School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Control System (AI) for Wrestling Robot

Fanhui Meng

Submitted in accordance with the requirements for the degree of
MSc Advanced Computer Science (AI)

Session 2019/2020

The candidate confirms that the following have been submitted*:*

<As an example>

Items	**Format**	**Recipient(s) and Date**
Project Report	*Report*	*Supervisor, assessor, SSO (28th, August, 2020)*
Source code	*URL:* https://github.com/yeswhos/COMP5200M-MSc-Project	*Supervisor, assessor (28th, August, 2020)*
Demonstration video	*URL:* https://youtu.be/RahYl37PvtM *and* https://youtu.be/Me6_-5wUBW4	*Supervisor, assessor (28th, August, 2020)*

Type of Project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) 

Summary

The artificial intelligence technology has developed for many decades and used in many fields. This project is focusing on using AI technology, design a high-level control system for Zumo robot in the Sumo robot league.

The Sumo robot league is a popular robot competition, and the main rule of this competition is two vehicle-like robots without mechanical arm trying to push each other out of the ring. The Zumo robot is the one that is going to use in this project. It's an off the shelf Arduino-based robot. Thus, no structure design or hardware design (e.g. circuit design) or low-level design (e.g. PWM motor speed control) in this project.

The high-level control system means the 'brain' of the robot. 'The brain' should have strategies to cope with different situations. There are many machine learning methods to develop strategies for the robot. Mainly, there are two kinds of machine learning can be applied in this project, which is the supervised learning and unsupervised learning. One is to develop the control system by telling the robot, which is the right thing to do and what is wrong. Another is to develop the control system by not telling what the robot should do, but only reward or punishment the robot according to the rules and it's behaviours.

This project aims to implement and compare two different control systems, one is supervised, and another is unsupervised.

Furthermore, this project will do more research about how the same control systems or the same ideas can be applied in other robot competitions or the robot in daily life (e.g. sweeping robot).

Acknowledgements

First of all, I would like to thank my supervisor Dr Samuel Wilson and Dr Amy Lowe, for their patient and excellent help and support. The weekly virtual meetings with them guide me through every stage of research, experiment and writing of my thesis. Also, I'm grateful to Dr Mehmet Dogar. As my assessor, he gave me valuable feedback, insightful comments and guided me on how to conduct a good project.

Secondly, I would like to express my gratitude to my tutor Dr Netta Cohen, for her concern and guidance about my academic and daily life throughout my whole master year.

Last but not least, I do appreciate the support and encouragement from my parents, grandparents and other relatives from house Meng. I wouldn't go this far without their concern and help, especially during this global pandemic period.

Table of Contents

Summary……iii
Acknowledgements……iv
Table of Contents……..v
List of Figures……vii
List of Tables……....ix
List of Abbreviations…..……x
Chapter 1 Introduction……1
1.1 Project Aim……...…...1
1.2 Objective……..……...…...2
1.3 Deliverables..……...…...…2
1.4 Ethical, legal, and social issues..……..……....…2
Chapter 2 Background Research……3
2.1 Literature Survey……3
2.1.1 Sumo league rules ……..……..……3
2.1.2 Fuzzy logic control ……..…..……4
2.1.3 Neuro-Fuzzy hybird system ……..……..…5
2.1.4 Genetic algorithm ……..……..……6
2.2 Methods and Techniques…..……8
2.2.1 Fuzzy logic based control system ……...……..……8
2.2.2 Decision tree based control system..……..…8
2.2.3 Artificial neural network based control system…...……10
2.2.4 Naïve bayesian model based control system..……..……10
2.2.5 Genetic algorithm based control system...……..11
2.2.6 Reinformancement learning based control system.……..…..13
2.3 Choice of Methods…..……..……..14
Chapter 3 Software Requirements and System Design……18
3.1 Software requirements……..……..……18
3.2 System design……..……..……20
Chapter 4 Software Implementation……..……24
4.1 Decision Tree Based Control System Implementation……..……24
4.1.1 Technologies employed in this system ……..……..……24
4.1.2 Create the dataset ……..……..……25
4.1.3 Generate the decision tree ……..……27

4.1.4 Code in the Zumo robot ……30
4.2 Decision Tree Based Control System Implementation……33
4.2.1 Technologies employed in this system ……33
4.2.2 Create simulated environment and Zumo robot ……34
4.2.3 During the assessment ……37
4.2.4 GA process ……38
4.2.5 Plot the fitness score ……41
4.2.6 Traditionally controlled robot simulation……41
Chapter 5 Software Testing and Evaluation ……43
5.1 Decision tree based control system testing……43
5.2 GA based control system testing……44
5.2.1 Tournament selection ……44
5.2.2 Ranking selection ……46
5.2.3 Roulette wheel selection ……47
5.2.4 GA agent vs traditionally controlled robot ……49
Chapter 6 Conclusions and Future Work……51
6.1 Conclusions……51
6.2 Future work……52
List of References……54
Appendix A External Materials……56
Appendix B Ethical Issues Addressed……57
Appendix C Source Code and Set-up Guide ……58

List of Figures

Figure 1.1 Main features of the Zumo 32U4 robot

Figure 2.1 Decision tree example for surviver on Titanic (Wikipedia Contributors, 2019)

Figure 2.2 General process for GA

Figure 2.3 General procedure for reinforcement learning

Figure 3.1 Decision tree based control system general architecture

Figure 3.2 The general architecture of GA control system

Figure 4.1 Example code for display sensor reading

Figure 4.2 Experiment reading and distance in real world

Figure 4.3 Part of the dataset for decision tree

Figure 4.4 Code of fetch data in Colab

Figure 4.5 Code of generate dictionary to store data

Figure 4.6 Generated DataFrame data and print the result

Figure 4.7 Code of data serialization

Figure 4.8 Generate decision tree

Figure 4.9 Detail of generated decision tree

Figure 4.10 Test one predicted result

Figure 4.11 Visualization and save the decision tree

Figure 4.12 Visualized decision tree

Figure 4.13 Transfer the decision tree inot JSON format

Figure 4.14 Part of the JSON decision tree

Figure 4.15 Initialize essential sensors

Figure 4.16 Code for JSON format decision tree.

Figure 4.17 ‘Object detected’ corresponding control mode

Figure 4.18 *turn()* function in control mode

Figure 4.19 Code for not running out of the ring

Figure 4.20 Code for the ring object.

Figure 4.21 Code for the agent

Figure 4.22 Code for implement classes into simulation class

Figure 4.23 Comparison of real world and simulation

Figure 4.24 The sensor range comparison in real world and simulation

Figure 4.25 *ShrewSimulation* Code for population and assesssment set

Figure 4.26 Code for reset

Figure 4.27 Fitness code for ZumoKing

Figure 4.28 Fitness code for ZumoQueen

Figure 4.29 Code for selection

Figure 4.30 Code for crossover and mutation

Figure 4.31 Code for *ZumoLord* tradition control

Figure 5.1 Storage and memory occupation for decision tree control system

Figure 5.2 Tournament selection fitness score for ZumoKing

Figure 5.3 Tournament selection fitness score for ZumoQueen

Figure 5.4 Rank selection fitness score for ZumoKing

Figure 5.4 Rank selection fitness score for ZumoQueen

Figure 5.5 Roulette wheel selection fitness score for ZumoKing

Figure 5.6 Roulette wheel selection fitness score for ZumoQueen

Figure 5.7 GA agent fight with traditionally controlled enemy

Figure 6.1 Robots demonstration in RoboMaster competition

List of Tables

Table 2.1 Fuzzy rules for sumo robot (Erdem, 2007)

Table 2.2 Genes for robot attributes (Lehner et al., 2019)

Table 2.3 Occurrence of top 10 fittest genetic combinations (Lehner et al., 2019)

Table 2.4 Bayesian example for sumo robot

Table 2.5 Example for Q chart

Table 3.1 Function requirement for decision tree based control system

Table 3.2 Function requirement for GA based control system

Table 4.1 Libraries and purposes in decision tree based control system

Table 4.2 Libraries and purposes in decision tree based control system

Table 4.3 Reading and distance comparison table

Table 4.4 Libraries and purposes in GA based control system

List of Abbreviations

BEAST	Bio-inspired Evolutionary Agent Simulation Toolkit
GA	Genetic Algorithm
FLC	Fuzzy Logic Control
NF	Neuro Fuzzy
FIS	Fuzzy Inference System
ANN	Artifical Neural Network
NP	Non-deterministic Polynomial Problem
MDP	Markov Decision Process
FNN / FFN	Feedforward Neural Network / Feedforward Network

Chapter 1
Introduction

1.1 Project Aim

This project aims to design an excellent robot high-level control system, which is the 'brain' of the non-arm Zumo robot. The control system would have its own strategy and drive the robot's movement during the competition. As for the strategy, it can be assigned with a specific action in different cases, or the robot can develop its own strategy, which may be related to the evolutionary algorithm. And this will be discussed later in the report.

The main goal of this project is to make the Zumo robot be competitive and perform well in the Sumo robot competition. Besides, this project will find out if the idea of a control system can be applied to a broader range of different robots, such as other robot competition or the robot service in daily life.

This project is going to use the Zumo 32U4 robot, which is a complete, versatile robot controlled by an Arduino-compatible Atmega32U4 microcontroller. Therefore, extra hardware structure and improvement is not considered in the project. The Zumo robot has two motors, one Atmega32U4 chip as the brain and a variety of sensors, including two front proximity sensors, two side proximity sensors, five linesensors (three of them are the receiver, two of them are emission) and accelerometer. So the Zumo robot can detect the opponent and run towards or away from it, which satisfy every requirement of Sumo league.

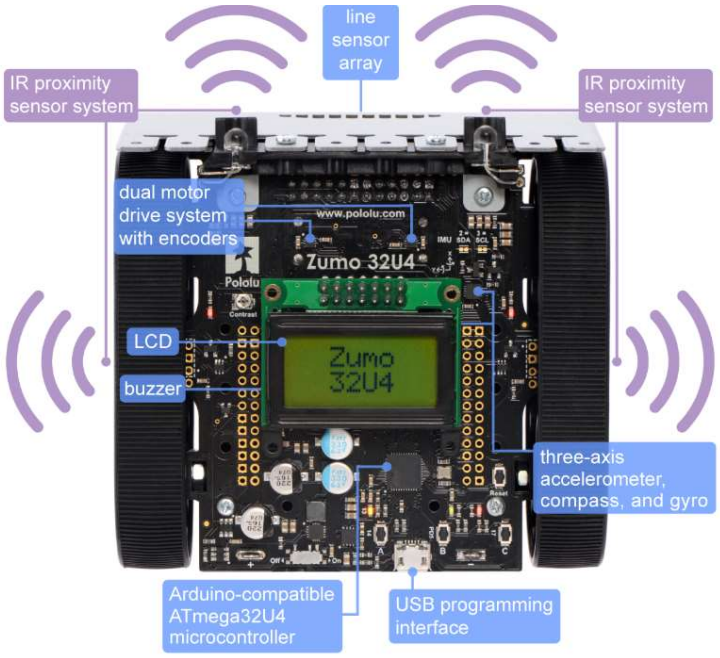

Figure 1.1: Main features of the Zumo 32U4 robot

Sumo robot league is a prevalent international robot wrestling competition, which is two robots attempt to push each other out of the ring. Thus, the wrestling robot must be capable of autonomously locating the opposing robot and pushing it out of the ring (Wilson et al., 2016). The last stand robot in the ring is the winner. Also, there are other rules in the Sumo league, which the control system should be designed according to these rules. And these rules will be explained in details in the next chapter.

1.2 Objectives:

- To get familiar with the hardware functions of the Zumo robot.
- Conduct a theory study to compare different control system in the wrestling case.
- Implement two or more different control system and compare its advantages and disadvantages.
- Create a wrestling simulation environment. (Due to the lockdown policy, it's hard to find the opponent in the real world)
- Evaluate the results with its performance in different simulation cases.

1.3 Deliverables

1. A Github repository that contains the source code of both decision tree and genetic algorithm based control system. The repository also includes the set-up guidance for the users.
2. Two demonstration videos show the experiment performance of decision tree and genetic algorithm based control system.
3. The MSc project report.

1.4 Ethical, legal, and social issues

This project is trying to avoid all ethical issue. All the data used in developing the software are from the Zumo robot sensor data and the BEAST, no personal data or classify data contains in this project. All Arduino library and BEAST external code that used in developing the software are open sources. And these codes were not credited as own work. These codes will be downloaded and put in a separate folder in the repositories to avoid plagiarisms.

And this project's code and ideas should not be used in the military field or any field that could cause violence action.

Chapter 2
Background Research

2.1 Literature Survey

2.1.1 Sumo League Rules

The sumo robot contest feature two self-controlled robots trying to push each other out of a ring. (Anon, n.d.) The basic rules are the first robot that touches outsides of the ring loses the round, and the last stand robot wins the round. The match is the best of three sets. And the robot wins the most matches are winning the contest.

Several structure requirements for the participant robot (e.g. robot size, mass, etc.) are clearly defined in the sumo robot rules. But these requirements are not being considered in this report, since this project's experiment and simulation are based on ready-made Zumo robot, which already satisfies all the hardware and structure aspect requirements.

Regarding the contest environment, it's a large, flat ring. The ring made of smooth, rigid wood. The top surface is dull black, and all of these black areas are in bounds.

A robot usually started by pressing a button or other ways, such as hand-clapping, a whistle and so forth. After start command, the contestants immediately leave the outer area around the ring. Each robot must not move completely within five seconds.

Five seconds after pressing the button, the robot should move and start the competition. However, if the robot does not move for a long time after the start command. It will be punished or judged for losing the round.

During the round, all people and object must be kept out of the ring to avoid distracting the robots or altering its outcome.

When any part of the robot (including the touch sensor, beard, scoop or skirt) touches the outside of the ring, the robot will be considered as lose its round. No matter how the robot falls out of the ring. Or any components of the robot falling out of the circle are also considered to have lost the round.

If the match satisfy any of the conditions below, the reference might choose to restart a round or give warning to the contestants.

- Three minutes have expired.
- No progress has been made in some period of time
- The robots fall to touch each other for some period of time
- The robots are hopelessly entangled or otherwise deadlocked

- Both robots fail to start or both contestants signal stoppage.

This project design is based on the sumo rules above. The robot's action and strategy, and the way that evolve the robot's strategy should be impacted by the rules as well. For example, the robot's should not stay still too long, even this is part of the strategy, because the rule will punish the robot that doesn't move for an extended period.

There are pretty much studies in Sumo robot control and even more research on the general intelligence control system. And there are also different aspects of this project. Some are fucus on the low-level development, and some aim to design the hardware or the construction of the Sumo robot, others are focus on electronics aspect and so on. And this project is going to focus on the high-level design.

2.1.2 Fuzzy Logic Control

One study used a single fuzzy logic control (FLC) as a microcontroller to detect and track enemy inside the sumo ring. Three sharp infrared sensors are used for opponent detection. Then, the fuzzy controller fuses the sensor data and send the signal to the motor to drive the robot heading to the enemy (Erdem, 2007).

Then, the author developed the fuzzy rules. These intuitive rules are developed according to all possible scenarios using input sensor readings. For instance, one of the fuzzy rules is that if both S1 and S2 detect the weak signals, and S3 detects a medium signal, the reasonable action for the robot is to control the motor to turn a small circle to the right. Figure 2.1 shows the optimization rules used to detect and track targets.

Rule	S1	S2	S3	Motion	Rule	S1	S2	S3	Motion
Rule1	Low	Low	Medium	Small R.	Rule12	Medium	Medium	High	Center
Rule2	Low	Low	High	Full R.	Rule13	Medium	High	Low	Small L.
Rule3	Low	Medium	Low	Center	Rule14	Medium	High	Medium	Center
Rule4	Low	Medium	Medium	Small R.	Rule15	Medium	High	High	Center
Rule5	Low	Medium	High	Small R.	Rule16	High	Low	Low	Full L.
Rule6	Low	High	Low	Center	Rule17	High	Medium	Low	Small L.
Rule7	Low	High	Medium	Small R.	Rule18	High	Medium	Medium	Center
Rule8	Low	High	High	Center	Rule19	High	High	Low	Center
Rule9	Medium	Low	Low	Small L.	Rule20	High	High	Medium	Center
Rule10	Medium	Medium	Low	Small L.	Rule21	High	High	High	Center
Rule11	Medium	Medium	Medium	Center					

Table 2.1: Fuzzy rules for sumo robot (Erdem, 2007)

For every sensor, there are three fuzzy sets, including Low, Medium, High. The five singleton membership functions are Full Left, Small Right, etc. Based on the output of the fuzzy controller, the motor will make the robot turn left or right.

The experiments result concluded as two aspects. First is the sensors perception area, when sumo robot is staying still, three proximity sensors detect moving object nearly for 0.02 second. The second experiment is when the head of the robot is facing the opposed of the opponent, and the sensor cannot detect the opponent. The robot then executes the search mode and turn its head towards the target, then accelerate until it hits the target and pushes the opponent out of the ring. The results show that under this condition, that is, in the search mode and rotating motion, the robot can respond and rotate, and detect the opponent within 0.2 seconds because it is within the sensor range and hits the opponent every time.

All in all, the designed system would execute in much lower time when it compares to the traditionally controlled sumo robot. And it also has advantages in fast detecting, positioning and manoeuvring in time.

2.1.3 Neuro-Fuzzy Hybrid System

Similarly, another project also uses fuzzy logic as the main idea of sumo robot control. It uses the Neuro-Fuzzy (NF) hybrid system as the control system (Erdem, 2011). In other words, it uses two systems, namely artificial neural network and fuzzy inference system (FIS). FIS is used to detect and track enemy, which correlates sensor output (IR sensor) with motor control pulses. ANN is used for rule extraction and adjustment of FIS parameters.

Furthermore, the NEFCLASS model is introduced, which can be evaluated as a hybrid ANN-based system. NEFCLASS has a structure similar to ANN. It uses fuzzy parameters or can be interpreted as FIS implemented in a parallel structure. The primary function of NEFCLASS is to use learning algorithms to build a structure (rule base) and adjust the fuzzy classifier's parameters (fuzzy set) based on system data. Or, in other words, NEFCLASS adds and deletes rules to optimizes FIS parameters.

In this way, with FIS and NEFCLASS, the number of original fuzzy rules can be pruning down to less. With the optimized number of fuzzy rules, the robot’s control can respond faster than before.

And the result shows that this NF control system can improve robot performance in the wrestling league. Compared with the traditional method (single fuzzy control), the application of the learning algorithm improved the robot's response speed much more than expected. It improves the response time by reducing 30% of the rule. Thus, the robot can attack the opponent with minimal movement and manoeuvres. Performance can be improved by adding additional sensors to perceive opponents, which can be regarded as the direction of future research.

This is an excellent thought to develop a good control system. It's just like human work out to improve physical strength, but I would focus on brain development. However, the wrestling

environment is uncertain, and the data is non-linear. Thus, the method that using ANN to improve fuzzy control is a good thought for high-level control in general.

However, this previous study uses the fuzzy control system to improves the robot response during the competition by defining the relationship between sensor output signals and motors control pulses. The result turns out remarkable, but when it applies in high-level design, there should be a simpler solution with the same outcome. Especially in this project, the sensor range on the Zumo robot is from zero to six, which is not accurate as motor pulses. The reading is fuzzy enough, and it doesn't need more fuzzy inference rules.

2.1.4 Genetic Algorithm

In addition, some studies use genetic algorithms (GA) to optimize the control of sumo robots. A study used Java, Eclipse and the dyn4j physics engine to simulate the battle of a sumo robot (Lehner et al., 2019). Each robot has six attributes (such as speed, search range), which is GA controlled. And these attributes also are assigned to corresponding to the gene positions. Characteristics are the intensity of each genetically controlled capability, and they are represented with a value between 1 and 9.

The genes and the corresponding control capabilities are shown below.

Position	Attribute	Genetically controlled capability	Characteristic
0	A	Speed: The speed of the robot implies higher force in case of a collision, more pushing power, etc.	1-9
1	B	Search Range: It defines at what distance another robot is recognized and consequently can be attacked.	1-9
2	C	Wall Detection Range: It defines at what distance an arena wall is detected.	1-9
3	D	Size: The size of the robot is here a measure for the mass of the robot and therefore leads to a higher physical impact in collisions.	1-9
4	E	Strategy Persistency: Robots have four different modes to operate (search, attack, survive, cruise). A strategy persistency value of 9 means that a robot sticks to its mode for the next 40 calculations. A value of 1 means that it only sticks for the next 10 calculations to its mode and then randomly changes to a new mode. For the values 2 to 8, it works correspondingly between 10 and 40.	1-9
5	F	Driving Accuracy: Defines the accuracy of a robot to drive straighter to its target with less intermediate random movements in other directions.	1-9

Table 2.2: Genes for robot attributes (Lehner et al., 2019)

At the beginning of each battle, each robot will receive a randomly selected combination of values, which is the combination characteristics of the robot. Then fight to determine the highest fitness. Fitness is defined as how many actions need to be performed until being defeated or standing at the end. The next step is the crossover and selection of the next generation. This project uses roulette wheel selection, which assigns the probability of selecting a robot based on the relative likelihood of each robot's fitness. The 50% mutation rate of each gene can also be applied to increase the probability of randomly changing characteristics. Finally, create a new generation and transfer the new generation into a brand new cycle. After thousands of generations of efforts, the last robot standing on the ring is the ultimate success gene of GA optimization.

The experiment summarized the top fittest genetic combination and produced 630 different gene sets from up to 1200 generations. Results shows that in different variants, there are no winning genes observed.

Hence, the advantage of the winning gene is random because there is no combination of winning genes emerges in more than one variant. One robot cannot get a successful strategy from the isolated analysis of a single functional level. Because in the winning genes, the dominant attribute does not have the highest feature value. Thus, as comparing individuals with mixed genes, there is no evidence for the gene with high characteristic value exist. The reason for this is because the starting position of each robot in each new fight is random, and the researchers say this needs further research.

Genes Capability (A B C D E F)	Fittest	Second Fittest	Third Fittest
214319	10	10	10
422612	10	10	10
787931	10	10	10
141523	10	10	9
693548	10	10	9
444383	10	8	8
327534	9	10	8
936353	9	9	10
982196	9	8	5
665166	9	7	9

Table 2.3 Occurrence of top 10 fittest genetic combinations (Lehner et al., 2019)

Another result shows that the selection methods in genetic algorithm do influence the success of the robot. Roulette wheel selection may led this experiment to a low dominance of the generated genes when it compares to the results when using the randomness, because this selection methods may allocate gene in proportion.

2.2 Methods and Techniques

2.2.1 Fuzzy Logical Control System

Fuzzy control is an intelligent control method based on fuzzy set theory and fuzzy logic inference. This is a smart control method that can imitate people's fuzzy reasoning and decision-making process based on their behaviour. The technique first compiles the experience of the operator or expert into fuzzy rules, then fuzzifies the real-time signal from the sensor. Use the fuzzed signal as the input of the fuzzy rule, completes the fuzzy inference and output the inference. (Ma and He, 1998)

Several things are essential for defining a fuzzy controller. First is the fuzzy interface, which is for performing a conversion from a specific real value of the input into a fuzzy set. For example, in this project, it may transfer the proximity sensor output, such as a number value, to a fuzzy set, like low, medium or high. Secondly, the knowledge base, which is consist of database and rule base. The database stores all the input and the fuzzy set. At the same time, the rule base is the fuzzy rules usually based on expert knowledge or personal experiment. A series of relational words often connect fuzzy rules, and the most commonly used relational words are if-then, else, etc. Finally, is the inference and defuzzy-interface.

The result of the inference shows the completeness of the rule inference function of fuzzy control. However, the result obtained still cannot be used directly as a control variable. A conversion must be made to have a precise control variable output, which is the defuzzification. In this project, this could be used for determination of real value for directing of motors or other controls.

Fuzzy logic control doesn't require an accurate mathematical model. However, the factors such as the integrity of fuzzy control rules, the definition of fuzzy subsets and the fuzzy inference mechanism will have an impact on the performance of the fuzzy controller. Thus, most of the factors depend on the experience of experts. For this project, the primary basis of control is based on a large amount of personal experience, knowledge and strategy in the sumo league.

2.2.2 Decision tree based Control System

Similarly to FLC, the performance of the decision tree based control system also depends more on personal experience and strategy in a wrestling robot. As one kind of supervised learning, the decision tree can be well applied in the classification as well as the regression problem. (Utgoff et al., 1997)

The decision tree algorithm uses a tree structure and uses layers of reasoning to achieve the final classification. It’s consist of one root node, leaf node and internal node. Root node

contains a full set of samples; the internal node corresponding characteristic attribute and the internal node represented the final result or decision.

For example, in Figure 2.1 below, the root node is the ‘gender’ node, which must be the purest feature. ‘age’, ‘sibsp’ are the internal node. ‘survived’, ‘died’ are the predicted result, which is the leaf node.

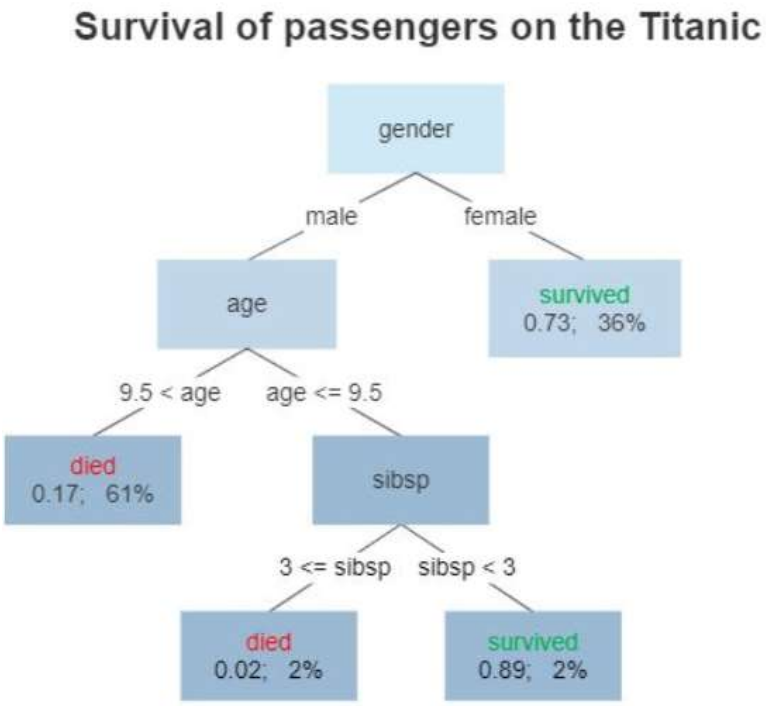

Figure 2.1: Decision tree example for the survivor on Titanic (Wikipedia Contributors, 2019)

Basically, there are three-step to build the decision tree. First is the feature selection, and it's determining which features are going to make judgements. In the training data set, each sample may have many attributes, and the effects of different features are different. Therefore, the function of feature selection is to screen out the features that are more relevant to the classification results. In short, it's intended to find out the features with strong classification ability. After the features are selected, the next step is to generate the decision tree. It's triggered from the root node, calculated purity for the node. The feature with the most purity is selected as the node feature. As for its child nodes are established according to different values of the attributes. Each child node is generated in the same way until purity is the lowest, or there are no features left to choose from.

The key things here is the word ‘purity’. It’s deciding which is the optimal partition feature. And it means if most of the sample that contained in one tree node belongs to one category, then we say this tree node is pretty pure. There are mainly two ways of representing purity, which is the information entropy and Gini index. According to this, three decision tree algorithm is introduced, including ID3, C4.5 and CRAT. ID3 based on information gain to choose features. C4.5 is quite similar to ID3, which use information gain ratio. And CRAT

uses the Gini index to select features, which can be used in both classification and regression problem.

After generating the decision tree, a final step is an option, which is pruning. The primary purpose of pruning is to prevent overfitting by removing unnecessary branches.

Furthermore, the random forest is another possible solution for the control system, which made of many independent decision trees. When conducting a classification task, new input samples will be entered. Then the tree in the forest will begin to judge and classify these samples separately. Finally, each decision tree will conduct its own classification result. The conclusion that most of the decision tree classified will be the final result for the random forest.

2.2.3 Artificial neural network based control system

Artificial Neural Network (ANN) is a computational model consisting of a huge number of interconnected artificial neurons. Each node represents a specific output function. Each connection between two nodes represents a weighted value and possibly a basis for the signal passing through the connection. The network itself normally can be seen as an approximation of a specific algorithm or function or an expression of a logical strategy. (Fine, 1996).

Feedforward neural network (FNN) is one kind of ANN. It's the most straightforward and widely used type of ANN. In this network, the information moves in only by forward. There are no cycles or loops in the network.

For this project, the sensor value can be the input of the network, and output is the generated strategy. Moreover, the ANN's parameter (e.g. weight, basis) can be adjusted or optimized or evolved through training. Thus, a different way of optimization can be applied, such as Neuro-fuzzy or GA with FNN.

2.2.4 Naïve bayesian model based control system

Bayesian is a kind of classification method. For example, suppose a robot's right proximity sensor detects obstacle while the left IR detect nothing. In that case, this situation can be classified as an obstacle in the right of the robot. Content of the classification algorithm is to require a given feature to let us get the category, which is an important idea to solve all the classification problems. And how to specify the characteristics to get our final category is the key thing to be considered. (Yao and Zhou, 2012)

One example of the Zumo robot bayesian model shows in Table 2.4. For proximity sensor, 0 means the distance to the obstacle is close, one means the distance is far, two means too far. For linesensor, 0 means out of the ring, one expects in the ring.

Proximity Sensor	Proximity Sensor	Line Sensor	Line Sensor	Label

Front Right (S1)	Front Left (S2)	Right (S3)	Left (S4)	
0	0	1	1	Enemy Front
0	1	1	1	Enemy Front Right
0	2	1	1	Enemy Front Right
1	0	1	1	Enemy Front Left
1	1	1	1	Enemy Front
1	2	1	1	Enemy Front Left
…	…	…	…	…

Table 2.4 Bayesian example for sumo robot

The bayesian formula is $p(B|A) = \frac{P(A|B)P(B)}{P(A)}$. A means the features, and B means the label or category. And each feature should be independent.

So in the sumo case example, if it require to calculate the possibility if the enemy is in the front or left or right, and the sensor values are S1, S2, S3, S4 respectively. Then use bayesian formula to calculate the possibility of front or left or right respectively.

First, calculate the possiblity of enemy in the front. It can be represented as below.
$P(Enemy\ Front|S1, S2, S3, S4) = \frac{P(S1,S2,S3,S4 \mid Enemy\ Front)\ P(Enemy\ Front)}{P(S1,S2,S3,S4)}$

Because of each feature is independent, $P(S1, S2, S3, S4\ |Enemy\ Front) = P(S1|Enemy\ Front)P(S2|Enemy\ Front)P(S3|Enemy\ Front)P(S4|Enemy\ Front)$.

And $P(S1, S2, S3, S4) = P(S1)P(S2)P(S3)P(S4)$

After getting everything we need, $P(Enemy\ Front|S1, S2, S3, S4)$ can be calculate.

Then do the same to $P(Enemy\ Front\ left|S1, S2, S3, S4)$ and $P(Enemy\ Front\ right|S1, S2, S3, S4)$. The posibliity with the largest value is the final predicted result.

In general, the good thing about bayesian is that it's easy to implement and supposed to have less error rate.

2.2.5 Genetic algorithm based control system

Genetic Algorithm (GA) is based on Darwin's theory of evolution, simulating natural selection and the survival of the fittest. The primary purpose of GA is to keep the best gene so that it will evolve better and more fit gene or individuals. (Whitley, 1994)

GA usually uses a fixed-length linear binary representation for its genotype. It can be designed as containing the control strategy of the sumo robot. And it's a heuristic algorithm,

which is suitable for Non-deterministic polynomial (NP) problem. This will be explained in the next section.

The general steps of GA is shown in Figure2.2.

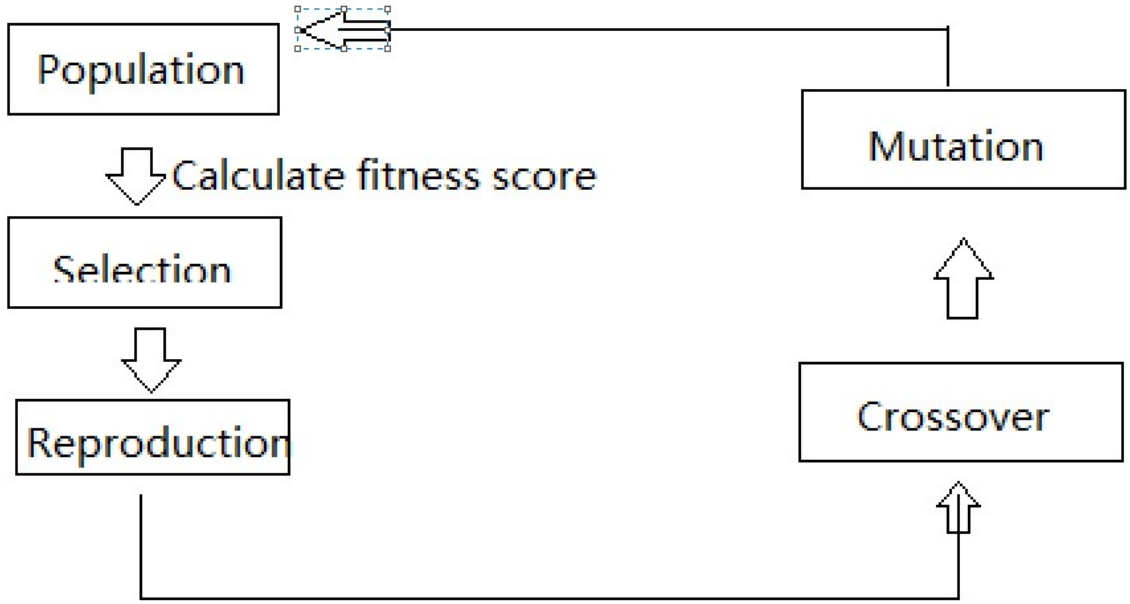

Figure2.2: General process for GA

At the first generation, the population will be the group of randomly generated solution. Then use the fitness function to calculate the fitness of each individual separately.

Fitness is the most crucial thing about GA. It's playing the role of 'God' in GA. Fitness score measures the pros and cons of individual and decides who is going to stay or be eliminated. For example, in sumo cases, the fitness can be if the robot wins the most matches, then it'll get more fitness score, while the one loses the most games are tend to get a lower score. GA will perform N generations, and each generation will generate several individuals, which is population. The fitness function will give a score to all the individual to judge the fitness for each individual. Only the individual with higher fitness are retained, so the quality of the population will become better and better after several iterations.

After calculating the fitness score, the next step will be the selection, which is also quite crucial for GA. There are several ways of selection. For example, the main idea of roulette wheel selection is that individuals with better fitness values are more likely to be selected. The tournament selection first choose a number of individuals from the population each assessment. Then selects the best one according to the fitness score to enter the offspring population. Repeat this operation again and again, until the new population has the same population size as the original. Ranked based selection, which only selects the top individual according to fitness score. There are also many other selection methods, like stochastic tournament, excepted value selection, truncation selection, and so on.

After selection, GA will finally go to the evolution process. Do the crossover first, then mutation. The crossover needs to find two chromosomes from the selected individuals of the previous generation. Then cut a specific position of the two chromosomes and splice them together to generate a new chromosome. This new chromosome contains a certain number of the two individuals' genes from the last generations. And there are many different ways of design the crossover proportion. For example, 70% of the new chromosome comes from the highest fitness score, 30% from the second. Mutation means to change the small part of the genes randomly. Introduce mutation can help the solution escape from local optimal and is helpful for the algorithm to find the global optimal solution.

Then calculate these new generations' fitness score, and do the selection, crossover, mutation again until the result is good enough or reach the number of iterations limits.

2.2.6 Reinforcement learning based control system

Reinforcement learning is also unsupervised learning. And it's usually described by Markov Decision Process (MDP). Then the overall concept is for reinforcement learning is that a machine is in an environment, and it can only interact with the environment through actions. And the state is the machine's perception of the current environment. When the robot make an action, the environment will feedback to the robot a reward based on the potential reward function. (François-Lavet et al., 2018).

Also, it will make the environment transfer to another state according to a certain probability. In summary, reinforcement learning mainly includes four elements, which are state, action, transition probability and reward function.

The general procedure for reinforcement learning shown in Figure 2.3.

Figure 2.3 General procedure for reinforcement learning

As the agent performs a certain task, it first interacts with the environment to generate a new state, and the environment gives a reward at the same time. Then continue the iteration, the agent and the environment continue to interact to generate more data, and modify its own

action strategy according to these new data. After several epoches, the agent will learn the action strategy which is needed to complete the task.

In this project, the agent would be the Zumo robot. And state including the environment state, which is the sumo ring, agent state, which is the input data for agent and information state, which include useful information needed for future prediction. The environment is the partially observable environment, which means the agent will discover the environment by with its sensor.

There are several algorithms for reinforcement learning, including Q learning, Sara, Deep Q network and so on. For example, the famous Q learning based on Q chart like the chart below.

Q	A1	A2
S1	-3	3
S2	-4	4

Table 2.5 Example for Q chart

S means the state, and A means action in Table 2.4. In this Table 2.4, $Q(S1, A1) < Q(S1, A2)$, which means A2 rewards more than A1. Thus, it would be reasonable to choose A2 as an action. Then comes to S2, repeat the previous steps, choose the most rewards action and get to the next state.

However, the Q chart includes the estimated value, so this chart needs to be updated. For example, $Q(S1, A1) < Q(S1, A2)$, so multiply the max value by an attenuation value gamma (e.g. 0.8) and add the reward when reaching S2. And this value is seen as the actual value. Now we got the actual and estimated value, and the Q chart is able to be updated. According to the difference between estimation and actual, multiply this difference by a learning rate and add the old estimation value $Q(S1, A2)$, then got the new updated value.

2.3 Choice of methods

There are many AI methods out there for designing the control system for the Zumo robot. I always do believe the same that the best way to evaluate one method or idea is to experiment with it. However, there is no time for me to implement and test all the AI technologies in the Zumo robot. So I have to choose a suitable method according to theoretical research.

In my opinion, two different kinds of methods are necessary to be implemented to the control system in this project. One is supervised learning, and another is unsupervised learning.

Supervised learning means that with input variable and output variable, and use a certain algorithm to learn the function from input to output. By this method, the human strategy can be implemented into the Zumo robot, and the control system can design as what people want the behaviours of the robot.

As for the decision tree, which is one kind of supervised learning, it's easy to understand and explain, can be visualised and analysed. And it's an excellent way to get a start and familiar with this project. It's able to handle both data type and regular type attributes. Also, when dealing with the data set, the program run time can be very fast, which means the time complexity is low. The speed of running code is quite important because the Zumo robot is off the shelf robot, the processor and memory on the Zumo robot are fixed. And it can't compile too many codes and unable to run the code with much time complexity. The sensor output of the robot can be outdated soon, so the robot should react timely according to the current sensor output. And these conditions emphasize that the code should not be too complicated, and it should be as easy as possible. For the decision tree, the code can be less complex and easy to compile and run, and it will perform the strategy correctly as expected.

As for unsupervised learning, which is going to be considered as the majority part of this project. Because in the sumo league, it's hard to say what is the right thing to do and what is wrong. And no one knows the absolutely undefeated, unrivalled strategy. However, this is just subjective thinking, which kind of learning method have better performance has to be determined by experiment.

Wrestling robot fight can be defined as an NP problem. Don't know if a correct solution or strategy exists for sumo league. However, for any possible solution or strategy can be verified in polynomial time. For NP problem as well as sumo league, there are no perfect solution or global optimal. In this case, the heuristic algorithm is suitable for this kind of problem, and it will provide a feasible solution within an acceptable time.

GA is a kind of unsupervised learning as well as a heuristic algorithm, which is suitable for this project. GA was introduced in *COMP5400M Bio-inspired computing* module last semester, which gives me a general understanding of GA, including the concept of heredity, variation, selection, etc. (Marc, 2020). Also knowing that GA has many applications, including automatic design robotic lifeforms (Lipson and Pollack, 2000), optimise the shape of airfoils (Jahangirian and Shahrokhi, 2011) and so on. A GA simulation toolkit called BEAST is also introduced in this module. BEAST can be used to simulate many cases using GA, like the mouse with cheese, predators and prey simulation. (Marc, 2020).

Since the parameters of the problem space cannot be solved by GA, it must be expressed as chromosomes or individuals in the genetic space through coding. This is a kind of problem representation. And as *COMP5450M Knowledge Representation and Reasoning*

module mentioned that one excellent problem representation should be completeness, soundness and nonredundancy. (Brandon Bennett, 2019).

In this case:

- Completeness means, all candidate solutions in the problem space can be represented as chromosomes in the GA search space.
- Soundness means, the chromosomes in the GA space can correspond to all candidate solutions in the problem space.
- Nonredundancy means, one to one correspondence between chromosomes and candidate solutions.

As mentioned above, this project is an NP problem, it's impossible to list to represent all the solutions by using chromosomes. Thus, we hope the way of representation should be as completeness and soundness as possible, and GA is suitable for this project.

Chromosome, as well as GA, has a large search space, the crossover and mutation can create multiple combinations as the candidate solutions. With a large search space, GA would reduce the chances of falling into a local optimal. The GA would only use probabilistic transition rules to guide his search direction rather than using deterministic rules. The fitness function can be customized, which can have more possibilities. And these characteristics of GA makes GA get more 'fitness score' in this project.

FNN and GA can be a great combination, tuning of the parameters of an FNN using GA (Leung et al., 2003). The way of choosing the number of hidden nodes is to start with a small number, then increase the number until the learning performance or the fitness score is good enough. The results show that using the improved genetic algorithm, the proposed neural network can learn the input-output relationship and network structure of the application. Finally, after learning, a given fully connected neural network can be reduced to a partially connected network. Therefore, this way can reduce the implementation cost of FNN.

With BEAST, the idea of GA with FNN can be implemented, an Animat class with a built-in feed-forward network and GA compatibility is already provided. This class returns the neural net's configuration vector (parameters like weight and basis in FNN) as its genotype. It initialises the net with one input per sensor and one output per control (e.g. the right and left wheel in the basic Animat). The built-in FNN's input, output, number of the hidden layer are able to set. All in all, the control method would be: feed the input from the sensor into the FNN and then sets the motors using the network's output. And the way to optimise the parameter of FNN and configure the network would be using GA.

The simulated agent in BEAST is wheel drive robot. As the robot structure introduced in *COMP3211 intelligence system and robotics*. There are two types of wheel configurations,

one is the two-wheel differential drive, another is the car with front-wheel drive. In this project, the Zumo robot is the first type, and this kind of robot can rotate around its centre. Any motion control algorithm needs to respect the underlying kinematics. For example, no sideways motion is possible with wheels, so all the two-wheel differential drive robots are nonholonomic motion. (Mehmet, 2019). And BEAST agents is based on such type of robot rather than the car with front-wheel drive. Thus, the motion control algorithm in BEAST is suitable for Zumo robot simulation, which is one reason for using BEAST simulation.

Another reason why I prefer the simulation in this project is that GA is usually going through hundreds or even thousands of generations. There are tons of experiments in GA, and it would be impossible to experiment with the whole evolution progress in the real world. Another reason is that this year is a special year due to Covid-19, and the closedown of the school, the lockdown of the country make it’s hard to find another real sumo robot as opponents. Many competitions are cancelled, including the School of Computing Zumo Wrestling league. Thus, simulation robot wrestling would fit this project best, and this project is also going to emphasis more on the simulation.

Chapter 3
Software Requirements and System Design

3.1 Software Requirements

The project requirements focus on the design of a well-performed high-level control system. And there are two ways of designing the control system in this project, including decision tree and GA based control system. Thus, two different systems with different functional requirements (FR) are going to be listed in this chapter.

Table 3.1 state the decision tree based control system. Table 3.2 state the GA based control system.

As for the decision tree based control system. The process of generating decision tree should import essential API and libraries in Table 4.1. If you run this program in Colab then you don't need to worry the library and API, just make sure the data file is in the correct file location as it's decribed in the code. Suppose you run this program in Jupiter notebook. In that case, it's necessary to install these libraries and Python 3 and later version in advance, and put the data file in the right location.

The code in Zumo robot requires to use Arduino IDE to upload the code into Zumo robot. And essential libraries in Table 4.2 request.

After meet these software environment requirements, the decision tree based control system can be well performed, and the functional requirements are shown in Table 3.1.

FR 3.1	Requirements
FR 3.1.1	System shall generate decision tree accoding to the data of sensor reading and label.
FR 3.1.2	System shall provide three different algorithm option to generate decision. (ID3, C4.5 and CRAT)
FR 3.1.3	System shall allow detail changes for the content of action model (label). (e.g. adjust speed or rotate speed)
FR 3.1.4	System shall allow adding more kind of sensor data
FR 3.1.5	System shall visualize the generated decision tree
FR 3.1.6	System shall show each node's detail of the generated decision tree. (feature, gini/ information gain ratio, number of sample, etc.)

Table 3.1 Function requirement for decision tree based control system

As for the GA based control system, it's running in the simulation environment, which is called BEAST. BEAST can only run in Ubuntu operating system, and several set-ups are requested, which will demonstrate in the README file in my repository. The scripts for generating plot according to the GA result, are written in Python. Thus, Python 3 and later version and essential libraries in Table 4.4 request as well.

After fulfilling these software running requirements, the expected functional requirements are shown in Table 3.2.

FR 3.2	Requirements
FR 3.2.1	System shall have two independent agent class represent two Zumo robot.
FR 3.2.2	System shall create the simulation wrestling environment. (Two robots and one ring with the same proportional when compared to the real world)
FR 3.2.3	System shall have the same kind of sensor or the same function of sensor that Zumo robot has in real world. (Proximity sensor, line sensor)
FR 3.2.4	System shall have three selections algorithm options. (Roulette wheel selection, ranking selection, tournament selection)
FR 3.2.5	System shall allow the changes of parameter of GA. (number of population, crossover proportion, mutation pability)
FR 3.2.6	System shall allow the changes of FNN. (The number of input, output and hidden layer)
FR 3.2.7	System shall present the average fitness and the best fitness score in each generation.
FR 3.2.8	System shall present the time of each generation.
FR 3.2.9	System shall include the sumo robot league rules. (If one of the robot run out of the ring, restart the game and reset both robot's location, orientation, etc.)
FR 3.2.10	System shall draw the plot for the final results after evolution. (Two agents' average and best fitness score in each generation)

Table 3.2 Function requirement for GA based control system

Both two systems' requirements will be evaluated and addressed in Chapter 4 and 5, which are implementation and software testing chapter. Moreover, the README file will be provided.

For the record, because hardware and structure are not considered in this project. The sumo robot league rules like *'if any piece of the robot touches or fall outside the ring, the robot is considered out'* are not part of this project's requirement.

3.2 System Design

As the description above, this project will design two control system for Zumo robot, one is unsupervised, and another is supervised learning. In this design section, two control systems' architecture and technology will be discussed respectively.

3.2.1 Decision tree based control system design

The Zumo robot is controlled by an Arduino-compatible Atmega32U4 microcontroller, which is a fine chip, but it's memory and calculates ability are limited. So the process of generating the decision tree is going to execute in another environment. Then transfer the generated decision tree into the Zumo robot program.

In the Zumo robot program, input the sensor reading to the decision tree, then the output would be the label. The label is then corresponding to a control mode. The control mode can be written as a function, and the content of this kind of function would be the motor control. For example, if one label or control mode comes to 'Charge', then the content of this function would be set both motors at maximum speed.

And the dataset that used to generate decision tree would be written by hand, but also comes from the real experiment result. First, writing a simple program that the LCD screen would display the sensors real-time data. Then use a cup as the robot's opponent put the cup at different position of the robot and observe the sensor readings. Assign a different label or action mode to different sensor reading. Then come up with the data; each line of the data has the reading from a separate sensor and a label. And the data set would be saved in a txt file. The code should read the file and extract the data and label from it.

The dataset would be small. One reason is the microcontroller wouldn't process too much data. Another reason is that the explanation in *section 2.3 choice of method* that the decision tree is not going to be the major part of this project. Thus, a feasible but not very detailed dataset would fit this project.

As for the algorithm for generating the decision tree, there will be slightly different between different algorithm because the dataset is quite small. This project would use the CRAT algorithm, which means to use the Gini index minimization criterion to select features and

divide. However, the system would still provide three options for choosing an algorithm and use CRAT as the implementation. And the comparison of these algorithms wouldn't be involved in this report.

The general architecture from a logical point of view shown below in Figure 3.1.

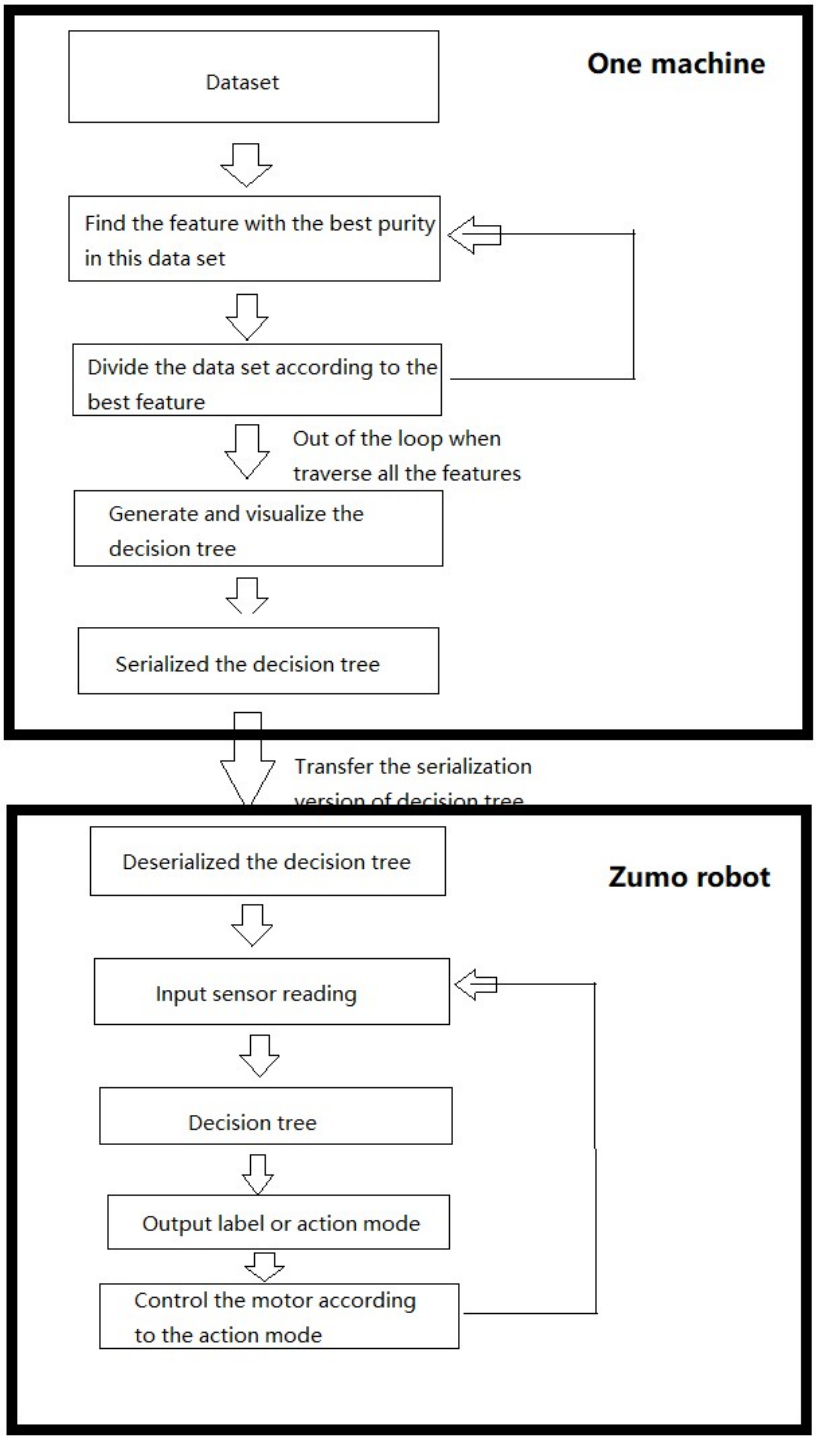

Figure 3.1: Decision tree based control system general architecture

3.3.2 GA based control system

The GA based control system is going to have an experiment in a simulated environment, including the simulated agent, sumo environment, etc.

There are two agents class in the environment. And these two agents class are not sharing the same GA process, which means their evolutions are independent, but also can be seen as co-evolution or competition relationship. This relationship just like predator and prey, they are two different species, but their evolution progress can affect each other. The common things of these two agents are that they have the same framework, which is both controlled by FNN, and GA is used to adjust the parameter of FNN. Also, the physical structure of agents is the same, including the same sensors, same size and same motors, which means the same maximum speed.

The assessment means the wrestling robot fight in within time. It can be called as test time or battle time. Consider that even the agent with better behaviour could lose one or two battles, the assessment won't stop just because one agent is out of the ring. Only when the time is up, the assessment ends. And suppose one agent is out of the ring while the assessment is not over. In that case, the system will reset everything in the environment, including the agents' position, orientation and so on. Also, there are two counters, which record how many times each agent has run out of the ring within one single assessment. And the counters can be seen as one major factor to calculate the fitness score.

Then general architecture of GA control system is shown in Figure 3.2.

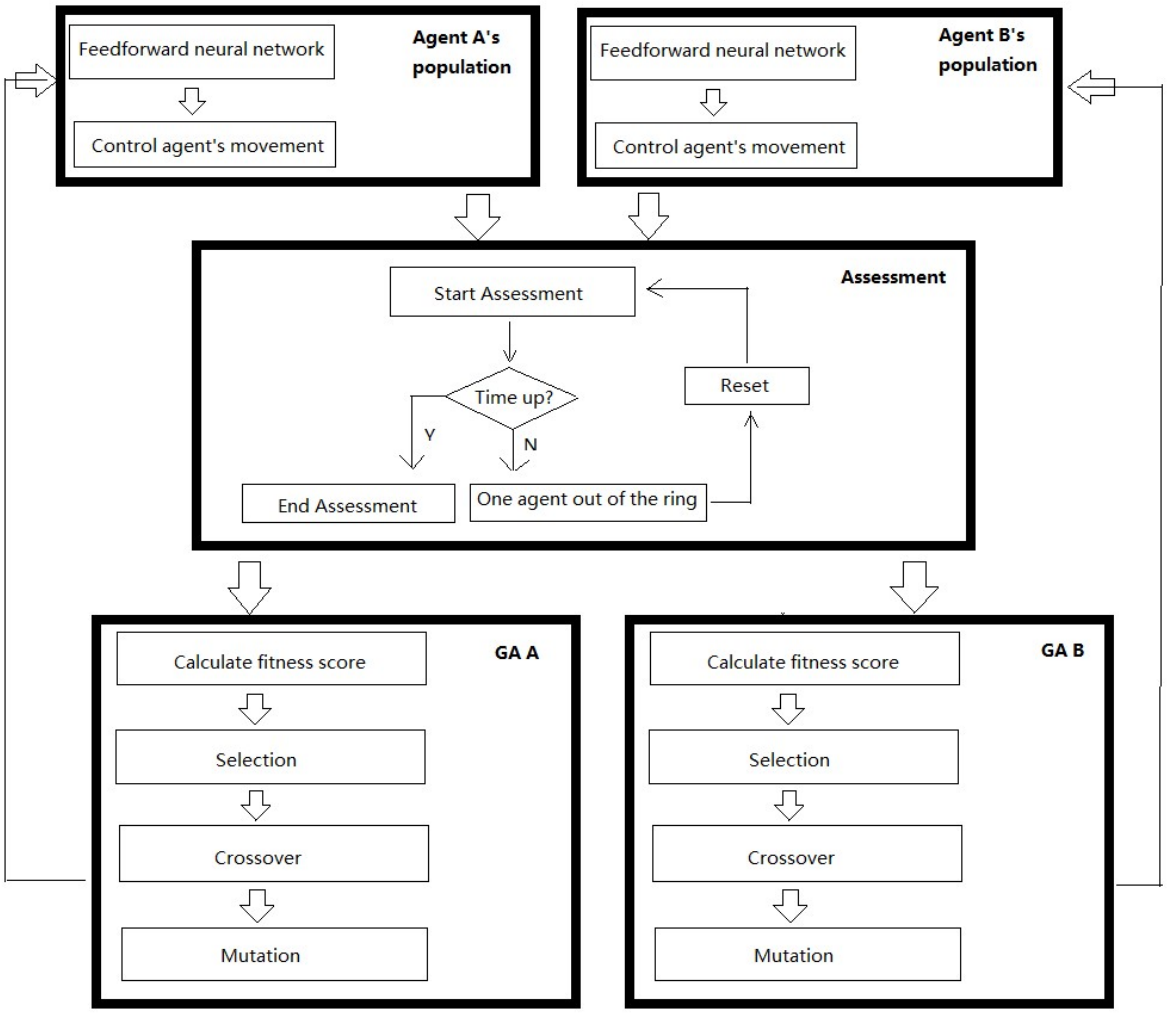

Figure 3.2: The general architecture of GA control system

During the GA, the fitness score for the individual in each agents' populations will save in a txt file. After the GA, the scripts for plotting will plot the best fitness and average fitness line in each generation according to the fitness score. This scripts wouldn't affect the GA result, but help to evaluate and analyze the system.

Chapter 4
Software Implementation

This project has designed two control system, as mentioned above. Therefore, this chapter is going to discuss the two system implementation, respectively.

4.1 Decision tree based control system implementation

Follow the discussion in Chapter 3. This system is going to implement in two different environments, one is for generating the decision tree; another is to guide the robot's behaviour according to the decision tree output. The next few sections are going to discuss the implementation in detail.

4.1.1 Technologies employed in this system

Because this project is about hight-level control system design, so it's reasonable to use object-oriented programming language.

The decision tree generating progress would use Python, and several libraries, which will be listed in Table 4.1. The file type of this code would be .ipynb, which means this code is able to run in Colab or Jupyter notebook environment. For some of the library may not be pre-installed in the user's local machine, run the code in Colab would be my recommendation.

Table 4.1	Library	Purpose
T 4.1.1	from sklearn.preprocessing import LabelEncoder	For data preprocess, convert string to incremental value for serialization.
T 4.1.2	from sklearn.externals.six import StringIO	Mainly used to read and write data in the memory buffer.
T 4.1.3	from sklearn import tree	The main library for generating the decision. Provide the decision tree model for solving classification and regression problems.
T 4.1.4	import pandas	In order to serialize string type data, pandas data was needed, which is convenient for the serialization work.
T 4.1.5	import pydotplus	Used for the visualization of the decision tree.
T 4.1.6	import JSON	Used for transfer the generated decision tree to the Zumo robot code.

Table 4.1 Libraries and purposes in decision tree based control system

As for the code that is going to compile and run in Zumo robot, it's written in C++, and the software for compile and run is Arduino IDE. The type of the file is .ino, which are able to be

upload to Zumo robot microcontroller. And the libraries and purpose are shown in Table 4.2 below.

Table 4.2	Library	Purpose
T 4.2.1	ArduinoJson	A simple and efficient JSON library for embedded C++. (Blanchon, 2020)
T 4.2.3	Zumo32U4	Zumo 32U4 robot Arduino library for the Arduino IDE that helps interface with the on-board hardware on the Pololu Zumo 32U4 robot. (e.g. line sensor, proximity sensor, motor, LED, LCD, button, accelerometer, etc.)

Table 4.2 Libraries and purposes in decision tree based control system

4.1.2 Create dataset

As mentioned in section 3.2, the idea of creating the dataset to generate the decision tree is to write by hand. Then write and upload a simple code to Zumo robot, which is to display the real-time sensor reading in the LCD screen. Part of the code shown in Figure 4.1.

```
//Display position
lcd.gotoXY(2,0);
//Store the sensor reading, and scale down the value from 0 - 6 to 0 - 3
frontleft = map(proxSensors.countsFrontWithLeftLeds(), 0, 6, 0, 3);
//Display the sensor reading
lcd.print(frontleft);
```

Figure 4.1 Example code for display sensor reading

Experiment and find out the sensor reading corresponding to the distance by using a cup as the opponent. About the proximity, I only experiment the front left one, because the proximity sensor in the Zumo robot is the same. And the result is shown below. (Note: the distance is an approximate value, measured by using the *'Measure'* app in iPad. There is a deviation of two or three centimetres).

Proximity sensor reading	Distance (Near end)	Distance (Remote end)
0	N/A (Too close to measure)	More than 60 cm
1	N/A (Too close to measure)	37cm – 60 cm
2	1cm – 5 cm	20 cm – 37 cm
3	6cm – 19cm	6 cm – 19 cm

Table 4.3 Reading and distance comparison table

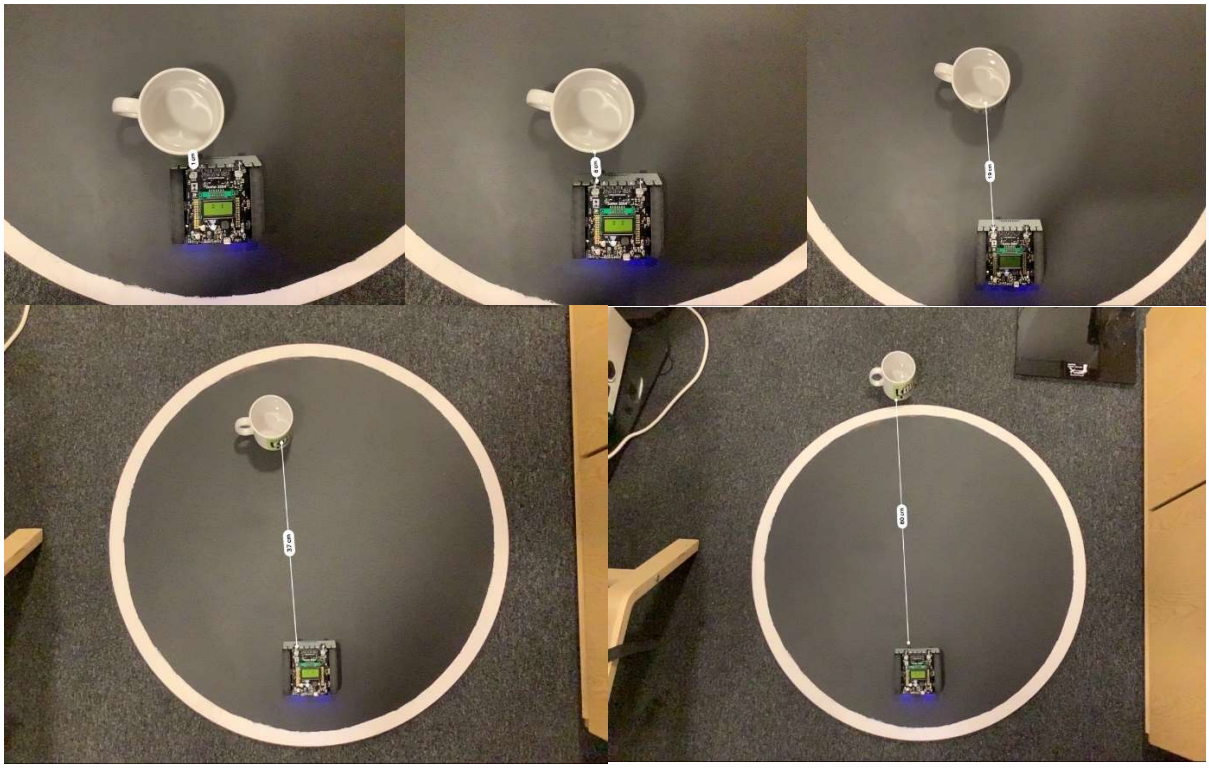

Figure 4.2 Experiment reading and distance in real world

And do the same for the line sensor, which is used to detect the black ground. After the experiment, the threshold is approximate 1100. If any line sensor's reading is large than 1100, then it's out of the ring or touching the white border. Else, it's still inside the ring.

After knowing the reading and distance, the next work would be assigned the label. For example, if both proximity sensor reading is 2 or 3, which means opponents detected, and the label would be *object seeing*.

If the reading is 0 or 1, there are two possibilities. One is the robot has hit, or extremely close with the opponents, Another situation is the robot is extremely far from the opponents or no opponents detected. Thus, only two proximity sensor reading could be misleading, so the accelerometer is introduced in this project. One represents detecting speed change, which means hit the opponents. 0 means everything is normal. Therefore, with an accelerometer, it'll be more transparent for the generated decision tree to make a decision.

Part of the dataset is shown in Figure 4.3. The whole dataset would be available in my repository.

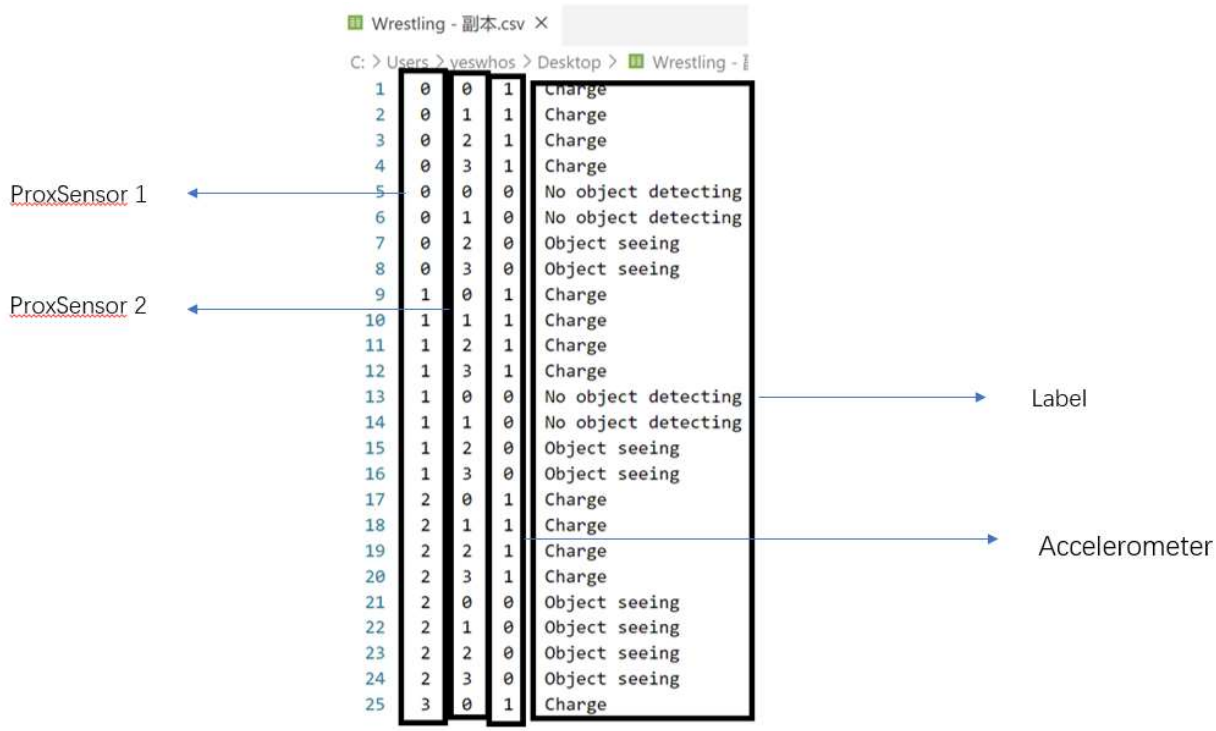

Figure 4.3 Part of the dataset for decision tree

4.1.3 Generate decision tree

Step one: Fetch the data.

Save the data file in Google drive, then use Colab fetch the data.

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3 filepath = "drive/My Drive/Wrestling.txt"
```

```
Go to this URL in a browser: https://accounts.g

Enter your authorization code:
 · · · · · · · · · · ·
Mounted at /content/drive
```

Figure 4.4 Code of fetch data in Colab

Step two: Create a dictionary called *mode_dict* to store the data.

Load the data file and process. Create the feature label array as *modeLabels*. In *mode_dict* the key is the feature label, value is the rest data in each line, except the label.

```
with open(filepath, 'r') as fr:                                    #Load file
    mode = [inst.strip().split('\t') for inst in fr.readlines()]   #Process file
modeLabels = ['proxSensor1', 'proxSensor', 'accelerometer']        #feature
mode_list = []                                                     #temp list to store data
mode_dict = {}                                                     #dictionary to store the data
for each_label in modeLabels:                                      #extract data to generate dictionary
    for each in mode:
        mode_list.append(each[modeLabels.index(each_label)])
    mode_dict[each_label] = mode_list
    mode_list = []
```

Figure 4.5 Code of generate dictionary to store data

Step three: Use *mode_dict* to generate DataFarme type of data. And the we may check it by print it.

```
mode_pd = pd.DataFrame(mode_dict)                                  #generate pandas.DataFrame
print(mode_pd)                                                     #print pandas.DataFrame
```

	proxSensor1	proxSensor	accelerometer
0	0	0	1
1	0	1	1
2	0	2	1
3	0	3	1
4	0	0	0
5	0	1	0
6	0	2	0
7	0	3	0
8	1	0	1
9	1	1	1
10	1	2	1
11	1	3	1
12	1	0	0
13	1	1	0
14	1	2	0
15	1	3	0
16	2	0	1

Figure 4.6 Generated DataFrame data and print the result

Step four (optional): Serialization.

Create LabelEncoder object helps to do the serialization, which means transfer the non-numeric data into number data, in case of the data is not number. However, in this case, all the data is number. Thus, this step is not necessary, but it's feasible, even if some of the data need to be changed to non-numeric data.

```
le = LabelEncoder()                                                #create LabelEncoder() object
for col in mode_pd.columns:                                        #Serialization
    mode_pd[col] = le.fit_transform(mode_pd[col])
print(mode_pd)
```

Figure 4.7 Code of data serialization

Step five: Generate decision tree by using sklearn.

Sklearn.tree provide decision model, which makes it's easy to generate, only by selecting the criterion and put the data into the decision tree. Once created, it's easy to test the predicted result as well.

Figure 4.8 shows the generate code.

```
clf = tree.DecisionTreeClassifier(criterion = 'gini')                    #create DecisionTreeClassifier()
clf = clf.fit(mode_pd.values.tolist(), mode_target)                    #apply data into decision tree
```

Figure 4.8 Generate decision tree

Figure 4.9 shows the detail information about the generated decision tree.

```
1 print(clf)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Figure 4.9 Detail of generated decision tree

Figure 4.10 shows the example of predicting the result by using the generated decision tree.

```
1 print(clf.predict([[3, 0, 1]]))
```

```
['Charge']
```

Figure 4.10 Test one predicted result

Step six: Visualize and save the decision tree.

By using sklearn, one line of code can do the visualization. And by using pydotplus, save the visualized decision into a PDF file.

```
tree.export_graphviz(clf, out_file = dot_data,                    #Visualized decision tree
                     feature_names = mode_pd.keys(),
                     class_names = clf.classes_,
                     filled=True, rounded=True,
                     special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("drive/My Drive/tree.pdf")                    #Save decision tree into PDF file
```

Figure 4.11 Visualization and save the decision tree

The visualizaiton decision and detail information in each tree node are shown in Figure 4.12.

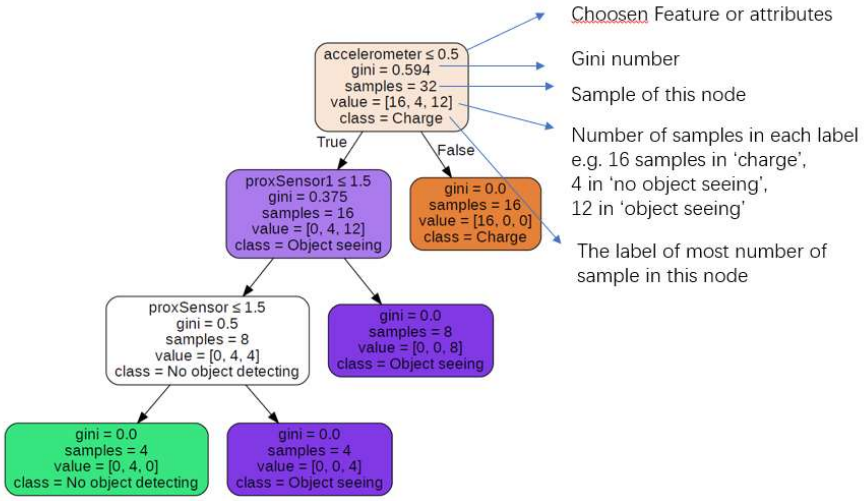

Figure 4.12 Visualized decision tree

Step seven: Transfer the decision tree into JSON format.

Also by one line of code, it's easy to transfer the decision tree into JSON format. And then copy this JSON string, paste it into the Arduino code.

Figure 4.13 shows the code of encoding the generated decision tree into the JSON format. And Figure 4.14 shows the JSON format decision tree.

```
1 import  json
2
3 b  =  json.dumps(clf)
4 print(b)
```

Figure 4.13 Transfer the decision tree inot JSON format

```
{"Contact": {"0": {"Front left Sensor Left": {"0": {"Front Sensor Right": {"0": "No object seeing", "1": "No object seeing", "2":
<class 'str'>
```

Figure 4.14 Part of the JSON decision tree

4.1.4 Code in Zumo robot

First, define some useful sensors, including line sensors, proximity sensors and accelerometer. For the accelerometer, it's controlled by LSM303 chip. Thus, initialize the LSM303 is equivalent to the accelerometer. The code for defining and initializing these sensors are shown in Figure 4.15.

```
void setup()
{
  lineSensors.initFiveSensors();
  proxSensors.initFrontSensor();
  // Initialize the Wire library and join the I2C bus as a master
  Wire.begin();
  // Initialize LSM303
  lsm303.init();
  lsm303.enable();

#ifdef LOG_SERIAL
  lsm303.getLogHeader();
#endif

  randomSeed((unsigned int) millis());
  ledYellow(1);
  //buzzer.playMode(PLAY_AUTOMATIC);
  waitForButtonAndCountDown(false);

}
```

Figure 4.15 Initialize essential sensors

And the *waitForButtonAndCountDown* function is written to satisfy the sumo rule. When the reference starts the competition, press button A. And it'll go for this function. Zumo robot then delays for 5 seconds, while the yellow led blink every second. After 5 seconds, the program goes for the rest of the program.

Then Allocate variables with different sensor values. Put the JSON format decision tree into the Arduino code. And input sensor values into the decision tree, get the output. The code is shown below.

```
const size_t capacity = 0 + 390;
DynamicJsonDocument doc(capacity);
char json[] = "{\"Contact\": [{\"0\": [{\"0\": [{\"0\": \"No object seeing\", \"1\": \"No object seeing\", \"
DeserializationError error = deserializeJson(doc, json);
if (error) {
  Serial.print(F("deserializeJson() failed: "));
  Serial.println(error.c_str());
  return;
}
const char *value = doc["Contact"][0]["0"][reading_a][(String) reading_a][reading_b][(String) reading_b];
```

Figure 4.16 Code for JSON format decision tree.

According to the decision tree output value, assign its control mode. For example, if the label is 'No object seeing', then execute *search()* mode. If 'Object detected', then execute other modes. The code for object detected in Figure 4.17.

```
else{
  Serial.println("Object");
  if (reading_a < reading_b)
  {
    // The right value is greater, so the object is probably
    // closer to the robot's right LEDs, which means the robot
    // is not facing it directly.  Turn to the right to try to
    // make it more even.
    turn(RIGHT, true);
  }
  else if (reading_a > reading_b)
  {
    // The left value is greater, so turn to the left.
    turn(LEFT, true);
  }
  else if(reading_a == reading_b){
    goCharge();
  }
}
```

Figure 4.17 'Object detected' corresponding control mode

And one of the control modes, like the *turn()* function, are shown below. It gets the value of direction (e.g. RIGHT or LEFT), and randomize. If it's true, then the speed of the motor will remain uncertain when turning, which increased randomness of the robot's movement.

```
void turn(char direction, bool randomize)
{
  static unsigned int duration_increment = TURN_DURATION / 4;

  // motors.setSpeeds(0,0);
  // delay(STOP_DURATION);
  motors.setSpeeds(-REVERSE_SPEED, -REVERSE_SPEED);
  delay(REVERSE_DURATION);
  motors.setSpeeds(TURN_SPEED * direction, -TURN_SPEED * direction);
  delay(randomize ? TURN_DURATION + (random(8) - 2) * duration_increment : TURN_DURATION);
  int speed = getForwardSpeed();
  motors.setSpeeds(speed, speed);
  last_turn_time = millis();
}
```

Figure 4.18 *turn()* function in control mode

And the primary condition is to keep the robot remain inside the ring. So at the beginning of the loop function, it will first consider the value of the line sensor, then control the robot behaviour if necessary. The code for this pupose is shown in Figure 4.19.

```
lineSensors.read(lineSensorValues);
lineSensor1 = lineSensorValues[0];
lineSensor2 = lineSensorValues[2];
lineSensor3 = lineSensorValues[4];
//Serial.println(lineSensor1);
warnRight = (lineSensor1 <= sensorThreshold) && (lineSensor3 >= sensorThreshold);
warnLeft = (lineSensor1 >= sensorThreshold) && (lineSensor3 <= sensorThreshold);
warnCenter = (lineSensor1 <= sensorThreshold) && (lineSensor3 <= sensorThreshold);
if(warnRight){
  //Serial.println("右侧");
  turn(RIGHT, true);
}
else if(warnLeft){
  //Serial.println("左侧");
  turn(LEFT, true);
}
else if(warnCenter){
  //Serial.println("中间");
  turnAround();
}
```

Figure 4.19 Code for not running out of the ring

4.2 GA based control system implementation

4.2.1 Technologies employed in this system

Same reason as the decision tree based control system, this is going to use object-oriented programming language. The whole GA based control system is going to implemented in the simulation environment, which is in the BEAST. And the BEAST is using C++, so this project in GA is going to use C++ as well.

The libraries are going to use in GA are in the table below. The libraries are quite different from the previous control system. Because it's based on simulation, most of the libraries are the built-in head file in the simulation tool.

Table 4.4	Library	Purpose
T 4.4.1	C++ STL (Standard Template Library)	Basic function and type of data in C++
T 4.4.2	animat.h (BEAST built-in library)	Including Animat class and associated constants, which the agent shall extend this class
T 4.4.3	sensor.h, sensorbased.h (BEAST built-in library)	Include some useful functions which set up the most common types of sensor
T 4.4.4	beast.h (BEAST built-in library)	The main include file for the simulation environment

T 4.4.5	simulation.h (BEAST built-in library)	The framework providing the facilities for implementing a range of different types of simulation
T 4.4.6	geneticalgorithm.h (BEAST built-in library)	Contains GA process function, including selection, crossover, setting population group, etc.
T 4.4.7	worldobject.h (BEAST built-in library)	Provide many overridable methods which ensure any kind of thing can be represented in the simulation environment, including the ring in sumo league
T 4.4.8	import numpy	Store fitness score data temporary. Used for data processing
T 4.4.9	Import matplotlib	Used for printouts of the plots of the fitness score.

Table 4.4 Libraries and purposes in GA based control system

4.2.2 Create simulated environment and Zumo robot.

The sumo league environment is made of a big black ring and two wrestling robot. First all, create three classes, represent the ring, one wrestling robot and another wrestling robot, respectively.

The ring class would extend the *WorldObject* class in *worldobject.h* file. Then set its size, location, and colour. The code for defining these features are shown in Figure 4.20.

```
class Circle : public WorldObject
{
public:
    Circle()
    {

        This.Radius = 200.0f;
        This.SetColour(ColourPalette[COLOUR_BLACK]);
        //This.InitRandom = true;
        SetLocation(400, 300);
        This.Location = GetLocation();
    }
    virtual ~Circle(){}

};
```

Figure 4.20 Code for the ring object.

Then two agents class, *ZumoKing* and *ZumoQueen*, both initialize only with its essential attributes, including its sensors, location, orientation, size, etc. And these two agents class extend the *EvoFFNAnimat* class, which is in the *animat*.h file. The initialization code for agent are shown in Figure 4.21.

```
class ZumoKing : public EvoFFNAnimat
{
public:

    ZumoKing():lines(0)
    {
        This.Add("lineSensor", ProximitySensor<circleCenter>(2 * PI, 200.0f, -PI));
        //Sensors["lineSensor"] -> SetMatchingFunction(new MatchExact<circleCenter>);
        This.Add("left", ProximitySensor<ZumoQueen>(PI/4, 200.0, -PI/20));
        This.Add("right", ProximitySensor<ZumoQueen>(PI/4, 200.0, PI/20));
        This.SetStartOrientation(PI);
        //This.SetInitRandom(true); // Start in random locations
        This.InitRandom = false;
        This.SetStartLocation(Vector2D(500, 300));
        This.Radius = 25.0;
        SetMinSpeed(0.0);
        This.InitFFN(5);

    }
```

Figure 4.21 Code for the agent

Then, Figure 4.22 shows the implementation of these classes in the *ShrewSimulation* class. Everything that appear in the simulation process should be implemented in *ShrewSimulation* class, which will be discussed later.

```
This.Add("ZumoKing", This.popKing);
This.Add("ZumoQueen", This.popQueen);
This.Add("Circle", This.theCircle);
This.Add("circleCenter", This.theCircleCenter);
```

Figure 4.22 Code for implement classes into simulation class

After having these classes, the ring and robot in the simulated environment must have the same size proportion as the proportion in the real world. In real-world, the diameter of the ring is approximately 72 cm, and the robot length and width are approximate 9 cm, the size proportion is 1:8. Therefore, set the radius of the ring is 200 in the simulation environment. The radius of the robot is 25 then. (In the simulation, the agent is a round shape, while it's a square shape in the real world. The area of the robot will be considered, but this shape difference will not discuss in this project).

The real world and simulation environment comparison shown below.

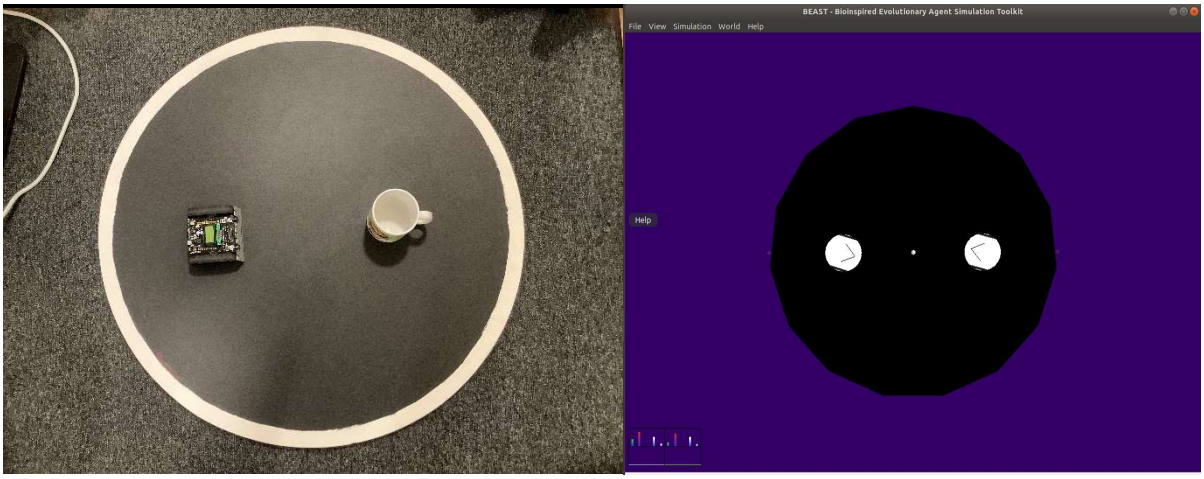

Figure 4.23 Comparison of real world and simulation

Then define the same sensor function as the sensor on the Zumo robot. There are two kinds of sensor, proximity sensor and line sensor.

The proximity sensor is a built-in function. All the thing that needs to be done is to add the sensor in the robot, define it's scope, range and orientation. And these three inputs should be set according to the real world as well. With the experiment in the real world, the maximum range is about 60 cm, and the orientation is facing forward. As the measures in Figure 4.24, get the length of the isosceles triangle. Then use the trigonometric function and inverse trigonometric function to calculate the angle, which is the sensor range. And the result is approximately 45 degree, which is $\frac{\Pi}{4}$.

The line sensor is not a built-in function. However, some changes can make the built-in proximity sensor has the same function as the line sensor. First, create a small object in the centre of the ring. It's a dot with no collision volume. Then add one proximity sensor that detects nothing but this little dot. Keep returning the distance with the small dot, and if the distance exceeds the radius of the ring, then this agent is out of the ring.

From Figure 4.24, it's quite clear that a white dot in the middle of the ring, which is useful for the line sensor. The white transparent thing in the simulation is the detect range of all the sensors. From the code in Figure 4.22, there are one line sensor and two proximity sensors added in the agent, which simulated the Zumo robot. And the ring is also well simulated in BEAST.

The agent is controlled by FNN, as mentioned in Chapter 3. And both agent class is extended from EvoFFNAnimat class, so the control function is built-in. From Figure 4.21, it's also initialised the FeedForwardNet (FFN) with five hidden nodes. And the number of inputs and outputs is decided by the number of sensors and the number of motors. Thus, the output of FFN controls the two motors and the movement of the agents. As for the control function, it's not necessary to override it.

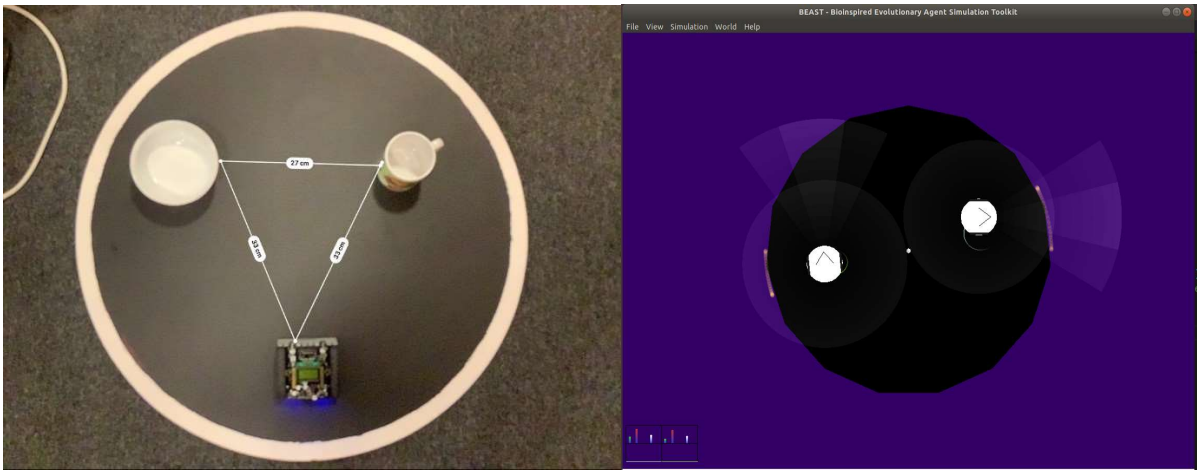

Figure 4.24 The sensor range comparison in real world and simulation

4.2.3 During the assessment

In each generation, the number of assessment and the time of each evaluation can be set. Figure 4.25 shows that the time of each assessment will set as 20 seconds, and five assessments in each generation. The number of population for each agent class in each generation is five. Each assessment will be the one versus one competition, and in the next assessment will be another two agents play the game. The simulation class will make sure that the following two agents (one from *ZumoKing*, another from *ZumoQueen*) are picked. And all the necessary things in the simulation that metioned above are implemented in the *ShrewSimulation* class. And the code for it shown in Figure 4.25.

```
popKing(5, gaKing),
popQueen(5, gaQueen),
theCircle(1),
theCircleCenter(1)
{

    popKing.SetTeamSize(1);
    popQueen.SetTeamSize(1);
    SetAssessments(5);
    This.Add("ZumoKing", This.popKing);
    This.Add("ZumoQueen", This.popQueen);

    This.Add("Circle", This.theCircle);
    This.Add("circleCenter", This.theCircleCenter);
    This.SetTimeSteps(500);
```

Figure 4.25 *ShrewSimulation* Code for population and assesssment set

During the assessment, if one robot is fall out of the ring, then the code in Figure 4.26 will reset everything, including the position, speed, etc.

To be clear, the rule of *'Upon pressing the start buttons, each robot must not move at all for five seconds'* is not considered in this GA based control system implementation. Which means, the robot wouldn't wait for five seconds at the beginning of each assessment. It will save much time in each assessment.

```
void reLocate(){
    This.SetLocation(Vector2D(300, 300));
    This.SetOrientation(0.0);
}

void OnCollision(WorldObject* obj){
    This.lines = This.Sensors["lineSensor"] ->GetOutput();
    if(This.lines < 0.015 && This.lines != 0){
        inRangeB = false;
        tempB = false;
    }
    if (inRangeB == false){
        counterB++;
        reLocate();
        inRangeB = true;
    }else if(temp == false){
        reLocate();
        temp = true;
        inRangeB = true;
    }
    //setEnermy(enermy);
    FFNAnimat::OnCollision(obj);
}
```

Figure 4.26 Code for reset

As for the *inRangeA* or *inRangeB in* Figure 4.26, these two variable are boolean type. And *counterA* and *counterB* are integers for count how many times each agent has run out of the ring separately. If *ZumoQueen* is out of the ring, then *inRangeB* is set to false, *counterB* plus 1, then reset *inRangeB* to true. And do the same for *ZumoKing*.

By the end of each assessment comes to the GA.

4.2.4 GA process

Here comes the fitness score, selection, mutation and crossover. Most of the function has already been implemented in the *geneticalgorithm.h* file.

As for fitness score, this built-in function needs to be overridden.
The *ZumoKing* and *ZumoQueen* in this project have different fitness function. As for *ZumoKing*, the code of fitness is in Figure 4.27. The score is related to counterA and counterB, which are the number of times it runs out of the ring and the number of times its opponents run out of the ring. For ZumoKing, it's reasonable that fitness score has a positive correlation with counterA, while has a negative correlation with counterB.

According to the rules, *'However, if the robot does not move for a long time after the start command. It will be punished or judged for losing the round'* and *'The robots fail to touch each other for a while, the reference may choose to restart a round'*. The fitness would encourage the robot's movement rather than stand still by introducing *DistanceTravelled*. Thus, *DistanceTravelled* would also have a positive correlation with fitness score, but wouldn't affect the score as much as *counterA* and *counterB.*

Because there are three selection choices, including roulette wheel selection, which is the probability selection according to fitness score, thus, the fitness score should be positive rather than a negative number.

In Figure 4.27, the ZumoKing would reward five times score for each time the opponent out of the ring. And it will be punished less for out of bounds itself. Because the *DistanceTravelled* is a large number and we don't want it to affect too much to the score, it's reasonable to assign it with a small weight.

In some exceptional cases, like if the score less than zero, then it's seen as zero. If both agents haven't run out of the ring during the assessment, which means no progress has made in the match. Then we recognise it as passive behaviour in the game according to the rules. Thus, both agents will get zero for their fitness score, regardless of their travelled distance score.

```
virtual float GetFitness()const
{
    float score = 0.0;

    // score = 2 * counterC - counterA + 0.0001 * This.DistanceTravelled;
    // if (counterC == 0){
    //  score = 0.0;
    // }
    score = 5 * counterB - 0.5 * counterA + 0.0005 * This.DistanceTravelled;
    //cout << This.DistanceTravelled << endl;
    if ((counterA == 0) && (counterB == 0)){
        score = 0.0;
    }
    if (score < 0){
        score = 0.0;
    }
    // cout << counterC << endl;
    // counterC = 0;
    counterA = 0;

    float a = score;
    ofstream out("/home/beast/COMP5400M-Bio-inspired-Computing/CW1/Demo/ZumoKing.txt", ios::app)
    out << a << endl;
    out.close();
    //cout << score << endl;
    return score;
```

Figure 4.27 Fitness code for ZumoKing

The same fitness rules for *ZumoQueen*, but has slightly difference with *ZumoKing*. In Figure 4.28, the agent would reward for its opponents out of bounds, while it will be punished five times more for out of the bounds itself. And the travel distance rewards the same as *ZumoKing*.

```
float score = 0.0;

score = counterA - 5 * counterB + 0.0001 * This.DistanceTravelled;
//score = counterA - 5 * counterB;
if ((counterA == 0) && (counterB == 0)){
    score = 0.0;
}
if (score < 0){
    score = 0.0;
}
counterA = 0;
float b = score;
ofstream out("/home/beast/COMP5400M-Bio-inspired-Computing/CW1/Demo/ZumoQueen.txt", ios::app);
out << b << endl;
out.close();
//cout << score << endl;
return score;
```

Figure 4.28 Fitness code for ZumoQueen

Also, the fitness scores of the two agents will be stored in files separately, which may provide data for drawing the plot.

Although the only slight difference is made for these two agents' fitness, the expected behaviours can be very different. *ZumoKing* is expected to be more aggressive as well as offensive, and it will care more about the attack and care less about keep itself inside the ring. In the opposite, *ZumoQueen* is expected to be more defensive, and it'll avoid running out of the ring and wait for the opponent to make mistakes. These two strategies will lead these two agents into different evolutionary process and different behaviours.

This is only one way of defining the fitness function, more way of calculating fitness score (with the same variables) will be examined in the next chapter.

For selection function, it's a built-in function; it only needs one line of code to implement it. There are three selection methods, including the roulette wheel, rank, tournament selection. The selection code in Figure 4.29 is based on rank proportional selection. Other two selections are similar and will test in the next chapter as well.

```
This.gaKing.SetSelection(GA_RANK);
This.gaKing.SetParameter(GA_RANK_SPRESSURE, 2.0);
This.gaQueen.SetSelection(GA_RANK);
This.gaQueen.SetParameter(GA_RANK_SPRESSURE, 2.0);
```

Figure 4.29 Code for selection

For crossover and mutation function, it's also a built-in function. In Figure 4.30, it means the crossover probability is 0.7, and the mutation probability is 0.05.

```
gaKing(0.7f, 0.05f),
gaQueen(0.7f, 0.05f),
```

Figure 4.30 Code for crossover and mutation

4.2.5 Plot the fitness score

As the code in Figure 4.27 and Figure 4.28, the fitness scores for both agents are stored in two files separately. I write a Python code for plotting it for better observe and analysis.

First, read the data stored file, then calculate the best and average fitness score in each generation. Then save the best and average score in the separate list, called *best* and *ave*. Plot these two lists. And set detail of the plot, like the line style, colour, title, etc. Finally, save the plot image.

The code for plotting is not the main code, and it's more like the supporting role of the project. Thus, I won't write details about this section but only describe it concisely. And the codes are available in the repository as well.

4.2.6 Traditionally controlled robot simulation

One special agent would be introduced in this project. It's a traditionally controlled agent, extended from *animat.h* class, called *ZumoLord*. *ZumoLord* has the same structure as the other two agents, including the same sensor, size, etc. The only difference is its control method is the traditional control. Which means it born with strategy, and it's wont change it's behaviour after generations. Thus, it's not controlled by FNN, and no GA process. *ZumoLord* control code is in Figure 4.31.

```
if(This.lines < 0.02 && This.lines != 0){
    This.Controls["left"] = -lines;
    This.Controls["right"] = -lines;
}
double ProxRight = This.Sensors["right"] -> GetOutput();
double ProxLeft = This.Sensors["left"] -> GetOutput();
if((ProxLeft == 0) && (ProxRight == 0)){
    This.Controls["left"] = -0.8;
    This.Controls["right"] = 0.5;
}else{
    // This.Controls["left"] =  1.5 * ProxLeft;
    // This.Controls["right"] =  1.5 * ProxRight;
    This.Controls["left"] =  0.7;
    This.Controls["right"] =  0.7;
}
```

Figure 4.31 Code for *ZumoLord* tradition control

The strategy for *ZumoLord* is to follow it's opponents according to its proximity sensor. When the opponent is in front of *ZumoLord,* it will run towards it with the maximum speed. It's strategy sounds quite similar to the decision tree based control system strategy. The only difference is when the opponent is not detected, *ZumoLord* will turn round until it finds its opponent.

The purpose of designing this agent is to find out: if the GA controlled robot is better than the traditionally controlled robot or the trained robot without evolution. And the experiment result will be shown and explained in the next chapter.

Chapter 5
Software Testing and Evaluation

5.1 Decision tree based control system testing

The decision tree generating progress is using the CRAT algorithm. It will calculate the Gini index of each feature. Smaller of Gini index, more purity of the feature.

As the visualised decision tree in Figure 4.11, it's pointed out the details of the result. In each root node, the Gini index is 0, which means the sample has been well divided according to features. Every root node is completed purity.

After generated and transferred the decision tree into the Zumo robot program, compile and upload the program. Figure 5.1 shows that the code only takes 53% of the program storage space, and global variables use 25% of dynamic memory. Even with limited storage and memory, the program still leaves much space, which leaves the space for future work. Like improve the control system by a more complicated decision tree, or the control mode can be more agile and hard to predict it's next move.

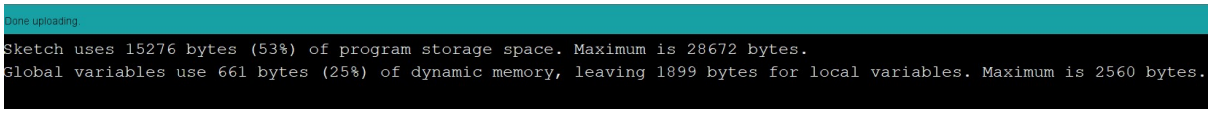

Figure 5.1 Storage and memory occupation for decision tree control system

As for its behaviours or control system judgement, it's hard to be objective. The only fair way of judging the robot's control system that I can think of is according to the number of winning matches and total matches. Suppose one robot winning the most of matches (assuming all of the participant robots has the same hardware and structure conditions) in the league. In that case, it definitely has the best control system. However, as mentioned earlier, it's hard to find the opponents or attend the sumo league during this COVID-19 global epidemic period. Thus, the best way that I can think of is using a cup controlled by hand as it's an imaginary enemy.

The testing experiments are from simple to complicated. The first test put the cup inside the Zumo robot's sight, and the cup remains still (no hand control). This experiment shows that the Zumo robot can easily locate the cup and trying to push it out of the ring. Due to no sufficient contact surface, the Zumo robot fails to push it out of the ring even with the maximum horsepower. But it'll work if the opponent is the same structured Zumo robot rather than a cup.

The second is to put the none controlled cup in the left or right-hand side of the robot, respectively. And this experiment is to make sure that the robot wouldn't see the cup in the first place. The robot will do the search mode first, and then it will hit the boundary of the ring. Turn back, and continue to search the cup. Once detected, trying to push it out of the

ring. Both experiments results are as expected. The Zumo robot will quickly find the cup and push it while making sure it wouldn't run out of the ring itself.

The third test is using the hand-controlled cup to compete with the Zumo robot. Because the Zumo motor is far too weak compared to human strength, I can defeat it with one finger. But to be fair, I have to keep my strength, trying to move the cup at the same speed as the robot. The robot doing quite well, it continues to collide with the cup. Since the Zumo robot is trying it's best to push the cup out of the ring, I let it win.

During this test, some unexpected behaviour can be observed. For example, the robot will charge to a direction where there is no opponent. That is because the test site, which is my accommodation is not big enough, the robot will effect by another object, such as the wardrobe. Thus, in open ground, where no other non-relevant object, this problem should disappear.

As the saying goes, talk is cheap, show me the code. Thus, not only the code is provided in the repository, but also a video for decision tree based control system experiment is available. This video including the three experiments, which are mentioned above. The link for the video is https://youtu.be/RahYl37PvtM.

5.2 GA based control system testing

In the simulation environment, the agents' behaviour and fitness score can be observed during the generations. This section will discuss according to these.

And there are four experiments done in this project, including three different selection methods and one special case. All of these simulation experiments are recorded and edited in the video, the agents' behaviours changes and the fitness score changes are visible in the video (https://youtu.be/Me6_-5wUBW4) as well.

5.2.1 Tournament selection

The robot's movement in the first few generations are random and make no sense. The robot moves around with a different radius, and it will hit the opponent if it's lucky. It's just an initial start, so execute 'High Speed Mode' in BEAST to pass the few meanless generations.

In the 18th generation, the agents on the right-hand side, which is the ZumoKing. One of the agent in ZumoKing population seems is able to locate the opponent and trying to push it. Very closely in the same generation, the ZumoQueen also has sign of locating opponents, but it fails to push it every time. In these generations, ZumoKing clearly takes more advantages, while ZumoQueen is the underdog one.

However, the next many generations show that ZumoKing population eliminated the agent with better fitness score and better behaviour. It fails to locate the opponents and keep running out of the ring itself. And ZumoQueen gains more score because of it. The

ZumoQueen just need to make sure that it won't run out of ring itself, it'll still get the high score.

As for the observed behaviour, this experiment shows the counterexample of co-evolution. If one agent keep making mistakes, or it's cannot out of local optimal. Its opponent will not evolve better strategy as well. As the Chinese Confucian philosopher, Mencius said, one prospers in worries and hardships and perishes in ease and comfort. In this case, both agents wouldn't develop a good strategy.

As for the fitness score, the plot in Figure 5.2 and Figure 5.3 demonstrate the same thing. In the first 50 generations, *ZumoKing* wins most of the matches and get a higher score. After 50 generations, it's going downhill for hundreds of generations, and *ZumoQueen* begins to take advantages.

A few things that need to declare, about the plot, you may see some blue lines in every plot. That's just random and meanless lines, and it doesn't affect the observation and the result, so please ignore it.

About the value on the y-axis of the plot. The fitness is from 0 to 80 in ZumoKing, while it's from 0 to around 8 in ZumoQueen. This difference is because of different fitness score calculation methods, so it's meanless to compare these two values. The most important things are the trend of the best and average score (the average score explain the trend better for most of the time) and the generations in each plot.

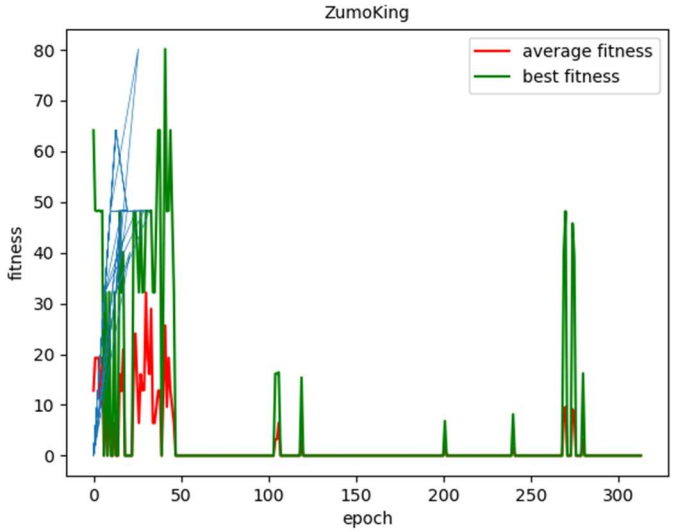

Figure 5.2 Tournament selection fitness score for ZumoKing

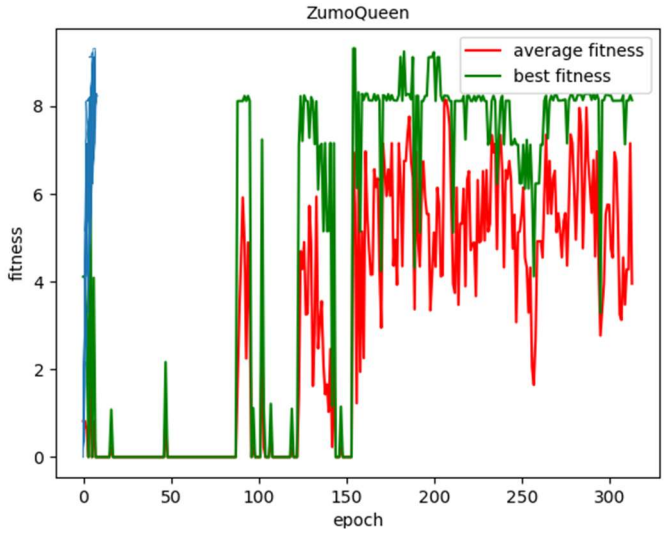

Figure 5.3 Tournament selection fitness score for ZumoQueen

5.2.2 Ranking selection

The first few generations are the same. The robot moves around randomly.

From 70th generation, the ZumoQueen begin to have some intelligence behaviour, which is to locate the opponents, run towards it slowly then trying to push it out of the ring. Then ZumoQueen has dominated the game until around 350th generation.

Around 420th generation, the ZumoKing gradually learned how to avoid the ZumoQueens attack, and let it run out of the ring itself. Thus, the ZumoKing begin to winning more matches.

Around 500th generation, the ZumoQueens are winning the game again. Its behaviour becomes more straight forward. The most important thing is this strategy works well when it engages with its opponents' current strategy.

From the plot in Figure 5.4 and Figure 5.5, it shows the same thing as the behaviour observation. It also shows more information while we skip some of the generations. The ZumoQueen begin to rule the game until around 320th generation. Then the ZumoKing raises until around 540th generations. Then the ZumoQueen rule it again.

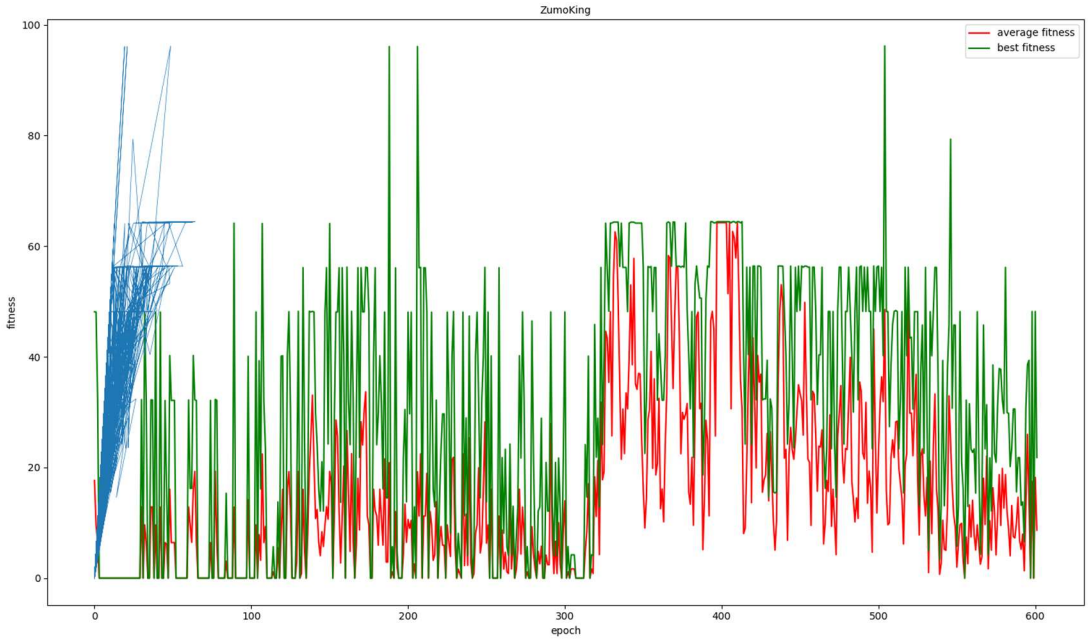

Figure 5.4 Rank selection fitness score for ZumoKing

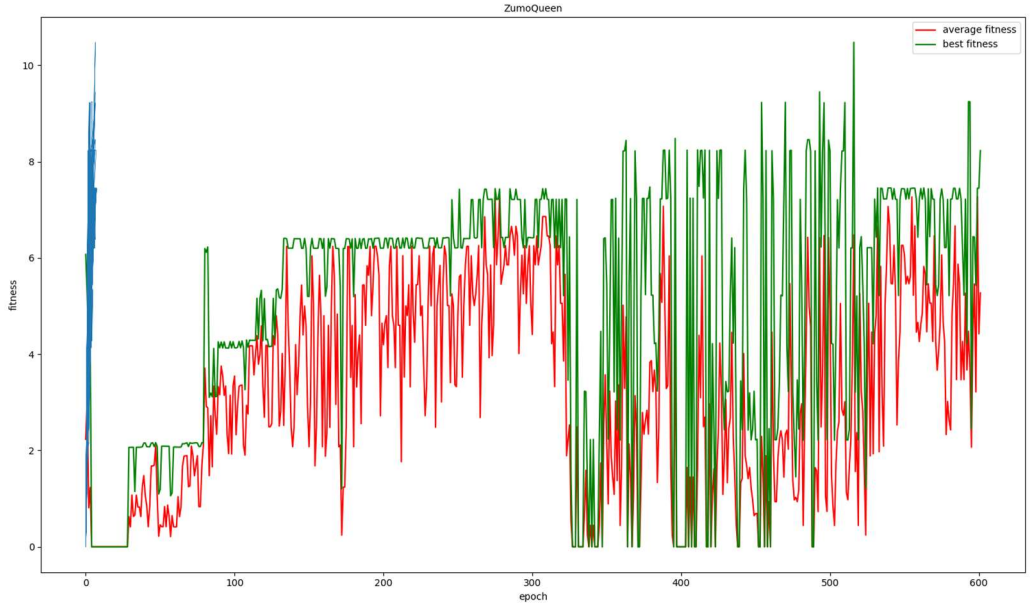

Figure 5.4 Rank selection fitness score for ZumoQueen

This experiment shows the co-evolution of these two agents. In the aspect of behaviour, if one agent develops it's intelligence behaviour first, then it would win more game. As for its opponents, it will evolve its strategy according to the winning one, and after hundreds of generations, it's losing situation will change.

5.2.3 Roulette wheel selection

The first few generations are the same.

When it comes to 56th generation, ZumoKing begins to win more matches, and it also evolved behaviour that can locate the opponent and trying to push it.

From 358th generation, ZumoQueen has developed the strategy of avoiding ZumoKing's attack, and it seems work. ZumoQueen begins to become the one with better behaviour. Especial in the generation of around 440th, ZumoQueen barely lose a match, it's primary strategy is to move to the back of the ZumoKing, then push it out of the ring.

Figure 5.5 and Figure 5.6 shows the plot of fitness changes of two agents. From the plot, it's clear that ZumoKing is dominant the game from 50 to 300. Then it's average fitness score is going down. From 400th generations to 550th generations, it's going to the lowest point. After that, the fitness score for ZumoKing begins to rise again. As for ZumoQueen, it's opposite to ZumoKing, it's fitness line remain low when ZumoKing is high, and it's going up when Zumo is going down.

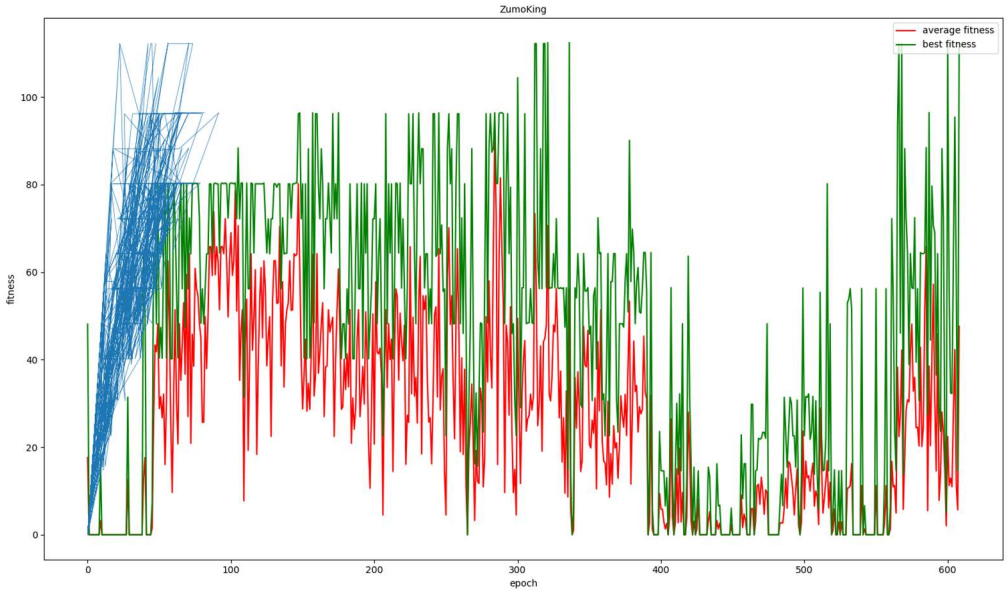

Figure 5.5 Roulette wheel selection fitness score for ZumoKing

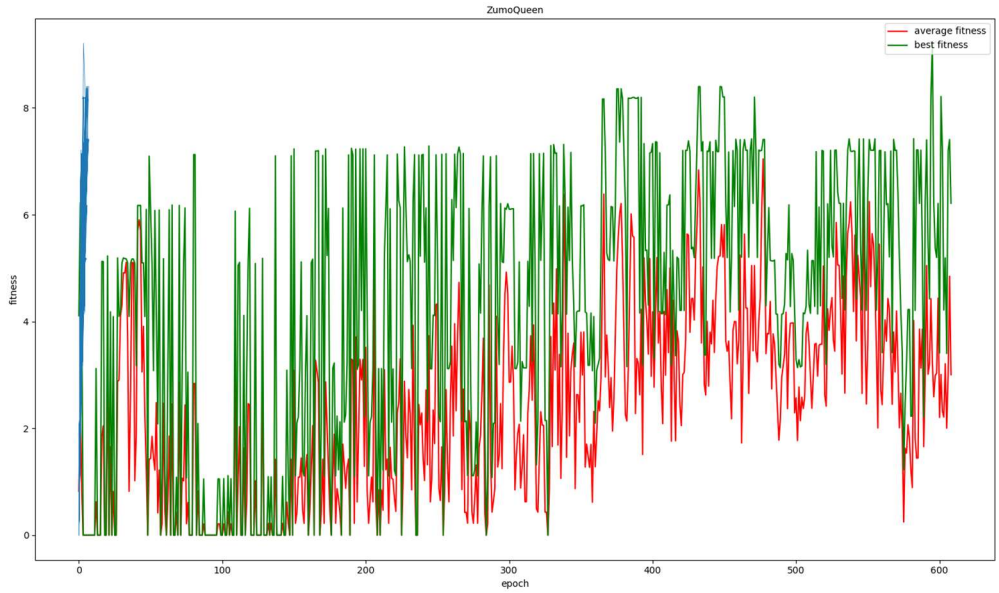

Figure 5.6 Roulette wheel selection fitness score for ZumoQueen

After the experiments with three different selection methods, evaluating selection methods in a subjective point of view, which is by observing it's best-evolved behaviour after the same generations. Roulette wheel selection may have better performance. The evolved agent can locate the opponent and run to the opponents back, then push it. In this way, the agent is out of the opponents' sensor range, while keeping the opponents inside its own sensor range, which is quite an intelligence.

From an objective point of view, the roulette wheel selection evolved intelligence behaviour more early. For both the best fitness and average fitness score, the Ranking and roulette are quite similar. However, tournament selection is not performing well, the ZumoKing score remains nearly zero for many epochs, and it doesn’t evolve competitive agents. While in the other two selection, the co-evolution process is clearly observed. Besides, the competitive relationship between ZumoKing and ZumoQueen always exists. Even one agent is dominating the game, the other agent is not completed defeated. In the loses agent’s population, there are a few agents in the same assessment that can win a few rounds, which can be observed in the fitness plot.

5.2.4 GA agent vs. traditionally controlled robot

This experiment is a little different from the tests above. It would be interesting to find out if the GA agent starts with random behaviour will beat an already implemented strategy agent without GA. The strategy for the agent without GA is to go round when no opponent detected, and speed up when the opponent inside the sensor range.

As the experiments above with different selection, roulette wheel selection has slightly better performance. Its speed of evolving is faster, and the behaviour looks better to compare with the other two selection results after the same generations. Thus, in this experiment, the selection will go for roulette wheel selection.

In the first many generations, the GA agents are defeated for most of the time, just as expected. Then in the 500th generations and later, it develops an excellent strategy to cope with its enemy. And it defeats it’s opponents many times.

Figure 5.7, which is the plot for the fitness score, shows the same thing. The average score line goes up after iterations. And the best score is high and stable as well.

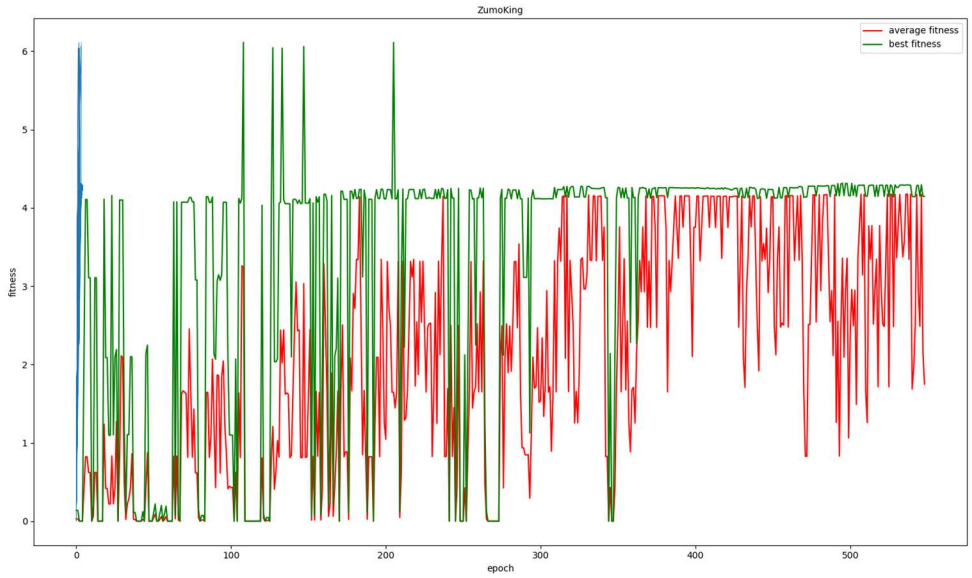

Figure 5.7 GA agent fight with traditionally controlled enemy

The experiment shows that the agent can learn from the failures and improve during the GA process. Experiment with different behaviours, then finally come up with better strategies that can win the match.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

This project implements two kinds of the control system, and they both perform well and meet the project requirements. As the experiment shows, the decision tree control system can execute a strategy without training the robot. While for the GA control system, it would take hundreds of generations to develop intelligence behaviour. Even with the help of the simulation tool, it's still taking me nearly half an hour to train the agents for 600 generations. Thus, unsupervised learning will spend more time and pay more effort to get a satisfying result. And supervised learning usually will find the way very soon.

However, in sumo league, the judgement standard is determined by the robot's behaviour or strategy, rather than it's training or generating time. In the final experiment in Chapter 5, I experimented with the competition GA agents vs traditional controlled robot. The result turns out that the agent will lose most of the matches in the previous generations, but it'll learn to fight back and develop the strategy against it, which is impressive. And it turns out that GA can formulate a targeted plan according to different opponents. As long as we get more information about opponents' behaviour or strategy, it's easy to develop targeted against it.

As for the decision tree, it's would be more stubborn and not suitable for various situations. It's performance also depends more on its developer's strategy. In the aspect of artificial intelligence, it would be more reasonable it's developing its own intelligence strategy just like a human being.

Compare with these two kinds of control systems is pretty like two species' ecological phenomena in nature. The baby sheep can walk and possess the basic cognitive ability, like stay with their population and recognise the dangerous situation, etc., only a few hours after they born. As for the human infant, they can't walk or speak or know things for one or two years after they were born. However, when they grow up, human beings are much more intelligence and athletic than sheep for sure. Thus, a long time for growing or developing or training doesn't mean a bad thing. Taking time as the cost can exchange for better strategy and more benefits.

Furthermore, because this project, as well as my major, is about artificial intelligence, I'd like to discuss the word 'intelligence'. Generalized intelligence does not need to be as advanced as human-level intelligence. The imitation of human intelligence would be too complicated and even impossible at this moment. (Brooks, 1987) Complicated intelligence could be built by multiple simple intelligence, such as the behaviour and strategy in this project.

Brooks also mentions that reaction to the dynamic world is the crucial point of intelligence. In this project, although the ring is fixed, both agents are keeping moving. They collect

information about the environment, respond to the information from the proximity and line sensors. Adjust their actions according to the data to get a good fitness score.

The ultimate purpose of the intelligence of creatures is to reproduce. As for the GA control system, the behaviours of both ZumoKing and ZumoQueen aim to get high fitness scores and then replicate themselves, which meet the characteristic of intelligence as well.

Overall, I believe this project is bio-inspired and intelligence.

6.2 Future Work

Although this project has met the requirements and the control system perform well, it still has many things that can be improved.

Regarding the decision tree, the data can be more complicated by introduced more labels and different responses. And the control mode in the Arduino code can be more accurate and more randomness.

As for GA, more selection methods can experiment, such as truncation selection, Monte Carlo selection, Boltzmann selection, etc. Also, the GA experiment is in the simulation environment; the generated strategy after generations are not implemented in real-world in this project. Through serialization and deserialization could transfer the simulation result to the real Zumo robot. And experiment with the real world will be interesting.

Many other methods are mentioned in section 2.3 can experiment as well. That will take a lot of efforts to implement them. But, as I said before, I always do believe that the best way to evaluate one method or idea is to experiment with it. Thus, it’s worth to be done in future work.

I hope someday when the COVID-19 is gone. I’d have a chance to attend sumo league to test and verify my design. That would be fun.

Besides, the idea of this project shouldn’t only be limited by the sumo league. There are many exciting project or competition out there.

The RoboCup, which is an annual international robotics competition, aims to promote robotics and AI research by offering a public challenge. (Kitano et al., 1998). The RoboCup including many sub-leagues, such as soccer simulation, rescue robot league, soccer league, etc.

The RoboMaster competition is a very prestigious robotics competition in China. This competition is primarily for students in colleges and universities, and it’s attracted 228 teams from more than 160 universities around the world. (Anon, n.d.) The competition is a five vs five matches. If you ever played MOBA (Multiplayer Online Battle Arena) video games such as league of legends or DOTA2, it’s pretty much the same. The participating teams independently develop different types of robots and launch projectiles in the complex

battlefield. At the end of the match, the team with the highest remaining Base HP wins the game. Different robots participating in the competition are shown in Figure 6.1.

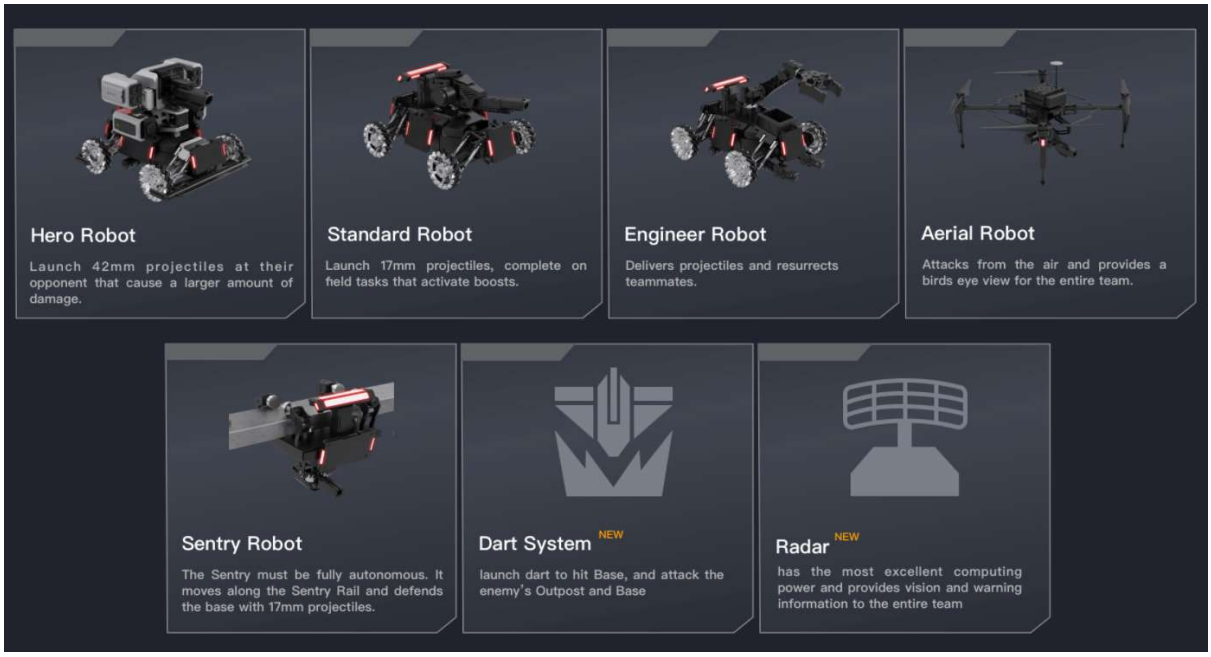

Figure 6.1 Robots demonstration in RoboMaster competition

The GA idea definitely can be applied in this dynamic violently competition environment. Testing every possible strategies and situation, evolve the best strategy after generations. But this is also going to be very challenging. Because the sensor reading, engaged with the enemy and the environment is more complicated than in sumo league. And this is going to having more fun.

It's inspiring to imaging my project idea can apply in so many competitions. But, to do that are going to need more efforts and do more in-depth research about GA, other AI methods, competition rules and so on.

List of References

Anon n.d. Autonomous Sumo Robot Rules - Robot Room. *www.robotroom.com*. [Online]. Available from: https://www.robotroom.com/SumoRules.html#:~:text=Mini%2Dclass%20Sumo%20robots%20may%20be%2010%20centimeters%20(3.93%20inches.

Anon n.d. RoboMaster 机甲大师赛. *RoboMaster 机甲大师赛*. [Online]. [Accessed 23 August 2020b]. Available from: https://www.robomaster.com/en-US/robo/rm.

Bennett, B. 2019. Classical Logic Ⅱ: Formal Systems, Proofs and Semantics, Lecture notes distributed in COMP5450M Knowledge Representation & Reasoning. November, Univeristy of Leeds.

Blanchon, B. 2020. bblanchon/ArduinoJson. *GitHub*. [Online]. [Accessed 16 August 2020]. Available from: https://github.com/bblanchon/ArduinoJson.

Brooks, R. 1987. Intelligence without representation. *Artificial Intelligence*. **139**–**159**(47).

Dogar, M. 2019. Motion and Control. Lecture notes distributed in COMP3611 Intelligent Systems and Robotics. October, University of Leeds.

Erdem, H. 2007. A Practical Fuzzy Logic Controller for Sumo Robot Competition. *Springer-Verlag Berlin Heidelberg 2007*. (LNCS 4815), pp.217–225.

Erdem, H. 2011. Application of Neuro-Fuzzy Controller for Sumo Robot control. *Expert Systems with Applications*. **38**(8), pp.9752–9760.

Erlan, C., Lima, O., Almeida De Araújo, F., Bibiano Da, M., Júnior, S., Edson, A., Filho, R., De Andrade, R., Rabêlo, L., Allisson, T., Da Silva, R., Jose, A. and Alves, O. 2013. *An Enhancement in Conventional Potential Field Using a Fuzzy System for Navigation of a Sumo Robot*.

Fine, T.L. 1996. Fundamentals of Artificial Neural Networks [Book Reviews]. *IEEE Transactions on Information Theory*. **42**(4), p.1322.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J. 2018. An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*. **11**(3–4), pp.219–354.

Jahangirian, A. and Shahrokhi, A. 2011. Aerodynamic shape optimization using efficient evolutionary algorithms and unstructured CFD solver. *Computers & Fluids*. **46**(1), pp.270–276.

Kamps, M. 2020. Genetic Algorithms. Lecture notes distributed in COMP5400M Bio-Inspired Computing. March, University of Leeds.

Kamps, M. 2020. Genetic Algorithms. [Software source code accessed through Minerva]. COMP5400M Bio-inspired Computing. March, University of Leeds.

Kitano, H., Asada, M., Noda, I. and Matsubara, H. 1998. RoboCup: robot world cup. *IEEE Robotics & Automation Magazine*. **5**(3), pp.30–36.

Lehner, J., Dornberger, R., Simić, R. and Hanne, T. 2019. *Optimization of Multi-Robot Sumo Fight Simulation by a Genetic Algorithm to Identify Dominant Robot Capabilities*.

Leung, F.H.F., Lam, H.K., Ling, S.H. and Tam, P.K.S. 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*. **14**(1), pp.79–88.

Lipson, H. and Pollack, J.B. 2000. Automatic design and manufacture of robotic lifeforms. *Nature*. **406**(6799), pp.974–978.

Ma, X.-J. and He, Y.-Y. 1998. Analysis and Design of Fuzzy Controller and Fuzzy Observer. *IEEE TRANSACTIONS ON FUZZY SYSTEMS*. **6**(1), p.41.

Utgoff, P., Berkman, N. and Clouse, J. 1997. Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*. **29**(5–44 (1997)), pp.5–44.

Whitley, D. 1994. A genetic algorithm tutorial. *Statistics and Computing*. **4**(2).

Wikipedia Contributors 2019. Decision tree learning. *Wikipedia*. [Online]. Available from: https://en.wikipedia.org/wiki/Decision_tree_learning.

Wilson, B., Author, S., Germann, T. and Al-Olimat, K. 2016. *Sumo Robot Competition* [Online]. [Accessed 4 August 2020]. Available from: http://people.cst.cmich.edu/yelam1k/asee/proceedings/2016/student_regular_papers/2016_asee_ncs_paper_58.pdf.

Yao, Y. and Zhou, B., 2010, October. Naive Bayesian rough sets. In *International Conference on Rough Sets and Knowledge Technology* (pp. 719-726). Springer, Berlin, Heidelberg.

Appendix A
External Materials

BEAST (Bioinspired Evolutionary Agent Simulation Toolkit)

Used for GA simulation in this project. The homepage of BEAST as follows.

http://www.comp.leeds.ac.uk/ar23/BEAST/index.php

Zumo robot

Used in this project as the wrestling robot

https://www.pololu.com/docs/0J63

Arduino IDE

Used for upload program to the robot

https://www.arduino.cc/en/Main/Software

Visual Studio Code

Used for writing and reading the code

https://code.visualstudio.com/

Colaboratory

Used for running python code and generating decision tree.

https://colab.research.google.com/notebooks/intro.ipynb#recent=true

Adobe Premiere

Used for editing demonstration video

https://www.adobe.com/uk/products/premiere.html?sdid=88X75SKT&mv=search&s_kwcid=AL!3085!10!79645965460475!79646037857712&ef_id=XmVlRgAAAJzPTRJS:20200824101727:s

Appendix B
Ethical Issues Addressed

This project is trying to avoid all ethical issue. All the data used in developing the software are from the Zumo robot sensor data and the BEAST, no personal data or classify data contains in this project. All Arduino library and BEAST external code that used in developing the software are open sources. And these codes were not credited as own work. These codes will be downloaded and put in a separate folder in the repositories to avoid plagiarisms.

And this project's code and ideas should not be used in the military field or any field that could cause violence action.

Appendix C
Source Code and Set-up Guide

The source code for this project can be found at:

https://github.com/yeswhos/COMP5200M-MSc-Project

All required dependencies have been uploaded to the repository. And the set-up guide can be found at README in each folder.

The video demonstration for decision tree based control system can be found at:

https://youtu.be/RahYl37PvtM

The video demonstration for genetic algorithm based control system can be found at:

https://youtu.be/Me6_-5wUBW4