Evolutionary Networks for Multi-Behavioural Robot Control

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Science
in
Computer Science

Massey University, Albany,
New Zealand.

Adam Roger John Jordan

supervised by
Dr. Napoleon Reyes and Dr. Andre Barczak

2015

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Abstract

Artificial Intelligence can be applied to a wide variety of real world problems, with varying levels of complexity; nonetheless, real world problems often demand for capabilities that are difficult, if not impossible to achieve using a single Artificial Intelligence algorithm. This challenge gave rise to the development of hybrid systems that put together a combination of complementary algorithms. Hybrid approaches come at a cost however, as they introduce additional complications for the developer, such as how the algorithms should interact and when the independent algorithms should be executed. This research introduces a new algorithm called Cascading Genetic Network Programming (CGNP), which contains significant changes to the original Genetic Network Programming. This new algorithm has the facility to include any Artificial Intelligence algorithm into its directed graph network, as either a judgement or processing node. CGNP introduces a novel ability for a scalable multiple layer network, of independent instances of the CGNP algorithm itself. This facilitates problem subdivision, independent optimisation of these underlying layers and the ability to develop varying levels of complexity, from individual motor control to high level dynamic role allocation systems. Mechanisms are incorporated to prevent the child networks from executing beyond their requirement, allowing the parent to maintain control. The ability to optimise any data within each node is added, allowing for general purpose node development and therefore allowing node reuse in a wide variety of applications without modification. The abilities of the Cascaded Genetic Network Programming algorithm are demonstrated and proved through the development of a multi-behavioural robot soccer goal keeper, as a testbed where an individual Artificial Intelligence system may not be sufficient. The overall role is subdivided into three components and individually optimised which allow the robot to pursue a target object or location, rotate towards a target and provide basic functionality for defending a goal. These three components are then used in a higher level network as independent nodes, to solve the overall multi-behavioural goal keeper. Experiments show that the resulting controller defends the goal with a success rate of 91%, after 12 hours training using a population of 400 and 60 generations.

Acknowledgements

Would just like to thank my supervisors, friends and family; all of whom helped in their own respective ways, be it bouncing ideas off one another or listening to my rants when something didn't work as planned.

I would also like to thank Massey University for the Masterate Scholarship I was given to complete this research, it was of significant help throughout the degree.

Contents

List of Figures

List of Tables

List of Codes

List of Equations

Chapter 1

Introduction

As the technological world further develops into unexplored territory due to consumer requirements or research breakthroughs in varying fields, additional requirements and constraints are placed upon computer systems for complicated behaviours, analysis or detection algorithms. Artificial Intelligence is a mainstream solution for these increasing requirements, however often the requirements set by these industries are beyond the scope of any single Artificial Intelligence algorithm. This limitation incites further research to develop hybrid algorithms or use a combination of multiple intelligent architectures to meet the expectations placed upon developers and researchers alike. Hybrid systems introduce considerations as to how these algorithms should interface with one another and using multiple intelligent systems to solve a problem introduces further decisions as to under which conditions these algorithms should be independently executed.

Genetic Network Programming was introduced by Katagiri et al.[5], this algorithm is a network of nodes connected to one another in a directed graph. These nodes can be either Judgements for branching and decision making, or Processing nodes for actions or interfacing with the environment they are designed for. This algorithm has found success in solving agent based systems[5, 6], double-deck elevator control systems[7] and some robotic applications[9, 10].

This Genetic Network Programming architecture has the ability to optimise the connections between these nodes and the node types themselves from a predefined library of potential node types. While this algorithm efficiently finds solutions for the required tasks the networks are designed for, the networks themselves are limited by the contents of the node library. The contents of the node library are specifically designed for the problem that the network is to achieve, increasing the required programmer development time to specialize nodes to this task. For complicated behaviours with a large number of objectives, the networks designed using this algorithm will become complicated for human interpretation.

The research discussed within this thesis proposes the Cascaded Genetic Network Programming architecture that is significantly modified from the original, to facilitate multiple layers of network structures with varying complexity, general purpose nodes reducing the need for problem specific node development and the optimisation of internal node data to exceed the limitations of predefined node libraries. This new architecture bridges the gap between different Artificial Intelligent systems, by treating them as independent nodes within a network and handles problem complexity, by allowing subdivision of any task into a new Cascaded Genetic Network Programming layer.

This new architecture is applied to the goal keeper behaviour in robot soccer, where the environment is repeatedly changing and the goal keeper must adhere to multiple objectives in order to efficiently defend the goal. The goal keeper problem is one possible case where multiple artificial intelligent systems are required to achieve the overall objective with a high success rate. To aid in the training optimisation a simulation environment is implemented to accurately represent the platform at Massey University, this is done by utilizing realistic physics and accurately representing the shape of the robot and environment. The goal keeper controller developed in conjunction with this research could be used in the Massey University robot soccer team for the FIRA MiroSot Middle League[12] competition.

1.1 Research Objectives

The primary objective of this thesis is to enhance the Genetic Network Programming architecture for environments that require multiple objectives, allowing it to extend to problems that are too complicated for the original. This new architecture should allow GNP to cascade into multiple sub-layers and allow more general purpose nodes to be developed by optimising the data internal to every node, these general purpose nodes will then be applicable to any problem thus reducing any required programming time. To test the capabilities of this new architecture a real life applicable, multi-behavioural goal keeper controller for the Massey University robot soccer platform is developed. It should have a high success rate when defending the goal, remain within the goal area so that it isn't left undefended and be able to reposition itself if knocked out of the goal area.

1. Develop a Cascading GNP architecture that facilitates subdivision of Artificial Intelligence problems into smaller, manageable tasks by allowing CGNP's to be contained within another in a complex multilayer network.

2. Investigate the current GNP architecture and make any changes required to allow a full object orientated design that can easily integrate any Artificial

Intelligence algorithm as either a processing or judgement node.

3. Develop a method to allow trained CGNP networks to be saved and imported into another CGNP network as a pre-trained Artificial Intelligence node for either processing or judgement.

4. Develop a method to represent all the node properties in a chromosome and provide an interface for future node development.

5. Develop a robot soccer simulation which accurately represents the FIRA MiroSot platform, using a full-fledged physics engine.

6. Build a platform independent C++ library for the new CGNP architecture.

7. Develop an interface that can bind the simulation environment with the CGNP architecture and allow modification to the CGNP properties for each node.

8. Prove that the CGNP architecture can be used to identify and fix issues in pre-existing AI systems.

9. Train and test the new architecture on a complex multi-behavioural controller such as the robot soccer goal keeper.

1.2 Significance of the Research

This research investigates the use of the Genetic Network Programming architecture for complex tasks where multiple objectives must be met and shows a modified implementation of the architecture that will significantly improve the development times of future AI research and allow trained AI systems to be reused in other applications. The GNP architecture can be used to solve problems or to improve existing solutions developed using any AI algorithm.

To aid in the re-usability of GNP systems a new approach is implemented to allow GNP to cascade into multiple layers of independent GNP systems, where a pre-trained system (or even untrained) can be treated as a node within another GNP, this allows complex behaviours to be designed and trained using predefined sub-behaviours. Since GNP is inherently a diverse algorithm, by using it to solve generic problems the resulting systems can be easily reused in other applications within the same scope, therefore there is no need to create a new controller to move a new robot if one has already been created for another.

This research is undertaken due to the recent research by Wenhan Wang[10] at Massey University, while he used the GNP architecture with Reinforcement Learning nodes he did not optimise data within Judgement nodes or other Processing

nodes. Within his research long training times were encountered for a Target Pursuit with Wall Avoidance system. This research further extends on Wang's research by allowing all data within nodes to be optimised regardless of type, develops the new Cascading GNP architecture and develops a robot soccer goal keeper, which is considered to be a more complicated behaviour than Target Pursuit with wall Avoidance.

1.3 Scope and Limitations

This research is directly applicable to the FIRA MiroSot Middle League[12] environment however, with modifications the GNP architecture could be applied to any AI based application including but not limited to, stock markets, game development or other robot controllers.

Within the robot soccer simulation, only two dimensions are simulated and therefore does not take into consideration any situation where the ball or robot bounce or jump, nor in the situation where a robot is somehow pushed onto its side.

This research does not consider AI systems that transmit data to others, therefore all nodes act independently. Data transmission between nodes could be implemented and would be a useful feature, this is further discussed in the Further Research section (Section 8.2.1).

As Genetic Algorithms are used within the GNP architecture, there is no guarantee that the best solution will be received. However with a large enough population and number of generations, the optimisation algorithm will find a solution that can solve the problem assuming a well defined objective function is used.

1.4 Structure of the Thesis

This thesis begins with an introduction to two relevant intelligent systems; Fuzzy Logic and Genetic Algorithms, both of these are used regularly throughout the controller design process. Next a detailed review on the current Genetic Network Programming architecture, other approaches to complete the goal keeping behaviour and other relevant research. Then the software design and implementation is explained with descriptions and examples from the external libraries that are used in this application. Following this is a chapter that describes the simulation platform developed in conjunction with this research and specific design considerations.

Chapter 6 discusses the original GNP architecture and then all modifications required to produce the desired Cascaded GNP architecture. Chapter 7 covers the controller designs for Target Pursuit, Target Rotation, basic Goal Defending and the Multi-Behavioural Goal Keeper, this chapter discusses each component in detail

and how they could be further improved. Lastly the conclusions of the research, a summary of the achievements made by this research and a series of future research topics using CGNP and other components of this research.

Chapter 2

Primer on Relevant Intelligent Systems

This chapter covers a basic outline on Fuzzy Logic and Genetic Algorithms; two Intelligent Systems used within this research. Fuzzy Logic is generally used as a type of controller where outputs are based on some form of input and Genetic Algorithms are used for optimisation purposes when specific parameters need to be optimised for a certain objective function.

2.1 Fuzzy Logic

Fuzzy Logic is a computational technique that reasons with the environment using imprecise terms and operates on a set of rules to generate a precise output. Fuzzy Logic is useful in situations where there is not a definite answer to a problem, but can be interpreted as a degree of multiple answers. One such example is climate control, where the temperature can be stabilized between Warm and Hot.

The algorithm has had a long history, where the term being initially defined by Lotfi A. Zadeh[13] in his 1965 paper regarding fuzzy set theory. Takagi and Sugeno[14] later expanded on Zadeh's ideas to include a degree of membership for each fuzzy set, so instead of each fuzzy set outcome being true or false, they can be any real number between 0 and 1 to donate the degree of membership. The Takagi-Sugeno Fuzzy Logic systems are used in this research.

Fuzzy Logic can be separated into three concise parts, Fuzzification, Fuzzy Rule Sets and Defuzzification, Figure 2.1 shows these stages in a fuzzy controller and that they interact with an external system; which could be anything from climate control[15] to parking a car[16].

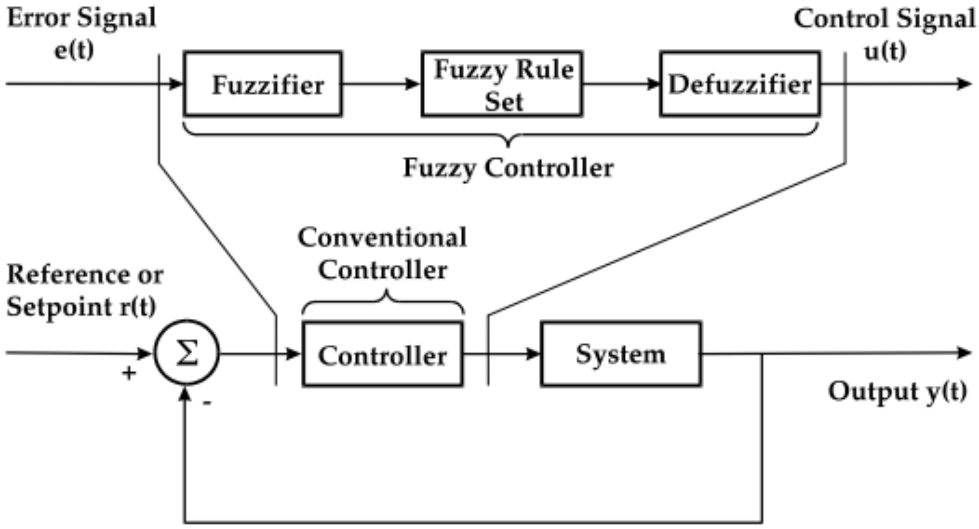


Figure 2.1: Fuzzy Logic in a Control Loop [1]

2.1.1 Fuzzification

The Fuzzification process itself has three components, the input values, membership functions and degrees of membership. All inputs that are used in the fuzzy system are taken from the environment that they are being used to control, be it a output voltage error, volume levels of a tank or output from another fuzzy system. Membership Functions are a collection of shapes that span the potential values for each input and are used to map the input value to a set of degrees of membership which are defined as real numbers between zero and one; trapeziums, triangles and bell curves are often used as membership functions. Figure 2.2 shows an example set of membership functions which takes a voltage as an input.

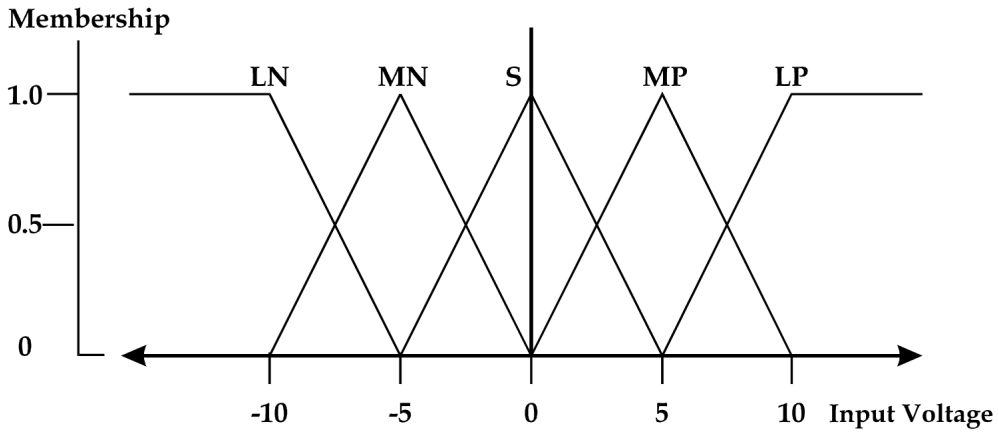


Figure 2.2: Example set of membership functions [1]

Looking at Figure 2.2, if the voltage input was 2.5v then the first two membership functions and the last (LN, MN and LP) would get a membership degree of 0.0, both S and MP would get a membership degree of 0.5 as that is where a value of 2.5 would intercept the membership functions.

2.1.2 Rule Evaluation

Once Fuzzification has been evaluated the degree of membership for each membership function must be assigned to an output value. This is done with a predefined set of if/then rules called the fuzzy rule set, there is one fuzzy rule for every membership function.

Each output is assigned a value that is relative to the problem, in the case of the voltage membership functions the output could control the speed of an engine. This could range between zero and one, zero being off and one being full power.

Shown in Code 2.1 and Code 2.2 are examples of what these output values and fuzzy rule sets could be. When the fuzzy rule sets are defined the process moves onto Defuzzification.

Code 2.1: Example Fuzzy Logic, output values

```
TURNOFF     = 0.0
SMALL       = 0.2
MEDIUM      = 0.5
LARGE       = 0.7
VERYLARGE   = 1.0
```

Code 2.2: Example Fuzzy Logic, fuzzy rule sets

```
if voltage is LN then output TURNOFF
if voltage is MN then output SMALL
if voltage is S then output MEDIUM
if voltage is MP then output LARGE
if voltage is LP then output VERYLARGE
```

2.1.3 Defuzzification

The last stage of Fuzzy Logic is Defuzzification, this involves taking the degree of membership values and translating them into a single crisp output value by using both the output values and fuzzy rule sets, which can then be used in the control system it is built for.

This is done with some form of equation, the most common of which is the Weighted Average equation shown in Equation 2.1.

$$CrispOutput = \frac{\Sigma_{i=0}^{n}(M_i O_i)}{\Sigma_{i=0}^{n}(M_i)}. \tag{2.1}$$

Taking the previously defined data for the Membership Functions (Figure 2.2), Output Values (Code 2.1) and Fuzzy Rule Sets (Code 2.2), along with the voltage input of 2.5v then the evaluated fuzzy logic system is shown in Equation 2.2.

$$\begin{aligned} CrispOutput &= \frac{\Sigma_{i=0}^{n}(M_i O_i)}{\Sigma_{i=0}^{n}(M_i)}, \\ &= \frac{0.0(0.0) + 0.0(0.2) + 0.5(0.5) + 0.5(0.7) + 0.0(1.0)}{0.0 + 0.0 + 0.5 + 0.5 + 0.0}, \\ &= 0.6 \end{aligned} \tag{2.2}$$

Therefore the value of 0.6 will be the final output of the fuzzy system and would then set the engine to just over half speed.

2.2 Genetic Algorithms

Genetic Algorithms are a form of optimising algorithm where given an objective function which defines the ideal scenario, can evolve a population of candidate solutions using biology inspired natural selection to find an 'acceptable' solution. This algorithm is well suited to situations where there are many local maxima, so long as the population remains diverse, but it should be noted that it does not guarantee finding the global maximum for any situation.

First introduced by John H. Holland in the 1970s[17], Genetic Algorithms have had many applications over the past forty years as it is a general purpose optimisation algorithm that can be applied to almost any situation. To name a few proven applications genetic algorithms have been used optimise a dynamic anti-terrorism strategy for maritime piracy[18] and even airline revenue management systems[19].

The algorithm is best described in three parts, the population which contains all candidate solutions, evaluation is the process of testing each candidate solution and evolution breeds a new generation of the population and incorporates any form of mutations. Figure 2.3 shows a basic flowchart of the process.

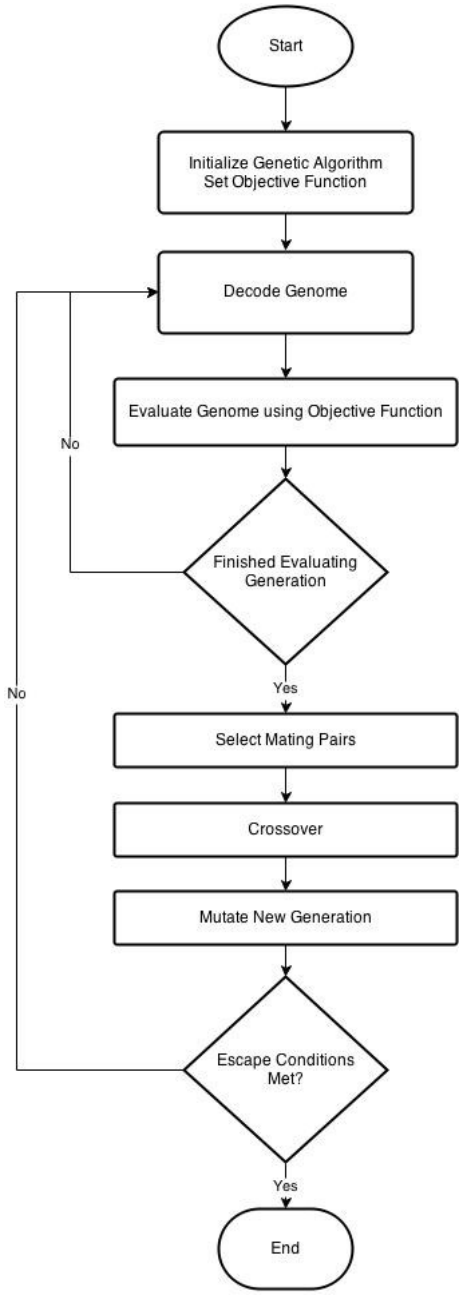


Figure 2.3: Genetic Algorithm Flowchart

2.2.1 Population

The population is of a user defined size which contains a set of candidate solutions (also called chromosomes or genomes), these candidate solutions contain a set of parameters or genes that are the subject of this optimisation process, this is shown in Figure 2.4. Therefore every variable that should be optimised should be accounted for in the predefined chromosome blueprint, along with all its potential values.

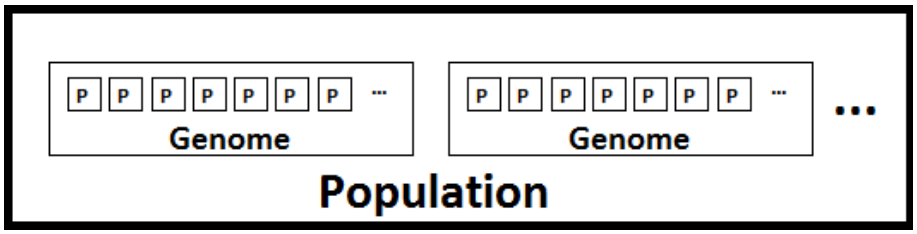


Figure 2.4: Genetic Algorithm Population Contents, P denotes parameter

There are two prominent encoding schemes that are used within genetic algorithms, both can apply to any situation but they both have benefits and flaws that may influence the applications they are applied to. They are binary and floating point encoding, binary encoding is the original design set forward by John H. Holland[17] and floating point encoding was later developed to make floating point value representation easier.

Binary encoding views all data as a binary string, which is composed of all genes stitched together in their binary form. This works well for integers and boolean values as they are natively represented in an expandable binary string and can be easily stitched together. There is a small issue with integer representation, when using ranges that do not match the maximum representation with the same number of bits. If the developer is to represent the range 0-14 which has a maximum binary value of 1110, since there are four binary digits in this maximum value using binary representation will also output 1111 (15) at some point throughout the optimisation process. This will therefore exceed the desired range of 0-14, which means special considerations need to be made when the value of 15 is used. Usually the value 15 will simply be converted to 14 as it is the maximum desired value, however then the value 14 is twice as likely to occur in the population meaning the population is no longer uniformly randomised.

One solution to this issue is using unary encoding, however this introduces a large amount of redundancy in the chromosome length. The 0-14 range in unary encoding will use 14 binary bits, these 14 bits will be summed together to receive the final value. This exponentially increases the length of a single gene and will consequently increase the required time conducting crossover or mutation.

Another consideration and the main reason why floating point encoding would be used, is to represent a floating value. Using binary encoding to represent a floating point value, there needs to be two binary strings one for the whole number and one for the decimal. These values will later need to be decoded into the floating point value these strings represent, this introduces additional time requirements to do this conversion.

Both of these issues make binary encoding difficult to initially design, but once

it is created it uses less memory than floating point encoding and gains more diversification when evolving (discussed further in Section 2.2.3).

The floating point method encodes the data as an array of floats, this increases the memory usage in contrast to binary encoding, but it can easily represent values within any range (both whole values and floating point) with very little effort from the developer. With a large number of parameters, floating point representation excels in execution time as it more accurately represents the format that the data will be used in, therefore not requiring constant decoding that binary encoding requires[20] (decoding is discussed in Section 2.2.2).

When the Genetic Algorithm initially begins the entire population is randomly generated to conform to the chromosome requirements, the population is then used in each of the successive stages of the algorithm and is ultimately modified to better suit the problem by the end of each generation.

2.2.2 Evaluation

For every candidate solution within the population, a fitness value must be found; this is always specific to the problem and often involves some form of external simulation along with an objective function, which defines what is a desirable outcome. The objective function can either be maximized or minimized depending on the requirements of the problem, an example would be to attempt to minimize drag force in a wind tunnel, which would be useful for designing the contours of a plane or car.

In order for any evaluation to occur, the gene data must be readable by the simulation or test environment, therefore it must be decoded for every evaluation step. If this decoding process or the simulation evaluation takes a long time, then the the genetic algorithm process can easily take an excessive amount of time; since they are evaluated for every candidate solution in every generation. The process of binary decoding involves separating a binary string and converting it to their natural integer or real valued forms, floating point encoding does not require any form of decoding as it remains in its natural state throughout the evolution process.

2.2.3 Evolution

Genetic Algorithms replicate the concept of breeding organisms by the use of Crossover, randomized mutations are also included into the evolution process in attempt to keep the population diverse. The more diverse the population, the more likely that an acceptable solution will be encountered by crossover.

Crossover will start by randomly choosing mating pairs of Genomes, the decision as to which individuals are chosen to form a mating pair is based on the results from the Objective Function. The Genome with the most ideal fitness value are more

likely to breed than those that are not ideal, this process is called Roulette Wheel Selection. Producing the next generation of Genomes can be done in many ways, an example of this is One Point Crossover. One Point Crossover will produce two offspring for every mating pair by randomly choosing a position in the Genome to split the data. The first child will be given the data to the left of this point from the first parent and to right of this point from the second parent. The second child will get the opposite, this is shown in Figure 2.5. Once all of the mating pairs have produced offspring, mutation of the new generation occurs.

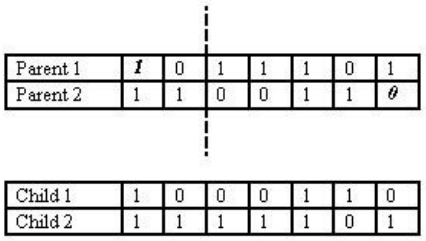


Figure 2.5: Binary Encoded One Point Crossover [2]

Mutation replicates the propensity to have genetic anomalies with each generation of a species, within the algorithm random offspring may mutate by modifying a Gene in its genome. Mutation adds small amounts of variation to the population in the hope that it is beneficial in successive generations. Each gene has a preset chance each generation to mutate after crossover; binary encoding does this by changing individual bit values and floating point encoding will re-randomize the entire gene.

As previously mentioned genetic algorithms excel when the population is diverse, when executing the algorithm diversity will be different between encoding types. Binary encoding does its crossover at the binary level, where every bit in the string could be selected as a crossover point. Mutation also occurs at a binary level, so all changes (if any) will be spread throughout the genome. Floating point representation conducts both mutation and crossover at the gene level, where all mutations change an entire parameter value and crossover splits the array at a random index rather than a random bit. Floating point representation therefore may encounter diversification issues unless the mutation rate is high.

Another useful feature in the evolution stage of the genetic algorithms is elitism; it is used when the population in a new generation has a lower optimal fitness than that of the previous generation. It involves replacing the worst candidate solution in the current population, with the best from the previous generation. This prevents the overall population from deteriorating, so the best overall solution encountered will be available at the end of the optimisation process regardless of any population

changes.

After the Evolution process has finished, the next generation will begin and this will continue until a specified number of generations has elapsed or when the population converges on an optimal solution.

Chapter 3

Related Literature

3.1 Genetic-Fuzzy

Genetic-Fuzzy is a hybrid algorithm that combines Fuzzy Logic (Section 2.1) and Genetic Algorithms (Section 2.2), it provides optimisation of Fuzzy Logic systems using the evolution and natural selection process in Genetic Algorithms. Any fuzzy logic component can be optimised or even the entire system at once, however this may result in a large search space.

This hybrid algorithm removes or at least reduces the need for hand calibrated Fuzzy Logic systems, which is often a long and tedious process. Instead the developer needs to produce test scenarios to evaluate the candidate solutions and a objective function that numerically defines which solutions are better than others. The process of creating test cases and objective function can also be a difficult task, but once these are created any future optimisation or change of requirements will be quickly solved using the previously developed test cases and objective function.

Within this section a few papers that utilize Genetic-Fuzzy are investigated, specifically the inverted pendulum problem (3.1.1) and controlling pH level (3.1.2).

3.1.1 Integrating Design Stages of Fuzzy Systems using Genetic Algorithms

In 1993 Michael Lee and Hideyuki Takagi from the University of California explored the integration of Genetic Algorithms and Fuzzy Logic, to optimise a controller for Inverted Pendulum Problem [3]. The Inverted Pendulum Problem involves balancing a pole on a moveable cart, this is shown in Figure 3.1. In their simulation this cart may move infinitely to the left or right, therefore it may not be feasible to apply their controller to a real life situation where the cart is on a limited surface area. However unless large forces are acting on the pendulum this controller will work well anywhere.

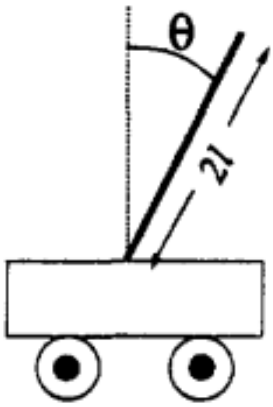


Figure 3.1: Inverted Pendulum Cart [3]

The Fuzzy System used has two input parameters representing:

1. Angle of the pole in degrees (shown in Figure 3.1 as θ)
2. Angular velocity of the pole (degrees per second)

The final output from the Fuzzy System is a force that should be applied to the cart, the overall system architecture is shown in Figure 3.2.

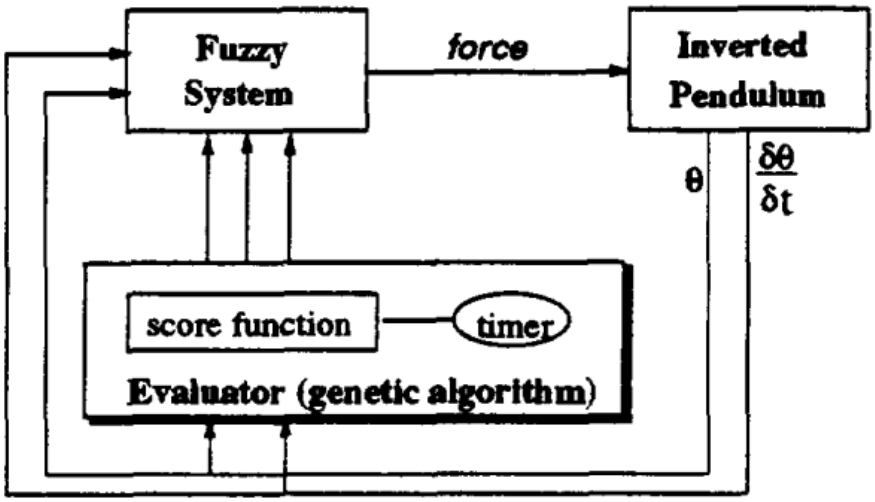


Figure 3.2: Inverted Pendulum System Diagram [3]

The Genetic Algorithm component (Figure 3.2) binary encodes the entire Fuzzy System into a Genome. In this encoding scheme there are ten triangle membership functions for each of the input parameters, and 100 rule outputs. This encoding scheme results in a Genome with 360 Genes containing 8 bits of data each, a total of 2880 bits to represent the entire Genome (shown in Figure 3.3 and Figure 3.4). Since

the fuzzy system only requires integer values and only small precision, using binary encoding would be better suited to the application than floating point encoding. The Genetic Operators involved in the research implementation are One Point Crossover and Bit String Mutation.

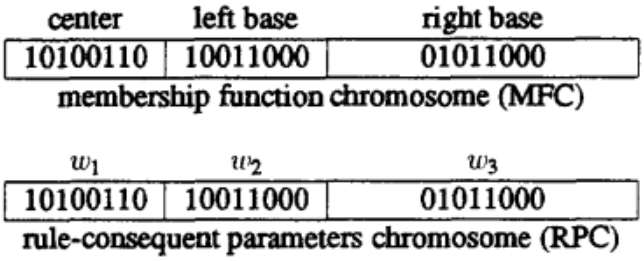


Figure 3.3: Chromosome substructures [3]

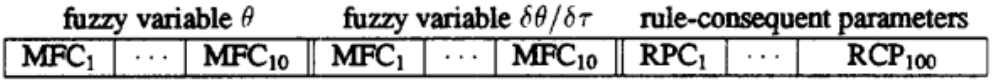


Figure 3.4: Entire chromosome [3]

The raw score is based on the termination criteria of the simulation, which are to either successfully balance the pendulum, the pendulum falls over or when the simulation time expires. A significant reward is given to a Genome if it successfully balances the pendulum earlier in the simulation process, likewise a large penalty is given if the pendulum falls (Figure 3.5).

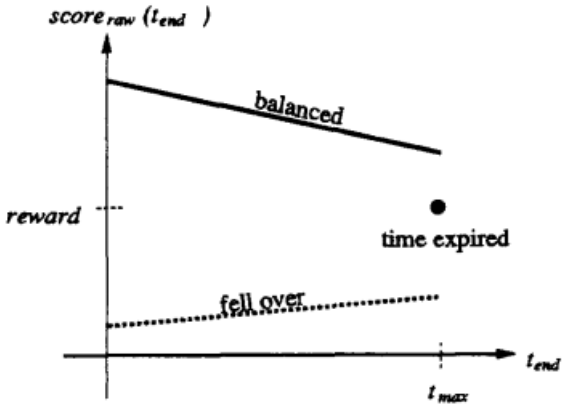


Figure 3.5: Raw Score for the objective function[3]

Additional to the raw score, the objective function strives to minimise the angle of the pendulum at each time step of the simulation and the overall number of rules

in the Fuzzy System. Minimising the number of rules is very useful for robots that have limited memory storage or processing power, as would be the case with the cart used in this research. The final objective function is shown in Figure 3.6.

$$score(t_{end}) = \frac{\left(score_{raw}(t_{end}) + c\sum_{0}^{t_{end}} |\theta_t|\right)}{\text{number of rules} + \text{offset}_{\text{rules}}}$$

Figure 3.6: Inverted Pendulum Objective Function [3]

3.1.2 Fuzzy Control of pH Using Genetic Algorithms

In the same year as the above research paper, Charles Karr and Edward Gentry used Genetic Algorithms to adapt a pre-existing Fuzzy Logic Controller for maintaining a desired pH level [4]. This controller is produced with the primary purpose of real world application. The pH environment is modified by 5 separate pumps, two of which are Control Streams where their throughput is maintained by the Fuzzy Logic Controller and the remaining three are external influences. Figure 3.7 shows these pH influences.

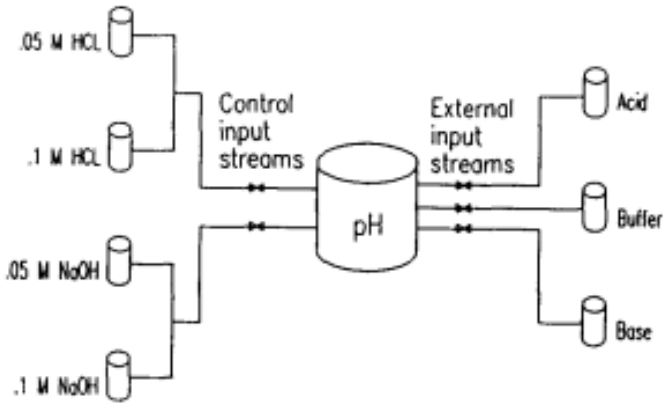


Figure 3.7: System wide pH Influences [4]

As mentioned the Fuzzy Logic Controller will maintain the Control Input Streams shown in Figure 3.7, these are the outputs from the Fuzzy Controller. The two inputs to the Controller is the pH Error (desired actual) and the rate of change of that error, Figure 3.8 shows these inputs and outputs. In this figure pH denotes the pH error and not the actual pH, which was not initially clear when reading this paper.

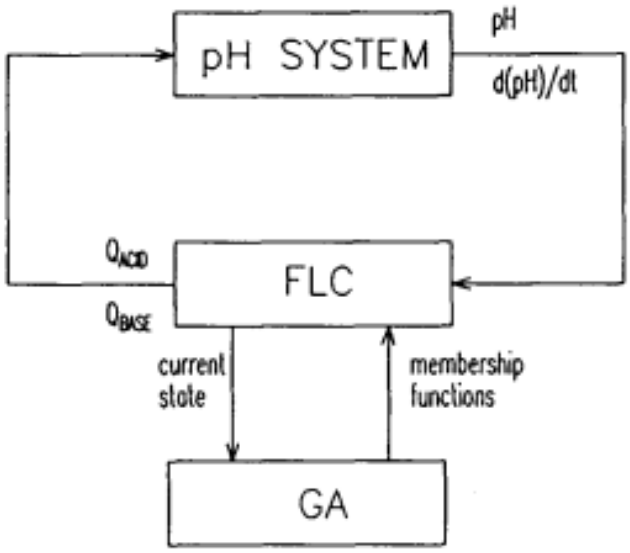


Figure 3.8: Genetic-Fuzzy interfacing with the pH environment [4]

Optimisation of the Fuzzy Logic Controller is solely done on the Trapezoidal Membership Functions, each parameter is represented by seven bits in the binary encoded Genome, which are subsequently decoded to a floating point value within a user defined range. This method of encoding has a very low level of precision, but for the given task it is acceptable. By solely using 7 bits per Membership Function the overall length of the binary string is considerably reduced, but at the cost of precision. To allow for Genetic Optimisation, the pH environment is simulated and then the resulting controller is tested on the real implementation. The overall resulting controller had significant improvements over the pre-existing controller, however there remains a small amount of 'jitter' in the output graph shown in Figure 3.9. This variable output when trying to stabilise the pH could be reduced by increasing the precision of each parameter, this would increase the required execution time for the Genetic Algorithm. If the optimisation process is only going to be executed once for this given set-up, there is no reason why it shouldn't be as accurate as possible. It would also be interesting to see this controller tested in situations where the desired pH level gradually increases or decreases.

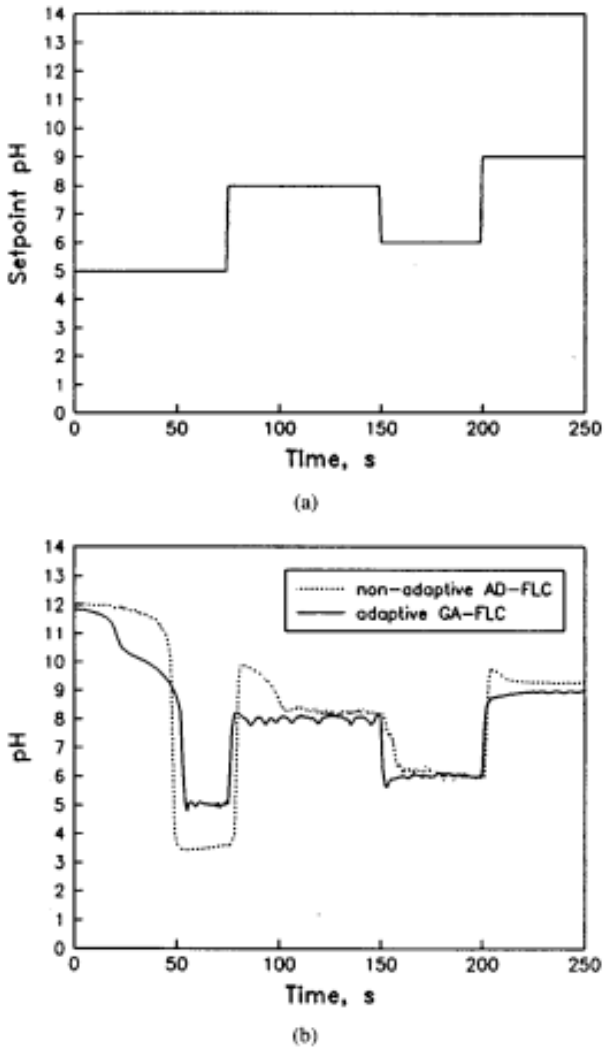


Figure 3.9: Final Output Data of the pH Controller (a) desired, (b) actual [4]

3.2 Genetic Network Programming

Genetic Network Programming (GNP) is a directed graph based algorithm where nodes interlink each other to define a sequence of steps that should be executed to achieve the problem it is designed for. There are three node base types that make up a GNP; the start node is where the network begins, processing nodes which are a form of action and judgement nodes which are decision making nodes which branch out to other nodes.

With repeated executions of GNP it will retain the knowledge of its previous node and execute from that point, this allows it to implicitly remember the current and previous states. The entire network and its nodes are optimised using genetic algorithms, this is further discussed in Chapter 6.

GNP has not been applied to many multi-behavioural robot control applications,

despite being proposed in 2000 by Katagiri, Hirasawa and Hu[5]. This algorithm has however been applied to agent based systems[5, 6], double-deck elevator control systems[7] and some robotic applications[9, 10] to name a few.

3.2.1 Genetic network programming - application to intelligent agents

Katagiri et al.[5] proposed the initial Genetic Network Programming architecture in this paper, along with a small problem to prove its functionality and real world applications.

The problem case involves using intelligent agents to navigate a two dimensional grid that contains obstacles and push tiles into holes, this is a simple task but would be useful to test the basic functionality of the algorithm, proving that GNP can be optimised and that it can solve problems. The overall objective of the agents is to push the tiles into the holes as soon as they can, each candidate solution is partially rewarded for getting close to the hole but not necessarily dropping the tile into the hole and lastly is rewarded for finishing the task with the least number of steps.

Once a tile fills a hole then both will disappear and the location is treated like an ordinary floor. The agent may not push a tile when either a barrier or another agent is in the way, this means that the task can be failed if any tile gets stuck in a corner or between agents. Figure 3.10 shows an example tile world along with a key that is used in the training process.

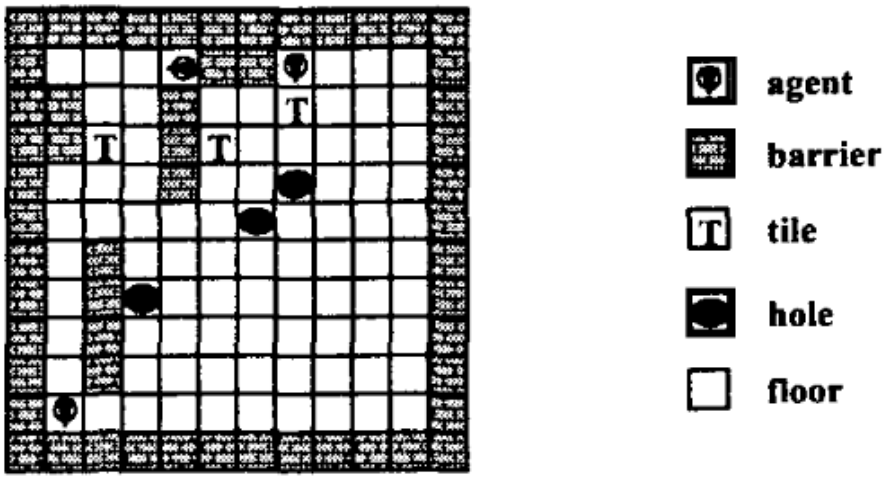


Figure 3.10: Example Tile World[5]

Two approaches were used in the test cases, absolute process which defines agent movements as general directions relative to the world and relative process which considers the problem from the agents perspective, where they may move forward or

rotate. Absolute Process retrieves cell information based on North, South, East and West and moves the agent in a similar manner. Relative Process retrieves cell information from behind, ahead, left and right and may only move the agent forwards, but introduces a rotate left and rotate right. These Judgement and Processing nodes for each perspective are shown in Figures 3.11 and 3.12 respectively.

Judge	1 North cell information
	2 East cell information
	3 West cell information
	4 South cell information
	5 The direction of the nearest tile
	6 The direction of the nearest hole
	7 The direction of the second nearest tile
	8 The direction of the nearest hole from the nearest tile
Process	1 Move to North
	2 Move to East
	3 Move to West
	4 Move to South
	5 Stay here

Figure 3.11: Absolute Process Nodes[5]

Judge	1 Ahead cell information
	2 Right cell information
	3 Left cell information
	4 Behind cell information
	5 The direction of the nearest tile
	6 The direction of the nearest hole
	7 The direction of the second nearest tile
	8 The direction of the nearest hole from the nearest tile
Process	1 Move Ahead
	2 Turn right by 90 degree
	3 Turn Left by 90 degree
	4 Stay here

Figure 3.12: Relative Process Nodes[5]

This paper made three findings which incited further research into the GNP algorithm over the next few years; the first was that the more instances of nodes in a training session the more likely that a better result will be found, secondly using the relative process nodes there is a significant improvement in fitness over that of absolute process as the same problem can be solved with a smaller number of nodes and lastly that GNP is useful for generalizing a problem based on its past behaviour, therefore it would be useful in other applications.

The GNP training process may yield better results overall if additional node types were included to determine if a tile can be pushed in a specific direction, this would prevent agents repeatedly attempting to push a tile into a barrier or another agent as they would with the current nodes and force them to make alternative routes. It would not benefit the scenario where a tile is stuck in a corner, but it would help with barriers that are in the way of the nearest hole.

3.2.2 Performance of genetic network programming for learning agents on perceptual aliasing problem

Genetic Network Programming has been applied to the Perceptual Aliasing problem, where a robot can not differentiate between locations in a maze solely using sensor data. This sensor data could be camera input or sonar to name a few, it would be very common for a robot to encounter this perceptual aliasing in a maze or office layout.

Murata et al. proposed solving this problem with GNP [6] as a comparison against another algorithm that is regularly used as a solution for this perceptual

aliasing problem (PAP), called Adaptive Genetic-Programming Automata (AGPA) which is a tree based structure that has a series of branches and conducts an action at a terminal node (shown in Figure 3.13). AGPA has flaws where it will generate too many rules for each solution, the theory was that GNP could solve the same problem with a much smaller number of rules.

This paper defines a rule as an IF/THEN statement, both AGPA and GNP can be unrolled to these rules by following the node structure until either a terminal node or processing node is reached.

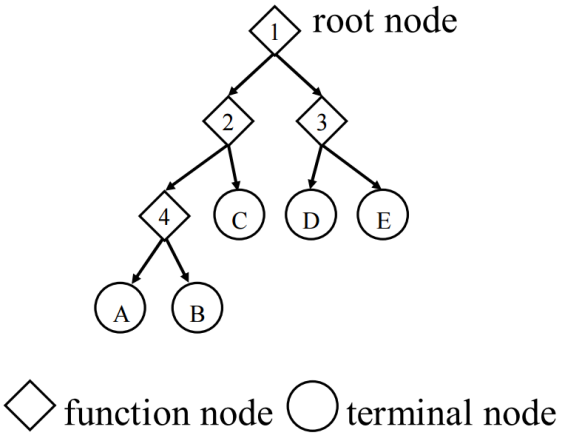


Figure 3.13: Adaptive Genetic-Programming Automata example [6]

To prove their theory both AGPA and GNP are applied to two mazes which regularly encounter PAP scenarios, their results are later compared in terms of average fitness and number of rules used in the solution. One of the mazes used in this test is shown in Figure 3.14, 'S' is the robot start location, 'G' is the goal location and each circle is a location where the PAP is encountered.

To evaluate the fitness of this simulation the proposed method is to assign points to each tile and when the robot begins its action on a tile, it is awarded points from that particular tile (shown in Figure 3.15). This point system increments as the robot gets closer to the goal and when reaching the goal, it continues to be awarded these points. While this fitness evaluation may separate those candidate solutions that can reach the goal in the most efficient manner, it neglects to include the secondary objective where they should attempt to minimize the number of rules. This could be added by simply adding $1/ruleNumber$ to the end of the fitness function, this would continue to encourage robots to reach the objective as efficiently as possible, but would further separate these 'ideal solutions'.

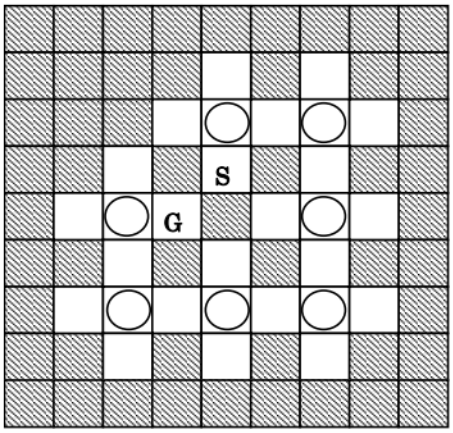


Figure 3.14: Simulated Perceptual Aliasing Maze [6]

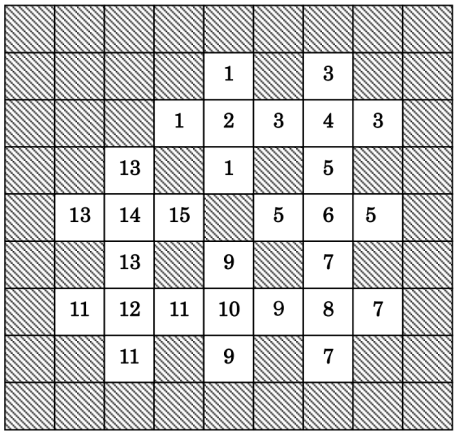


Figure 3.15: Simulated Perceptual Aliasing Maze Point System [6]

The paper does not state the type of judgement and processing nodes used in this experiment (nor function nodes and terminal nodes for AGPA), it is likely that they only use these nodes to evaluate sensor data to determine if there is a wall present in each direction and to move in the four directions. Without additional environmental data or even the inclusion of heuristics, the absolute best solution will only be able to navigate the maze it is trained on and would severely struggle with any other scenario which would likely affect its real life applicability.

The results from this paper suggests that GNP outperforms tree based structures like AGPA in terms of both number of rules and the average fitness of the entire population. This is due to GNP's recursive design where loops are implicitly created, this significantly reduces the need for any duplicated nodes and allows it to represent more complicated behaviours.

3.2.3 A Double-Deck Elevator Group Supervisory Control System Using Genetic Network Programming

In 2008 Hirasawa et al.[7] proposed a solution to a Double-Deck Elevator Control problem using Genetic Network Programming. This problem involves optimising a building elevator that has two cages in a single shaft, this elevator design reduces the amount of floor area required for efficient transportation in high-rise buildings.

However this design comes with a few case specific considerations that would deteriorate the transport throughput. If both cages are to release passengers at the same floor only one cage could be serviced at time, leaving the passengers in the other cage to wait until the first cage has been emptied, the passengers in the second cage may experience stress if they have not been in a double-deck elevator before or if they are running late and waiting for another cage to empty. Another consideration is the number of passengers in each cage, where those with a large number should probably get first priority.

Figure 3.16 shows the double-deck elevator control problem in a simulation environment along with how it would relate to the resulting GNP controller.

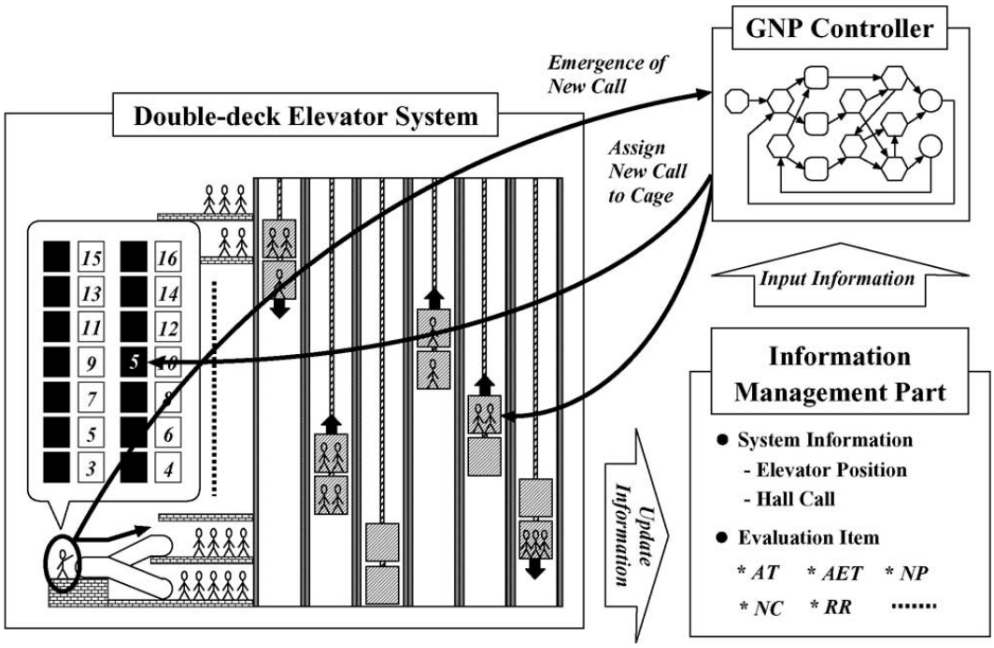


Figure 3.16: Proposed structure for a double-deck elevator system [7]

Hirasawa et al. approach this problem using 28 node types designed specifically for this problem, they include judgement nodes to determine the destination floor, directions of the destination floor and nodes to determine which elevator cage should reach the destination floor.

These nodes and the GNP structure are optimised to complete a few select objectives, specifically minimizing the wait time for passengers, maintaining a comfortable ride for passengers and eliminating loops within the network, these loops would prevent the solution from returning to the start state and may effectively cut out required logic.

Each candidate solution is evaluated by randomly simulating arrival times and destination floor of 300 to 4500 people per hour, to give a broad test of possible situations. After the population of 300 candidate solutions are evaluated for 300 generations, the GNP solution reduced the average service times (wait times + transport times) to 4.8 seconds, in the situation where 3600 people need transportation in an hour.

This research shows that while GNP can be applied to complicated tasks, it uses a large amount of custom nodes that would require a lengthy amount of programming time, some of the elements could have been independently optimised using artificial intelligence nodes. Specific examples, such as speeding up the elevator and decelerating could be done with Fuzzy Logic, or Reinforcement Learning could be used to determine which elevator cages should answer calls. Using Artificial Intelligence nodes the average waiting time may further decrease, this would also reduce the amount of programming time for problem specific nodes. The idea to include conditions that would eliminate loops from the population would likely be useful for any application to prevent the algorithm getting stuck.

3.3 Robot Control

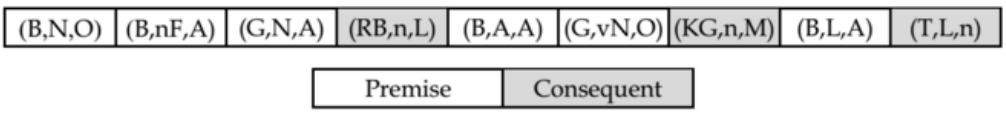
Robotics can be used for a variety of purposes, they can be used to replace limbs, manufacture goods or entertainment purposes, all forms of robotics require a controller. These controllers state what actions should be taken and how they should be achieved, robotics are becoming increasingly complicated as their requirements expand.

This section describes a few approaches at solving robotic control systems that are relevant to this research, they include using Fuzzy Logic to determine actions a robot should take[8] and two different applications using GNP to control robot systems[9, 10].

3.3.1 Evolving Fuzzy Rules for Goal-Scoring Behaviour in Robot Soccer

The book chapter entitled Evolving Fuzzy Rules for Goal-Scoring Behaviour in Robot Soccer by Jeff Riley [8], utilises Genetic Algorithms to optimise a decision

based rule set for implementing Goal-Scoring Behaviour. The input to this Fuzzy Rule set is the Degree of Membership for the Distance and Direction of both the Ball and Goal locations, based on pre-compiled Membership Functions and environmental data. The output from the Fuzzy System is the Action to be taken and a crisp output that denotes either power or direction, therefore the behavioural decisions are not concerned about the underlying action implementation. The process of optimising for this behaviour is done by utilising a variable length, Messy-coded gene encoding scheme[21], which defines the entire rule set and its consequent outputs. Each rule contains one or more Premise and a Consequent, as shown by the colour coding in Figure 3.17. Note that the lower case 'n' within the premise, stands for the "not" operator, while the 'n' in the consequent is a placeholder for unnecessary data. An example of this placeholder is shown in Rule 1, the rule does not require a direction (second parameter) as the required action in the first parameter (run towards the ball) implicitly defines the direction.



Rule 1: if Ball is Near or Ball is not Far and Goal is Near then RunTowardBall Low
Rule 2: if Ball is At and Goal is VeryNear then KickTowardGoal MediumPower
Rule 3: if Ball is Left then Turn Left

Figure 3.17: Chromosome Encoding and Decoded Fuzzy Rules [8]

Each Premise is made up of an Object, Qualifier, Distance or Direction and a Connector, the range of possible values for each parameter are shown in Figure 3.18.

Object:	{ BALL, GOAL }
Qualifier:	{ IS, IS NOT }
Distance:	{ AT, VERYNEAR, NEAR, SLIGHTLYNEAR, MEDIUMDISTANT, SLIGHTLYFAR, FAR, VERYFAR }
Direction:	{ LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }
Connector:	{ AND, OR }

Figure 3.18: Premise Components [8]

Similarly each Consequent contains an Action, Direction and Power (shown in 3.19), the use of Direction or Power will be based on the type of action that is used, for example in Rule 3 of Figure 3.17 there is no need for a power parameter if the robot is turning left. When a parameter is unused it will be represented as null.

Action: { TURN, DASH, KICK, RUNTOWARDGOAL, RUNTOWARDBALL, GOTOBALL, KICKTOWARDGOAL, DRIBBLETOWARDGOAL, DRIBBLE, DONOTHING }

Direction: { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }

Power: { VERYLOW, LOW, SLIGHTLYLOW, MEDIUMPOWER, SLIGHTLYHIGH, HIGH, VERYHIGH }

Figure 3.19: Consequent Components [8]

This research uses a variable length encoding scheme to allow the premise and consequent to shift anywhere among the chromosome, to facilitate this Cut and Splice crossover is used, this is similar to One Point Crossover except each parent has its own separation point resulting in different length children.

Utilising the RoboCup Soccer Simulator the Goal-Scoring behaviour is optimised using the formula in Figure 3.20. The objective function is conditionally split into sub parts, the primary objective is to get as many soccer goals as possible, if this cannot be attained using the current candidate solution then the distance to the goal is minimised. This is to increase the amount of diversity in the objective function and in the hope that the successive population will successfully get a goal. Since variable length encoding is used it would be useful to add components to this fitness function, that attempt to minimize the length of the candidate solution otherwise the controller optimisation could result in huge fuzzy system.

$$f = \begin{cases} \begin{cases} 1.0 & , kicks = 0 \\ 0.5 + \dfrac{dist}{2.0 \times fieldLen} & , kicks > 0 \end{cases} & , goals = 0 \\ \dfrac{1.0}{2.0 \times goals} & , goals > 0 \end{cases}$$

where

$goals$ = the number of goals scored by the player during the trial
$kicks$ = the number of times the player kicked the ball during the trial
$dist$ = the minimum distance of the ball to the goal during the trial
$fieldLen$ = the length of the field

Figure 3.20: Goal Scoring Objective Function [8]

This chapter contains a lot of useful information regarding robot controllers. In particular the use of a variable length genome could be used as a future extension to this research in order to try minimize the number of nodes in a GNP system, rewarding those systems that remove any unnecessary nodes. The paper successfully instructs the robot on actions that should be taken using fuzzy rules, Genetic

Network Programming would also be suitable for solving this problem if nodes were created to encompass all possibilities of both the premise and consequent components.

3.3.2 Using Genetic Network Programming to Get Comprehensible Control Rules for Real Robots

Murata and Okada applied a Genetic Network Program to a Sony AIBO ERS-7M2 robot[9] with the objective of reaching a ball as quickly as possible. The robot contains dog-like features, including four legs, tail and ears all of which can move. This robot has seen many entertainment uses, but has seen a lot of success in the RoboCup Four-Legged League between the years 1999 and 2008. Figure 3.21 shows this robot and its ball.

This robot contains only a select few sensors most of which are touch based, therefore the robot unfortunately lacks the ability to sense distance to the ball. It can however detect the direction of the ball, which is used as the primary input within this research.

Figure 3.21: Sony AIBO Robot and a Ball [9]

In order to allow efficient GNP optimization a basic simulation environment was used, which defines the robot as a circle with a 15 unit radius, with the target ball being a 10 unit radius circle, shown in Figure 3.22. Each candidate solution underwent 100 randomly generated trials, with randomized starting positions and rotations for the robot and ball.

Three processing nodes were used which allow the robot to move forward, left and right. Only one judgement node was used within the design which specifically determines the direction of the ball in six directions. These directions are either 'left', 'right', 'center' which conforms to the robot's field of vision, then there is

'lost' which is anywhere outside of the previous three directions. This lost state is further separated into two more states 'lost left' and 'lost right' which occur when the ball was previously located in a specific direction, but then moves into the lost area.

These lost states would be very useful for when the ball is moving past the robot, as it will retain a direction in which it should move to relocate the ball. The states of the robot vision and also the one judgement node is shown in Figure 3.23.

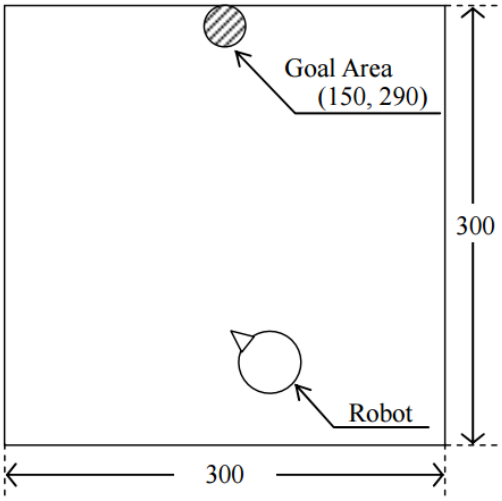


Figure 3.22: AIBO Simulation [9]

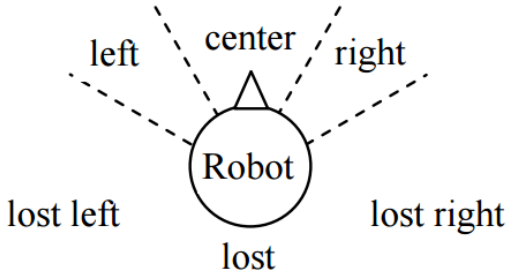


Figure 3.23: AIBO Proposed States [9]

For this problem a rather sophisticated fitness function is used, which incorporates all features required by the problem. It attempts to minimize the number of movements and turns that the robot must undergo, therefore implicitly minimizing the time it takes to reach the ball. A penalty is included when the robot loses sight of the ball and a set reward for seeing the ball at least once, a large reward is given to any controller that manages to reach the ball. These rewards and penalties are summed and then averaged over the 100 randomized trials. This paper also includes another component to the fitness function which minimizes the overall number of nodes used in the GNP, this is especially useful considering all processing will be done on the robot itself and it has limited processing power.

From a set of ten GNP training sessions the best five controllers are then applied to the real robot in four different scenarios, to test the states of the judgement node. The best system overall is shown in Figure 3.24, the lower six arrows on the key denote specific transitions from the judgement node, the acronyms within the figure are as follows:

- BW - Ball Where, is a judgement node that states the direction of the ball in one of six states, which are listed in the figure

- L - Left, a processing node that turns the robot to the left
- R - Right, processing node that turns the robot right.
- MF - Move Forward, another processing node that simply moves the robot forward

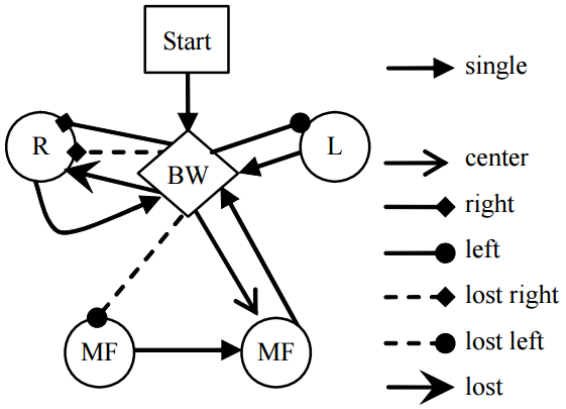


Figure 3.24: AIBO Best Trained Network [9]

This overall system is what would be expected from a controller that simply pursues a ball, with one exception; when the ball is 'lost left' the robot will undergo two forward movements. This is totally against what should happen in this scenario it would be better to have the robot move left, as left was the last known direction that the ball was in. This likely occurred due to the simulation randomization, randomization is typically a good feature to use when diversifying a problem, but something this simple it would be detrimental. In this problem, only four different base types of node are used and the only optimization involved is the node types and their connections and only six states that actually matter in the simulation, better results would have been obtained from hard coding the simulation states to test these six problems.

A more complicated problem would have tested the GNP functionality more thoroughly, this could be finding a ball and then kicking it to a target destination. It is much more difficult, but could have been done in the same simulation environment and would require more than one judgement node.

3.3.3 Genetic Network Programming with Fuzzy Reinforcement Learning Nodes for Multi-Behavioural Robot Control

In a previous year Wenhan Wang at Massey University conducted research into GNP with Reinforcement Learning nodes for the Target Pursuit with Wall Avoidance behaviour in robot soccer[10]. It is based on the same platform as this research however his simulation environment does not use realistic physics, it lacks proper friction with the field and the actual shape of the robot used is not accurate. Due to these inaccuracies any controller built from the simulation will not accurately represent the real platform, therefore they could not be used in any situation other than the simulation. His soccer field is shown in Figure 3.25.

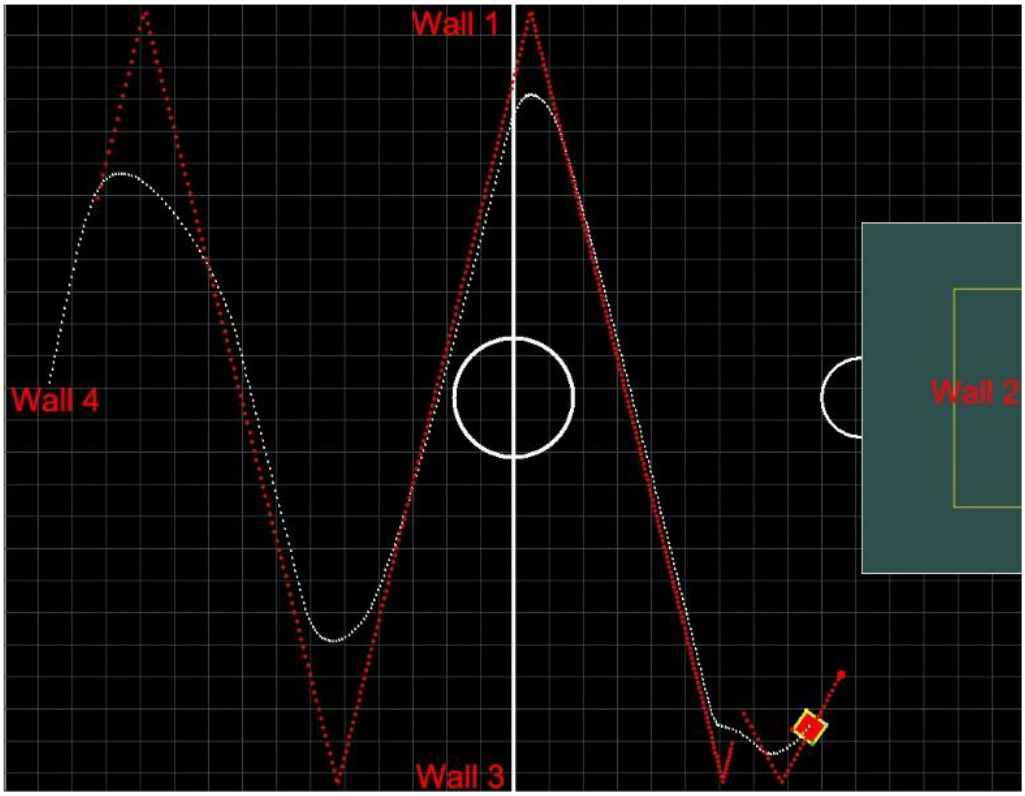


Figure 3.25: Soccer Field [10]

The use of reinforcement learning nodes would significantly increase the likelihood of reaching an acceptable solution (defined by the objective function) as it provides online learning, meaning that it will continue to optimise during the simulation process. In contrast GNP without reinforcement learning only conducts its optimisation process after a simulation has been executed, including reinforcement learning nodes would provide this functionality without affecting the overall GNP architecture.

Wang uses a very complicated fitness function to conduct target pursuit with wall avoidance behaviour, with very rigid and large conditional statements attached to rewards. One of which requests specific speeds depending on how far away from the wall the robot is. These fitness functions can be considerably decreased in size especially considering that a penalty for the wall collisions is not actually applied, this alone would make a lot of the fitness function redundant. It was mentioned that additional objectives would make the fitness function too complicated, this is true for all forms of optimisation a potential solution is allowing modularization of the problem, if the problem is split into smaller tasks then they are easier to achieve for both humans and genetic optimization.

The overall approach at wall avoidance could be improved Wang prioritizes wall avoidance over everything else, therefore if the ball is near to the wall the robot will never actually get close to it, this is visible by looking at the robot and ball paths in Figure 3.26. A solution that prioritizes the target pursuit would significantly increase the amount of time the robot possesses the ball, as the robot would be able to move closer to the wall and push the ball back into the playing field.

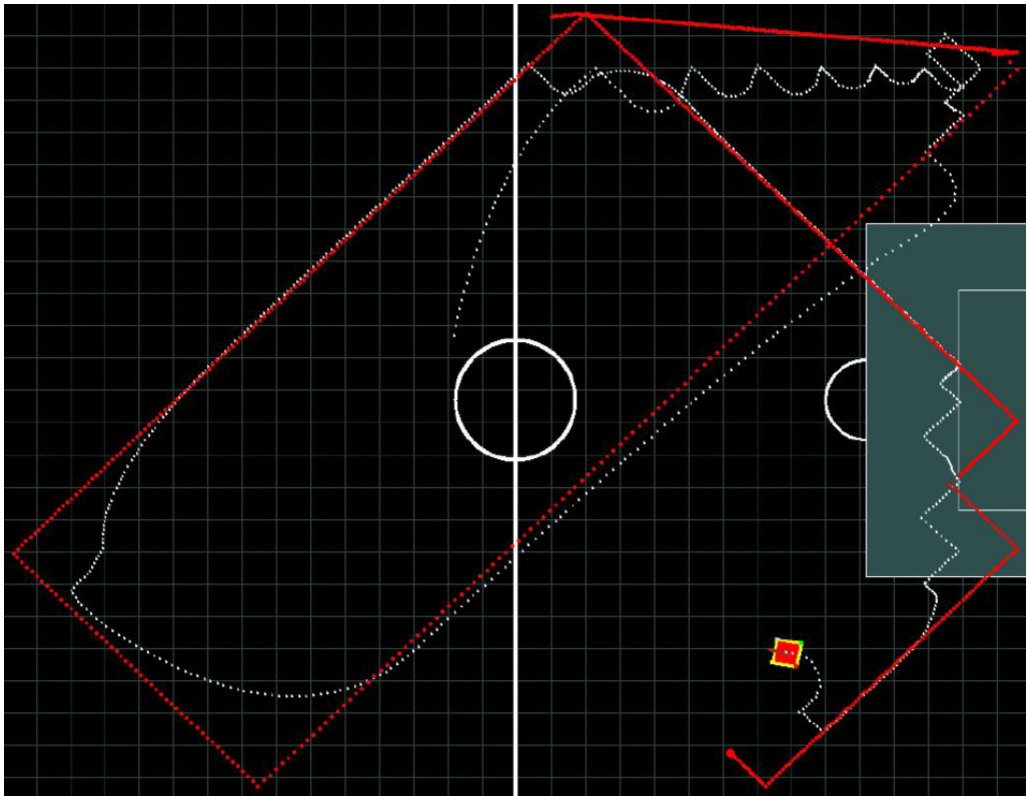


Figure 3.26: Trained Target Pursuit with Wall Avoidance [10]

The inclusion of Hill Climbing is another useful feature as it would further improve the results after genetic evolution, since genetic algorithms never guarantee a perfect result nor a local maximum result, using Hill Climbing in conjunction with

GA's would guarantee finding a local maximum. Hill Climbing was utilised in this research and is further discussed in Sections 6.1.3 and 6.2.3.

This paper has a very interesting approach at multi-behavioural controllers and provides a lot of insight on many difficulties that should be considered in any further research within this platform.

3.3.4 Reinforcement Learning Approach to AIBO Robot's Decision Making Process in Robosoccer's Goal Keeper Problem

Using the AIBO robots in 2011, Mukherjee et al. developed a Reinforcement Learning controller for the goal keeper problem[11]. Their approach is focused on the movement decision making of the robot, they use predefined functions for all movement and the Reinforcement Learning controller will only decide which direction the robot should move in, or if it should remain stationary.

In their solution the field is separated into seven potential states for the ball location. It should be noted that within the paper states Five and Six are exactly the same, it is likely that they intended one of these states to be the near right corner. Each of these states are defined below:

- State One: Ball is at the far left
- State Two: Ball is at the far right
- State Three: Ball is at the far middle
- State Four: Ball is in front and heading towards the goal keeper
- State Five: Ball is near the left corner
- State Six: Ball is near the left corner
- State Seven: Ball is near the middle

These individual states are used within the Reinforcement Learning algorithm to allow it to learn the optimal movements based on the ball location. Three movement types are considered within the algorithm, that is move left, move right and remain stationary. They do not consider the location of the goal keeper itself within the algorithm and it is likely that they assume that the robot is always in the centre of the goal area. This would be detrimental for situations where the robot moves to one side to protect the goal, then without giving the goal keeper enough time to reposition itself, an attacker shoots at the other side.

The controller was optimised on a set of one attacker configurations and managed to achieve a 80.7% success rate in defending the goal from a single attacker (shown in Figure 3.27). This controller achieved a slightly better performance compared to a hand calibrated system that they aim to replace, this is a significant result considering that the controller began with zero knowledge of the problem.

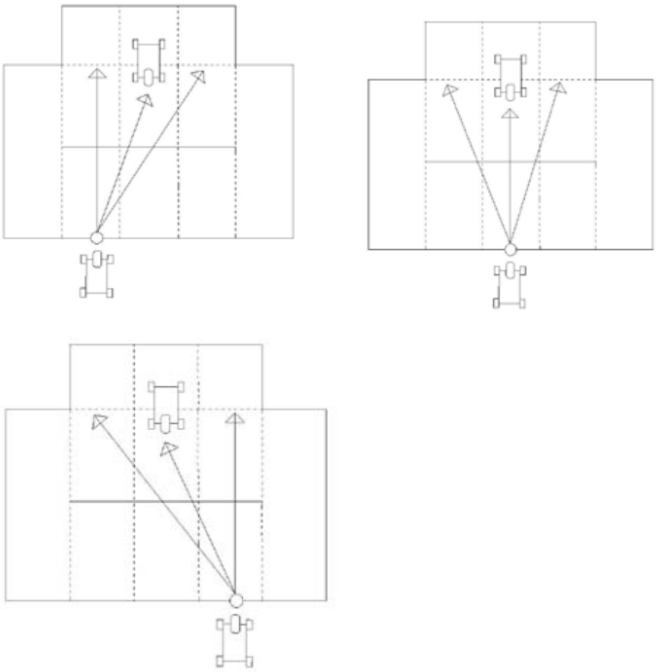


Figure 3.27: Single goal attacker layouts [11]

To further test the controller optimised using the single goal attacker, it is tested on multiple goal attacker configurations where an attacker will pass the ball to another attacker that is located near the goal. Figure 3.28 depicts the four additional configurations that this controller is tested on.

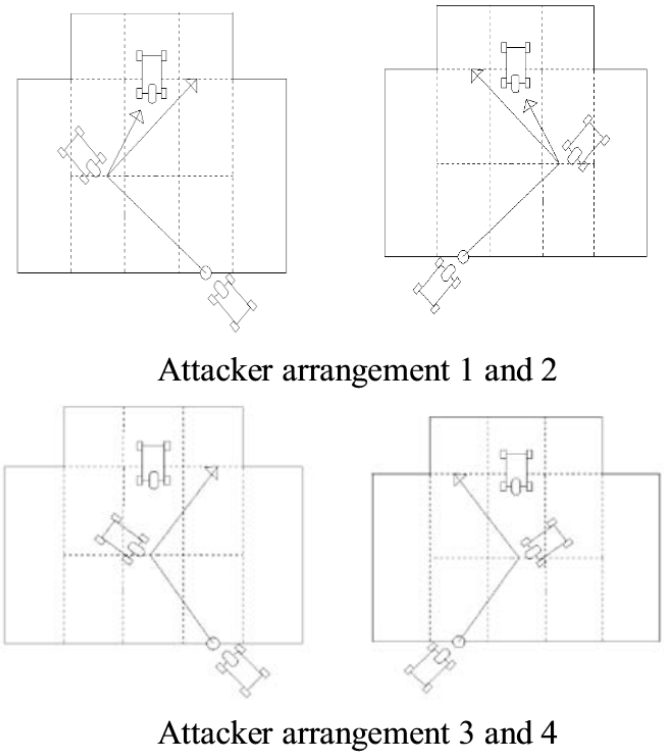


Figure 3.28: Multiple goal attacker layouts [11]

Mukherjee et al. achieved an overall success of 77.5% for the goal keeper problem using Reinforcement Learning with seven states. The overall success of the controller would be limited by the movement capabilities of the individual movement functions, as they may not move in an optimal manner. This controller may be further improved with additional states and considering the position of the goal keeper within the goal keeper may further improve results.

3.4 Summary

In this chapter many diverse papers relating to this research have been discussed and evaluated, they all directly relate to the tasks this research hopes to achieve or are using Genetic Network Programming as the primary algorithm.

If Fuzzy Logic nodes are incorporated into a GNP network and that network optimises those nodes in some way, then that task is inherently Genetic-Fuzzy therefore understanding how Genetic-Fuzzy can be optimised is a useful addition to this research. Lee and Takagi[3] proposed using a binary encoded string to represent each point in the membership functions and the rule system. While binary encoding would be useful for small systems, this research aims to develop an expandable GNP network which is only limited by computer hardware. Therefore an encod-

ing scheme will need to be created for every parameter type, using floating point encoding would be much easier to implement.

It is proven that GNP can adapt to highly complicated dynamic environments like the double-deck elevator control system[7] and that the algorithm can be applied to real robots outside of a simulation environment[9]. It can be used to control the overall actions of agents[5, 6] and can use other artificial intelligent nodes to complete control based tasks[10]. Therefore if GNP is powerful enough to do each of these tasks individually and excel in the respective roles, then it should also be able to tie these components together to make a single controller. This controller would not only instruct the robot on how to move, but also control the overall role as an agent. This task would be far too complicated to optimise in one session as the objective function will become excessively large as seen in Wang's research[10]. Thus the objective of this research is to modularize the GNP architecture, so that components can be trained individually and then be bound together to complete complicated behaviours.

Lastly it is proven that the goal keeping behaviour can be solved with some success using Reinforcement Learning[11], therefore it is likely that the same behaviour can be solved with other algorithms. Using a Reinforcement Learning node within a GNP would likely result in a similar or better outcome compared to what is achieved using Reinforcement Learning alone.

Chapter 4

Software Architecture and Implementation

Within this chapter the various external libraries are discussed, with examples of their usage and any specific considerations that were made. Towards the end of the chapter the application design is briefly discussed and has diagrams to show how each section relate.

4.1 Artificial Intelligence Libraries

Two external Artificial Intelligence libraries are used within this research, by using external libraries it saves a considerable amount of time which would be better used exploring the GNP architecture.

Fuzzylite is the first of these two libraries, which is a free and open source fuzzy logic control library developed by Juan Rada-Vilela[22]. This library is used within a robot processing node in GNP, it therefore allows any user defined fuzzy logic system to interface with the simulation environmental variables and control the robot based upon them.

The second library is GALib written by Matthew Wall[23] at the Massachusetts Institute of Technology, this library is used as a foundation in which the GNP architecture is built. Although other Genetic Algorithms can be used for GNP, this library is definitely easy to understand and has many features.

4.1.1 Fuzzylite

As previously mentioned, the Fuzzylite library provides some of the robot motor control within the GNP controllers. While the library is free and open source, the complimentary application 'QtFuzzyLite' is not. This application graphically displays and modifies fuzzylite systems and allows the user to export to a file or

source code. While fuzzy systems can also be built in code it can get very long and tedious, which is where QtFuzzyLite excels. Shown in Code 4.1 is an excerpt of C++ code which shows the creation of a fuzzylite system with one input and one output, each containing only one term, where typically fuzzy systems contain upwards of three terms per input and output and often multiple inputs and outputs themselves.

Code 4.1: Creating a basic fuzzylite system

```
// Creating a Fuzzylite engine
Engine* engine = new Engine;
engine->setName("TargetPursuitDefault");

// Defining a single input variable
InputVariable* inputVariable = new InputVariable;
inputVariable->setEnabled(true);
inputVariable->setName("TargetAngle");
inputVariable->setRange(-180.000, 180.000);
inputVariable->addTerm(
        new Trapezoid("straightAhead", -27.000,
        -18.000, 18.000, 27.000));
engine->addInputVariable(inputVariable);

// Defining an output variable
OutputVariable* outputVariable = new OutputVariable;
outputVariable->setEnabled(true);
outputVariable->setName("LeftWheel");
outputVariable->setRange(0.000, 1.000);
outputVariable->fuzzyOutput()->setAccumulation(fl::null);
outputVariable->setDefuzzifier(new WeightedSum);
outputVariable->setDefaultValue(1.000);
outputVariable->setLockPreviousOutputValue(false);
outputVariable->setLockOutputValueInRange(true);
outputVariable->addTerm(new Constant("medium", 0.370));
engine->addOutputVariable(outputVariable);

// Describing how the input relates to the output
RuleBlock* ruleBlock = new RuleBlock;
ruleBlock->setEnabled(true);
```

```
ruleBlock->setName("");
ruleBlock->setConjunction(new Minimum);
ruleBlock->setDisjunction(fl::null);
ruleBlock->setActivation(fl::null);
ruleBlock->addRule(fl::Rule::parse(
        "if TargetAngle is straightAhead
        then LeftWheel is medium", engine));
engine->addRuleBlock(ruleBlock);
```

As can be seen by the code excerpt in Code 4.1 manually writing a fuzzy system can be very long but thankfully fuzzylite comes with the ability to import these fuzzy engines, significantly decreasing the amount of code required and making the overall application more adaptable, which was one of the major factors considered when choosing an appropriate Fuzzy Library.

Within the application produced by this research, the initial fuzzy systems would be designed in QtFuzzyLite and then imported as a GNP node for further optimization. The code used within the application to import a fuzzylite engine is shown in Code 4.2, it is very generic code that could be used in any application using fuzzylite.

Code 4.2: Import a fuzzylite system

```
String fileContents = Read Entire File Into the String
fl::Engine* engine = NULL;
fl::Importer* importer = new fl::FllImporter;
try {
        engine = importer->fromString(fileContents);
} catch (fl::Exception& ex) {
        // Error Reading Engine
}
```

Once a fuzzylite engine has been imported or coded via one of the above methods (Code 4.1 or Code 4.2, the only remaining requirement is the actual usage. To use the engine it only takes a few lines of code to set the input variables, process using the engine and then read the final output for use in other areas of the application. A small example of this process is shown in Code 4.3.

Code 4.3: Executing a fuzzylite system

```
// Setting the Input for Target Angle
engine->getInputVariable("targetAngle")->
        setInput(targetAngleValue);
// Process the Fuzzy System
```

```
engine->process();
// Get the Fuzzy Logic Output
OutputSpeedLeftWheel = engine->
        getOutputVariable("leftWheel")->defuzzify();
```

Overall fuzzylite is a well designed library that is simple to use and understand, but even easier along with QtFuzzyLite as a design and debug tool.

4.1.2 GAlib

GAlib is the genetic algorithm library used within this research, written by Matthew Wall at the Massachusetts Institute of Technology in 1995 and was regularly updated until 2007. It contains many different classes and examples to suit almost any situation, while providing the foundations for those situations it doesn't directly support. Simply using one of the pre-built classes is very easy due to the numerous examples, however custom classes can be reasonably complicated to write if one is actually required. Within this research the GARealGenome was used as it is both easy to use and can represent a diverse number of variable types. The fact that GARealGenome can easily represent multiple data types makes it perfect for the GNP architecture, as it requires to be able to represent floating numbers, boolean values and integers.

The most utilized function when creating a GARealGenome in this genetic algorithm library is add, which is used in defining each gene or parameter to be optimized. It takes five parameters to define the allele of each gene. The first two are the Min and the Max values, the third is an Increment value to define the potential values between Min and Max and the last two define whether Min and Max are included or excluded in the list of potential values. In the case where the increment is not evenly spaced between the Min and Max, the maximum value will not be placed into the potential outcomes at all.

The variable staticObjectiveFunction is an actual function within the program which the genetic algorithm will repeatedly execute for every candidate solution within the evolution process. The entire process of creating a GARealGenome is shown in Code 4.4.

Code 4.4: Designing a GAlib chromosome

```
GARealAlleleSetArray* encoding = new GARealAlleleSetArray();
encoding->add(Min,Max,Increment,
        GAAllele::INCLUSIVE,GAAllele::EXCLUSIVE);
// The above line can be repeated to suit the number of
// required Array Encoded parameters.
```

```
GARealGenome genome(*encoding, staticObjectiveFunction);
```

Once the GARealGenome is initialized the Genetic Algorithm process can begin, shown in Code 4.5 the required C++ code to create the Genetic Algorithm engine, along with the definition of the various settings that should be defined before execution. The line that contains minimize and maximize shows the functions that should be run to define whether the ideal situation from the Objective Function is minimizing or maximizing, internally the library simply translates each minimizing fitness value into a maximization when executing.

Code 4.5: Initialize a GAlib genetic algorithm

```
GASimpleGA geneticAlg(genome);
geneticAlg.populationSize(100);
geneticAlg.nGenerations(50);
geneticAlg.pMutation(0.08);
geneticAlg.pCrossover(0.6);
geneticAlg.minimize(); //OR geneticAlg.maximize();
geneticAlg.elitist(gaTrue);
```

Lastly to begin the Genetic Algorithm, the simplistic approach is to execute the evolve() function. This Function will perform the entire Genetic Algorithm process and call the Objective Function when a Genome needs evaluation and once the defined number of generations has been completed then the Function will end. The evolve function is displayed in Code 4.6, along with the method of obtaining the best candidate solution from the library.

Code 4.6: Executing a GAlib genetic algorithm

```
geneticAlg->evolve();

GARealGenome* currentBest = (GARealGenome*)
        geneticAlg.population().best().clone();
```

4.2 Physics Engine

Box2D is the Physics Engine used in this implementation as it is a highly recommended two dimensional physics engine that has been used in many applications, it was created by Erin Catto[24] in 2007 and has been significantly improved since the initial release.

A two dimensional engine is used in this research as the only additional accuracy from having three dimensions is when a ball bounces or when a robot falls over,

both of which are very unlikely to happen within robot soccer. A limitation to using Box2D in a top down view is that objects would be considered floating in mid air; since these objects are considered floating no form of friction will be evaluated. Friction was implemented in the simulation platform and is further discussed in Section 5.2.

When initially using Box2D a b2World object must be created and this is easily done using two lines of code, the first defines the world gravity and the second creates the world. In this application the gravity should be zero in both dimensions as its being used as a top down view, shown in Code 4.7.

Code 4.7: Creating a Box2D world

```
b2Vec2 gravity = new b2Vec2(0,0);
b2World world = new b2World(*gravity);
```

Within the world there are two main types of bodies; these are static bodies which will never move and dynamic bodies which can move. They are both created in a similar process with a select few differences; for example, a static body would not require density or restitution as they will never move.

Code 4.8 shows a box being created as a static object and Code 4.9 shows a circle being created as a dynamic object. The code example for the static box is one section of code used to create the outside wall for the soccer field and the dynamic circle is the code used for creating the ball in the simulation.

Code 4.8: Creating a static body for Box2D

```
// Define the shape of the static body, this case is a box
b2PolygonShape* boxDef = new b2PolygonShape();
boxDef->SetAsBox(boxHeight, boxWidth, boxCenter, angle);

// Container to hold the shape definition
b2FixtureDef fixtureDef;
fixtureDef.shape = boxDef;

// Define the body as a static object
b2BodyDef* bodyDef = new b2BodyDef();
bodyDef->type = b2_staticBody;

// Create Static Body
body = world->CreateBody(bodyDef);
// Attach the Shape to the body
body->CreateFixture(&fixtureDef);
```

Code 4.9: Creating a dynamic body for Box2D

```
// Define the shape of the dynamic body, this is a circle
b2CircleShape* circleDef = new b2CircleShape();
circleDef->m_radius = 21.335;

// Container to hold the shape definition
b2FixtureDef fixtureDef;
fixtureDef.shape = circleDef;

// set the density of the ball and restitution
fixtureDef.density = 1.291;
fixtureDef.restitution = .9;

// Define the body as a dynamic object
b2BodyDef* bodyDef = new b2BodyDef();
bodyDef->type = b2_dynamicBody;

bodyDef->position.Set(500,500);
bodyDef->angle = 0;

// Create Dynamic Body
body = world->CreateBody(bodyDef);
// Attach the shape to the body
body->CreateFixture(&fixtureDef);
```

Once all the bodies have been added to the world, the physics can be emulated using the world object to step through time. There are a few parameters used for this function, TimeStep, VelocityIterations and PositionIterations. TimeStep is the amount of time that should pass with each call, using 1/60 as the TimeStep means 60 frames per second. VelocityIterations improve the stability of the simulation, while PositionIterations increases the collision detection accuracy[25]. The higher these values the more accurate the simulation, but there will be an increased execution time. The simulation in this application uses 6 for the VelocityIterations and 2 for the PositionIterations, the code excerpt in Code 4.10 is all that is needed to move a simulation by one TimeStep.

Code 4.10: Executing a Box2D simulation

```
world->Step(TimeStep, VelocityIterations, PositionIterations);
```

4.3 Application Architecture

The application built in this research was designed and produced as three separate components, the User Interface, Simulation Environment and GNP Library. Both the Simulation Environment and GNP Library can be used totally independent of one another, but the User Interface is primarily there to bind these two separate components and make it easy to understand how the controllers are running.

4.3.1 User Interface

The Interface had specific requirements that were defined prior to development, it needed to visually display the simulation environment and the GNP that was currently being executed within the simulation. It also needed the ability to load and save GNP controllers, view and modify parameters within each node in the controller and lastly the ability to initiate the GNP training process. The resulting interface from these requirements, is shown in Figure 4.1.

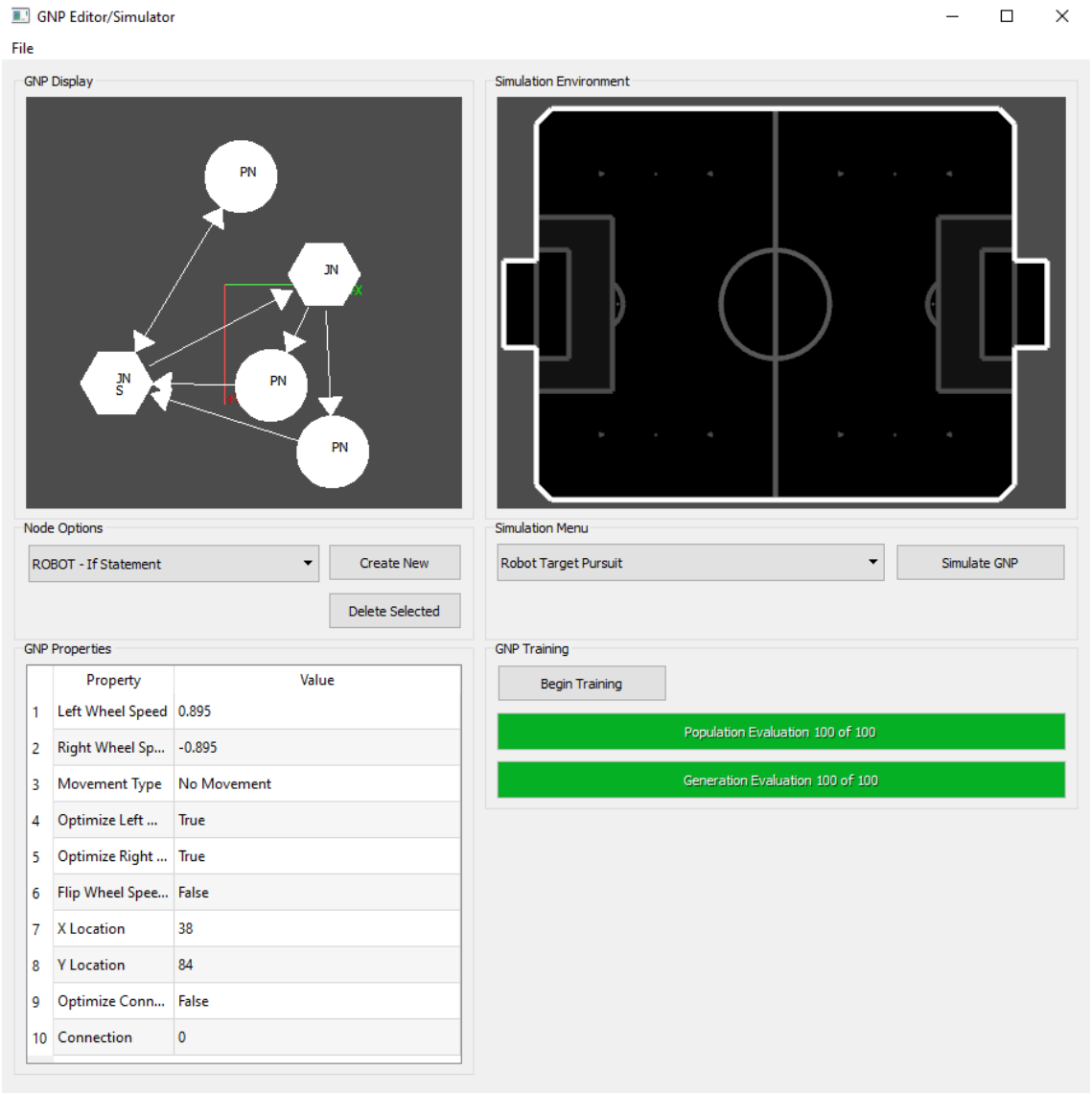


Figure 4.1: Shows the full application User Interface

Looking at Figure 4.1, at the top left there is a graphical display of a GNP controller, every node within the display can be moved or selected. Selecting a node within the GNP display will allow the node to be deleted or its internal data viewed within the GNP Properties window at the bottom left, every modifiable parameter for each node can be changed within the properties section, this also remains true for both Fuzzy Logic nodes and GNP nodes which have the ability to import and export other pre-built controllers. Each GNP node when selected will allow the user to move down into the visible cascades or move back up a level, selecting these options will change the visible GNP Controller within the display. Between the GNP display and properties, are the controls for creating and deleting nodes within the currently visible GNP.

On the right side of the user interface are the controls for selecting a simula-

tion to be executed, a button to begin training the GNP controller for the selected simulation along with progress bars for that process and lastly the simulation environment is displayed at the top, which can execute the GNP controller on the selected simulation by pressing the 'Simulate GNP' button.

Within the 'File' menu at the very top is the controls for loading, saving and creating a new GNP controller. Examples of network save files are shown in Appendices A.1,B.1,C.1 and D.1.

4.3.2 Simulation Environment

When designing the simulation environment the main consideration was re-usability, it should allow other developers to easily create their own simulation for use within the application, regardless of whether it is Robot Soccer or something else. A small diagram displaying the major components of the simulation is shown in Figure 4.2.

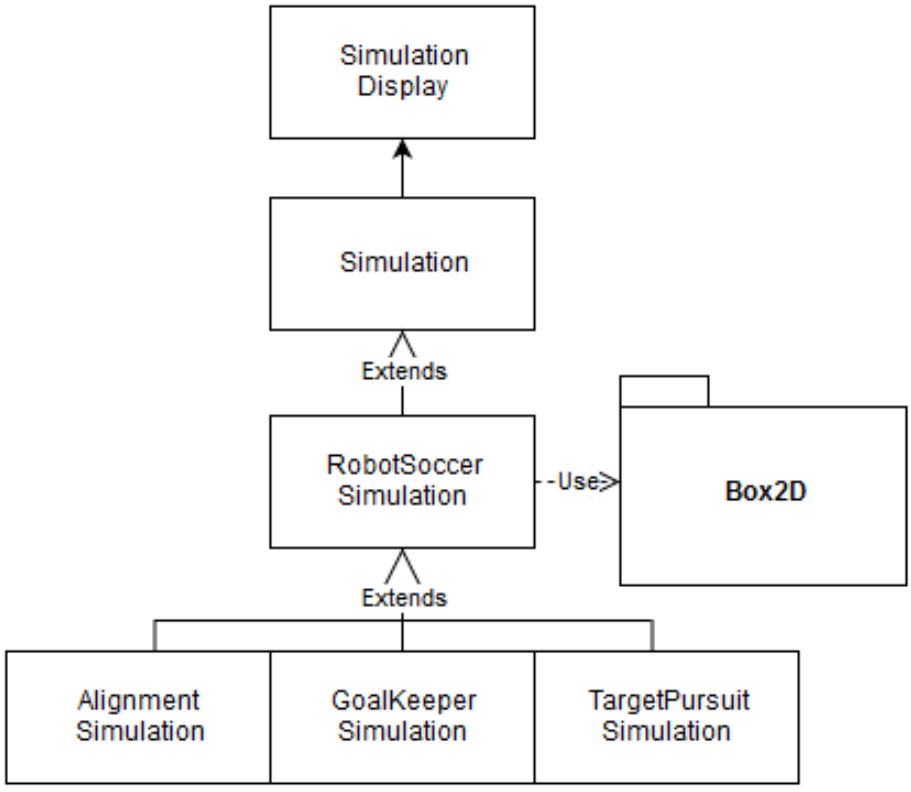


Figure 4.2: A diagram of the Simulation Environment design

The base class for the simulation environment is 'Simulation', this class contains basic functions for starting, stopping and drawing the current simulation, it also provides interfaces keyboard and mouse control which can be implemented if required. Everything that extends the 'Simulation' class must implement a 'Run' function, this is the main function within the RobotSoccer Simulation and should

contain all processing logic, once this function finishes the simulation is considered complete.

RobotSoccer Simulation both creates the Box2D world and builds the FIRA MiroSot Middle League[12] field into it, it controls the creation of Robots and Balls by providing an 'addEntity' function and provides various interfaces for any class that extends it. Some of the interfaces this class provides are various execution stages or collision events, these execution stages include PreRun, which is used for initializing data before the simulation begins and PerRun which is executed every display frame of the simulation (currently 60 per second) to name a few.

Each of the remaining classes that extend RobotSoccer Simulation provide any additional functions or data specific to that purpose, this includes the actual robot placements within the field, how the GNP controllers affect these robots and defines the overall objective function for each task.

4.3.3 GNP Library

As is with the Simulation Environment the main consideration was re-usability, but also the ability to cascade GNP controllers. It should be designed in a way that allows any type of Processing or Judgement node to be included into a GNP controller and thus this library should only provide the bare minimum to allow further specialization without compromising the algorithm behaviour. The diagram in Figure 4.3 shows the components in this GNP Library.

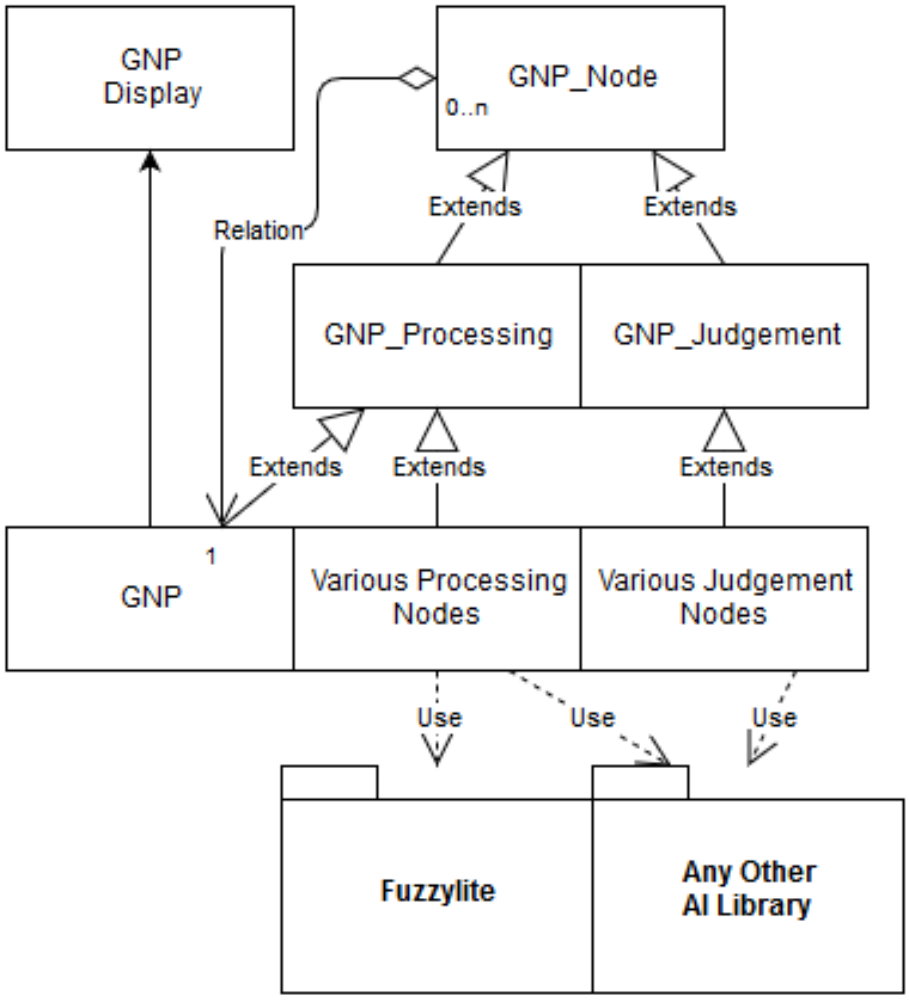


Figure 4.3: A diagram of the GNP Library design

Everything within this GNP design stems from the `GNP_Node` class, which controls movement of each node within the display and provides a large number of interfaces which should be implemented in either `GNP_Processing` or `GNP_Judgement`, most notably the loading and saving functions and an 'execute' function that can be called to run any code related to the specific node.

Both `GNP_Processing` and `GNP_Judgement` are very similar, the only significant difference between the two is that `GNP_Processing` can only have one connection and `GNP_Judgement` can have as many as it requires. Both classes implement the 'execute' function and have some security within them to prevent the GNP controller going out of bounds in the case where something goes wrong, this function then calls either 'Judge' or 'Process' respectively. These functions are defined in any subclass as it is where the bulk of the computation is done, they should be custom designed to the situation and for any additional libraries that they may use, an example would be the fuzzy logic nodes which interface directly with Fuzzylite.

A GNP in itself is a type of `GNP_Processing` this allows it to be treated as

a `GNP_Node`, therefore a GNP that contains many `GNP_Node`s will not only create the desired ability to have any type of node within a GNP but also facilitate the cascading GNP feature. There is not much difference between a GNP and other types of processing nodes except that it has functions for inserting and deleting nodes and the content within its 'Process' function would be unique, which is further discussed in Section 6.2.5.

What isn't shown in the above diagram is a class called '`GNP_Genetic`' which directly interfaces with GALib and controls the entire optimization process (discussed in Section 6.2.3).

Chapter 5

Robot Soccer Simulation Platform

Massey University has an implementation of the FIRA MiroSot Middle League soccer field located at the Albany campus, it is located in a basement level with no windows and lights are uniformly distributed across the room to minimize variable lighting conditions. A camera is located directly above this soccer field which is used to identify the location and orientation of the robots and ball on the field, robots are uniquely coloured for unique identification. The images from this overhead camera is sent directly to a computer for the bulk processing, once images have undergone colour correction and the locations of all objects have been identified, it is up to the controllers to determine the next course of action. These controllers submit a set of wheel speeds to each robot roughly three times a second via an XBee Radio Frequency transmitter. The entire platform is shown in Figure 5.1.

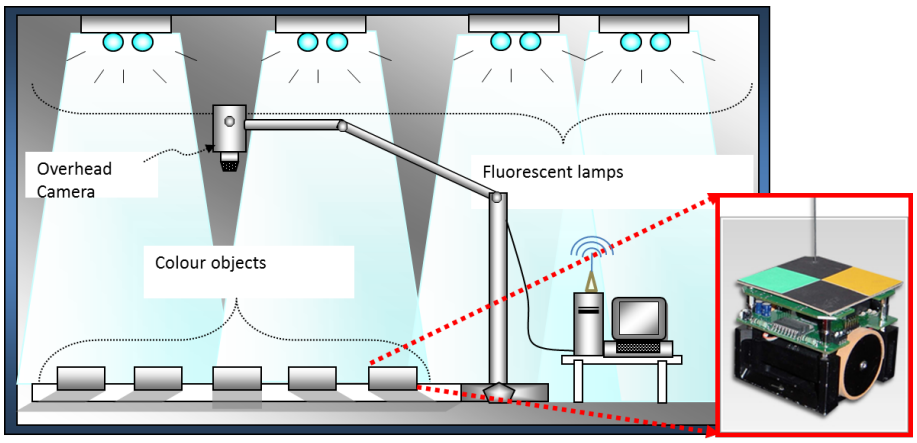


Figure 5.1: Depiction of the Massey University Robot Soccer Platform

The real system would likely struggle under a competition scenario as any number of problems can occur. For example, the time it takes between object location data updates is totally unaccounted for. While it is a small amount of time, this reduces the accuracy of the overall system. Other problems that may be encountered are that

the camera could temporarily lose connection with the computer or an unexpected lag in the colour classification process, both of these events would cause the robots to falter or even crash.

This simulation was built with the intention that it could be integrated with the real robot soccer system and open the potential for further research, where robot and ball data are updated in the simulation environment as the data becomes available. This would not only estimate the robot location data between the data updates, but it could also help protect from any unforeseen issues. Integration into the real platform would also allow opposition move prediction systems to be developed and therefore allow more accurate strategies to be devised, it would have the additional benefit of creating smoother robot movement as they could receive commands at the same rate that the simulation is evaluated.

With all these considerations in mind when developing the robot soccer simulation, the accuracy of the simulation was paramount. Therefore this simulation is based on the FIRA MiroSot Middle League and uses exact measurements of the robots used in the Massey University system.

5.1 Playing Field and Robot Design

The soccer field design must replicate the FIRA field specifications in Figure 5.2 and must be done using the Box2D physics library, otherwise wall collisions would not be evaluated. The best way to form the required shape is by stitching together primitive shapes, as Box2D doesn't accept concavity in its shapes or holes in the middle of each shape.

Therefore this field built using eight rectangles and four triangle, all defined as static objects so they will not move at all when a robot or ball collides into them. There are four rectangles that comprise the outside border, they are placed against the northern and southern walls and then span the width of the field at the back end of each goal. The last four rectangles form the sides of each goal and creates the back wall above and below the goals. The triangles are the placeholders in each corner of the field, which are there to prevent the ball getting stuck in the corners[12]. View Figure 5.3 for a depiction of these wall placements. For code examples on how to create static and dynamic objects refer to section 4.2.

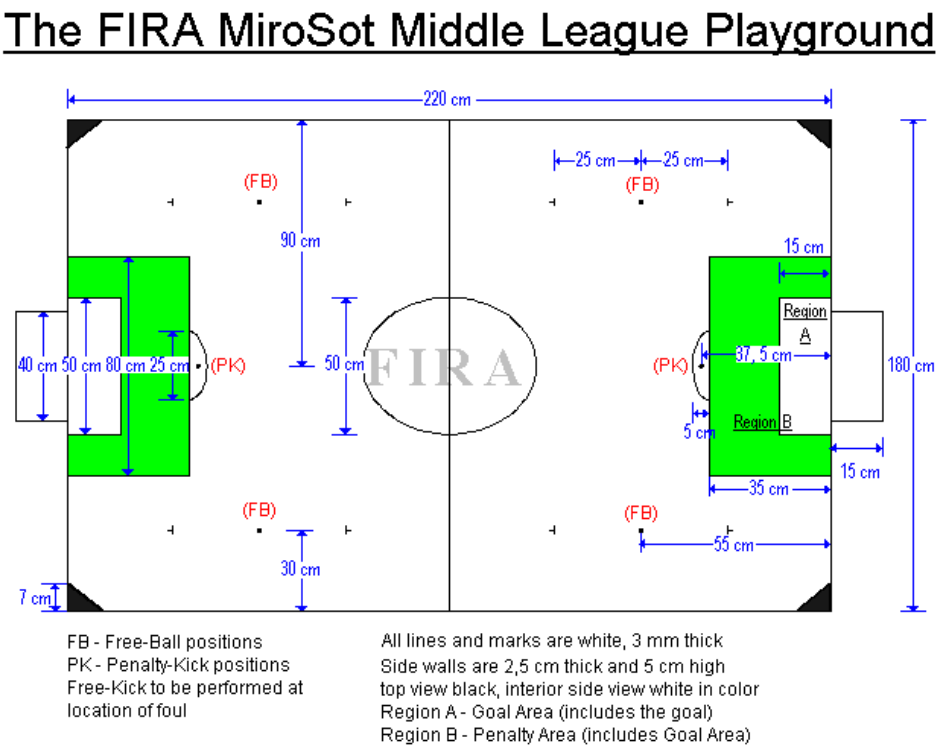


Figure 5.2: The FIRA MiroSot Middle League field specifications [12]

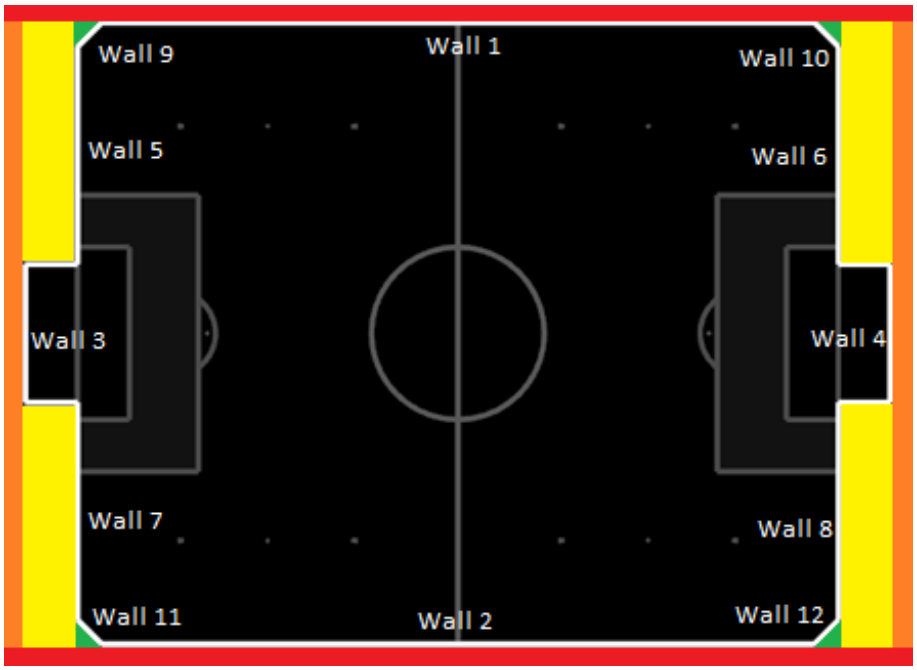


Figure 5.3: The soccer field wall that is comprised of twelve basic shapes in Box2D

As for the robot that is used in the simulation it is in itself a complex shape made up of primitive shapes, it is 7.5cm by 7.5cm and has scoops on both the front and back, which are ideal for catching and kicking the ball. If these scoops were not simulated then any controllers created using the simulation would not be applicable to the real implementation.

There is the main body of the robot which is 55mm by 75mm, two rectangles on each side of the scoop and then three triangles to make up the rest of the scoop.

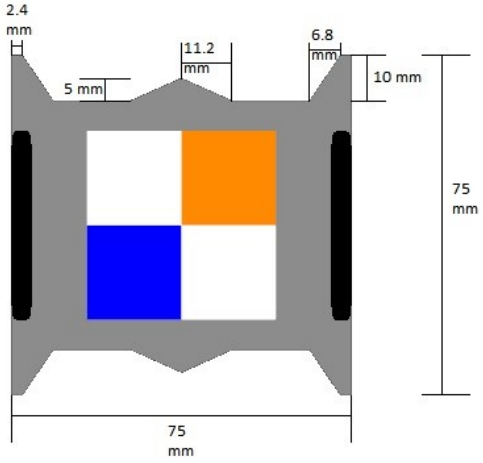


Figure 5.4: Robot Design and Dimensions

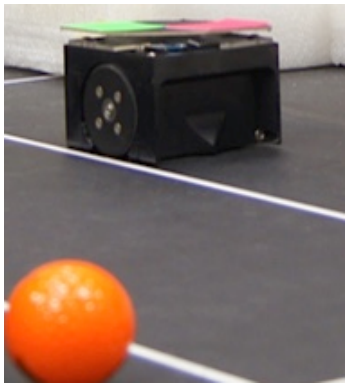


Figure 5.5: Real Robot and Ball

5.2 Physics Considerations

Unfortunately the default settings for Box2D involve at least one of the four directions having some form of gravity, therefore every object that isn't touching the ground is considered to be free-falling. The Robot Simulation requires a birds-eye view, totally negating all forms of gravity because all objects are considered to already be on the ground.

Disabling gravity is a simplistic task within Box2D as it only requires changing a vector value before the world is created. The issue encountered is that all objects are practically floating as far as the physics engine is concerned, therefore no form of friction (unless moving along a wall) is computed by the engine. Robots colliding would either send them perpetually spinning, sliding in some direction or a combination of the two and the ball would only ever lose momentum when colliding with a wall or robot.

In order to correct this fault the friction calculations had to be implemented for these three situations, forward velocity, lateral velocity and angular velocity. To limit the complexity lateral velocity was completely removed from the robots, as they have rubber wheels and would be unlikely to slide unless another robot collides

into the side. Forward and angular velocities are dampened, but not removed to allow the ball and robot to slow down and eventually stop in a realistic manner. Forward and lateral directions are shown in Figure 5.6.

A Box2D tutorial available on the internet by iforce2d[26] looks at a similar situation, involving a car moving around a track in a birds-eye view. The approach used in this tutorial was implemented into the Soccer Simulation and can be further optimized to replicate the friction within the real robot soccer implementation.

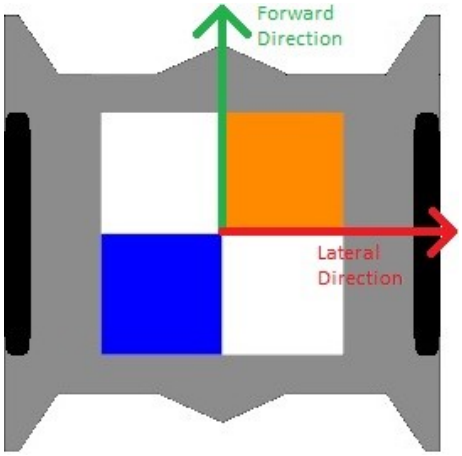


Figure 5.6: The Forward and Lateral directions of the robot

There are three sections of code that are executed in every time step within the Soccer Simulation. The first removes lateral velocity (to the side) for robots by calculating the force required to negate this velocity and applying an impulse in the opposite direction, this is shown in Code 5.1.

Code 5.1: Removing lateral velocity

```
b2Vec2 currentRightNormal = body->
        GetWorldVector(b2Vec2(1,0));
b2Vec2 lateralVelocity = b2Dot(currentRightNormal,
        body->GetLinearVelocity()) * currentRightNormal;
b2Vec2 impulse = body->GetMass() * -lateralVelocity;
body->ApplyLinearImpulse(impulse,
        body->GetWorldCenter(),true);
```

The second piece of code reduces the angular velocity for both the robot and the ball, but not all of this angular velocity should be removed, since both Robot and the ball can still rotate or be forced into a spin via collisions. Code 5.2 shows the required steps and can be tweaked by modifying the 0.1f to a value that closely represents the real environment.

Code 5.2: Reducing angular velocity

```
body->ApplyAngularImpulse(0.1f * body->GetInertia()
        * -body->GetAngularVelocity(),true);
```

Similar to the code which removes lateral velocity, the last code excerpt shown in Code 5.3, introduces a form of drag which continues to reduce the speed of the ball and robot until they stop. Once again, the strength of the drag can be modified by changing the 2.0f value.

Code 5.3: Introducing a drag force

```
b2Vec2 forwardNormal = body->GetWorldVector( b2Vec2(0,1) );
b2Vec2 forwardVelocity = b2Dot( forwardNormal,
        body->GetLinearVelocity() ) * forwardNormal;
float forwardSpeed = forwardVelocity.Normalize();
float dragForceMagnitude = -2.0f * forwardSpeed;
body->ApplyForce(dragForceMagnitude * forwardVelocity,
        body->GetWorldCenter(),true);
```

Chapter 6

Genetic Network Programming

Within this chapter the original architecture is discussed which covers various applications, approaches and the types of nodes used. It also discusses the modified architecture which is the approach made by this research and all changes to the original architecture. All experiments in subsequent chapters are based on the modified architecture.

6.1 Original Architecture

The original Genetic Network Programming architecture was first introduced and described by Katagiri, Hirasawa and Hu[5] in their research paper "Genetic network programming - application to intelligent agents". This initial structure has remained unchanged since its initial proposal, the main differences between research papers are the type of nodes used and the problem it has been used to resolve.

GNP is a collection of nodes that interlink in a directed graph, there is no termination point in the network as it is designed to recursively loop based on the state of the environment it is being used for and the previous states of the network. Each GNP system contains one 'start' node, zero or more 'judgement' nodes and zero or more 'processing' nodes. Figure 6.1 shows a GNP system with each of the node types.

The start node simply states the initial position in the GNP by connecting to the node where execution should begin the first time it is used.

Judgement nodes have two or more connections, these judgement nodes are branching points like an if statement it will move to one node if a condition is true and move to another if the condition is false. These judgement nodes can be coded in many different ways, one would be to simply use a conditional statement internally, they could use a series of these conditional statements or use artificial intelligence libraries to decide which node should be executed next.

Processing nodes are the last type of node in any GNP system, they are a form of action that influences the environment it is designed for. They could be adjusting a motor on a robot, modifying some data or anything else related to the problem at hand. These nodes only have one connection, which states the next node after the execution has been completed.

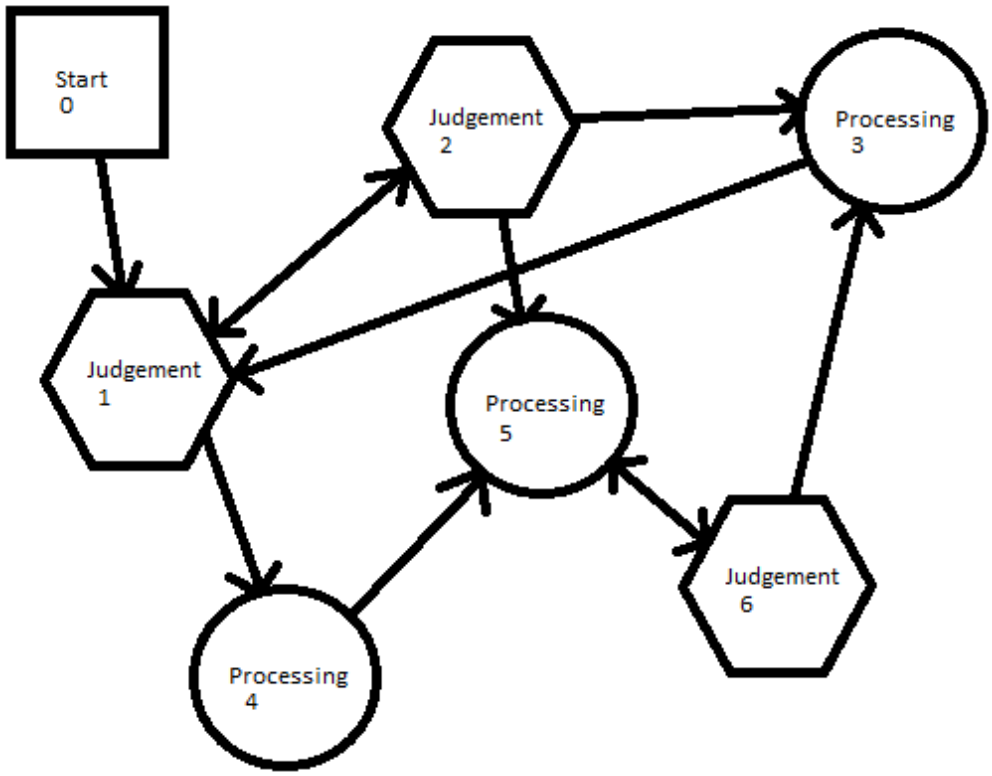


Figure 6.1: Example GNP System

These judgement and processing nodes can form a series of implicit actions that are often seen in general programming environments. A chain of judgement nodes can be nested to form an AND configuration, a node will only be executed if two or more conditions are true, an example of this situation is shown in Figure 6.2 for node 2. Similarly a chain of Judgement nodes can form an or configuration, this is also shown in Figure 6.2 as node 1. One judgement node and a set of processing nodes can be formed into a loop which directly replicates the behaviour of the while loop in most programming languages, the behaviours will continue until a certain condition is met, these could also be nested within one another. Lastly if a single judgement node has a connection that links back to itself, then this behaviour acts like a wait or sleep until a certain condition is reached. Some of these behaviours may or may not be useful for specific problems, but if they are not required then they are likely to be optimised out of the population.

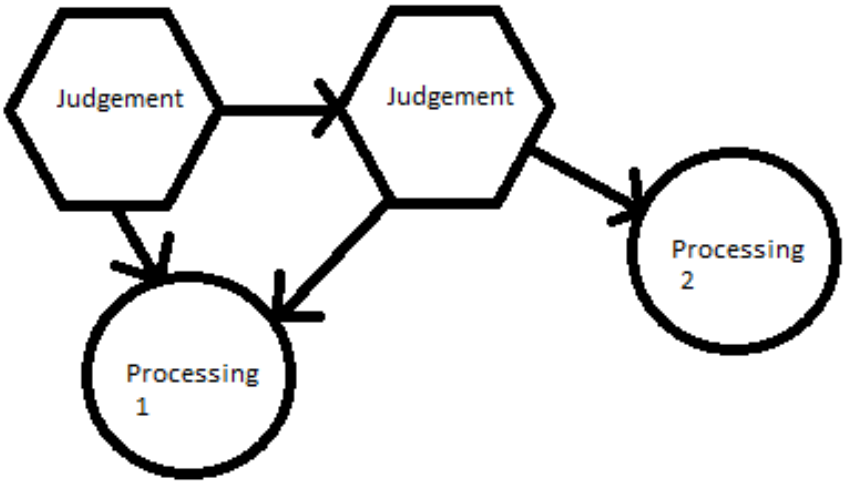


Figure 6.2: Possible GNP Node Configurations: Node 1 is in an OR and Node 2 is an AND

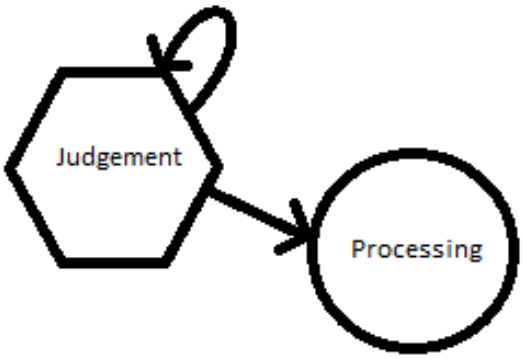


Figure 6.3: Possible GNP Node Configurations: Wait until condition

As stated previously the network begins at the start node when it is executed for the first time, the network is executed whenever the robot or platform requires an action to be made, this could be run repeatedly at certain time intervals or whenever data becomes available. At the end of each execution the network retains the knowledge of the state(node) it was in and the next execution it will continue from that previous state, this prevents the system from repeatedly executing the same nodes. In the original architecture a maximum of five judgement nodes or up to four judgement nodes and one processing node are executed each time the network is used, as soon as a processing node is executed the network ends its execution at that point. The network execution ends as soon as a processing node is evaluated to prevent another processing node from overwriting any changes to the system that

the previous may have done.

A time delay concept is also included which is to represent any delays between making an action and being able to make another action, or a judgement and an action. However this time delay is largely unused despite stating that it seems to make improvements when this node transition delay is used[5, 27].

6.1.1 Node Types

Most applications that use GNP use very specific nodes to the problem, for agent based systems there could be nodes to send the agent forward, backwards or conditions to test whether they can move in a certain direction[5]. If a node is too specific to the problem it is designed for, then the code and networks will not be reusable for other applications. However if generic nodes like 'move forward' are used to train a complicated behaviour, then these individual nodes can be replaced with nodes that provide the same functionality for another robot, then this new robot will posses the same complicated behaviour and requires very little to no additional training.

Only recently have additional artificial intelligent systems been implemented as nodes within a GNP network, this is generically Reinforcement Learning or Fuzzy Logic[10, 27]. While these two algorithms are useful and for the tasks they are required to do, it would be interesting to see a broader selection of node types included in the networks, specifically path finding nodes or neural networks.

Wang[10] proposed including pre-calibrated AI systems into a GNP network, up until this point all nodes were trained along with the GNP which is not practical when it comes to large control issues. The ability to include previously developed AI systems is the first step towards a modularized GNP.

6.1.2 GNP Individual Definition

In Figure 6.4 the gene structure for a single node is displayed, K is the node type being either Judgement or Processing nodes, ID is an identification number for a node within their node library and T is predefined to one for Judgement nodes and five for processing nodes.

Connections are listed as C, this is an identifier to the next nodes that should be processed. CT stands for Connection Time, this is a delay between one node and the next node. The parameter is never actually used within any research and seemingly redundant for the architecture, if a waiting period is actually required for a network then the design in Figure 6.3 could be used or even modified to include a specific number of loops.

The figure displaying the gene structure (Figure 6.4) is slightly deceiving as in the paper that originally proposed GNP[5] it states that only the connections are

optimised. Therefore in actual fact the gene structure would only contain a series of connections (C), the remaining parameters could in theory be optimised except for K but they never are.

If K were to be optimised then there will be significant redundancies when listing the connections and connection times, as it will have to account for every potential situation, specifically the maximum number of connections used in any given node within the node library.

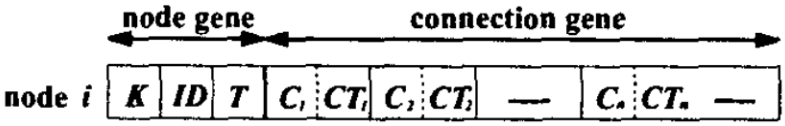


Figure 6.4: Original Architecture Gene Structure for Node i[5]

More recent advances in the GNP architecture that includes Reinforcement Learning nodes allow optimisation of their Q Values and the node types internally[10, 27], while this is a significant improvement over the original GNP it is still limited by the contents of the node library used in the optimisation process. If a required node or variant of a node is not present within the node library, then the overall architecture will struggle to complete the task, this does include the reinforcement learning aspect.

6.1.3 Training Strategy

The typical approach at optimisation in Genetic Network Programming is to solely use Genetic Algorithms, while GA's are effective at finding solutions when the solution space is large, it does not guarantee an optimal result. In previous research there has been very few attempts to further enhance the optimisation process, with the exception of Wang[10] who incorporated Hill-Climbing every five generations for the top three individuals and Yu et al.[28, 29] who modified the Genetic Algorithm to include Ant Colony Optimisation.

Hill-Climbing is an iterative search in the local area of a solution space, where it modifies individual parameters in order to obtain a locally optimal fitness. Due to the iterative nature of the algorithm, it is often time consuming especially if the candidate solution has neighbours with a relatively small increase in solution fitness (a plateau on the solution space). Conducting this search repeatedly throughout the genetic optimisation process, as was proposed by Wang[10] would significantly increase training times.

The Hybrid GNP with Ant Colony Optimisation approach introduced by Yu et al.[28, 29], where every ten generations in the typical genetic process there is a "spe-

cial generation". In each special generation, crossover and mutation are computed with a weighted probability based on pheromones. The pheromones are influenced by the overall fitness of the current generation and the number of transitions that occur from one node to another. Therefore the more often a particular node connection is used within the population, the more likely it is that the successive generation would use the same connection and visit its parent node. Yu et al. show that this hybrid optimisation is more efficient than simply using Genetic Algorithms; however more tests should be conducted, as the reduction in population diversity may be detrimental to the operation of the standard GA component.

The actual encoding scheme used in previous research is not often discussed in detail, but those that do mention it are using binary encoding[7] scheme. However it has been shown by Janikow and Michalewicz[20] that floating point representation is more time efficient as they do not require decoding, this is especially true with a large number of optimized parameters.

Only two different types of parameter are optimised within this GNP architecture, that is the connections which define all the transitions from one node to the next and in some cases the node identification number is optimised[5, 27]. In terms of the identification number a library of potential nodes is used, this library is indexed by this identification number thus allowing the actual nodes to be optimised based on this library. This however means that only nodes within this library are considered and nothing outside of it. While this will keep the solution space small, it will reduce the likelihood of reaching the best potential solution, if a required node or variation of a node is not included in the node library.

When a node identification is being optimised by the GNP, the selection process will be limited to a subset of the actual node library; effectively those nodes with the same node type and the same number of connections. Otherwise, swapping nodes with different node types (or number of connections) will result in an incompatible number of node connections. This problem can be avoided by optimising an excessive number of connection data for each node, the actual amount of connection data would reflect the worst case node (node with the most connections). For any node that uses less than this number of connections, there is redundant optimisation parameters and thus increases the solution space exponentially with the number of nodes in the network.

In this architecture uniform crossover is used, this is similar to one point crossover discussed in Section 2.2.3 except that every point in the binary string is considered as a crossover point based on the set probability of crossover. Mutation occurs in a very similar way, using the defined probability of crossover, each bit has a chance of being flipped.

6.1.4 Pseudo-code

The actual code that executes a GNP system is not very large, as most functionality is included in the nodes themselves. Before the system is run for the first time the "Initialization" section of Code 6.1 is run, then subsequently every execution stage is run every time it is required, this could be in set time intervals or when new data becomes available.

The lines that state "CurrentNode = CurrentNode.NextNode" may well be more complicated depending on the developer, this line is used to represent both Judgement and Processing nodes returning the next node to be executed. This may in fact require searching through an array or table of nodes.

Code 6.1: Original GNP Architecture Pseudo-code

```
Initialization:
CurrentNode = StartNode
------
Each Execution:
JudgementCount = 0
While CurrentNode is JudgementNode and
                  JudgementCount not equal 4
        Execute CurrentNode
        CurrentNode = CurrentNode.NextNode
        JudgementCount = JudgementCount + 1
Execute CurrentNode
CurrentNode = CurrentNode.NextNode
```

6.1.5 Strengths and Limitations

The original architecture works well with most applications, it is proven to work in many diverse problems such as agent control[5, 6], elevator systems[7] and robot control[9, 10]. However the algorithm will struggle with larger, more complicated systems for multiple reasons:

- Using the binary encoding scheme[5] will increase optimisation times in large problems
- Genetic Algorithms alone do not guarantee finding an optimal solution, or even a local maximum
- Networks are restricted to relatively small tasks, otherwise the solution space becomes too large

- It is difficult to remove sections of a network and reuse them in a different application
- Most node types are problem specific and only a few types of AI nodes have been implemented. The full-potential of GNP architectures using various AI nodes is not yet fully explored.
- Optimisation is limited to the contents of the node library that is used, an acceptable solution will not be found if a required node or variant of a node is not present in the library

6.2 Major Architecture Modifications

For the most part the Modified GNP or also known as Cascading GNP is very similar to the original, it will work for all applications that the original has been applied to and will only require very small changes to work with any network that has previously been optimised with the original GNP architecture.

Changes have been made to how execution time operates, increased re-usability from general purpose nodes, it has become modular by allowing problem subdivision into multiple layered networks, the training process adds hill climbing and the internal data of all nodes are now optimised.

Execution Time

First, the execution time has changed; in the original architecture when the network is run, it will execute a maximum of five judgement nodes or less than five judgement nodes and a processing node. Instead of having this strict limit on judgement nodes a time system is implemented, where every time the network is run there is a maximum time allowance of 1. Each node type will have its own time requirements, so a simple IF statement judgement node could have a requirement of 0.1 in which case a maximum of ten of these IF statements will be executed per network run. If there is not enough time remaining for a node to be processed, in the situation where 0.1 time is remaining and the current node requires 0.2 then the network will end at this point. When the network is run again, it will have 1 time remaining and that current node will be executed. This is similar to the original architecture but this change was required to implement cascading, which is discussed further in Section 6.2.1.

Object Orientated Design

The node structure is developed in a pure object orientated design, which implements most of the node functionality and allows new nodes to be easily developed and integrated into the network. While this is not a significant change, it does increase the re-usability of the algorithm by allowing other developers to add new node types to the GNP. Unfortunately this does come with one limitation where nodes can not transmit data between each other, this is not a specific requirement of the GNP architecture and can be fixed sometime in the future, it would be a useful feature to have.

Repeated Execution

To facilitate the modularization of GNP using cascading a new feature is implemented for nodes that require an excessive amount of time, it allows processing nodes retain its current state for a set number of GNP executions, effectively allowing them to loop temporarily before continuing towards the next node. After each of these loops the GNP network will finish as the current node is a processing node, therefore it loops with each call of the network. Currently this is implemented by adding an integer parameter that counts down to zero, upon reaching zero the current node will be updated with the next node in the network. This additional parameter could be removed later and the node execution time mentioned above could be used to implement this feature. This feature is initially implemented to allow Cascaded GNP networks, but would be useful for nodes that conduct complicated manoeuvres.

Cascaded GNP networks are further discussed in Section 6.2.1 and all changes to the training process are discussed in Section 6.2.3.

6.2.1 Node Types

Three changes regarding the node types have been made, the first is that the system is built in a way that allows any node type (artificial intelligence or problem specific nodes) to be developed and integrated with ease, the only limitation is that they must act independently and can not transmit data between nodes but this could be implemented in the future. This change also allows Genetic optimisation to be given to any node that requires it, a fuzzy logic node that has a connection to itself and is optimised via GNP is effectively Genetic-Fuzzy, this remains true for any future integrated nodes.

Secondly the use of a specific “Start” node has been removed as its only purpose

was to identify the first node to be executed, instead the start is identified as a property of the network.

The third however is the ability to have a Cascading GNP network. These Cascading GNP's are GNP networks that contain other GNP networks, theoretically there is no limit to the number of layers that can be used, however this has only been tested on two layered networks. A developer could have a top level network that acts as a dynamic role allocation system, where a robot will change roles based on the environment, then the next layer could control specific actions regarding roles and then a further layer could describe how the robot should move. These Cascading GNP (CGNP) will be especially useful for complicated robotics with many motors and behaviours, for example a robotic hand or even a complete humanoid robot. A very small CGNP network is shown in Figure 6.5, the top level network has three nodes one of which is another GNP network, the second layer only has two nodes that point to one another.

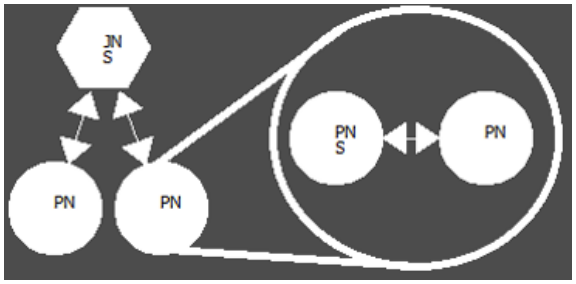


Figure 6.5: Cascaded Genetic Network Programming Example

There were two main problems preventing the use of CGNP in the original architecture, both of which are discussed in Chapter 6.2. These are the time facility and the need for repeated executions.

In the original architecture a maximum of five judgement nodes or less than five judgements and one processing node are executed, if that same theory was applied to CGNP then with a large number of layers the system would be allowed to infinitely traverse through new layers. This would result in a system that hangs, lags or otherwise freezes while waiting for a response from the GNP. This is because each layer of GNP would itself be allowed to process these five judgements and or less than five and one processing node, a CGNP itself is a processing node as they will influence the system in some way, so the first layer could execute four judgements and then a CGNP, the second layer could execute four judgements and a CGNP etcetera. Therefore by allowing a single layer of GNP to have a maximum of 1 time allowance, a second GNP layer could cost a total of 1 time and it will then only be executed when the network is freshly run without any previous node execution. This

will cause a delay of one network run before the next layer is executed however this would be negligible for most applications, for time critical applications the remaining time could potentially be passed as a parameter to the next GNP layer.

Repeated executions are required for CGNP as when a multi layer network is executed without the repeated execution facility, then the second GNP layer will be executed and may reach another GNP node however this third layer will never be executed because the second layer has finished its single evaluation and the top level network has already moved to the next node. Instead the required number of repeated executions can be trained using the genetic algorithm, so the third layer may only require one execution, the second layer could require three executions and then the top layer will not leave the CGNP node prematurely. Whenever a CGNP node becomes the current node in the layer above the new layer will reset its current node to the start. Therefore the subsequent layer will act like it has never been executed before, with every repeated execution the state will remain the same as it was in the previous execution. This allows all nodes to be visited with repeated executions but will also behave in the same or similar way each time the node is visited.

6.2.2 GNP Individual Definition

The approach taken by this research does not allow node types to be optimised like the original (discussed in 6.1.2), instead this thesis proposes the use of general purpose nodes which both increases re-usability of nodes and provides more optimisation.

Prior to optimisation the developer can add any number of nodes to a GNP, these could be Fuzzy Logic nodes, problem specific nodes, simple IF statement Judgement nodes, Reinforcement Learning, literally any type of node. The node types do not change within the optimisation process but data within the nodes can change, each node can request any data optimisation that it may require and depending on the developers preferences. A Fuzzy Logic node could optimise the membership functions, inputs, outputs, rules or any combination of these. A Reinforcement Learning node could use the GA to set initial states for the Q values or a simple IF statement Judgement node could choose its own input parameters and formulate its own conditional statement. Additional to the internal node data, the node connections can be optimised as well as the Repeated Execution parameter described in Section 6.2.

The gene structure for Node i is depicted in Figure 6.6, R is an integer value that states the number of Repeated Executions, there is a series of numbers that define the next potential nodes (C) and lastly is ND which is the Node Data, all node specific data is concatenated onto the end of each individual nodes gene structure.

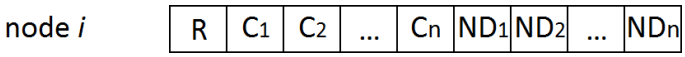


Figure 6.6: Modified Architecture Gene Structure for Node i

As an example a Fuzzy Logic Node's gene structure is shown in Figure 6.7, in this case it is only optimising the membership functions as denoted by MF and its Input type which is marked by an I, this input type is selected from a list of potential environmental variables. This could expand to include all other Fuzzy components but for the sake of simplicity only these two components are shown.

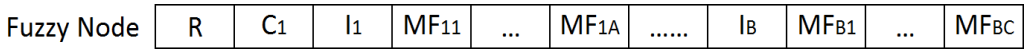


Figure 6.7: Example Gene Structure for a Fuzzy Node

This optimisation process can span multiple CGNP layers, this would be done in cases where a sub-layer has a similar objective function to the top layer but contains different nodes. Of course the sub-layer optimisation can be disabled for situations where pre-trained systems are being imported as CGNP nodes. Figure 6.8 displays how these node genes (N) are stitched together to form the CGNP Chromosome, at the very start is SNID which the Start Node Identification number. If multiple layers of CGNP is optimised at the same time, then this same structure will also be used as one of the internal node gene structures.

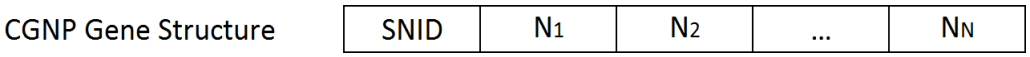


Figure 6.8: CGNP Chromosome with Nodes

Allowing a node to optimise its internal data means that better solutions will be found in the situations where the original architecture does not contain fine tuned nodes in its library of potential nodes. This increases the solution space as each node will potentially add more data parameters, this will increase the required population and generation thus increasing the overall training times. This is not necessarily a limitation of the data optimisation, as it is scalable from mere connection optimisation to multiple level CGNP's with Reinforcement Learning, but should be considered when designing a network for optimisation.

6.2.3 Training Strategy

The method used to train the network and the encoding scheme has also been improved, by the use of Hill Climbing at the end of the Genetic Algorithm process

and using floating point encoding.

Hill Climbing is implemented at the very end of the GA training process which is unlike any other GNP research, this can be beneficial in situations with any number of local maxima as the GA will find an acceptable solution and Hill Climbing will further improve this result to the local maxima of the best solutions. Instead of applying hill climbing every few generations as is done by Wang[10], it is applied at the very end of the GA optimisation process only to the top 5% of the population. This increases the overall optimisation time but guarantee at least a local maxima solution, if the GA is lucky enough to have one of the top solutions somewhere near the global maximum then using hill climbing will find it by iteratively searching the local area of the solution space. As a further research this could be changed further to include Ant Colony Optimisation (ACO) as well, as it is proven to work more efficiently than GA alone[28], ACO works in tandem with Genetic Algorithms therefore using Hill Climbing at the end of the optimisation process will remain beneficial to the overall optimisation process. A simplified implementation of the Hill Climbing algorithm used in CGNP is shown in Code 6.2. Within the pseudo-code, neighbouring values N in G, refers to finding potential values near to the current value for a specific gene, that will potentially increase the fitness of the individual GNP. The search may proceed by decreasing or increasing the gene G in each iteration of the hill-climbing process.

Code 6.2: Simplified Hill Climbing Pseudo-code used for CGNP

```
Best = Best individual in the population
For each solution Current in the top 5% of the population
        For every gene G in Current
                For every neighbouring value N in G
                        Potential = Current with gene N
                        Evaluate Potential
                        if Potential.fitness > Best.fitness
                                Best = Potential
```

Since the CGNP is designed to be expandable to any number of nodes, it is not feasible for binary encoding to be used otherwise the time required to decode candidate solutions would become exponential[20]. Therefore to prevent the need for repeatedly decoding the data, the data itself is kept in its natural form using Floating point representation (discussed in Section 2.2.1). The crossover method used for this encoding type is Uniform Crossover, while it is similar to the binary version instead of separating the parents on the binary level, they are instead split at an array level therefore individual numbers are not split in any way. Mutation is conducted by randomly selecting a gene in the array and re-randomizing it, this

includes additional variation to the population but sticks to the constraints set by the individual gene, therefore it will not go out of a predefined set of potential values.

6.2.4 Cascaded GNP Example

This section steps through a multiple layer Cascading Genetic Network Programming example for a total of six executions, each execution of the network is displayed in an image and a accompanying table. Within each table 'RE' is the Repeated Executions parameter at each stage, the images are labelled with the current step in Red, these steps directly correspond to the table.

Within these figures many features of the CGNP architecture are depicted, specific components that should be noted are:

- In the first execution, the new processing node is not executed because it lacks enough remaining time allowance to execute the node.
- Within the multiple layered networks, the Repeated Execution parameter is reduced for every layer.
- Since CGNP does not contain termination nodes, the parent network will decide when the child network should end. This is shown in the fourth execution, step 8 where the child network ends because the Repeated Executions reach zero.

Not shown in this example is what will happen when a previously visited CGNP processing node is revisited, each new visit to the node will reset the current node to its start position and it will then continue as normal.

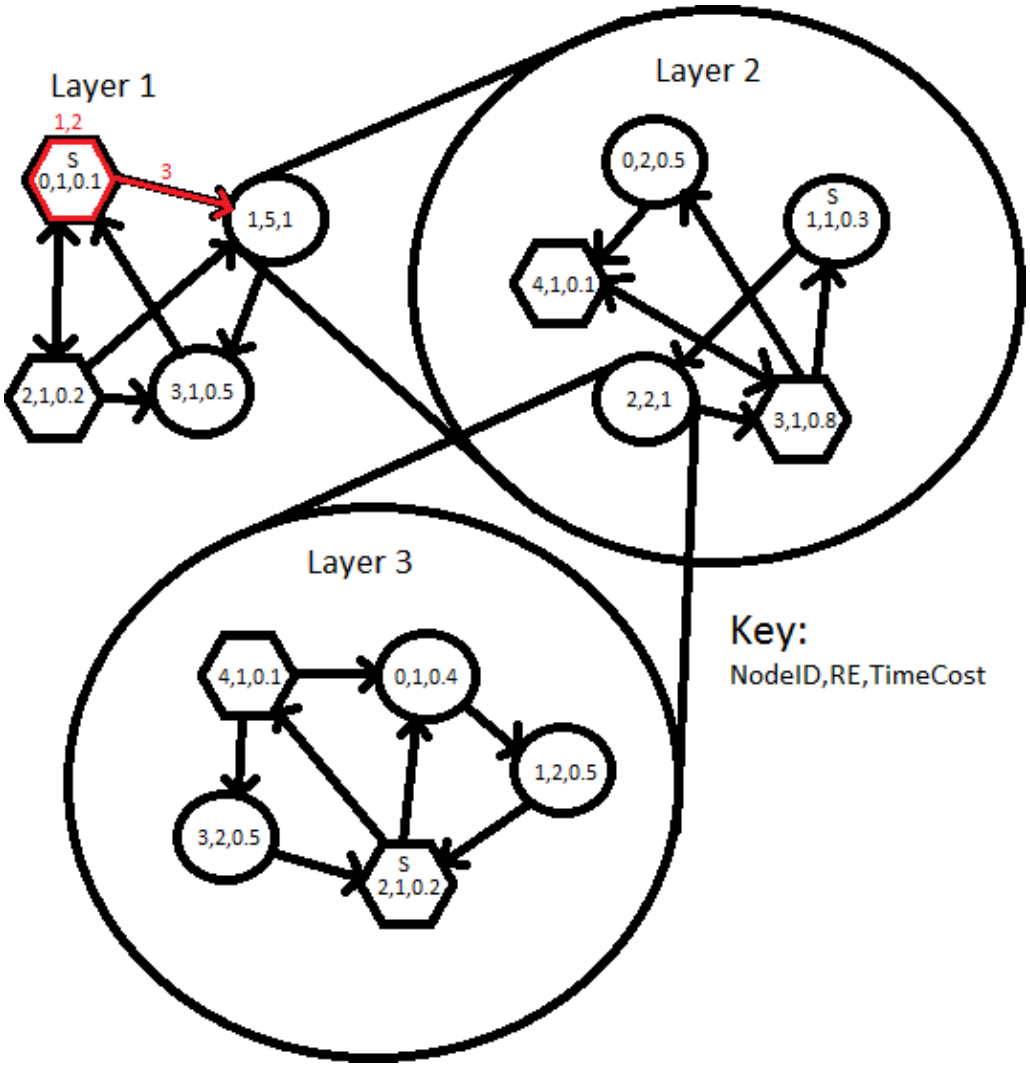


Figure 6.9: Cascading GNP Example Execution: 1

Step	Action	Layer	Node	RE	Time
1	Begin	1	0	1	1
2	Judge	1	0	0	0.9
3	Transition	1	1	5	0.9

Table 6.1: CGNP Example Steps Execution: 1

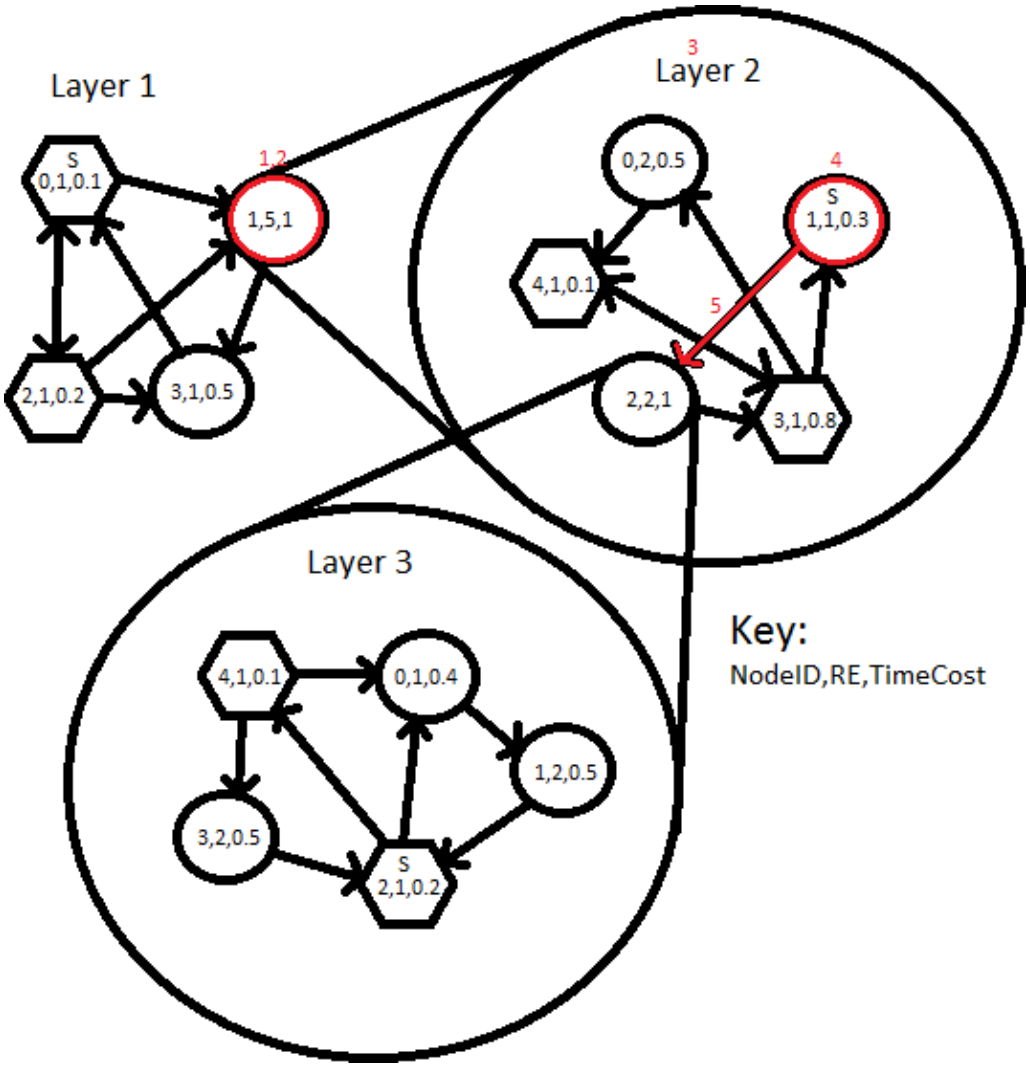


Figure 6.10: Cascading GNP Example Execution: 2

Step	Action	Layer	Node	RE	Time
1	Begin	1	1	5	1
2	Process	1	1	4	0
3	New Layer	2	1	1	1
4	Process	2	1	0	0.7
5	Transition	2	2	2	0.7

Table 6.2: CGNP Example Steps Execution: 2

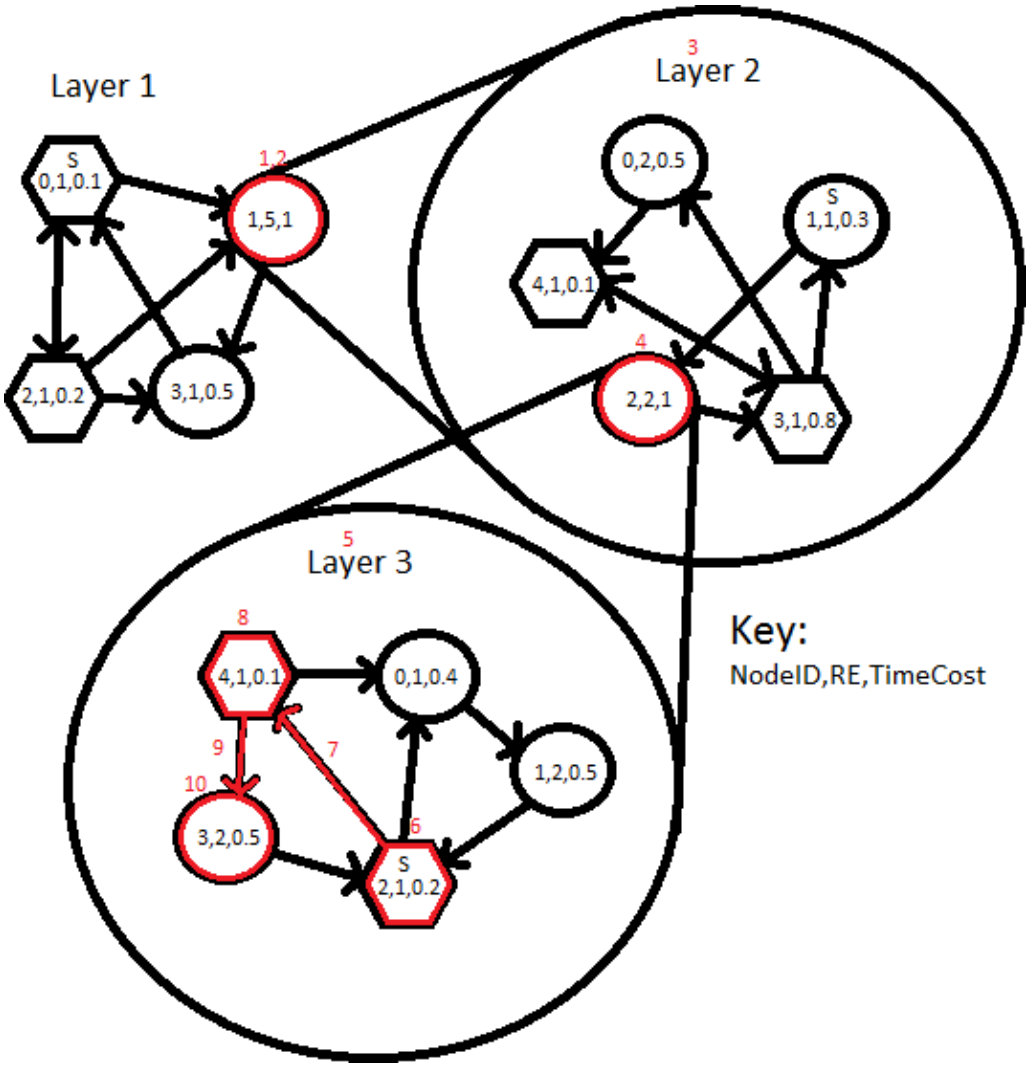


Figure 6.11: Cascading GNP Example Execution: 3

Step	Action	Layer	Node	RE	Time
1	Begin	1	1	4	1
2	Process	1	1	3	0
3	New Layer	2	2	2	1
4	Process	2	2	1	0
5	New Layer	3	2	1	1
6	Judge	3	2	0	0.8
7	Transition	3	4	1	0.8
8	Judge	3	4	0	0.7
9	Transition	3	3	2	0.7
10	Process	3	3	1	0.2

Table 6.3: CGNP Example Steps Execution: 3

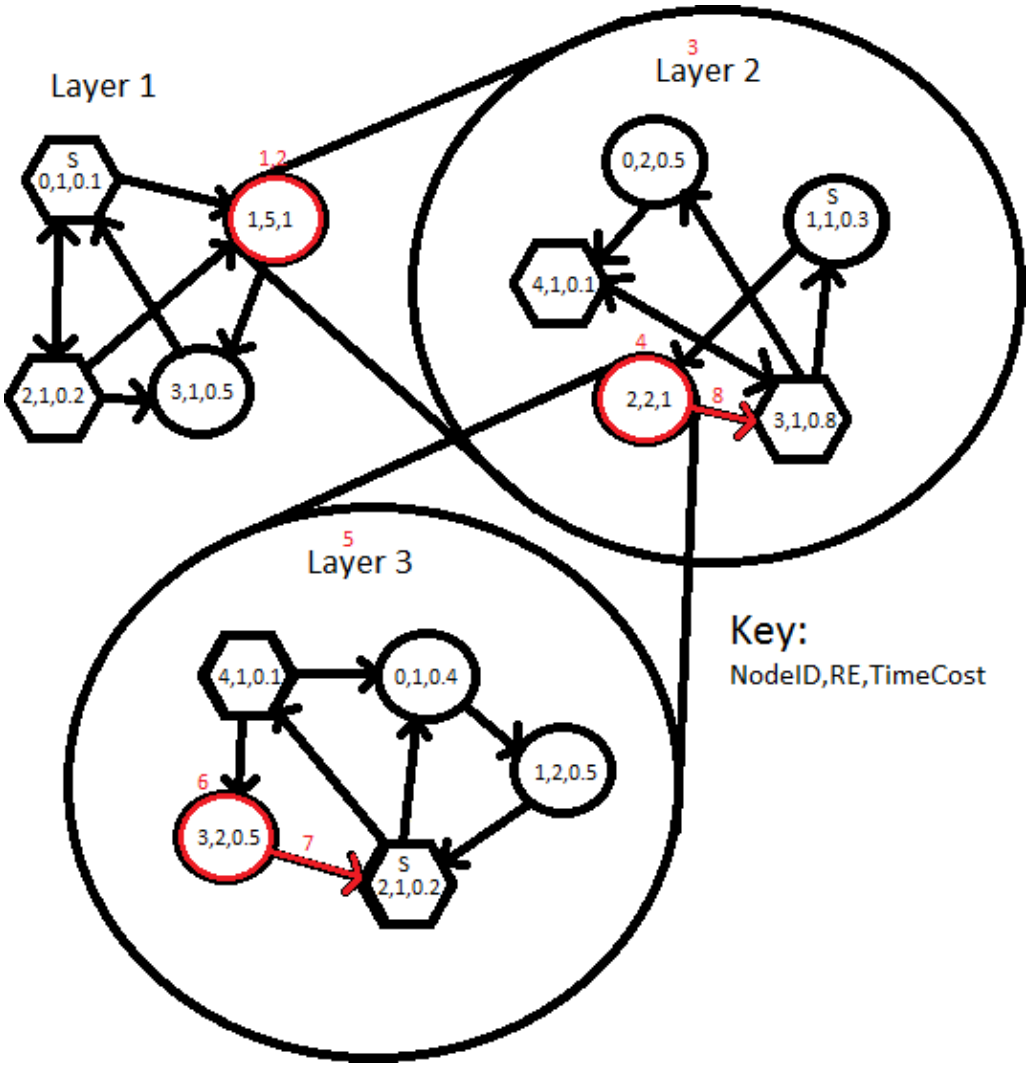


Figure 6.12: Cascading GNP Example Execution: 4

Step	Action	Layer	Node	RE	Time
1	Beginning	1	1	3	1
2	Process	1	1	2	0
3	New Layer	2	2	1	1
4	Process	2	2	0	0
5	New Layer	3	3	1	1
6	Process	3	3	0	0.5
7	Transition	3	2	1	0.5
8	Transition	2	3	1	0

Table 6.4: CGNP Example Steps Execution: 4

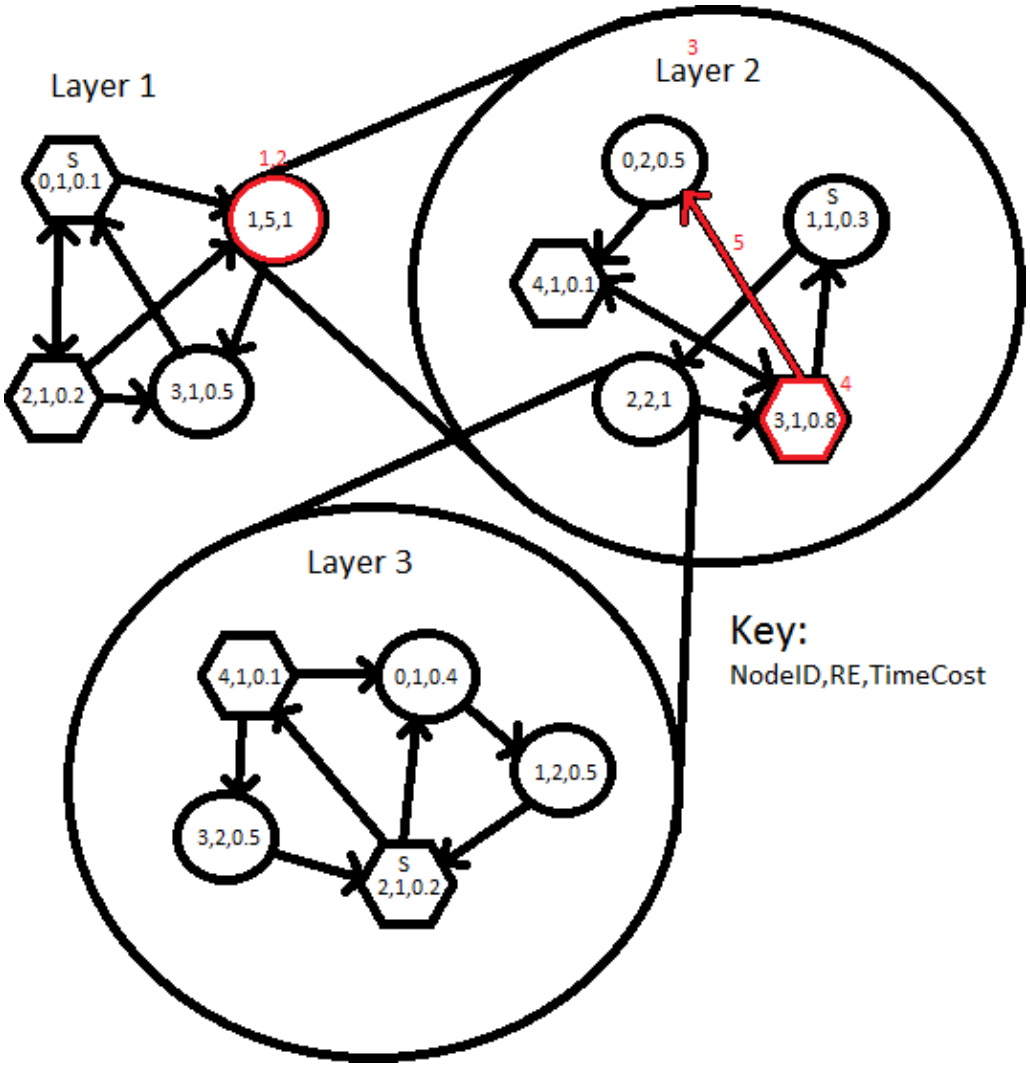


Figure 6.13: Cascading GNP Example Execution: 5

Step	Action	Layer	Node	RE	Time
1	Beginning	1	1	2	1
2	Process	1	1	1	0
3	New Layer	2	3	1	1
4	Judge	2	3	0	0.2
5	Transition	2	0	2	0.2

Table 6.5: CGNP Example Steps Execution: 5

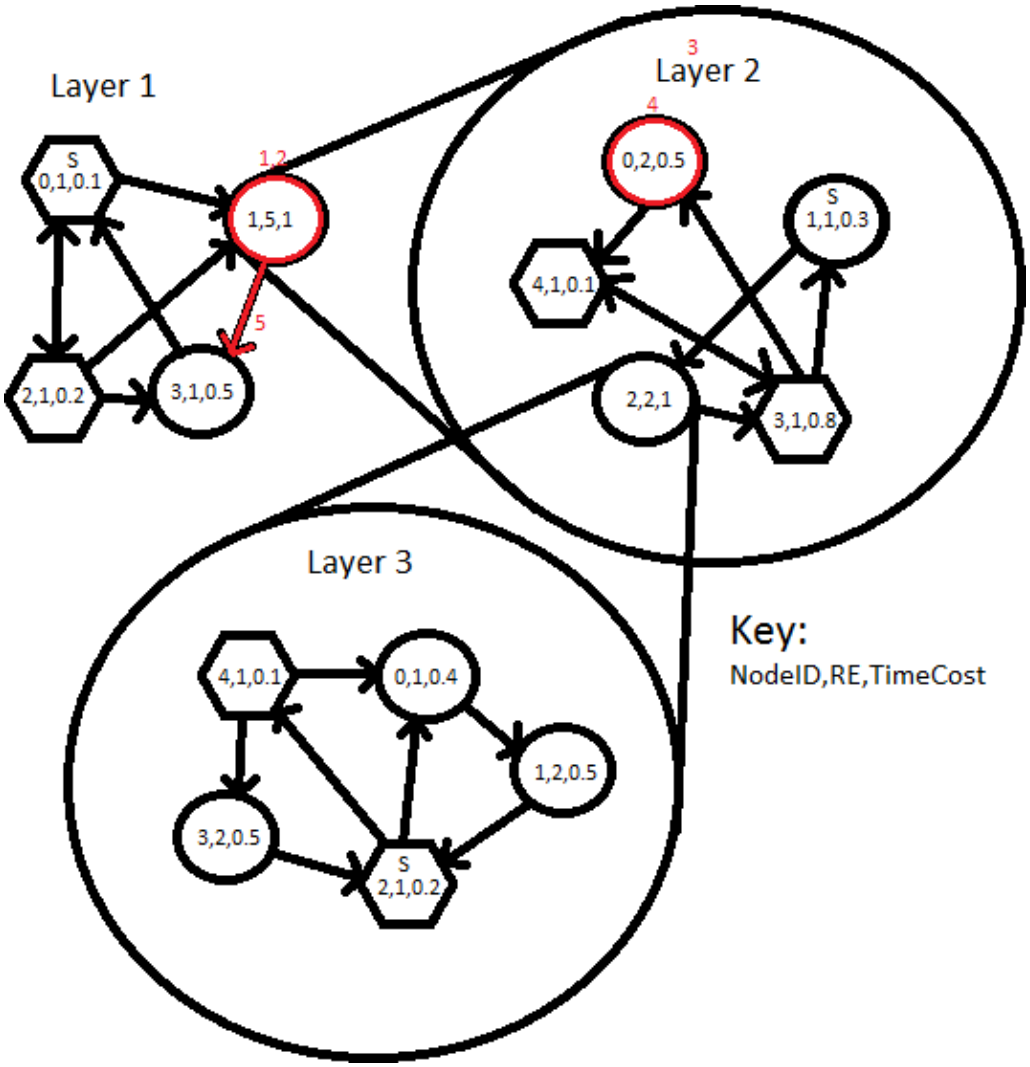


Figure 6.14: Cascading GNP Example Execution: 6

Step	Action	Layer	Node	RE	Time
1	Beginning	1	1	1	1
2	Process	1	1	0	0
3	New Layer	2	0	2	1
4	Process	2	0	1	0.5
5	Transition	1	3	1	0

Table 6.6: CGNP Example Steps Execution: 6

6.2.5 Pseudo-code

In comparison to the pseudo-code for the original architecture shown in Code 6.1, the code for CGNP (Code 6.3) has a few noticeable changes specifically the time management and the repeated executions. There is one other aspect that is not

covered throughout the thesis, that is the "Reset CurrentNode" line this is used to reset any data within a node which may persist between repeated executions. Currently the only use for this section is to reset any subsequent CGNP layers to the start node, this may find use in other nodes developed in the future.

Code 6.3: CGNP Architecture Pseudo-code

```
Initialization:
CurrentNode = StartNode
CurrentExecutionCount = CurrentNode.RepeatedExecutions
------
Each Execution:
TimeAllowance = 1
While TimeAllowance is greater than or
                equal to CurrentNode.Time
        Execute CurrentNode
        CurrentExecutionCount = CurrentExecutionCount - 1
        TimeAllowance = TimeAllowance - CurrentNode.Time
        PreviousNode = CurrentNode
        If CurrentExecutionCount equals 0
                Reset CurrentNode
                CurrentNode = CurrentNode.NextNode
                CurrentExecutionCount =
                        CurrentNode.RepeatedExecutions
        If  PreviousNode is ProcessingNode then Finish
```

6.2.6 Strengths and Limitations

Cascading Genetic Network Programming contains many strengths it is, directly applicable to all all cases that the original architecture is used, it is also more adaptable for larger systems as it can be modularized into smaller components that can be individually trained. Since the cascading feature allows problem subdivision and that the training methods allow data optimisation within the nodes themselves, the developed nodes and trained systems are reusable and can be applied to other related problems. It also allows any other AI system to be easily integrated into a CGNP, either trained or untrained. The algorithm also guarantees finding a local maxima due to the addition of Hill Climbing at the end of the Genetic optimisation process.

All nodes should act independently as there is no form of data transmission between nodes, except for any changes one node makes to the environment, data

transmission between nodes could be developed in the future. Unlike the original architecture the types of nodes themselves can not be changed within the optimisation process, however by training data within the nodes this overcomes this limitation.

6.3 Summary

Genetic Network Programming is a directed graph algorithm that uses Processing Nodes (actions) and Judgement Nodes (decisions), the network is traversed by passing from one node to another completing the respective tasks. Nodes themselves can be specific to the problem they are hoping to achieve, such simply setting the speed of a motor or Artificial Intelligence nodes that also have the ability to control motors, but have additional intelligence. In the original architecture only the connections between nodes are optimised, but has the ability to optimise node types from a pre-compiled library of potential nodes. This optimisation process is achieved via the use of Genetic Algorithms, however could be further improved using Ant Colony Optimisation or Hill Climbing.

The original architecture has been applied to complicated tasks such as elevator control[7]. However the original architecture does contain limitations such as; using binary encoding will increase optimisation times in larger problems, networks are trained all at once therefore its restricted to small problems otherwise the solution space will become too large and the contents of nodes themselves are not optimised therefore a perfect solution will not be found unless the required node or variation is present in the node library.

To approach these limitations a new architecture is proposed in this thesis, called Cascading Genetic Network Programming. It allows multiple level networks to be designed and executed, this increases re-usability of networks and allows problem division into modular components that can be easily solved, while providing the functionality to execute these separately trained networks as part of a cascaded network. To facilitate the new architecture two main changes were made; one of which views the execution times of nodes in terms of a remaining time allowance, if there is not enough time remaining the node will not be executed until the next time the network is called. The second change allows time consuming nodes to repeatedly execute before progressing to the next node, this is specifically useful to allow an infinite number of cascading levels in the network design.

The data optimisation methods also changed so the data internal for every node can be optimised at the developers discretion, this allows more general purpose nodes to be developed, significantly increasing the re-usability of nodes. This also means that there is no need to keep a library of potential nodes, as all nodes in the network are statically defined prior to the optimisation process. An example node would be

a simple IF statement node, within the optimisation process it could decide on its own conditional statement, from the input parameters to the comparator. Therefore this one node type can be used in every situation or problem where an IF statement may be required.

To improve the overall training results, Hill Climbing was included at the end of the Genetic training process. This increases the training time, but guarantees a local maxima result at minimum. This would be useful for any situation where the controller is absolutely critical to the situation.

Cascading Genetic Network Programming has enhanced re-usability and is more modular than the original architecture and can therefore be applied to larger problems, it can also scale down to apply to the same situations where the original architecture was applied and any pre-trained GNP systems can be ported to CGNP with ease.

Chapter 7

Genetic Network Programming Controller Design

In this chapter an approach at making a Cascaded GNP for Robot Soccer goal keeping is covered, the success of the final goal keeper will prove the effectiveness of the overall CGNP architecture introduced in Section 6.2. This goal keeper contains multiple objectives as it must successfully defend the goal, remain inside the goal area and if it is pushed out of the goal area for any reason it must be able to reposition itself completing each of these objectives will result in an autonomous goal keeper controller.

As all these objectives will amount to a rather complicated fitness function, with very little guarantee of finding an acceptable solution the overall controller is split into three subsections; Target Pursuit, Target Rotation and Goal Defending. Splitting the controller into smaller parts also shows how a CGNP controller can be modularized, this allows individual components to be retrained with additional requirements and then plugged into the final controller without the need to retrain the entire network.

7.1 Target Pursuit

Target Pursuit is one of the fundamental aspects of Robot Soccer, it allows the Robot to move to a specific position on the field, pursue a ball or even another robot. In an ideal situation, the robot would get to the specific target as soon as possible. The robot should also end the pursuit facing the target, if the robot wasn't aimed at the target then subsequently kicking a ball would require rotating first. Therefore it would be advantageous if one system both pursued a target and was facing the correct direction.

A Target Pursuit system for Robot Soccer has previously been developed using

Genetic optimised Fuzzy Logic, however this system needs some improvements. If the robot begins very near to the target (around 180 millimetres) at close to 90 degrees from the robots initial position, instead of rotating on the spot and then moving towards the target the Genetic Fuzzy controller will rotate around the ball in a circle. This is shown in Figure 7.1, the red line is the previous path of the robot.

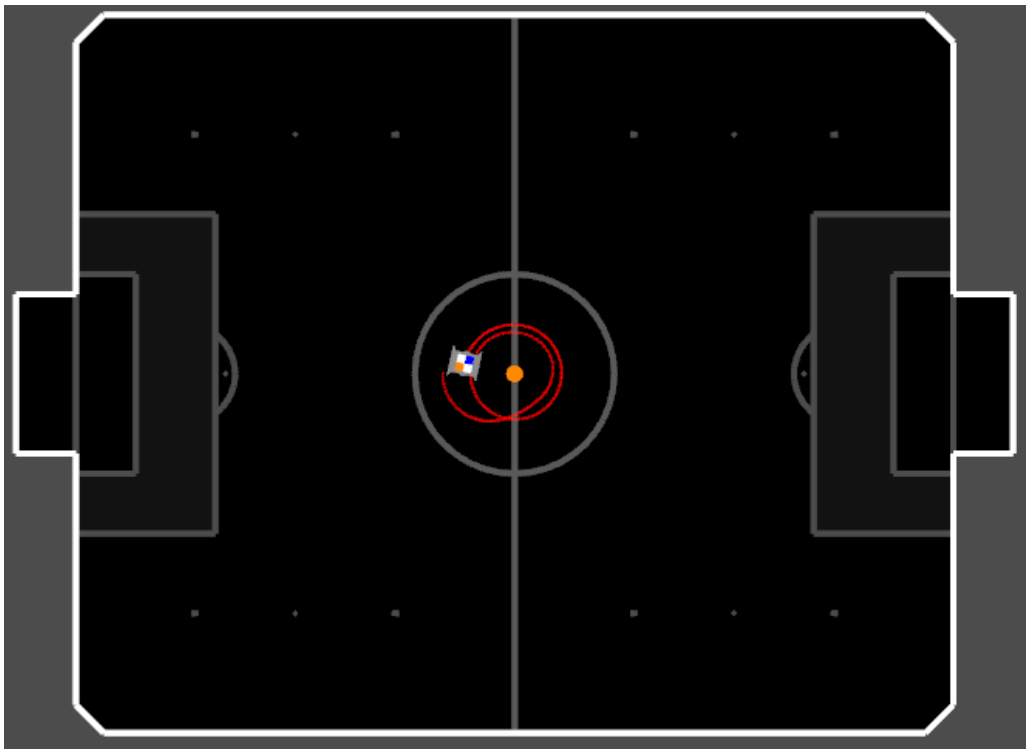


Figure 7.1: Issue with the Genetic Fuzzy Target Pursuit

While this issue could be corrected by retraining the Genetic Fuzzy system, it could also be fixed by using additional basic nodes in a CGNP. Therefore this first CGNP test will prove its ability to improve a pre-trained system and the ability to import other AI systems into the network.

7.1.1 Test Simulation

The Target Pursuit behaviour should be represented in a simulation with specific objectives. The easiest way to represent a Target Pursuit system is when a Robot pursues a ball, a collision between the robot and the ball would show the target has been reached. As previously stated the robot should also face its target once it reaches the target, therefore an ideal simulation would be where a robot collides with a ball and attempts to minimize the angle to the ball when the collision occurs.

In order to represent the entire target pursuit problem for robot soccer, one option is to have a robot for every single location and rotation within the playing

field, however the time it would take to evaluate all these possibilities would be unrealistic. A subset of 24 locations evenly spread across the field, with four 90 degree rotations for each location, will execute within a more realistic time frame and provide a general representation of the problem. This test is conducted five times with different ball starting positions, Figure 7.2 displays the robot starting positions and the numbers represent the ball starting points, these robot positions are shuffled around to accommodate the ball position in the respective test.

Figure 7.2: Target Pursuit Test Cases

Disabling collisions between robots allows multiple instances to be executed on the same field, so multiple robots can be at the same location without interfering with one another, however some collision detection is still required to identify when a robot collides with a ball. This simulation will end as soon as all robots have collided with the ball or after a set time frame.

7.1.2 Objective Function

As is with all objective functions the required components of an objective function comes directly from the requirements of the task, in the case of Target Pursuit the robot should get to the ball as fast as it can and be facing the ball when it collides.

To further improve the controller the angle to the ball at every given step should be minimized, therefore the controller the robot will move directly towards the ball

rather than a slow rotation. Another consideration is that the robot will often oscillate its movement, where the angle to the ball will repeatedly increase and decrease especially when the angle to the ball is small. Equation 7.1 displays the final fitness function attained from these requirements, with the addition of a Worst Robot parameter which ensures that the entire population does well in a task, this prevents the robots from exceeding expectations in some situations and failing to meet the basic requirements in other scenarios.

$$Fitness = \frac{1}{\Sigma_{n=0}^{nRobots}(w_1 \cdot CollisionIteration_n + w_2 \cdot |CollisionAngle_n| + IterSum_n) + w_5 \cdot WorstRobot},$$

where

$$IterSum_n = \frac{\Sigma_{m=0}^{nIterations}(w_3 \cdot |AngleToBall_{nm}| + w_4 \cdot |AngleDelta_{nm}|)}{nRobots \cdot nIterations}. \quad (7.1)$$

Each component of the fitness function is described below:

- CollisionIteration is the effectively the time in which robots collide with the ball
- CollisionAngle is the angle to to the ball from the robots perspective at the time of collision
- AngleToBall is again the angle to the ball from the robots perspective
- AngleDelta is evaluated as the actual angle change from the previous movement to the current position, if it has increased from the previous movement if it decreases then it is treated as zero for that movement
- WorstRobot is the summation of the previous weighted components for the robot that performed the worst in the test

The first four major components of the fitness function (CollisionIteration, CollisionAngle, AngleToBall, AngleDelta) are scaled down to between zero and one and are then weighted, as for the WorstRobot it is scaled up to between zero and 6.3 (the sum of the first four component's weights) before being weighted itself. Therefore the absolute worst fitness possible would be $\frac{1}{11.34}$.

The five weights are used to bias each of the main components of the fitness function. These were were manually changed until the desired results were achieved, their values are listed below:

- $w_1 = 1.8$
- $w_2 = 0.5$
- $w_3 = 0.8$
- $w_4 = 3.2$
- $w_5 = 0.8$

7.1.3 System Training

Very early on when developing this target pursuit controller a need for symmetry was identified when solving this problem, the first few controllers would prefer to travel in a certain direction regardless of what may be an ideal situation. Figure 7.3 shows an extreme situation encountered in these early developments, the nodes used here are the Genetic Fuzzy Target Pursuit, a set of Judgement nodes for the angle to the ball and Processing nodes that set wheel speeds. The problem was not specifically the objective function as it was optimising well, it was more related to personal preference for the task since it would be difficult to modify the fitness function to compensate for this, symmetry was forced upon some of the nodes. The Genetic Fuzzy Target Pursuit system alone is shown in Figure 7.4 as a comparison against the GNP trained version shown in Figure 7.3. In the GNP trained solution, trail marks show relatively straight lines towards the target, which is indicative of better fitness. However, the desired behaviour is a smooth path (without jagged edges) that efficiently reaches the target.

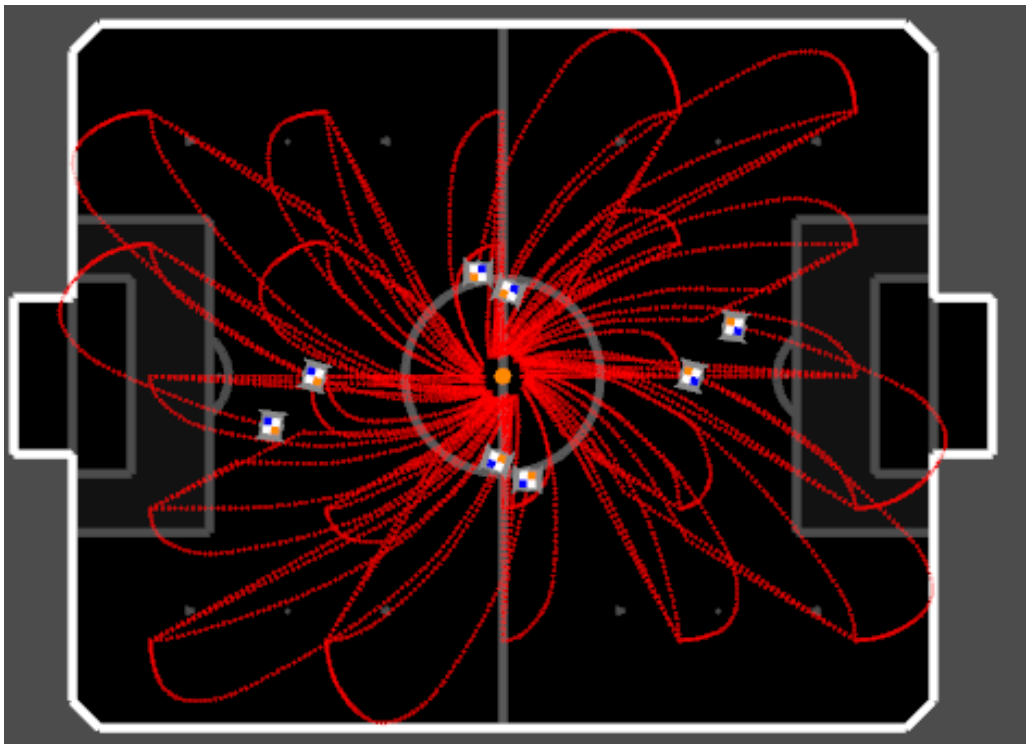


Figure 7.3: Target Pursuit Symmetry Issue

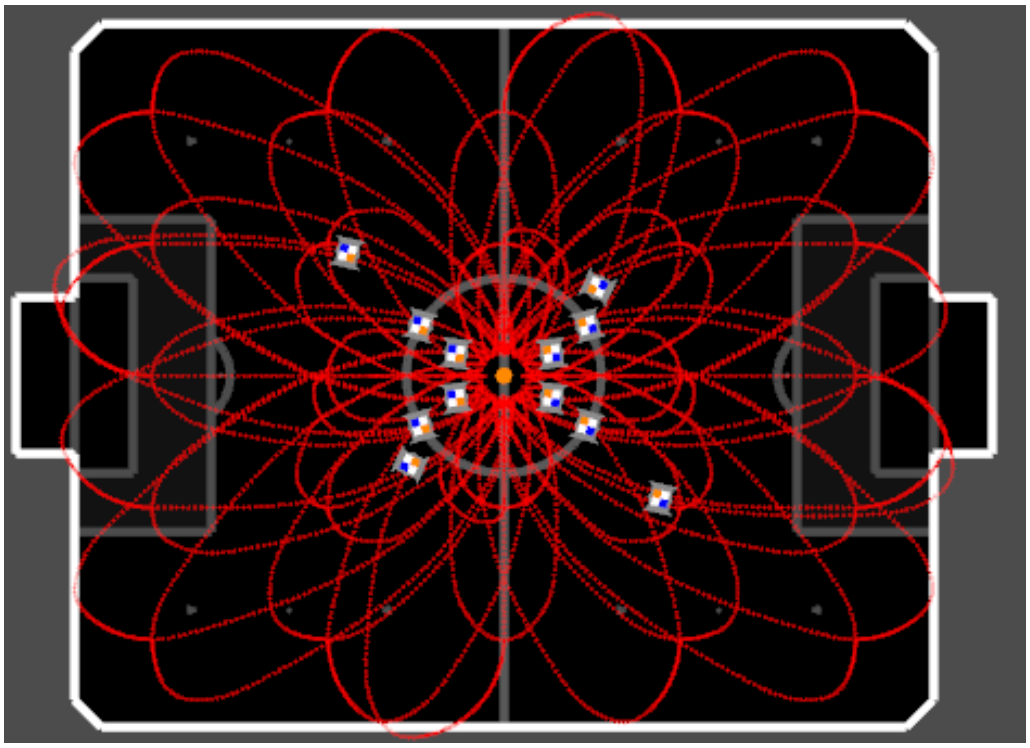


Figure 7.4: Genetic Fuzzy Target Pursuit Simulation

Since the current CGNP architecture does not have the ability to optimise global parameters that can be used to force symmetry across different nodes, a new node type was developed by modifying the original node that purely set wheel speeds to specified values. It was modified so that if the target was on the left of the robot, the wheel speeds will be reversed (right wheel will become left wheel, left wheel will become right wheel). The ability to force a robot to rotate on the spot was also included at this time, so using these two modifications to the original set wheel speed node a single parameter is optimised for both wheel movements. This parameter will be somewhere between negative one and one (inclusive), using a single parameter to represent both wheels significantly reduces the solution space and enforces the symmetrical nature that is desired. This is definitely not an elegant solution, but it worked for the task that was required.

As this situation is primarily to improve on the Genetic Fuzzy controller, it was determined that only a very small network is required. In fact only three nodes were used in the end, the Genetic Fuzzy node which does not optimise any internal data, a modified set wheel speed node which can only rotate on the spot and an if statement judgement node. Despite the small size of the network it contains a large solution space, as both the if statement and set wheel speed nodes optimised internal data. The connections for this network were set as it is such a small network that any connection optimisation would only be detrimental.

Within this training process the network used three nodes:

- Set Wheel Speeds Processing Node with only rotation and using forced symmetry, can be any value between -1 and 1 to three decimal places.
- If Statement Judgement Node set to be the Absolute Angle to the Ball, comparison operator could either be less than or greater than and the comparison value can be between 0 and 180
- Fuzzy Logic Processing Node, this node is set to use a pre-calibrated Target Pursuit system and does not conduct any optimisation

With the optimisation of these nodes the total solution space is 724,362 potential solutions, the genetic optimisation process uses a population of 120 and is evolved for 40 generations, potentially covering 4,920 candidate solutions which includes the initial random generated population.

- Population: 120
- Generations: 40
- Probability of Crossover: 60%
- Probability of Mutation: 8%

7.1.4 Results and Analysis

Shown in Figure 7.5 is the final trained Target Pursuit network, there is not much data shown on this figure as the connections remained the same throughout the training process. The If Statement Judgement node result is: if the Absolute Target Angle is greater than 81 degrees then move to the set wheel speed node, otherwise move to the Fuzzy Logic node. The set wheel speed node was trained to move the wheels at 89.5% power when the node is called, forcing it to spin on the spot towards the ball location.

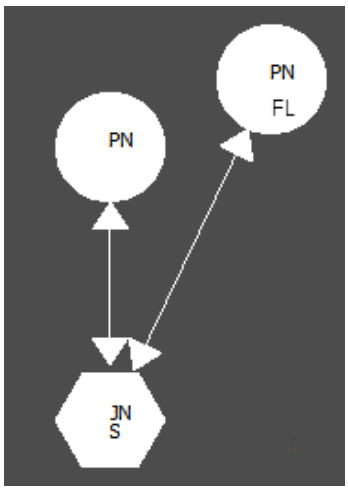


Figure 7.5: Target Pursuit Network

Comparing against the Genetic Fuzzy controller simulation in Figure 7.4 the final GNP controller simulated in Figure 7.6 has much more compact paths than the pure Genetic Fuzzy, they maintain the same shape as they are both symmetrical however the GNP controller requires less time to get to the ball in all cases.

The GNP controller has an overall fitness value of 0.7394, where the pure Genetic Fuzzy has only 0.6539. They may not differ much in overall value but in time critical situations for example intercepting a ball, the GNP controller will get there first.

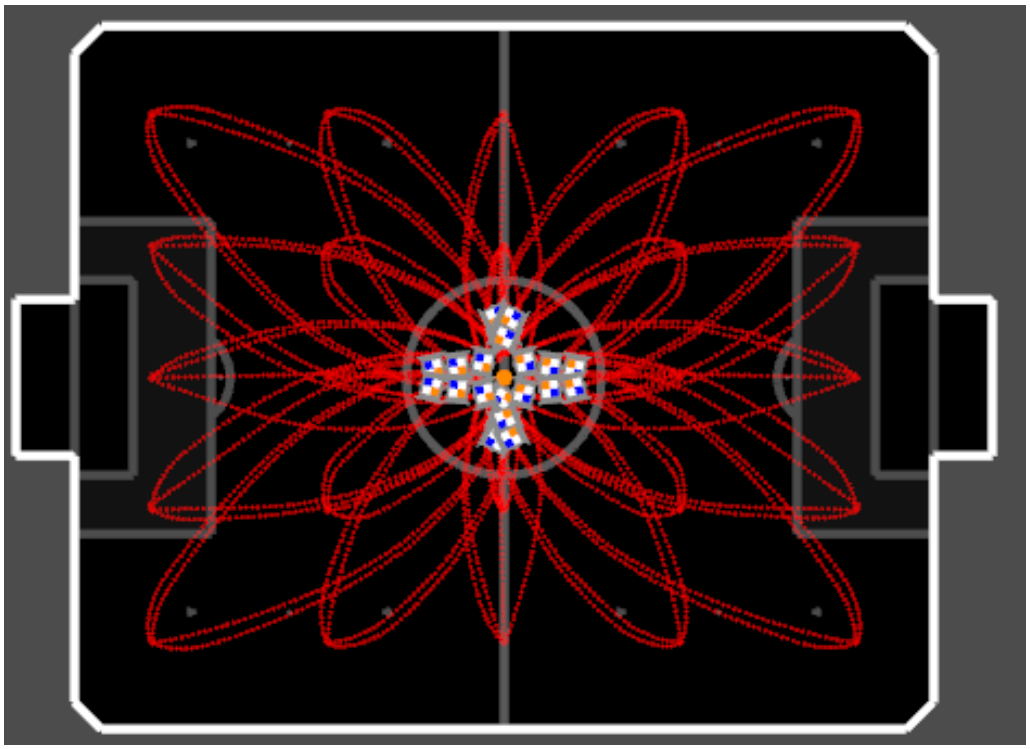


Figure 7.6: Simulation of the best Target Pursuit individual

Referring back to the original problem with the Genetic Fuzzy controller (shown in Figure 7.1) that instigated the need to further optimise the Target Pursuit problem, the GNP version was not trained on this specific issue but it is solved after the optimisation process (shown in Figure 7.7).

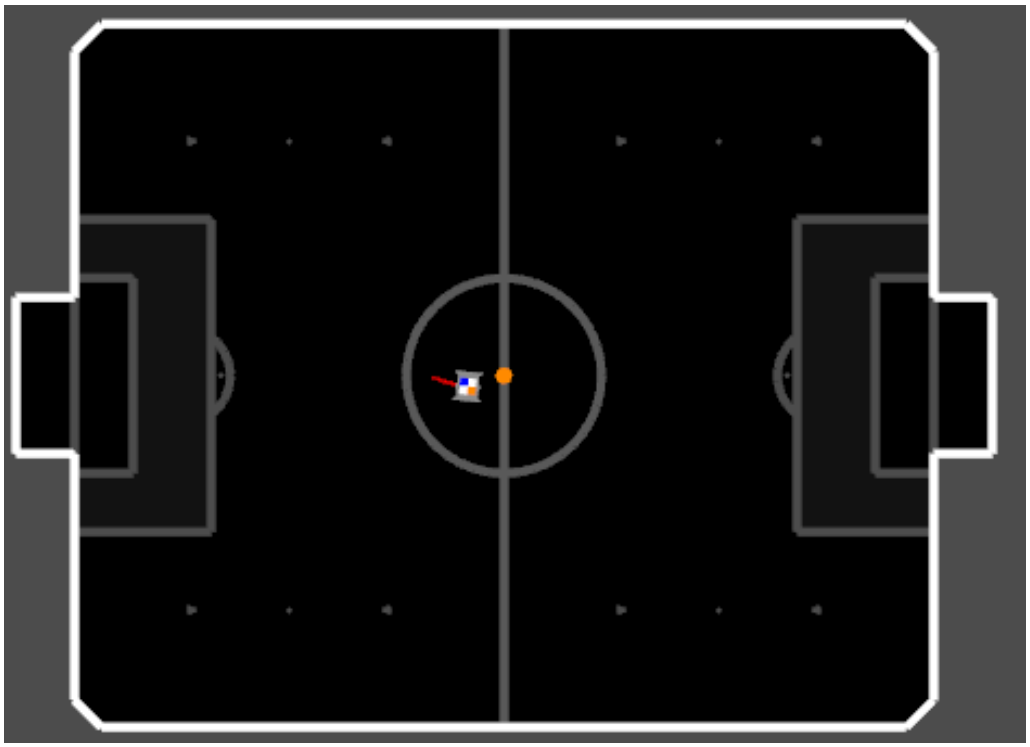


Figure 7.7: Genetic Fuzzy Target Pursuit problem corrected using a GNP

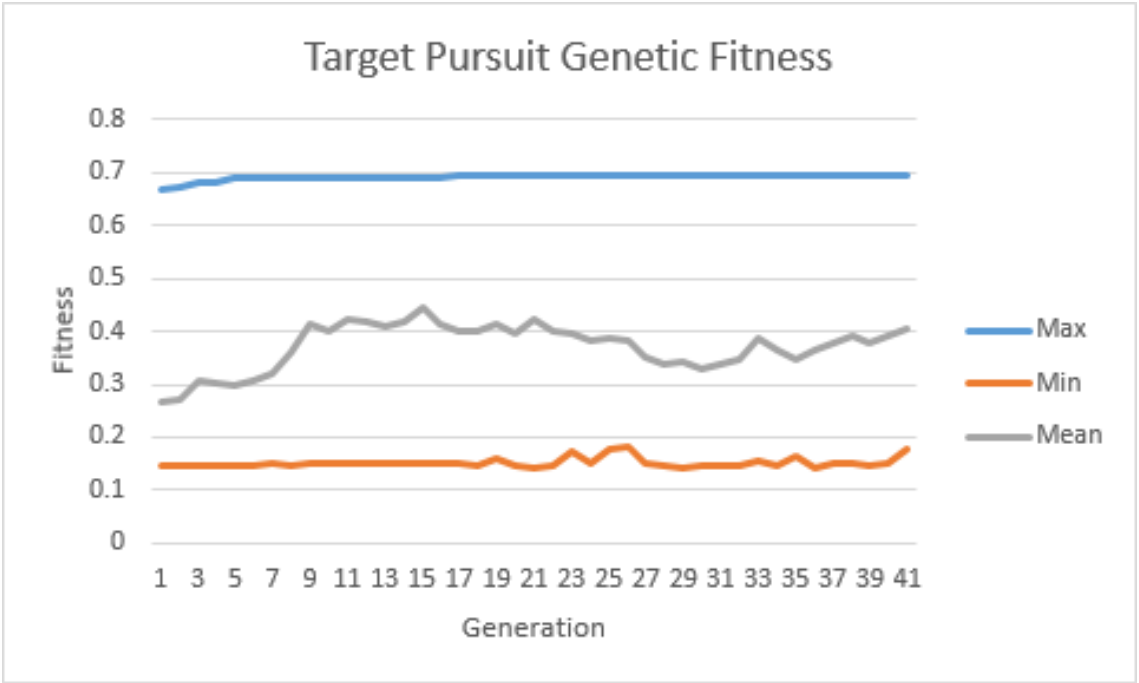


Figure 7.8: Target Pursuit Genetic Optimisation Fitness

Shown in Figure 7.8 is fitness data (computed from Equation 7.1) for each generation in the genetic component of the optimisation process, after this was completed

the best individual had a fitness of 0.69516, this was further improved using hill climbing to a fitness of 0.73947. The maximum value never decreases throughout the optimisation process due to elitism and it can be observed that, the mean values went up and down during the optimisation period, and this likely reflects the evolutionary mechanism to maintain population diversity.

In Figure 7.8 the maximum fitness value has not increased much in actual value since the beginning of the optimisation process, however this slight change significantly affects the performance of the actual controller. In relation to the objective function for this problem (Equation 7.1), an increase of fitness from 0.69516 to 0.73947 as is achieved in the hill climbing could result in a reduction of the average time to collide with the ball by two and a half seconds. With a longer training time this fitness would likely increase further, but gains in actual fitness score will diminish as the population moves closer to a global maximum result.

This optimisation process is using set connections and only a very small network meaning it is not as difficult to train and could potentially have been manually calibrated in less time than the optimisation process. However this was the first test conducted for a CGNP within this research, so this test is used as an initial proof of concept and testing environment while the CGNP architecture was being developed. It could be further improved by reducing the number of robots in the simulation environment, which will in turn speed up the evaluation times. Increasing the variation of node types and also the instances of them would also likely bring better results.

The result described in this section is the best result achieved over 14 different tests, this is in terms of fitness and visual inspection of the controller execution. The save file for this controller can be viewed in Appendix A.1 and figures depicting various steps in the controllers execution is visible in Appendix A.2.

7.2 Target Rotation

Despite the fact that Target Pursuit should directly face a target, there is a trade-off between the time it takes to get to the target and the accuracy of the final angle to the target. Reducing the final angle to an absolute minimum requires more time. Due to this predicament, Target Pursuit was designed to minimize the angle as much as possible, while remaining within time constraints.

There are times where a high angular accuracy is required despite time constraints. One example of this would be when a team is awarded a corner kick, where a robot has the chance to kick the ball towards another team member, in the hopes that they manage to score a goal. Target Rotation is one solution to this high accuracy requirement.

The controller is to be built without any form of external AI systems, to prove that with only simple GNP nodes that this behaviour can still be sufficiently solved. A scenario with very little environmental data and no external forms of Artificial Intelligence, could be considered the worst case when developing any form of AI.

7.2.1 Test Simulation

Since this behaviour is similar to that of Target Pursuit the simulation would also be similar, however the main difference between the two is that Target Rotation requires a higher accuracy. Where Target Pursuit has many robots in a grid with an even spread of angles, Target Rotation should be randomized for both location and rotation this will force the controller to learn how to rotate a fraction of a degree. This is shown in Figure 7.9.

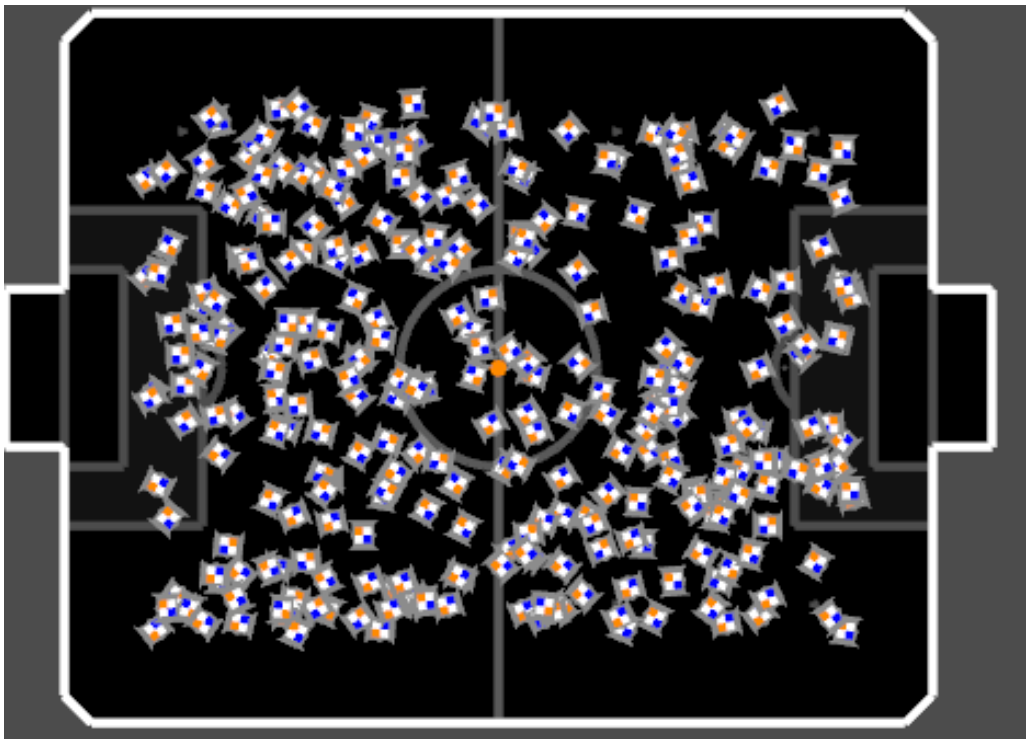


Figure 7.9: Target Rotation Simulation

Like Target Pursuit, this simulation executes multiple independent robots on the same field by disabling physics collisions for the robots. This simulation will run until time has elapsed regardless of what happens, as are not any realistic escape conditions (like colliding with a ball).

One assumption was made when developing this test simulation, that is robots can not actually move from their current position they may only rotate on the spot. Therefore nodes should only be included in the network that adhere to this

assumption, otherwise the desired results will not be achieved.

7.2.2 Objective Function

For Target Rotation the objective function is similar to that of Target Pursuit described in Section 7.1.2, this is because the problem itself is very similar. In fact rotating towards a target could be considered a precursor to any Target Pursuit system. For this test however no collisions are evaluated at all because it is assumed that robots can not move from their initial positions.

There is only three components in this objective function (Equation 7.2), they are AngleToBall, FinalPenalty and again the WorstRobot. These have similar usage as in the Target Pursuit system, AngleToBall the angle to the ball from the robots perspective. FinalPenalty calculated after the simulation has completed it is again the angle to the ball, but is included as an additional penalty because at the end of the simulation process all robots should be directly facing the target with little variation. Lastly the WorstRobot component is used to ensure that the controller is useful in all situations rather than excel under a few conditions. These components are also scaled in a similar way to the Target Pursuit system, where AngleToBall and FinalPenalty is between zero and one before weighting and WorstRobot is between zero and four.

The AngleDelta component in the TargetPursuit system could have been used in this training as well, but was omitted since this system is required to direct itself towards the target as soon as possible regardless of any smooth movement.

$$Fitness = \frac{1}{\Sigma_{n=0}^{nRobots}(IterSum_n) + w_2 \cdot FinalPenalty + w_3 \cdot WorstRobot},$$

where

$$IterSum_n = \frac{\Sigma_{m=0}^{nIterations}(w_1 \cdot |AngleToBall_{nm}|)}{nRobots \cdot nIterations}. \tag{7.2}$$

In this objective function $w_1 = 3.0$ and $w_2 = w_3 = 1.0$, this system did not require much fine tuning at all due to its simplicity.

7.2.3 System Training

This network design uses multiple instances of two different node types, both of which were used in the Target Pursuit controller (Section 7.1.3). The overall network begins with four copies of an If Statement Judgement Node, that is set to use the

Absolute Target Angle as its comparison parameter and four copies of the modified Set Wheel Speed Processing Node, that will flip the wheel speeds if the target is to the left of the robot. This processing node also only allows the robot to rotate on the spot and not physically move from its starting position.

In this instance all eight nodes have their connections optimised as well as some internal data, the Judgement node optimises its comparison operator and the comparison value can be from zero to 180 degrees. Again the processing node will optimise one parameter for the wheel speed, which is between negative one and one to three decimal places. This problem has huge solution space which may not be feasible for finding a global maxima, however since Genetic Algorithms are useful for finding acceptable solutions and the inclusion of hill climbing will find a local maxima, it is likely that a decent result will be achieved from the training process.

Training used the same Genetic Algorithm parameters as were previously used, primarily to see what sort of results will be achieved from this large solution space. These parameters are as follows:

- Population: 120
- Generations: 40
- Probability of Crossover: 60%
- Probability of Mutation: 8%

7.2.4 Results and Analysis

After the training process it is noticed that two nodes that have been optimised out of the network, the final network in Figure 7.10 shows this. On the far left there are two Judgement nodes that do not have a direct link from the start node to the nodes themselves, therefore these nodes will never be executed.

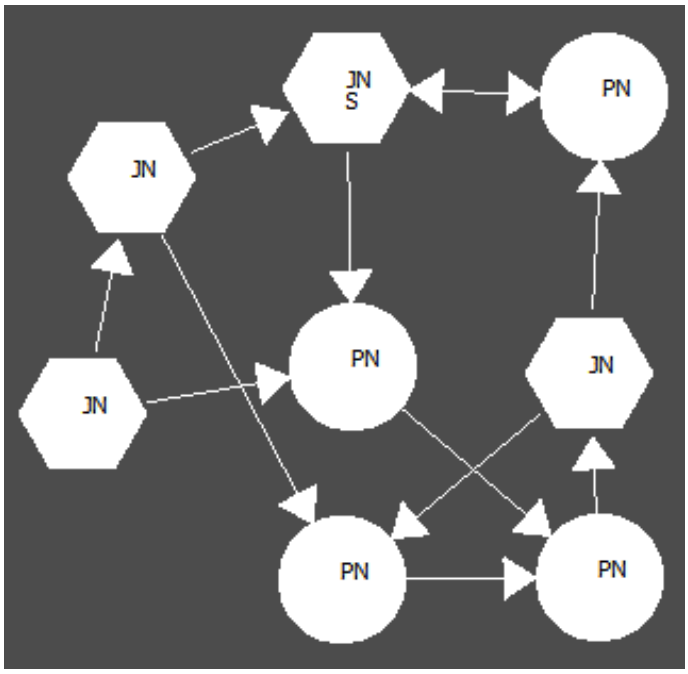


Figure 7.10: Target Rotation Network

This network is described in pseudo-code due to its complexity, therefore in terms of the Judgement nodes it is described in Code 7.1.

Code 7.1: Target Rotation Description

```
// First Judgement Node (Top Left, also Start Node)
if AbsTargetAngle < 90 degrees then
        Rotate toward the ball with 10.5% strength
        Rotate toward the ball with 2% strength
else
        Rotate toward the ball with 69% strength

// Second Judgement Node (Middle Right)
if AbsTargetAngle > 27 degrees then
        Rotate toward the ball with 69% strength
else
        Rotate toward the ball with 0.5% strength
        Rotate toward the ball with 2% strength
```

There are no paths that can be drawn since the robots do not actually move, Figure 7.11 shows the result of the simulation after only a few seconds of execution begins. All robots are facing towards the ball in this figure, however every robot slightly osculates its angle in the actual implementation. This is because the nodes

themselves do not have the ability to provide smooth movement like Fuzzy Logic does. The controller itself has the ability to rotate towards the target at variable distances from the target, but the movement will not be smooth and it will not stop moving once it has reduced the angle to the target down to zero.

This controller could be further improved by using Fuzzy Logic nodes and with a larger population since only a small number of candidate solutions are evaluated. For networks with large number of nodes like this one, it will likely yield an easier network to read if the number of nodes were minimized in the fitness function.

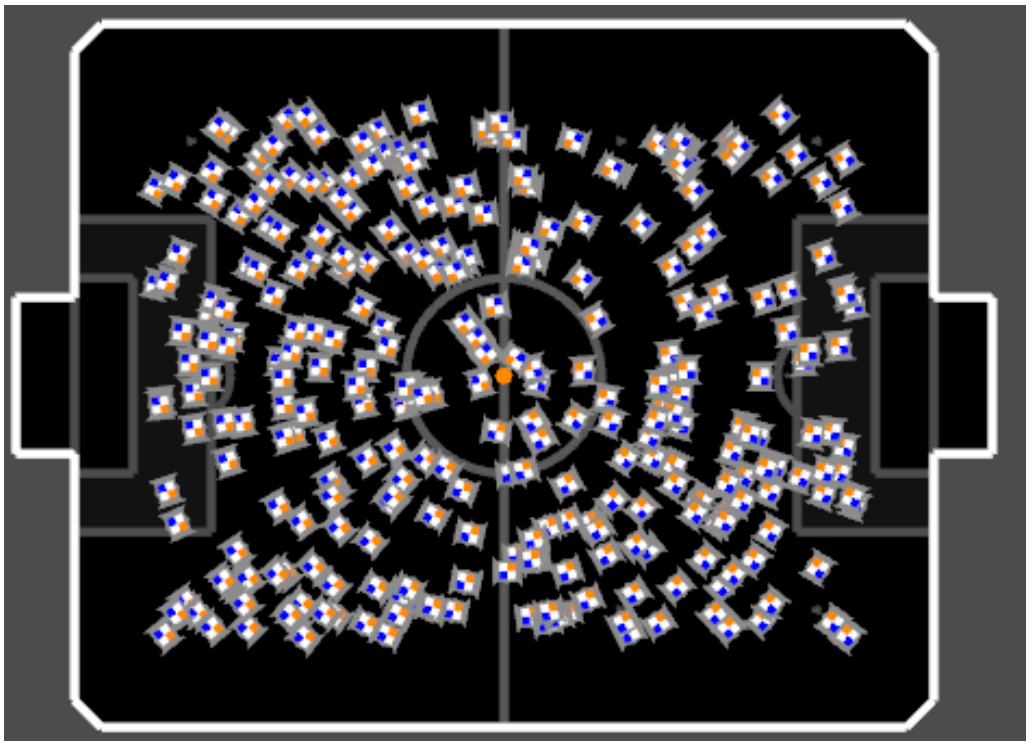


Figure 7.11: Simulation of the best Target Rotation individual

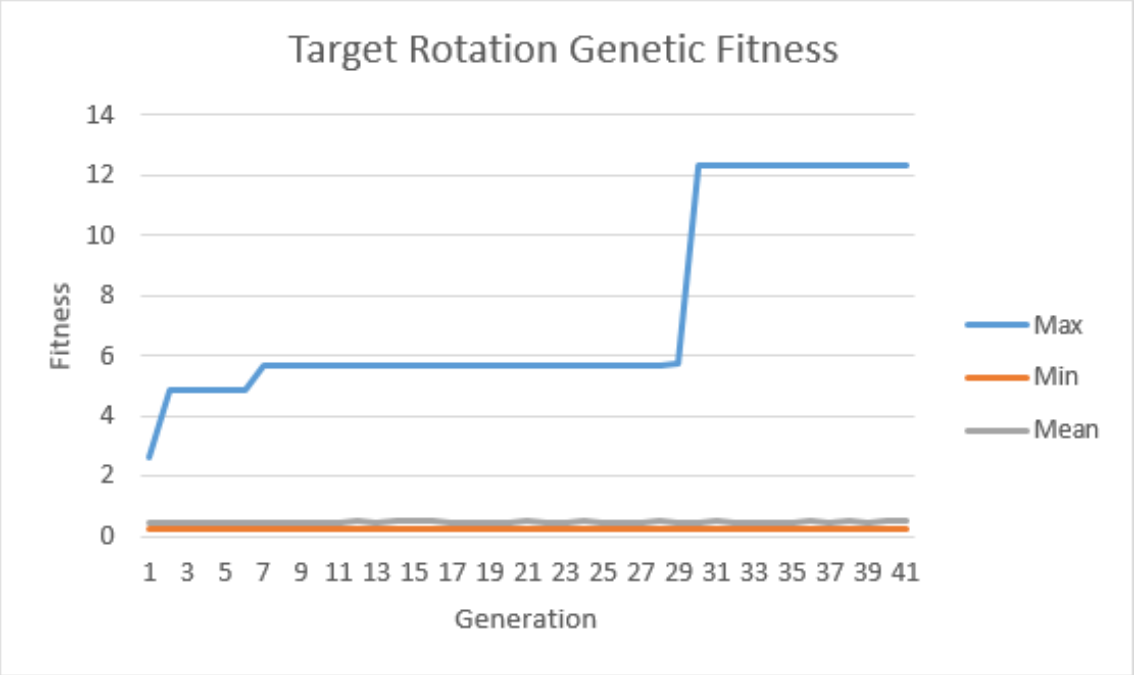


Figure 7.12: Target Rotation Genetic Optimisation Fitness

The Target Rotation Fitness data is shown in Figure 7.12, calculated using the objective function (Equation 7.2). The genetic process obtained an individual with a fitness of 12.3490 and this was further improved to 13.8176 using Hill Climbing. It is noticeable that there is a considerable fitness improvement at generation 29. The stochastic search was fortunate in finding a solution that considerably improved over the previous best result. This controller has a very large solution space and that is apparent in this graph, as the best individual at each generation is effectively an outlier in comparison to the rest of the population.

The solution described in this section is the best result achieved in terms of fitness, over five different training sessions with varying number of nodes. The save file for this controller is shown in Appendix B.1 and figures depicting its execution are in Appendix B.2.

7.3 Goal Defending

Goal Defending/Keeping requires very little explanation, it is arguably the most significant role in soccer. Primarily there to protect the goal from any attacks by the opposing team, if the keeper manages to defend the goal from all incoming attacks, then in the worst case scenario the game will end in a draw.

Programmatically defending a goal is not a difficult task, a simple comparison is the retro computer game 'Pong', where the player takes command of a paddle by

moving it up and down the side of the field to defend an area. The approach to keep the robot parallel to the goal area at all times is useful to minimize any unnecessary movements including rotations, therefore the robot will be able to directly move forwards and backwards to defend the goal without any 'lag' time where it is rotating.

7.3.1 Test Simulation

As the basic goal defending behaviour is being modelled after the 'Pong' game, the Robot should start in the middle and parallel to the goal line. The controller should move up and down to protect the goal as required. A few assumptions were made when developing the controller for this behaviour, that the ball can not move the robot with enough momentum and that the goal keeper always begins within the goal area.

Unlike both Target Pursuit and Target Rotation, the Goal Defending test cases can not be run on the same field. This is because the ball-robot collision physics are required to create a controller that represents the real scenario, instead the robot will be reinitialized to its starting position after each test for a maximum of 1000 test cases. For all test cases the ball starting position, target location and speed will be randomized, each simulation execution will remain the same to give a fair comparison between results. Each test case within the simulation will end when the ball gets into the goal area or is successfully blocked by the robot, after all test cases are evaluated the simulation will end.

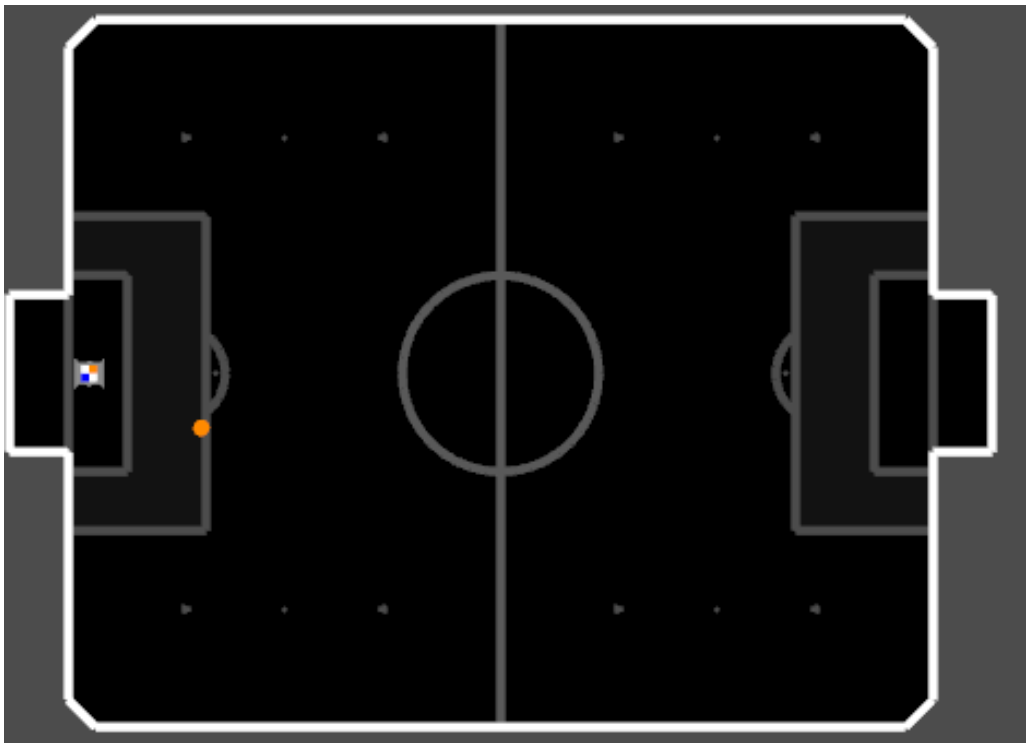


Figure 7.13: Goal Defending Simulation

The initial position for the robot and one potential starting point for the ball is shown in Figure 7.13.

7.3.2 Objective Function

For a goal defending objective function the actual fitness evaluation could be as simple as attempting to minimize the number of goals that the ball achieves. While this would likely be sufficient for the task, there is no method of distinguishing between two different solutions that successfully defend the goal against the same number of shooting attempts.

Therefore it would be better if the problem itself is further separated into into subcomponents, in Equation 7.3 two of these potential parameters are included. For situations where the robot successfully defends a goal the DistanceToBall is included, this component is the closest distance between the robot and the ball within a single test case. Including this component rewards the robot for colliding with the ball directly in the middle of the robots side and slightly punishes it for colliding on the corner of the robot. This separates a save where the robot only just manages to reach a position to protect the goal, from saves where the robot is prepared for the incoming shooting attempt and blocks it in the middle of the robots body.

To help increase the chances of protecting the goal, the interception point where the ball will cross the goal line is calculated, the robot is further rewarded for being

on this spot before the ball reaches the goal. This reduces the need for additional movement when the ball gets near to the goal.

$$Fitness = \frac{1000}{40 \cdot NumberOfGoals + \frac{40}{1000} \cdot \Sigma_{t=0}^{nTests}(DistanceToBall_t) + AvgIter},$$

where

$$AvgIter = \frac{\Sigma_{n=0}^{nIterations}(20 \cdot DistanceToBallInterception_n)}{nIterations}. \tag{7.3}$$

To show a different approach to the previous two controllers, the fitness components are not scaled down to be less than one, they are however weighted by their importance. Each goal that the ball manages to achieve comes with a penalty of 40 points. Since the closest distance to the ball component is only included in cases where the goal was actually protected using its weight it is scaled between zero and 40 points, this keeps the value under the 40 point penalty which is given for an unsuccessful save. This approach was inspired by Riley's[8] objective function for goal scoring (Figure 3.20).

The last component is the DistanceToBallInterception, has a reasonably small weight of 20 as it is significantly less important than protecting the goal but is useful to reduce the amount of movement required when the ball is near to the goal.

7.3.3 System Training

The design of this network possesses an intentionally large solution space, which is likely too large for the Genetic Algorithm to sufficiently solve. The node connections are not optimised within this network, but each node optimises all internal data like previous networks. Considering that the Target Rotation controller in Section 7.2 successfully completes the basic objective despite a large solution space, this test is to evaluate how it will perform in a larger solution space with a more complicated problem.

This network contains four If statement Judgement Nodes and five wheel speed setting Processing Nodes, these nodes are structured to follow a set of logical steps for solving the problem. The first judgement node determines if any movement should actually take place, or if the robot is already positioned on the interception point between the ball and the goal. The next judgement if movement is required determines if the interception point is in front of the robot or behind, then the last set of judgements evaluate the distance of the robot from the interception point thus allowing fine tune movements when the robot is near to the interception point. Each of the processing nodes are forced into only moving forwards and backwards

with no rotation, as this controller is modelled after the paddle in the 'Pong' game. This network structure can be seen in Figure 7.14, the expected output for a trained network is that one of the pairs of nodes at the bottom of the figure will be positive and the other pair would be negative, one node inside each pair should also be much closer to zero than the other to allow for fine tuned movements.

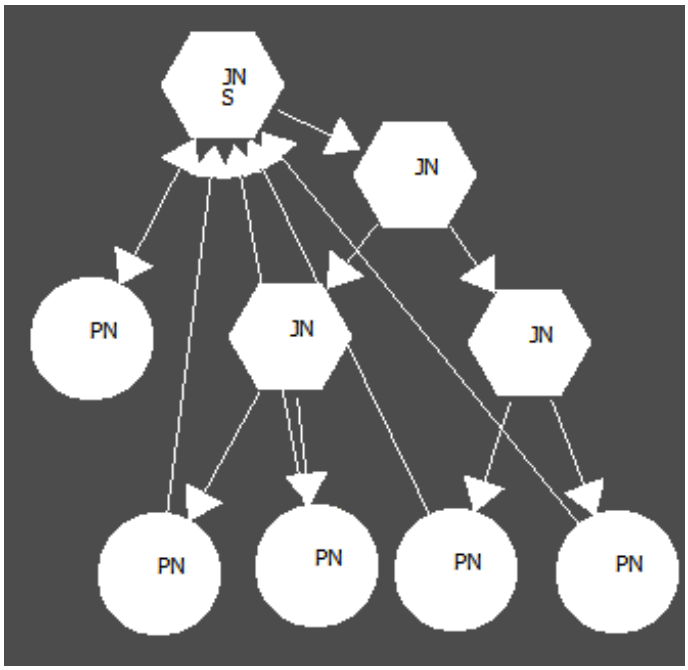


Figure 7.14: Goal Keeping Network

As the Goal Defending network has a larger solution space than both the Target Pursuit and the Target Rotation networks, this network uses an increased population size and number of generations. The initial predictions for this network optimisation is that the Genetic Algorithm will not be able to complete the objective, due to the large solution space and the complexity of the problem.

The Genetic Algorithm parameters are as follows:

- Population: 400
- Generations: 60
- Probability of Crossover: 60%
- Probability of Mutation: 8%

7.3.4 Results and Analysis

Since this network contains constant connections, it does not have the ability to remove unrequired nodes from the network. After the training process however it

is noticed that some of the judgement nodes have conditions that have one particular outcome in nearly every situation. This is effectively the optimisation process removing one of the branches from the judgement, which also makes the judgement node itself redundant. The only judgement node that will regularly visit both branches is the node that determines if the interception point is in front of the robot or behind. Therefore only three nodes will be regularly used in the goal defending CGNP, that is the judgement node that determines if the ball is in front or behind and two processing nodes that will either move the robot forward or back.

This outcome is due to an insufficient objective function and the assumption that multiple processing nodes will be required to regulate the robots speed. The robot will move forward as far as it can after colliding with the ball and successfully defending the goal, the robot will then collide with the northern wall and remain there until the next test begins. When this new test begins its position is reset to the middle of the goal area, this behaviour can not be used in the real implementation or the multi-behavioural controller as the robot itself will never have its position reset. This goal keeper fault is shown in Figure 7.15.

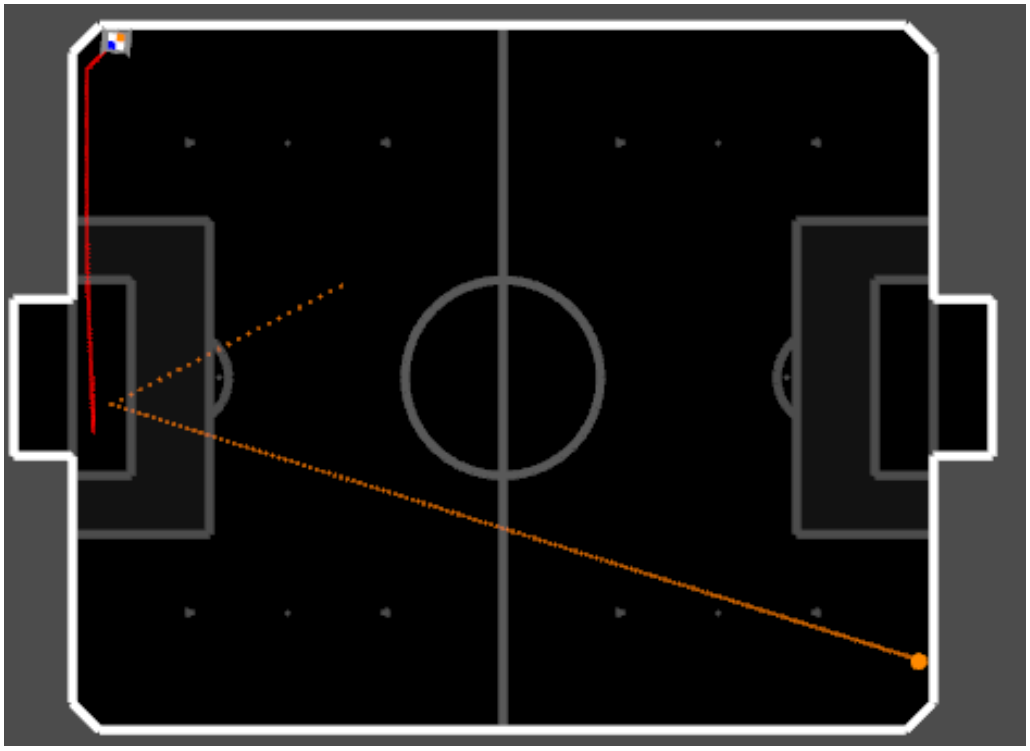


Figure 7.15: Goal Keeping Fault

To correct this flaw the network was manually modified to stop movement when within 5 millimetres of the interception point, for future calibration the objective function could be modified to include conditions that will force the robot to reposition itself in the middle of the goal area. New nodes should also be added to judge

the distance to the middle of the goal area. This controller managed to defend the goal from 99.1% of the incoming shooting attempts despite this flaw. But as the controller requires manual modification to make it feasible for the real application, this success rate should not be expected from the Multi-Behavioural controller.

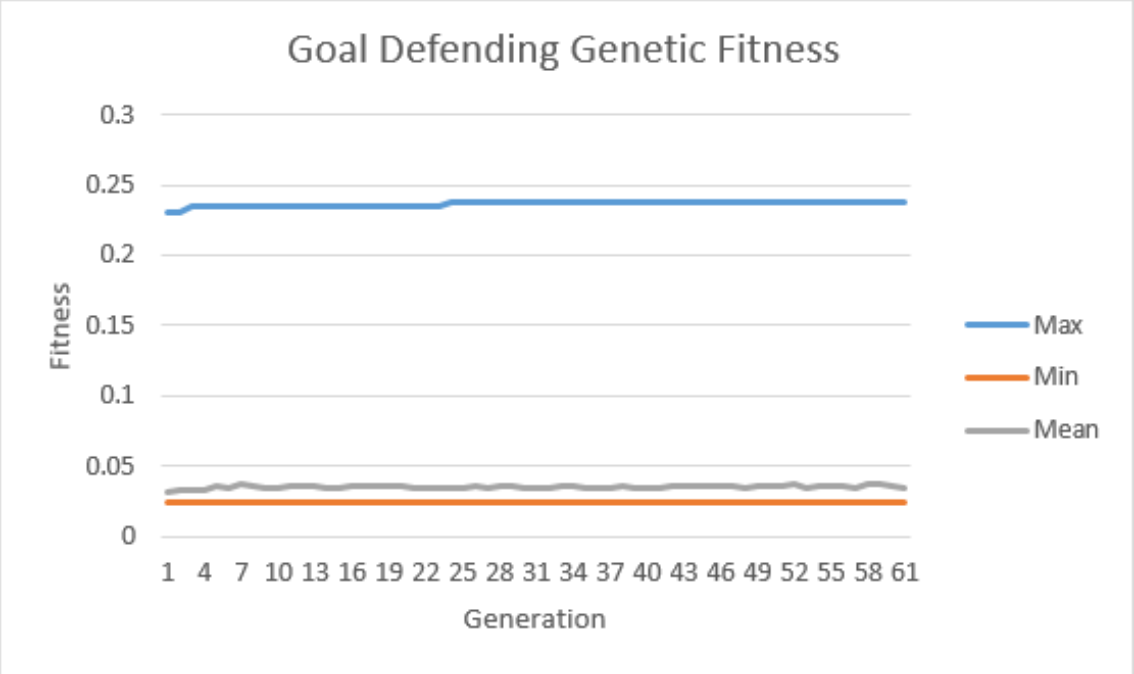


Figure 7.16: Goal Defending Genetic Optimisation Fitness

Figure 7.16 shows the fitness data (from Equation 7.3) for each generation, the best individual had a fitness of 0.23841. Hill climbing slightly improved this result to 0.23852. Within the graph there is a considerable difference between the Maximum values and the Mean, therefore it is likely that the population achieved a high fitness at the beginning of the training process by chance and that the population may not have been large enough to sufficiently represent the solution space. Further training for this controller should be tested with a larger population for any differences in the training process, however a larger population will result in significantly increased training times.

In the initial randomization of this controller, a solution was obtained with a large fitness. This is largely due to the size of the initial population and luck, other training situations for this controller had a more noticeable fitness increase when training. As the initial population contains a large fitness individual, the subsequent gains are minimal. This could also be attributed to the optimisation parameters themselves, where a difference in one degree for a judgement node that evaluates an angle will make very minimal change in overall fitness.

This controller was one of four results acquired from different training processes,

these training processes were used to fine tune the objective function by visual inspection of the trained controller execution. The save file for this controller is shown in Appendix C.1 and various stages of execution for this controller is shown in Appendix C.2.

7.4 Multi-Behavioural Goal Defending

Unfortunately a Pong-like goal defender produced in section 7.3 is not enough for the robot soccer environment, if the robot is hit with enough force then it can be pushed out of the goal area or rotated - leaving the goal defenceless.

A controller with the ability to defend a goal, maintain its position within the goal area and had a high accuracy when rotating to become parallel with the goal, would be superior to its predecessor which will only move forward and back in the hopes of defending the goal.

This goal keeper is made using the cascading facilities in CGNP, it allows integrating the three previous controllers into a new layer. This new layer will only need to decide under what conditions the sub-layers are executed, this is a relatively small problem in comparison to some of the previous controllers, but is necessary to provide the additional functionality.

7.4.1 Test Simulation

The multi-behavioural goal defending simulation is very similar to the one described in section 7.3.1, instead the robot will no longer have its location reset after every test case and the goal keeper can begin its first test anywhere within the left quarter of the field. This is shown in Figure 7.17.

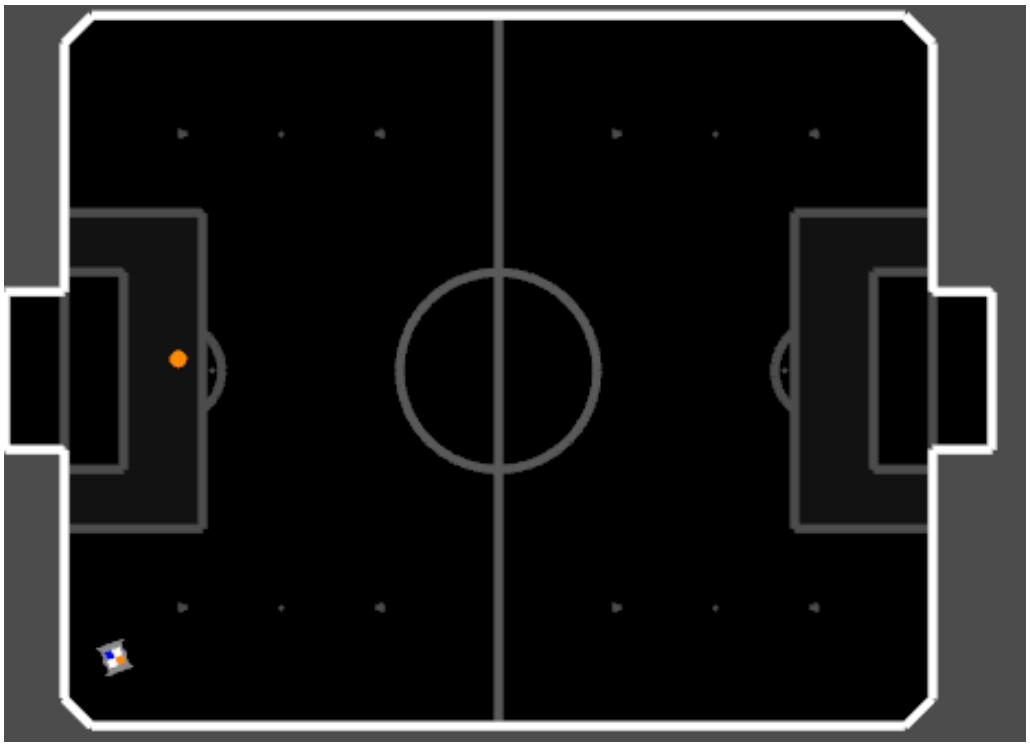


Figure 7.17: Multi-Behavioural Goal Keeper Simulation

In this situation the robot will need to initially position itself correctly within the goal area and rotate to the correct angle in order to defend the goal. With subsequent collisions between the robot and ball, the robot will eventually need to reposition itself in the goal area otherwise it will get stuck and every subsequent ball will go straight into the goal.

Once again the controller will encounter 1000 randomized test cases, once all have either been averted or failed the simulation will end. Each execution of the simulation will remain the same to allow a fair comparison between controller results.

7.4.2 Objective Function

This multi-behavioural goal keeper could use the exact same fitness function for the previous goal defender in Equation 7.3. However it will take a lot longer to find a suitable solution since this multi-behavioural goal keeper can begin anywhere on the left quarter of the field and the previous version only considered defending the goal. Any controller built using the previous equation for this test environment will result in a robot that attempts to defend the goal wherever it is, be it next to the goal or half way across the field.

Equation 7.3 can be extended to consider the robots position relative to the goal line and its angle as well, this extension is shown in Equation 7.4.

$$Fitness = \frac{1000}{40 \cdot NOfGoals + \frac{40}{1000} \cdot \Sigma_{t=0}^{nTests}(DistanceToBall_t) + 20 \cdot AvgIter},$$

where

$$AvgIter = \frac{\Sigma_{n=0}^{nIterations}(DistBallIntercept_n + 2.5 \cdot GoalLineDist_n + |AngToN_n|)}{nIterations}. \tag{7.4}$$

This equation introduces two additional components, the distance to the goal line and the angle to north. The goal line in this case is an arbitrary line that is in-front of the goal area, this is where the robot should be positioned and is close enough to the goal for the robot to defend it, without actually entering the goal area. The distance to the goal line is from the robots current position, to the closest point on this line. The angle to north is quite literally the angle from the current rotation of the robot, to facing directly up from the robot in this birds eye view.

The angle to north is weighted by 20 points as it is less important than the actual goal defending and the distance to the goal line is weighted by 50 points as the robot should always be in the goal area regardless of whether it successfully defends the goal or not, they serve as a recommendation that the robot should be in-front of the goal and face directly north so any incoming balls will hit the side of the robot.

7.4.3 System Training

The Multi-Behavioural Goal Defending CGNP is an amalgamation of the three previously optimised controllers, Target Pursuit, Target Rotation and Goal Defending. This network is designed with one instance of each of these separate CGNP controllers, as theoretically additional instances would be redundant. Four Judgement nodes are also included into this network to allow it to branch between each controller as the environment changes.

All the second layer CGNP nodes remain constant throughout the optimisation process, as they have already been optimised for their respective tasks. Within the top layer network all connections between the nodes are optimised, as well as the internal data for the Judgement nodes. The four Judgement nodes have their input data set, the comparison operator and comparison values will both be optimised. These judgement nodes are as follows:

- One node that evaluate the distance from the robot to the goal line, which allows the decision when Target Pursuit should be used

- One node to evaluate the absolute angle to north, allowing the network to decide when to use Target Rotation
- One node to evaluate the distance between the robot and the interception point, once again this node is assumed to be redundant for the application
- The last node will evaluate whether the interception point is in front of the robot or behind, this is included as it was the most prominent node within the Goal Defending behaviour

For both Target Pursuit and Target Rotation nodes, the actual target is manually changed. Target Pursuit will now pursue the nearest position on the goal line and Target Rotation will rotate towards the northern wall (top wall) of the soccer field. The Goal Defending nodes remain unchanged from the original optimisation process.

Despite that all complicated behaviours have been previously optimised and this controller is only used to amalgamate them by optimising the connections and conditions that they are executed, a large population and number of generations are used to ensure that an effective multi-behavioural goal keeper is achieved at the end of the training process. The GA parameters are the same as those used in the Goal Defending optimisation, these are:

- Population: 400
- Generations: 60
- Probability of Crossover: 60%
- Probability of Mutation: 8%

7.4.4 Results and Analysis

The resulting network contains a two Judgement nodes that will be evaluated to the same result in every situation and one Judgement that will never be executed as it does not have a direct connection from the start node, these nodes are shown in Figure 7.18. These excess nodes can be manually removed and the connections updated with the final result, this is shown in Figure 7.19.

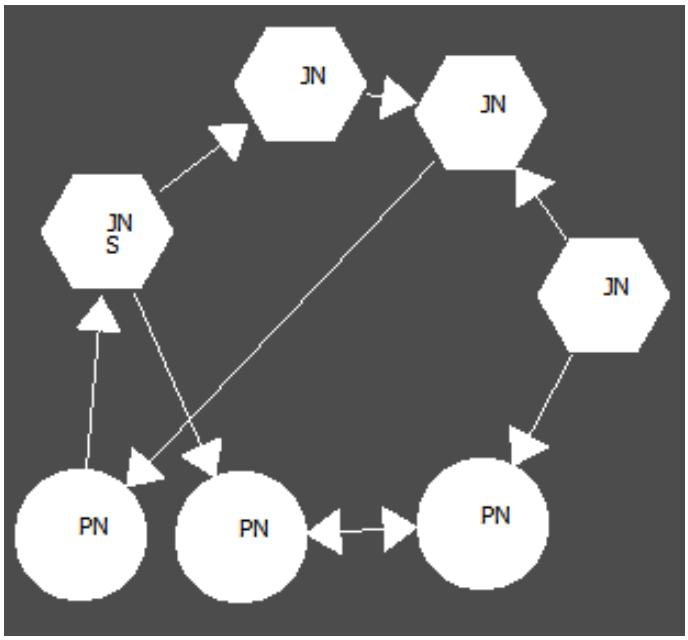


Figure 7.18: Multi-Behavioural Goal Keeper Trained Network

As can be seen in Figure 7.19, there is only one Judgement node remaining and three Processing nodes. This Judgement node will evaluate whether the distance from the robot is larger than 113.72 millimetres, if it is then the Target Pursuit node (bottom left) will be engaged. Another aspect to note with this network is that once the Robot is near to the goal line, control will then begin repeatedly looping between Target Alignment and Goal Defending. Once this loop begins the Target Pursuit node will never be executed again, therefore in extreme situations where the Robot is moved a large distance from the goal area, the goal keeper will not be able to realign itself if required.

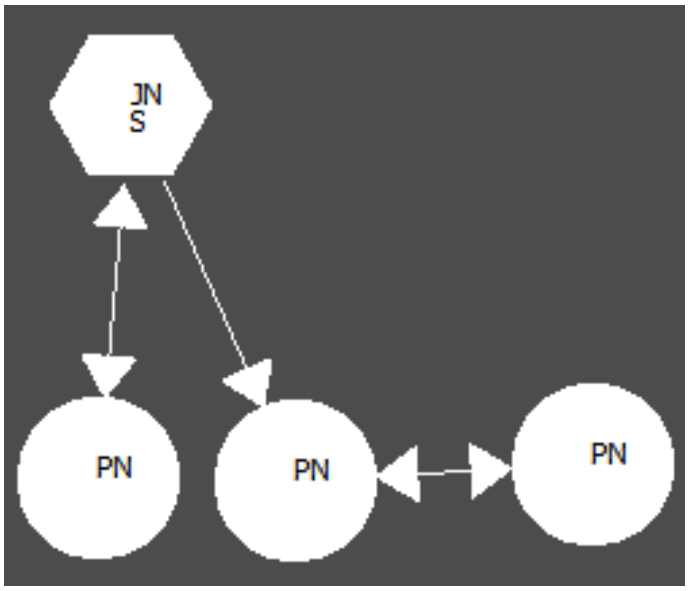


Figure 7.19: Multi-Behavioural Goal Keeper Reduced Trained Network

This trait is undesired but easily corrected with one of two solutions. The Goal Defending node (far right) could connect back to the start node or alternatively half way through the training process, the robot's position could be randomly relocated to another location on the field, this would force the optimisation process to retain access to Target Pursuit if it is required. It is also noted that an additional penalty should be added to the objective function for this controller, to penalize the robot for entering the goal itself.

Despite the limitation in the network, this multi-behavioural goal keeper successfully positions itself in the goal area at the beginning of the simulation (shown in Figure 7.20) and protects the goal from 91% of the goal shooting attempts. This result would likely be improved with additional training for both the basic Goal Defending controller and the Multi-Behavioural controller.

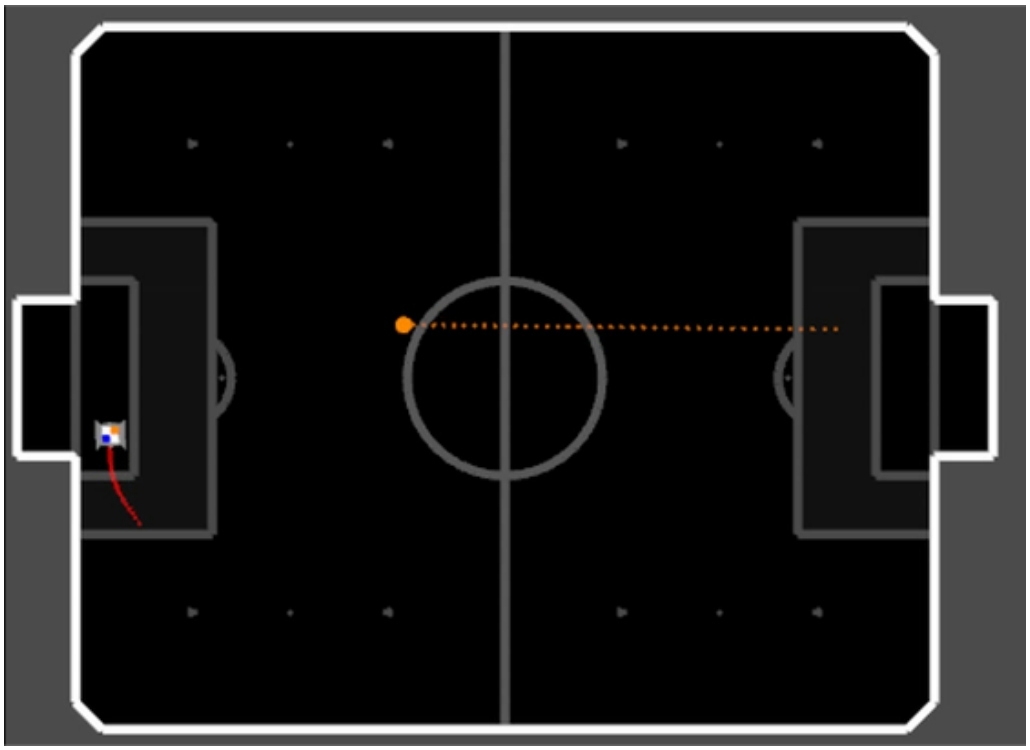


Figure 7.20: Multi-Behavioural Goal Keeper Repositioning Itself

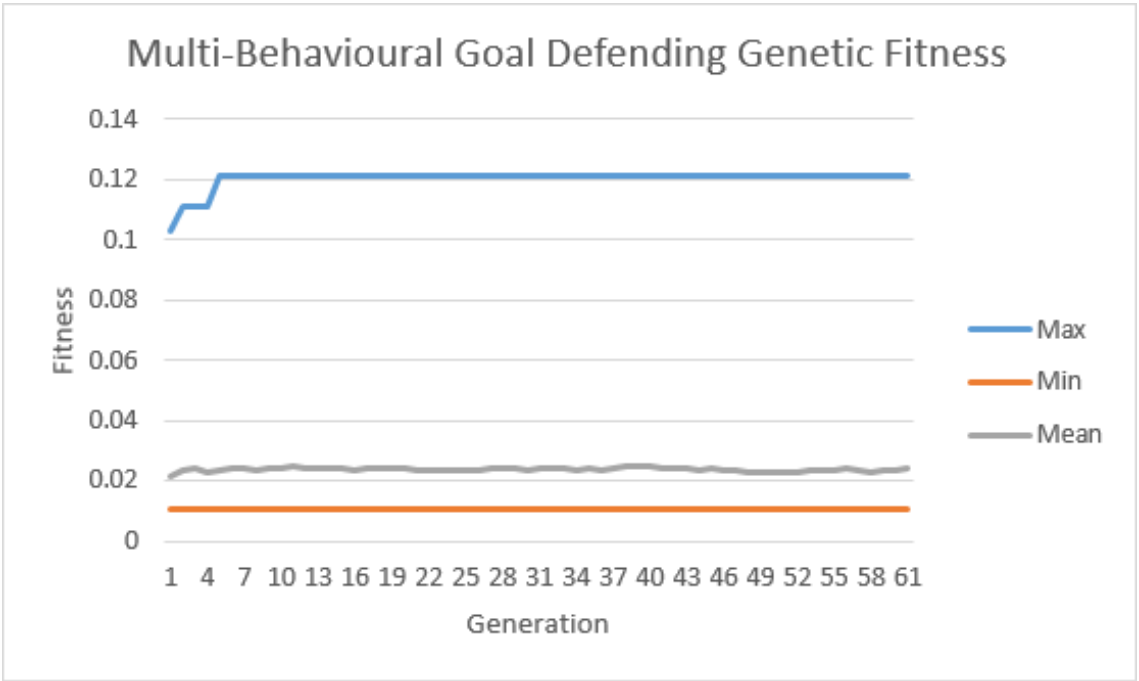


Figure 7.21: Multi-Behavioural Goal Defending Genetic Optimisation Fitness

Fitness data for this controller is shown in Figure 7.21, this is calculated by evaluating each candidate solution and computing the fitness from Equation 7.4.

The best individual had a fitness of 0.12105 and was improved using Hill climbing to 0.12163. As is with the previous Goal Keeper and Target Rotation training, this controller has a considerable difference between the Maximum values and the Mean. All three of these controllers have a large solution space in comparison to the Target Pursuit controller, therefore it is likely that the high initial result was achieved by chance and that a larger population will be beneficial for training these controllers.

As previously stated this controller optimises the internal data for four judgement nodes and the connections between the nodes themselves. The comparison value for the judgement nodes have a very large solution space, where each value could potentially have a thousand or more potential values for the individual parameter. Since the overall solution space is excessively large, there is not enough variation in the population. A higher mutation rate would have added more variation into the population and would have likely increased the resulting gains from the genetic optimisation process. As is with the Target Pursuit controller, a small increase in fitness constitutes a large difference in controller effectiveness.

Over the course of the genetic component of the optimisation process, the goal defending success rate was increased by 3.5%. The controller performance initially began with a high success rate and the final goal defending success rate at the end of this process is more than 90%. This increase is significant and would greatly effect the outcome of the overall goal defence in a competition scenario.

The save file for this controller is shown in Appendix D.1 and various stages of execution for this controller is shown in Appendix D.2.

Chapter 8

Conclusions

This thesis introduces an improved algorithm, Cascaded Genetic Network Programming (CGNP) that contains significant changes over the original architecture. This algorithm provides the ability to develop complex networks of nodes, over multiple levels of CGNP. This facilitates the simplification of optimisation based problems into small tasks that can be individually trained and reused. These multiple levels or layers can be scaled across a spectrum of complexity. In terms of robot soccer a low level network may control wheel movement and top level networks could dynamically manage player roles. After multiple level CGNP development, low level networks can be replaced with new networks that exceed the previous ability at the specific task, reducing the need to optimise a complete controller for every modification. Mechanisms such as time management and repeated executions of nodes are incorporated into this architecture, to prevent the top level network from losing control of the actions in lower layers. These mechanisms prevent infinite loops in lower layers, by giving the parent layer the ability to halt execution of a lower level after a set number of repeated executions.

Using a complete object orientated design approach any Artificial Intelligence algorithm can be incorporated into a CGNP network as either a Judgement or Processing node, these new node types are developed as general purpose nodes to allow re-usability in all situations where they are required. Every node within a CGNP network has the ability to add its respective data parameters into the chromosome for optimisation purposes, all nodes are therefore completely optimised for the required task while retaining re-usability in other problems.

A multiple layer CGNP network was created for a robot soccer goal keeper, by subdividing the problem into tasks for Target Pursuit, Target Rotation and basic Goal Defending. To facilitate the training of each of these tasks, a simulation environment was developed and tailored to the FIRA MiroSot Middle League robot soccer platform, while using Box2D to provide accurate physics calculations. Test situations within this simulation platform and objective functions were developed

for each task to individually evaluate the performance of trained CGNP networks. It was proven that a CGNP can further improve a pre-existing Artificial Intelligence system, by adding primitive nodes to a network along with a pre-trained Genetic Fuzzy Target Pursuit system, the optimisation process identified the conditions and values within these primitive nodes and increased the overall fitness of the system. After optimising each of these tasks, the resulting CGNP networks were imported into a new layer as independent nodes and a multi-behavioural goal keeper is optimised. This multi-behavioural controller allows the robot to align itself within the goal area and successfully defend the goal from 91% of all goal shooting attempts.

The Cascaded Genetic Network Programming architecture developed within this research has the ability to represent complicated behaviours in multiple layered networks which are modular and human interpretable. This architecture can be used to solve situations where multiple Artificial Intelligent systems are required to achieve a task and could be used in a wide variety of fields beyond the scope of robot soccer.

8.1 Summary of Achievements

This research contains a series of achievements and unique contributions, some of which are beyond the initial scope of this research. These achievements are listed below:

1. Cascaded Genetic Network Programming (CGNP) algorithm was introduced (introduced in Section 6.2), which is based upon the original Genetic Network Programming algorithm (described in Section 6.1).

2. CGNP allows complex controller design over multiple levels of networks (Section 6.2.1), much like a function in traditional programming.

3. Using an Object Orientated design any Artificial Intelligence algorithm can be incorporated into a CGNP network as a node (Section 6.2.1).

4. This algorithm can use generic internal nodes, which are reusable by all applications.

5. CGNP can optimise any data within Judgement or Processing nodes, instantly considering all comparators, operators, environmental data and their permutations (Section 6.2.2). Previous GNP architectures limit the allowable node variations stored in their fixed library of nodes.

6. The network allows injection of a priori knowledge about the problem, reducing the potential solution space.

7. A Hill Climbing algorithm is included at the end of the Genetic optimisation process (Section 6.2.3), to guarantee a local maxima solution from within the solution space.

8. A Robot Soccer simulation platform is developed using Box2D to produce realistic physics (Chapter 5), involving accurate collision detection, friction calculations and the complex shape of the robot and soccer field

9. Successfully applied the new CGNP architecture to a complex multi-behavioural control problem for a robot soccer goal keeper by using problem subdivision (Section 7.4).

10. Using this simulation platform, controllers for Target Pursuit (Section 7.1), Target Rotation (Section 7.2), basic Goal Keeping (Section 7.3) and the multi-behavioural goal keeper (Section 7.4) are optimised.

11. The CGNP architecture is developed to be multi-purpose and applicable to solving other multiple objective problems.

12. In the experiments Genetic Fuzzy processing nodes, If statement judgements and simple nodes to set the wheel speeds are are used, but this could expand to include any Artificial Intelligence or problem specific nodes (Section 6.2.1).

The Cascaded Genetic Network Programming algorithm allows a variation of optimisation settings, allowing a developer to define which individual parameters should be optimised. Some of the combinations that can be used are defined below:

1. Allowing only connections to change during training.

2. Allowing only internal node parameters to change in training.

3. Allowing both connections and internal parameters to change during training.

4. Allowing or disallowing some connections or some internal parameters to change during training, each node can be individually allowed or disallowed for both connections or internal parameters.

5. Theoretically multiple layers of a Cascaded GNP can be evolved at the same time, however this has not been tested.

8.2 Future Research

8.2.1 Cross Node Data Transfer

One aspect that is lacking in this architecture is the ability for nodes to transfer data between one another. This could be achieved by adding a new type of connection where nodes could "request" data and others could "provide" it, this would mean that a node could then obtain data from the previously evaluated node. If the previous node doesn't provide data then the input would likely default to zero, to overcome this issue environmental data and global variables could also be used as a "Data Node".

These Data Nodes would simply provide data to the network, they would only be evaluated when the current node requires information from them. Environmental data would be accessible by any node on the same CGNP layer and would usually involve interfacing with the environment to obtain the information required. Global variables would be accessible by every node within the same CGNP layer and any child layers, these global variables could be optimised within the genetic process and would allow the network to solve situations that should be solved using symmetry. Local variables that take on similar access rights to environmental data would also be a possibility, these could be optimised via the genetic process similar to the global variables.

This addition would bring CGNP closer to traditional programming methods and could solve more complicated situations, this addition will reduce the solution space for situations that can utilize symmetry, however, it will also increase this solution space if an excessive number of global variables are used. An excessive number of global variables will add to the solution space in two methods, the first being the data being optimised within each Data Node and the second is the "Data Connections" where all nodes that require data will have an additional potential input for each Data Node within the network.

With proper network planning the solution space could be significantly reduced using this feature, resulting in quick training times and more reliable training results.

8.2.2 Further Simulation Parallelism

Simulations are the most time consuming aspect of the CGNP optimisation in this research, these simulations are executed for every candidate solution during the optimisation process. Each individual in a generation can be evaluated independently from one another, thus there is the potential for massive parallelism. This parallelism could be done via a processing cluster or even on a graphics card (GPU), in the case of GPU parallelism the population can be split into processing blocks of

variable sizes depending on the GPU used and memory requirements, which will make large populations feasible for complicated problems.

The down side to the GPU option is that the entire simulation platform will need to be redesigned for the new architecture, this would take a considerable amount of time but would be worth the effort for a problem that requires high accuracy or requires regular optimisation. Once the simulation platform is implemented for GPU optimisation, incorporating it into the application developed in this research would be trivial in comparison and ultimately reduce a problem that took hours on the CPU to a matter of minutes. This reduced optimisation time would increase developer productivity as any bugs could be quickly identified and corrected.

8.2.3 Additional Artificial Intelligence Nodes

New node types should be incorporated into the CGNP architecture created alongside this research, specifically Reinforcement Learning and Neural Networks would be a useful addition. Both Neural Networks and Reinforcement Learning could apply to Judgement nodes, where n number of connections could span from the nodes to determine the next action based on environmental variables. The genetic component of CGNP could be used to set the initial values for these algorithms.

Neural Networks could also be used for a Processing node to train the ideal output speeds for the wheels based on the environmental variables, this would be especially useful for fine tuning the movements among obstacles.

8.2.4 Parallel CGNP Node Execution

The current CGNP architecture is confined to problems that can be solved sequentially, so setting motor speeds are done one at a time. This is a problem when developing AI for complicated robotics or other applications that are inherently parallel for example, a robotic hand. Any robotic hand would require motor movements for every knuckle to clench a fist, the current architecture would turn on each of these motors one at a time and then turn them off once they are no longer needed.

To compensate for these issues each motor could have its own CGNP system which is executed independently of one another, but that would require a significant amount of duplicated code as each CGNP system would have to independently decide when they should be turned on based on the environmental data.

A better solution would be to have a single CGNP system that controls the entire hand, which initiates nodes in parallel when required. The top level CGNP system would decide that it needs to clench the hand into a fist, then a theoretical "Parallel Node" would execute all the lower level CGNP systems required to complete

the clenching behaviour, control would return to the parallel node and would then continue on to the next node like normal.

This concept could be expanded to include image related problems that use the GPU or tasks that require a master slave cluster configuration.

8.2.5 Goal Shooter Role

Having successfully designed a Goal Keeper (GK) for the Robot Soccer platform, the next logical step would be the Goal Shooting (GS) behaviour. The GK controller could be used as an 'opposition' to any GS controllers therefore each controller could be further trained against one another. Although there is a possibility that these controllers would only optimise for each others behaviour meaning they could struggle against other systems.

8.2.6 Integrate the Simulation with the Real Platform

Reusing the simulation environment developed in conjunction with this research and integrating it with the Massey University robot soccer platform would likely improve the system stability by approximating object positions when environmental data is not available.

This inclusion could allow new research into opposition prediction systems, while giving the Massey University soccer team a competitive edge over opposing teams.

8.2.7 Implement Adaptive Genetic Algorithms

The traditional Genetic Algorithm used in this research involves presetting the probability of crossover and probability of mutation, this often results in a population that converges on a local maxima. Using an adaptive genetic algorithm[30] could avoid this problem by maintaining a diverse population using the two evolution probabilities. This addition would likely make the the CGNP architecture react more dynamically to different situations and potentially reduce the overall number of required generations evaluated, significantly improving the optimisation times.

8.2.8 Solution Space Look-up Table

Within this research the simulations used to evaluate candidate solutions required a significant amount of the training time. For situations where complicated and time consuming simulations, it may be worth implementing a solution space look-up table which could be either a database or a simple list of previously evaluated solutions. This would remove the need to re-evaluate any candidate solutions that

have already been examined, potentially reducing the overall genetic optimisation times.

This would only be feasible for simulations that are very time consuming, as this inclusion would add a few microseconds before every candidate solution evaluation to test whether it has already been evaluated. With small simulations this lookup process could eventually exceed the duration of the actual simulation, meaning this inclusion would be detrimental to the overall optimisation times.

8.2.9 Test Multiple Layer Optimisation

The Cascading Genetic Network Programming architecture introduces the ability to have multiple levels of intelligent systems, these multiple layers have the ability to add their respective data to the gene structure for optimisation.

Tests should be conducted to evaluate the performance of multiple layer optimisation within a CGNP, using one overall objective function and the potential for each sub-layer to have its own individual objective definition should be explored.

Bibliography

[1] J. Vernon. Fuzzy logic systems. Technical report, Control Systems Principles.

[2] G. Kendall. G5baim - artificial intelligence methods. `http://www.cs.nott.ac.uk/~gxk/courses/g5baim/004ga/GA06-reproductionmod.html`. Accessed: 03-01-2016.

[3] M.A. Lee and H. Takagi. Integrating design stage of fuzzy systems using genetic algorithms. In *Fuzzy Systems, 1993., Second IEEE International Conference on*, pages 612–617 vol.1, 1993.

[4] C.L. Karr and E.J. Gentry. Fuzzy control of ph using genetic algorithms. *Fuzzy Systems, IEEE Transactions on*, 1(1):46–, Feb 1993.

[5] H. Katagiri, K. Hirasawa, and J. Hu. Genetic network programming - application to intelligent agents. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 5, pages 3829–3834 vol.5, 2000.

[6] T. Murata, T. Nakamura, and S. Nagamine. Performance of genetic network programming for learning agents on perceptual aliasing problem. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 3, pages 2317–2322 Vol. 3, Oct 2005.

[7] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu, J. Hu, and S. Markon. A double-deck elevator group supervisory control system using genetic network programming. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(4):535–550, July 2008.

[8] J. Riley. Evolving fuzzy rules for goal-scoring behaviour in a robot soccer environment. In V. Papi, editor, *Robot Soccer*, chapter 7, pages 139–174. InTech, 2010.

[9] T. Murata and D. Okada. Using genetic network programming to get comprehensible control rules for real robots. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1983–1988, 2006.

[10] W. Wang. Genetic network programming with fuzzy reinforcement learning nodes for multi-behaviour robot control. Master's thesis, Massey University, 2014.

[11] S. Mukherjee, J. Yearwood, P. Vamplew, and S. Huda. Reinforcement learning approach to aibo robot's decision making process in robosoccer's goal keeper problem. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2011 12th ACIS International Conference on*, pages 24–30, July 2011.

[12] Fira mirosot game rules. `http://www.fira.net/contents/data/MiroSot_Rules.doc`. Accessed: 19-02-2016.

[13] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 6 1965.

[14] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(1):116–132, 1985.

[15] Jr. Davis, L.I., T.F. Sieja, R.W. Matteson, G.A. Dage, and R. Ames. Fuzzy logic for vehicle climate control. In *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*, pages 530–534 vol.1, Jun 1994.

[16] M.S. Lee, J.Y. Kim and J.J. Lee. Multi-objective evolutionary design of fuzzy logic controllers for car parking problem. In *Advanced Motion Control, 2006. 9th IEEE International Workshop on*, pages 607–612, 2006.

[17] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[18] J. Buurman, S. Zhang, and V. Babovic. Reducing risk through real options in systems design: The case of architecting a maritime domain protection system. *Risk Analysis: An International Journal*, 29(3):366 – 379, 2009.

[19] A. George, B. R. Rajakumar, and D. Binu. Genetic algorithm based airlines booking terminal open/close decision system. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 174–179. ACM, 2012.

[20] C.Z. Janikow and Z. Michalewicz. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In R. K. Belew and

L. B. Booker, editors, *Proc. of the 4th International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann, 1991.

[21] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, (3):493–530, 1989.

[22] J. Rada-Vilela. fuzzylite: a fuzzy logic control library. `http://www.fuzzylite.com`, 2014.

[23] M. Wall. Galib documentation. `http://lancet.mit.edu/galib-2.4/`. Accessed: 05-06-2015.

[24] E. Catto. Box2d. `http://www.box2d.org/`. Accessed: 10-06-2015.

[25] E. Catto. Box2d forums: Iterations? `http://www.box2d.org/forum/viewtopic.php?f=8&t=4396#p21191`. Accessed: 15-06-2015.

[26] iforce2d. Box2d c++ tutorials - top-down car physics. `http://www.iforce2d.net/b2dtut/top-down-car`. Accessed: 24-06-2015.

[27] S. Mabu, K. Hirasawa, and J. Hu. A graph-based evolutionary algorithm: Genetic network programming (gnp) and its extension using reinforcement learning. *Evolutionary Computation*, 15(3):369–398, Sept 2007.

[28] L. Yu, J. Zhou, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. Elevator group control system using genetic network programming with aco considering transitions. In *SICE, 2007 Annual Conference*, pages 1330–1336, Sept 2007.

[29] L. Yu, J. Zhou, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. Double-deck elevator group supervisory control system using genetic network programming with ant colony optimization. In *2007 IEEE Congress on Evolutionary Computation*, pages 1015–1022, Sept 2007.

[30] M. Srinivas and L.M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667, Apr 1994.

Appendices

Appendix A

Target Pursuit Control System

A.1 Network Save File

This section describes the best Target Pursuit individual, it is XML encoded for readability and re-usability.

Each node has its own unique data defined as properties in each element within the document. Properties that occur among most nodes are described below:

- xLocModifier - X Coordinate for the node location in the visual representation of the CGNP network
- yLocModifier - Y Coordinate for the node location in the visual representation of the CGNP network
- executionTime - The number of Repeated Executions for the node, described in Section 6.2
- connectionNumber - The number of outgoing connections for the node
- optimizeCon - Boolean value to define whether the connections should be optimised
- con[X] - All con values are an identification number to the next node to be executed, [X] represents the connection number and the value associated to the property is the node ID.

If statement judgement nodes contain unique properties, these are operate, compare and selectedDataType. The operate property is the operator for the comparison, either less than (value of zero) or greater than (value of one). The compare value is a comparison value, it contains a real value this is one of the main components in the if statement. Lastly is the selectedDataType, this is a text input of an actual environmental variable used within Robot Soccer. These components

work together to form the statement IF(selectedDataType operate compare), if this result is true then the next node is con0, otherwise it is con1.

The set wheel nodes also contain unique data, these are described below:

- speedL and speedR - These are values between -1 and 1, to define the actual speed percentage that the wheels should be set to when this node is visited.
- movementType - This allows the restriction of individual movement types. A value of zero is normal movement, both left and right wheels can have different values. One is no movement, meaning the robot can only rotate on the spot, the right wheel will be the negation of the left wheel speed. Lastly, two will be no rotation, this results in a robot that can only move forwards and backwards.
- optimizeLWheel and optimizeRWheel - These are boolean values to enable or disable the optimisation of the individual wheel speeds.
- flipIfNegativeTargetAngle - If this boolean value is one, then both wheel speeds will be negated if the TargetAngle environmental variable is negative.

Fuzzy Logic nodes do not contain additional properties, however the entire fuzzy system is included between the beginning and ending tags for the element. This code is sent directly to the fuzzylite decoder for interpretation, this is an example of how a node could store its own unique data and allow interfacing with a library without requiring a redefinition of the loading and saving functions.

The last node type is the GNP node itself, this contains two additional parameters. The first is start, this is an identification number for the start node in the network. Secondly there is root, which is a boolean value to state whether the current GNP node is the top level node or a child.

The Target Pursuit system is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<GNP_NodeProcessing_GNP xLocModifier="0" yLocModifier="0"
    executionTime="1" connectionNumber="1" optimizeCon="1"
    con0="-1" start="0" root="1">
    <GNP_Robot_IfStatement xLocModifier="-89"
        yLocModifier="82" executionTime="-1"
        connectionNumber="2" optimizeCon="0"
        con0="1" con1="2" operate="1" compare="81"
        selectedDataType="AbsTargetAngle"/>
    <GNP_Robot_SetWheels xLocModifier="-89"
        yLocModifier="-67" executionTime="4"
        connectionNumber="1" optimizeCon="0"
```

```
        con0="0" speedL="0.895" speedR="-0.895"
        movementType="1" optimizeLWheel="1"
        optimizeRWheel="1" flipIfNegativeTargetAngle="1"/>
    <GNP_Robot_FuzzyLogic xLocModifier="-1"
        yLocModifier="-104" executionTime="1"
        connectionNumber="1" optimizeCon="0"
        con0="0">
Engine: TargetPursuitDefault
InputVariable: AbsTargetAngle
  enabled: true
  range: 0.000 180.000
  term: straightAhead Trapezoid 0.000 0.000 18.000 27.000
  term: smallRight Trapezoid 15.000 25.000 41.000 61.000
  term: mediumRight Trapezoid 44.000 53.000 91.000 113.000
  term: largeRight Trapezoid 91.000 105.000 180.000 180.000
InputVariable: TargetDistance
  enabled: true
  range: 0.000 650.000
  term: smallDistance Trapezoid 0.000 0.000 100.000 133.000
  term: mediumDistance Trapezoid 93.000 184.000 292.000 402.000
  term: largeDistance Trapezoid 329.000 378.000 500.000 619.000
  term: massiveDistance Triangle 496.000 650.000 650.000
OutputVariable: LeftWheel
  enabled: true
  range: 0.000 1.000
  accumulation: none
  defuzzifier: WeightedSum Automatic
  default: 1.000
  lock-previous: false
  lock-range: true
  term: still Constant 0.000
  term: small Constant 0.300
  term: medium Constant 0.400
  term: large Constant 0.750
  term: full Constant 1.000
OutputVariable: RightWheel
  enabled: true
  range: 0.000 1.000
  accumulation: none
```

```
  defuzzifier: WeightedSum Automatic
  default: 1.000
  lock-previous: false
  lock-range: true
  term: still Constant 0.000
  term: small Constant 0.300
  term: medium Constant 0.400
  term: large Constant 0.750
  term: full Constant 1.000
RuleBlock:
  enabled: true
  conjunction: Minimum
  disjunction: none
  activation: none
  rule: if AbsTargetAngle is straightAhead and TargetDistance
          is smallDistance then LeftWheel is medium and
          RightWheel is medium
  rule: if AbsTargetAngle is straightAhead and TargetDistance
          is mediumDistance then LeftWheel is large and
          RightWheel is large
  rule: if AbsTargetAngle is straightAhead and TargetDistance
          is largeDistance then LeftWheel is full and
          RightWheel is full
  rule: if AbsTargetAngle is straightAhead and TargetDistance
          is massiveDistance then LeftWheel is full and
          RightWheel is full
  rule: if AbsTargetAngle is smallRight and TargetDistance
          is massiveDistance then LeftWheel is full and
          RightWheel is large
  rule: if AbsTargetAngle is smallRight and TargetDistance
          is largeDistance then LeftWheel is full and
          RightWheel is medium
  rule: if AbsTargetAngle is smallRight and TargetDistance
          is mediumDistance then LeftWheel is large and
          RightWheel is medium
  rule: if AbsTargetAngle is smallRight and TargetDistance
          is smallDistance then LeftWheel is medium and
          RightWheel is small
  rule: if AbsTargetAngle is mediumRight and TargetDistance
```

```
          is massiveDistance then LeftWheel is full and
          RightWheel is medium
  rule: if AbsTargetAngle is mediumRight and TargetDistance
          is largeDistance then LeftWheel is large and
          RightWheel is medium
  rule: if AbsTargetAngle is mediumRight and TargetDistance
          is mediumDistance then LeftWheel is large and
          RightWheel is medium
  rule: if AbsTargetAngle is mediumRight and TargetDistance
          is smallDistance then LeftWheel is medium and
          RightWheel is small
  rule: if AbsTargetAngle is largeRight and TargetDistance
          is massiveDistance then LeftWheel is medium and
          RightWheel is small
  rule: if AbsTargetAngle is largeRight and TargetDistance
          is largeDistance then LeftWheel is large and
          RightWheel is still
  rule: if AbsTargetAngle is largeRight and TargetDistance
          is mediumDistance then LeftWheel is large and
          RightWheel is still
  rule: if AbsTargetAngle is largeRight and TargetDistance
          is smallDistance then LeftWheel is full and
          RightWheel is still
    </GNP_Robot_FuzzyLogic>
</GNP_NodeProcessing_GNP>
```

A.2 Figures of Network Execution

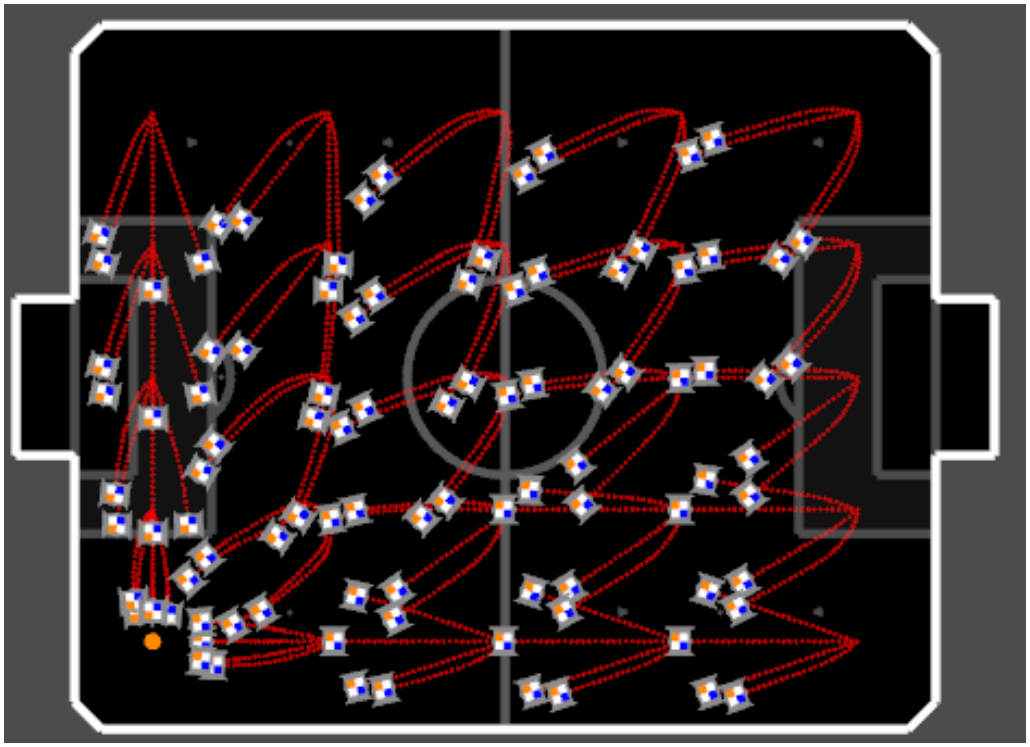


Figure A.1: Target Pursuit Execution Image: 1

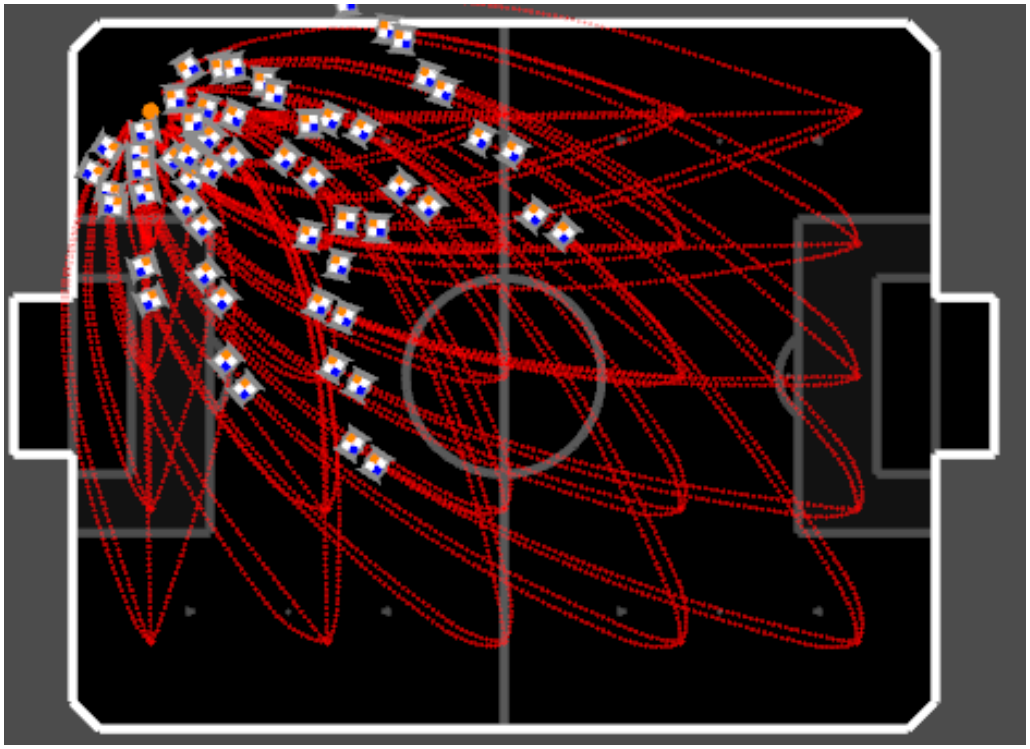


Figure A.2: Target Pursuit Execution Image: 2

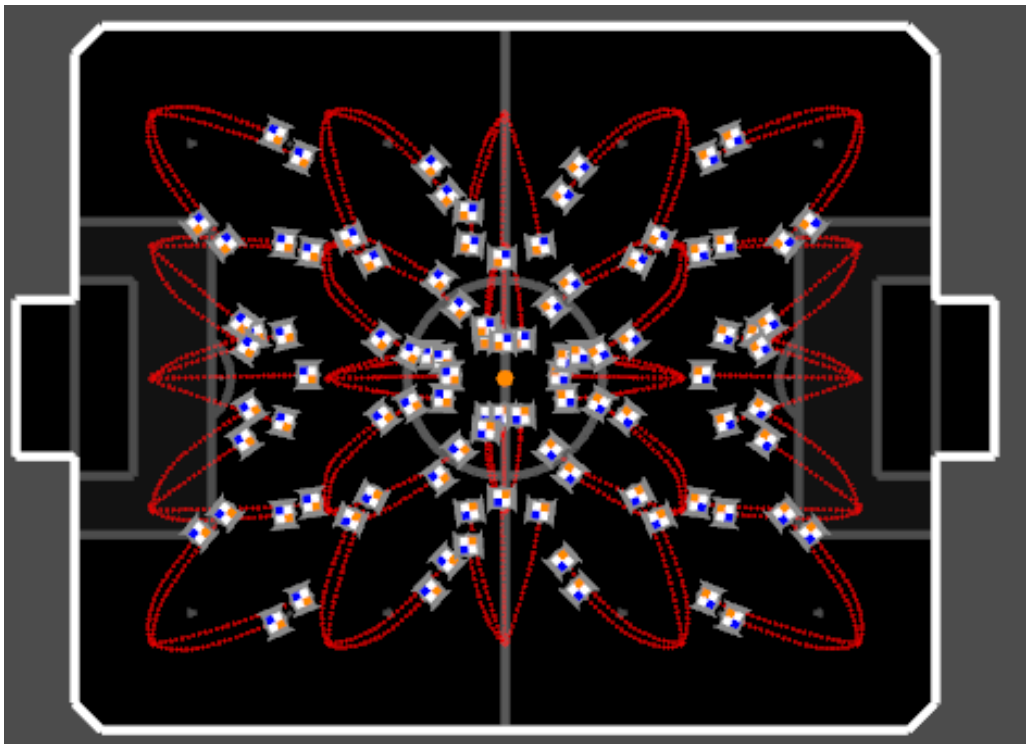


Figure A.3: Target Pursuit Execution Image: 3

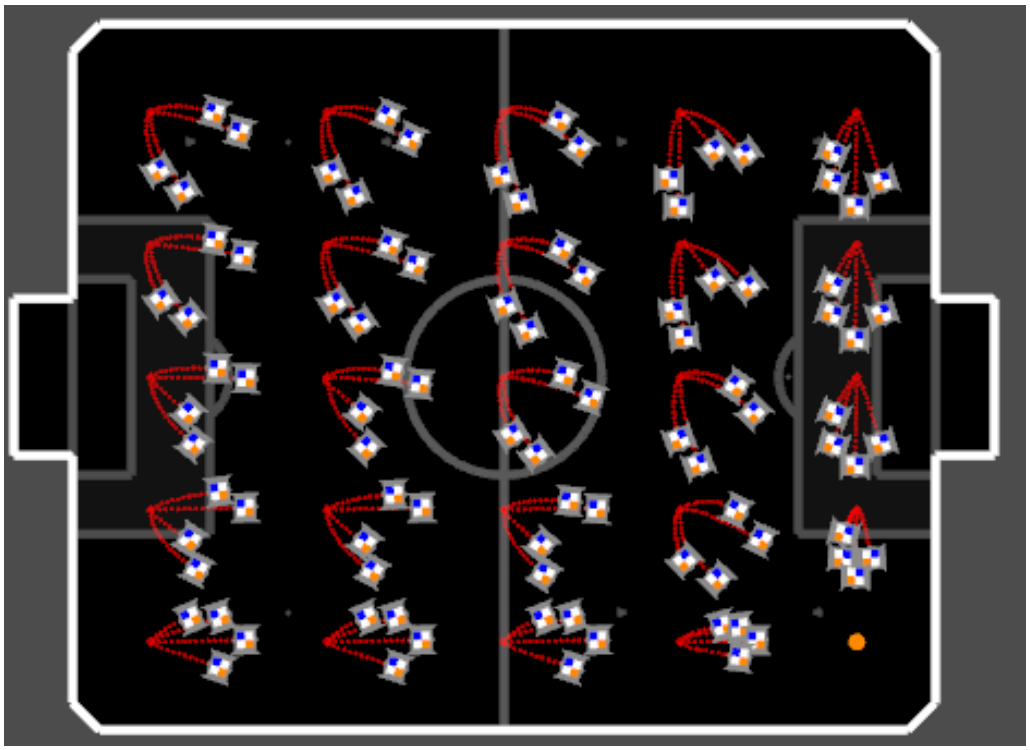


Figure A.4: Target Pursuit Execution Image: 4

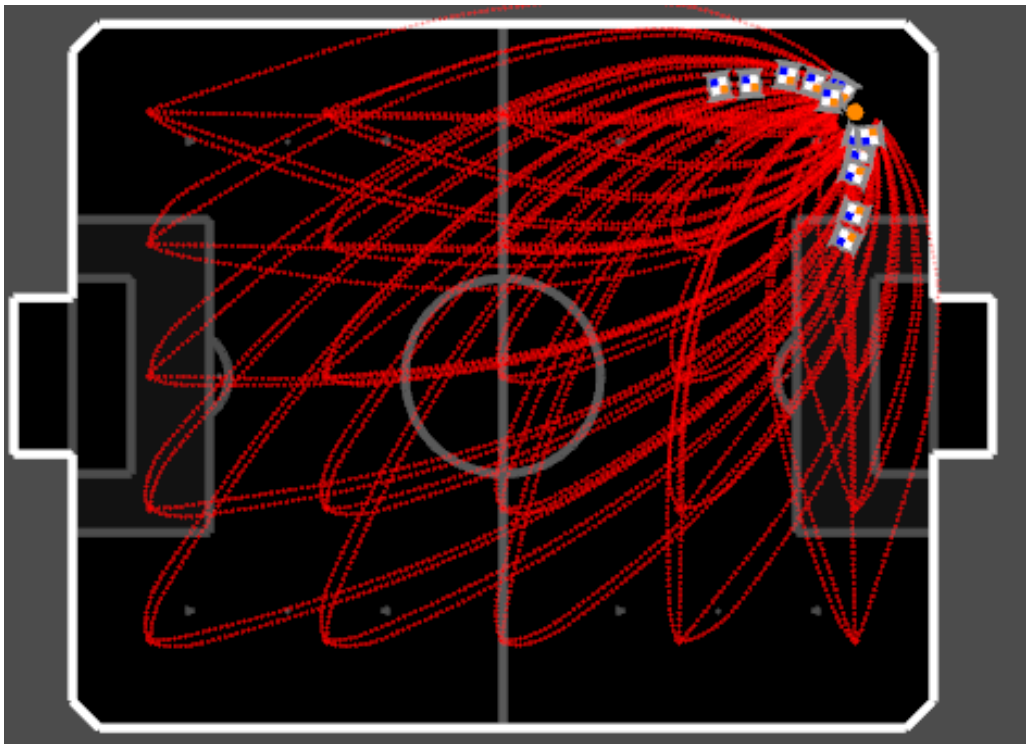


Figure A.5: Target Pursuit Execution Image: 5

Appendix B

Target Rotation Control System

B.1 Network Save File

A description of all parameters within the save file is located in the Target Pursuit section, Appendix A.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<GNP_NodeProcessing_GNP xLocModifier="-16" yLocModifier="110"
    executionTime="1" connectionNumber="1" optimizeCon="1"
    con0="4" start="4" root="1" optimizeGNP="0">
    <GNP_Robot_SetWheels xLocModifier="-35" yLocModifier="-124"
        executionTime="3" connectionNumber="1" optimizeCon="1"
        con0="4" speedL="0.69" speedR="-0.69" movementType="1"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="0"/>
    <GNP_Robot_SetWheels xLocModifier="138" yLocModifier="41"
        executionTime="3" connectionNumber="1" optimizeCon="1"
        con0="12" speedL="0.105" speedR="-0.105" movementType="1"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="0"/>
    <GNP_Robot_SetWheels xLocModifier="-155" yLocModifier="78"
        executionTime="3" connectionNumber="1" optimizeCon="1"
        con0="12" speedL="0.005" speedR="-0.005" movementType="1"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="0"/>
    <GNP_Robot_SetWheels xLocModifier="77" yLocModifier="136"
        executionTime="3" connectionNumber="1" optimizeCon="1"
        con0="5" speedL="0.02" speedR="-0.02" movementType="1"
        optimizeLWheel="1" optimizeRWheel="1"
```

```
    flipIfNegativeTargetAngle="0"/>
<GNP_Robot_IfStatement xLocModifier="88" yLocModifier="-121"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="10" con1="6" operate="0" compare="90"
    selectedDataType="AbsTargetAngle"/>
<GNP_Robot_IfStatement xLocModifier="-39" yLocModifier="-55"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="6" con1="7" operate="1" compare="27"
    selectedDataType="AbsTargetAngle"/>
<GNP_Robot_IfStatement xLocModifier="-134" yLocModifier="-131"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="8" con1="0" operate="0" compare="0"
    selectedDataType="TargetAngle"/>
<GNP_Robot_IfStatement xLocModifier="-131" yLocModifier="-95"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="9" con1="2" operate="0" compare="0"
    selectedDataType="TargetAngle"/>
<GNP_Robot_SetWheels xLocModifier="-139" yLocModifier="-98"
    executionTime="3" connectionNumber="1" optimizeCon="1"
    con0="4" speedL="-0.69" speedR="0.69" movementType="1"
    optimizeLWheel="1" optimizeRWheel="1"
    flipIfNegativeTargetAngle="0"/>
<GNP_Robot_SetWheels xLocModifier="-114" yLocModifier="-4"
    executionTime="3" connectionNumber="1" optimizeCon="1"
    con0="12" speedL="-0.005" speedR="0.005" movementType="1"
    optimizeLWheel="1" optimizeRWheel="1"
    flipIfNegativeTargetAngle="0"/>
<GNP_Robot_IfStatement xLocModifier="106" yLocModifier="-41"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="11" con1="1" operate="0" compare="0"
    selectedDataType="TargetAngle"/>
<GNP_Robot_SetWheels xLocModifier="69" yLocModifier="14"
    executionTime="3" connectionNumber="1" optimizeCon="1"
    con0="12" speedL="-0.105" speedR="0.105" movementType="1"
    optimizeLWheel="1" optimizeRWheel="1"
    flipIfNegativeTargetAngle="0"/>
<GNP_Robot_IfStatement xLocModifier="-8" yLocModifier="78"
    executionTime="-1" connectionNumber="2" optimizeCon="1"
    con0="13" con1="3" operate="0" compare="0"
```

```
        selectedDataType="TargetAngle"/>
    <GNP_Robot_SetWheels xLocModifier="-60" yLocModifier="143"
        executionTime="3" connectionNumber="1" optimizeCon="1"
        con0="5" speedL="-0.02" speedR="0.02" movementType="1"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="0"/>
</GNP_NodeProcessing_GNP>
```

B.2 Figures of Network Execution

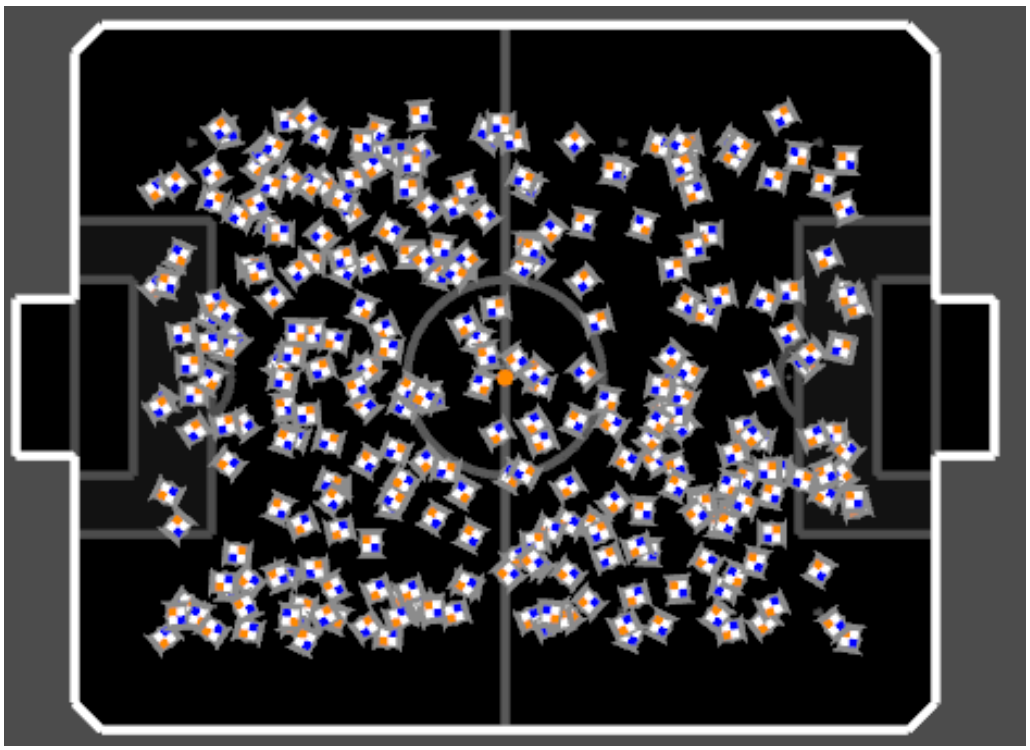


Figure B.1: Target Rotation Execution Image: 1

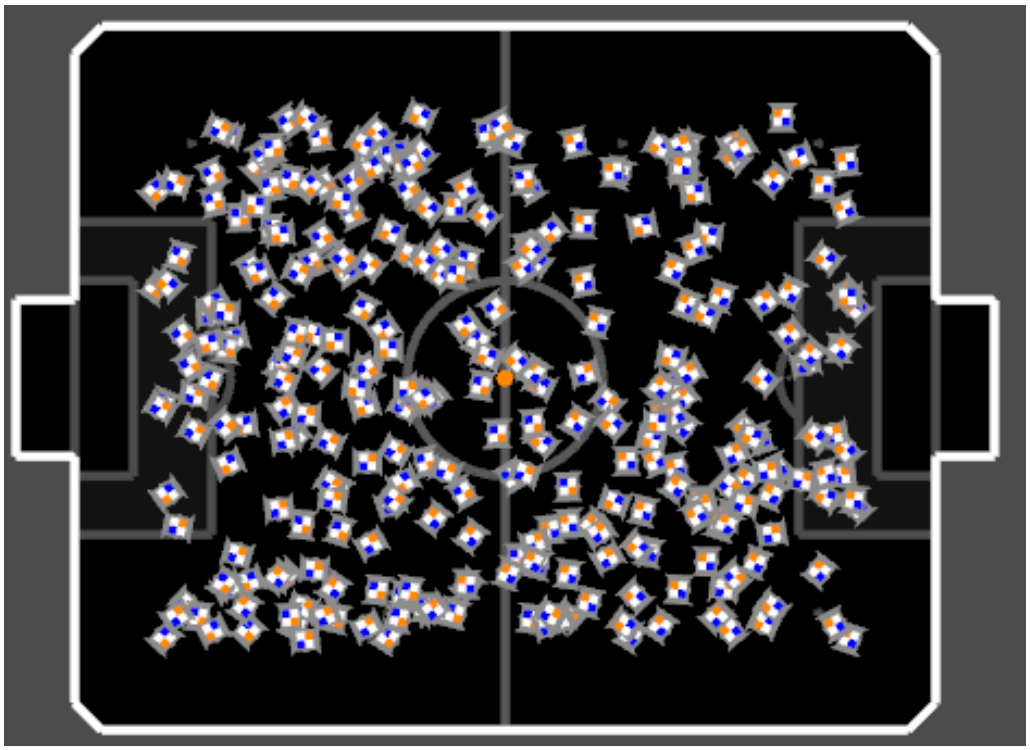


Figure B.2: Target Rotation Execution Image: 2

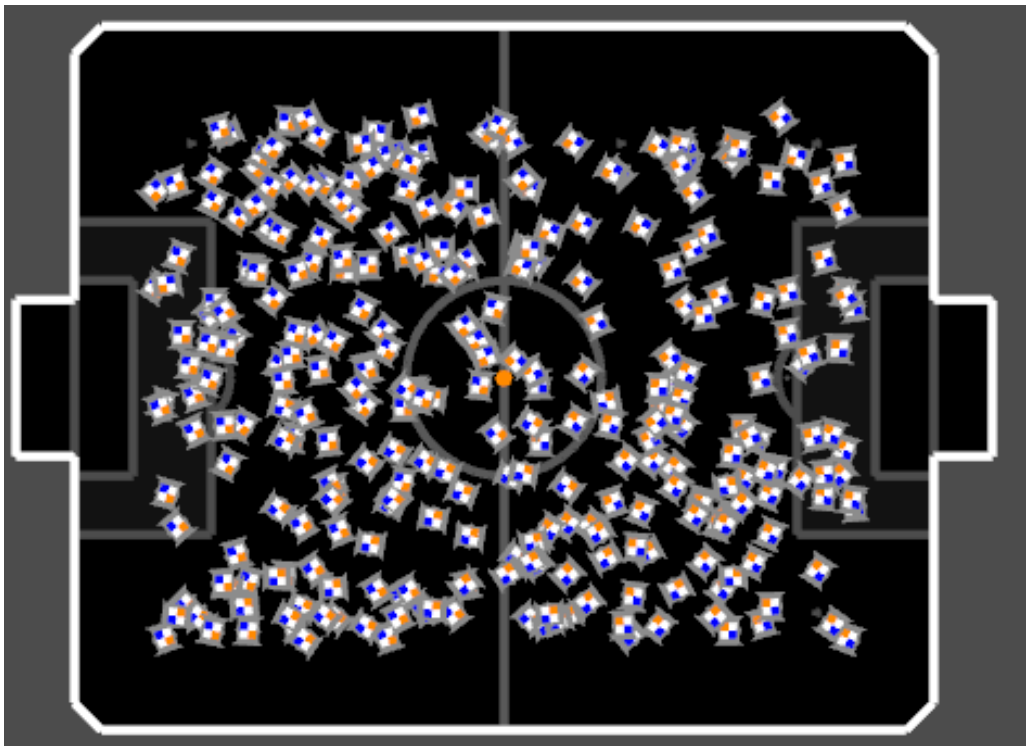


Figure B.3: Target Rotation Execution Image: 3

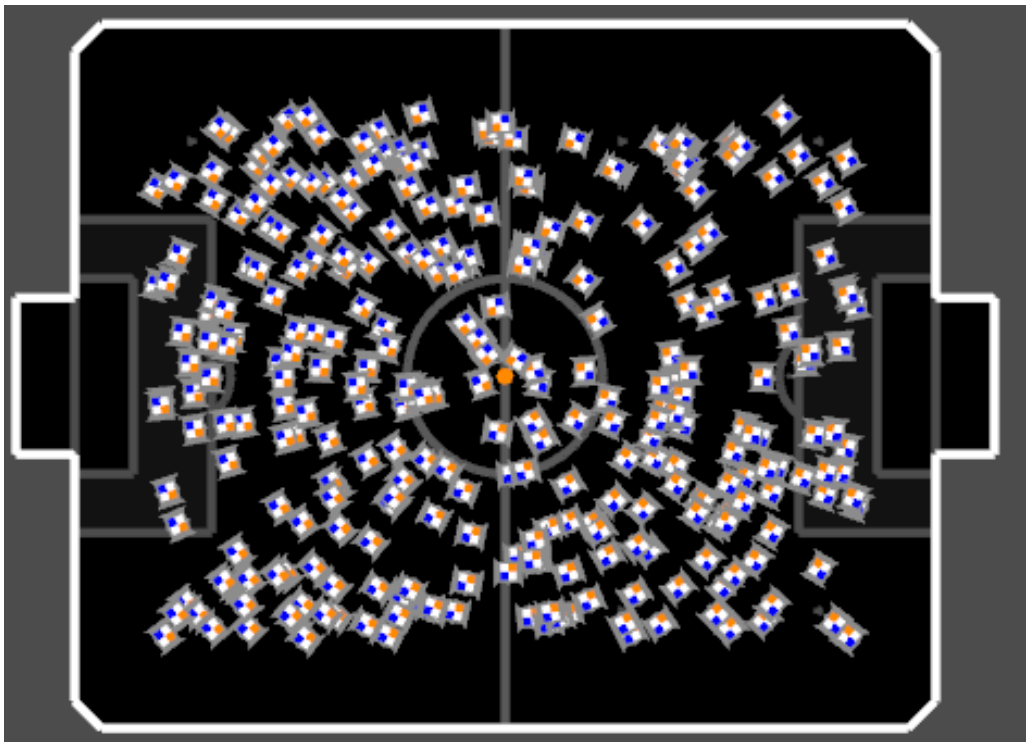


Figure B.4: Target Rotation Execution Image: 4

Appendix C

Goal Defending Control System

C.1 Network Save File

A description of all parameters within the save file is located in the Target Pursuit section, Appendix A.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<GNP_NodeProcessing_GNP xLocModifier="93" yLocModifier="105"
    executionTime="1" connectionNumber="1" optimizeCon="1"
    con0="3" start="0" root="1" optimizeGNP="0">
    <GNP_Robot_IfStatement xLocModifier="-50" yLocModifier="-115"
        executionTime="-1" connectionNumber="2" optimizeCon="0"
        con0="1" con1="2" operate="0" compare="5"
        selectedDataType="LeftGoalInterceptDistance"/>
    <GNP_Robot_SetWheels xLocModifier="-108" yLocModifier="17"
        executionTime="3" connectionNumber="1" optimizeCon="0"
        con0="0" speedL="0" speedR="0" movementType="2"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="1"/>
    <GNP_Robot_IfStatement xLocModifier="67" yLocModifier="-72"
        executionTime="-1" connectionNumber="2" optimizeCon="0"
        con0="5" con1="6" operate="1" compare="86.4"
        selectedDataType="RobotLeftGoalInterceptAbsAngle"/>
    <GNP_Robot_SetWheels xLocModifier="-60" yLocModifier="122"
        executionTime="3" connectionNumber="1" optimizeCon="0"
        con0="0" speedL="0.96" speedR="0.96" movementType="2"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="1"/>
    <GNP_Robot_SetWheels xLocModifier="88" yLocModifier="124"
```

```
        executionTime="3" connectionNumber="1" optimizeCon="0"
        con0="0" speedL="0.665" speedR="0.665" movementType="2"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="1"/>
    <GNP_Robot_IfStatement xLocModifier="9" yLocModifier="15"
        executionTime="-1" connectionNumber="2" optimizeCon="0"
        con0="3" con1="7" operate="1" compare="1620.51"
        selectedDataType="LeftGoalInterceptDistance"/>
    <GNP_Robot_IfStatement xLocModifier="142" yLocModifier="15"
        executionTime="-1" connectionNumber="2" optimizeCon="0"
        con0="4" con1="8" operate="0" compare="2331.26"
        selectedDataType="LeftGoalInterceptDistance"/>
    <GNP_Robot_SetWheels xLocModifier="16" yLocModifier="120"
        executionTime="3" connectionNumber="1" optimizeCon="0"
        con0="0" speedL="-0.98" speedR="-0.98" movementType="2"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="1"/>
    <GNP_Robot_SetWheels xLocModifier="156" yLocModifier="125"
        executionTime="3" connectionNumber="1" optimizeCon="0"
        con0="0" speedL="-0.735" speedR="-0.735" movementType="2"
        optimizeLWheel="1" optimizeRWheel="1"
        flipIfNegativeTargetAngle="1"/>
</GNP_NodeProcessing_GNP>
```

C.2 Figures of Network Execution

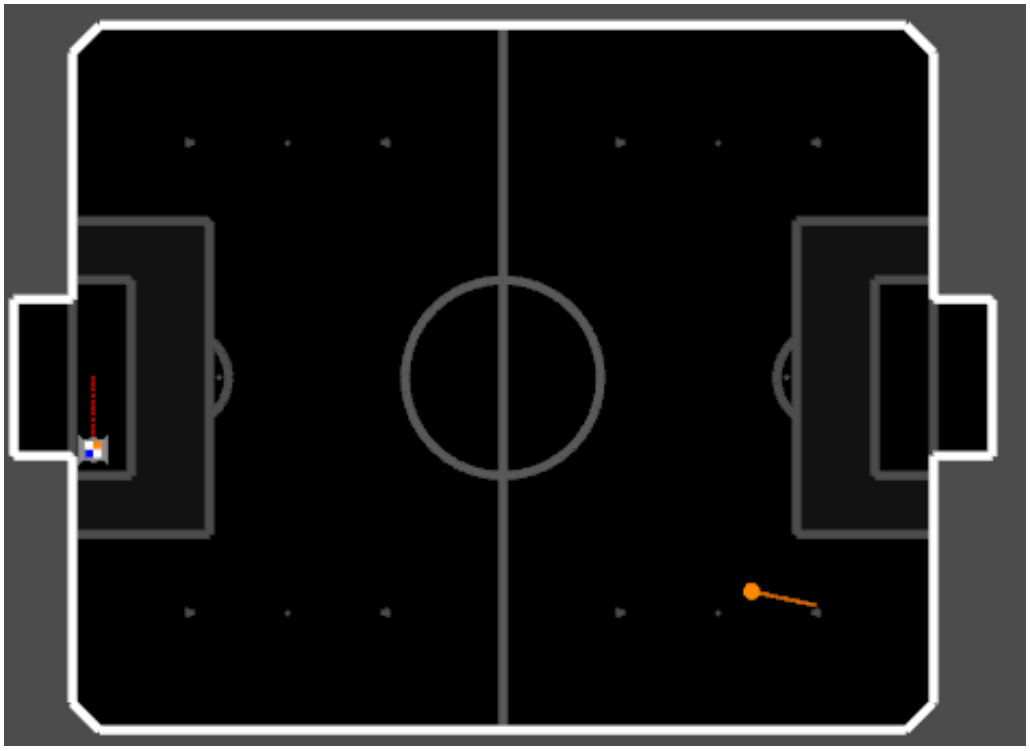


Figure C.1: Goal Defending Execution Image: 1

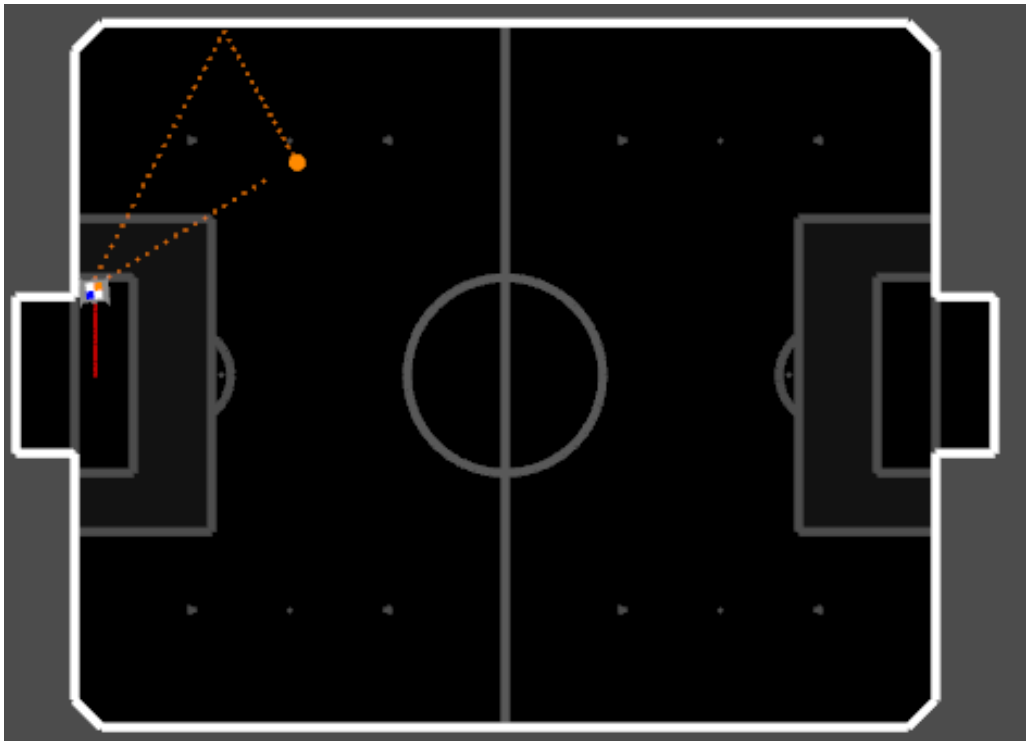


Figure C.2: Goal Defending Execution Image: 2

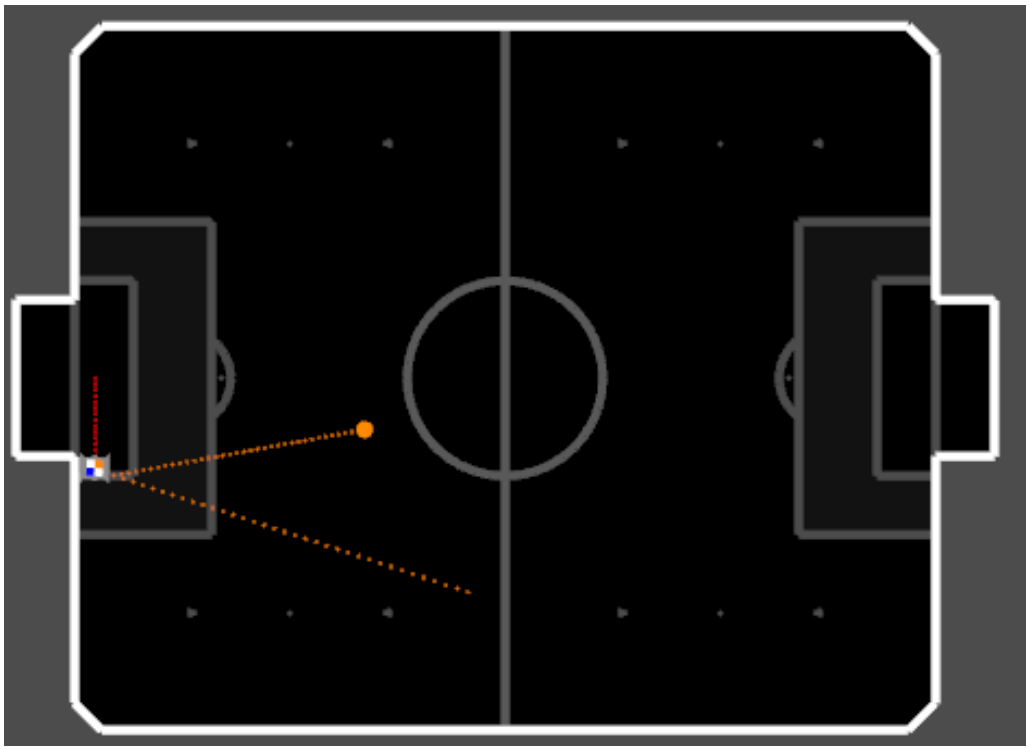


Figure C.3: Goal Defending Execution Image: 3

Appendix D

Multi-Behavioural Goal Keeper Control System

D.1 Network Save File

A description of all parameters within the save file is located in the Target Pursuit section, Appendix A.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<GNP_NodeProcessing_GNP xLocModifier="0" yLocModifier="0"
    executionTime="1" connectionNumber="1" optimizeCon="1"
    con0="-1" start="1" root="1" optimizeGNP="1">
    <GNP_NodeProcessing_GNP xLocModifier="-90" yLocModifier="109"
        executionTime="4" connectionNumber="1" optimizeCon="1"
        con0="1" start="0" root="1" optimizeGNP="0">
```

Target Pursuit Contents (Appendix A.1)

```
    </GNP_NodeProcessing_GNP>
    <GNP_Robot_IfStatement xLocModifier="-78" yLocModifier="-29"
        executionTime="-1" connectionNumber="2" optimizeCon="1"
        con0="2" con1="3" operate="1" compare="113.72"
        selectedDataType="RobotLeftGoalDistance"/>
    <GNP_Robot_IfStatement xLocModifier="10" yLocModifier="-97"
        executionTime="-1" connectionNumber="2" optimizeCon="1"
        con0="5" con1="5" operate="1" compare="144"
        selectedDataType="AbsRobotNorthAngle"/>
    <GNP_NodeProcessing_GNP xLocModifier="-17" yLocModifier="110"
        executionTime="1" connectionNumber="1" optimizeCon="1"
        con0="4" start="4" root="1" optimizeGNP="0">
```

Target Rotation Contents (Appendix B.1)

```
    </GNP_NodeProcessing_GNP>
    <GNP_NodeProcessing_GNP xLocModifier="93" yLocModifier="104"
        executionTime="4" connectionNumber="1" optimizeCon="1"
        con0="3" start="0" root="1" optimizeGNP="0">
```

Goal Defending Contents (Appendix C.1)

```
    </GNP_NodeProcessing_GNP>
    <GNP_Robot_IfStatement xLocModifier="92" yLocModifier="-83"
        executionTime="-1" connectionNumber="2" optimizeCon="1"
        con0="5" con1="0" operate="0" compare="178"
        selectedDataType="RobotLeftGoalInterceptAbsAngle"/>
    <GNP_Robot_IfStatement xLocModifier="148" yLocModifier="0"
        executionTime="-1" connectionNumber="2" optimizeCon="1"
        con0="5" con1="4" operate="1" compare="230"
        selectedDataType="LeftGoalInterceptDistance"/>
</GNP_NodeProcessing_GNP>
```

D.2 Figures of Network Execution

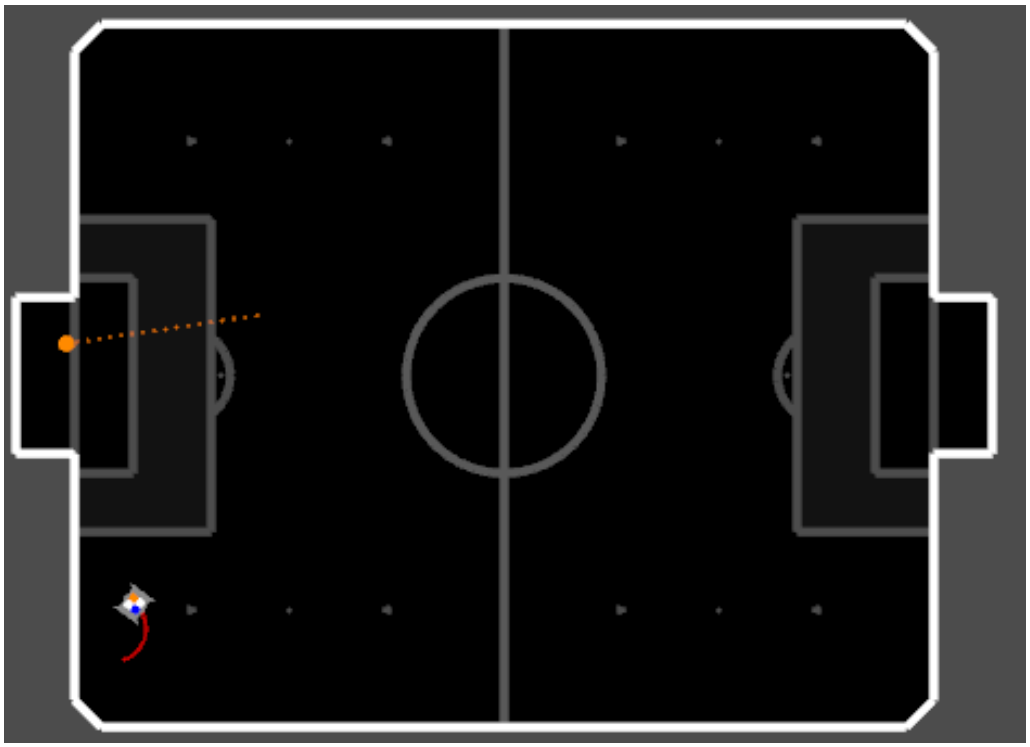


Figure D.1: Multi-Behavioural Goal Keeper Execution Image: 1

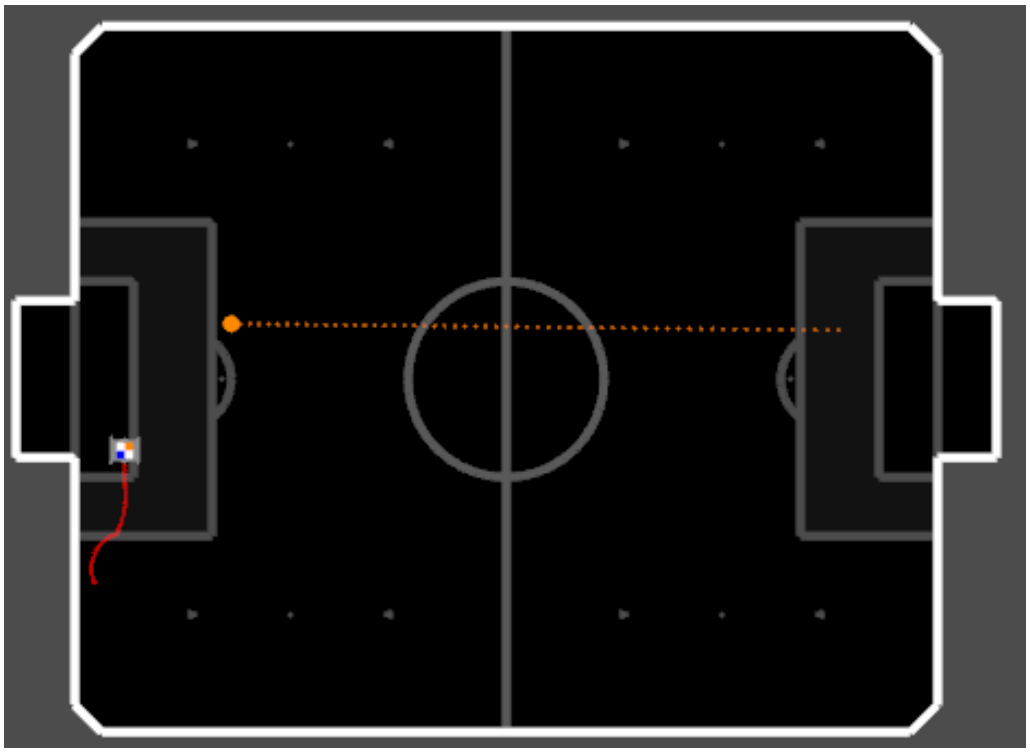


Figure D.2: Multi-Behavioural Goal Keeper Execution Image: 2

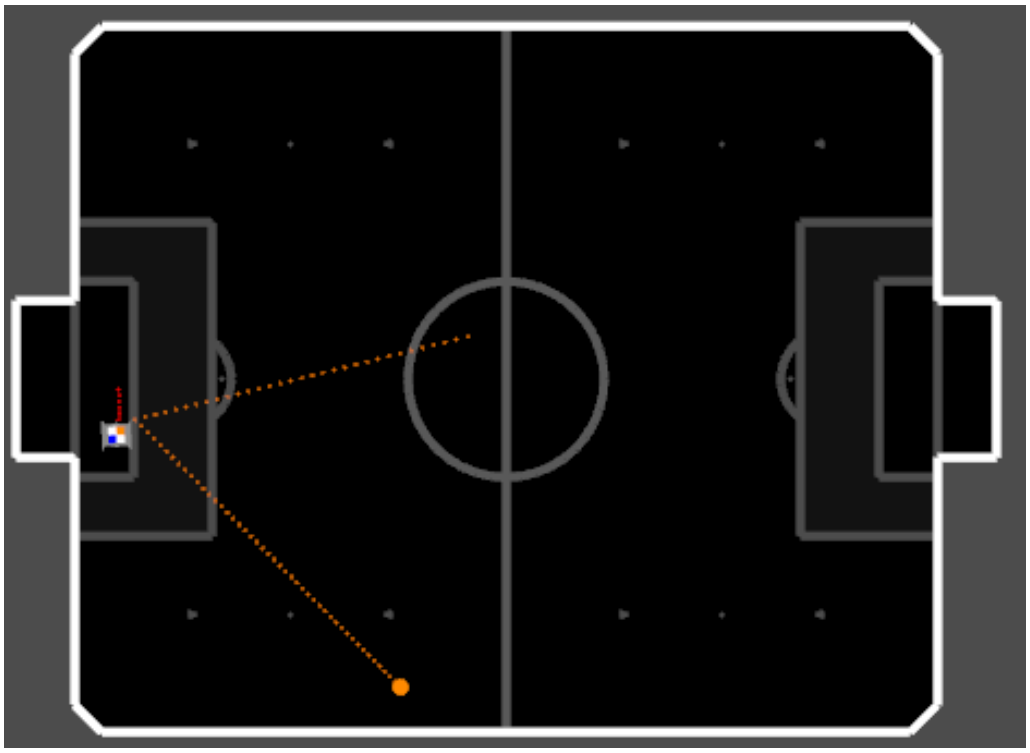


Figure D.3: Multi-Behavioural Goal Keeper Execution Image: 3

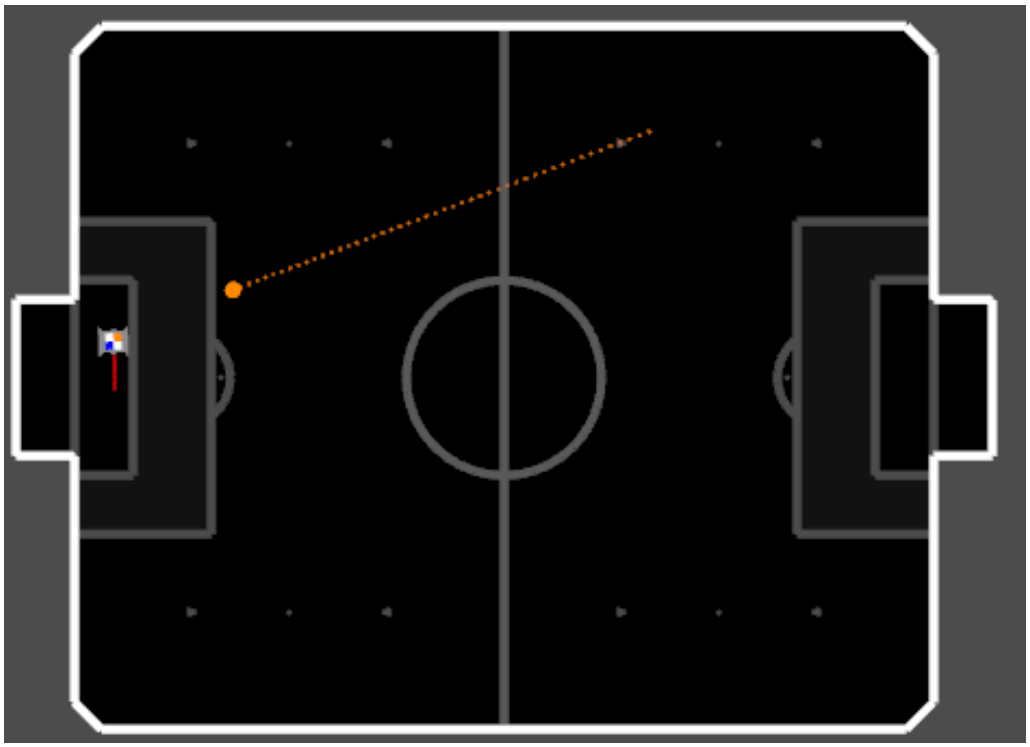


Figure D.4: Multi-Behavioural Goal Keeper Execution Image: 4

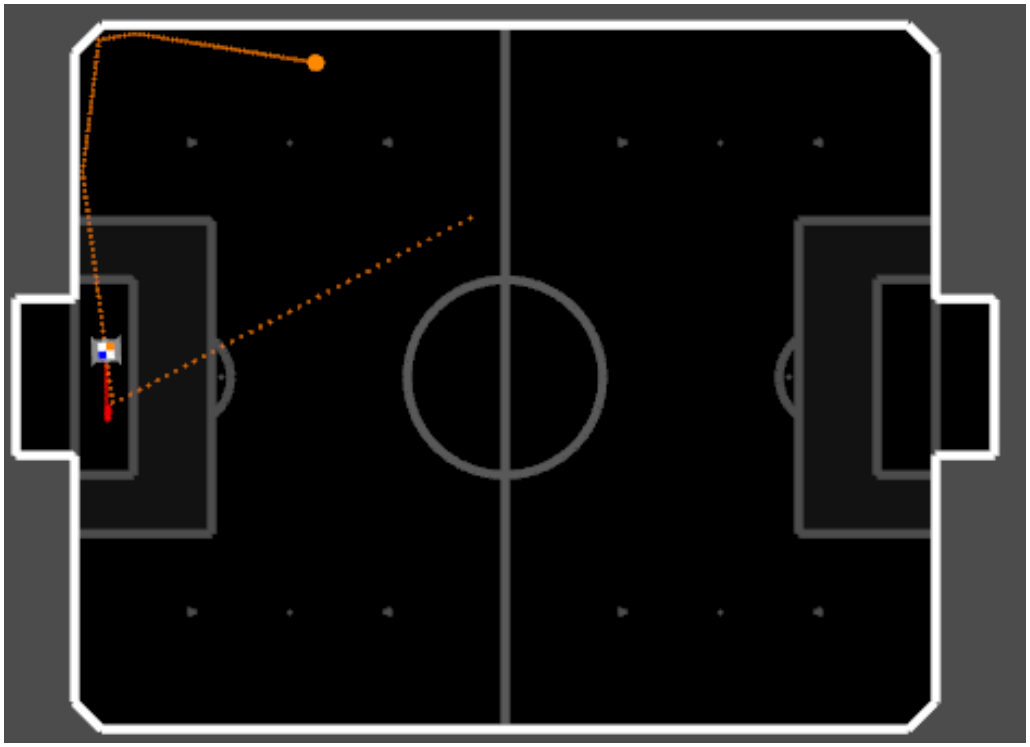


Figure D.5: Multi-Behavioural Goal Keeper Execution Image: 5

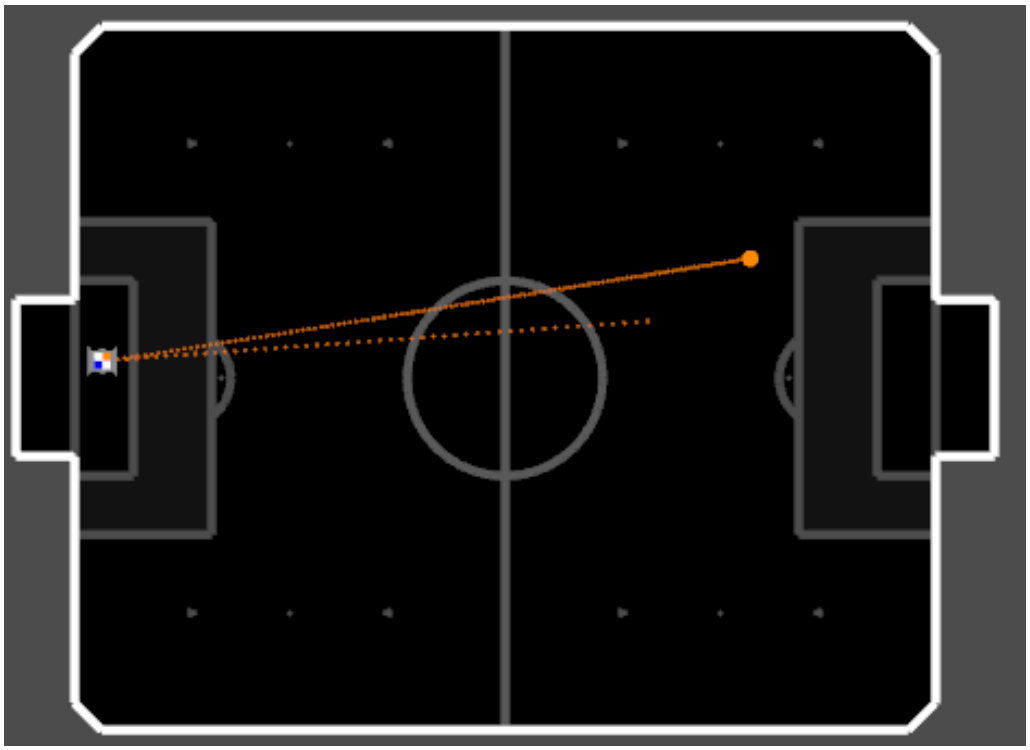


Figure D.6: Multi-Behavioural Goal Keeper Execution Image: 6