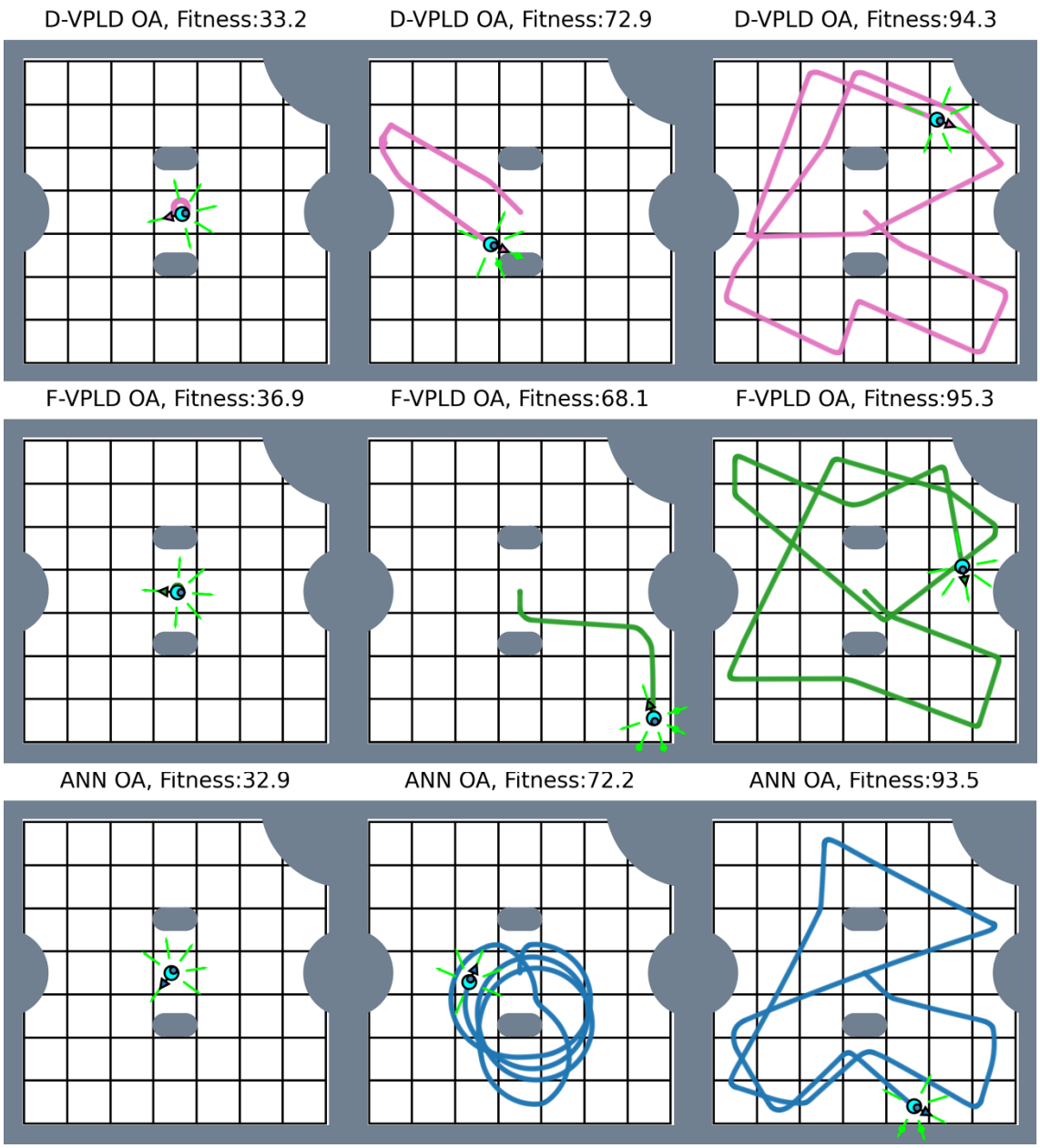

Fig. 4-56 Robot path for obstacle avoidance at different fitness levels. All controllers are run for 40 seconds from the same starting position and Orientation

A breakdown of the fitness of each example solution is shown in Table 4-23, Table 4-24 and Table 4-25. For the two VPLDs it is the area explored and speed component that prevent the configurations from reaching a high fitness level. For the ANN it is the area explored component and straight motion components that prevent the configurations from reaching a high fitness level. (Note: these are observations based on the example results which do not reflect every result collected for each controller).

Table 4-23 Fitness Breakdown of D-VPLD Example 2WD OA Results

Parameter	D-VPLD (C1)	D-VPLD (C2)	D-VPLD (C3)
F_{Area}	0.045	0.227	0.886
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Straight}$	0.001	0.992	0.99
$F_{Forward}$	0.001	1.0	1.0
F_{Speed}	0.108	0.445	1.0
F_{OA}	33.2	72.9	94.3

Table 4-24 Fitness Breakdown of F-VPLD Example 2WD OA Results

Parameter	F-VPLD (C1)	F-VPLD (C2)	F-VPLD (C3)
F_{Area}	0.023	0.182	0.841
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Straight}$	0.001	1.0	0.998
$F_{Forward}$	0.001	1.0	1.0
F_{Speed}	0.302	0.307	1.0
F_{OA}	36.9	68.1	95.3

Table 4-25 Fitness Breakdown of ANN Example 2WD OA Results

Parameter	ANN (C1)	ANN (C2)	ANN (C3)
F_{Area}	0.023	0.409	0.864
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Straight}$	0.001	0.005	0.988
$F_{Forward}$	0.001	1.0	1.0
F_{Speed}	0.107	1.0	1.0
F_{OA}	32.9	72.2	93.5

The evolved VPLD and ANN controllers are evaluated in a test environment and compared to their training environment behaviour, for the evaluation the robots are started in the same position and allowed to explore the environments for 60s. It can be seen in Fig. 4-57, Fig. 4-58 and Fig. 4-59 that the evolved behaviour of all three architectures, D-VPLD, F-VPLD and ANN is transferable from the training environment to the test environment; for all three architectures the robot can explore a large area within the environment without crashing.

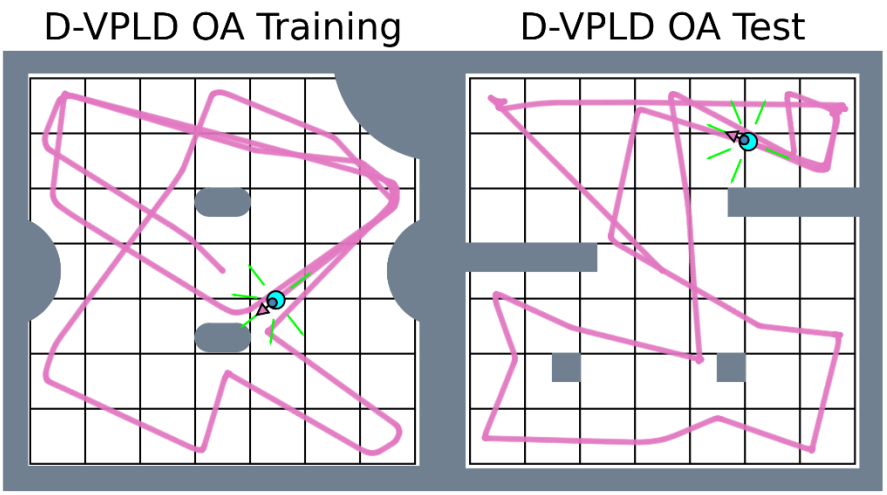

Fig. 4-57 2WD D-VPLD controller evaluated in the test environment

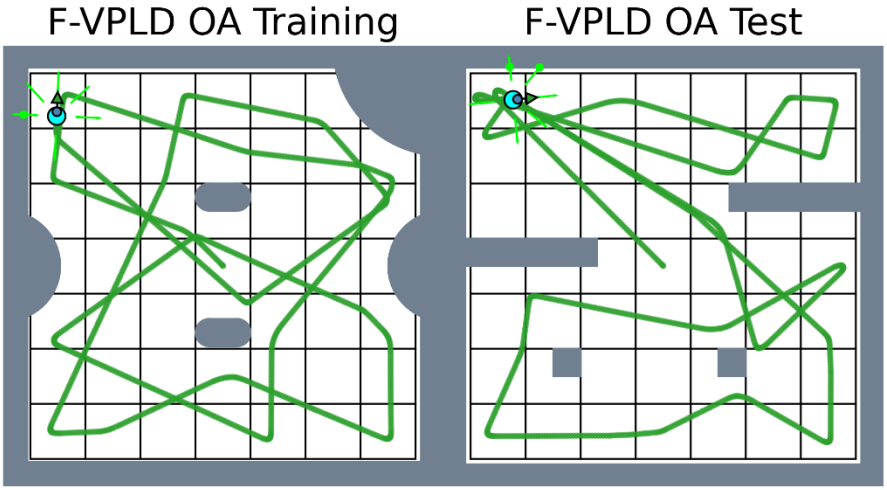

Fig. 4-58 2WD F-VPLD controller evaluated in the test environment

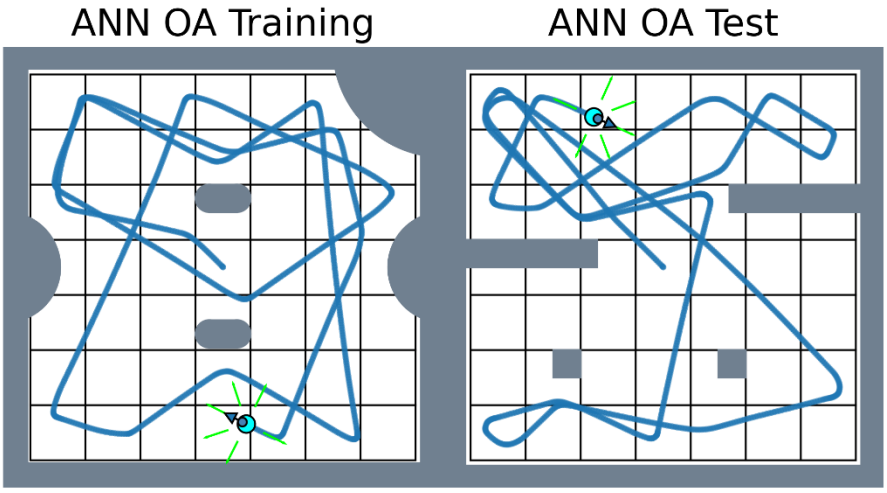

Fig. 4-59 2WD ANN controller evaluated in the test environment

4.2.8 Results: Light Following

The same GA run for 250 generations is used to evolve all three controllers, D-VPLD, F-VPLD and ANN. For the light following task the results used to compare the VPLDs and ANN are made up of one hundred individual runs of the GA for each controller. The simulation time used to evaluate the fitness of an individual is 10 seconds, all controllers use the same input data from the simulation and are evolved to output the required wheel velocities to drive the robot. An optimal controller for the light following task allows the robot to orientate itself to the heading of the light source and move towards it with a direct path.

Again, as with the obstacle avoidance task, the parameters used to evaluate the controllers are: a) evolutionary efficiency, and b) the controller performance. The evolved controller's performance is then evaluated in a test environment that differs from the training environment.

Evolutionary efficiency

For the light following control problem the F-VPLD has a much higher initial evolutionary rate reaching a high level of fitness very quickly. All three controllers have very similar average fitness levels achieved after the 250 generations (Fig. 4-60).

The D-VPLD and ANN controllers have a comparable average number of generations to converge to an optimal fitness level. The F-VPLD required less than half the generations the D-VPLD and ANN controllers did (Table 4-26). The LF results show that for all controllers this task is simpler to solve than the OA tasks, requiring fewer generations to evolve. This may be because the number of sensor inputs is less, (two for LF task and six for the OA task) the GA has to evolve the controllers so that all the sensors contribute to the resulting motor control signals positively, which is easier with less input signals.

Table 4-26 The Number of Generations to Evolve LF Task

Controller	Mean	Median	CV
D-VPLD	59.54	54	0.46
F-VPLD	23.12	20	0.85
ANN	63.55	61	0.38

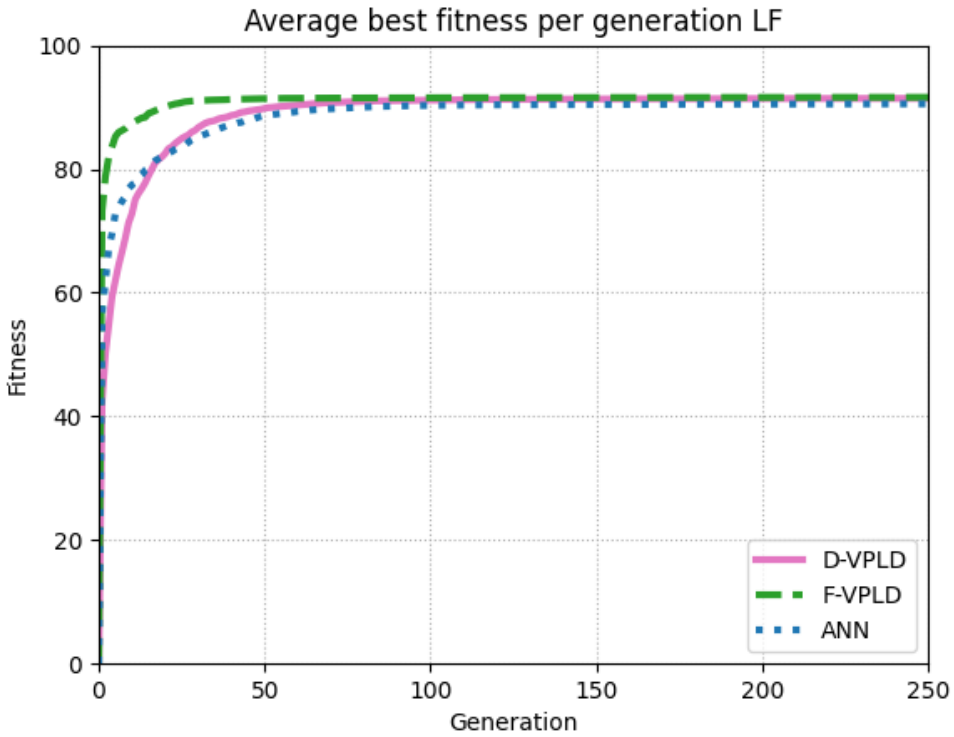

Fig. 4-60 A comparison of the LF fitness per generation of the three controllers

When looking at the computation time to evolve for the LF task (Table 4-27), all controllers require less than 30 seconds of computation time to evolve. The F-VPLD is the fastest requiring approximately 8 seconds on average to evolve an optimal solution. As expected with the difference in the required generations, the D-VPLD and ANN requires over double the computation time. The LF task required significantly less generations for each controller to evolve compared to the OA task, along with the simulation time only being 10 seconds versus 40 seconds for the OA task, both these factors explain the majority of the time difference between the two tasks. The OA task is also slower because the simulation of the IR sensors is significantly more computationally intensive than the light sensors which impacts the overall computation time of the EA. The distribution of the time to evolve solutions is shown in Fig. 4-61.

Table 4-27 Computation Time Required to Evolve in Seconds LF Task

Controller	Mean[s]	Median[s]	CV
D-VPLD	25.39	22.88	0.49
F-VPLD	8.22	6.84	0.98
ANN	22.96	21.84	0.42

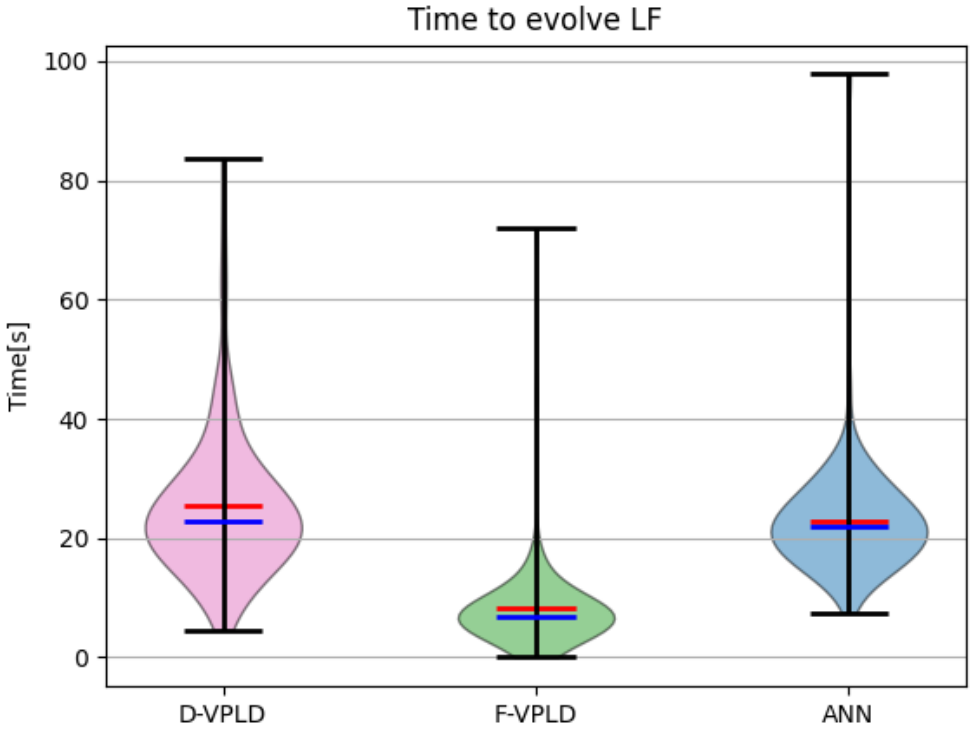

Fig. 4-61 Violin Plot of Time to Evolve for each run for LF results. Shows distribution of results, Maximum/Minimum time, median(blue line) and mean(red line)

Controller Performance

All three controllers have very similar average and max fitness levels achieved after the 250 generations. The D-VPLD has the highest max fitness level achieved, the F-VPLD has the highest average fitness level achieved but the difference between the results is minor (Table 4-28).

Table 4-28 The Fitness reached for the LF Task

Controller	Max	Mean	Median	CV
D-VPLD	94.97	91.43	91.84	0.020
F-VPLD	91.83	91.6	91.67	0.004
ANN	92.15	90.54	91.2	0.020

For all three controllers, the level of fitness achieved for all one hundred results is very consistent, as with the obstacle avoidance results. The distribution of the fitness level in the results is shown in Fig. 4-62.

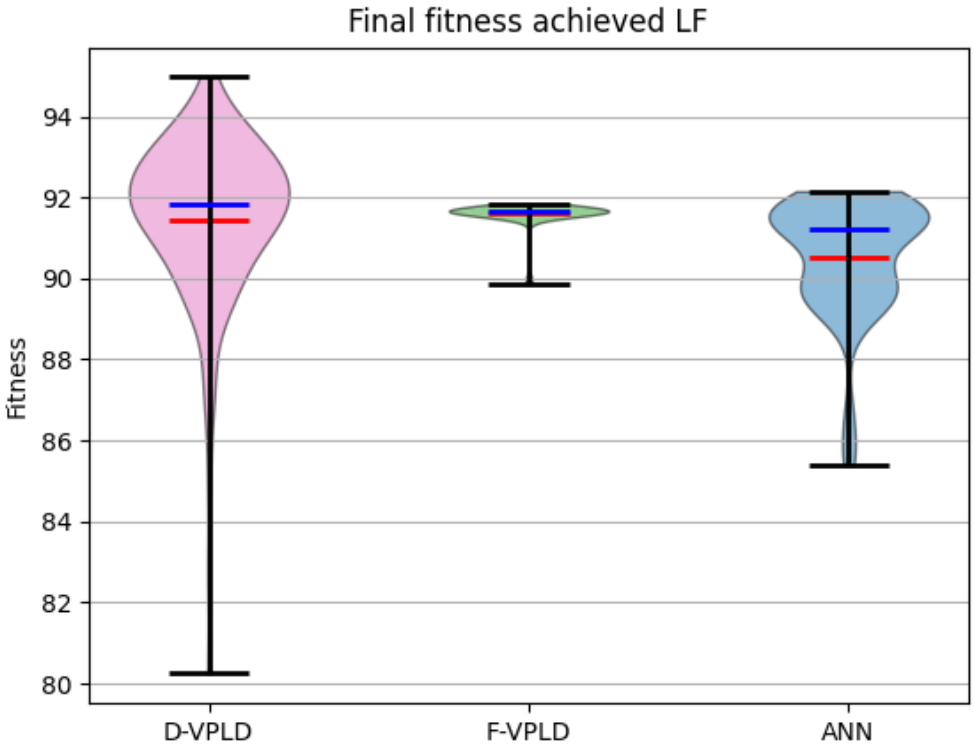

Fig. 4-62 Violin Plot of Final Fitness for each run for LF results. Shows distribution of results, Maximum/Minimum fitness, median(blue line) and mean(red line)

The different stages of the evolved light following control systems in simulation are shown below. All controllers at the early stages of the evolution process find solutions that are traveling towards the light but are moving at a slow speed, or are stopping periodically to rotate because they either: 1) lose track of the light due to noise; or 2) the light falls out of specific value range required for tracking (column one of Fig. 4-63). With further evolution time the controllers can move closer to the light, and from starting locations reach the light, however due to slow speeds combined with curved paths the robots do not reach a high level of fitness (column two of Fig. 4-63). When looking at example controllers at the end of evolution (column three of Fig. 4-63). it is difficult to distinguish between the D-VPLD, F-VPLD and ANN results. The controllers rotate at the start position to orientate to the light source then move in its direction in a straight line or a very minor arc. The robots stop at the light source.

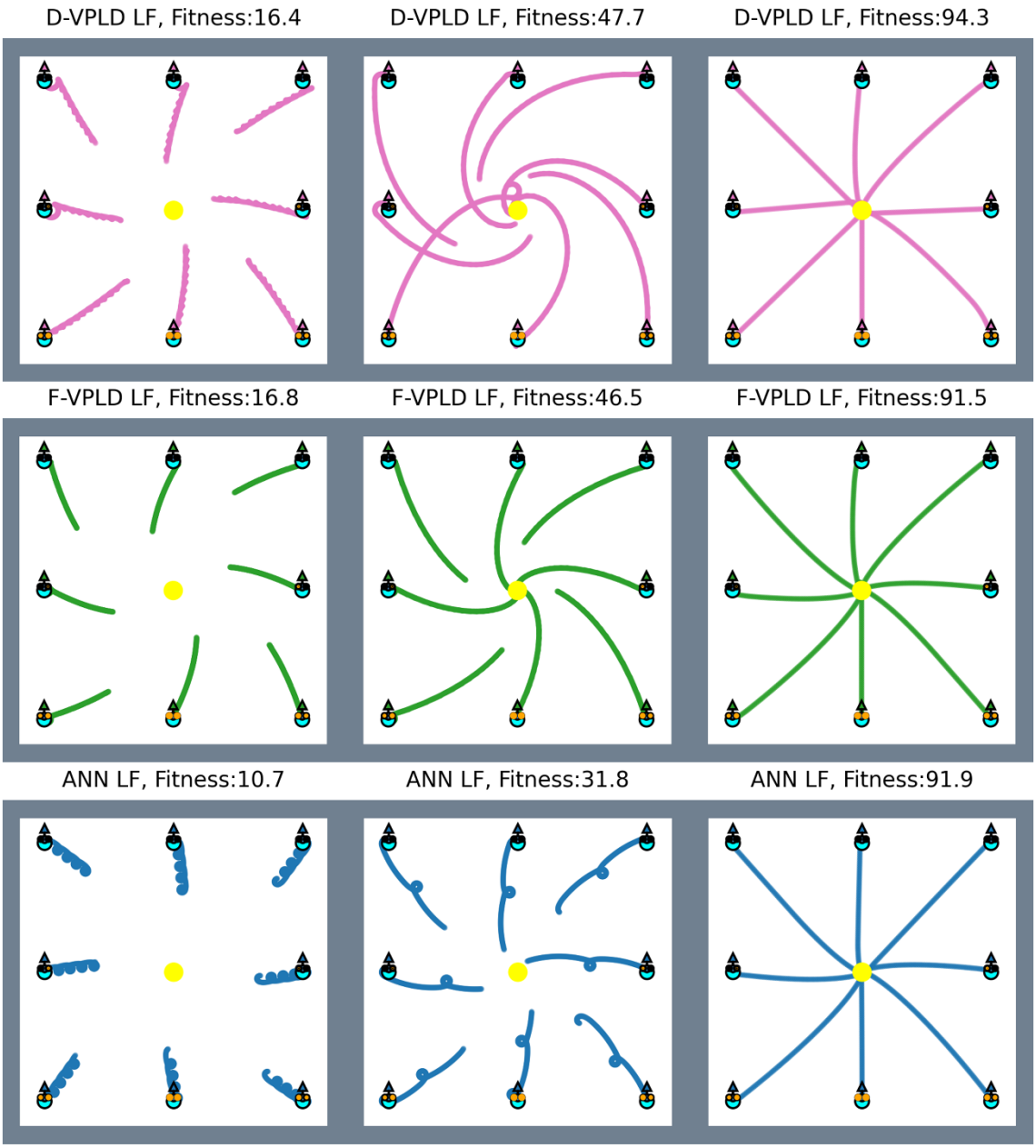

Fig. 4-63 Robot path of evolved controllers for light following task at different fitness levels, all controllers are run for 10 seconds from the same starting positions and orientations

Each component of the light following controller fitness is broken down for each example solution in Table 4-29, Table 4-30 and Table 4-31. For all three controllers it is the path component of fitness that prevents the configurations from reaching a high level of fitness. To maximize the path fitness component the robot must move quickly and directly to the light source, slow speed or curved trajectories will decrease the configurations fitness; this is what is seen in the example results (Fig. 4-63).

Table 4-29 Fitness Breakdown of D-VPLD Example 2WD LF Results

Parameter	D-VPLD (C1)	D-VPLD (C2)	D-VPLD (C3)
$F_{Reached\ Light}$	0.56	0.823	0.996
F_{Path}	0.44	0.572	0.946
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Forward}$	0.646	0.994	1.0
F_{LF}	16.4	47.7	94.3

Table 4-30 Fitness Breakdown of F-VPLD Example 2WD LF Results

Parameter	F-VPLD (C1)	F-VPLD (C2)	F-VPLD (C3)
$F_{Reached\ Light}$	0.452	0.848	0.988
F_{Path}	0.393	0.59	0.926
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Forward}$	0.915	0.9	1.0
F_{LF}	16.8	46.5	91.5

Table 4-31 Fitness Breakdown of ANN Example 2WD LF Results

Parameter	ANN (C1)	ANN (C2)	ANN (C3)
$F_{Reached\ Light}$	0.311	0.653	0.998
F_{Path}	0.337	0.474	0.921
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Forward}$	1.0	1.0	1.0
F_{LF}	10.7	31.8	91.9

As with the OA task the evolved VPLD and ANN controllers are evaluated in a test environment and compared to their training environment behaviour. For the LF task there is a limited number of variations to the environment, only the light position and the starting positions can be altered. For the evaluation the robots are started in the same positions but with alternative orientations, the robots are given 10 seconds to reach the light source. It can be seen in Fig. 4-64, Fig. 4-65 and Fig. 4-66 that again as in the OA results the evolved behaviour of all three architectures, is transferable from the training environment to the test environment. For all three architectures the robot orientates itself towards the light source and moves directly towards it, the robots stop at the light source.

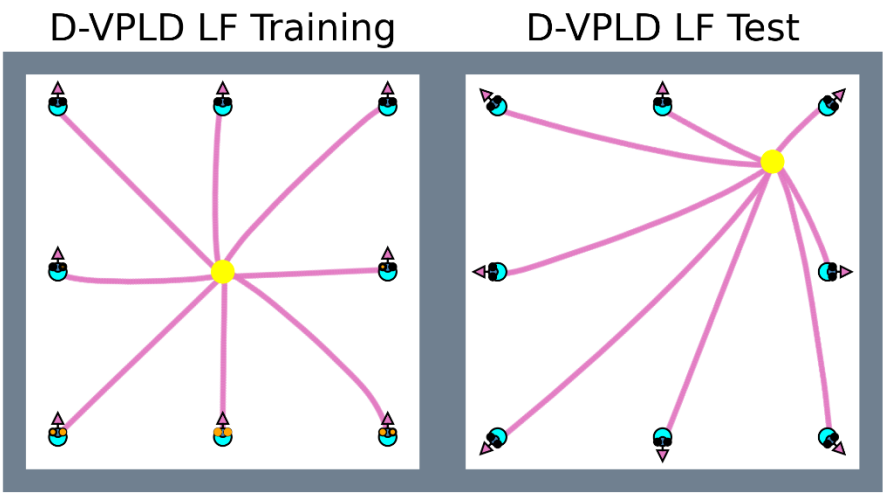

Fig. 4-64 2WD D-VPLD controller evaluated in the LF test environment

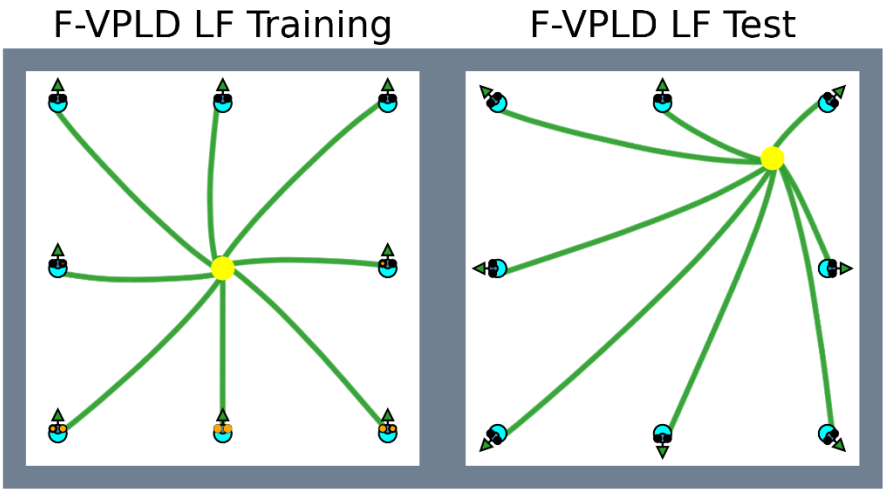

Fig. 4-65 2WD F-VPLD controller evaluated in the LF test environment

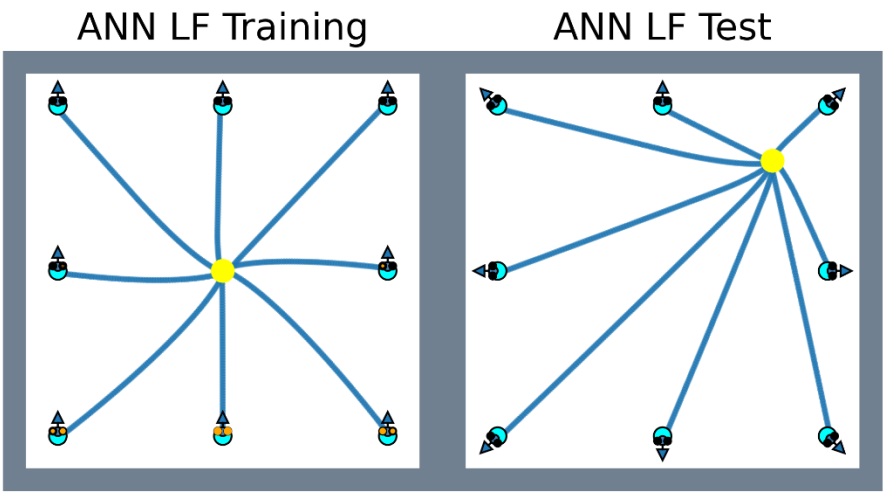

Fig. 4-66 2WD ANN controller evaluated in the LF test environment

4.2.9 Results: Obstacle Avoidance while Light Following

For the combined obstacle avoidance and light following task, the same GA is run for 250 generations to evolve the three control systems, D-VPLD, F-VPLD and ANN. To obtain the data to compare the robotic control systems, one hundred individual runs of the GA for each controller are performed to find the required number of generations and time to evolve, as well as the max fitness level achieved. The simulation time used to evaluate the fitness of an individual is 10 seconds, all controllers use the same input data from the simulation and are evolved to output the required wheel velocities to drive the robot. This combined problem uses the same fitness function as the light following problem, therefore an optimal controller for the combined task allows the robot to orientate itself to the heading of the light source and move towards it with a short path. However, in this task the robot can no longer travel directly to the light, it must navigate around obstacles in its path to reach the light.

The parameters used to evaluate the controllers are: a) evolutionary efficiency, and b) the controller performance The deployed controller performance in a test environment that differs from the training environment is also investigated.

Evolutionary efficiency

It can be seen (Fig. 4-67) that the F-VPLD has a much higher initial evolutionary rate compared to the D-VPLD and ANN, and the F-VPLD reaches a higher level of fitness on average. When looking at the total generations to converge to an optimal solution the F-VPLD requires the least, on average the D-VPLD requires 30 generations more, and the ANN requires 16 more (Table 4-32).

Table 4-32 The Number of Generations to Evolve OA-LF Task

Controller	Mean	Median	CV
D-VPLD	157.97	162.5	0.34
F-VPLD	127.56	117.5	0.47
ANN	144.23	137.5	0.32

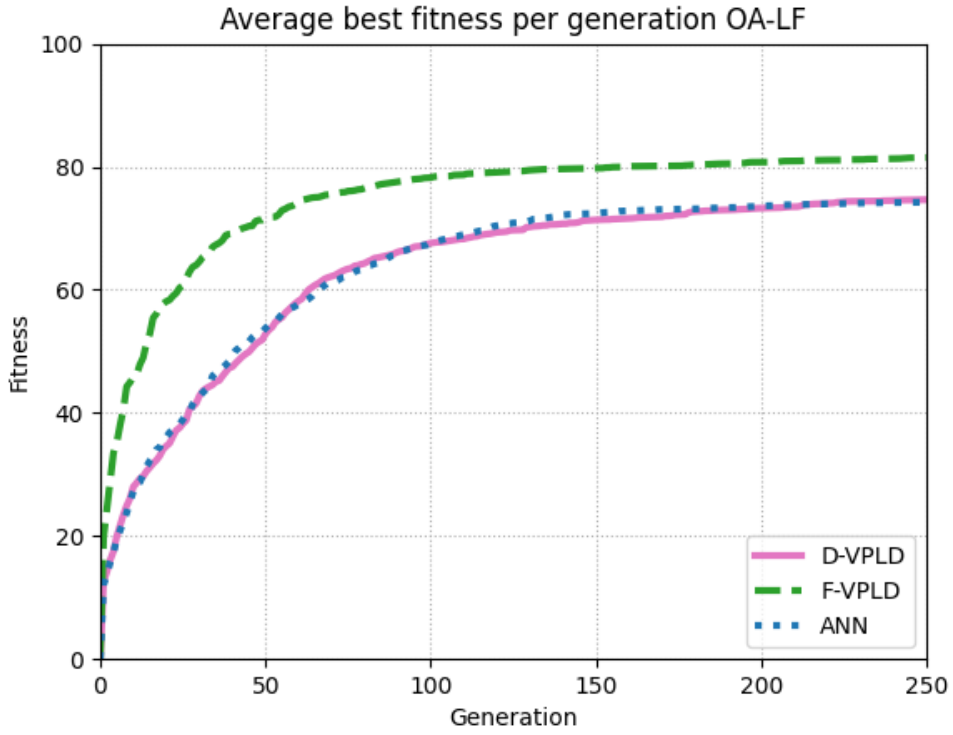

Fig. 4-67 A comparison of the OA-LF fitness per generation of the three controllers

When looking at the computation time to evolve for the combined OA-LF task (Table 4-33), the F-VPLD median time is approximately 61 seconds of computation time to evolve. The F-VPLD is the fastest because it required the least generations to evolve. The ANN required approximately a further 7 seconds on average to evolve. The D-VPLD required approximately 31 seconds on average more computation time to evolve than the F-VPLD. Two factors can be attributed to the increased computation time for the D-VPLD to evolve; 1) the added generations required to evolve the D-VPLD, and 2) the computation required to convert the 8 sensor inputs and 2 motor outputs between binary and floating-point values. The distribution of the time to evolve solutions is shown in Fig. 4-68.

Table 4-33 Computation Time Required to Evolve in Seconds OA-LF Task

Controller	Mean[s]	Median[s]	CV
D-VPLD	98.44	100.94	0.36
F-VPLD	67.08	61.31	0.52
ANN	73.83	69.47	0.35

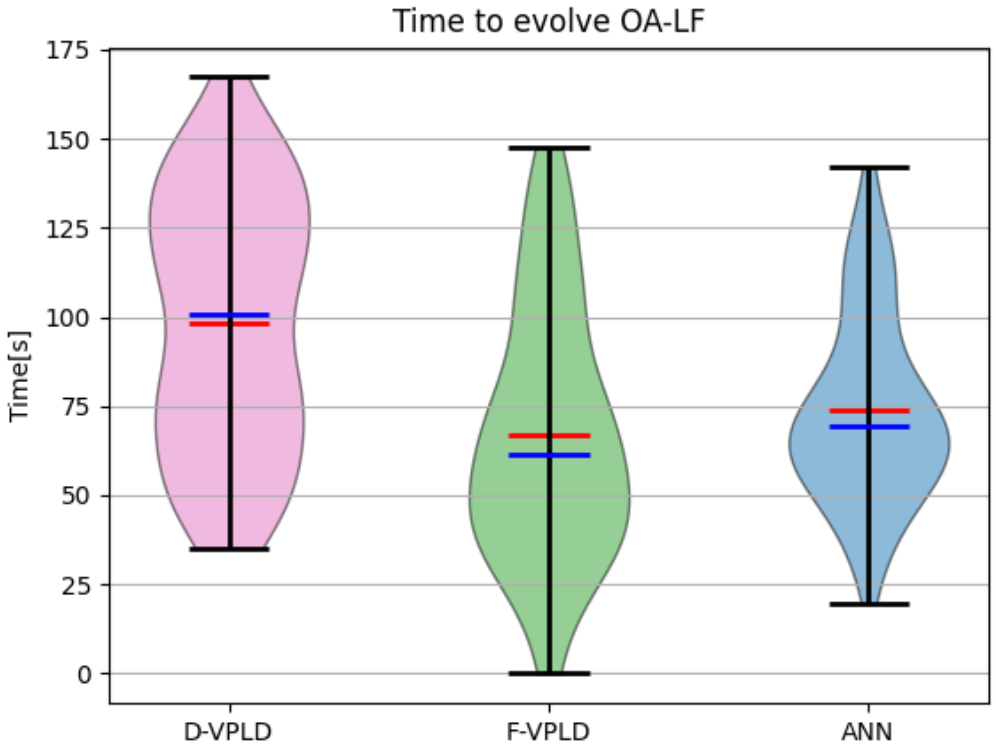

Fig. 4-68 Violin Plot of Time to Evolve for each run for OA-LF results. Shows distribution of results, Maximum/Minimum time, median(blue line) and mean(red line)

Controller Performance

In the obstacle avoidance while light following problem, the F-VPLD controller reaches a higher level of fitness on average (Table 4-34). The D-VPLD and ANN average fitness levels are comparable. All controllers achieve a high max fitness.

Table 4-34 The Fitness reached for the OA-LF Task

Controller	Max	Mean	Median	CV
D-VPLD	88.22	74.64	76.93	0.13
F-VPLD	89.77	81.49	83	0.07
ANN	86.12	74.24	75.71	0.13

For all three controllers, the level of fitness achieved for all one hundred results is consistent still but less so than for the individual obstacle avoidance and light following tasks. The F-VPLD has the lowest coefficient of variance at approximately 0.07, while the ANN and D-VPLD have a comparable coefficient of variance at approximately 0.13. The distribution of the fitness level in the results is shown in Fig. 4-69.

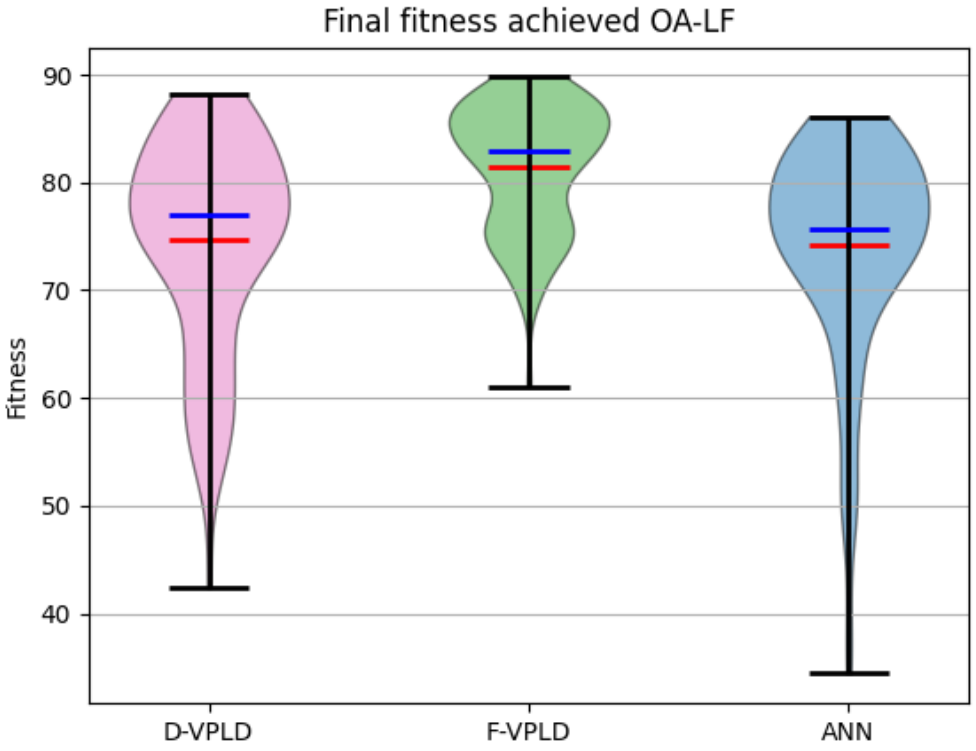

Fig. 4-69 Violin Plot of Final Fitness for each run for OA-LF results. Shows distribution of results, Maximum/Minimum fitness, median(blue line) and mean(red line)

The stages of the evolved control systems for the combined obstacle avoidance and light following task are shown below. In the initial stages of evolution (column one of Fig. 4-70), as was seen with pure light following task the solutions are traveling towards the light but are moving at a slow speed, or are stopping periodically to rotate. Part way through the evolutionary process (column two of Fig. 4-70) the controllers can avoid the obstacles but have non-direct paths to the light and do not reach the light source from all start positions. The example configurations for each controller after 250 generations of evolution can all reach the light source while avoiding the obstacles in the robots path (column three of Fig. 4-70).

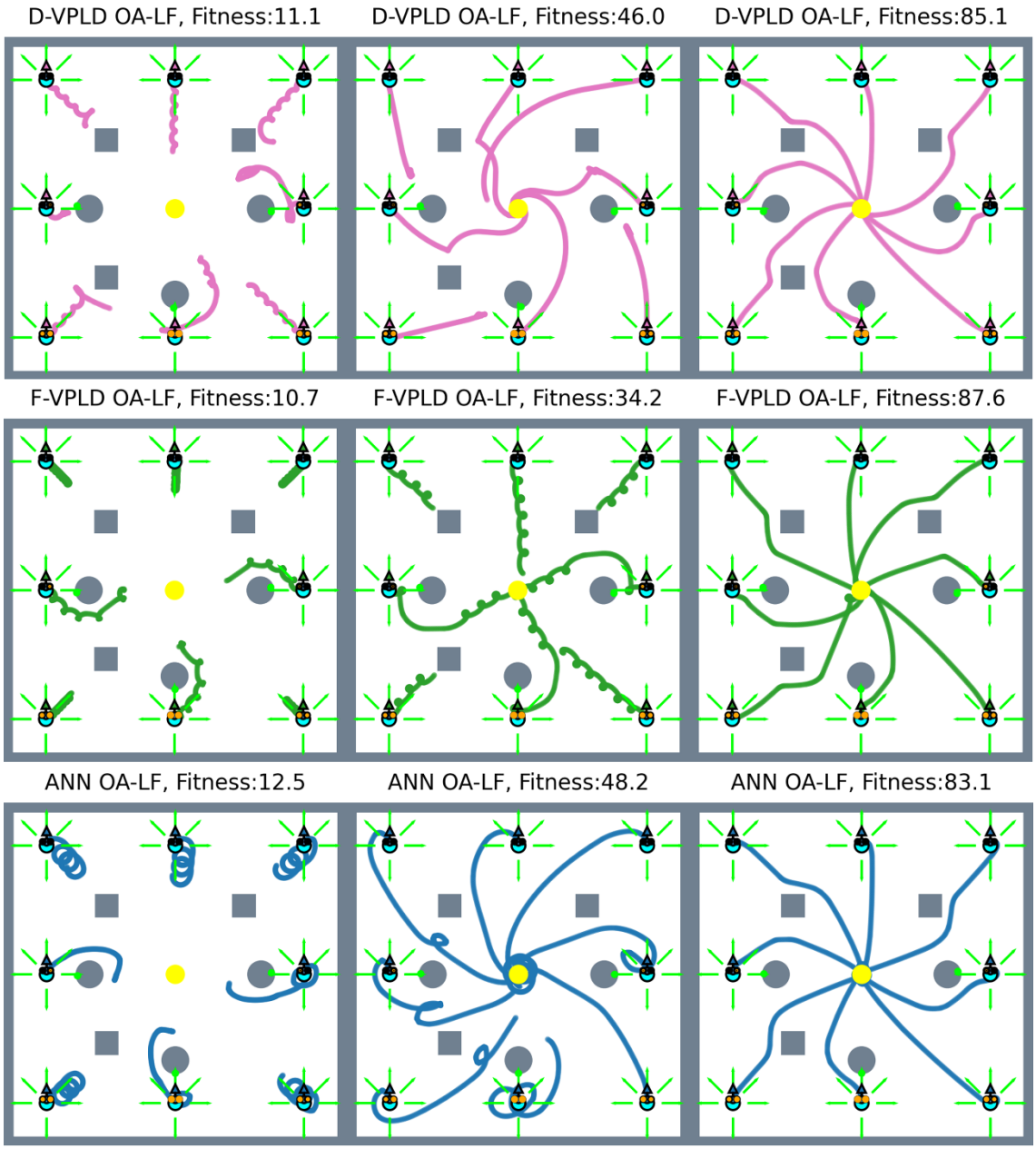

Fig. 4-70 Robot path of evolved controllers for combined task at different fitness levels, all controllers are run for 10 seconds from the same starting positions and orientations

Each component of the controller fitness for the combined obstacle avoidance and light following task is broken down for each example solution in Table 4-35, Table 4-36 and Table 4-37. As with the pure light following controllers, for all three controllers it is the path component of fitness that prevents the configurations from reaching a high level of fitness. As the OA-LF task uses the same fitness function as the LF, comparing the results, as expected the OA-LF controllers get a lower fitness because the robot can no longer travel in a straight line to the light, it must take an extended path decreasing that component of fitness relative to the LF results.

It is also observed when viewing the robots in simulation that for the controllers to direct the robots to the light source and avoid obstacles, traveling at the max speed is not possible which also impacts the path component of the fitness function. However, as can be seen in Fig. 4-70. All the controllers can successfully complete the combined OA-LF task.

Table 4-35 Fitness Breakdown of D-VPLD Example 2WD OA-LF Results

Parameter	D-VPLD (C1)	D-VPLD (C2)	D-VPLD (C3)
$F_{Reached\ Light}$	0.386	0.742	0.989
F_{Path}	0.391	0.605	0.861
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Forward}$	0.693	0.904	1.0
F_{LF}	11.1	46.0	85.1

Table 4-36 Fitness Breakdown of F-VPLD Example 2WD OA-LF Results

Parameter	F-VPLD (C1)	F-VPLD (C2)	F-VPLD (C3)
$F_{Reached\ Light}$	0.328	0.669	0.989
F_{Path}	0.324	0.516	0.886
$F_{Avoidance}$	1.0	0.862	1.0
$F_{Forward}$	0.946	1.0	1.0
F_{LF}	10.7	34.2	87.6

Table 4-37 Fitness Breakdown of ANN Example 2WD OA-LF Results

Parameter	ANN (C1)	ANN (C2)	ANN (C3)
$F_{Reached\ Light}$	0.323	0.838	0.973
F_{Path}	0.345	0.561	0.854
$F_{Avoidance}$	1.0	1.0	1.0
$F_{Forward}$	0.999	0.97	1.0
F_{LF}	12.5	48.2	83.1

The evolved VPLD and ANN controllers are evaluated in a test environment with different obstacles located at new positions within the environment, the light position is also moved. For the test evaluation the robots are started in alternative positions and are given 15s to reach the light source without crashing into an obstacle. It can be seen in Fig. 4-71, Fig. 4-72 and Fig. 4-73 that the evolved behaviour of all three architectures is transferable from the training environment to the test environment; however the controllers have evolved a clear strategy to move either left or right around an object which doesn't always result in the optimal path to the light source, also if the evolved strategy is not a possible option to move around a given obstacle, then the robot would become stuck and begin spinning to try and orientate itself to, and move towards the light source.

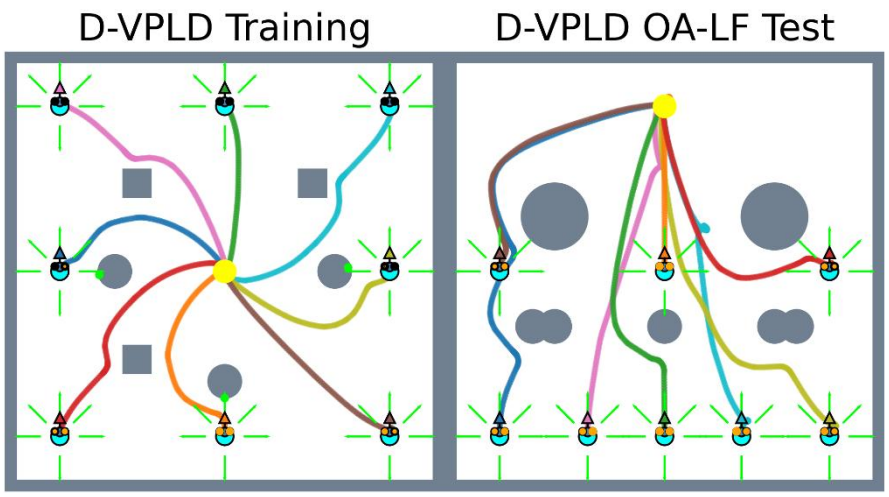

Fig. 4-71 2WD D-VPLD controller evaluated in the OA-LF test environment

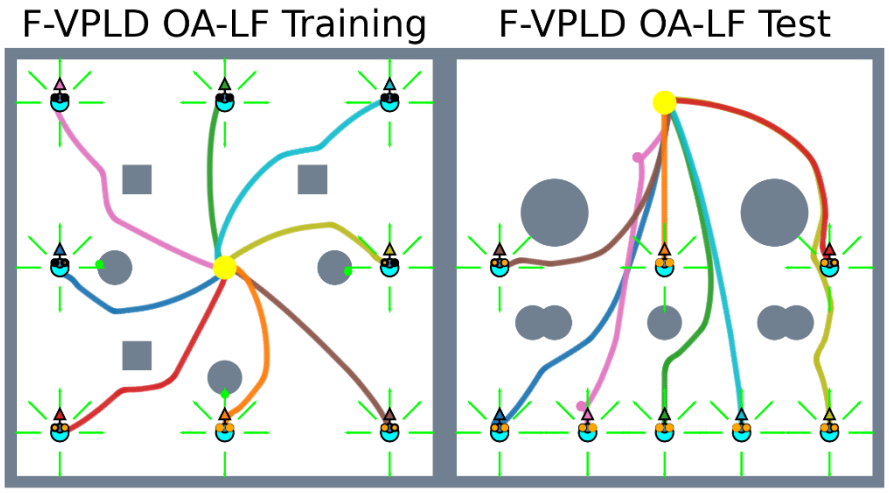

Fig. 4-72 2WD F-VPLD controller evaluated in the OA-LF test environment

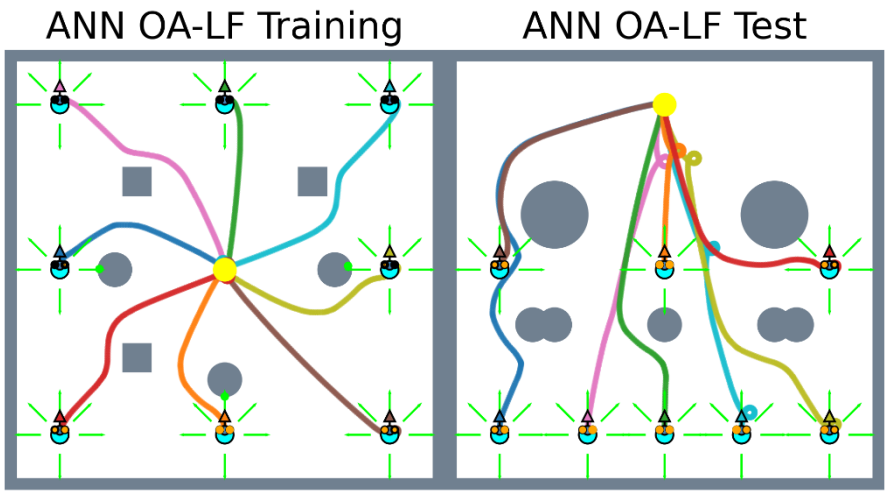

Fig. 4-73 2WD ANN controller evaluated in the OA-LF test environment

4.2.10 Conclusions

In this chapter a comparison was made between floating-point and digital versions of the VPLD, and an ANN used to control a 2WD mobile robot for navigation tasks. The robotic controllers are evolved in a simulation for, 1) obstacle avoidance, 2) light following, and 3) light following while avoiding obstacles.

For the individual obstacle avoidance and light following problems all the controllers reached a similar controller performance successfully completing the given tasks. In the obstacle avoidance task, all controllers had a comparable evolutionary efficiency, meaning they all reached an optimal fitness level in the same number of generations. In the light following task the F-VPLD evolved in half the generations required by the D-VPLD and ANN controllers.

For the combined problem of light following while avoiding obstacles, the F-VPLD was the best, reaching the highest average fitness across the 100 results in the least generations. When comparing the light following and combined obstacle avoidance and light following tasks, as was expected, using the same fitness function for both the tasks, the combined obstacle avoidance and light following results are at a lower fitness level. The fitness level is reduced because the robots can no longer take the optimal direct path to the light source, they had to instead navigate around the obstacles to reach the light, thus extending the path travelled.

When looking at the examples of the final configurations of each control system after evolution, it is impossible to distinguish the three controllers from each other, as there is no distinct behaviour that could be contributed to either of the VPLDs or the ANN. Considering the D-VPLD used a binary system it may have been expected to have a more stepped logic response either choosing to move in a particular direction or rotate but instead like the F-VPLD and ANN that uses floating point data it has a fluid motion.

Further comparing the three problems, the combined obstacle avoidance & light following task took the most generations to evolve, this is expected as it is the most complex of the three problems to solve. The obstacle avoidance & light following task has the greatest amount of sensor input data which the controllers are required to interpret in order to correctly operate the robot. If you were to create a combined obstacle avoidance & light following controller that used subsumption, meaning the obstacle avoidance controller is combined with the light following controller by a top-level

decision entity, the generations required to evolve the individual OA and LF controllers is comparable to the generations required to evolve the monolithic architecture implemented in this research.

The optimal evolved VPLD and ANN controllers are also deployed in new environments in the simulation, for the individual obstacle avoidance and light following task the behaviour evolved in the training environment transferred well to the new test environment. In the combined obstacle avoidance while light following, the evolved controllers allowed the robot from all start positions to reach the light source, however due to the nature of the evolved behaviour and the minimal information the robot has about its environment the most optimal path is not always taken.

It has been shown in this chapter that the VPLD either using digital logic, or floating-point arithmetic, is a viable alternative to an ANN for the control of a 2WD mobile robot.

Chapter 5

Chapter 5: Virtual Programmable Logic Device for Pattern Recognition

This chapter investigates how the VPLD performs in comparison to an ANN for pattern recognition. Pattern recognition is an important aspect of machine learning as it has many real-world applications such as, OCR for language translation, and patient diagnosis based on medical imaging (x-rays, MRI or CT scans). The VPLD and ANN are compared in two different pattern recognition applications, looking specifically at 1) character recognition, and 2) melanoma detection. Character recognition is a classic pattern recognition problem in the machine learning domain, which has been well studied. As described in section 2.1.4, EHW architectures have been evolved for various pattern recognition tasks and specifically binary character recognition, however these EHW where only evolved to recognize the characters A-P, in this research the ANN and VPLD are evolve to recognize the characters A-Z. Machine learning pattern recognition models are often used in medical imaging, one such task is melanoma detection. In this research the accuracy of the VPLD and ANN is investigated for the binary classification of melanomas, the skin lesions are either benign (non-cancerous) or malignant (cancerous melanoma) skin lesions.

For both the character recognition and melanoma detection, two versions of the VPLD are investigated: 1) the D-VPLD using traditional digital logic similar to EHW research; and 2) the F-VPLD using floating point data types and mathematical functions. The F-VPLD architecture using floating point mathematics, can run on a CPU without needing additional resources, as opposed to an EHW system that requires an FPGA with a high number of logic elements to generate multiple floating point mathematical operations. The accuracy and evolutionary efficiency of the D-VPLD and F-VPLD systems will be compared to an ANN system. The ANN, D-VPLD and F-VPLD are evolved with the same GA.

5.1. Character Recognition

This section describes the character recognition problem being investigated and the system architectures and respective chromosomes of the D-VPLD, F-VPLD and the ANN used. As with the previous robotic control instances, all the controllers are evolved using the same GA. The aim of the GA is to find optimal configurations for the D-VPLD, F-VPLD and ANN architectures allowing these systems to recognize all the characters A-Z in the alphabet. The results of the evolved three systems are analysed, and a comparison is made between the D-VPLD, F-VPLD and ANN.

5.1.1 Character Recognition Method

The characters (Fig. 5-1) used are 7x5 binary characters A-Z. These alphabet characters are fed into the ANN, D-VPLD and F-VPLD systems to obtain a set of 26 outputs indicating which letter is recognised.

Fig. 5-1 7x5 character images, 35 pixels total. Used for the character recognition problem

Each character is converted into a binary array (0-black pixel or 1-white pixel) with a length of 35 (Fig. 5-2). This process is performed to create a suitable data input for all the systems (D-VPLD, F-VPLD and ANN), as the D-VPLD is a digital system it requires a digital input. Note pseudo binary is used where an integer can have only two values, 0 and 1.

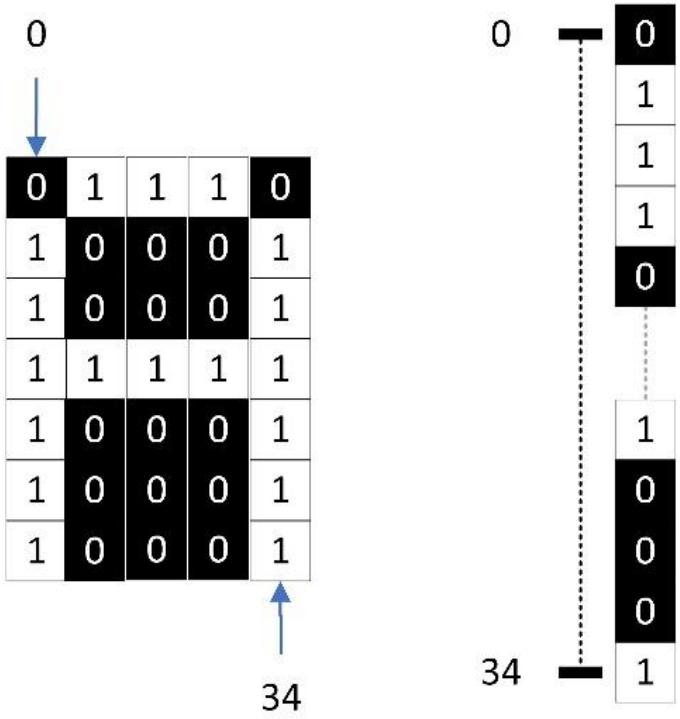

Fig. 5-2 Flattening 7x5 image to array of length 35

The system architectures have 26 outputs, one for each letter of the alphabet, with the output that has the highest value determining which letter is selected. Several architectures for the D-VPLD, F-VPLD and ANN are evaluated using either, one or three combined outputs for each letter, with the latter proving to be more effective. A comparison of all the VPLD architectures evaluated can be found in Chapter 6. The three outputs for each letter are combined using either the mean or maximum of the three outputs. The 26 outputs for each letter are then passed through a SoftMax function. The SoftMax function transforms the 26 outputs into values between 0 and 1, such that all the outputs sum to 1, creating a normalized probability distribution. The best design for each architecture, D-VPLD, F-VPLD and ANN is discussed in detail below.

5.1.2 D-VPLD and F-VPLD Classifier Architecture

The D-VPLD and F-VPLD architectures have the same layout (Fig. 5-3) except the D-VPLD FAB uses binary numbers and digital logic, whereas the F-VPLD FAB uses floating-point numbers and mathematical functions. The architectures have 35 binary inputs one for each pixel of the character, and 78 outputs, which undergo post processing to have a final total of 26 outputs one for each letter in the alphabet. Both architectures have two layers, with 52 FABs in the first layer and 78 FABs in the second layer. The 78 FABs in the second layer are grouped into threes, (26x3). These 26 grouped outputs are fed into the SoftMax function to calculate the final prediction. The VPLD FABs are configured by the CBS.

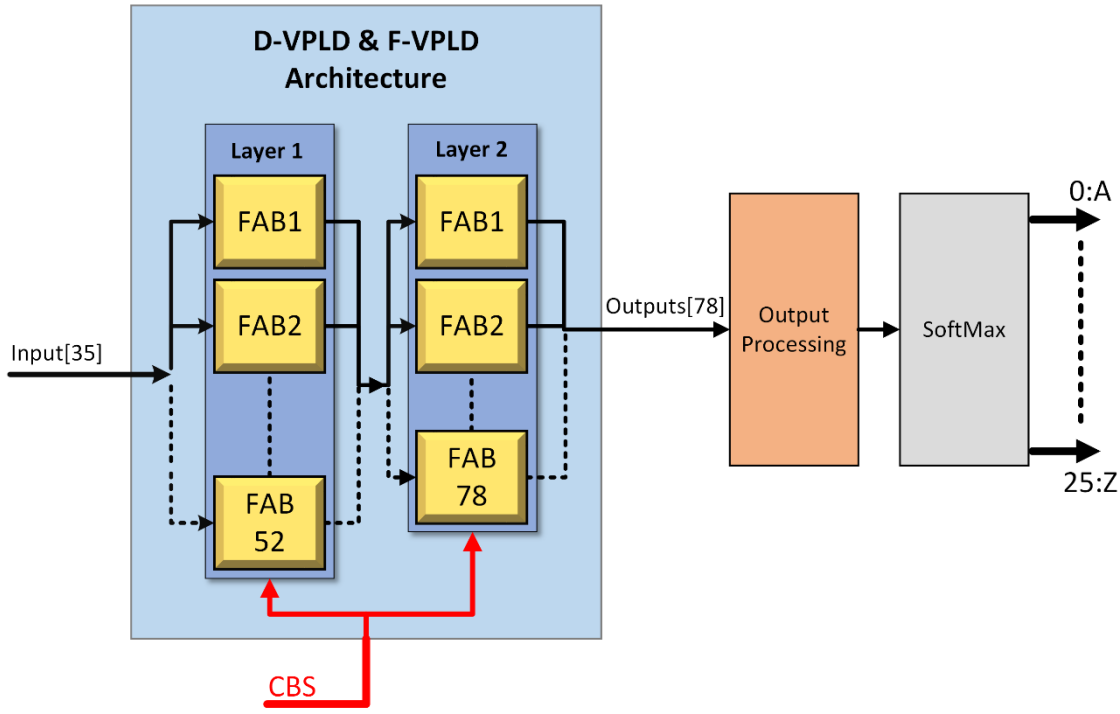

Fig. 5-3 D-VPLD and F-VPLD 52-78-layer architecture. Layer one contains 52 FABs, layer 2 contains 78 FABs

All the FABs for both the D-VPLD and F-VPLD contains four multiplexers and a function element. The FE used in the FABs is a LUT which is comprised of combinational logic (D-VPLD) or mathematical functions (F-VPLD). The D-VPLD LUT (Table 5-1) uses 31 combinational logic functions producing a single bit output from the four binary inputs. The F-VPLD LUT (Table 5-2) uses mathematical functions utilizing the addition, subtraction and multiplication operators producing a single floating-point value from the four inputs. The function f is the rectified linear unit (ReLu) function used in machine learning. The F-VPLD LUT contains 21 functions.

Table 5-1 Character Recognition D-VPLD FE LUT

No.	Logic Function	No.	Logic Function
0	(A&B)^(C|D)	16	((!A)&B)|((!C)&D)
1	A&B&C&D	17	(A&(!B))|(C&(!D))
2	!(A&B&C&D)	18	((!A)&(!D))|((!B)&(!C))
3	(!A)&B&C&D	19	(A^B)&(C^D)
4	A&(!B)&C&D	20	(A^C)&(B^D)
5	A&B&(!C)&D	21	(A^D)&(C^B)
6	A&B&C&(!D)	22	((!A)&B)^((!C)&D)
7	A&(!B)&(!C)&(!D)	23	(A&(!B))^(C&(!D))
8	(!A)&B&(!C)&(!D)	24	((!A)&(!D))^((!B)&(!C))
9	(!A)&(!B)&C&(!D)	25	(A|B)&(C|D)
10	(!A)&(!B)&(!C)&D	26	(A|C)&(B|D)
11	A|B|C|D	27	(A|D)&(C|B)
12	!(A|B|C|D)	28	(A^B)&(C^D)
13	(A&B)|(C&D)	29	(A^C)&(B^D)
14	(A&C)|(B&D)	30	(A^D)&(C^B)
15	(A&D)|(B&C)		

Table 5-2 Character Recognition F-VPLD FE LUT

No.	Logic Function	No.	Logic Function
0	*f*((A+B)*(C-D))	11	*f*(-A)+B+C+D)
1	*f*(A+B+C+D)	12	*f*(A+(-B)+C+D)
2	*f*(-(A+B+C+D))	13	*f*(A+B+(-C)+D)
3	*f*(A*B*C*D)	14	*f*(A+B+C+(-D))
4	*f*(-(A*B*C*D))	15	*f*((A*B)+(C*D))
5	*f*((A+B)*(C+D))	16	*f*((A*C)+(B*D))
6	*f*((A+C)*(B+D))	17	*f*((A*D)+(B*C))
7	*f*((A+D)*(B+C))	18	*f*((A*B)-(C*D))
8	*f*((A+B)-(C+D))	19	*f*((A*C)-(B*D))
9	*f*((A+C)-(B+D))	20	*f*((A*D)-(B*C))
10	*f*((A+D)-(B+C))		

The outputs of the 78 FABs in the final layer are grouped into threes, each group of three outputs are combined producing 26 total outputs. Several combination methods of the three grouped outputs are tested, with the best method for each architecture used. For the D-VPLD and F-VPLD the combination function is the mean method (Fig. 5-4) where the mean value of the three outputs is taken and then passed through the SoftMax function to provide an output probability for each character. For the D-VPLD the combined value ranges from 0 to 3, whereas the F-VPLD mathematical processing of the binary inputs produce an output that can have a range from 0 to 256 for the F-VPLD.

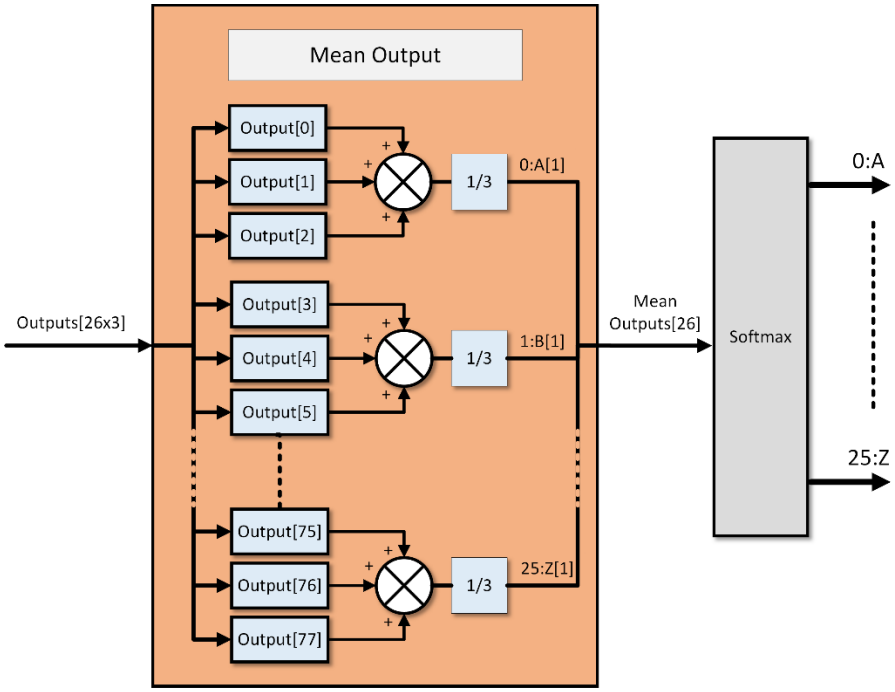

Fig. 5-4 Mean method used by the D-VPLD and F-VPLD architectures to group the 26x3 outputs into one output for each character

Chromosome and Crossover

The chromosome for the VPLDs is the CBS, and as the architectures are identical the chromosome is the same for both the D-VPLD and F-VPLD. As seen in Chapter 3 and Chapter 4, a set of two-dimensional arrays of integers is used to store the chromosome. The search space for the D-VPLD and F-VPLD CBS is different for the character recognition problem, due to having different sized FE LUTs, otherwise the search space would be the same for both architectures. The search space is calculated (using equation 4.11) as follows,

$$S = (L1_{inputs}{}^{L1_{MUXs}})(\,L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(\,L2_{functions}{}^{L2_{FEs}})$$

$$S_{D-VPLD} = (35^{4*52})(\,31^{52})(52^{4*78})(\,31^{78})$$

$$S_{D-VPLD} = 2.73 \cdot 10^{1050}$$

$$S_{F-VPLD} = (35^{4*52})(\,21^{52})(52^{4*78})(\,21^{78})$$

$$S_{F-VPLD} = 2.81 \cdot 10^{1028}$$

The crossover operation performed is the same multipoint method used in the 2WD robotic control problem (section 4.2.1). The child chromosomes are mutated with a 0.5% mutation rate.

5.1.3 ANN Classifier Architecture

The ANN like the VPLDs, uses a character image converted to an array in a binary form (0-black pixel and 1-white pixel) for its inputs. The output is a set of 26 values, one for each letter in the alphabet. The output value with the highest value determines the selected character. The architecture of the ANN is shown in Fig. 5-5. It is a 35-52-78 architecture; this means there are 52 neurons in the hidden layer and 78 neurons in the output layer. The outputs of the ANN go through the same post processing as the VPLDs, where the 78 neurons in the final layer are grouped into threes, (26x3). These are then processed to provide the 26 outputs. The neurons in the hidden layer and the output layer use the ReLu activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01.

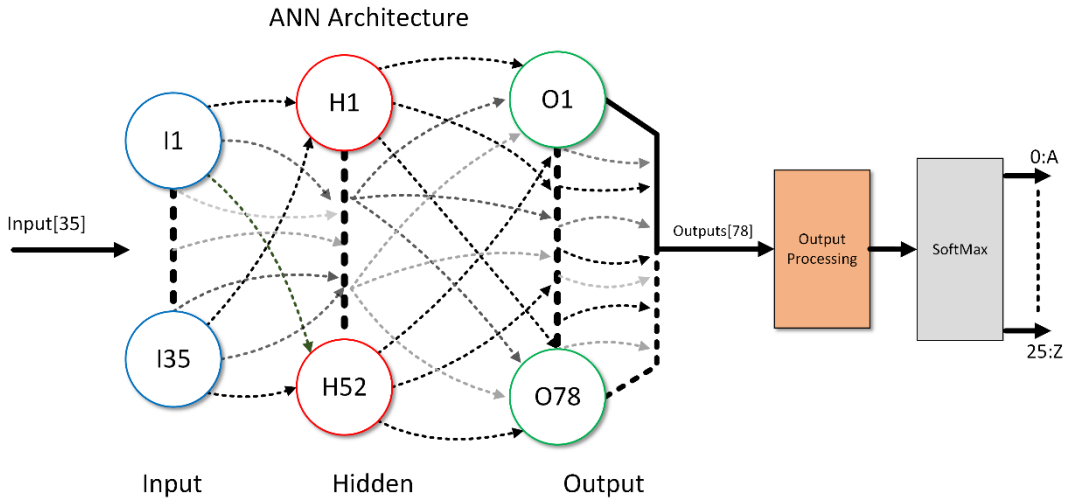

Fig. 5-5 ANN 35-52-78 architecture, 52 hidden layer neurons, 78 output neurons

The 78 outputs of the ANN are floating-point values and grouped into three outputs for each letter. The maximum of these three outputs (Fig. 5-6) is passed through the SoftMax function to provide an output probability for each of the 26 characters.

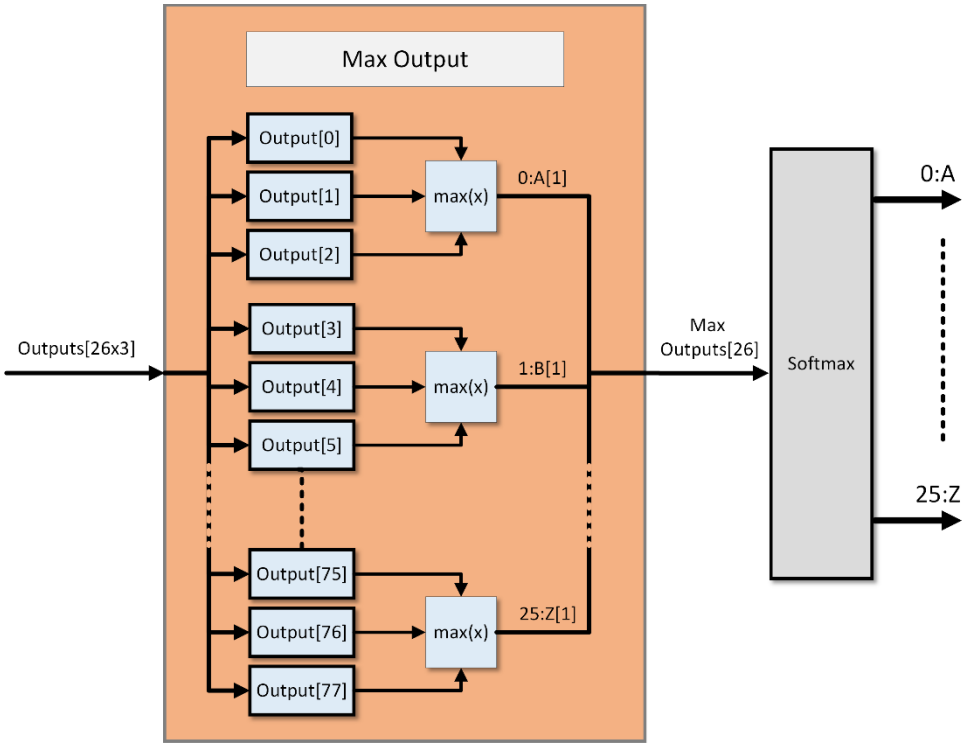

Fig. 5-6 Max method used by the ANN architecture to group the 26x3 outputs into one output for each character

Chromosome and Crossover

The ANNs chromosome is its weights and biases which are stored as a floating-point variable in a set of two-dimensional arrays. The hidden layer component of the chromosome is an array of size 52x36, and for the output layer an array of size 78x53; this gives a search space of $1.02 \cdot 10^{6924}$. The search space for the ANN can be calculated (using equation 4.34) as follows.

$$S = (no.of\ values^{H_{weights+bias}+O_{weights+bias}})$$

$$no.of\ values = 1 + \frac{max - min}{increment\ size} = 1 + \frac{1 - -1}{0.01} = 201$$

$$S = 201^{1872+4134}$$

$$S = 1.02 \cdot 10^{6924}$$

The same multipoint crossover operation used for the 2WD robotic controller evolution (4.2.4) is also employed for this problem. The child chromosomes are mutated with a 0.5% mutation rate.

5.1.4 Genetic Algorithm

The genetic algorithm used for evolving the VPLDs and ANN systems is very similar to the GA used in the robotic control problems, but with different GA parameters (Table 5-3). The same GA with a population of 100 individuals is applied to all the systems, D-VPLD with digital functions, F-VPLD with arithmetic functions and the ANN. As with the previous robotic control problem the population size of 100 is chosen to ensure a more diverse population particularly in the early stages of the evolution process. An increased population is possible which may further improve diversity; however, this will increase the computation time required to complete a generation. Vice versa a reduced population size may improve the computation time of each generation but can reduce the evolutionary rate. The GA process is run until the VPLDs, and ANN systems can successfully recognise each character. The reproduction process in the GA uses multi-point crossover with a 0.5% mutation rate. The mutation rate is reduced for this problem compared to the robotic control problems as the chromosomes of the evolved systems are much larger. If the same mutation rate is used it results in more FABs and neurons being mutated, which is more damaging to useful genes inherited from the parent individuals during the reproduction step.

Table 5-3 GA Parameters used for Character Recognition evolution

Parameter	Value
Population Size	100
Mutation Rate	0.5%
Selection	Two stage binary tournament
Crossover	Multipoint

5.1.5 Fitness Evaluation

The fitness evaluation of the ANN and VPLDs is based on the ability of each system to recognise each character correctly. For a set of 26 characters each system can have a fitness from -26 to 26 (-1 to +1 for each character) where 26 is the optimal fitness. For a given character A for example, if A is the maximum and correct value, it gets half the optimal fitness. As there can be multiple outputs with the same value, it gets the second half of the optimal fitness if the systems activation for the A character is greater than 1/26 (e.g. if all the outputs are the same value, then each would have a value of 1/26). Thus, for each output that is higher than or the same as the character A the fitness is reduced by 1/26. Equations (5.7) - (5.10) define the fitness evaluation used in the GA process.

$$Fitness = \sum_{i=1}^{26} f(x_i) + g(x_i) + h(x_i) \tag{5.1}$$

$$f(x_i) = \begin{cases} 0.5, max(x_i) = i \\ 0, max(x_i) \neq i \end{cases} \tag{5.2}$$

$$g(x_i) = \begin{cases} 0.5, x_i[i] > 1/26 \\ 0, x_i[i] \leq 1/26 \end{cases} \tag{5.3}$$

$$h(x_i) = 1/26 + \sum_{j}^{26} \begin{cases} -1/26, x_i[j] \geq x_i[i] \\ 0, x_i[j] < x_i[i] \end{cases} \tag{5.4}$$

5.1.6 Results

The D-VPLD, F-VPLD and ANN systems are evolved to recognise 26 character images (A-Z), an optimal solution is a system that can recognise each character. The GA experiments are repeated one hundred times and the average results of each of these experiments are used to assess the system performance. During each GA generation, the best fitness is recorded, and the time taken to reach the final solution is recorded, this information allows us to determine the evolutionary efficiency of each system, (how many generations and the computation time required to reach a solution).

Evolutionary Efficiency

The fitness vs generations plot (Fig. 5-7) shows that all three architectures have a similar rate of evolution until the final stages of the evolutionary process. However, the D-VPLD takes significantly more generations to get every character correct, as one letter is incorrect which is not easily seen in the graph.

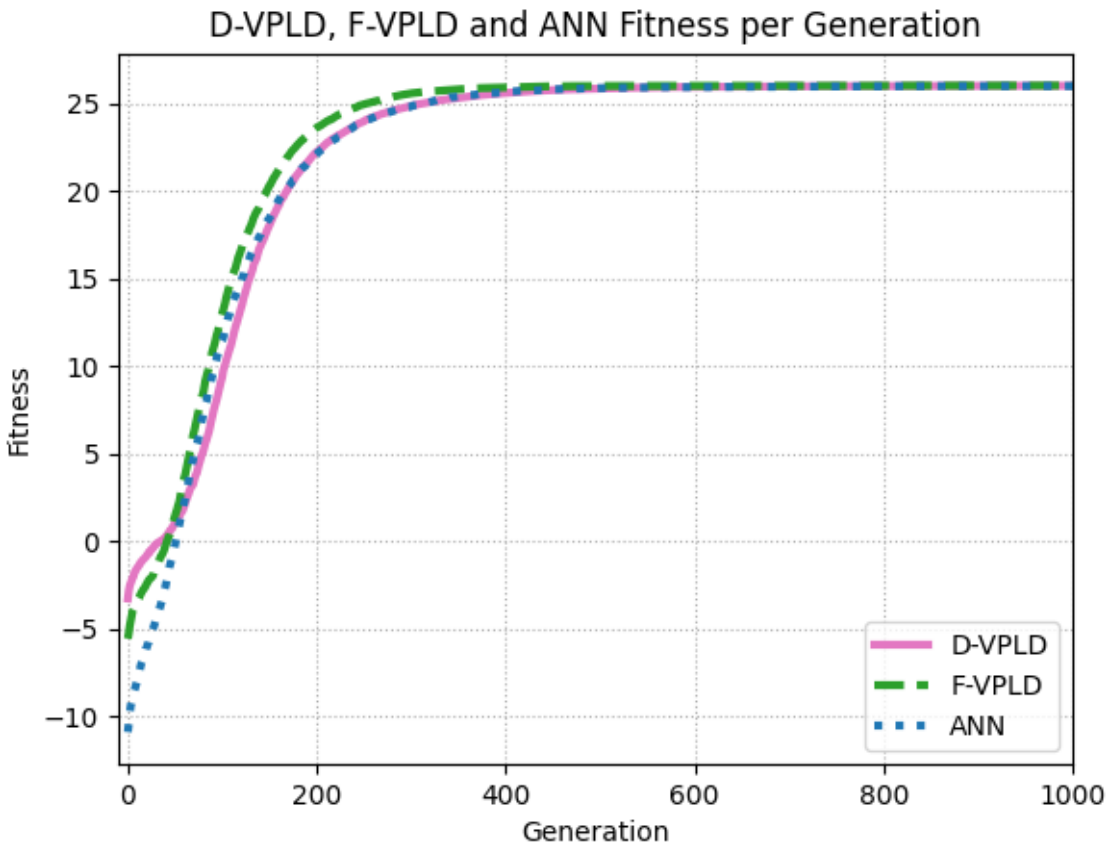

Fig. 5-7 Character Recognition Best Fitness per Generation

This can be more clearly seen in the table below (Table 5-4), where the F-VPLD and the ANN both have a comparable number of generations for a successful solution. However, the median number of generations to reach a solution for the D-VPLD is twice that of the F-VPLD and ANN. This may be due to the binary nature of the D-VPLD and the fact that it has a more limited number of possible output values for each character. Which would mean a more specific circuit would be required to achieve the maximum fitness.

Table 5-4 Character Recognition Number of Generations to Evolve

Architecture	Mean	Median	CV
D-VPLD	930	780	0.55
F-VPLD	417	392	0.34
ANN	387	355	0.35

The violin plot (Fig. 5-8) shows the evolution time of the three architectures, indicating that the F-VPLD and ANN take a similar time to evolve to a successful solution, whereas the distribution of the D-VPLD is worse. The evolution of the F-VPLD and ANN is more consistent than the D-VPLD, again suggesting there are fewer optimal solutions in the fitness landscape of the D-VPLD.

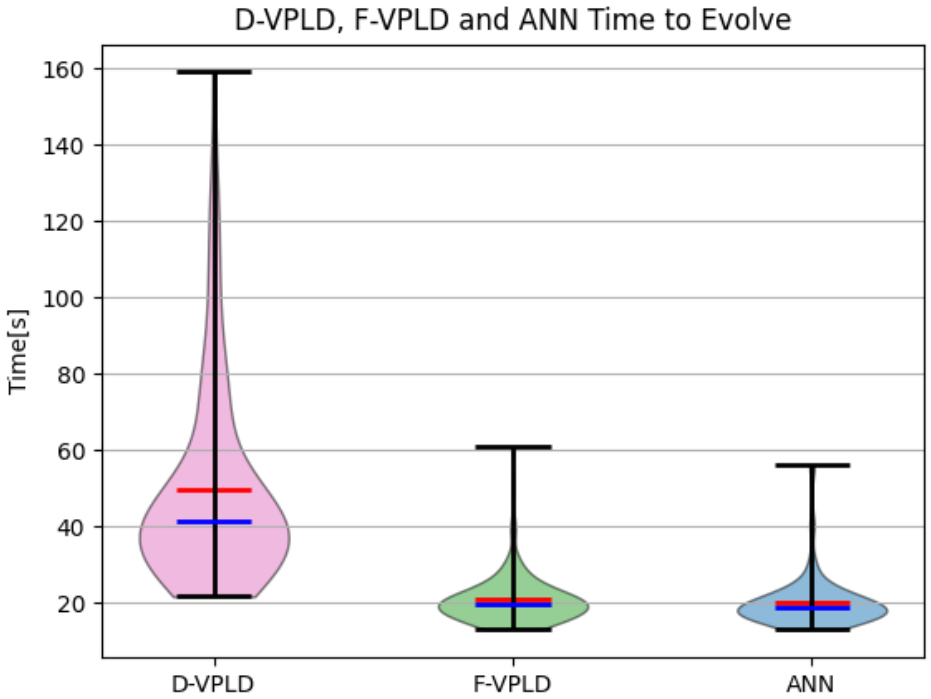

Fig. 5-8 A Violin plot of the time to evolve a solution. Shows distribution of results, Maximum/Minimum fitness, median(blue line) and mean(red line)

The time taken to reach a solution (Table 5-5) is relatively quick with the median time for both the F-VPLD and ANN under 20 seconds. The D-VPLD had an evolutionary time of approximately 40 seconds, twice that of the F-VPLD and ANN.

Table 5-5 Character Recognition Time to Evolve (seconds)

Architecture	Mean	Median	CV
D-VPLD	49.5	41.5	0.54
F-VPLD	21	19.8	0.33
ANN	20.1	18.7	0.34

5.1.7 Conclusions

In this section VPLDs using both: 1) binary numbers and digital logic used in FPGA based EHW research (D-VPLD); and 2) floating point numbers and simple mathematical arithmetic (F-VPLD); are compared to an ANN for a character recognition task. The three architectures are evolved using a GA to recognise the characters A-Z.

It was found that the F-VPLD and ANN required a similar number of generations taking 417 and 387 generations on average to evolve respectively. The time to evolve both the F-VPLD and ANN is also comparable with both having a median time to evolve under 20 seconds. The D-VPLD on the other hand takes over twice as many generations to evolve than the F-VPLD and ANN, however it still found solutions with a median time of approximately 40 seconds. The longer time to find a solution can be contributed to the nature of the binary logic used by the D-VPLD. By using binary numbers and digital logic, it has a more limited number of possible output values for each character. Which would mean a more specific circuit would be required to achieve the maximum fitness. This was further highlighted by the violin plots showing the F-VPLD and ANN being more consistent with the time to evolve compared to the D-VPLD, suggesting fewer optimal solutions in the fitness landscape of the D-VPLD. The D-VPLD also requires extra computation time dealing with pseudo bits, opposed to the F-VPLD and ANN which use data types more native to the computer they are running on.

The evolved solutions could recognise all characters A-Z, in past publications investigating evolvable hardware architectures, only the characters A-P were investigated. The binary character recognition problem has been a useful first step in investigating the VPLD for use in pattern recognition problems and has shown the VPLD is usable in this domain. The knowledge gained from this problem is applied to the more complicated melanoma detection task in the following section, which will test whether the VPLD is a true viable alternative to ANNs for pattern recognition.

5.2. Melanoma Classification

This section describes the melanoma classification problem used to evaluate the system architectures of the D-VPLD, F-VPLD and the ANN used to classify the data. As with the previous problems, all three classifiers are evolved using the same GA. The aim of the GA is to find optimal configurations such that the D-VPLD, F-VPLD and ANN systems can classify benign (non-cancerous) and malignant (cancerous) skin lesions. The results of each architectures evolution are analysed, and a comparison is made between the D-VPLD, F-VPLD and ANN.

5.2.1 Melanoma Classification Method

The method used for the classification of melanomas (Fig. 5-9), has 4 main stages; 1) the images undergo preprocessing for illumination correction, hair removal and noise reduction; 2) a two-step segmentation using k-means clustering followed by the Otsu thresholding method; 3) feature extraction, where a set of 16 features is extracted using the ABCD method [84] and the Grey Level Co-Occurrence Matrix [53]; and 4) classification where the D-VPLD, F-VPLD and ANN are used to identify from the extracted features whether the skin lesion is benign or malignant.

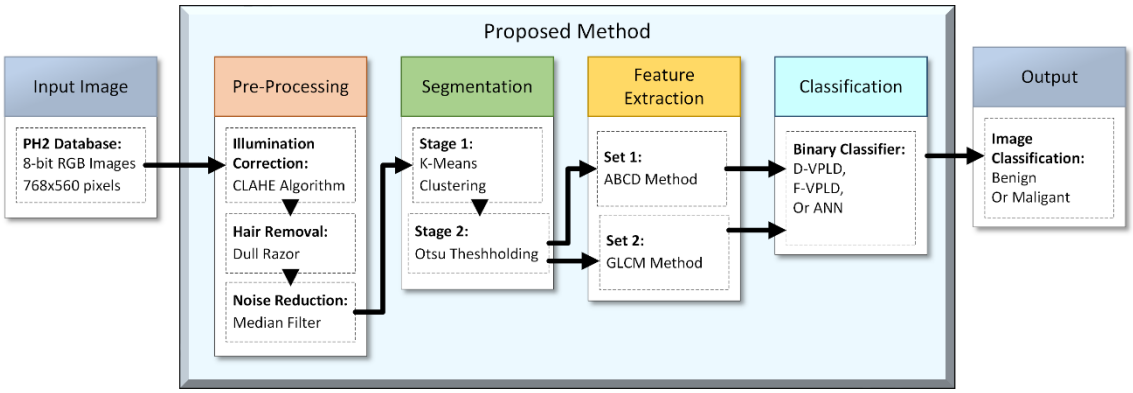

Fig. 5-9 Proposed Method for melanoma detection from images of benign and malignant skin lesions a part of the PH2 database

Dataset

The dataset used is the PH2 Database[21], which is a set of 200 768x560 pixel dermoscopic images, with 80 common nevi (non-cancerous), 80 atypical nevi (non-cancerous), and 40 melanomas (cancerous). To balance the dataset, augmentation is performed by applying random transformations on the 40 melanoma images to have a total of 160 malignant images, which is balanced by the 160 benign images. Examples of the images are shown below in Fig. 5-10.

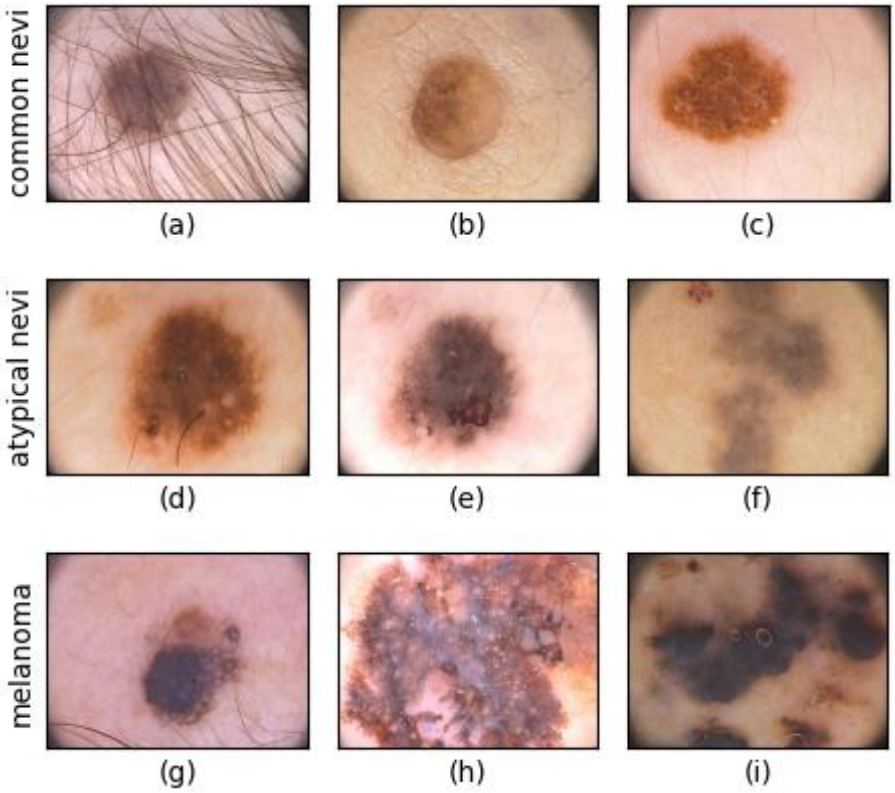

Fig. 5-10 Examples from PH2 Database. Top(a-b-c), common nevus(common benign lesion); middle(d-e-f), atypical nevus(atypical benign lesion, has common properties to malignant lesions but are non-cancerous); bottom(g-h-i), melanoma(malignant skin lesions)

For training, the dataset is split 75% for training and 25% for testing. The dataset is split randomly but still class balanced, so the split of malignant and benign lesions is 50:50 for both the training and test sets.

Image pre-processing

First the raw images from the dataset undergo illumination correction, using contrast limited adaptive histogram equalization (CLAHE). The CLAHE algorithm is an adaptive histogram equalization method used to improve the contrast of the images. After illumination correction, the next step is to perform hair removal using the dull razor algorithm (Fig. 5-11), the algorithm works best at removing dark hairs from the images. The algorithm can be implemented in four steps; 1) converting the original image to greyscale; 2) perform the black hat morphological transform on the greyscale image, which will enhance dark objects i.e. hairs in the image; 3) apply binary thresholding to obtain a mask for the hairs in the image; and 4) use the mask as a reference for inpainting on the original image to replace the regions of hair with new image data that match neighbouring pixels.

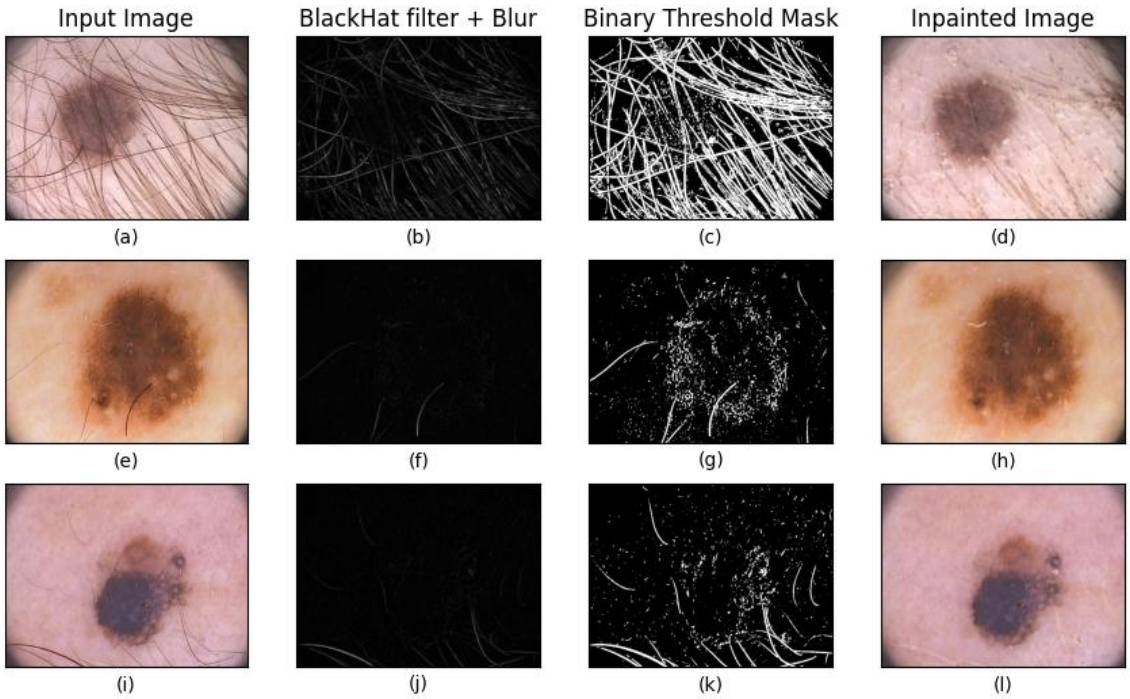

Fig. 5-11 Dull Razor Method implemented on three examples, from left to right; 1) input image that has undergone illumination correction; 2) result of Blackhat filter and gaussian blur applied to greyscale image of the lesion; 3) binary threshold applied to generate the final mask; and 4) the output image after inpainting to remove hairs

Prior to segmentation, a median filter is applied to attempt to remove any fine hairs not dealt with by the dull razor algorithm, and to reduce any additional unwanted artefacts in the image. The median blur improves the performance of the segmentation algorithm.

Segmentation

Segmentation is very important to the correct classification of the images; poor segmentation will reduce model accuracy as the skin lesion data has low variance between benign and the malignant (melanoma) lesions. Any errors in segmentation will result in the data not correctly representing the class of the image.

A two-stage segmentation process is used, the first step uses k-means clustering with 10 clusters, maximum iterations set at 100 and a required accuracy of 85%. K-means groups similar pixels into clusters based on a mean pixel value, ideally separating skin-lesion pixels from background skin pixels. The second step uses the Otsu thresholding method applied to the k-means output converted to greyscale. Otsu is an automatic image thresholding method that is used to find the optimal threshold level that separates foreground (skin lesion) and background (normal skin) pixels. The optimal threshold level minimizes the intra class intensity variance. The output of the Otsu thresholding is the mask used to segment the image (Fig. 5-12).

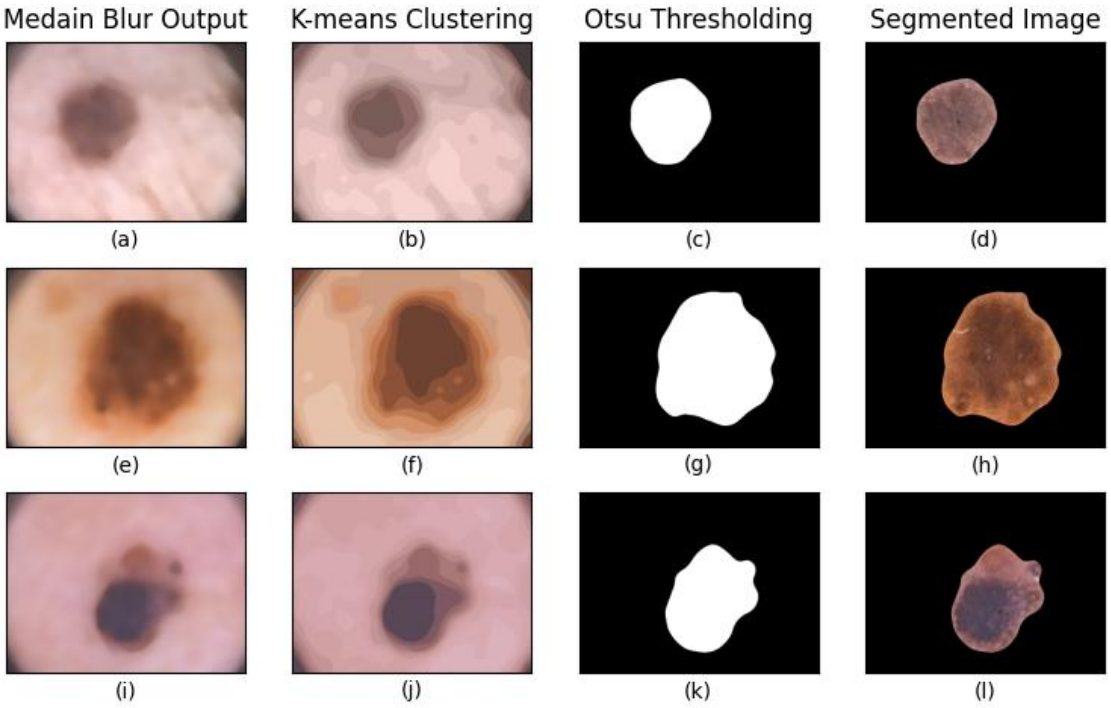

Fig. 5-12 Segmentation examples. 1) (a-e-i) image with median blur applied; 2) (b-f-j) first stage of the segmentation using k-means clustering; 3) (c-g-k) second stage of the segmentation using the Otsu Thresholding method; and 4) (d-h-l) the final segmented image ready for feature extraction

Feature Extraction ABCD

The first set of features extracted from the final segmented image is a set of ABCD features[84, 85], (Asymmetry, Border, Colour and Diameter). Asymmetry is used as benign lesions are generally more symmetric in shape compared to malignant lesions. To calculate the asymmetry features the segmented skin lesion must be orientated and centred in the image frame (Fig. 5-13).

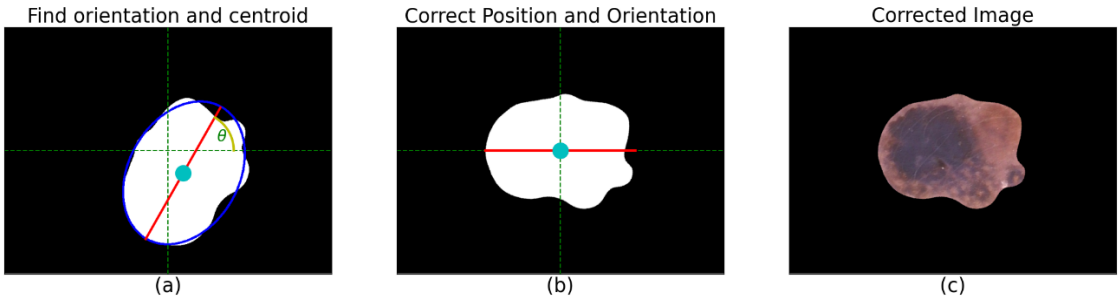

Fig. 5-13 Orientation and position correction of skin lesion, using OpenCV the best fitted ellipse is used to find the orientation and the moments of the skin lesion contour are used to find its centroid. Using this information the skin lesion is reoriented to the centre of the image

Two asymmetry features are calculated, one for symmetry along the x-axis (A_1) and one for along the y-axis (A_2). To calculate each value the skin lesion is flipped along the respective axis and the difference between the original and flipped images are calculated, this gives us a value (ΔL_x or ΔL_y) that represents the area of the skin lesion that is not symmetrical, this value is then divided by the total area of the skin lesion. Equations (5.5) & (5.6) are used to calculate A_1 and A_2.

$$A_1 = \frac{Area\ of\ \Delta L_x}{Area\ of\ L}, \qquad \text{where } \Delta L_x = L \oplus L_x \tag{5.5}$$

$$A_2 = \frac{Area\ of\ \Delta L_y}{Area\ of\ L}, \qquad \text{where } \Delta L_y = L \oplus L_y \tag{5.6}$$

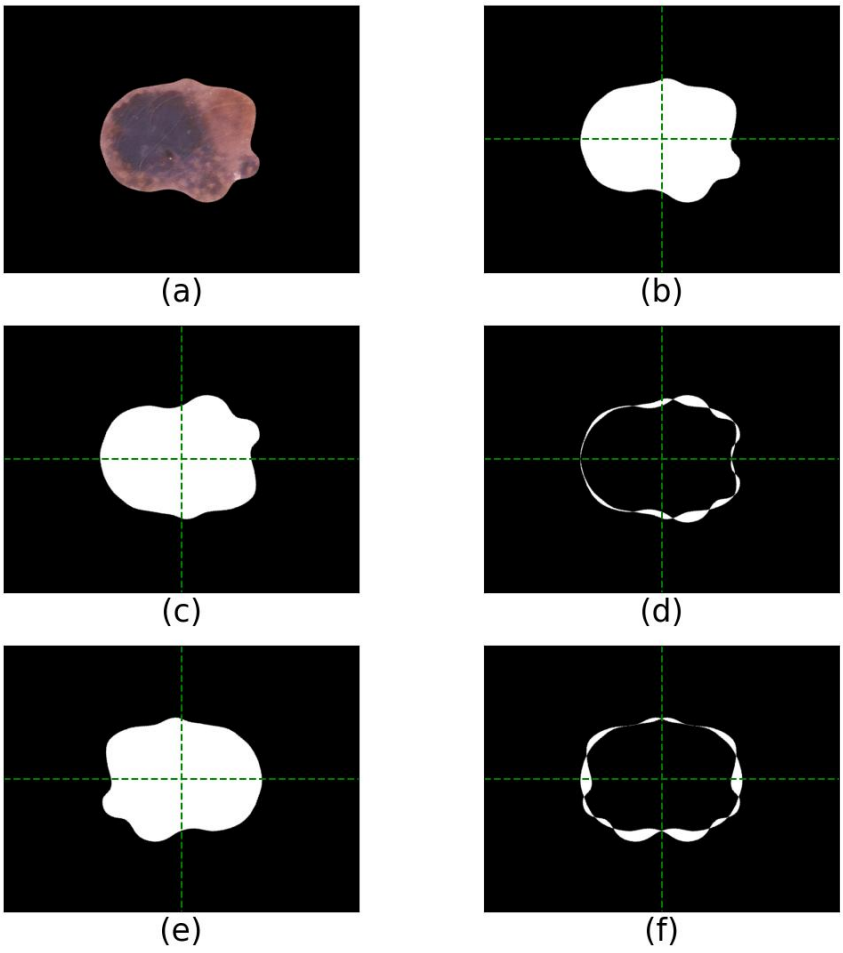

Fig. 5-14 Asymmetry feature calculation visualized. a) original lesion image (L); b) the binary mask of lesion (L), c) lesion (L) flipped along x-axis to get L_x, d) ΔL_x the difference between the original image and L_x, e) lesion (L) flipped along y-axis to get L_y, f) ΔL_y the difference between the original image and L_y.

The border of malignant skin lesions normally has abnormalities such as jagged or blurry(undefined) edges. For the border features, the length of the perimeter and the area of the skin lesion is found and used to calculate three features. B_1 is the ratio of the area (A) to the perimeter (P). B_2 is a compactness index, lesions that are closer to a perfect circle will have a B_2 value of 1, skin lesions with border abnormalities will have a B_2 value that approaches closer to zero the more irregular the border is. B_3 is the product of the area and perimeter of the skin lesion.

$$B_1 = \frac{A}{P} \tag{5.7}$$

$$B_2 = \frac{4\pi A}{P^2} \tag{5.8}$$

$$B_3 = AP \tag{5.9}$$

The final shape feature is the diameter of the lesion, malignant lesions tend to be larger than benign lesions (greater that 6mm in general, however it is possible to have melanomas smaller than 6mm but it is uncommon [86]). Two diameter feature values are calculated; D_1, the average diameter of the lesion; and D_2, the difference between the principal axis lengths of the best fitted ellipse (major axis Q, minor axis q).

$$D_1 = \frac{D_\alpha + D_\beta}{2}, \text{ where } D_\alpha = \frac{\sqrt{4A}}{\pi} \text{ and } D_\alpha = \frac{Q+q}{2} \tag{5.10}$$

$$D_2 = Q - q \tag{5.11}$$

The last feature from the ABCD method, is the colour features, malignant lesions tend to have multiple colours present, whereas benign lesions tend to have only one or two colours present. The colours checked for are, those identified from the PH2 database[21], white, red, light-brown, dark-brown, blue-grey and black (the specific 8-bit RGB threshold values are shown in Table 5-6).

To determine the colour score C_1, each colour is checked using a threshold. If a colour is present in 5% of the skin lesion area, then the colour score is incremented by one, else a value is added based on the ratio of the pixels within the colour threshold, equation (5.12). The skin lesion colour score C_1 ranges from 1 to 6. Three other colour features are also used, C_2 average red value, C_3 average green value, C_4 average blue value.

$$C_1 = \sum_{i=1}^{6} \begin{cases} \frac{max(\boldsymbol{P}_i)}{5}, & \text{for } max(\boldsymbol{P}_i) < 5 \\ 1\,, & \text{for } max(\boldsymbol{P}_i) \geq 5 \end{cases} \tag{5.12}$$

where, i is the colour index, and $\boldsymbol{P_i}$ is an array of the percentage of pixels within the colour threshold for each segment.

$$C_2 = \frac{R_{avg}}{R_{avg} + G_{avg} + B_{avg}} \tag{5.13}$$

$$C_3 = \frac{G_{avg}}{R_{avg} + G_{avg} + B_{avg}} \tag{5.14}$$

$$C_4 = \frac{B_{avg}}{R_{avg} + G_{avg} + B_{avg}} \tag{5.15}$$

Table 5-6 Thresholds for each colour channel RGB, used to determine colour score

Colour	Red Value	Green Value	Blue Value
White	$R \geq 204$	$G \geq 204$	$B \geq 204$
Red	$R \geq 112$	$60 \geq G \geq 30$	$60 \geq B \geq 30$
Light brown	$234 \geq R \geq 72$	$221 \geq G \geq 48$	$202 \geq B \geq 48$
Dark brown	$143 \geq R \geq 61$	$100 \geq G \geq 0$	$100 \geq B \geq 0$
Blue grey	$150 \geq R \geq 0$	$150 \geq G \geq 0$	$255 \geq B \geq 112$
Black	$62 \geq R \geq 2$	$62 \geq G \geq 2$	$62 \geq B \geq 2$

All of the ABCD features are normalized in their respective categories to have values that range from 0 to 1, ready to be used for classification.

Feature Extraction GLCM

The Grey Level Co-Occurrence Matrix originally introduced by Haralick et. Al [53], is a method used for texture analysis of images. The GLCM is generated by calculating the frequency of occurrence for pixel pairs with specific grey levels and spatial relationships with a specific pixel offset. Three parameters are used in the calculation; 1) pixel offset, distance between pixel pairs; 2) angle between pixels (adjacency direction), typically 0, 45, 90 and 135; and 3) the number of grey levels (N) in the images being analysed. The parameters used in this research are shown in Table 5-7. Once the GLCM is found, a set of features commonly known as Haralick features can be obtained from calculations using the GLCM, in this study the dissimilarity, correlation, contrast, energy and homogeneity features are used.

Table 5-7 GLCM Parameters

Parameter	Value
Pixel pair distance offset	3
adjacency direction	0
Grey levels	256

Dissimilarity is a measure of the average difference between neighbouring pixel intensities, a high value would indicate a large diversity in the lesion texture (5.16).

$$DIS = \sum_{i=1}^{N}\left\{\sum_{j=1}^{N}\{|i-j| \cdot p_{i,j}\}\right\} \tag{5.16}$$

Correlation measures the linear dependency of neighbouring pixels, a high correlation value would mean the texture is more predictable or consistent, therefore the number of abrupt changes in texture is low (5.17).

$$COR = \frac{\sum_{i=1}^{N}\{\sum_{j=1}^{N}\{i \cdot j \cdot p_{i,j}\}\} - \mu_x\mu_y}{\sigma_x\sigma_y} \tag{5.17}$$

where, $\mu_x = \sum_{i=1}^{N} i \cdot \{\sum_{j=1}^{N} p_{i,j}\}$, $\mu_y = \sum_{j=1}^{N} j \cdot \{\sum_{i=1}^{N} p_{i,j}\}$,

${\sigma_x}^2 = \sum_{i=1}^{N}(i-\mu_x)^2 \cdot \{\sum_{j=1}^{N} p_{i,j}\}$, ${\sigma_y}^2 = \sum_{j=1}^{N}(j-\mu_y)^2 \cdot \{\sum_{i=1}^{N} p_{i,j}\}$.

Contrast is a measure of local grey level variation of the lesion image, smooth images have a low contrast value, rough images have a high contrast value (5.18).

$$CON = \sum_{i=1}^{N}\left\{\sum_{j=1}^{N}\{(i-j)^2 \cdot p_{i,j}\}\right\} \tag{5.18}$$

Energy is the square root of the Angular Second Moment; it is a measure of uniformity of the lesion image. If the lesion is completely uniform it will have a high energy value (5.19).

$$E = \sqrt{\sum_{i=1}^{N}\left\{\sum_{j=1}^{N}\{(p_{i,j})^2\}\right\}} \tag{5.19}$$

Homogeneity is used to measure how close the elements of GLCM are to the diagonal. If the elements are concentrated along the diagonal, the homogeneity value will be high. As with the energy value, homogeneity is another feature used to assess the uniformity of the lesion.

$$H = \sum_{i=1}^{N} \left\{ \sum_{j=1}^{N} \left\{ \frac{p_{i,j}}{1 + (i-j)^2} \right\} \right\} \tag{5.20}$$

All of the GLCM features are normalized in their respective categories to have values that range from 0 to 1, ready to be used for classification.

Classification

The calculated Haralick features from the GLCM are combined with the ABCD features to form the complete set of 16 features. The inputs are normalised to a floating-point value ranging from 0 to 1 and are used as the input into the D-VPLD, F-VPLD and ANN architectures for classification. These three architectures are used to generate a single output value for the binary classification of the skin lesion (Fig. 5-15). The skin lesions can be either malignant (cancerous melanoma) or benign (non-cancerous common/atypical nevus). For an output value greater than 0.5 the lesion is classified as malignant lesion, if the output is less than or equal to 0.5 the lesion is classified as benign.

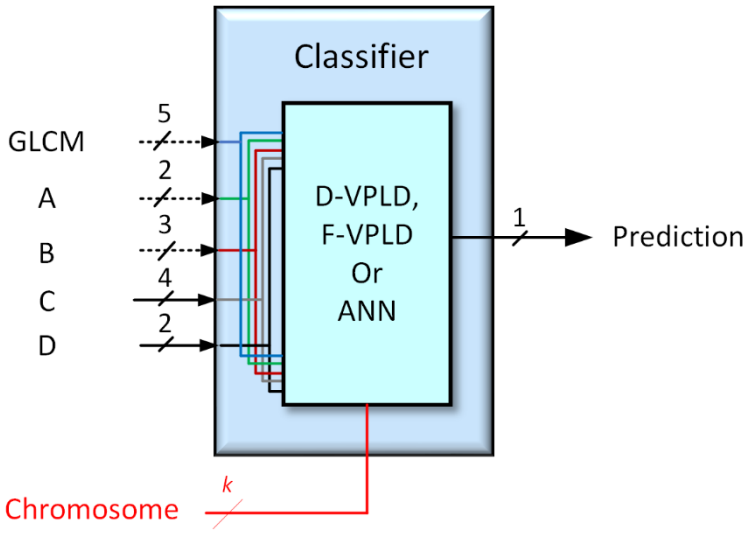

Fig. 5-15 Classifier architecture used for the melanoma classification problem

5.2.2 D-VPLD Classifier Architecture

A variety of architectures are investigated for the binary classification of melanomas, with the most optimal D-VPLD being a two-layer architecture (Fig. 5-16) with 16 FABs in the first layer and 9 FABs in the second layer (results of the other architecture evaluated can be found in section 6.4). The FABs in both layers have 4 multiplexers and a FE with 21 functions (Table 5-9), this FE is the same as the one that is most optimal for the 2WD problem. As the D-VPLD uses pseudo binary values, the inputs of the 16 features are quantized into 16 8-bit values using a threshold encoding technique similar to "one-hot" encoding. The 9 final layer FAB outputs are grouped into three 3-bit outputs and summed which results in 3 values ranging from 0-3 (Fig. 5-17). The mean of these three values is calculated for the final output of the D-VPLD, resulting in an output ranging from 0-3.

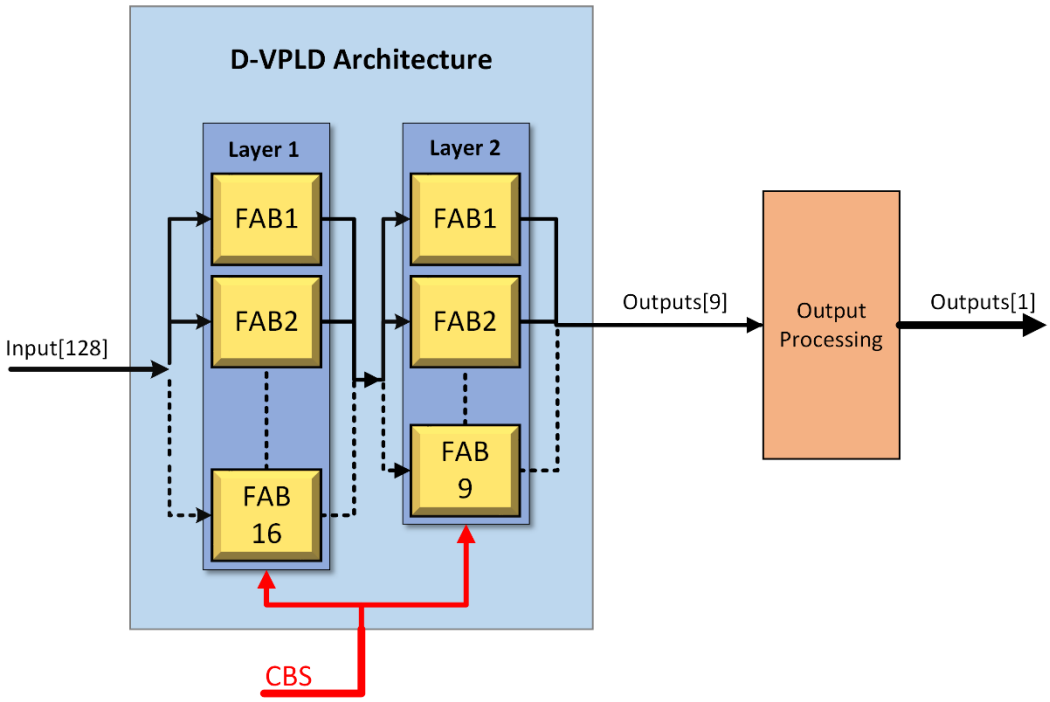

Fig. 5-16 D-VPLD Architecture for binary classification of melanomas. Two-layer architecture with 16 FABs in the first layer and 9 FABs in the second layer

Table 5-8 D-VPLD Input encoding technique

Value	8-bit Encoding
$x = 0$	0000 0000
$0.125 \cdot Sensor_{max} \geq x > 0$	0000 0001
$0.25 \cdot Sensor_{max} \geq x > 0.125 \cdot Sensor_{max}$	0000 0011
$0.375 \cdot Sensor_{max} \geq x > 0.25 \cdot Sensor_{max}$	0000 0111
$0.50 \cdot Sensor_{max} \geq x > 0.375 \cdot Sensor_{max}$	0000 1111
$0.625 \cdot Sensor_{max} \geq x > 0.50 \cdot Sensor_{max}$	0001 1111
$0.75 \cdot Sensor_{max} \geq x > 0.625 \cdot Sensor_{max}$	0011 1111
$0.875 \cdot Sensor_{max} \geq x > 0.75 \cdot Sensor_{max}$	0111 1111
$1.00 \cdot Sensor_{max} \geq x > 0.875 \cdot Sensor_{max}$	1111 1111

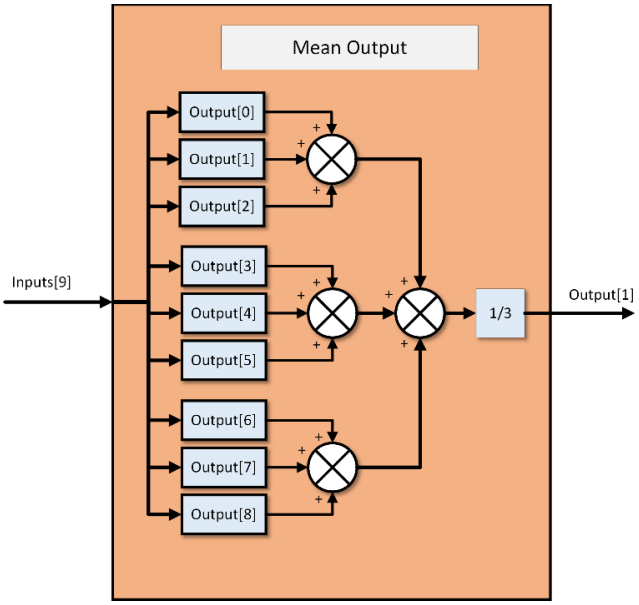

Fig. 5-17 Mean method used by the D-VPLD architecture to group the 3x3 outputs into one output used to determine the classification of a skin lesion

Table 5-9 D-VPLD Melanoma Classification FE LUT

No.	Logic Function	No.	Logic Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

Chromosome and Crossover

The chromosome of the D-VPLD in the melanoma classification problem uses the same format as seen previously, a set of two-dimensional arrays of integers. The first layer uses an array of size 16x5, and the second layer an array of size 9x5; this gives a search space of $1.84 \cdot 10^{211}$. The search space is calculated (using equation 4.11) as follows.

$$S = (L1_{inputs}{}^{L1_{MUXs}})(\,L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(\,L2_{functions}{}^{L2_{FEs}})$$

$$S = (128^{64})(\,21^{16})(16^{36})(\,21^{9})$$

$$S = 1.84 \cdot 10^{211}$$

For this problem, the multipoint crossover introduced in section 4.2.1 is used. The child chromosomes are mutated based on a 0.5% mutation rate.

5.2.3 F-VPLD Classifier Architecture

The F-VPLD uses a two-layer architecture with 16 FABs in the first layer, and 3 FABs in the second layer (Fig. 5-18). This two-layer architecture was found to be the most optimal for the F-VPLD, other architectures are evaluated and the results for which can be found in section 6.4. The FABs in both layers have 4 multiplexers and a FE with 21 functions, this FE is similar to the one that was used for the 2WD problem and the character recognition problem, instead of using a piece-wise linear function or the ReLu function, the sigmoid function is used (Table 5-10). The inputs are the 16 features extracted from the lesion image, no pre-processing is required on the inputs as they are normalised floating-point values ranging from 0-1, suitable for input to the F-VPLD. The mean of the 3 output values is calculated to generate the final output used for the classification of the skin lesion, the output ranges from 0-1.

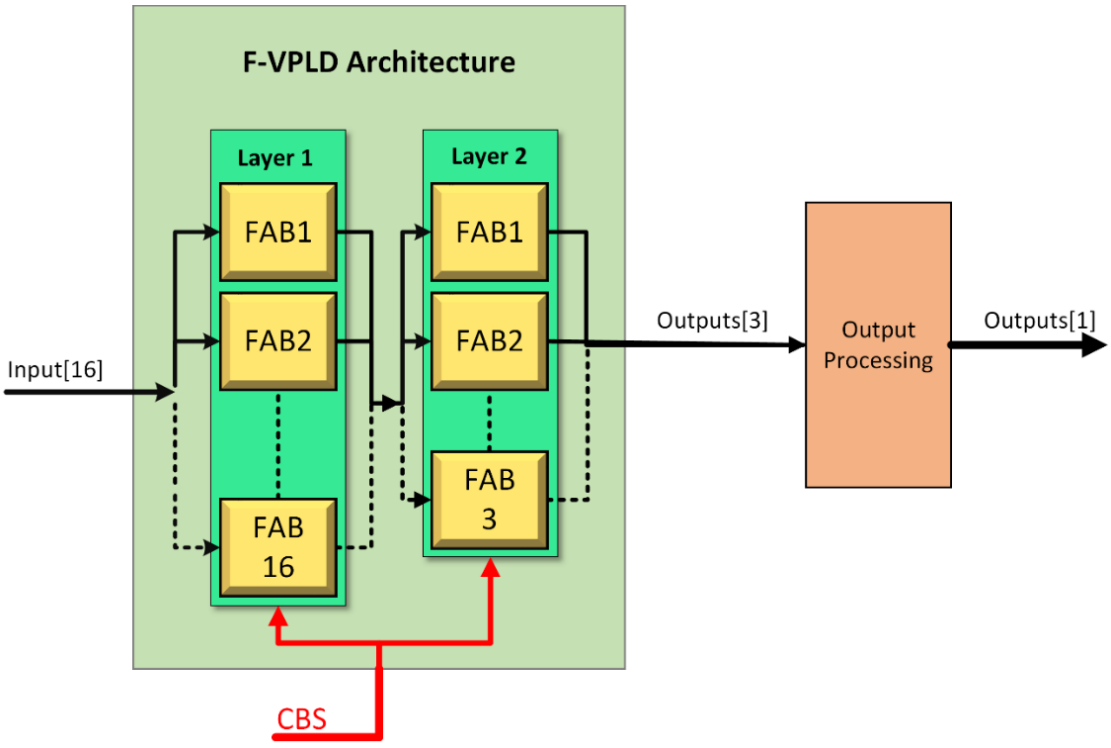

Fig. 5-18 F-VPLD Architecture for binary classification of melanomas. Two-layer architecture with 16 FABs in the first layer and 9 FABs in the second layer

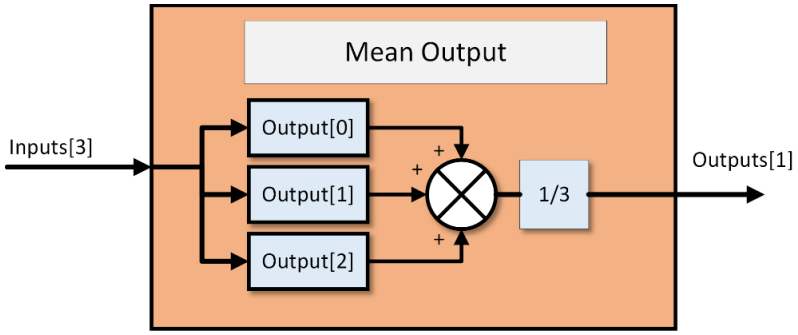

Fig. 5-19 Mean method used by the F-VPLD architecture to group the 3x3 outputs into one output used to determine the classification of a skin lesion

Table 5-10 Melanoma Classification F-VPLD FE LUT

No.	Function	No.	Function
0	*f*((A+B)*(C-D))	11	*f*((-A)+B+C+D)
1	*f*(A+B+C+D)	12	*f*(A+(-B)+C+D)
2	*f*(-(A+B+C+D))	13	*f*(A+B+(-C)+D)
3	*f*(A*B*C*D)	14	*f*(A+B+C+(-D))
4	*f*(-(A*B*C*D))	15	*f*((A*B)+(C*D))
5	*f*((A+B)*(C+D))	16	*f*((A*C)+(B*D))
6	*f*((A+C)*(B+D))	17	*f*((A*D)+(B*C))
7	*f*((A+D)*(B+C))	18	*f*((A*B)-(C*D))
8	*f*((A+B)-(C+D))	19	*f*((A*C)-(B*D))
9	*f*((A+C)-(B+D))	20	*f*((A*D)-(B*C))
10	*f*((A+D)-(B+C))		

Chromosome and Crossover

The F-VPLD chromosome uses the same format as the D-VPLD. A set of two-dimensional integer arrays. For the first layer this is an array of size 16x5, and for the second layer an array of size 3x5; this gives a search space of $4.31 \cdot 10^{116}$. The search space is calculated (using equation 4.11) as follows.

$$S = (L1_{inputs}{}^{L1_{MUXs}})(\,L1_{functions}{}^{L1_{FEs}})(L2_{inputs}{}^{L2_{MUXs}})(\,L2_{functions}{}^{L2_{FEs}})$$

$$S = (16^{64})(\,21^{16})(16^{12})(\,21^{3})$$

$$S = 4.31 \cdot 10^{116}$$

The F-VPLD uses the same multipoint crossover used on the D-VPLD chromosome, this is the same crossover method used for the 2WD robot control (section 4.2) and character recognition (section 5.1) problems. The child chromosomes are mutated based on the mutation rate of 0.5%.

5.2.4 ANN Classifier Architecture

As with the D-VPLD and F-VPLD various ANN architectures are investigated with the most optimal being a 16-16-3 fully connected two-layer feedforward ANN (Fig. 5-20). The results of the other architectures tested can be found in Appendix A. The 16 neurons in the hidden layer and the 3 neurons in the output layer all have a bias value and use the hyperbolic tangent activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01. The ANN like the F-VPLD requires no preprocessing of the input feature data. The outputs of the ANN use the same postprocessing method as the F-VPLD, where the mean of the 3 output values is calculated to generate the final output used for the classification of the skin lesion, the output ranges from 0-1.

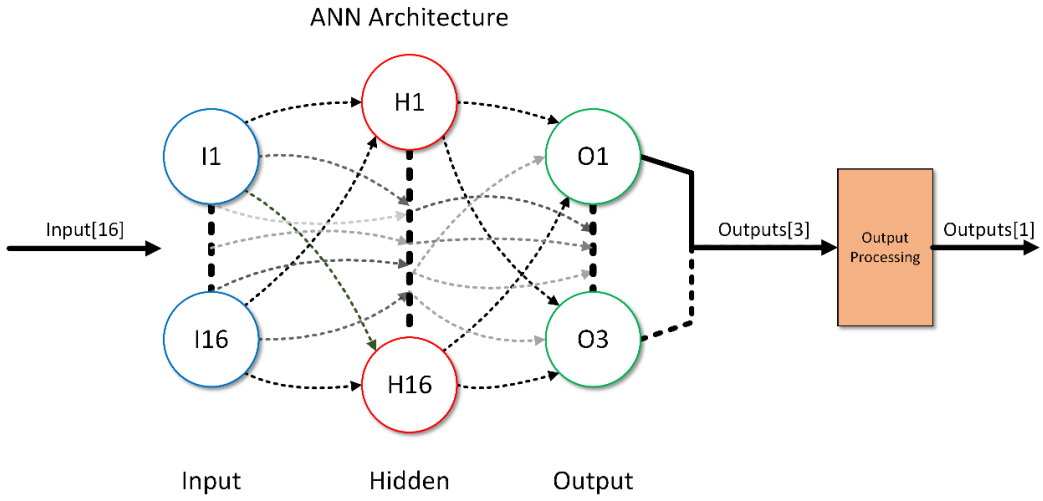

Fig. 5-20 ANN Architecture for binary classification of melanomas

Chromosome and Crossover

The chromosome of the ANN is a pair of two-dimensional arrays of floating-point values. For the first layer this is an array of size 16x17, and for the second layer an array of size 3x17; this gives a search space of $8.55 \cdot 10^{743}$. The search space for the ANN can be calculated (using equation 4.34) as follows.

$$S = (no.of\ values^{H_{weights+bias}+O_{weights+bias}})$$

$$no.of\ values = 1 + \frac{max - min}{increment\ size} = 1 + \frac{1 - -1}{0.01}$$

$$S = 201^{272+51}$$

$$S = 8.55 \cdot 10^{743}$$

The ANN uses a multipoint crossover, this is the same crossover method used for the 2WD robot control (section 4.2) and character recognition (section 5.1) problems. The child chromosomes are mutated based on the 0.5% mutation rate.

5.2.5 Genetic Algorithm

The genetic algorithm used for evolving the D-VPLD, F-VPLD and ANN systems for the melanoma classification is the same GA used in the previous robotic control problems and the character recognition problem. However, the GA parameters vary from those used in the previous problems. The GA process uses a population of 100, the population evolved for 1000 generations at which point the GA is stopped. At the end of the GA process the individual with the highest training accuracy is saved as the most optimal solution. The reproduction process in the GA uses the outlined multi-point crossover method with a 0.5% mutation rate. The selection process is the two-stage binary tournament method.

5.2.6 Fitness Evaluation

The fitness evaluation of the VPLDs and ANN is based on the ability of each system to classify the skin lesions correctly, the skin lesions can be either malignant (cancerous melanoma) or benign (non-cancerous common/atypical nevus). The fitness value is the training data accuracy of the individual, ranging from 0-1.

$$Fitness = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.21}$$

where TP and TN are the number of true positive and true negative classifications respectively, FP and FN are the number of false positive and false negative classifications respectively. The sum of TP, TN, FP and FN is equal to the number of samples in the training data (Table 5-11).

Table 5-11 Confusion matrix, for binary classification of skin lesions. Skin lesions are either malignant (cancerous melanoma) or benign (non-cancerous common/atypical nevus)

	Ground Truth: Malignant	**Ground Truth:** Benign
Predictions: *Malignant*	TP	FP
Predictions: *Benign*	FN	TN

5.2.7 Results

The D-VPLD, F-VPLD and ANN systems are evolved to classify skin lesions either as malignant or benign. An optimal solution is a evolved system that could correctly classify all the skin lesions (100% accuracy). To compare the three architectures, each is evolved one hundred times to assess their performance. For each evolution attempt the best fitness (training accuracy) per generation and test accuracy of the best individual per generation. The test accuracy is recorded for validation of the evolutionary process. Along with the accuracy data the number of generations and time required to evolve are recorded. The VPLDs and ANN are compared using: a) evolutionary efficiency, (how many generations and the computation time required to reach a solution); and b) the classifier performance (classification accuracy).

Evolutionary Efficiency

The training accuracy (fitness) per generation plot (Fig. 5-21) shows that the VPLDs have a slightly lower initial rate of evolution than the ANN, however the VPLDs converge to an optimal solution faster. The D-VPLD and ANN start on average at a high level of accuracy, but within a few generations the F-VPLD catches up. At the end of the 1000 generations, the ANN reaches the highest training accuracy on average, however the difference between the three architectures is less than 1%.

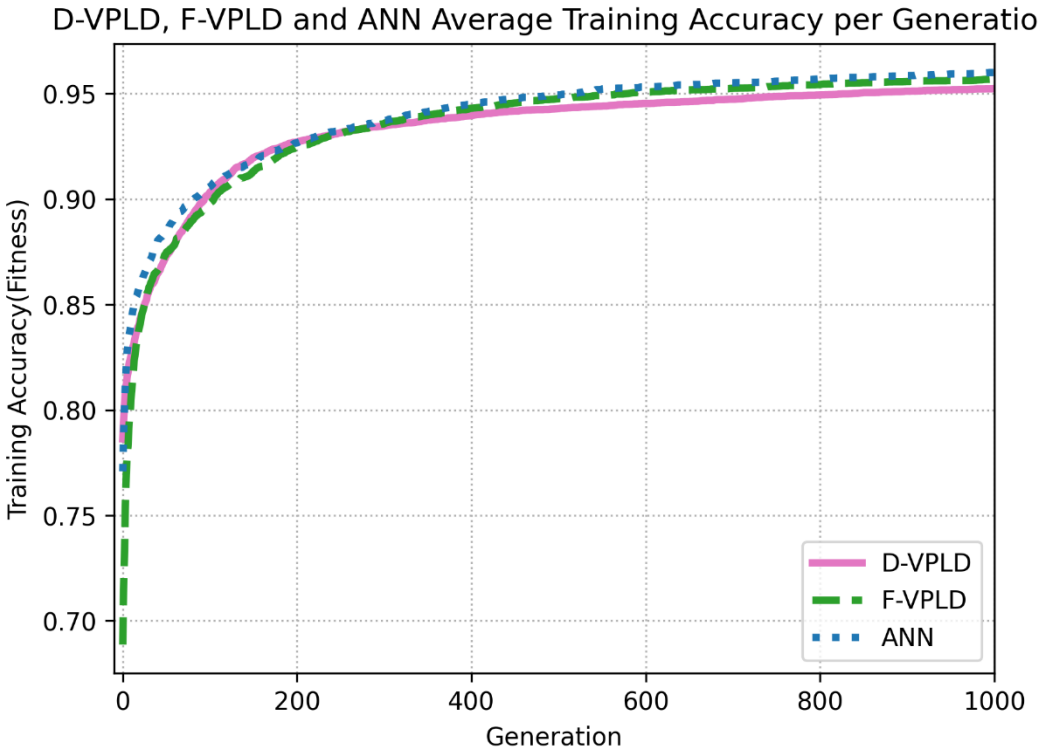

Fig. 5-21 Average training accuracy (fitness) per generation. Results for the most optimal D-VPLD, F-VPLD and ANN

The three architectures require a comparable number of generations to reach an optimal solution (Table 5-12). The F-VPLD requires the least generations at approximately 707 on average; compared to the F-VPLD, the ANN requires approximately 7% more generations, and the D-VPLD requires approximately 4% more generations to evolve. The F-VPLD and ANN take a very similar time to evolve, both reaching a high level of accuracy in a short amount of time, approximately 18.5 seconds on average. The D-VPLD requires the most time, approximately 3 times the computation time of the F-VPLD and ANN (Table 5-13). As seen in previous sections, the D-VPLD requires extra computation time dealing with pseudo bits as opposed to the F-VPLD and ANN which use data types more native to the computer they are running on. The D-VPLD has to convert the data into binary strings for input and convert the output from binary to the data format required for classification.

Table 5-12 Melanoma Classification Number of Generations to Evolve

Sys	Mean	Median	CV
D-VPLD	738.11	781.5	0.27
F-VPLD	707.44	736	0.27
ANN	757.89	796.5	0.24

Table 5-13 Melanoma Classification Time to Evolve

Sys	Mean[s]	Median[s]	CV
D-VPLD	59.9	63.39	0.26
F-VPLD	18.57	19.26	0.27
ANN	18.51	19.28	0.24

The data distribution of the three architectures is shown in Fig. 5-22, both the F-VPLD and ANN have a similar range of results with approximately 20 seconds between the minimum and maximum time to evolve, the D-VPLD on the other hand has approximately 65 seconds between the minimum and maximum time to evolve.

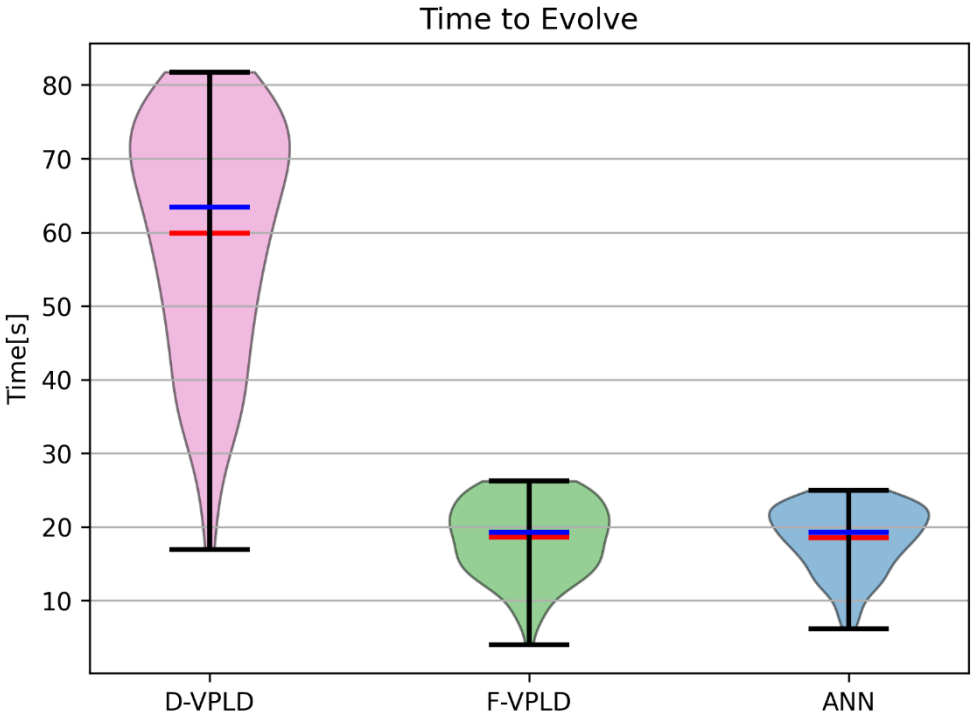

Fig. 5-22 A violin plot comparison of the time to evolve a solution. Shows distribution of results, Maximum/Minimum fitness, median (blue line) and mean (red line)

Classifier Performance

The D-VPLD, F-VPLD and ANN performance is based on the accuracy levels each achieved. As is seen in Fig. 5-21 all three reached a high training accuaracy, the test accuracy is also recorded along with the training accuracy as a means to validate the evolution process. It can be seen that the rate of improvement in test accuracy is comparable for each of the three architectures, the F-VPLD has a slightly higher rate of improvement in accuracy (Fig. 5-23). The F-VPLD finishes with the highest test accuracy on average, but is only marginaly better than the ANN and D-VPLD; the F-VPLD achieves approximately 1% better accuracy than the D-VPLD, and 0.1% better than the ANN.

Table 5-14 Training Accuracy (Fitness) for Melanoma Classification problem

Sys	Mean	Median	CV
D-VPLD	95.25	95.42	0.01
F-VPLD	95.7	95.83	0.01
ANN	96	95.83	0.01

Table 5-15 Test Accuracy for Melanoma Classification problem

Sys	Mean	Median	CV
D-VPLD	94.44	95	0.02
F-VPLD	95.46	96.25	0.01
ANN	95.31	95	0.02

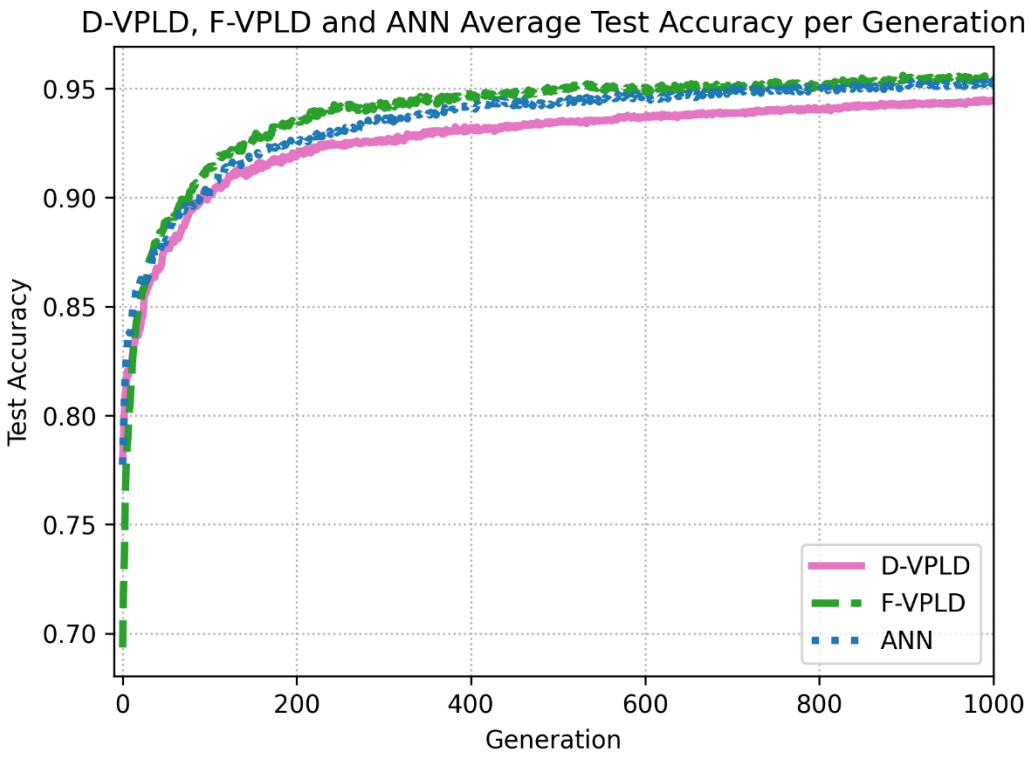

Fig. 5-23 Average test accuracy per generation. Results for the most optimal D-VPLD, F-VPLD and ANN

Across the 100 results for each of the three architectures, they have a consistent accuracy, in particular for the training accuracy (Fig. 5-24) with a 4-5% variance across all the results for the F-VPLD and ANN, and a 7.5% difference between the minimum and maximum training accuracy for the D-VPLD. For the test accuracy (Fig. 5-25), the F-VPLD has the smallest distribution ranging from 92.5% to 97.5% accuracy. The D-VPLD and ANN have a larger distribution ranging from 85% to 97.5% and 91.25% to 97.5% respectively.

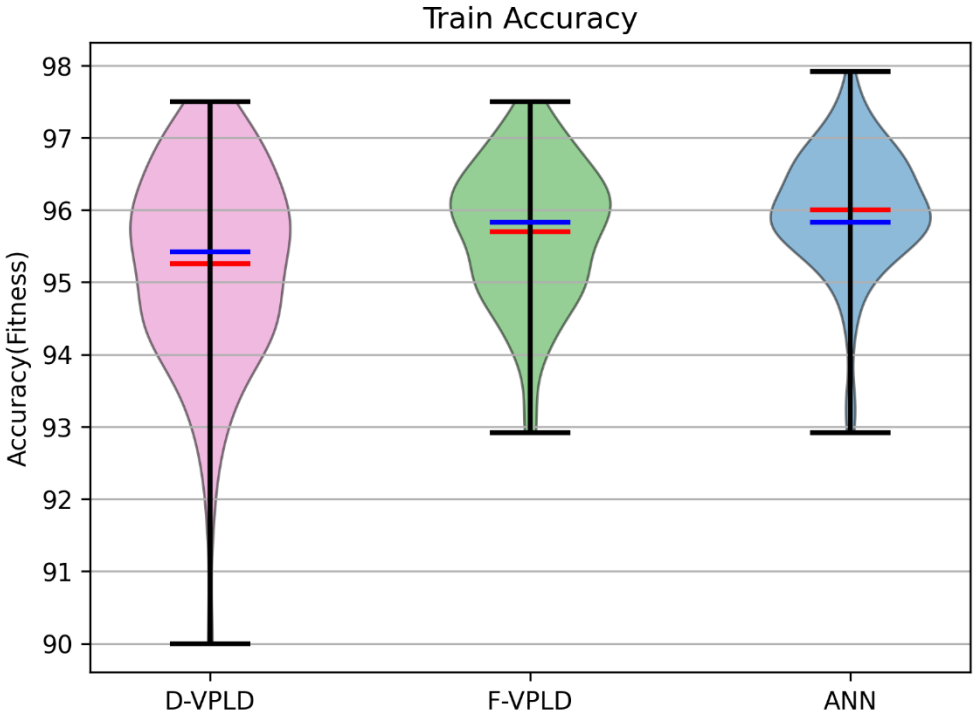

Fig. 5-24 A violin plot comparison of the training accuracy. Shows distribution of results, Maximum/Minimum fitness, median(blue line) and mean(red line)

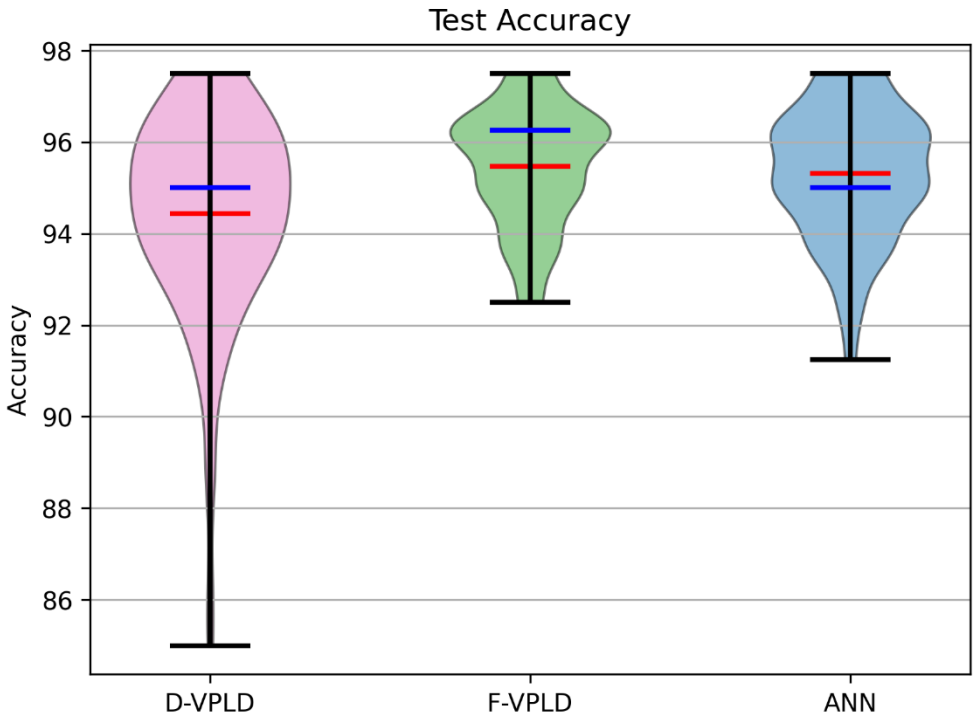

Fig. 5-25 A violin plot comparison of the test accuracy. Shows distribution of results, Maximum/Minimum fitness, median (blue line) and mean (red line)

Examples of Evolved solutions

The examples shown are the solutions that achieved the highest training accuracy across the 100 results of each architecture; of the three examples, the ANN has the highest training accuracy at 97.9%, both VPLDs achieve a training accuracy of 97.5% (Table 5-16). All three D-VPLD, F-VPLD and ANN achieve the same test accuracy at 96.3%; there are results that achieved a higher test accuracy but did not achieve the highest training accuracy, for example the D-VPLD and F-VPLD both have a result with 97.1 training accuracy and 97.5% test accuracy. The example D-VPLD has one false positives and two false negative (Fig. 5-26), the F-VPLD result has two false positives and one false negative (Fig. 5-27) and the ANN result has two false positives and one false negative (Fig. 5-28). In the case of melanoma classification sensitivity (5.22) is the probability that a skin lesion is correctly classified as malignant (cancerous), specificity (5.23) is the probability the skin lesion is correctly classified as benign (non-cancerous).

$$sennsitivity = \frac{TP}{TP + FN} \tag{5.22}$$

$$specificity = \frac{TN}{TN + FP} \tag{5.23}$$

Table 5-16 Example Solution Accuracy (Fitness) for Melanoma Classification

Result	Training Accuracy	Test Accuracy	Sensitivity	Specificity
D-VPLD	97.5%	96.3%	97.5%	95%
F-VPLD	97.5%	96.3%	95%	97.5%
ANN	97.9%	96.3%	100%	92.5%

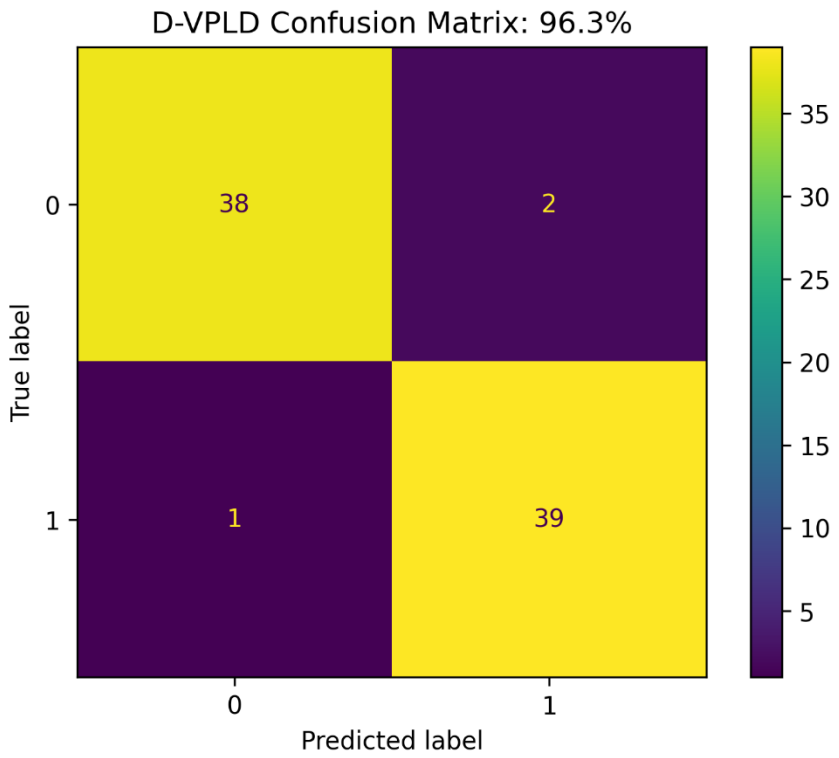

Fig. 5-26 Confusion Matrix for D-VPLD example, 97.5% training accuracy, 96.3% test accuracy, 97.5% sensitivity, 95% specificity

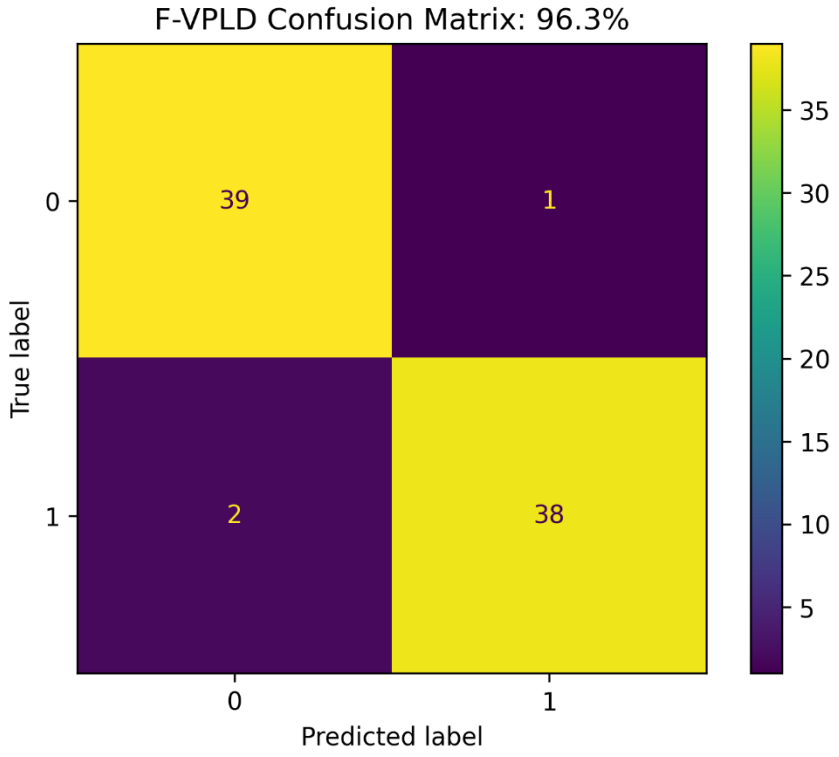

Fig. 5-27 Confusion Matrix for F-VPLD example, 97.5% training accuracy, 96.3% test accuracy, 95% sensitivity, 97.5% specificity

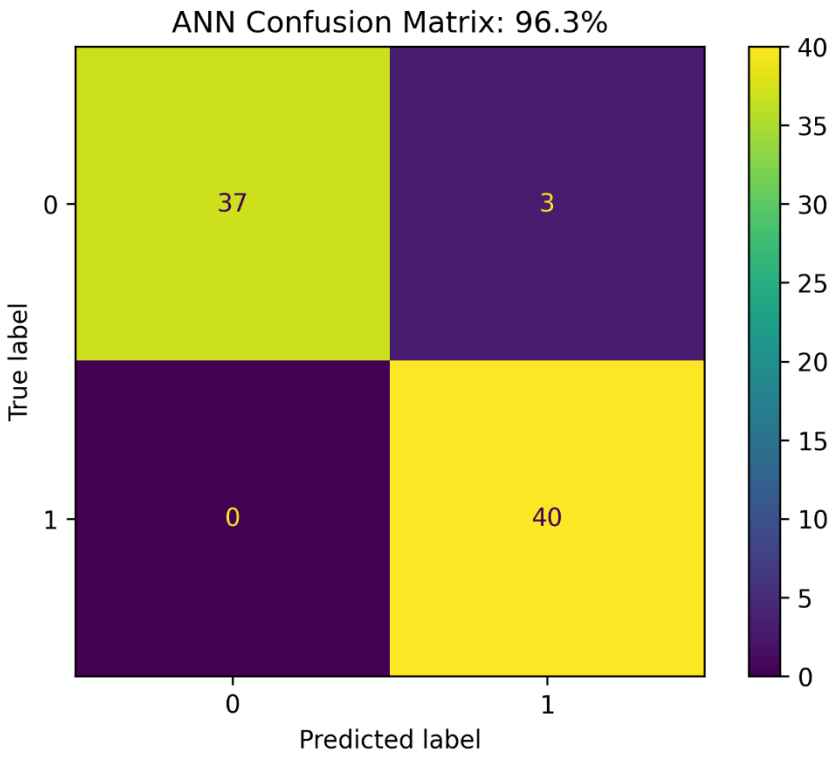

Fig. 5-28 Confusion Matrix for ANN example, 97.9% training accuracy, 96.3% test accuracy, 100% sensitivity, 92.5% specificity

5.2.8 Conclusions

In this section the D-VPLD using digital logic, and the F-VPLD using floating-point data types and simple mathematical arithmetic, were compared to an ANN for the classification of skin lesions as either cancerous melanomas or non-cancerous common/atypical nevus. The three architectures are evolved using the same GA to classify the lesions using images from the PH2 database. The raw images underwent preprocessing prior to segmentation of the skin lesion, after segmentation a set of ABCD and GLCM features are extracted which are used as the input into the classifiers.

Comparing the evolutionary efficiency of the three architectures, all three required a comparable number of generations to converge to an optimal solution. In regard to computation time, the D-VPLD required approximately three times the computation time the ANN and F-VPLD required to evolve. The D-VPLD requires more computation time because of the extra operations required to deal with pseudo bits. The D-VPLD has to convert the extracted feature data into to binary strings for input and convert the output from binary to the data format required for classification.

The D-VPLD, F-VPLD and ANN all had excellent classifier performance, with the best solutions of each achieving a test accuracy greater than 96%. When looking at the average accuracy achieved across the one hundred evolution attempts for each architecture, the D-VPLD, F-VPLD and ANN all have comparable results. This experimentation has shown the VPLD using both digital logic and mathematical arithmetic can achieve comparable accuracy with an ANN when evolved for the classification of melanoma.

In future work the VPLD could be investigated for other pattern recognition tasks, such as object recognition for a pick and place robot, or further applications in medical data classification.

Chapter 6

Chapter 6: The Effect of VPLD Hyperparameters

This chapter investigates the effect of the VPLD hyperparameters for robotic control and pattern recognition. Both the evolutionary efficiency and the controller/classifier performance are considered. For each controller a set of experiments are conducted to evaluate multiple architectures for four tasks: 1) gait control of a hexapod; 2) 2WD robot autonomous navigation; 3) character recognition; and 4) melanoma classification. These experiments are used to find the optimal architectures for the VPLD. Different hyperparameters are implemented for each experiment, to find the effects of altering the number of layers, number of FABs and types of functions within the FAB.

The results of the experimentation conducted for each of the four tasks will be discussed individually, and at the end of the chapter the overall effect of the hyperparameters will be discussed. Each subsection will also discuss how/why the optimal architectures used in Chapter 4 and Chapter 5 are chosen.

The variation in the ANN architectures are listed in Appendix A.

6.1. Hexapod Robotic Control

This section contains the description and results of the alternative VPLD and EHW architectures that are investigated for hexapod gait control but found to have lower performance than the controllers described in sections 4.1. A discussion is made on the different hyperparameters used and the effect they have on the outcome of evolution.

6.1.1 VPLD and EHW Control System Architectures

The VPLD/EHW architectures described below use the same inputs and outputs as the architecture described in section 4.1, but differ in the number of layers and the number of FABs in each layer (Table 6-1).

Table 6-1 Summary of D-VPLD architectures Evaluated for the Hexapod robot

System	Configurations Tested
D-VPLD	• 5-5 • 10-5 • 20-5 • 8-8-8-8-5

The specific architectures tested are, a) a five-layer 8-8-8-8-5 architecture (Fig. 6-1) and b) three, two layered architectures 20-5, 10-5 and 5-5 (Fig. 6-2).

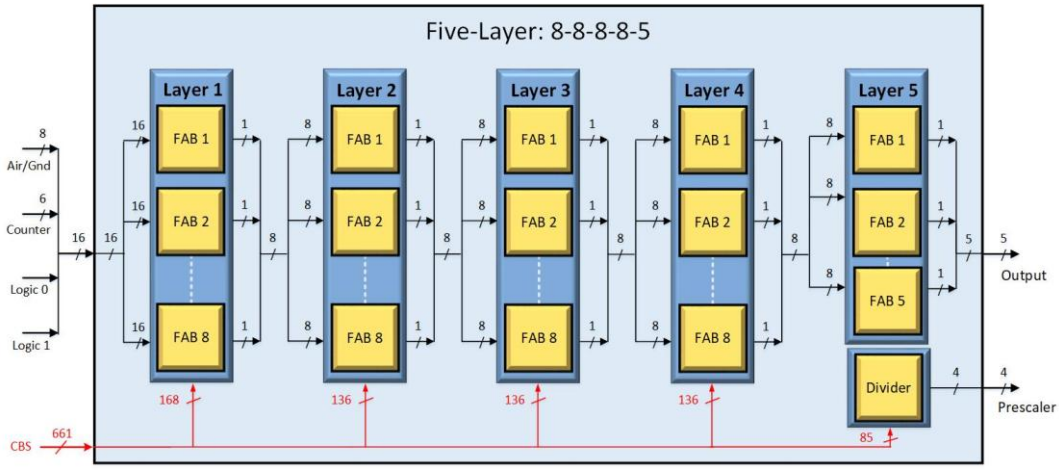

Fig. 6-1 Five-layer cartesian architecture, 8-8-8-8-5. Layers one to four contain eight FABs, layer five contains five FABs

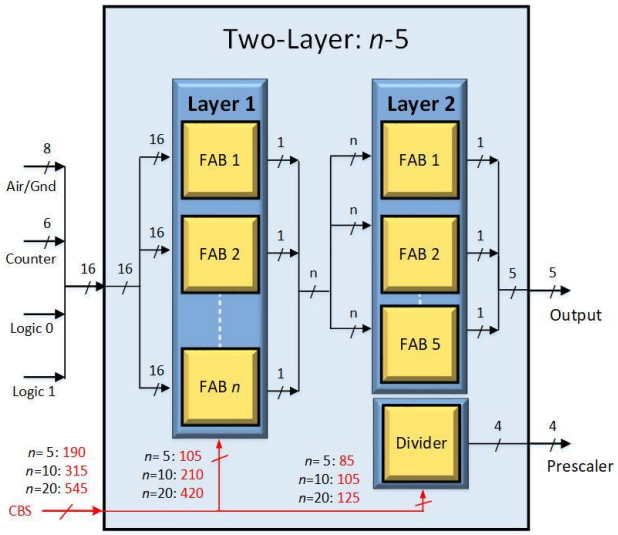

Fig. 6-2 Two-layer cartesian architecture, n-5. Layer one contains *n*=20 or 10 or 5 FABs, layer two contains five FABs

As a recap the complete VPLD controller used to drive a leg of the hexapod is made up of three individual VPLD units. The architecture for each VPLD unit requires three groups of inputs to show the current state of the leg motion and two outputs to control the respective leg servo motor. The inputs are the ground/air phase signal, a counter signal, and logic 0 and 1 signals. The first of the two outputs is a signed 5-bit value that gives the angular change of the joint for the given step. The second output is a 4-bit prescaler value that is used to scale the magnitude of the angular change.

For all four architectures each FAB contains four multiplexers and a single function element comprised of a LUT. All the architectures use the same function element (Table 6-2), which consists of 32 digital functions using standard combinational logic, with 'AND', 'OR', 'XOR' and 'NOT' logic operators.

Table 6-2 VPLD and EHW Function Element LUT

No.	Function	No.	Function
0	*A*	16	*(A&C)^(B&D)*
1	*B*	17	*(A&D)^(C&B)*
2	*C*	18	*(!A)&B&C&D*
3	*D*	19	*A&(!B)&C&D*
4	*!A*	20	*A&B&(!C)&D*
5	*!B*	21	*A&B&C&(!D)*
6	*!C*	22	*(A|B)&(C|D)*
7	*!D*	23	*(A|C)&(B|D)*
8	*A&B&C&D*	24	*(A|D)&(C|B)*
9	*!(A&B&C&D)*	25	*(A^B)&(C^D)*
10	*A|B|C|D*	26	*(A^C)&(B^D)*
11	*!(A|B|C|D)*	27	*(A^D)&(C^B)*
12	*(A&B)|(C&D)*	28	*A&B*
13	*(A&C)|(B&D)*	29	*A&C*
14	*(A&D)|(C&B)*	30	*A&D*
15	*(A&B)^(C&D)*	31	*(A&B)^(C|D)*

6.1.2 Results

In this section the results from the experiments performed for each architecture are compared, to show how and why the optimal architecture discussed in section 4.1 is chosen. The controllers are evolved to produce an optimal walking gait for a hexapod robot using the same GA. The walking gait allowed the robot to walk forward in a straight line, maintaining a constant heading and body attitude. Three hundred solutions for each controller are evolved, with the maximum number of generations limited to five hundred for each phase.

The controllers are evaluated based on their evolutionary efficiency, so how many generations and what is the computation time required to reach a solution. Two platforms are used to implement the controllers (Table 6-3).

Table 6-3 Development platforms processor properties

Platform	Controllers Tested on Platform	Hardware Description
Terrassic DE10 Nano	EHW	Cyclone V FPGA with a Dual-Core ARM Cortex-A9 SOC hardcore processor (925 MHz).
Desktop PC	VPLD	Intel i7-11700k Octa-Core processor (3.6 GHz)

Evolutionary Efficiency

The graph of the fitness per generation (Fig. 6-3) of the four architectures for the VPLD and EHW show the controller performance (best fitness) per generation and the evolutionary efficiency (number of generations to reach a solution). The EHW and VPLD results are similar, as was previously stated this is because the GA, controller architecture, and chromosome are identical. Any differences in the results are due to the random initial population, crossover, and mutation.

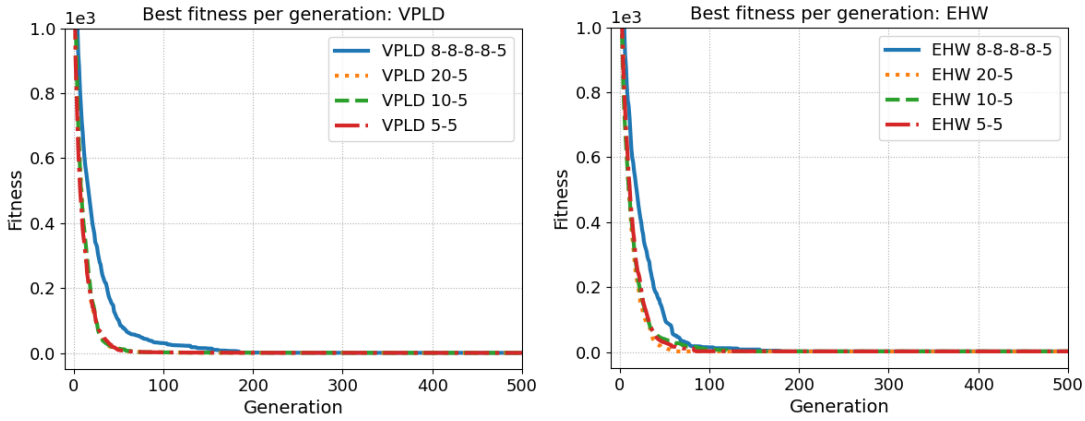

Fig. 6-3 Comparison of VPLD (left) and EHW (right) best fitness per generation averaged over 300 runs

All the architectures evolved to the same controller performance with only a small number of generations required to evolve the required fitness. All architectures other than the 8-8-8-8-5 evolved to the required fitness in less than 100 generations. The same results have been tabulated, (Table 6-4 - Table 6-7) showing both the number of generations and the time taken to evolve to the required fitness.

Table 6-4 VPLD number of generations required to successfully evolve a controller

Architecture	Mean	Median	CV
VPLD 8-8-8-8-5	105.06	94.5	0.5
VPLD 20-5	60.04	56.5	0.4
VPLD 10-5	62.98	61	0.35
VPLD 5-5	68.63	61	0.43

Table 6-5 EHW number of generations required to successfully evolve a controller

Architecture	Mean	Median	CV
EHW 8-8-8-8-5	111.12	93.5	0.61
EHW 20-5	65.11	59	0.38
EHW 10-5	68.51	63.5	0.39
EHW 5-5	79.04	71	0.42

Table 6-6 VPLD computation time required to successfully evolve a controller

Architecture	Mean[s]	Median[s]	CV
VPLD 8-8-8-8-5	2.53	2.28	0.49
VPLD 20-5	1	0.93	0.39
VPLD 10-5	0.81	0.78	0.33
VPLD 5-5	0.75	0.67	0.41

Table 6-7 EHW computation time required to successfully evolve a controller

Architecture	Mean[s]	Median[s]	CV
EHW 8-8-8-8-5	398.7	337.29	0.6
EHW 20-5	170.92	155.35	0.37
EHW 10-5	135.74	126.2	0.38
EHW 5-5	131.28	118.13	0.41

The evolution time reduced as the number of layers reduced, and the number of FABs within the layers are reduced, with the *8-8-8-8-5* architecture requiring significantly more time to evolve than the other architectures (Table 6-6 & Table 6-7). This is because a larger number of layers requires a larger chromosome, necessitating more processing time for the GA process. The number of generations to evolve must also be considered, however as is seen the 5-5 architecture for the VPLD and EHW is the fastest to evolve but requires a few more generations on average to evolve than the 10-5 and 20-5 architectures. The consistency in the required time to evolve is much better for the two layer architectures compared to the larger five layer architecture.

As all the architectures could evolve a solution, the deciding factor is evolutionary efficiency, with the overriding factor being the time taken to evolve a solution, rather than the number of generations. The 5-5 had the best evolution time of 0.75 seconds for the VPLD and 131.28 seconds for the EHW.

6.1.3 Conclusions

For the hexapod controller architectures two hyperparameters were changed between the architectures; 1) number of layers, two-layer architectures versus a five-layer architecture; and 2) varying number of FABs in the first input layer for a two-layer architecture. The two-layer architectures are faster to evolve, requiring less than 70 generations on average, the larger multilayer architecture required more generations at approximately 105 on average. The evolution time reduced as the number of layers reduced, with the 8-8-8-8-5 architecture requiring significantly more time to evolve than the other architectures.

For the two-layer architectures as the number of FABs in the input layer increase the time to evolve increased, but the number of generations decrease. This is because the smaller the VPLD architecture, the faster it is to perform the crossover operation during reproduction, and also difference in propagation delay, although almost negligible accumulates over thousands of times the VPLD is executed during the fitness assessment.

All architectures are evolved to the same fitness level, and can be used to control the robot in simulation and on the physical robot.

6.2. 2WD Mobile Robot Control

This section contains the description and results of the alternative VPLD architectures that are investigated for the control of a 2WD mobile robot. These alternative architectures are found to have lower performance than the architecture described in section 4.2. The description of the robotic control problem and GA used to evolve the VPLD architectures discussed below can be found in sections 4.2.1 and 4.2.5 respectively. To evaluate the VPLD architectures they are evolved specifically for the combined obstacle avoidance and light following problem.

6.2.1 D-VPLD Control System Architectures

For the 2WD robotic control problem five architectures with FABs using the same function element are evaluated (Table 6-8).

Table 6-8 Summary of D-VPLD architectures Evaluated for the 2WD mobile robot

System	Configurations Tested
D-VPLD	• 4-4 • 8-4 • 16-4 • 16-16-4 • 16-16-16-4

Three two-layer architectures (Fig. 6-4) with variations in the number of FABs in the first layer, 4-4, 8-4, and 16-4. Along with the two-layer architectures, another two multi-layer architectures are investigated, 16-16-4, and 16-16-16-4 (Fig. 6-5). Each architectures use the same encoded inputs from the IR proximity sensors and light intensity sensors required for the combined obstacle avoidance and light following task (as detailed in section 4.2.1). The four output FABs combine to produce a 4-bit signed value used to set the RPM of one of the robot's wheels. As discussed in section 4.2.1 two D-VPLD units are combined to drive the robot, one controls the left wheel, the other controls the right wheel.

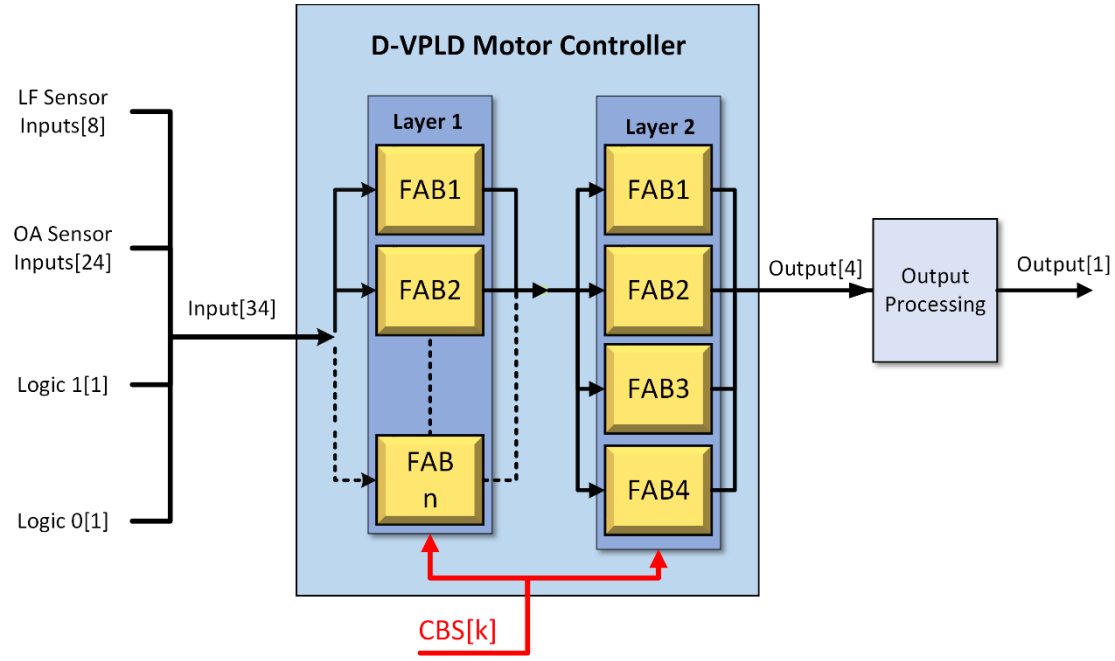

Fig. 6-4 D-VPLD Architectures for 2WD robot control; two-layer architecture with n FABs in the first layer (n=4, 8, or 16) and 4 FABs in the second layer.

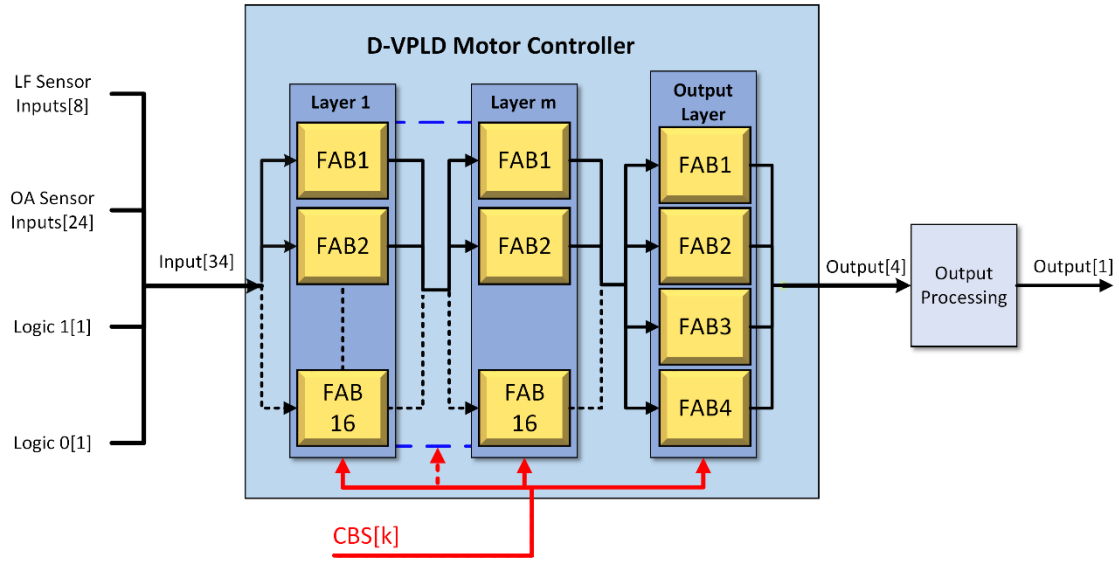

Fig. 6-5 D-VPLD Architectures for 2WD robot control; multi-layer architecture with 16 FABs in each layer except for the final output layer which uses 4 FABs. Architectures with m=2, or 3 layers (excluding output layer) are evaluated

For all five architectures each FAB contains four multiplexers and a single function element comprised of a LUT. All D-VPLD architectures use the same function element, which consists of 21 digital functions using standard combinational logic, with 'AND', 'OR', 'XOR', and 'NOT' logic operators (Table 6-9).

Table 6-9 D-VPLD 2WD Robot FE LUT

No.	Function	No.	Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

6.2.2 D-VPLD Results

The five D-VPLD architectures are evolved for the control of the 2WD robot for the combined obstacle avoidance and light following task; the controllers are required to drive the robot to the light source without crashing into obstacles in the path of the robot or the environment boundaries. The evolutionary efficiency and controller performance for each architecture is investigated. One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs are used to assess the system performance. Each architecture evolution is limited to 250 generations.

Each architecture is compared to show how/why the optimal architecture discussed in section 4.2 is chosen. The effect of the hyperparameters that differ between the five architectures are discussed.

Evolutionary Efficiency

The results from the evolution of the D-VPLD architectures is shown below; in the fitness per generation graph (Fig. 6-6) it can be seen clearly that the three two-layer architectures are the faster to evolve and converge to a higher fitness level on average. When looking at the computation time and number of generations to evolve (Table 6-10 & Table 6-11) it can be seen that the two-layer architectures require less generations and significantly less time to evolve than the multi-layer architectures.

When considering just the two-layer architectures, as the number of FABs in the first layer increases the generations to evolve decrease. However the trend changes slightly for the time to evolve, this is because as the architecture gets larger the computation time increases for the production of offspring in the reproduction phase of a GA; also the propagation delay increases by a minor amount for a larger architecture, but this accumulates due to thousands of executions performed by the VPLD during the fitness assessment. For the multi-layer architectures, comparing 16-4, 16-16-4, and 16-16-16-4, both the required generations and computation time to converge to a solution increases as the number of layers increases.

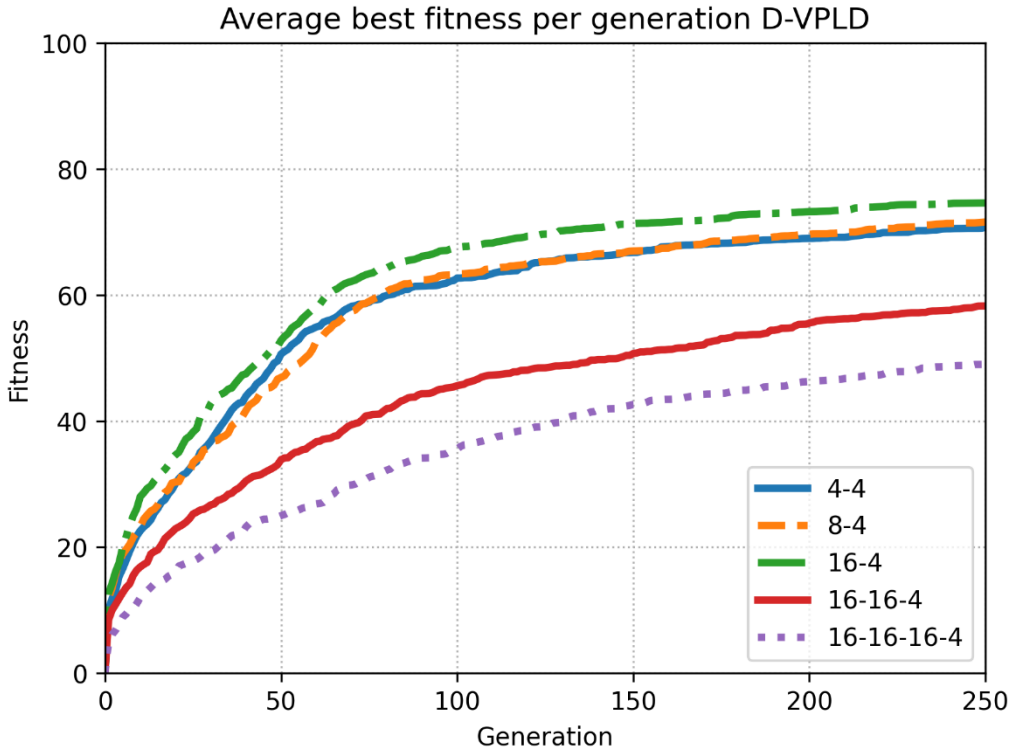

Fig. 6-6 A comparison of the OA-LF fitness per generation of the five D-VPLD architectures

Table 6-10 The Number of Generations to Evolve D-VPLD for OA-LF Task

Controller	Mean	Median	CV
4-4	164.73	175	0.36
8-4	162.3	168.5	0.37
16-4	157.97	162.5	0.34
16-16-4	172.95	183	0.35
16-16-16-4	171.69	180.5	0.33

Table 6-11 Computation Time Required to Evolve D-VPLD for OA-LF Task in Seconds

Controller	Mean	Median	CV
4-4	100.13	107.34	0.37
8-4	97.87	101.54	0.39
16-4	98.44	100.94	0.36
16-16-4	124.6	129.6	0.36
16-16-16-4	140.72	152.17	0.34

Controller Performance

For the controller performance it can be seen for the three two-layer architectures as the number of FABs in the first layer increases, the average fitness level increases (Table 6-12). However, when the number of layers increases the fitness level decreases significantly. This may be because the 250-generation limit applied to the evolution is not long enough to find an optimal configuration for the larger architecture. The decreased fitness and slow evolution of the multi-layer architectures could also be contributed to the multi-point crossover being overly destructive to the larger architectures, chains of genes passing information between layers could be broken during the multi-point crossover, in future work the use of different crossover operations could be investigated.

Table 6-12 The D-VPLD Fitness reached for the OA-LF Task

Controller	Mean	Median	CV
4-4	70.76	74.64	0.18
8-4	71.66	75.23	0.18
16-4	74.64	76.93	0.13
16-16-4	58.28	58.94	0.27
16-16-16-4	49.25	46.56	0.33

The optimal controller used in section 4.2 is the 16-4 architecture, of the five architectures evaluated the 16-4 architecture is the most optimal because it required the least generations on average and achieved the highest average fitness. The 16-4 architecture also had the lowest variance in fitness.

6.2.3 F-VPLD Control System Architectures

For the F-VPLD in the 2WD robotic control problem, five architectures with FABs using the same function element are investigated (Table 6-13).

Table 6-13 Summary of D-VPLD architectures Evaluated for the 2WD mobile robot

System	Configurations Tested
F-VPLD	• 4-4 • 8-4 • 16-4 • 16-16-4 • 16-16-16-4

Three two-layer architectures (Fig. 6-7) with variations in the number of FABs in the first layer, 4-1, 8-1, and 16-1; as well as two multi-layer architectures are investigated, 16-16-1, and 16-16-16-1. The five architectures input is the floating-point data from the IR proximity sensors and light intensity sensors required for the combined obstacle avoidance and light following task as detailed in section 4.2.3. The F-VPLD has a single output FAB which is a floating-point value used to set the RPM of one of the robot's wheels. As discussed in section 4.2.3 two F-VPLD units are combined to drive the robot, one controls the left wheel, the other controls the right wheel.

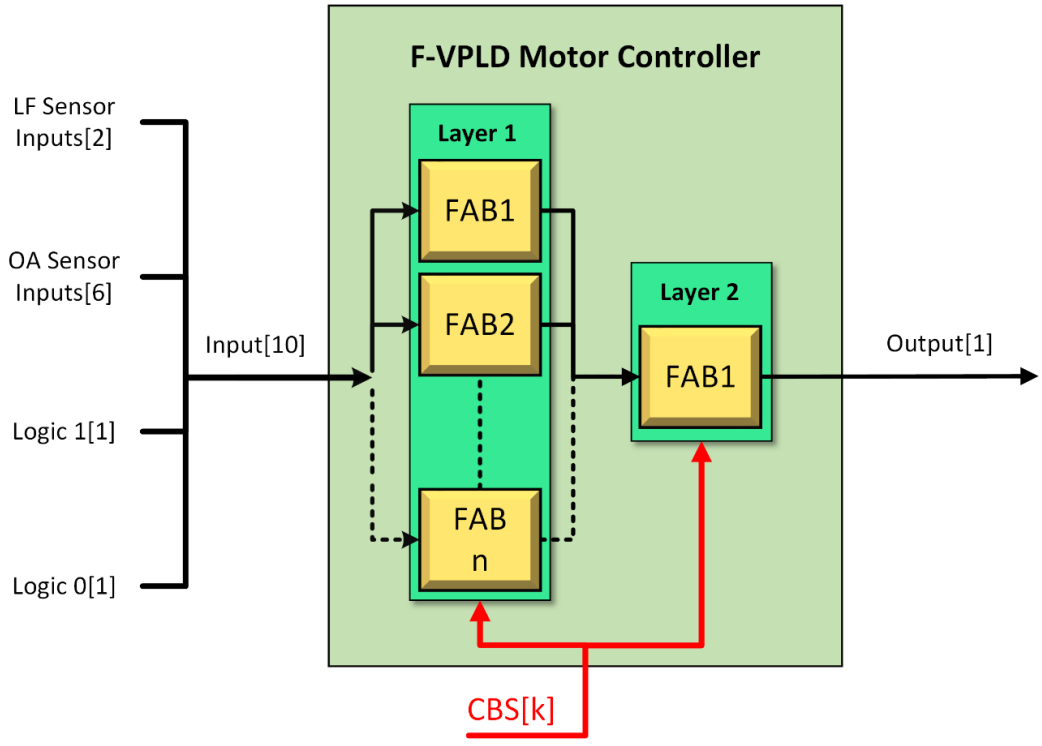

Fig. 6-7 F-VPLD Architectures for 2WD robot control; two-layer architecture with n FABs in the first layer (n=4, 8, or 16) and 1 FAB in the second layer.

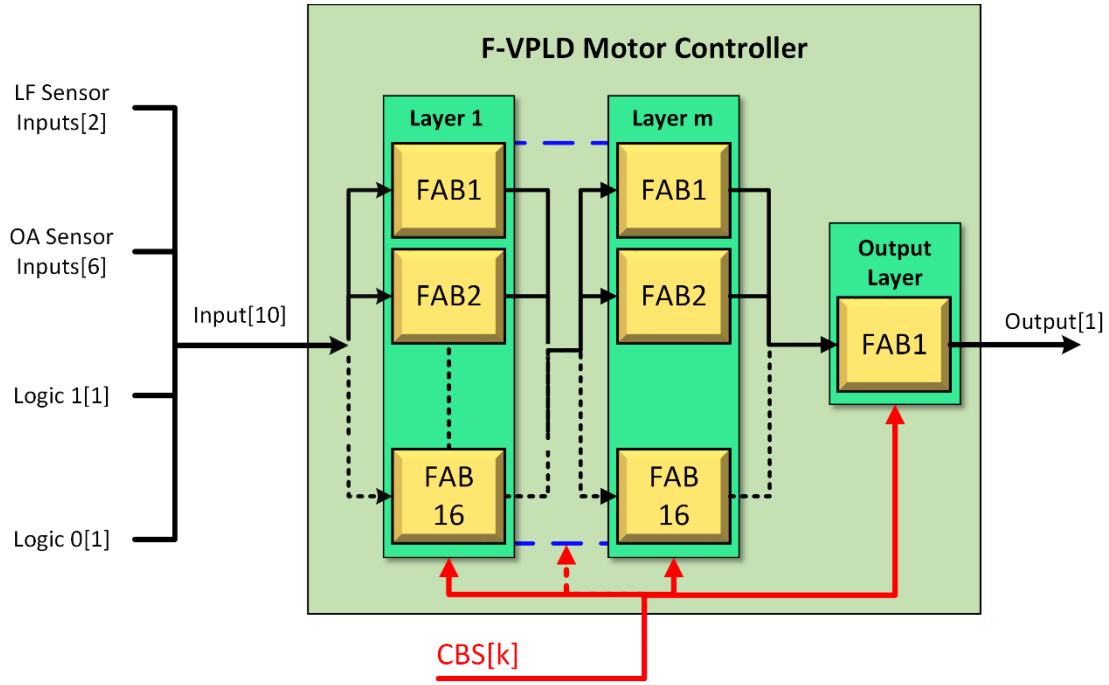

Fig. 6-8 F-VPLD Architectures for 2WD robot control; multi-layer architecture with 16 FABs in each layer except for the final output layer which uses 1 FAB. Architectures with m=2, or 3 layers (excluding output layer) are evaluated

The three two-layer architectures and the two multi-layer architectures use the same FAB. Each FAB contains four multiplexers and a function element. The function element LUT (Table 6-14) has 21 functions which use standard mathematical arithmetic, with 'plus', 'minus' and 'multiply' operations. The activation functions *f* is a piece-wise linear activation function (with output ranges from -2 to +2).

Table 6-14 F-VPLD 2WD Robot FE LUT

No.	Function	No.	Function
0	f((A+B)*(C-D))	11	f((-A)+B+C+D)
1	f(A+B+C+D)	12	f(A+(-B)+C+D)
2	f(-(A+B+C+D))	13	f(A+B+(-C)+D)
3	f(A*B*C*D)	14	f(A+B+C+(-D))
4	f(-(A*B*C*D))	15	f((A*B)+(C*D))
5	f((A+B)*(C+D))	16	f((A*C)+(B*D))
6	f((A+C)*(B+D))	17	f((A*D)+(B*C))
7	f((A+D)*(B+C))	18	f((A*B)-(C*D))
8	f((A+B)-(C+D))	19	f((A*C)-(B*D))
9	f((A+C)-(B+D))	20	f((A*D)-(B*C))
10	f((A+D)-(B+C))		

6.2.4 F-VPLD Results

As for the D-VPLD, the five F-VPLD architectures are evolved for the control of the 2WD robot for the combined obstacle avoidance and light following task; the controllers are required to drive the robot to the light source without crashing into obstacles in the path of the robot or the environment boundaries. The evolutionary efficiency and controller performance for each architecture is investigated. One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs are used to assess the system performance. Each architecture evolution is limited to 250 generations.

Each architecture is compared to show how/why the optimal architecture discussed in section 4.2 is chosen. The effect of the hyperparameters that differ between the five architectures are discussed.

Evolutionary Efficiency

When viewing the average fitness per generation graph (Fig. 6-9), two observations (which are the same for the D-VPLD) are noted; 1) for the two-layer architectures, the rate of evolution increases with an increase in FABs in the first layer; and 2) for the multi-layer architectures considering the 16-4, 16-16-4, and 16-16-16-4, as the number of layers increases the rate of evolution decreases. Also, for the generations and computation time to evolve similar trends are observed for the D-VPLD are present. The number of generations to evolve is larger for the 16-16-4 and 16-16-16-4 multilayered architectures compared to the 16-4 architectures, however there is not a linear relationship between the number of layers and the average generations to evolve (Table 6-7).

Table 6-15 The Number of Generations to Evolve F-VPLD for OA-LF Task

Controller	Mean	Median	CV
4-4	135.12	134.5	0.47
8-4	125.56	123.5	0.55
16-4	127.56	117.5	0.47
16-16-4	151.19	168	0.44
16-16-16-4	150.66	150.5	0.44

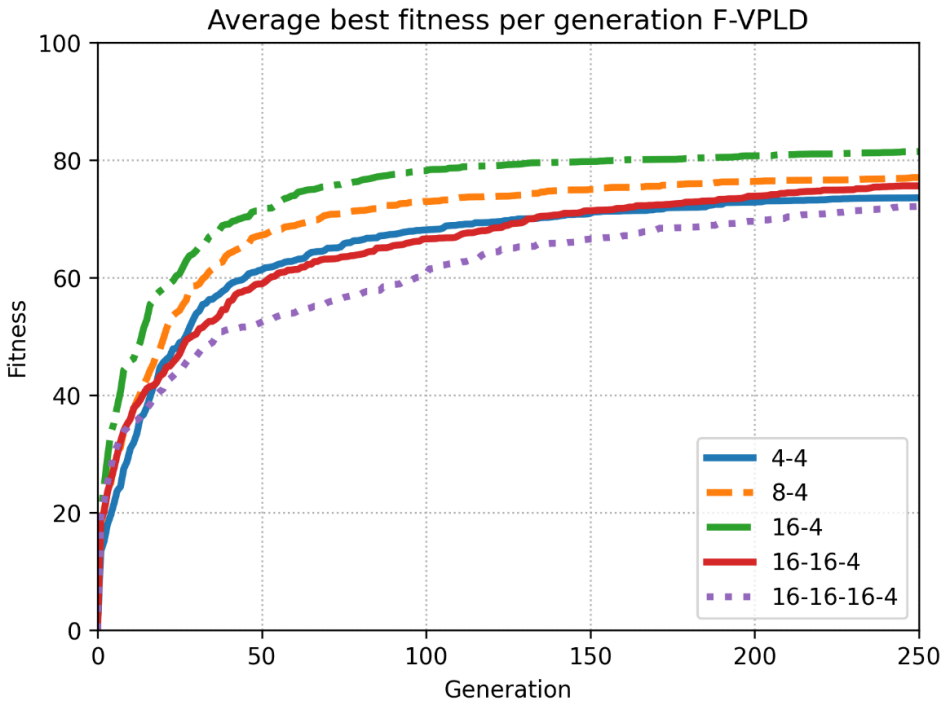

Fig. 6-9 A comparison of the OA-LF fitness per generation of the five F-VPLD architectures

Comparing the D-VPLD and F-VPLD time to evolve (Table 6-11 & Table 6-16), for the same layer and FAB architecture the F-VPLD is faster to evolve on average, this is because the F-VPLD use the floating-point data type native to the CPU and maths coprocessor environment, whereas the D-VPLD use binary digital logic which requires the sensors input data to be encoded as binary data, and then output has to be converted back to floating point to drive the robot, these extra operations slows down the D-VPLD evolution.

Table 6-16 Computation Time Required to Evolve F-VPLD for OA-LF Taskin Seconds

Controller	Mean	Median	CV
4-4	68.03	66.7	0.5
8-4	62.63	58.1	0.6
16-4	67.08	61.31	0.52
16-16-4	89.52	97.97	0.48
16-16-16-4	98.67	96.69	0.46

Controller Performance

Again, similar trends as were seen with the D-VPLD are observed for the F-VPLD. For the two-layer architectures, the average evolved fitness increase with an increase in the number of FABs in the first layer (Fig. 6-17). For the multilayered architectures again as the layers increase the fitness decreases, however compared to the D-VPLD fitness levels, the F-VPLD 16-16-4 and 16-16-16-4 architectures achieve a higher fitness.

Table 6-17 The F-VPLD Fitness reached for the OA-LF Task

Controller	Mean	Median	CV
4-4	73.61	76.62	0.22
8-4	77.07	76.39	0.13
16-4	81.49	83	0.07
16-16-4	75.72	77.12	0.14
16-16-16-4	72.17	74.5	0.17

When choosing the optimal F-VPLD architecture to use in section 4.2, both the evolutionary efficiency and controller performance are considered. The two best architectures are the 16-4 and 8-4. The 8-4 architecture is the fastest to evolve, however the 16-4 architecture only requires a few seconds extra on average. The 16-4 architecture reaches a higher fitness compared to the 8-4 F-VPLD. Ultimately the 16-4 F-VPLD architecture is chosen as the most optimal architecture because it reaches the highest fitness level and its time to evolve is comparable to the fastest F-VPLD architecture (8-4).

6.2.5 Conclusions

For the 2WD autonomous navigation control architectures three hyperparameters were changed between the architectures; 1) number of layers, (two, three, and four layer architectures were investigated; 2) varying number of FABs in the first input layer for a two-layer architecture; and 3) using digital variables and logic function array blocks versus using floating-point variables and arithmetic function array blocks.

D-VPLD

The D-VPLD, results where similar in the 2WD and the hexapod robotic control task. The smaller two-layer architectures require less time to evolve because of the reduced computation during the reproduction stage of the GA. As the number of FABs in the first layer increases the number of generations required to evolve the D-VPLD architecture decreases. The median time to evolve for the three two-layer architectures decreases with the number of FABs increasing in the first layer. This is because the 2WD robotic

simulation has a higher computational load compared to the simulation used to evolve the hexapod. Therefore, the extra fitness assessments required by taking more generations to evolve has a bigger impact on the time to evolve, rather than the computation time to complete the reproduction etc. because the size difference between the two-layer architectures is relatively small.

For the D-VPLD controller performance, the average fitness achieved decreased both for a reduced number of layers and for a reduced number of FABs in the first layer. The decreased fitness and slow evolution of the multi-layer architectures could be contributed to the multi-point crossover. This crossover method can be destructive to the multilayer architectures, because chains of genes passing information between layers could be broken during the multi-point crossover.

F-VPLD

When using the F-VPLD for the 2WD robot, similar trends to the D-VPLD are observed. The smaller two-layer architectures require less time to evolve than the three- and four-layer architectures. Time to evolve increases with an increase in the number of layers. For the two-layer architectures when increasing the number of FABs in the first layer, the median generations to evolve decreases the same as the D-VPLD. The F-VPLD controller performance (fitness) decreases with an increase in the number of layers. For the two-layer architectures, with an increase in FABs in the first layer, the average fitness level achieved increases, and the level of variance in the fitness level achieved decreases.

Comparing the use of digital logic FABs for the D-VPLD, and the use of floating-point arithmetic for the F-VPLD. The F-VPLD architectures require less generations and time to evolve and reach higher fitness levels on average. For the number of generations and the fitness level, the increased performance of the F-VPLD may be because the mathematical functions and floating-point data have a higher fidelity compared to digital logic, and this higher fidelity makes solving complex and dynamic control tasks such as the autonomous navigation task of a 2WD robot easier. For the difference in the time to evolve, this is obviously contributed to in part the increased generations, but also because the D-VPLD uses binary digital logic. The use of digital logic requires the sensors input data to be encoded as binary data, and then the output has to be converted back to floating point to drive the robot, these extra operations slows down the execution of the D-VPLD evolution.

6.3. Character Recognition

This section contains the description and results of the alternative VPLD architectures that are investigated for character recognition but found to have lower performance than the classifiers described in section 5.1. A description of the GA used to evolve the architectures discussed below can be found in section 5.1.4.

6.3.1 D-VPLD and F-VPLD Classifier Architectures

The D-VPLD and F-VPLD share the same architecture and the same FAB architecture, however the D-VPLD and F-VPLD use different function element operations and variable data type. The input for the D-VPLD and F-VPLD is a character image converted to an array. The input array is in a binary form (0-black pixel and 1-white pixel). The output is a set of 26 values, one for each letter in the alphabet. The output value with the highest value determines the selected character.

The VPLD hyperparameters tested are broken into three parts: 1) the number of FABs in the output layer of a two-layer architecture; 2) two different function elements; and 3) different post-processing operations on the final FABs outputs to get the prediction (Table 6-18).

Table 6-18 D-VPLD and F-VPLD architectures evaluated for Character Recognition

System	Configurations Tested

D-VPLD	• 52-26
	  ○ with FE LUT A
	  ○ with FE LUT B
	• 52-78
	  ○ with FE LUT A and mean output grouping
	  ○ with FE LUT B and mean output grouping

F-VPLD	• 52-26
	  ○ with FE LUT C
	  ○ with FE LUT D
	• 52-78
	  ○ with FE LUT C and mean output grouping
	  ○ with FE LUT D and max output grouping
	  ○ with FE LUT C and mean output grouping
	  ○ with FE LUT D and max output grouping

Two architectures for the D-VPLD and F-VPLD are investigated: 1) a 52-26 two-layer architecture, where a single FAB determines the level of activation of a given character; and 2) a 52-78 two-layer architecture, where 3 FABs are grouped together to determine the level of activation of a given character. The outputs of the D-VPLD and F-VPLD go through the SoftMax function.

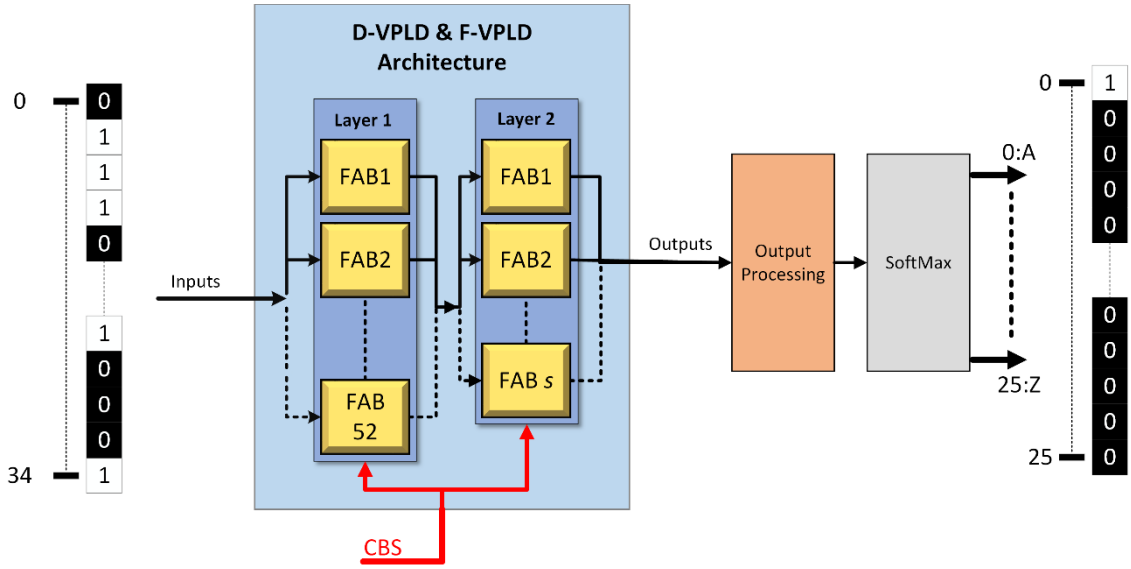

Fig. 6-10 VPLD and F-VPLD architecture. Layer one contains 52 FABs, layer 2 contains s=26 or 78 FABs

In both architectures each FAB contains four multiplexers and a function element, a detailed description of the FABs can be found in section 5.1.2. For both the D-VPLD and F-VPLD two different function element LUTs are investigated. This is done because the functions used in the robotic control problems may not be suitable for character recognition. For the D-VPLD, LUT-A (Table 6-19) has 21 digital functions and LUT-B (Fig. 6-20) has 31 digital functions.

Table 6-19 D-VPLD Character Recognition FE LUT-A

No.	Function	No.	Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

Table 6-20 D-VPLD Character Recognition FE LUT-B

No.	Function	No.	Function
0	(A&B)^(C|D)	16	((!A)&B)|((!C)&D)
1	A&B&C&D	17	(A&(!B))|(C&(!D))
2	!(A&B&C&D)	18	((!A)&(!D))|((!B)&(!C))
3	(!A)&B&C&D	19	(A^B)&(C^D)
4	A&(!B)&C&D	20	(A^C)&(B^D)
5	A&B&(!C)&D	21	(A^D)&(C^B)
6	A&B&C&(!D)	22	((!A)&B)^((!C)&D)
7	A&(!B)&(!C)&(!D)	23	(A&(!B))^(C&(!D))
8	(!A)&B&(!C)&(!D)	24	((!A)&(!D))^((!B)&(!C))
9	(!A)&(!B)&C&(!D)	25	(A|B)&(C|D)
10	(!A)&(!B)&(!C)&D	26	(A|C)&(B|D)
11	A|B|C|D	27	(A|D)&(C|B)
12	!(A|B|C|D)	28	(A^B)&(C^D)
13	(A&B)|(C&D)	29	(A^C)&(B^D)
14	(A&C)|(B&D)	30	(A^D)&(C^B)
15	(A&D)|(B&C)		

The F-VPLD uses floating-point variables and include basic arithmetic operators, ‘plus’, ‘minus’ and ‘multiply’. LUT-C and D both have the same 21 functions, but an activation function is used like those used in ANN. As can be seen in Table 6-21 & Table 6-22 the F-VPLD LUTs are the same but use two different activation functions *f* and *g*. The activation function f is the ReLu activation function, and g is a piece-wise linear activation function.

Table 6-21 F-VPLD Character Recognition FE LUT-C

No.	Function	No.	Function
0	f((A+B)*(C-D))	11	f((-A)+B+C+D)
1	f(A+B+C+D)	12	f(A+(-B)+C+D)
2	f(-(A+B+C+D))	13	f(A+B+(-C)+D)
3	f(A*B*C*D)	14	f(A+B+C+(-D))
4	f(-(A*B*C*D))	15	f((A*B)+(C*D))
5	f((A+B)*(C+D))	16	f((A*C)+(B*D))
6	f((A+C)*(B+D))	17	f((A*D)+(B*C))
7	f((A+D)*(B+C))	18	f((A*B)-(C*D))
8	f((A+B)-(C+D))	19	f((A*C)-(B*D))
9	f((A+C)-(B+D))	20	f((A*D)-(B*C))
10	f((A+D)-(B+C))		

Table 6-22 F-VPLD Character Recognition FE LUT-D

No.	Function	No.	Function
0	g((A+B)*(C-D))	11	g((-A)+B+C+D)
1	g(A+B+C+D)	12	g(A+(-B)+C+D)
2	g(-(A+B+C+D))	13	g(A+B+(-C)+D)
3	g(A*B*C*D)	14	g(A+B+C+(-D))
4	g(-(A*B*C*D))	15	g(A*B)+(C*D))
5	g((A+B)*(C+D))	16	g((A*C)+(B*D))
6	g((A+C)*(B+D))	17	g((A*D)+(B*C))
7	g((A+D)*(B+C))	18	g((A*B)-(C*D))
8	g((A+B)-(C+D))	19	g((A*C)-(B*D))
9	g((A+C)-(B+D))	20	g((A*D)-(B*C))
10	g((A+D)-(B+C))		

The output of the D-VPLD is an array of binary values, for the F-VPLD it is an array of floating-point values. For the 52-26 architecture, where there is one output for each character, the output arrays skip the grouping process and are passed through a SoftMax function. For the 52-78 architecture, the output is combined into groups of three (26x3). Once grouped values are either averaged or the max value of the group is chosen as the output for the respective character. Once the grouped outputs are processed resulting in an output array of size 26, the output is fed into the SoftMax function to get the final result. Note the Max output method is not evaluated with the D-VPLD as each output value is binary and it would not make sense to apply such a function to binary values. It also complicates the evolution for the D-VPLD, as the D-VPLD system has to evolve into a solution where at least one of the outputs in a group corresponding to the correct character is active, while all other groups with 3 FABs must all produce a 0 output.

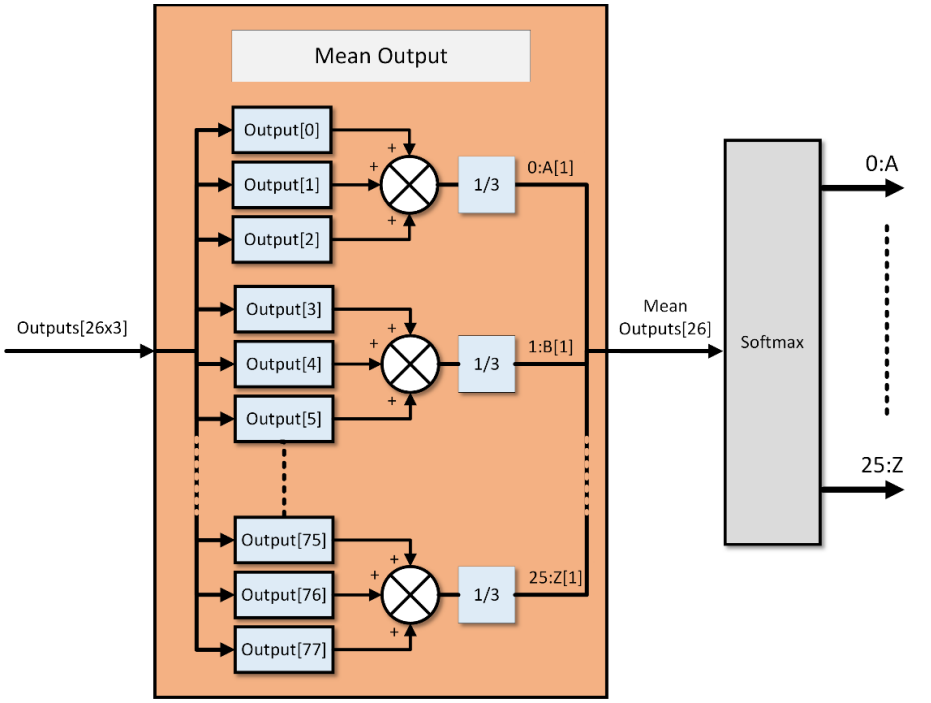

Fig. 6-11 Mean method used to group the 26x3 outputs into one output for each character

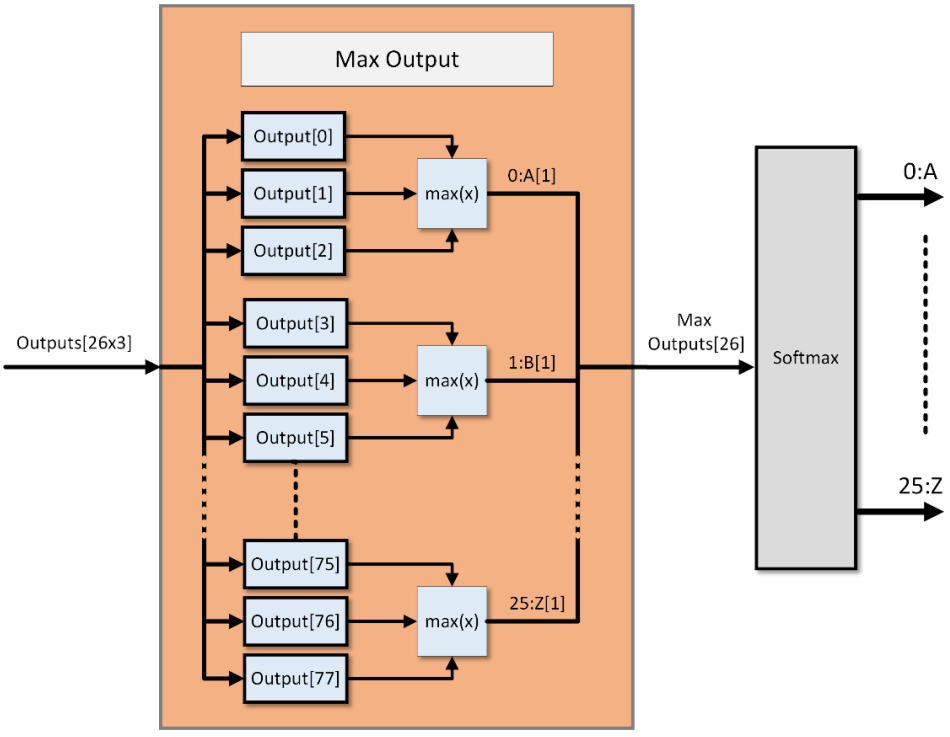

Fig. 6-12 Max method used to group the 26x3 outputs into one output for each character

6.3.2 D-VPLD and F-VPLD Results

Experiments are performed for the D-VPLD and F-VPLD with varying architectures. The D-VPLD and F-VPLD evolutionary efficiency for each architecture is investigated using two different function elements each. For the architectures using the grouped outputs, mean and max grouping functions are tested. The different configurations for each architecture are compared to find the most optimal solutions.

The D-VPLD and F-VPLD systems are evolved to recognise 26 binary character images(A-Z), an optimal solution is a system that can recognise each character. One hundred solutions are evolved for each architecture being evaluated, the data from each of these one hundred GA runs are used to assess the system performance. For each GA run the best fitness per generation and the time taken to run the GA system is recorded, this information allows us to determine the evolutionary efficiency of each system.

Evolutionary Efficiency

For the 52-26 D-VPLD, both the systems tested take over 3000 generations to evolve. The D-VPLD architecture using FE-B which has a bigger LUT is approximately 800 generations faster to evolve than the architecture using FE-A (Table 6-23). Looking at computation time to evolve, the architecture using FE-B is 1 minute 16 seconds faster to evolve than the architecture using FE-A (Table 6-24) both architectures took less than 3 minutes to evolve a solution on average.

For the 52-26 F-VPLD, there is a large difference between the F-VPLD system using FE-C and FE-D. The FE-C solution requires approximately 1633 less generations to evolve than the FE-D solution (Table 6-23). As the difference in number of generations to evolve is large, the computation time required to evolve is not close either, however both solutions take less than a minute on average to evolve. The FE-D solution is 3x slower to evolve (Table 6-24). Both FE-C and FE-D are very similar they only differ in the use of the Relu function versus a piece-wise linear activation function, all mathematical operations are the same. As with ANNs the use of such activation functions has an effect on performance and the learning rate of the architecture. In future work more functions could be investigated to see how they affect the rate of optimization.

Comparing the D-VPLD and F-VPLD results using the 52-26 architecture the best F-VPLD result required less generations and significantly less computation time to evolve. The average best fitness per generation of all the architectures evaluated is shown in Fig. 6-13.

Table 6-23 Character Recognition generations to evolve. 52-26 D-VPLD and F-VPLD comparison using FE-A and FE-B

System	Mean	Median	CV
D-VPLD FE-A	4045.78	3352.5	0.61
D-VPLD FE-B	3241.68	2693	0.59
F-VPLD FE-C	804.11	694	0.47
F-VPLD FE-D	2437.92	1855	0.75

Table 6-24 Character Recognition time to evolve. 52-26 D-VPLD and F-VPLD comparison using FE-A and FE-B

System	Mean[s]	Median[s]	CV
D-VPLD FE-A	153.52	111.1	0.68
D-VPLD FE-B	78.02	64.25	0.59
F-VPLD FE-C	17.28	15.05	0.46
F-VPLD FE-D	52.04	40.22	0.74

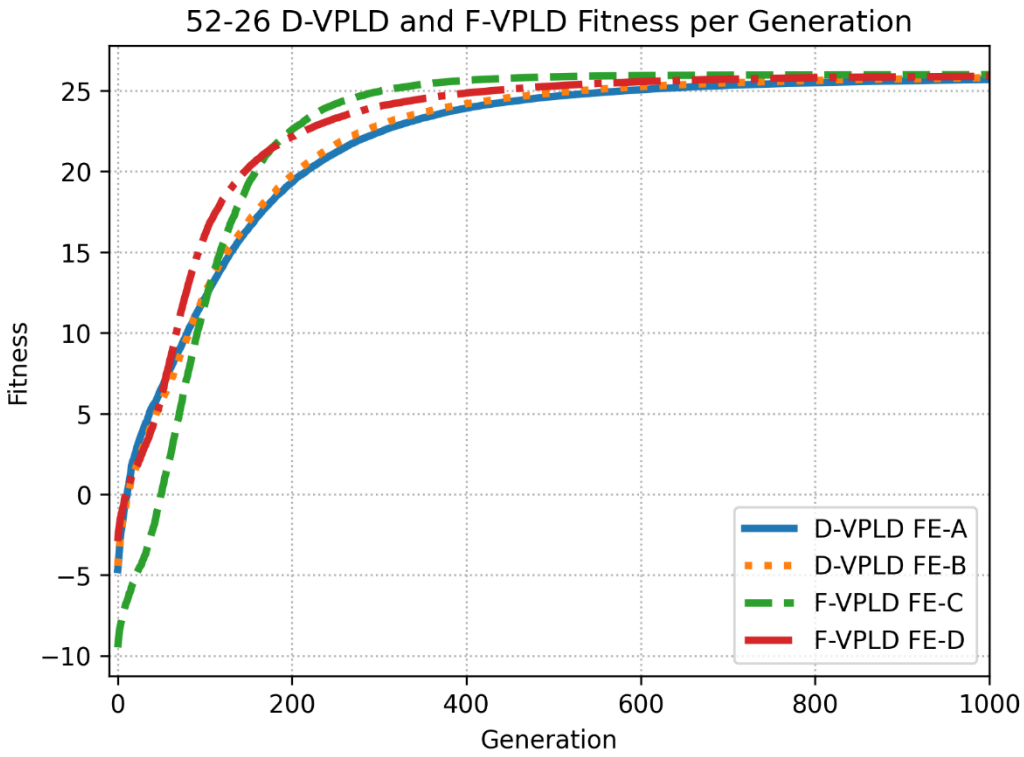

Fig. 6-13 Character Recognition best fitness per generation. 52-26 D-VPLD and F-VPLD comparison using FE-A , FE-B, FE-C and FE-D

Looking at the 52-78 architectures for the D-VPLD and F-VPLD, these architectures are faster to evolve than the 52-26 architecture. The 52-78 D-VPLD uses the mean output group function. Both results using the mean output group function are quick to evolve, however they require twice the number of generations of the F-VPLD results using the mean method (Table 6-25). This is also seen in the computation time required to evolve (Table 6-26). Specifically looking at the F-VPLD results, the max method is significantly slower than the mean method. The average best fitness per generation of all the 52-78 architectures evaluated is shown in Fig. 6-14, the D-VPLD and F-VPLD results using the mean method have a comparable initial rate of evolution but the D-VPLD requires more generations to reach a solution that can recognise all the characters.

The most optimal solution for the D-VPLD and F-VPLD is the 52-78 architecture using the mean output group method, specifically for the D-VPLD this is the architecture using FE-B, for the F-VPLD it is the architecture using FE-C; these architectures are used in the main chapters for the comparison with the ANN (section 5.1).

Table 6-25 Character Recognition generations to evolve, 52-78 D-VPLD and F-VPLD FE comparison

System	Mean	Median	CV
D-VPLD Mean FE-A	1176	826	0.77
D-VPLD Mean FE-B	930.3	780	0.55
F-VPLD Mean FE-C	417.1	392	0.34
F-VPLD Mean FE-D	512.73	479	0.34
F-VPLD Max FE-C	1071.72	704.5	0.96
F-VPLD Max FE-D	3204.82	2333.5	0.82

Table 6-26 Character Recognition time to evolve, 52-78 D-VPLD and F-VPLD FE comparison

System	Mean[s]	Median[s]	CV
D-VPLD Mean FE-A	62.17	44.78	0.76
D-VPLD Mean FE-B	49.45	41.56	0.54
F-VPLD Mean FE-C	21.02	19.85	0.33
F-VPLD Mean FE-D	25.26	23.88	0.33
F-VPLD Max FE-C	45.92	30.47	0.95
F-VPLD Max FE-D	139.75	103.87	0.81

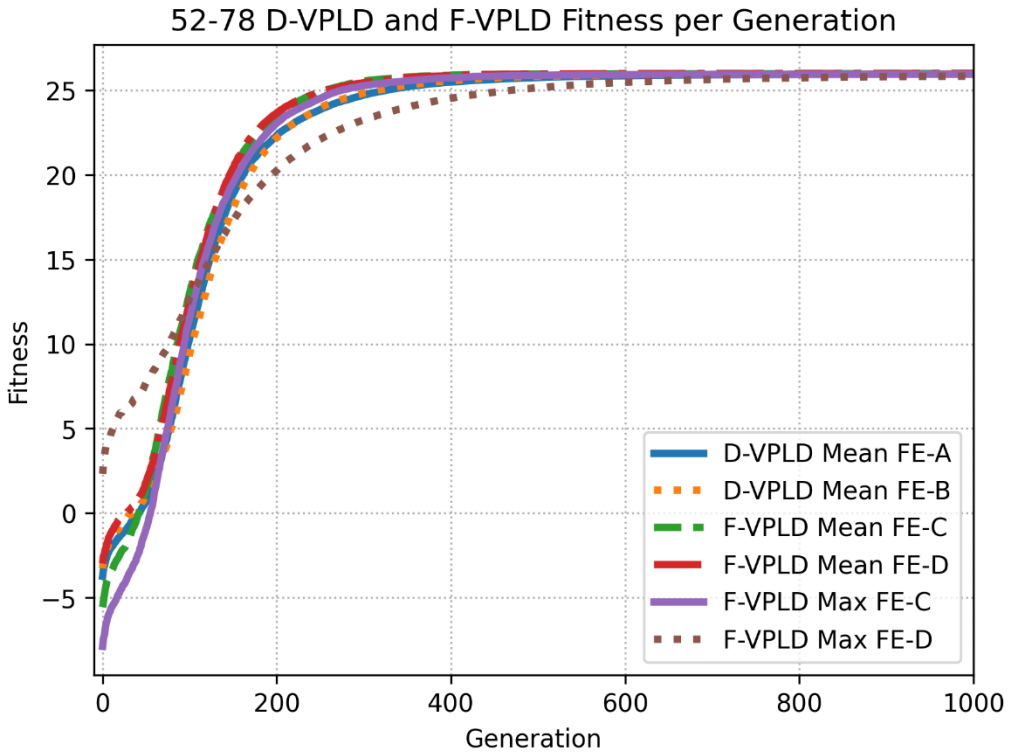

Fig. 6-14 Character Recognition best fitness per generation, 52-78 D-VPLD and F-VPLD comparison using FE-A, FE-B, FE-C and FE-D

6.3.3 Conclusions

For the character recognition problem three hyperparameters were changed between the architectures; 1) varying number of FABs in the output layer for a two-layer architecture; 2) different post-processing operations on the final FABs outputs to get the prediction, and 3) the use of two different FEs using digital logic and two different FEs using floating-point arithmetic.

D-VPLD

For the evolution of the D-VPLD in the character recognition problem, the architecture with more FABs in the output layer (52-78) required significantly less generations to evolve compared to the smaller 52-26 architecture when using the two different function elements (FE-A and FE-B). This is because the smaller architecture has a single binary value output for each of the 26 characters, to get a correct decision only the respective output can be active. When using multiple FABs for the output of each character, a level of activation can be obtained for each character using a post-processing function, this means the correct character only has to have the highest level of activation, the other outputs could have a minor activation and not affect the prediction. Because of the significant difference in the number of generations the smaller D-VPLD required more computation time to evolve.

Of the two D-VPLD FABs implemented the FAB with the larger LUT of functions (FE-B) required less generations and time to evolve. All the D-VPLD architectures have the same classifier performance as they can evolve to recognize all 26 characters A-Z.

F-VPLD

For the F-VPLD, for all six architectures evaluated, the results using FE-C with ReLu are faster to evolve than when using FE-D, requiring less generations to recognize all 26 characters. Like the D-VPLD the 52-78 architecture using a mean output processing function is faster to evolve than the smaller 52-26 architecture. However, the 52-78 architecture using a max output processing function is slower than the smaller F-VPLD. This may be because of the nature of a VPLD and specifically for using binary 1 and 0's as input to the F-VPLD for this problem, this would mean the levels of activation for a given function would remain the same as data is passed through the VPLD, resulting in multiple outputs more likely having the same level of activation. Meaning by having multiple FABs used to determine the activation of an individual character, it increase the probability that multiple character outputs are active, which makes making correct predictions more challenging. Unlike the ANN which uses weights and bias which can change the effect of signals as they pass through the network which is why the ANN performed better with the max output function (section 5.1) than the F-VPLD did. All the F-VPLD architectures have the same classifier performance as they can evolve to recognize all 26 characters A-Z.

Comparing the equivalent D-VPLD and F-VPLD architectures, the F-VPLD is faster to evolve. For the fastest 52-26 architectures the F-VPLD using FE-D is approximately 4x faster than the D-VPLD using FE-B. For the 52-78 architectures using the mean output processing, the F-VPLD using both FE-C and FE-D were faster than both D-VPLD implementations. So as with the 2WD robotic controllers, the VPLD when using FABs with floating-point arithmetic instead of digital logic requires less generations and computation time to evolve.

6.4. Melanoma Classification

This section contains the description and results of the alternative VPLD architectures that are investigated for the melanoma classification problem but found to have lower performance than the architecture described in section 5.2. The description of the proposed method and GA used to evolve the architectures discussed below can be found in sections 5.2.1 and 5.2.5 respectively.

6.4.1 D-VPLD Classifier Architectures

Three two-layer architectures (Fig. 6-15) with variations in the number of FABs in the output layer and the function elements are investigated (Table 6-27).

Table 6-27 D-VPLD architectures evaluated for melanoma classification

System	Configurations Tested

D-VPLD	• 16-1
	  ○ with FE LUT A
	  ○ with FE LUT B
	• 16-3
	  ○ with FE LUT A and mean output grouping
	  ○ with FE LUT B and mean output grouping
	• 16-9
	  ○ with FE LUT A and mean output grouping
	  ○ with FE LUT B and mean output grouping

For the 16-3 architecture, the three FABs are summed to determine the level of activation for a given input (Fig. 6-17). For the 16-9 architecture the output is split into groups of three, each group of three bits is summed, then the average activation of the groups is used as the final output for classification which is a floating-point value (Fig. 6-16).

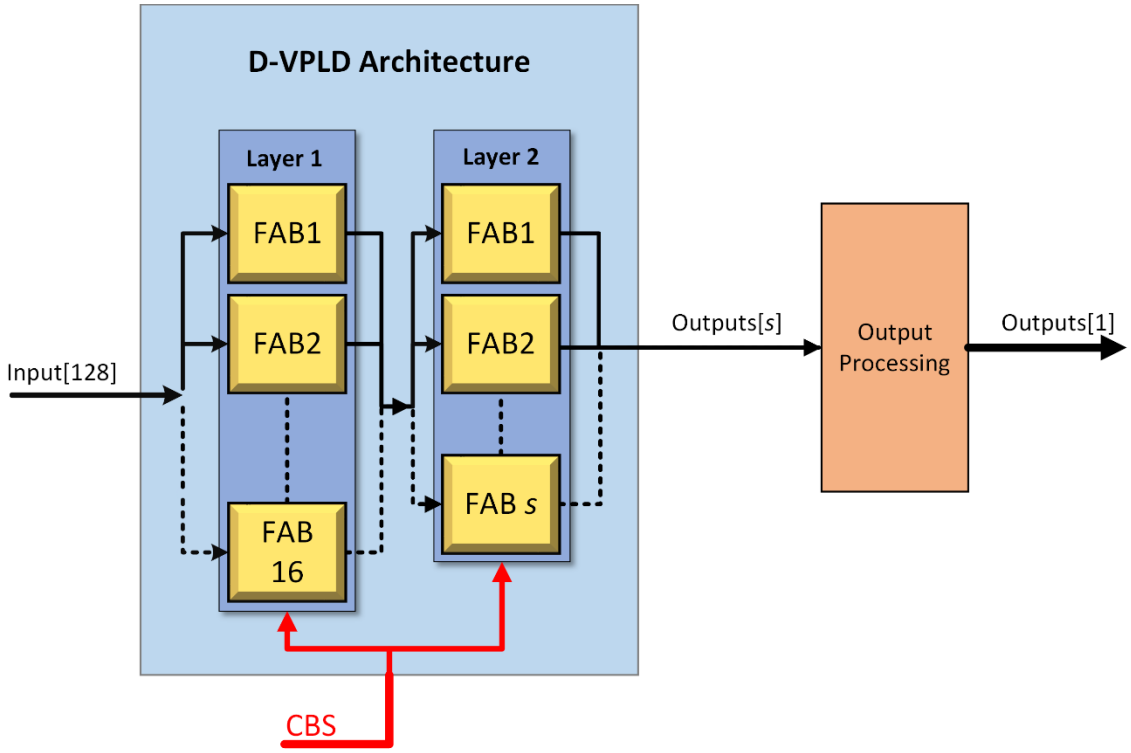

Fig. 6-15 D-VPLD architecture for binary classification of melanomas. Two-layer architecture with 16 FABs in the first layer and s FABs in the second layer. $s = 1, 3$ *or* 9

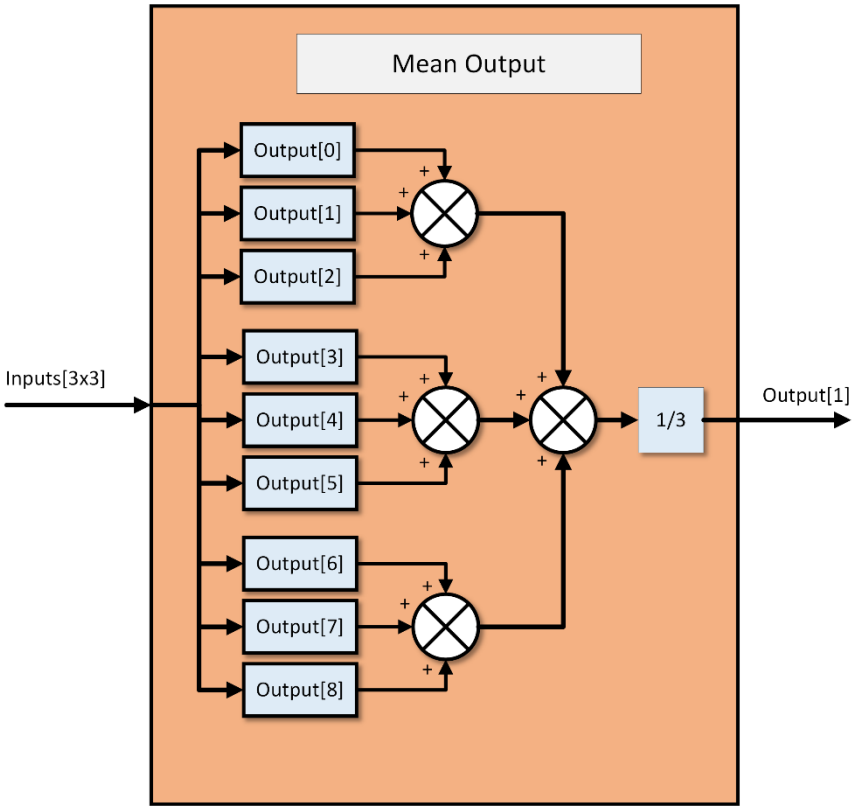

Fig. 6-16 Mean method used by the D-VPLD architecture to group 9 outputs into one output used to determine the classification of a skin lesion

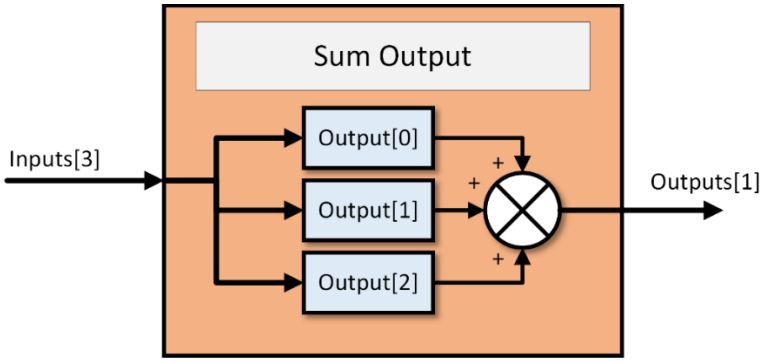

Fig. 6-17 Sum method used by the D-VPLD architecture to group 3 outputs into one output used to determine the classification of a skin lesion

The D-VPLD inputs are the 16 extracted ABCD and GLCM features described in section 5.2. The inputs are converted into a binary form suitable for the D-VPLD by quantizing them into 8 bits, and forming a binary array. As stated, after some post-processing the output for each architecture is a single value used for the classification of a given skin lesion. The output for the 16-3 and 16-9 architectures' ranges from 0-3. The 16-1 architecture output is either 0 or 1.

For both architectures each FAB contains four multiplexers and a single function element. Two different function element LUTs are investigated with these architectures. LUT-A has 21 digital functions and LUT-B has 31 digital functions, both LUT use standard combinational logic, with 'AND', 'OR', 'XOR', and 'NOT' logic operators. These are the same function element LUTs investigated for the Character Recognition problem.

Table 6-28 D-VPLD Melanoma Classification FE LUT-A

No.	Function	No.	Function
0	(A&B)^(C|D)	11	(!A)&B&C&D
1	A&B&C&D	12	A&(!B)&C&D
2	!(A&B&C&D)	13	A&B&(!C)&D
3	A|B|C|D	14	A&B&C&(!D)
4	!(A|B|C|D)	15	(A|B)&(C|D)
5	(A&B)|(C&D)	16	(A|C)&(B|D)
6	(A&C)|(B&D)	17	(A|D)&(C|B)
7	(A&D)|(B&C)	18	(A^B)&(C^D)
8	(A&B)^(C&D)	19	(A^C)&(B^D)
9	(A&C)^(B&D)	20	(A^D)&(C^B)
10	(A&D)^(B&C)		

Table 6-29 D-VPLD Melanoma Classification FE LUT-B

No.	Function	No.	Function
0	(A&B)^(C|D)	16	((!A)&B)|((!C)&D)
1	A&B&C&D	17	(A&(!B))|(C&(!D))
2	!(A&B&C&D)	18	((!A)&(!D))|((!B)&(!C))
3	(!A)&B&C&D	19	(A^B)&(C^D)
4	A&(!B)&C&D	20	(A^C)&(B^D)
5	A&B&(!C)&D	21	(A^D)&(C^B)
6	A&B&C&(!D)	22	((!A)&B)^((!C)&D)
7	A&(!B)&(!C)&(!D)	23	(A&(!B))^(C&(!D))
8	(!A)&B&(!C)&(!D)	24	((!A)&(!D))^((!B)&(!C))
9	(!A)&(!B)&C&(!D)	25	(A|B)&(C|D)
10	(!A)&(!B)&(!C)&D	26	(A|C)&(B|D)
11	A|B|C|D	27	(A|D)&(C|B)
12	!(A|B|C|D)	28	(A^B)&(C^D)
13	(A&B)|(C&D)	29	(A^C)&(B^D)
14	(A&C)|(B&D)	30	(A^D)&(C^B)
15	(A&D)|(B&C)		

6.4.2 D-VPLD Results

Experiments are performed for the three D-VPLD architectures with two different function elements. The evolutionary efficiency and classification accuracy for each architecture is investigated. The different configurations for each architecture are compared to show how/why the optimal architecture discussed in section 5.2 is chosen. The D-VPLD systems are evolved to classify skin lesions from the PH2 database, the lesions are either malignant (cancerous melanoma) or benign (non-cancerous common/atypical nevus). One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs are used to assess the system performance.

Evolutionary Efficiency

When looking at the training accuracy(fitness) per generation, there is a trend in the result, the training accuracy increases with the increase in FABs in the second output layer (Fig. 6-18). The difference in the results between the two different function elements tested is minor. The smaller 16-3 and 16-1 architectures require the least generations and computation time to converge to an optimal solution, but both reach a lower training accuracy compared to the 16-9 architecture (Table 6-30 & Table 6-31).

The 16-3 and 16-1 require less than 50s on average to evolve, whereas both configurations of the 16-9 architecture take approximately 60s to evolve, the extra computation time can be contributed to two factors, 1) the extra generations required to optimize the larger chromosome, and 2) the extra time required to process a larger VPLD in the fitness assessment and during the reproduction stage of the GA.

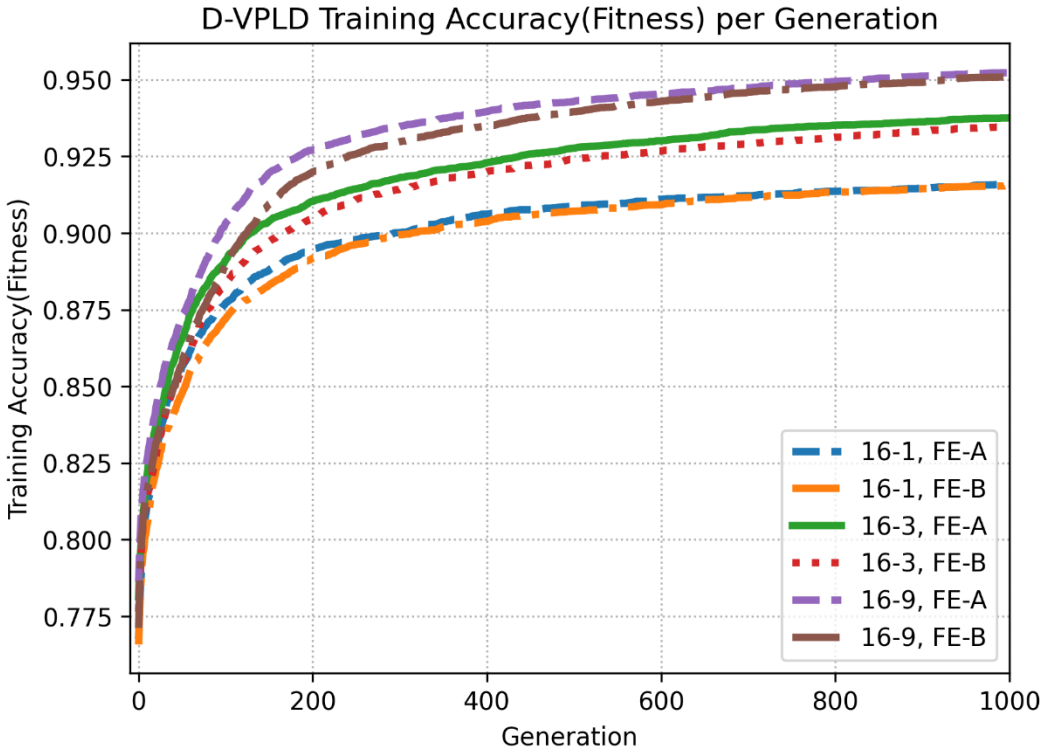

Fig. 6-18 D-VPLD Melanoma Classification. Average best training accuracy(fitness) per generation

Table 6-30 D-VPLD Melanoma Classification Number of Generations to Evolve

Sys	Mean	Median	CV
16-1, FE-A	647.59	682.5	0.37
16-1, FE-B	655.98	673	0.34
16-3, FE-A	716.73	725	0.27
16-3, FE-B	748.25	808.5	0.26
16-9, FE-A	738.11	781.5	0.27
16-9, FE-B	757.16	799.5	0.25

Table 6-31 D-VPLD Melanoma Classification Time to Evolve

Sys	Mean[s]	Median[s]	CV
16-1, FE-A	39.78	41.37	0.37
16-1, FE-B	39.72	40.66	0.34
16-3, FE-A	45.19	46.12	0.27
16-3, FE-B	48.47	51.54	0.25
16-9, FE-A	59.9	63.39	0.26
16-9, FE-B	62.99	65.9	0.24

Classifier Performance

The test accuracy of the architectures evaluated, follow the same trend as the training accuracy results (Fig. 6-19). The two configurations of the 16-1 architecture achieved the lowest average test accuracy across the 100 results, achieving approximately 92% using FE-A, and 90% test accuracy using FE-B (Table 6-33). This indicates when using less FABs in the output layer the VPLD is not able to generalize as well as the other architectures. The 16-3 architecture had comparable test accuracy when using FE-A and FE-B. The architecture with the highest accuracy is the 16-9 using FE-A, achieving average training and test accuracies of 95.3% and 94.4% respectively (Table 6-32 & Table 6-33). For all three architectures the configurations using FE-A, achieve a higher test accuracy compared to the configurations using FE-B. The accuracy level achieved by each architecture is very consistent, shown by the coefficient of variance calculated from the 100 results.

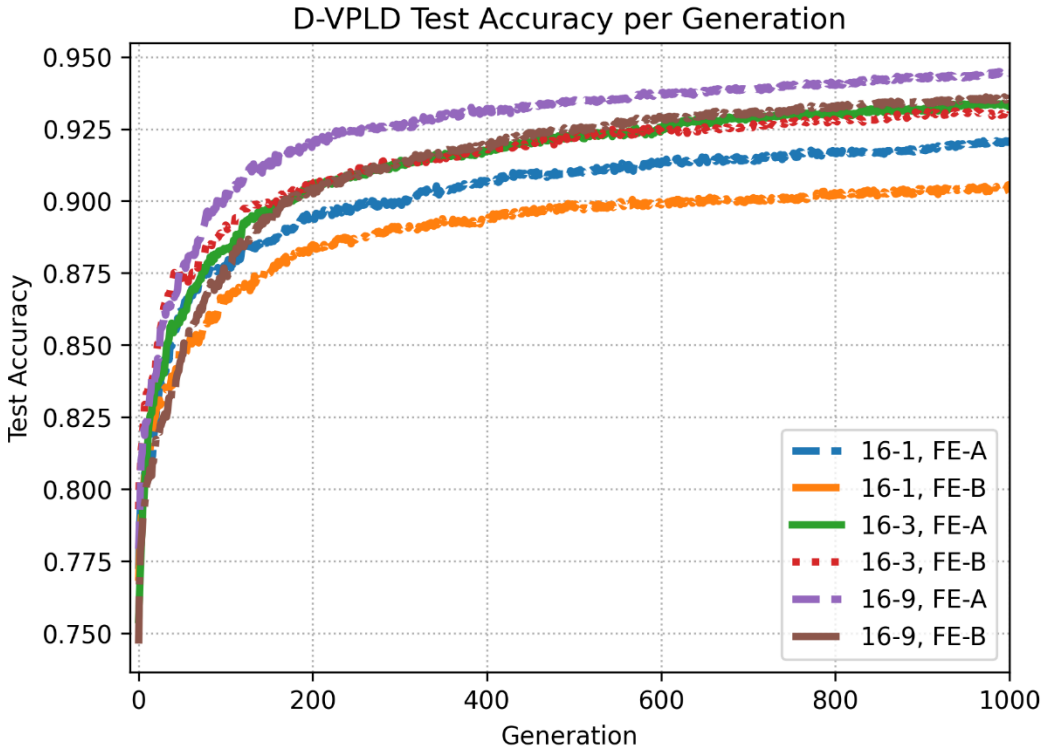

Fig. 6-19 D-VPLD Melanoma Classification, test accuracy per generation

Table 6-32 D-VPLD Training Accuracy (Fitness) for Melanoma Classification problem

Sys	Mean	Median	CV
16-1, FE-A	91.61	91.67	0.01
16-1, FE-B	91.57	91.67	0.01
16-3, FE-A	93.76	93.75	0.01
16-3, FE-B	93.47	93.75	0.01
16-9, FE-A	95.25	95.42	0.01
16-9, FE-B	95.1	95	0.01

Table 6-33 D-VPLD Test Accuracy for Melanoma Classification problem

Sys	Mean	Median	CV
16-1, FE-A	92.08	92.5	0.03
16-1, FE-B	90.42	90	0.04
16-3, FE-A	93.38	93.75	0.02
16-3, FE-B	93.09	93.75	0.03
16-9, FE-A	94.44	95	0.02
16-9, FE-B	93.6	93.75	0.03

The most optimal architecture is chosen based on the accuracy of the evolved classifiers not on the generations or computation time to evolve. This is because classification accuracy is the most important factor for a deployed classifier, particularly when used for medical diagnostics tasks, such as melanoma classification. Therefore, the most optimal architecture evaluated is the 16-9 using FE-A, achieving average training and test accuracies of 95.3% and 94.4% respectively. This architecture is compared with the most optimal F-VPLD and ANN architectures in section 5.2. If the most optimal architecture is chosen, purely on evolutionary efficiency, it would be the 16-1 architecture using FE-B, however this architecture has a lower average training and test accuracy and was able to achieve a high accuracy less consistently than the 16-9 architecture.

6.4.3 F-VPLD Classifier Architectures

For the F-VPLD two architectures are investigated with variations in the FE, and the number of FABs in the output layer (Table 6-34).

Table 6-34 F-VPLD architectures evaluated for melanoma classification

System	Configurations Tested

F-VPLD	• 16-1
	  ○ with FE LUT A
	  ○ with FE LUT B
	• 16-3
	  ○ with FE LUT A and mean output grouping
	  ○ with FE LUT B and mean output grouping

There are two F-VPLD architectures (Fig. 6-20) with variations in the function elements investigated, the first is a 16-1 two-layer architecture, second is a *16-3* two-layer architecture. The 16-3 architecture requires processing of the three FAB outputs to form a single output for classification, this is done by taking the mean of the 3 FABs outputs (Fig. 6-21). The 16-1 architecture requires no output processing. The F-VPLD inputs are the 16 extracted ABCD and GLCM features described in section 5.2.1.

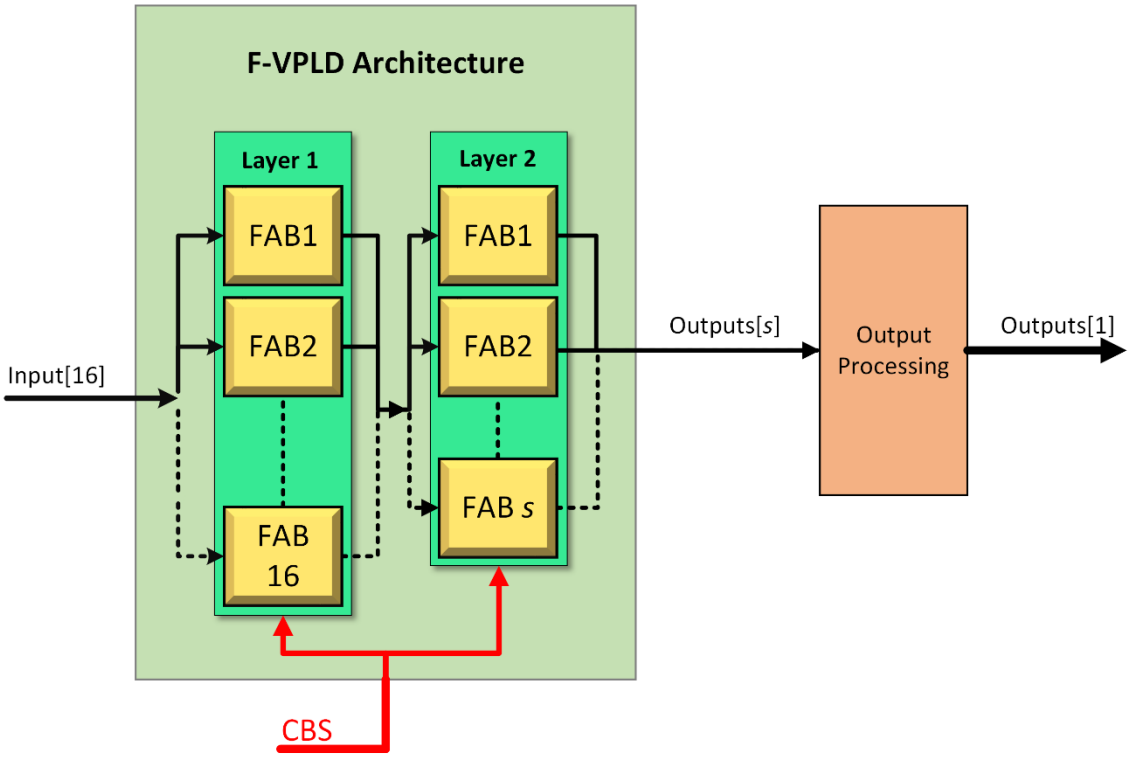

Fig. 6-20 F-VPLD Architectures for binary classification of melanomas. Two-layer architecture with 16 FABs in the first layer and s FABs in the second layer. $s = 1$ *or* 3

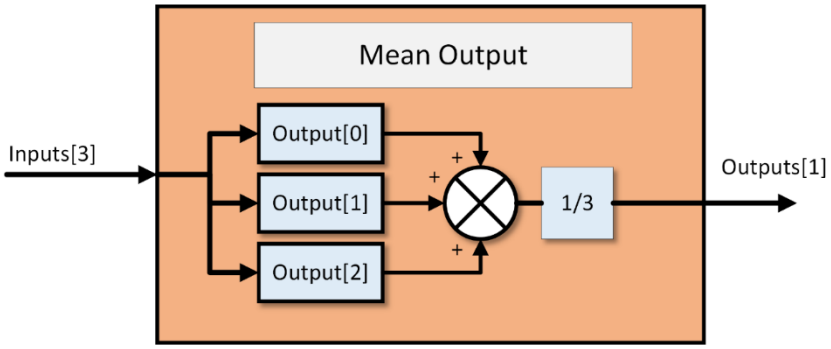

Fig. 6-21 Mean method used by the F-VPLD architecture to group 3 outputs into one output used to determine the classification of a skin lesion

For both architectures each FAB contains four multiplexers and a single function element. Two different function element LUTs are investigated with these architectures. LUT-A (Table 6-35) and LUT-B (Table 6-36) both have 21 digital functions, both LUT use standard mathematical arithmetic, with 'plus', 'minus' and 'multiply' operations. Where the two LUTs differ is in the different activation functions *f* and *g*. Where *f* is the Sigmoid activation function (output ranges from 0 to +1), and *g* is a piece-wise linear activation function (output ranges from -2 to +2).

Table 6-35 F-VPLD Melanoma Classification FE LUT-A

No.	Function	No.	Function
0	*g((A+B)*(C-D))*	11	*g((-A)+B+C+D)*
1	*g(A+B+C+D)*	12	*g(A+(-B)+C+D)*
2	*g(-(A+B+C+D))*	13	*g(A+B+(-C)+D)*
3	*g(A*B*C*D)*	14	*g(A+B+C+(-D))*
4	*g(-(A*B*C*D))*	15	*g(A*B)+(C*D))*
5	*g((A+B)*(C+D))*	16	*g((A*C)+(B*D))*
6	*g((A+C)*(B+D))*	17	*g((A*D)+(B*C))*
7	*g((A+D)*(B+C))*	18	*g((A*B)-(C*D))*
8	*g((A+B)-(C+D))*	19	*g((A*C)-(B*D))*
9	*g((A+C)-(B+D))*	20	*g((A*D)-(B*C))*
10	*g((A+D)-(B+C))*		

Table 6-36 F-VPLD Melanoma Classification FE LUT-B

No.	Function	No.	Function
0	*f((A+B)*(C-D))*	11	*f((-A)+B+C+D)*
1	*f(A+B+C+D)*	12	*f(A+(-B)+C+D)*
2	*f(-(A+B+C+D))*	13	*f(A+B+(-C)+D)*
3	*f(A*B*C*D)*	14	*f(A+B+C+(-D))*
4	*f(-(A*B*C*D))*	15	*f((A*B)+(C*D))*
5	*f((A+B)*(C+D))*	16	*f((A*C)+(B*D))*
6	*f((A+C)*(B+D))*	17	*f((A*D)+(B*C))*
7	*f((A+D)*(B+C))*	18	*f((A*B)-(C*D))*
8	*f((A+B)-(C+D))*	19	*f((A*C)-(B*D))*
9	*f((A+C)-(B+D))*	20	*f((A*D)-(B*C))*
10	*f((A+D)-(B+C))*		

$$g(x) = \begin{cases} 2, & x \geq 2 \\ x, & 2 > x > -2 \\ -2, & x \leq -2 \end{cases} \tag{6.1}$$

6.4.4 F-VPLD Results

The evolutionary efficiency and classification accuracy of the two F-VPLD architectures is investigated. The Experiments for each architecture are performed with two different function elements. The different configurations for each architecture are compared to determine the most optimal architecture for discussion in section 5.2. The F-VPLD is evolved to classify skin lesions from the PH2 database, the lesions are either cancerous melanoma or non-cancerous common/atypical nevus. One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs are used to assess the system performance.

Evolutionary Efficiency

All the F-VPLD architectures can achieve an optimal training accuracy in a short amount of time, the 16-1 results are the fastest to evolve, taking approximately 9-11s on average to evolve (Table 6-38). The 16-3 architectures take longer to evolve, requiring 21-23s on average to evolve. The difference in computation time is caused primarily by the extra generation required to reach an optimal solution, the 16-1 architectures require 626 and 506 generations on average to evolve, whereas the 16-3 architectures require 708 and 707 generations on average to evolve (Table 6-37). The 16-3 architecture requires more generations due to the output of this architecture relying on 3 FABs to determine the level of activation compared to a single FAB, because of this it is more likely the component of the chromosome relating to the output FABs of a child is “damaged” due to mutation, or genes are inherited that don’t improve accuracy. Although the 16-1 architecture is faster to converge, the 16-3 architecture converges to a higher training accuracy with greater consistency (Table 6-37 & Fig. 6-22). So as with the D-VPLD, with an increase in FABs in the output layer, the training accuracy increases.

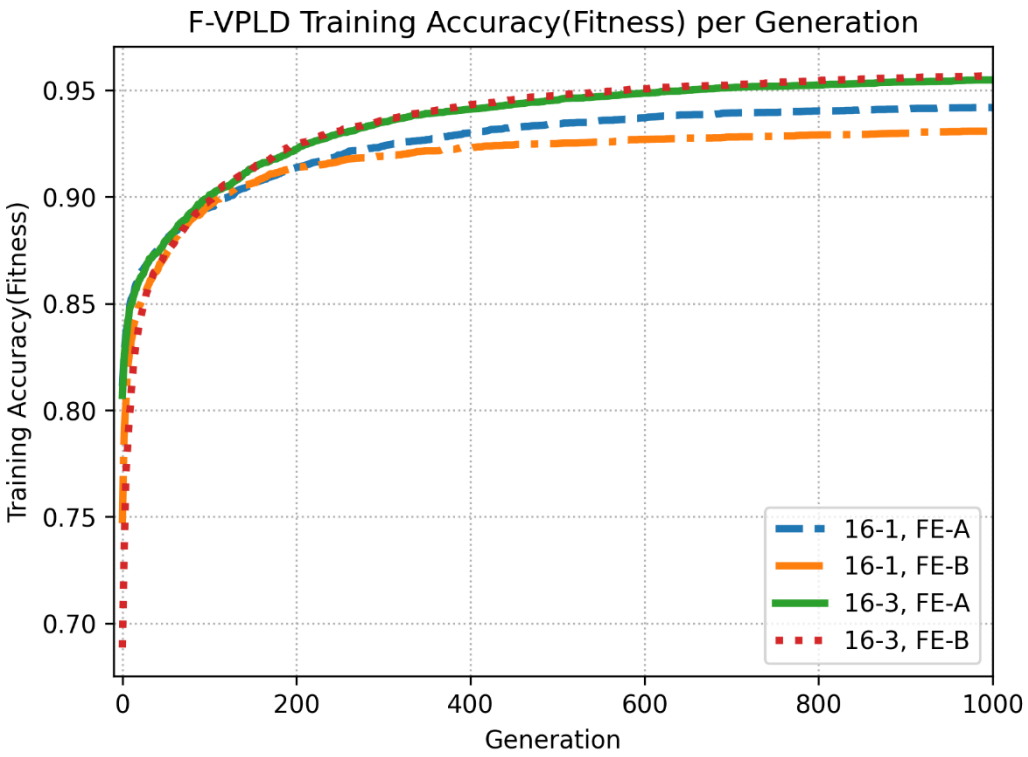

Fig. 6-22 F-VPLD Melanoma Classification. Average best training accuracy(fitness) per generation

Table 6-37 F-VPLD Melanoma Classification Number of Generations to Evolve

Sys	Mean	Median	CV
16-1, FE-A	626.04	627.5	0.34
16-1, FE-B	506.16	482	0.58
16-3, FE-A	708.05	703	0.27
16-3, FE-B	707.44	736	0.27

Table 6-38 F-VPLD Melanoma Classification Time to Evolve

Sys	Mean[s]	Median[s]	CV
16-1, FE-A	10.87	10.84	0.34
16-1, FE-B	8.89	8.52	0.58
16-3, FE-A	18.64	18.9	0.27
16-3, FE-B	18.57	19.26	0.27

Classifier Performance

When analysing the test accuracy per generation (Fig. 6-23), the 16-3 architecture using FE-B reaches the highest test accuracy on average, 95.7%. It can also be seen that both architectures when using FE-B had the fastest initial rate of improvement in test accuracy. Again, as with the D-VPLD, the F-VPLD test accuracy does follow the same trend as the training accuracy results, the two configurations of the 16-3 architecture reach a higher accuracy than the two configurations of the 16-1 architecture. For both the 16-3, and 16-1 architectures a higher test accuracy is achieved on average when using FE-B, however the difference in accuracy is just over 1%. This minimal difference may be because FE-A and FE-B are very similar except for the activation function each used. An interesting observation from the results is that the 16-3 architectures had a minor drop in accuracy between training and test whereas the 16-1 architectures had a more significant drop (Table 6-39 & Table 6-40). Finaly the accuracy level achieved by each architecture is very consistent, shown by the coefficient of variance calculated from the 100 results.

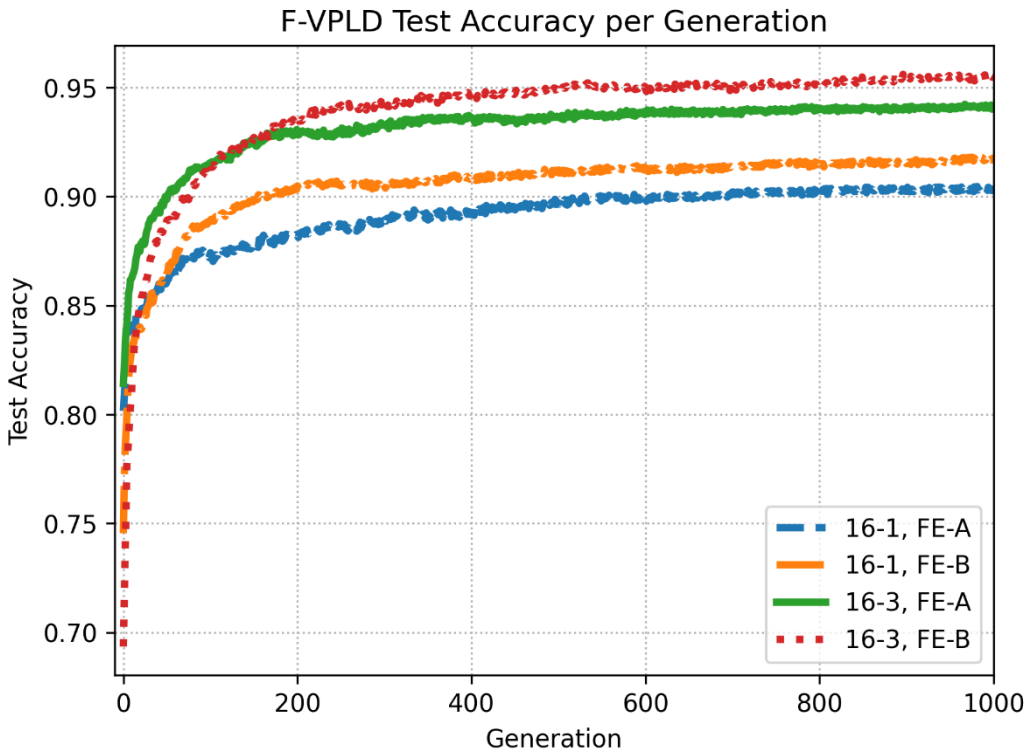

Fig. 6-23 F-VPLD Melanoma Classification, average test accuracy per generation

Table 6-39 F-VPLD Training Accuracy (Fitness) for Melanoma Classification problem

Sys	Mean	Median	CV
16-1, FE-A	94.18	94.17	0.01
16-1, FE-B	93.1	93.33	0.02
16-3, FE-A	95.48	95.62	0.01
16-3, FE-B	95.7	95.83	0.01

Table 6-40 F-VPLD Test Accuracy for Melanoma Classification problem

Sys	Mean	Median	CV
16-1, FE-A	90.38	90	0.03
16-1, FE-B	91.72	91.25	0.03
16-3, FE-A	94.12	93.75	0.02
16-3, FE-B	95.46	96.25	0.01

The most optimal architecture is chosen based on the accuracy of the evolved classifiers not on the generations or computation time to evolve. Therefore, the most optimal architecture evaluated is the 16-3 using FE-B, achieving average training and test accuracies of 95.7% and 95.5% respectively. This architecture is compared with the most optimal D-VPLD and ANN architectures in section 5.2. If the most optimal architecture was chosen, based on the evolutionary efficiency, it would be the 16-1 architecture using FE-B as it required the least generations and time to evolve. The 16-1 architecture however has a lower average training and test accuracy on average, at 93.1% and 91.7% respectively.

6.4.5 Conclusions

For the melanoma classification problem two hyperparameters were changed between the architectures; 1) varying number of FABs in the output layer for a two-layer architecture; and 2) the use of two different FABs using digital logic and two different FABs using floating-point arithmetic.

For the D-VPLD architectures, considering first a changing number of FABs in the output layer, as the number of FABs increases the generations and time to evolve increases. However, so does the average training and test accuracy. The two implementations of the 16-9 architecture with FE-A and FE-B, achieve the highest training and test accuracies. For the difference between the two D-VPLD function elements, FE-A architectures are faster to evolve and achieve higher training/test accuracies to the counterpart architectures using FE-B. This is the opposite to the results in the character recognition problem which uses the same function elements.

For the F-VPLD results, as with the D-VPLD a larger number of output FABs yields better results. Again, the architectures with a larger number of output FABs takes more generations and thus more time to evolve, however they achieve better training and test accuracies on average which is the important factor for such a classification task. There is only a minor difference between FE-A and FE-B for the F-VPLD, with no distinct difference in time to evolve or classification performance.

As seen in the character recognition problem and the 2WD robotic control problem the F-VPLD architecture requires less generations and less time to evolve. The time difference between the F-VPLD and D-VPLD is not just because of the extra generations, but also because the D-VPLD is slower in its execution. The D-VPLD is slower to execute because the 16 features input into the D-VPLD which are originally floating-point data have to be converted to binary strings, and then the D-VPLD output has to be converted back to be used to make a prediction. When considering the training and test accuracies, the top F-VPLD architecture (16-3 using FE-B) compared with the top D-VPLD architecture (16-9 using FE-A), the F-VPLD architecture achieves higher training and test accuracies on average, but the difference is marginal (less than 1%).

6.5. Conclusion

In this chapter the effect of the hyperparameters on the ability of the VPLD to accurately control robots or classifier data correctly was investigated. Both the evolutionary efficiency and the controller/classifier performance were considered. For each of the robotic control and pattern recognition problems investigated, a set of experiments were conducted to evaluate multiple architectures. These experiments were used to find the optimal architecture to use in the comparison between the VPLD and EHW, and the VPLD and ANN. However, they were also used as the initial investigation into the effect of the VPLD hyperparameters. For each problem different hyperparameters were tested, but collectively the effects of altering, number of layers, number of FABs and types of FAB was investigated.

Number of layers

In the hexapod and 2WD robot control problems, experiments were conducted with architectures with different numbers of layers. In the hexapod problem, the architecture with more layers took longer to evolve, requiring more generations and computation time to converge to an optimal solution. For the 2WD robot control problem D-VPLD and F-VPLD architectures with two, three and four layers were evaluated. As in the hexapod problem more layers resulted in more generations and more time to evolve. For both robotic control problems naturally more generations requires more time to evolve, but also the length of time per generation to execute the GA is longer for bigger architectures due to increased computation time to perform crossover operations and for executing the VPLDs during the fitness assessment.

In the hexapod control problem, all architectures are evolved to the same fitness level. In the more complex 2WD robot autonomous navigation task, the larger three and four layer architectures achieved a lower fitness level on average, compared to the smaller two layer VPLDs.

The decreased fitness and slow evolution of the multi-layer architectures could be contributed to the multi-point crossover being overly destructive to the multilayer architectures, chains of genes passing information between layers could be broken during the multi-point crossover. In future work alternative crossover operations could be investigated.

Number of FABs

In both robotic control problems, two-layer architectures with varying numbers of FABs in the first (input) layer were evaluated. For the hexapod control, as the number of FABs in the input layer increased the time to evolve increased, but the number of generations decreased. This is because for the smaller VPLD architecture it is faster to perform the crossover operation during reproduction, and also minor difference in propagation delay, although almost negligible accumulates from the thousands of times the VPLD is executed during the fitness assessment. In the 2WD autonomous navigation task, for both the D-VPLD and F-VPLD when increasing the number FABs in the first layer, the number of generations to evolve decreased as in the hexapod problem. For the fitness the D-VPLD and F-VPLD architectures achieved in the combined obstacle avoidance and light following task, the fitness level achieved on average increased with an increase in the number of FABs in the first layer.

In the two pattern recognition problems, D-VPLD and F-VPLD two-layer architectures with varying FABs in the final output layer were evaluated. For both VPLDs in the character recognition problem the larger 52-78 architecture with the mean output processing function are faster to evolve requiring less generations to evolve than the 52-26 architectures. All the architectures evolve to correctly predict all 26 characters. For the melanoma classification, the opposite is seen, the D-VPLD and F-VPLD architectures with more FABs in the output layer require more generations to evolve. However, they achieve better training and test accuracies on average which is the important factor for such a classification task.

Types of FABs

Function array blocks using both digital logic and floating-point arithmetic were evaluated in the 2WD robotic control problem, and both pattern recognition problems. For the 2WD robot autonomous navigation the F-VPLD architectures require less generations and time to evolve and reach higher fitness levels on average. For the number of generations and the fitness level, the increased performance of the F-VPLD may be because the mathematical functions and floating-point data have a higher fidelity compared to digital logic, and this higher fidelity makes solving more complex and dynamic control tasks easier. For the difference in the time to evolve, this is contributed

to in part the increased generations, but also the extra computation time to generate the binary data required by the D-VPLD, execute the binary functions, and then convert the data back to floating point to control the robot.

Similarly, for both the pattern recognition problems, the F-VPLD is faster to evolve requiring less time and generations. Again, this is contributed to both the added generations and the added time to execute the D-VPLD versus a F-VPLD. For the character recognition, all D-VPLD and F-VPLD architectures can evolve to recognize the 26 characters. When considering the training and test accuracies for the melanoma classification, the top F-VPLD architecture (16-3 using FE-B) compared with the top D-VPLD architecture (16-9 using FE-A), the F-VPLD architecture achieves higher training and test accuracies on average, but the difference is marginal (less than 1%).

In future work the effect of the different hyperparameters on the speed of optimization and on the performance of the resulting robotic controllers or pattern recognition classifiers should be investigated further. Also, methods for the optimization of said hyperparameters should be investigated.

Chapter 7

Chapter 7: Conclusions and Future Research

This thesis has added to the current body of knowledge in the machine learning domain in the fields of evolutionary robotics and pattern recognition. Specifically, this thesis contributes a novel machine learning architecture the Virtual Programmable Logic Device, inspired by PLD and EHW architectures. The VPLD is based on the architecture of a PLD which uses programmable function elements and routing between the function elements. However instead of using hardwired digital circuits, the VPLD is the software equivalent of the PLD written in a standard programming language that can be executed on a CPU. The configuration of the function elements and routing can be optimised using machine learning algorithms for application in fields such as robotics and pattern recognition.

First a digital VPLD architecture is implemented and validated against an equivalent EHW for the gait control of a hexapod, where the configuration bitstreams for the VPLD and EHW are evolved using a GA for the optimal control of a hexapod. In this validation a number of advantages of the VPLD over the EHW architecture are demonstrated; 1) a faster evolutionary time (175x faster to evolve), 2) reduced system complexity , and 3) reduced cost; the VPLD was implemented on a US$35 low cost raspberry pi, compared to the EHW running on the US$500 DE-10 FPGA board. As previously mentioned, the VPLD has other advantages including: a) a reduced time to develop and compile designs; b) increased simulation complexity; c) the use of floating-point variables and mathematical functions; and d) a large range of processor devices.

After validation of the VPLD, two types of VPLD, the D-VPLD and F-VPLD are benchmarked against the ANN in two application areas, 1) evolutionary robotics, specifically for the gait control of a hexapod robot and for autonomous navigation tasks of a two-wheel drive mobile robot; and 2) pattern recognition, for a binary character recognition task, and a melanoma detection task.

For the robotic control tasks the D-VPLD, F-VPLD and ANN reached a very similar controller performance, however the D-VPLD had a worse evolutionary efficiency compared to the F-VPLD and ANN.

In the character recognition experiment all three architectures had an excellent classifier performance, with all evolved controllers able to recognise the complete alphabet A to Z. For evolutionary efficiency, the F-VPLD and ANN required approximately the same generations to evolve, but the D-VPLD required approximately double the generations for the multi class classification problem. For the binary classification of melanoma all three architectures reached high training and test accuracy in a similar number of generations, however the D-VPLD took approximately three times the computation time the F-VPLD and ANN took to complete the same generations.

Across the four experiments various VPLD architectures were evaluated, observations were made on the effect of the hyperparameters, including the number of layers, number of FABs, and types of FABs. In the robotic control problems, more layers increased the time to evolve and reduced the final levels of fitness the GA converged to. In both the robotic control and pattern recognition domain, more FABs increased the controller and classifier performances. For all the experiments using digital and floating point VPLDs, the F-VPLD was faster to evolve and had better controller and classifier performance.

The following section discuss these points in detail. Also, as the VPLD is a novel architecture it opens up a range of new research areas including for the VPLDs architecture, optimisation and applications. Future research for the VPLD is also discussed below.

7.1. Conclusion

This thesis aimed to address one main question, *can a VPLD be created that could run on a processor instead of reconfigurable hardware, and how does it compare to an ANN when evolved for robotic control and pattern recognition?* To address this, five specific questions were asked:

1. *How can a PLD be virtualised in software?*
2. *How do VPLDs and EHW compare when evolved for robotic control?*
3. *How do VPLDs and ANNs compare when evolved for robotic control?*
4. *How do VPLDs and ANNs compare when evolved for pattern recognition?*
5. *What are the important hyperparameters for the VPLD?*

To answer these questions a VPLD architecture was designed and implemented. Then experiments on this VPLD architecture were conducted, using the results from this experimentation the final conclusions on the research questions are as follows.

7.1.1 How can a PLD be virtualised in software?

A VPLD can be implemented using the functional programming design paradigm in a standard programming language such as Python as is done in this thesis. The VPLD is a feed forward architecture which can be implemented very simply in comparison to the equivalent EHW. When using functional programming only three main functions are required, 1) feed-forward VPLD, 2) feed-forward layer, 3) feed-forward FAB. These functions are configured by the configuration bitstream to alter the function of the VPLD which can be used for robotic control and pattern recognition problems. For optimisation, the VPLD architecture is non-differentiable, therefore non-gradient based optimisation algorithms are required for the configuration bitstream optimization. Evolutionary algorithms such as the genetic algorithm can be used to evolve the VPLD CBS for various applications. Using a VPLD means the complete evolution environment which includes the VPLD itself, a genetic algorithm and a robotic simulation can be in a single programming environment, and this evolutionary environment can run on any system with a CPU.

7.1.2 How do VPLDs and EHW compare when evolved for robotic control?

A VPLD using digital logic was validated against the equivalent EHW architecture for the gait control of a hexapod robot. The VPLD was implemented on both a PC and a Raspberry Pi whereas the EHW was implemented on a Cyclone-5 with a SoC processor creating an environment that is significantly more complex, and has components spread across the FPGA fabric, embedded processors and a desktop PC. This makes the EHW implementation more difficult to debug and refactor in comparison to the VPLDs implementation.

As both the VPLD and EHW architectures are the same, but one is implemented in software and the other hardware, it was expected when evolved using the same GA, that the number of generations to evolve and the evolved controller performance would be the same. In the results it was observed that this was the case, the VPLD and EHW evolved in a very similar number of generations and reached the same level of controller performance. Where the VPLD improved over the EHW, was in the speed of evolution, due to the VPLD, GA, and robotic simulation running in the same environment. The VPLD when evolved on a PC was 175x faster than the EHW evolved on the Cyclone 5 development board.

The EHW is evolved on embedded ARM processors in the FPGA SoC which would be expected to be slower than a desktop PC, however it was shown the alternative of having the GA and simulation running on a PC and a serial link to the FPGA device would not provide any benefit to the speed of evolution, as the transfer of trial configurations and I/O for the simulator is a substantial bottleneck. It was also shown the VPLD could be evolved on a $35USD Raspberry Pi 20x Faster than the EHW, which was implemented on a Cyclone V FPGA boards like the $500USD Terasic DE10 and $225USD Terasic DE10 Nano development boards.

Overall, it is shown the VPLD is a) significantly faster to evolve for robotic control, b) much easier to implement, and c) significantly cheaper. As well the VPLD allows the use of, 1) floating-point variables and mathematical functions, 2) increased simulation complexity, and 3) a large range of processors. Compared to EHW the VPLD will make research on such architectures faster to complete and reduce the cost to entry, as cheap devices such as the Raspberry Pi and low cost off the shelf robotic platforms can be used, opposed to expensive custom hardware.

7.1.3 How do VPLDs and ANNs compare when evolved for robotic control?

The ANN is often used as a benchmark to other machine learning architectures, including EHW from which the VPLD was inspired. Because of this the VPLD was compared with the ANN for robotic control. A D-VPLD was first compared with the ANN for the hexapod gait problem, where the D-VPLD and ANN evolved the same level of performance, however the ANN took almost twice the generations to evolve. Even though the ANN took twice the generations both the D-VPLD and ANN evolve in approximately the same time. This is because a D-VPLD uses pseudo bits, so the input data from the sensors commonly used in robotics has to be converted from floating-point data to binary strings, then the output has to be converted from binary back to floating point values required to drive the robot. The data conversion process slows down the fitness assessment of the D-VPLD as the device must be executed 1000s of times during this stage of the GA process.

Next a D-VPLD, F-VPLD and an ANN were evolved for the autonomous navigation of a mobile robot. In the primary task of combined obstacle avoidance and light following both VPLDs reached a higher average fitness compared to the ANN. In regard to the evolutionary efficiency of the architectures in this problem the F-VPLD required less generations and time to evolve than the ANN. The D-VPLD however required more time and generations to evolve than both the F-VPLD and ANN, but not significantly more.

In the robotic control domain it has been shown the VPLD is a viable alternative to the ANN. The D-VPLD was able to evolve in a similar number of generations and reach the same performance as the ANN. The F-VPLD was shown to be faster to evolve and reach a higher fitness compared to the ANN in the more complex robotic navigation task.

7.1.4 How do VPLDs and ANNs compare when evolved for pattern recognition?

Both the D-VPLD and the F-VPLD were evaluated against the ANN in two pattern recognition problems. The first was a character recognition problem where the three architectures were evolved to recognise the 26 characters of the alphabet. All three architectures were successfully evolved to recognise all the characters. It was found that the F-VPLD and ANN required a similar number of generations and time to evolve, both having a median time to evolve under 20 seconds. The D-VPLD on the other hand required double the generations, however it still found solutions with a median time of approximately 40 seconds. The longer time to find a solution can be contributed to the nature of the binary logic used by the D-VPLD. By using binary values and digital logic,

the outputs are quantised reducing precision. This was further highlighted by the variance of each architecture across the 100 results, the F-VPLD and ANN were more consistent with the time to evolve compared to the D-VPLD, suggesting fewer optimal solutions in the fitness landscape of the D-VPLD. As observed in the robotic control experiments the D-VPLD also requires extra computation time dealing with pseudo bits, opposed to the F-VPLD and ANN which use data types more native to the computer they are running on. The second experiment was on the classification of skin lesions as cancerous melanoma, or non-cancerous typical/atypical nevus. The images from the PH2 database undergo a preprocessing and feature extraction process. The VPLDs and ANN are evolved to perform the binary classification using the extracted features. The D-VPLD, F-VPLD and ANN all achieved test accuracy greater than 96%. For both VPLDs and the ANN, across the 100 results the average training and test accuracies were within 1%. Comparing the evolutionary efficiency of the VPLDs and ANN, all three required a comparable number of generations to converge to an optimal solution. In regard to computation time, the D-VPLD required approximately 3x the computation time the ANN and F-VPLD required to evolve. As seen in the previous problems the D-VPLD requires more computation time because of the extra operations required to deal with pseudo bits. However it has a larger effect in this experiment because the size of the input being processed is significantly larger, with 16 data points converted each to 8-bits.

The VPLDs using both digital logic and floating-point arithmetic were shown to be viable alternatives to the ANN for pattern recognition, as both VPLD architectures achieved the same accuracy as the ANN.

7.1.5 What are the important hyperparameters for the VPLD?

For each of the four problems investigated, various architectures with different hyperparameters were evaluated to find an optimal architecture for each problem. The following observations are made on three key hyperparameters 1) number of layers, 2) number of FABs, and 3) type of FAB (digital or floating-point).

Number of layers: In the hexapod and 2WD robot control problems, experiments were conducted on architectures with different numbers of layers. In both problems the VPLD required more generations and computation time to converge to an optimal solution with an increasing number of layers. In the hexapod control problem, all architectures are evolved to the same fitness level. In the more complex 2WD robot autonomous navigation task, the larger three and four layer architectures achieved a lower fitness level on average, compared to the smaller two layer VPLDs. The decreased fitness and slow evolution of the multi-layer architectures could be contributed to the multi-point crossover being overly destructive to the multilayer architectures, chains of genes passing information between layers could be broken during the multi-point crossover. In future work alternative crossover operations could be investigated.

Number of FABs: In both robotic control problems, two-layer architectures with varying numbers of FABs in the first (input) layer were evaluated. It was observed in both the hexapod control and mobile robot control that with more FABs the number of generations to evolve decreased but the time to evolve increased. This is because for the smaller VPLD architecture it is faster to perform the crossover operation during reproduction, and also minor difference in propagation delay, although almost negligible accumulates from the 1000s of times the VPLD is executed during the fitness assessment. For the fitness the D-VPLD and F-VPLD architectures achieved in the combined obstacle avoidance and light following task, the fitness level achieved on average increased with an increase in the number of FABs in the first layer. In the two pattern recognition problems, D-VPLD and F-VPLD two-layer architectures with varying FABs in the final output layer were evaluated. For both VPLDs in the character recognition problem the larger architectures with more FABs were faster to evolve. All the different architectures evolve to correctly predict all 26 characters. For the melanoma classification, the opposite is seen, the D-VPLD and F-VPLD architectures with more FABs in the output layer require more generations to evolve. However, they achieve better training and test accuracies on average which is the important factor for such a classification task.

Types of FABs: Function array blocks using both digital logic (D-VPLD) and floating-point arithmetic (F-VPLD) were evaluated in the 2WD robotic control problem, and both pattern recognition problems. For the 2WD robot autonomous navigation the F-VPLD architecture requires less generations and time to evolve and reached higher fitness levels on average. For the number of generations and the fitness level, the increased performance of the F-VPLD may be because the mathematical functions and floating-point data have a higher fidelity compared to digital logic, and this higher fidelity makes solving the dynamic control tasks easier. For the difference in the time to evolve, this is contributed to in part the increased generations, but also the extra computation time to generate the binary data required by the D-VPLD, execute the binary functions, and then convert the data back to floating point to control the robot.

Similarly, for both the pattern recognition problems, the F-VPLD is faster to evolve requiring less time and generations. Again, this is contributed to both the added generations and the added time to execute the D-VPLD versus a F-VPLD. For the character recognition, all D-VPLD and F-VPLD architectures can evolve to recognize the 26 characters. When considering the best F-VPLD and D-VPLD architectures training and test accuracies for the melanoma classification, the F-VPLD achieves higher training and test accuracies on average, but the difference is marginal (less than 1%).

To summarize, across the four experiments various VPLD architectures were evaluated, observations were made on the effect of the hyperparameters, including the number of layers, number of FABs, and types of FABs. In the robotic control problems, more layers increased the time to evolve and reduced the final levels of fitness the GA converged to. In both the robotic control and pattern recognition domain, more FABs increased the controller and classifier performances. For all the experiments using digital and floating point VPLDs, the F-VPLD was faster to evolve and had better controller and classifier performance. In future work the effect of the different hyperparameters on the speed of optimization and on the performance of the resulting robotic controllers or pattern recognition classifiers should be investigated further. Particularly methods for the optimization of said hyperparameters should be investigated.

7.1.6 Main Question: *Can a VPLD be created that could run on a processor instead of reconfigurable hardware, and how does it compare to an ANN when evolved for robotic control and pattern recognition?*

This thesis has demonstrated that a VPLD can be implemented on a CPU, and in the experiments conducted, the VPLD has comparable performance to ANN for robotic control and pattern recognition.

Firstly, regarding the VPLDs implementation. The VPLD implementation is significantly simpler than the implementation of EHW, and the VPLD costs less to implement because expensive FPGA devices and custom hardware platforms are not required. The speed of evolution of the VPLD running on a PC was two orders of magnitude faster than the EHW running on the Cyclone V FPGA SoC. It was also shown in this thesis that complex function array blocks using floating-point arithmetic opposed to digital logic could be easily implemented in software. This type of architecture can be implemented in hardware but is complex and requires FPGAs with higher logic element counts, increasing the cost of the FPGA device.

Regarding the VPLDs performance in comparison to ANNs. In the robotic control domain, it has been shown the VPLD is a viable alternative to the ANN. The VPLD using both digital logic and floating-point arithmetic was able at the minimum to evolve in approximately the same number of generations and reach the same performance as the ANN, in some cases the VPLD surpassed the ANN both in the required generations and resulting performance. The F-VPLD for example was shown to be faster to evolve and reach a higher fitness compared to the ANN in the 2WD combined obstacle avoidance and light following, which is the most complex robotic control task evaluated in this thesis. This shows the benefit of using such an architecture compared to previous digital EHW. In previous literature that compared digital EHW and a ANN evolved for a similar autonomous navigation task[10], they showed the ANN required more generations but achieved a higher controller performance.

In the pattern recognition domain, the VPLD was also shown to be a viable alternative to the ANN. The VPLD using both digital logic and floating-point arithmetic achieved the same accuracy as the ANN. In the melanoma classification the highest accuracy achieved by both VPLDs and the ANN is very similar at approximately 98%. This level of accuracy is comparable to feed forward neural networks trained using the same dataset[85] and gradient based optimisation. The level of accuracy achieved in this

research by the VPLD and ANN is better than previous implementation of ANNs evolved using GAs for skin lesion classification[52, 54]. In regard to evolutionary efficiency in the melanoma classification, both VPLDs required less generations to evolve than the ANN. However, for computation time the D-VPLD requires more than the ANN. The F-VPLD and ANN computation time for the evolution process was approximately the same.

To finish, the VPLD compared to the previous EHW will make research in this domain faster to complete, because it is significantly easier to implement and the time to optimise the configuration is reduced. The cost to entry is reduced as cheap devices such as the Raspberry Pi and low cost off the shelf robotic platforms can be used, opposed to expensive custom hardware. The VPLD can be used as a viable alternative to the ANN in both evolutionary robotics and pattern recognition applications as it was shown the VPLD can achieve the same performance as ANNs and in some cases better performance.

7.2. Future Research

A new architecture such as the VPLD gives rise to many research opportunities, including five main facets; 1) VPLD architectures, including alternative FABs or recurrent architectures with feedback could be investigated; 2) analysis of hyper-parameters, detailed investigation into optimisation of the hyper-parameters; 3) analysis of optimisation algorithms, what effect do alternative nature inspired optimisation algorithms have on the time to optimises VPLD configurations, and on the resulting performance; 4) multiprocessing or GPU acceleration, can the VPLD be implemented to run on the GPU in the same environment as high performance simulators such as Nvidia's Isaac sim; and 5) applications of the VPLD such as online fault recovery for remote autonomous vehicles.

7.2.1 VPLD Architecture

How can a recurrent VPLD architecture be implemented, and what effect would it have on performance?

A recurrent VPLD architecture with pseudo memory could enhance the VPLD performance for certain applications such as speech recognition and natural language processing. Neural networks with memory have shown to do better at such tasks in comparison to feedforward artificial neural networks. A recurrent VPLD could also be investigated for applications in robotics, such as path planning and manipulation where sequences of actions need to be planned and executed to achieve the final desired results, i.e. the robot moves to the correct position, or the part is manipulated into the correct orientation.

Can alternative function array block architectures improve the VPLD performance?

Alternative function array block architectures could be investigated to see if this could reduce the time to evolve or improve the performance of the resulting VPLD controllers and classifiers.

7.2.2 VPLD Hyper-parameter optimisation

How can the VPLD hyperparameters be optimised to increase performance?

An investigation could be performed on how different methods such as grid search, random search, Bayesian optimisation, or evolutionary algorithms can be used to optimise the VPLD hyperparameters, namely 1) number of layers, 2) number of FABs in each layer, 3) type of FAB (digital, floating-point), and 4) FAB function lookup table. A method that can optimise these parameters will remove the need for manual tuning. Through this investigation a detailed analysis of the VPLD hyperparameters can be made to see if the same observation from this thesis are replicated.

7.2.3 VPLD Configuration bitstream optimisation

How do different evolutionary algorithms compare for the optimization of the VPLD configuration bitstream?

Evolutionary algorithms such as evolutionary strategies, differential evolution and particle swarm optimization should be compared with the genetic algorithm for evolving the VPLD for applications such as robotics and pattern recognition. This should be investigated because it is possible that these algorithms may have general or application specific benefits over the GA for the evolution of the VPLD CBS.

Can methods other than evolutionary algorithms be used for the optimisation of the VPLD configuration bitstream?

Could other optimisation methods such as Bayesian optimisation or simulated annealing be used to optimise the VPLD configuration bitstream. If non evolutionary methods can be used successfully, how do they compare to the evolutionary methods, both in computation time and level of performance of the resulting VPLD.

7.2.4 VPLD Multiprocessing

Can the VPLD be implemented using multiprocessing and would this reduce the time to evolve?

The use of parallel computing and multithreading could be investigated to speed up the execution of the VPLD. This may reduce the computation time to evolve the VPLD configuration bitstream. The VPLD is executed thousands of times during the fitness assessment stage of the GA, so any reduction in propagation delay will accumulate to a significant time reduction in time to evolve. Also, a reduced propagation delay will aid robotics application where significant DSP is required such as Kalman filters performing sensor fusion. A short propagation delay gives more time for these DSP processes to be executed in the control period.

Can the VPLD and a evolutionary algorithm be implemented on a GPU with Nvidia's Isaac Sim to further reduce the time to optimise the VPLD in complex robotics applications?

It should be investigated if the VPLD could be implemented and evolved directly within the GPU environment. Some modern robotics simulators such as Nvidia's photo-realistic Isaac sim run directly on the GPU, and for reinforcement learning, the process is slowed by transferring data between the GPU and CPU. Therefore, as is done now with neural networks, could the VPLD be implemented and trained/evolved on the GPU in the same environment as the photo-realistic simulation.

7.2.5 VPLD Applications

Can the fault adaption process be implemented on a robotic simulation, which is dynamically adjusted to the fault condition before final testing on the real robot, allowing the robot to adapt without causing further damage?

Current online fault adaption strategies detect that the robot has a fault from the robot's behavior and perform learning-based adaption strategies on the robot until the behavior is corrected[87-89]. These methods do not directly identify the fault and during the trialing period can cause further damage to the robot. An alternative safety-first strategy using the VPLD could be developed that uses sensory feedback to identify the actual fault. The fault information can then be passed to a simulation within the robot allowing the VPLD adaption process to be implemented in simulation before final testing on the real robot reducing the chance of damage. This strategy will use self-modeling techniques to dynamically update the simulation to accurately represent the real-world state of the robot. This thesis showed the VPLD could be evolved from scratch relatively quickly on the raspberry pi for the control of a hexapod. The VPLD could be evolved online in the robot with a simulation modified for the identified faults, allowing the robot to safely recover from the unforeseen damage.

Can the VPLD be used for segmentation of medical images for use in diagnosis tools?

It could be investigated if a VPLD architecture inspired by ANN architectures such as U-Net could be used for the segmentation of medical images. For example, in the melanoma classification task studied in this thesis, a VPLD inspired by U-Net could be used to segment the skin lesion instead of using manual image processing techniques. Architectures such as U-net have been shown to solve complex segmentation tasks very effectively, and using the VPLD in this domain further expands its range of applications.

Can a VPLD be used for natural language processing applications?

If a recurrent VPLD with "memory" is successfully implemented, it could be used for natural language processing tasks such as speech to text and text to speech. For such applications the VPLD could be compared to long short-term memory RNNs. Alternatively, it could be investigated if the VPLD could replace the feed-forward ANN in the transformer model to be used for more complicated NLP applications in generative AI with large language models. This could be text to image, and text to video applications.

References

[1] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 29, no. 1, pp. 87-97, 1999, doi: 10.1109/5326.740672.

[2] P. C. Haddow and A. M. Tyrrell, "Evolvable Hardware Challenges: Past, Present and the Path to a Promising Future," in *Inspired by Nature: Essays Presented to Julian F. Miller on the Occasion of his 60th Birthday*, S. Stepney and A. Adamatzky Eds. Cham: Springer International Publishing, 2018, pp. 3-37.

[3] A. Swarnalatha and A. P. Shanthi, "Complete hardware evolution based SoPC for evolvable hardware," *Applied Soft Computing,* vol. 18, pp. 314-322, 2014/05/01/ 2014, doi: https://doi.org/10.1016/j.asoc.2013.12.014.

[4] M. Okura, H. Matsumoto, A. Ikeda, and K. Murase, "Artifical evolution of FPGA that controls a Miniature Mobile Robot Khepera," in *SICE Annual Conference in Fukui*, Fukui University, Japan, 2003.

[5] K. C. Tan, C. M. Chew, K. K. Tan, L. F. Wang, and Y. J. Chen, "Autonomous robot navigation via intrinsic evolution," in *Evolutionary Computation,CEC '02. Proceedings of the 2002 Congress*, 2002, vol. 2, pp. 1272-1277.

[6] P. C. Haddow and G. Tufte, "An evolvable hardware FPGA for adaptive hardware," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, 2000, vol. 1: IEEE, pp. 553-560.

[7] P. C. Haddow and G. Tufte, "Bridging the genotype-phenotype mapping for digital FPGAs," in *Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001*, 2001: IEEE, pp. 109-115.

[8] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware," in *Evolvable Systems: From Biology to Hardware: 5th International Conference, ICES 2003 Trondheim, Norway, March 17–20, 2003 Proceedings 5*, 2003: Springer, pp. 186-197.

[9] J. Wang, C. H. Piao, and C. H. Lee, "FPGA Implementation of Evolvable Characters Recognizer with Self-adaptive Mutation Rates," in *Adaptive and Natural Computing Algorithms*, Berlin, Heidelberg, B. Beliczynski, A. Dzielinski, M. Iwanowski, and B. Ribeiro, Eds., 2007// 2007: Springer Berlin Heidelberg, pp. 286-295.

[10] M. Beckerleg, J. Matulich, and P. Wong, "A comparison of three evolved controllers used for robotic navigation," *AIMS Electronics and Electrical Engineering,* vol. 4, no. 3, pp. 259-286, 2020, doi: 10.3934/ElectrEng.2020.3.259.

[11] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 4, pp. 267-275.

[12] M. Beckerleg and J. Collins, "Evolving Electronic Circuits for Robotic Control.," presented at the 15th International Conference on Mechatronics and Machine Vision in Practice, Auckland, New Zealand, 2008.

[13] M. Beckerleg and J. Collins, "Using a Hardware Simulation within a Genetic Algorithm to evolve Robotic Controllers," presented at the International Conference on Intelligent Automation and Robotics (ICIAR'11), San Francisco, USA, 2011.

[14] D. Berenson, N. Estevez, and H. Lipson, "Hardware evolution of analog circuits for in-situ robotic fault-recovery," in *2005 NASA/DoD Conference on Evolvable Hardware (EH'05)*, 29 June-1 July 2005 2005, pp. 12-19, doi: 10.1109/EH.2005.30.

[15] O. Garnica, K. Glette, and J. Torresen, "Comparing three online evolvable hardware implementations of a classification system," *Genetic Programming and Evolvable Machines,* vol. 19, no. 1, pp. 211-234, 2018/06/01 2018, doi: 10.1007/s10710-017-9312-1.

[16] R. Yao, Y. Sun, K. He, and Y. Yang, "Online evolution of image filters based on dynamic partial reconfiguration of FPGA," in *2015 11th International Conference on Natural Computation (ICNC)*, 2015: IEEE, pp. 999-1005.

[17] P. Teerakittikul, G. Tempesti, and A. M. Tyrrell, "The application of evolvable hardware to fault tolerant robot control," in *2009 IEEE Workshop on Evolvable and Adaptive Hardware*, 30 April-2 March 2009 2009, pp. 1-8, doi: 10.1109/WEAH.2009.4925661.

[18] J. M. Hereford, "Fault-tolerant sensor systems using evolvable hardware," *IEEE Transactions on Instrumentation and Measurement,* vol. 55, no. 3, pp. 846-853, 2006, doi: 10.1109/TIM.2006.873791.

[19] C. Darwin, *On the origin of species by means of natural selection*. London: John Murray, 1859.

[20] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," ed, 2016.

[21] T. Mendonça, P. M. Ferreira, J. S. Marques, A. R. S. Marcal, and J. Rozeira, "PH2 - A dermoscopic image database for research and benchmarking," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 3-7 July 2013 2013, pp. 5437-5440, doi: 10.1109/EMBC.2013.6610779.

[22] M. Beckerleg and J. Collins, "Evolving Electronic Circuits For Robotic Control," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2-4 Dec. 2008 2008, pp. 651-656, doi: 10.1109/MMVIP.2008.4749607.

[23] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *Computers and Digital Techniques, IEE Proceedings,* vol. 151, no. 4, pp. 267-275, 2004.

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

[25] H. Seok, K. Lee, J. Joung, and B. Zhang, "An On-Line Learning Method for Object-Locating Robots using Genetic Programming on Evolvable Hardware," *International Symposium on Artificial Life and Robotics,* pp. 321--324, 2000. [Online]. Available: citeseer.ist.psu.edu/456254.html.

[26] K. Glette and P. Kaufmann, "Lookup table partial reconfiguration for an evolvable hardware classifier system," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 6-11 July 2014 2014, pp. 1706-1713, doi: 10.1109/CEC.2014.6900503.

[27] J. Wang, B. b. Tang, C. h. Piao, and G. h. Lei, "Statistical method-based evolvable character recognition system," in *2009 IEEE International Symposium on Industrial Electronics*, 5-8 July 2009 2009, pp. 804-808, doi: 10.1109/ISIE.2009.5213594.

[28] P. Rocke, B. McGinley, J. Maher, F. Morgan, and J. Harkin, "Investigating the Suitability of FPAAs for Evolved Hardware Spiking Neural Networks," in *Evolvable Systems: From Biology to Hardware*, Berlin, Heidelberg, G. S. Hornby, L. Sekanina, and P. C. Haddow, Eds., 2008// 2008: Springer Berlin Heidelberg, pp. 118-129.

[29] S. Lawrence, C. L. Giles, T. Ah Chung, and A. D. Back, "Face recognition: a convolutional neural-network approach," *IEEE Transactions on Neural Networks,* vol. 8, no. 1, pp. 98-113, 1997, doi: 10.1109/72.554195.

[30] M. Coşkun, A. Uçar, Y. Ö, and Y. Demir, "Face recognition based on convolutional neural network," in *2017 International Conference on Modern Electrical and Energy Systems (MEES)*, 15-17 Nov. 2017 2017, pp. 376-379, doi: 10.1109/MEES.2017.8248937.

[31] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*: Elsevier, 2024, pp. 269-287.

[32] A. Darwish, A. E. Hassanien, and S. Das, "A survey of swarm and evolutionary computing approaches for deep learning," *Artificial Intelligence Review,* vol. 53, no. 3, pp. 1767-1812, 2020/03/01 2020, doi: 10.1007/s10462-019-09719-2.

[33] T. Shinozaki and S. Watanabe, "Structure discovery of deep neural network based on evolutionary algorithms," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 19-24 April 2015 2015, pp. 4979-4983, doi: 10.1109/ICASSP.2015.7178918.

[34] A. Martín, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "EvoDeep: A new evolutionary approach for automatic Deep Neural Networks parametrisation," *Journal of Parallel and Distributed Computing,* vol. 117, pp. 180-191, 2018/07/01/ 2018, doi: https://doi.org/10.1016/j.jpdc.2017.09.006.

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

[36] D. Harter, "Evolving neurodynamic controllers for autonomous robots," in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, 31 July-4 Aug. 2005 2005, vol. 1, pp. 137-142 vol. 1, doi: 10.1109/ijcnn.2005.1555819.

[37] W. Elmenreich and 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, doi: 10.1109/micai.2007.13.

[38] W. Wahab, "Autonomous mobile robot navigation using a dual artificial neural network," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, 23-26 Jan. 2009 2009, pp. 1-6, doi: 10.1109/tencon.2009.5395892.

[39] P. K. Mohanty, D. R. Parhi, A. K. Jha, and A. Pandey, "Path planning of an autonomous mobile robot using adaptive network based fuzzy controller," in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, 22-23 Feb. 2013 2013, pp. 651-656, doi: 10.1109/IAdCC.2013.6514303.

[40] P. Karlra and 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)*, 8-8 Oct. 2003 2003, vol. 2, pp. 1979-1984 vol.2, doi: 10.1109/ICSMC.2003.1244702.

[41] G. B. Parker and 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), 2003, vol. 2: IEEE, pp. 1376-1381.

[42] J. C. Gallagher, "An evolvable hardware layer for global and local learning of motor control in a hexapod robot," *International Journal on Artificial Intelligence Tools,* vol. 14, no. 06, pp. 999-1017, 2005.

[43] C. F. Juang, Y. C. Chang, and C. M. Hsiao, "Evolving Gaits of a Hexapod Robot by Recurrent Neural Networks With Symbiotic Species-Based Particle Swarm Optimization," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 7, pp. 3110-3119, 2011, doi: 10.1109/TIE.2010.2072892.

[44] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864,* 2017.

[45] G. Brockman *et al.*, "OpenAI Gym," *arXiv:1606.01540,* 2016, doi: https://doi.org/10.48550/arXiv.1606.01540.

[46] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 7-12 Oct. 2012 2012, pp. 5026-5033, doi: 10.1109/IROS.2012.6386109.

[47] S. S. Yadav and S. M. Jadhav, "Deep convolutional neural network based medical image classification for disease diagnosis," *Journal of Big Data,* vol. 6, no. 1, p. 113, 2019/12/17 2019, doi: 10.1186/s40537-019-0276-2.

[48] D. R. Sarvamangala and R. V. Kulkarni, "Convolutional neural networks in medical image understanding: a survey," *Evolutionary Intelligence,* vol. 15, no. 1, pp. 1-22, 2022/03/01 2022, doi: 10.1007/s12065-020-00540-3.

[49] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical Image Analysis using Convolutional Neural Networks: A Review," *Journal of Medical Systems,* vol. 42, no. 11, p. 226, 2018/10/08 2018, doi: 10.1007/s10916-018-1088-1.

[50] X. Li, H. Chen, X. Qi, Q. Dou, C. W. Fu, and P. A. Heng, "H-DenseUNet: Hybrid Densely Connected UNet for Liver and Tumor Segmentation From CT Volumes," *IEEE Transactions on Medical Imaging,* vol. 37, no. 12, pp. 2663-2674, 2018, doi: 10.1109/TMI.2018.2845918.

[51] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "NAS-Unet: Neural Architecture Search for Medical Image Segmentation," *IEEE Access,* vol. 7, pp. 44247-44257, 2019, doi: 10.1109/ACCESS.2019.2908991.

[52] S. Chakraborty *et al.*, "Detection of skin disease using metaheuristic supported artificial neural networks," in *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, 16-18 Aug. 2017 2017, pp. 224-229, doi: 10.1109/IEMECON.2017.8079594.

[53] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-3, no. 6, pp. 610-621, 1973, doi: 10.1109/TSMC.1973.4309314.

[54] R. B. Aswin, J. A. Jaleel, and S. Salim, "Hybrid genetic algorithm — Artificial neural network classifier for skin cancer detection," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 10-11 July 2014 2014, pp. 1304-1309, doi: 10.1109/ICCICCT.2014.6993162.

[55] I. Rechenberg, "Evolutionsstrategie," *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution,* 1973.

[56] H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Springer, 1977.

[57] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – A comprehensive introduction," *Natural Computing,* vol. 1, no. 1, pp. 3-52, 2002/03/01 2002, doi: 10.1023/A:1015059928466.

[58] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization,* vol. 11, no. 4, pp. 341-359, 1997/12/01 1997, doi: 10.1023/A:1008202821328.

[59] J. H. Holland, "Genetic Algorithms," *Scientific American,* vol. 267, no. 1, pp. 66-73, 1992. [Online]. Available: http://www.jstor.org/stable/24939139.

[60] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[61] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press, 1992.

[62] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 27 Nov.-1 Dec. 1995 1995, vol. 4, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[63] H. Heijnen, D. Howard, and N. Kottege, "A testbed that evolves hexapod controllers in hardware," in *2017 IEEE International Conference on Robotics*

and Automation (ICRA), 29 May-3 June 2017 2017, pp. 1065-1071, doi: 10.1109/ICRA.2017.7989128.

[64] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review," *ICTACT journal on soft computing,* vol. 6, no. 1, 2015.

[65] J. C. Roach *et al.*, "Analysis of Genetic Inheritance in a Family Quartet by Whole-Genome Sequencing," *Science,* vol. 328, no. 5978, pp. 636-639, 2010, doi: doi:10.1126/science.1186802.

[66] K. Deep and H. Mebrahtu, "Combined mutation operators of genetic algorithm for the travelling salesman problem," *International Journal of Combinatorial Optimization Problems and Informatics,* vol. 2, no. 3, pp. 1-23, 2011.

[67] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering*, 2011, vol. 2, no. 1: International Association of Engineers Hong Kong, China, pp. 1-6.

[68] P. W. MOORE and G. K. VENAYAGAMOORTHY, "EVOLVING DIGITAL CIRCUITS USING HYBRID PARTICLE SWARM OPTIMIZATION AND DIFFERENTIAL EVOLUTION," *International Journal of Neural Systems,* vol. 16, no. 03, pp. 163-177, 2006, doi: 10.1142/s0129065706000585.

[69] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Transactions on Evolutionary Computation,* vol. 15, no. 1, pp. 4-31, 2011, doi: 10.1109/TEVC.2010.2059031.

[70] M. Georgioudakis and V. Plevris, "A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization," (in English), *Frontiers in Built Environment,* Original Research vol. 6, no. 102, 2020-July-09 2020, doi: 10.3389/fbuil.2020.00102.

[71] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, 6-9 July 1999 1999, vol. 3, pp. 1945-1950 Vol. 3, doi: 10.1109/CEC.1999.785511.

[72] V. G. Gudise and G. K. Venayagamoorthy, "Evolving digital circuits using particle swarm," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, 20-24 July 2003 2003, vol. 1, pp. 468-472 vol.1, doi: 10.1109/IJCNN.2003.1223391.

[73] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[74] J. Zhong, X. Hu, J. Zhang, and 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)*, 2005, vol. 2: IEEE, pp. 1115-1121.

[75] TurtleBot. "TurtleBot3." https://www.turtlebot.com/turtlebot3/ (accessed 1/3/22, 2022).

[76] N. Kau, "Stanford pupper: A low-cost agile quadruped robot for benchmarking and education," *arXiv preprint arXiv:2110.00736,* 2021.

[77] M. H. Rahman, S. B. Alam, T. D. Mou, M. F. Uddin, and M. Hasan, "A Dynamic Approach to Low-Cost Design, Development, and Computational Simulation of a 12DoF Quadruped Robot," *Robotics,* vol. 12, no. 1, p. 28, 2023. [Online]. Available: https://www.mdpi.com/2218-6581/12/1/28.

[78] M. Bernal and J. Civera, "LoCoQuad: A Low-Cost Arachnoid Quadruped Robot for Research and Education," *arXiv preprint arXiv:2003.09025,* 2020.

[79] R. Amsters and P. Slaets, "Turtlebot 3 as a Robotics Education Platform," Cham, 2020: Springer International Publishing, in Robotics in Education, pp. 170-181.

[80] H. Aagela, M. Al-Nesf, and V. Holmes, "An Asus_xtion_probased indoor MAPPING using a Raspberry Pi with Turtlebot robot Turtlebot robot," in *2017 23rd International Conference on Automation and Computing (ICAC)*, 7-8 Sept. 2017 2017, pp. 1-5, doi: 10.23919/IConAC.2017.8082023.

[81] J. Vega and J. M. Cañas, "PiBot: An Open Low-Cost Robotic Platform with Camera for STEM Education," *Electronics,* vol. 7, no. 12, p. 430, 2018. [Online]. Available: https://www.mdpi.com/2079-9292/7/12/430.

[82] R. Krauss, "Combining Raspberry Pi and Arduino to form a low-cost, real-time autonomous vehicle platform," in *2016 American Control Conference (ACC)*, 6-8 July 2016 2016, pp. 6628-6633, doi: 10.1109/ACC.2016.7526714.

[83] S.-E. Oltean, "Mobile Robot Platform with Arduino Uno and Raspberry Pi for Autonomous Navigation," *Procedia Manufacturing,* vol. 32, pp. 572-577, 2019/01/01/ 2019, doi: https://doi.org/10.1016/j.promfg.2019.02.254.

[84] F. Nachbar *et al.*, "The ABCD rule of dermatoscopy: High prospective value in the diagnosis of doubtful melanocytic skin lesions," *Journal of the American Academy of Dermatology,* vol. 30, no. 4, pp. 551-559, 1994, doi: 10.1016/S0190-9622(94)70061-3.

[85] S. Majumder and M. A. Ullah, "Feature extraction from dermoscopy images for melanoma diagnosis," *SN Applied Sciences,* vol. 1, no. 7, p. 753, 2019/06/20 2019, doi: 10.1007/s42452-019-0786-8.

[86] N. R. Abbasi *et al.*, "Early Diagnosis of Cutaneous MelanomaRevisiting the ABCD Criteria," *JAMA,* vol. 292, no. 22, pp. 2771-2776, 2004, doi: 10.1001/jama.292.22.2771.

[87] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free Trial-and-Error Learning for Robot Damage Recovery," *Robotics and Autonomous Systems,* vol. 100, pp. 236-250, 2018/02/01/ 2018, doi: https://doi.org/10.1016/j.robot.2017.11.010.

[88] R. Kaushik, P. Desreumaux, and J.-B. Mouret, "Adaptive Prior Selection for Repertoire-Based Online Adaptation in Robotics," (in English), *Frontiers in Robotics and AI,* Original Research vol. 6, no. 151, 2020-January-20 2020, doi: 10.3389/frobt.2019.00151.

[89] R. Kaushik, T. Anne, and J. B. Mouret, "Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 24 Oct.-24 Jan. 2021 2020, pp. 5269-5276, doi: 10.1109/IROS45743.2020.9341462.

Appendix A: ANN Architecture Comparison

This appendix contains the description and results of the alternative ANN architectures that are investigated for the four experimental tasks, 1) hexapod gait control, 2) 2WD robot autonomous navigation, 3) character recognition, and 4) melanoma classification. The architectures are assessed based on evolutionary efficiency and controller/classifier performance; a discussion is made to why the architectures discussed in the main text were chosen as the most optimal to compare to the VPLD.

1. Hexapod Robotic Control

This section contains the description and results of the alternative ANN architectures that are investigated for hexapod gait control but found to have lower performance than the controller described in sections 4.1.

1.1 ANN Control System Architectures

The ANN architectures described below use the same inputs and outputs as the architecture described in section 4.1.8. A three layer ANN architecture was implemented with the number of neurons in the hidden layer ranging from one to four (Figure 1). The results of these varying architectures are analysed and the choice of the optimal architecture used in section 4.1 is detailed. As a recap the ANN uses the same inputs as the VPLD, the air/ground phase and counter signals. The air/ground phase signal has two states 1 for ground and -1 for air. The counter has ten steps for each phase of the gait changing in value from 0 to 0.9 in steps of 0.1. The output of the ANN is the angular change in position of the three joints in the leg of a hexapod, ranging from -15 to +15.

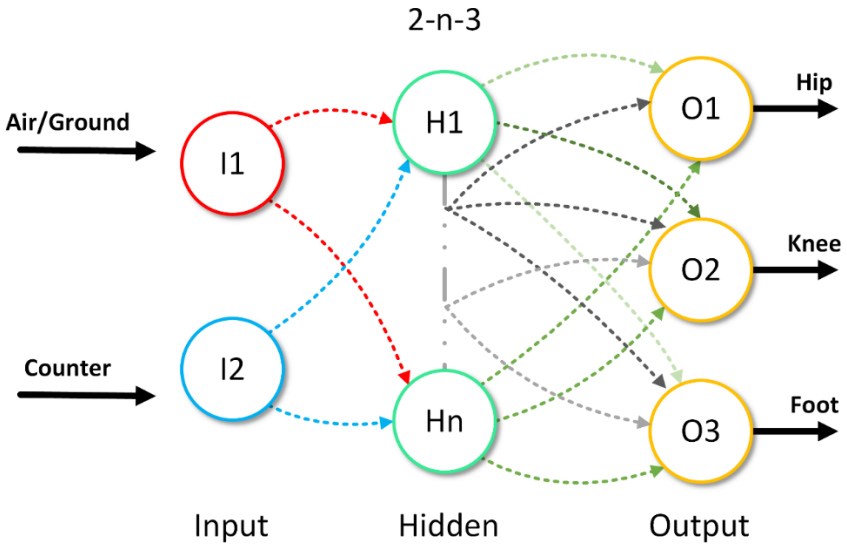

Figure 1 ANN *2-n-3*, two inputs, *n*=1,2,3,4 hidden layer neurons, three output neurons

The neurons in the hidden layer use a hyperbolic tangent activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01, giving an input range into the activation function of ±2.9. The output layer neurons use the same weights and biases as the hidden layer ranging from -1 to +1 in 0.01 steps.

1.2 Results

In this section the results from the experiments performed for each architecture are compared, to show how/why the optimal architecture discussed in section 4.1 was chosen. The ANN controllers are evolved using the same GA as the VPLD & EHW controllers to produce an optimal walking gait for a hexapod robot. The walking gait allowed the robot to walk forward in a straight line, maintaining a constant heading and body attitude. Three hundred solutions for each controller are evolved.

The controllers are evaluated based on their evolutionary efficiency, so how many generations and what was the computation time required to reach a solution. The ANN architectures are evaluated on the same PC as the VPLD which uses an Intel i7-11700k Octa-Core processor (3.6 GHz).

Evolutionary Efficiency

The fitness per generation plot (Figure 2) of the ANN controller architectures shows that all the architectures obtained a similar fitness within a comparable number of generations, taking less than 170 generations to evolve on average. The 2-2-3 ANN architecture required the least generations to evolve, requiring approximately 133 generations to evolve on average. The 2-4-3 ANN required the most generations, at approximately 169 generations (Table 1). The initial evolutionary rate of all the architectures is similar, with the 2-2-3 and 2-1-3 appearing to be slightly faster.

Table 1 ANN average number of generations required to successfully evolve controllers

Architecture	Mean	Median	CV
ANN 2-4-3	168.95	148	0.45
ANN 2-3-3	159.9	131	0.55
ANN 2-2-3	132.97	117	0.5
ANN 2-1-3	144.08	122	0.57

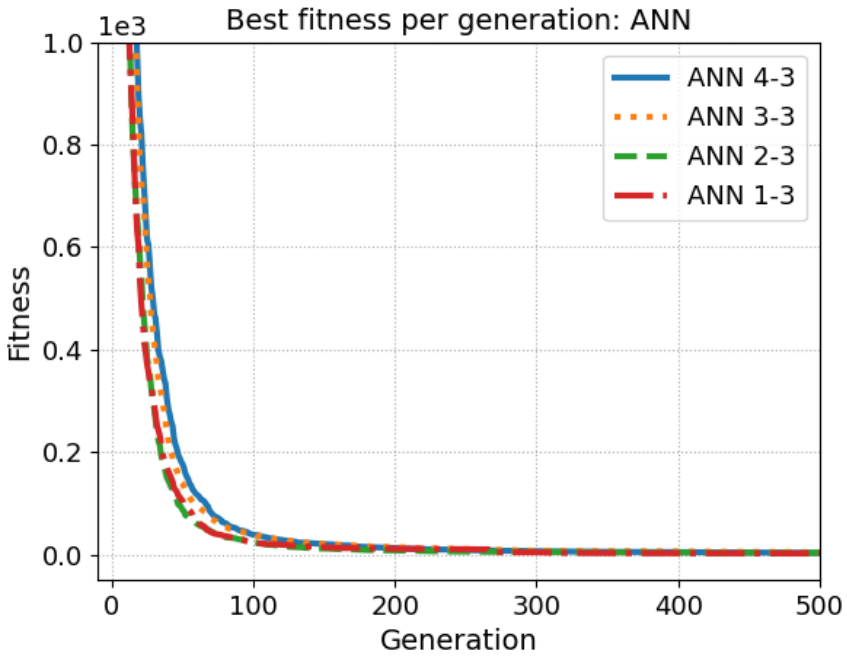

Figure 2 ANN best fitness per generation taken from the average of 300 evolution attempts

The computation time taken to evolve each architecture (Table 2). All the controllers have a comparable level of consistency in the required time to evolve. The evolution time reduced as the number of neurons decreased but only marginally, the required generations that had to be run was a greater factor on the overall computation time to evolve.

Table 2 ANN average computation time required to successfully evolve controllers

Architecture	Mean[s]	Median[s]	CV
ANN 2-4-3	1.01	0.9	0.45
ANN 2-3-3	0.95	0.78	0.55
ANN 2-2-3	0.79	0.7	0.49
ANN 2-1-3	0.82	0.71	0.55

The ANN 2-2-3 architecture required the least number of generations and computation time, making it the most evolutionary efficient architecture out of the four evaluated. This is why the 2-2-3 Architecture was chosen to be discussed in higher detail in section 4.1 as a part of the main text.

2. 2WD Mobile Robot Control

This section contains the description and results of the alternative ANN architectures that are investigated for the control of a 2WD mobile robot. These alternative architectures are found to have lower performance than the architectures described in section 4.2. The description of the robotic control problem and GA used to evolve the ANN architectures discussed below can be found in sections 4.2.1 and 4.2.5 respectively. To evaluate the ANN architectures, they are evolved specifically for the combined obstacle avoidance and light following problem.

2.1 ANN Control System Architectures

For the ANN five architectures are investigated. Three architectures with a single hidden layer (Figure 3), 10-4-1, 10-8-1, and 10-16-1; as well as two multi-layer architectures (Figure 4) are investigated, 10-16-16-1, and 10-16-16-16-1. All the neurons for the five architectures use the hyperbolic tangent activation function. The five architectures input is the floating-point data from the IR proximity sensors and light intensity sensors required for the combined obstacle avoidance and light following task as detailed in section 4.2.4. All the ANN architectures have a single output Neuron which gives a floating-point value used to set the RPM of one of the robot's wheels. As discussed in section 4.2.4 two ANN units are combined to drive the robot, one controls the left wheel, the other controls the right wheel.

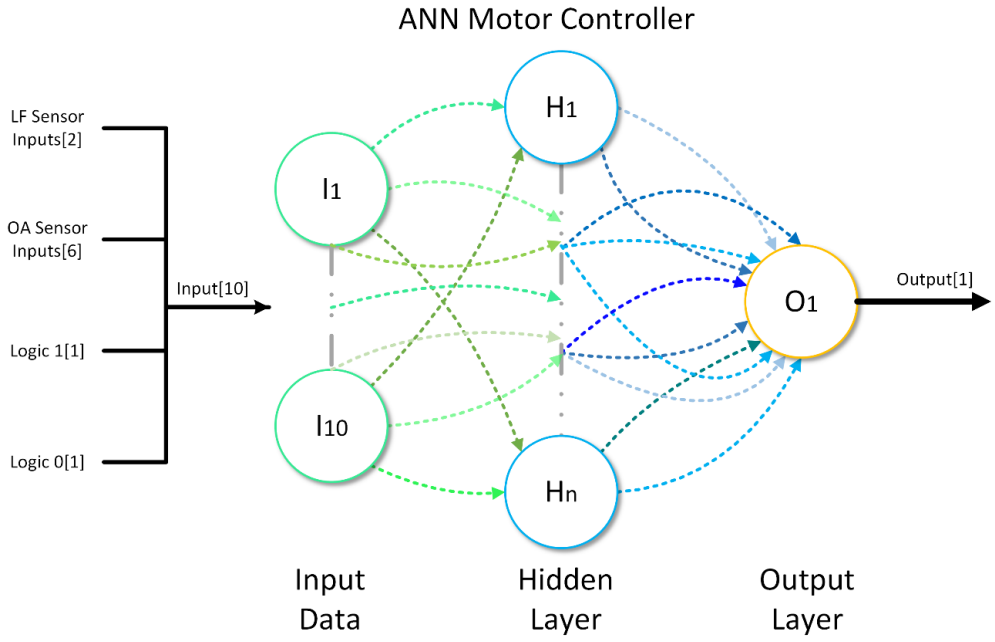

Figure 3 ANN Architectures for 2WD robot control; architecture with single hidden layer containing n neurons (n=4, 8, or 16) and 1 output neuron

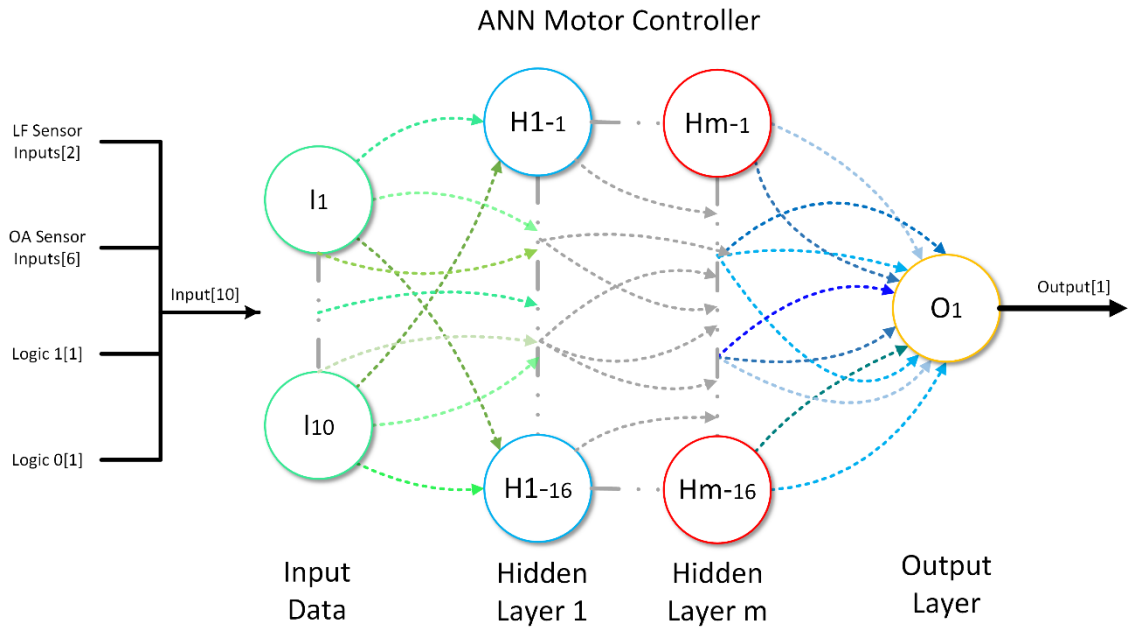

Figure 4 ANN Architectures for 2WD robot control; multi-layer architecture with 16 neurons in each hidden layer and for the final output layer which uses a single neuron. Architectures with m=1, 2, or 3 hidden layers are evaluated

2.2 Results

The five ANN architectures are evolved for the control of the 2WD robot for the combined obstacle avoidance and light following task; the controllers are required to drive the robot to the light source without crashing into obstacles in the path of the robot or the environment boundaries. The evolutionary efficiency and controller performance for each architecture is investigated. One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs is used to assess the system performance. Each architectures evolution is limited to 250 generations.

Each architecture is compared to show how/why the optimal architecture discussed in section 4.2 was chosen.

Table 3 Summary of ANN architectures Evaluated for the 2WD mobile robot

System	Configurations Tested
ANN	• 10-4-4 • 10-8-4 • 10-16-4 • 10-16-16-4 • 10-16-16-16-4

Evolutionary Efficiency

The fitness per generation graph (Figure 5) shows two trends, for the comparison of the architectures with a single hidden layer, as the number of hidden layer neurons decreases, so does the rate of evolution. The second trend is the two larger multilayer architectures with two and three hidden layers are slower to evolve than the architecture with a single hidden layer. The 10-4-4 and 10-8-4 are the fastest to evolve, converging to a solution in the least generations (Table 4) and in the least time (Table 5). As to be expected, the time to evolve across the five architectures increases with an increase in the number of neurons and hidden layers. This is because a larger chromosome particularly with more layers takes longer to process in the reproduction of the GA when producing new individuals

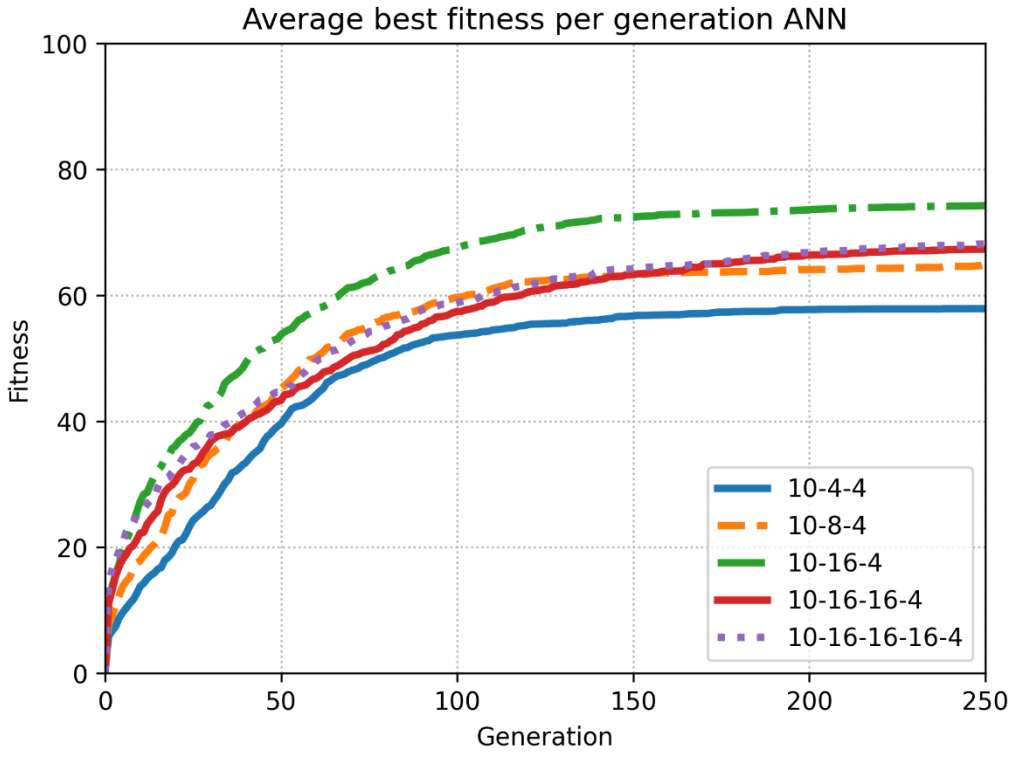

Figure 5 A comparison of the OA-LF fitness per generation of the five ANN architectures

Table 4 The Number of Generations to Evolve ANN for OA-LF Task

Controller	Mean	Median	CV
10-4-4	112.65	100.5	0.43
10-8-4	114.75	110.5	0.46
10-16-4	144.23	137.5	0.32
10-16-16-4	136.92	131	0.43
10-16-16-16-4	136.27	133	0.48

Table 5 Computation Time Required to Evolve ANN for OA-LF Task in Seconds

Controller	Mean	Median	CV
10-4-4	55.57	48.94	0.46
10-8-4	57.62	54.3	0.49
10-16-4	73.83	69.47	0.35
10-16-16-4	76.38	72.69	0.45
10-16-16-16-4	83.36	79.64	0.5

Controller Performance

For controller performance, in the architectures with the single hidden layer, as the number of hidden layer neurons decreases so does the fitness level. The two smallest architectures converge to the average lowest fitness level. When comparing the architecture with the single hidden layer to the multilayer networks, the larger networks achieve a lower fitness on average (Table 6).

Table 6 The ANN Fitness reached for the OA-LF Task

Controller	Mean	Median	CV
10-4-4	57.9	62.3	0.23
10-8-4	64.86	66.94	0.19
10-16-4	74.24	75.71	0.13
10-16-16-4	67.32	69.2	0.2
10-16-16-16-4	68.17	72.46	0.22

The decreased fitness and slower evolution of the larger multi-layer architectures could be contributed to the multi-point crossover being overly destructive to the larger chromosomes, chains of genes(weights and biases) passing information between layers could be lost during the multi-point crossover operation, in future work the use of different crossover operations could be investigated.

When choosing the optimal ANN architecture to use in section 4.2, both the evolutionary efficiency and controller performance are considered. The two fastest architectures are the 10-4-4 and 10-8-4. The 10-16-4 architectures achieves the highest average fitness level and does so more consistently than all the other architectures. The 10-16-4 architecture is chosen as the most optimal architecture because it reaches the highest fitness level and it's time to evolve is still comparable to the 10-4-4 and 10-8-4 architectures.

3. Character Recognition

This section contains the description and results of the alternative ANN architectures that are investigated for character recognition but found to have lower performance than the architecture described in section 5.1. A description of the GA used to evolve the architecture discussed below can be found in section 5.1.4.

3.1 ANN Classifier Architectures

The ANN architectures investigated are shown in Figure 6. First is a 35-52-26 architecture (52 neurons in the hidden layer and 26 neurons in the output layer), the second is a 35-52-78 architecture (52 neurons in the hidden layer and 78 neurons in the output layer). For the 35-52-26 architecture, a single neuron determines the level of activation of a given character, for the 35-52-78 architecture, the combination of 3 neurons determines the level of activation of a given character. The outputs of the ANN go through the same post processing as the D-VPLD and F-VPLD. The ANN uses the same input array in a binary form (0-black pixel and 1-white pixel). The output is a set of 26 values, one for each letter in the alphabet. The output value with the highest value determines the selected character.

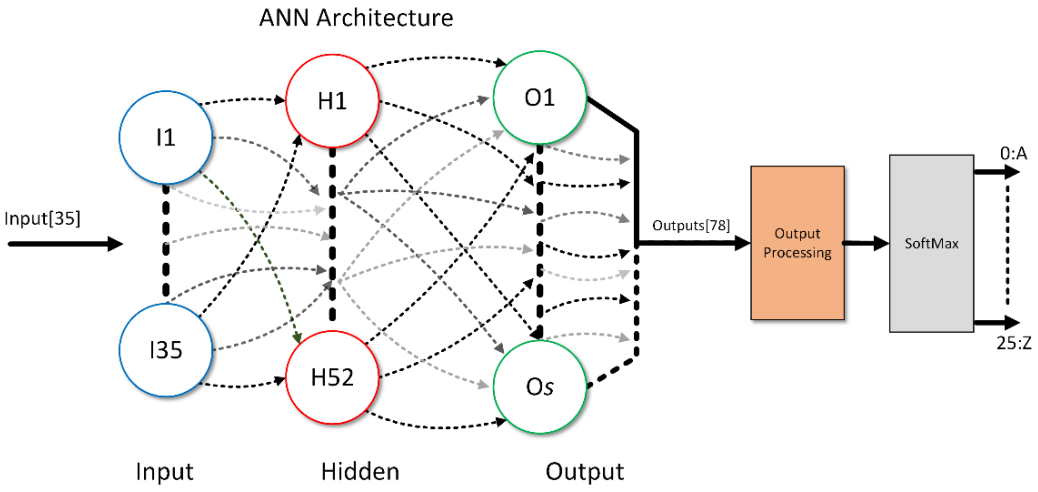

Figure 6 ANN 35-52-S, 52 hidden layer neurons, s=26 or 78 output neurons

The neurons in the hidden layer are tested with both a hyperbolic tangent activation function and the ReLu activation function. The weights & biases of the neurons range from -1 to +1 in steps of 0.01. The output layer neurons use the same weights and biases as the hidden layer ranging from -1 to +1 in 0.01 steps. The activation function of these neurons is the ReLu activation function.

The final output of the ANN it is an array of floating-point values of size 26. The same as the D-VPLD and F-VPLD, for the architecture with 26 output neurons, the output arrays skip the grouping process and is passed through a SoftMax function. For the architecture with the 78 output neurons, the output neurons are combined into groups of three. The grouped values are either averaged or the max value of the group is chosen as the output for the respective character. Once the grouped outputs are processed resulting in an output array of size 26, the output is fed into the SoftMax function to get the final result.

3.2 Results

Experiments are performed for the ANN with varying architectures. The evolutionary efficiency for each architecture is investigated using two different activation functions in the hidden layer. For the architectures using the grouped outputs, mean and max grouping functions are tested. The different configurations for each architecture are compared to find the most optimal solutions. The ANN systems are evolved to recognise 26 binary character images(A-Z), an optimal solution is a system that can recognise each character. One hundred solutions are evolved for each architecture being evaluated, the data of each of these one hundred GA runs are used to assess the system performance.

Table 7 Summary of D-VPLD and F-VPLD Configurations Evaluated

System	Configurations Tested

ANN	• 35-52-26
	  ○ with hyperbolic tangent
	  ○ with ReLu
	• 35-52-78
	  ○ with hyperbolic tangent and mean output grouping
	  ○ with hyperbolic tangent and max output grouping
	  ○ with ReLu and mean output grouping
	  ○ with ReLu and max output grouping

Evolutionary Efficiency

When comparing the 35-52-26 architecture solutions, the ANN using the hyperbolic tangent activation function required less generations to evolve on average, however, the ANN using the ReLu activation is not significantly slower. The ANN ReLu system required approximately 1.2x more generations to evolve (Table 8). Both 35-52-26 ANN solutions evaluated required less than 30 seconds to evolve (Table 9). The best fitness per generation of both the architectures evaluated is shown in Figure 7, the ANN using the hyperbolic tangent activation has a faster initial rate of evolution.

Table 8 Character Recognition generations to evolve, 35-52-26 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

Sys	Mean	Median	CV
ANN ReLu	931.2	571	1.00
ANN tanh	764.37	372.5	1.27

Table 9 Character Recognition time to evolve, 35-52-26 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

Sys	Mean[s]	Median[s]	CV
ANN ReLu	25.53	15.71	0.97
ANN tanh	22.66	11.32	1.24

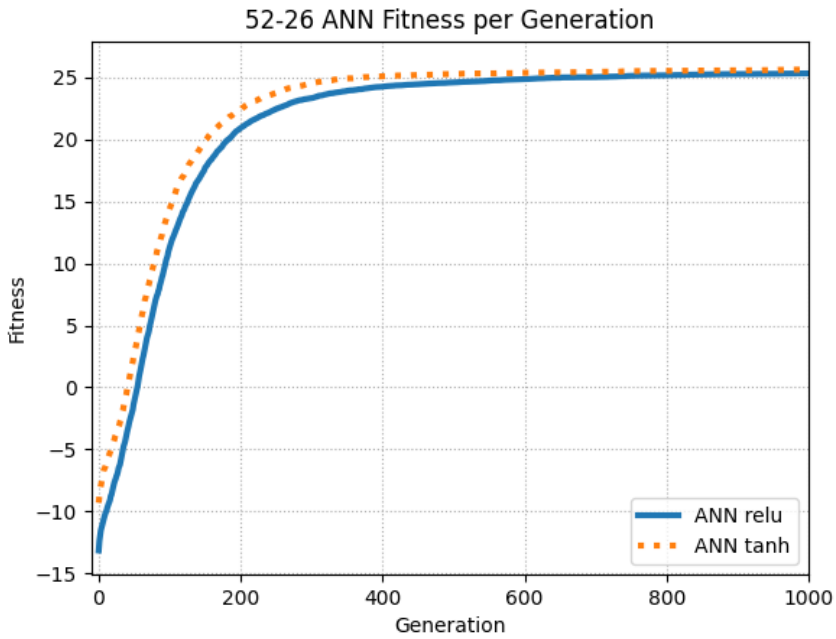

Figure 7 Character Recognition best fitness per generation, 35-52-26 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

For the 35-52-78 ANN four configurations are tested. Comparing the generations required to evolve for the two different output group functions are similar, all the solutions using the mean & max output group functions require less than 500 generations to evolve (Table 10). When comparing the mean & max output group functions, and specifically the differences in using the hyperbolic tangent and ReLu activation functions there is not a clear trend. For the ReLu solutions the ANN max system is faster to evolve than the ANN mean system, for the hyperbolic tangent solutions the ANN mean system is faster than the ANN max system. All the ANN solutions take less than 30 seconds on average to evolve (Table 11). The best fitness per generation of all the architectures evaluated is shown in Figure 8.

Table 10 Character Recognition generations to evolve, 35-52-78 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

Sys	Mean	Median	CV
ANN Mean ReLu	451.47	412.5	0.35
ANN Mean tanh	407.58	384	0.27
ANN Max ReLu	386.75	354.5	0.35
ANN Max tanh	450.78	381	0.68

Table 11 Character Recognition time to evolve, 35-52-78 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

Sys	Mean[s]	Median[s]	CV
ANN Mean ReLu	25.93	23.9	0.34
ANN Mean tanh	23.19	21.75	0.27
ANN Max ReLu	20.13	18.71	0.34
ANN Max tanh	23.06	19.45	0.67

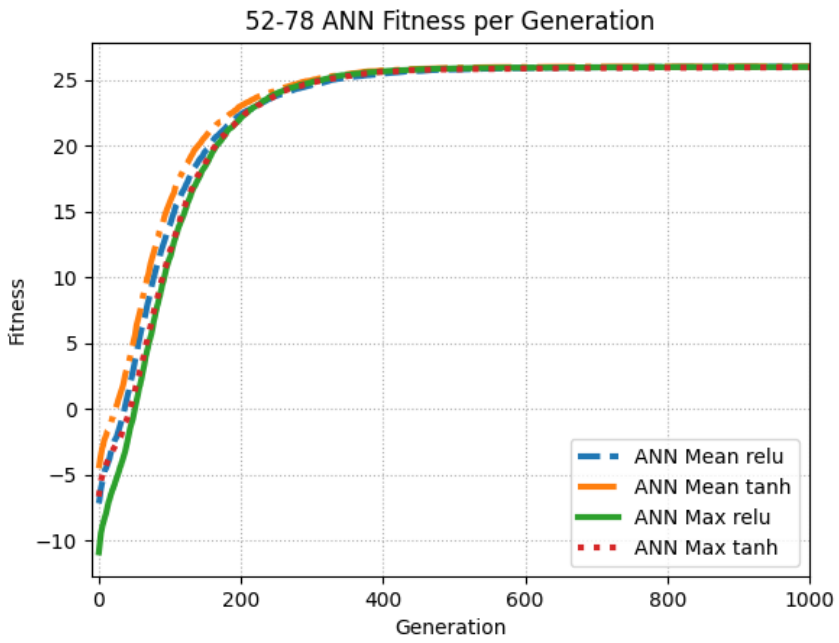

Figure 8 Character Recognition best fitness per generation, 35-52-78 ANN comparison using ReLu and Hyperbolic Tangent activation in the hidden layer

All of the 35-52-78 ANN systems are faster to evolve than the 35-52-26 ANN systems. Comparing the fastest 35-52-26 ANN & 35-52-78 ANN systems, the 35-52-26 ANN tanh requires approximately 2x the generations to evolve compared to the 35-52-78 ANN max ReLu system. Even though there is a larger difference in the required generations, due to the 35-52-26 ANN system having less neurons its time on average is very similar to evolve. The most optimal system of the architecture investigated was the 35-52-78 ANN max ReLu, this architecture is used in the main chapters for the comparison with the D-VPLD and F-VPLD (section 5.1).

4. Melanoma Classification

This section contains the description and results of the alternative ANN architectures that are investigated for the melanoma classification problem but found to have lower performance than the architectures described in section 5.2. The description of the proposed method and GA used to evolve the architectures discussed below can be found in sections 5.2.1 and 5.2.5 respectively.

4.1 ANN Classifier Architectures

For the ANN, two feedforward fully connected architectures (Figure 9) have been investigated using two different activation functions, the first is a 16-16-1 architecture (16 hidden layer neurons, 1 output neuron), the second is a 16-16-3 architecture (16 hidden layer neurons, 3 output neurons). The 16-16-3 architecture requires postprocessing, where the three neuron outputs are combined to form a single output for classification, this is done by taking the mean of the 3 outputs. The 16-16-1 architecture requires no output processing. The ANN inputs are the 16 extracted ABCD and GLCM features described in section 5.2.1.

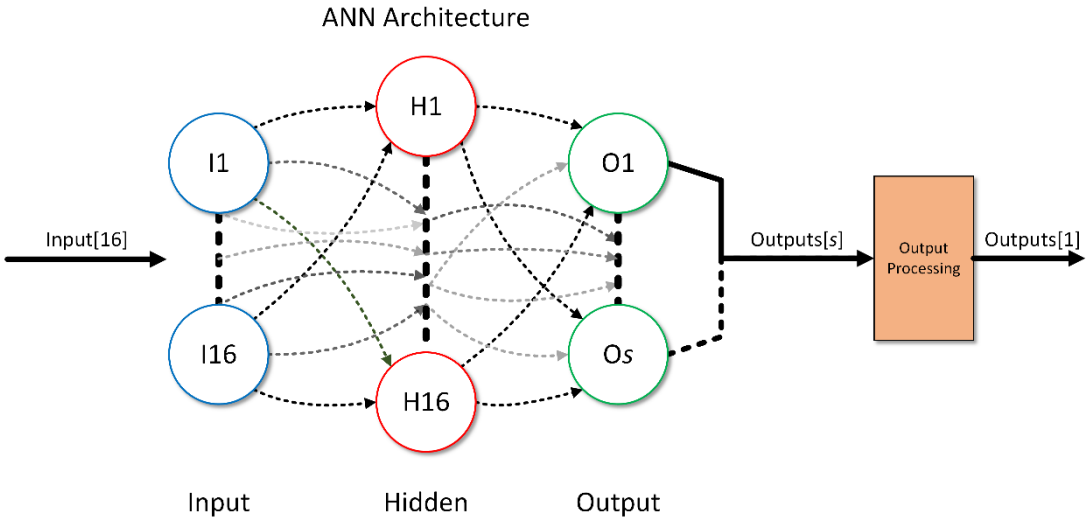

Figure 9 ANN Architectures for binary classification of melanomas; 16-16-s architecture with 16 hidden layer neurons and s neurons in output layer. $s = 1$ *or* 3

The neurons in both the hidden and output layer share the same activation function, both architectures are tested using the hyperbolic tangent and sigmoid activation functions. The neurons weights & biases of the neurons range from -1 to +1 in steps of 0.01. Using the hyperbolic tangent activation function, the neurons output ranges from -1 to +1. When using the sigmoid activation function, the neurons output ranges from 0 to +1.

4.2 Results

As with the VPLDs, the evolutionary efficiency and classification accuracy of the ANN is investigated. Two architectures are evaluated with two different activation functions (sigmoid and hyperbolic tangent). The different configurations for each architecture are compared to determine the most optimal architecture for comparison to the VPLD in section 5.2.

The ANN is evolved to classify skin lesions from the PH2 database, the lesions are either cancerous melanoma or non-cancerous common/atypical nevus. One hundred solutions are evolved for each architecture, the data of each of these one hundred GA runs is used to assess the system performance.

Table 12 Summary of ANN Configurations Evaluated

System	Configurations Tested

ANN	• 16-1
	  ○ with hyperbolic tangent
	  ○ with sigmoid
	• 16-3
	  ○ with hyperbolic tangent and mean output grouping
	  ○ with sigmoid and mean output grouping

Evolutionary Efficiency

The required number of generations to converge to an optimal solution is comparable for all configurations evaluated, both architectures evolve in close to 800 generations (Table 13). The computation time however is greater for the larger architecture (Table 14), this difference is caused by the added computation time required for a) the reproduction step of the GA process, and b) the fitness assessment stage of the GA process, for the post-processing operation performed on the ANN output for the 16-16-3 architecture. The fastest ANN architecture to converge to an optimal solution is the 16-16-1 architecture using the hyperbolic tangent activation function, approximately 770 generations in 12.59s of computation time on average. In Figure 10 showing the average training accuracy (fitness) per generation, it is observed that both architectures have a higher initial rate of evolution and reach a higher training accuracy when using the hyperbolic tangent activation function.

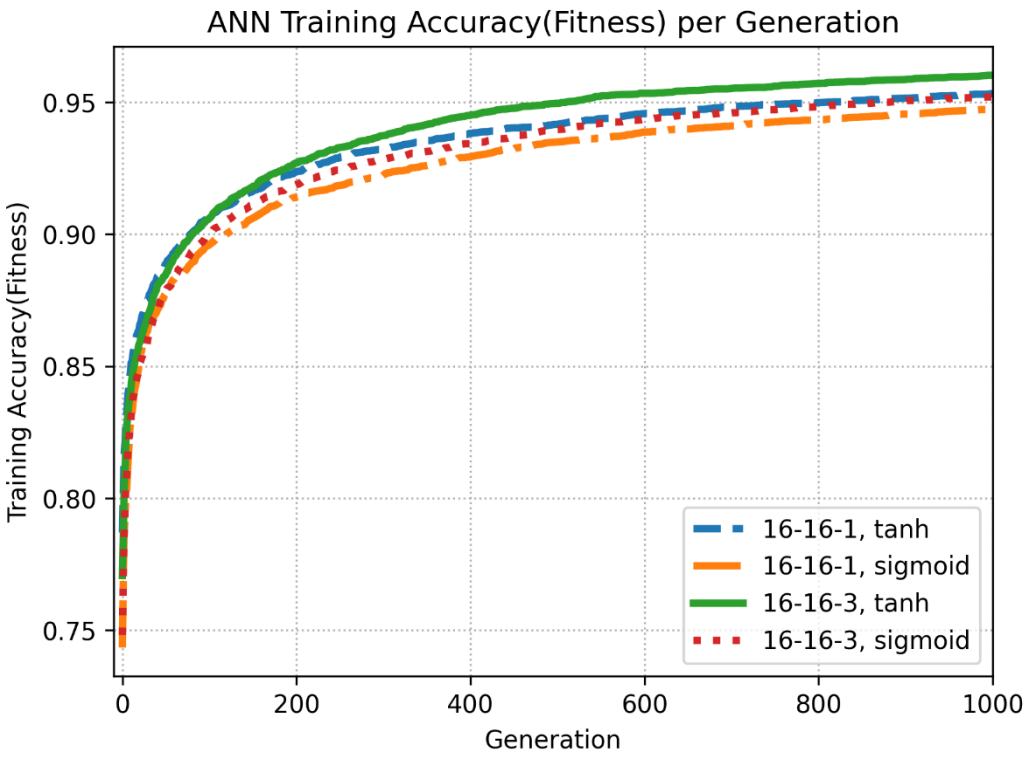

Figure 10 ANN Melanoma Classification, average best training accuracy(fitness) per generation

Table 13 ANN Melanoma Classification Number of Generations to Evolve

Sys	Mean	Median	CV
16-16-1, tanh	770.38	810.5	0.22
16-16-1, sigmoid	792.73	834	0.2
16-16-3, tanh	757.89	796.5	0.24
16-16-3, sigmoid	760.2	823	0.25

Table 14 ANN Melanoma Classification Time to Evolve

Sys	Mean[s]	Median[s]	CV
16-16-1, tanh	12.59	13.24	0.22
16-16-1, sigmoid	13.09	13.83	0.2
16-16-3, tanh	18.51	19.28	0.24
16-16-3, sigmoid	18.53	19.92	0.25

Classifier Performance

Both architectures with both configurations reach training and test accuracies higher than 90% (Figure 11, Table 15, & Table 16). The 16-16-3 architecture using the hyperbolic tangent activation function has the highest average training accuracy and the highest average test accuracy. Both architectures when using the hyperbolic tangent activation function have a higher test accuracy on average, than when using the sigmoid function. The accuracy level achieved by each architecture is very consistent, shown by the coefficient of variance calculated from the 100 results, all architectures with both configurations have a 1% CV for training accuracy, and a 2% CV for test accuracy (Table 15 & Table 16).

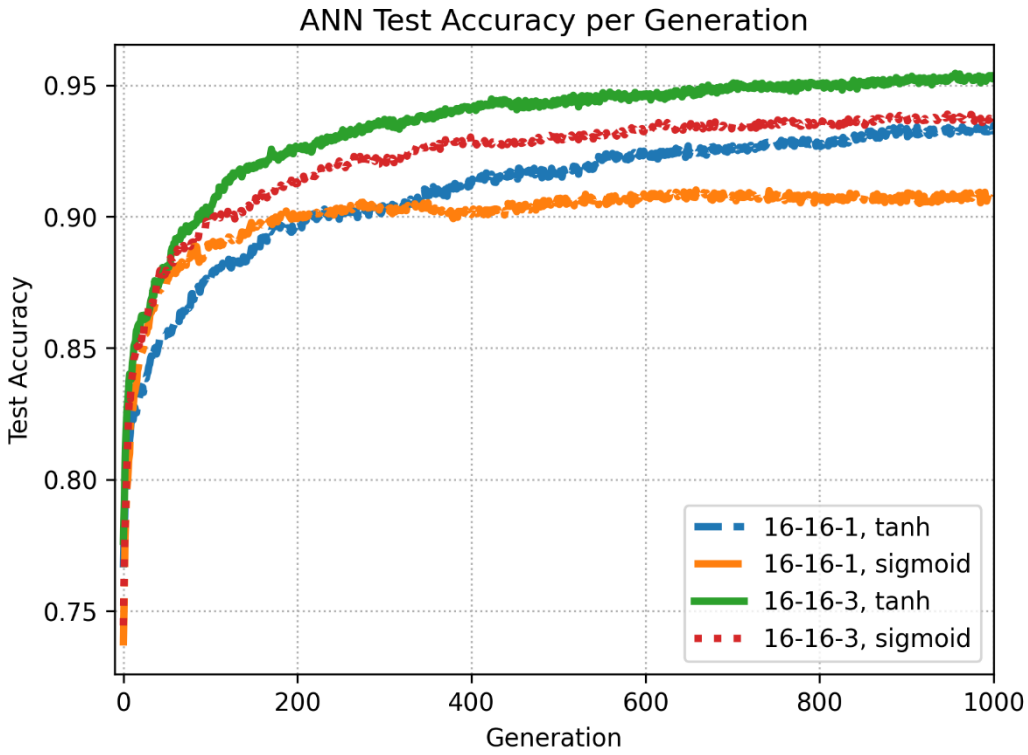

Figure 11 ANN Melanoma Classification, average test accuracy per generation

Table 15 ANN Training Accuracy (Fitness) for Melanoma Classification problem

Sys	Mean	Median	CV
16-16-1, tanh	95.32	95.42	0.01
16-16-1, sigmoid	94.74	95	0.01
16-16-3, tanh	96	95.83	0.01
16-16-3, sigmoid	95.2	95.42	0.01

Table 16 ANN Test Accuracy for Melanoma Classification problem

Sys	Mean	Median	CV
16-16-1, tanh	93.48	93.75	0.02
16-16-1, sigmoid	90.81	91.25	0.02
16-16-3, tanh	95.31	95	0.02
16-16-3, sigmoid	93.75	93.75	0.02

The 16-16-3 architecture using the hyperbolic tangent is chosen as the most optimal ANN architecture. As with both VPLDs, the most optimal architecture is chosen based on the classification accuracy, as this is the most important factor for a classifier model. When considering the training and test accuracy, the 16-16-3 architecture using the hyperbolic tangent function is the most optimal, having the highest training and test accuracy on average.

Appendix B: Supplementary Resources

Supplementary Resource 1

Hexapod Robotic Control-Deployed in Simulation Video
https://youtu.be/iWPSDBCe-C4

Supplementary Resource 2

Hexapod Robotic Control-Deployed to real robot Video
https://youtu.be/RgtlvyIOK_4