University of Alberta

Autonomous Robotic Reactive Behavior: Synthesis, Scalability, and Transparency

by

Paul Anthony den Boef

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

in

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2004

Your file Votre référence
ISBN: 0-612-95893-0
Our file Notre référence
ISBN: 0-612-95893-0

Dedications and Acknowledgments

I would like to dedicate this thesis to my wife, Jessica Suidan, for all of the support she has given me throughout my university education. Additional dedication is extended to my parents and in-laws, who have all been very supportive of my endeavors.

I would like to acknowledge my supervisor, Dr. Witold Pedrycz, for the direction and insight he has provided me during my graduate studies at the University of Alberta.

Table of Contents

CHAPTER 1 INTRODUCTION 1

1.1 INSPIRATION FROM BRAITENBERG 1
1.1.1 Vehicle 1 – Getting Around 2
1.1.2 Vehicle 2 – Fear and Aggression 3
1.1.3 Vehicle 3 – Love 4
1.1.4 Vehicle 4 – Values and Special Tastes 5
1.1.5 Vehicle 5 – Logic 7
1.1.6 Conclusion 7
1.2 BEHAVIOR SYNTHESIS 7
1.2.1 Fuzzy & Evolutionary Controllers 8
1.2.2 ANN-Based Controller 9
1.2.3 Behavior Evaluation 10
1.3 RULE EXTRACTION 11
1.3.1 Rule Extraction Taxonomy 11
1.3.2 Rule Extraction with Continuous Attributes 12
1.3.3 Decompositional vs. Black-Box 13
1.4 CONCLUSION 14

CHAPTER 2 EXPERIMENTATION PLATFORM 15

2.1 ROBOTIC PLATFORM 15
2.1.1 Khepera Overview 16
2.1.2 Khepera Development Logistics 17
2.2 KHEPERA'S ENVIRONMENT 18
2.2.1 Sensory Space 19
2.3 CONCLUSION 22

CHAPTER 3 BEHAVIOR SYNTHESIS 23

3.1 PROPOSED METHOD 24
3.1.1 Backpropagation 24
3.2 ACQUIRING DATA 26
3.3 ANN ARCHITECTURE 27
3.4 ANN LEARNING 32
3.5 BEHAVIOR GENERALIZATION 35
3.6 BEHAVIOR PERFORMANCE SCALING 41
3.7 CONCLUSION 45

CHAPTER 4 ANN RULE EXTRACTION 46

4.1 PROPOSED METHOD 47
4.1.1 Discretization 48
4.1.2 Rule Set Evolution 50
4.2 EXPERIMENTATION 52
4.2.1 Iris Plant 53
4.2.2 Complex ANN 57
4.2.3 Moderate ANN 62
4.2.4 Simple ANN 64
4.2.5 Re-Extraction 70
4.3 DISCUSSION 73
4.4 CONCLUSION 75

CHAPTER 5 CONCLUSION 76

BIBLIOGRAPHY 78

List of Tables

TABLE 3.1 – SAMPLE ANN ARCHITECTURES EXPLORED 29
TABLE 3.2 – SAMPLE ANN ARCHITECTURES PERFORMANCE INDEX 30
TABLE 3.3 – ANN CONNECTION WEIGHTS 34
TABLE 3.4 – TABULATED TRAJECTORY ERROR 41
TABLE 3.5 – TABULATED ANN AND TELEOPERATION (HUMAN) ERROR 43

TABLE 4.1 – DATA SORTED BY INTERVALS 49
TABLE 4.2 – INTERVALS, CLASS FREQUENCIES, AND X^2 VALUES 54
TABLE 4.3 – INTERVALS, CLASS FREQUENCIES, AND X^2 VALUES (2) 54
TABLE 4.4 – EVOLUTION RESULTS FOR 1 THROUGH 5 RULES 55
TABLE 4.5 – DISCRETIZED INTERVALS FOR THE WALL FOLLOWING DATA 58
TABLE 4.6 – DISCRETIZED INTERVALS FOR THE *MODERATE* WALL FOLLOWING DATA 63
TABLE 4.7 – DISCRETIZED INTERVALS FOR THE *SIMPLIFIED* WALL FOLLOWING DATA 65
TABLE 4.8 – PERFORMANCE OF THE DISCRETE RULE SETS 70
TABLE 4.9 – DISCRETIZED INTERVALS FOR THE *RE-EXTRACTION* WALL FOLLOWING DATA .. 71
TABLE 4.10 – ORIGINALLY EXTRACTED RULES VS. RE-EXTRACTED RULES 72

List of Figures

FIGURE 1.1 – VEHICLE 1: GETTING AROUND 2
FIGURE 1.2 – VEHICLE 2: FEAR AND AGGRESSION 3
FIGURE 1.3 – FEAR AND AGGRESSION TRAJECTORY 4
FIGURE 1.4 – VEHICLE 3: LOVE 4
FIGURE 1.5 – LOVE TRAJECTORY 5
FIGURE 1.6 – VEHICLE 4: VALUES AND SPECIAL TASTES 6
FIGURE 1.7 – VALUES AND SPECIAL TASTES TRAJECTORY 6

FIGURE 2.1 – KHEPERA MINATURE ROBOT 16
FIGURE 2.2 – KHEPERA DATA LOG 17
FIGURE 2.3 – KHEPERA ENVIRONMENT 18
FIGURE 2.4 – GENERALIZED I7 SENSORY SPACE 19
FIGURE 2.5 – MEASURED I7 SENSORY SPACE 20
FIGURE 2.6 – MODELED I7 SENSORY SPACE 21

FIGURE 3.1 – METHODOLOGY FOR BEHAVIOR SYNTHESIS 24
FIGURE 3.2 – GENERALIZED MULTILAYER ANN 25
FIGURE 3.3 – TRAINING AND TESTING DATA 27
FIGURE 3.4 – GENERALIZED TWO-LAYERED ANN 28
FIGURE 3.5 – PERFORMANCE INDEX VS. NO. PAST VECTORS 31
FIGURE 3.6 – PERFORMANCE INDEX VS. NO. HIDDEN NEURONS 31
FIGURE 3.7 – TESTING AND TRAINING PERFORMANCE INDICES 33
FIGURE 3.8 – GRAYSCALE REPRESENTATION OF ANN CONNECTIONS 34
FIGURE 3.9 – TRAINED ANN WORLD1 TRAJECTORY 35
FIGURE 3.10 – TRAINED ANN WORLD2 AND WORLD3 TRAJECTORY 37
FIGURE 3.11 – TRAINED ANN WORLD4 AND WORLD5 TRAJECTORY 38
FIGURE 3.12 – TRAINED ANN WORLD6 TRAJECTORY 39
FIGURE 3.13 – TRAINED ANN WORLD7 AND WORLD8 TRAJECTORY 40
FIGURE 3.14 – SUMMARY OF TRAJECTORY ERROR 41
FIGURE 3.15 – TRAINED ANN WITH UNITY OUTPUT SCALING 42
FIGURE 3.16 – TRAINED ANN WITH OUTPUT SCALING OF 2X 43
FIGURE 3.17 – TRAINED ANN VS. HUMAN OPERATOR 44

FIGURE 4.1 – METHODOLOGY FOR DISCRETE RULE SET EXTRACTION 47
FIGURE 4.3 – INDIVIDUAL CHROMOSOME 51
FIGURE 4.4 – CROSSOVER AND MUTATION 52
FIGURE 4.5 – EVOLUTION WITH 3 RULES 56
FIGURE 4.6 – DISCRETE RULE SET WITH THE HIGHEST TESTING FITNESS 56
FIGURE 4.7 – IRIS 3-RULE EXTRACTED RULE SETS 56

FIGURE 4.8 – IRIS 4-RULE EXTRACTED RULE SETS 56
FIGURE 4.9 – WALL FOLLOW CLASSIFICATION SPACE 58
FIGURE 4.10 – CHROMOSOME BEST FITNESS FOR COMPLEX WALL 60
FIGURE 4.11 – CHROMOSOME CLASSIFICATION ERROR FOR COMPLEX WALL 60
FIGURE 4.12 – AVERAGE RULE COMPLEXITY FOR COMPLEX WALL 60
FIGURE 4.13 – CHROMOSOME BEST FITNESS FOR COMPLEX WALL (2) 61
FIGURE 4.14 – CHROMOSOME CLASSIFICATION ERROR FOR COMPLEX WALL (2) 61
FIGURE 4.15 – AVERAGE RULE COMPLEXITY FOR COMPLEX WALL (2) 62
FIGURE 4.16 – CHROMOSOME BEST FITNESS FOR MODERATE WALL 63
FIGURE 4.17 – CHROMOSOME CLASSIFICATION ERROR FOR MODERATE WALL 64
FIGURE 4.18 – AVERAGE RULE COMPLEXITY FOR MODERATE WALL 64
FIGURE 4.19 – CHROMOSOME BEST FITNESS FOR SIMPLIFIED WALL 65
FIGURE 4.20 – CHROMOSOME CLASSIFICATION ERROR FOR SIMPLIFIED WALL 66
FIGURE 4.21 – AVERAGE RULE COMPLEXITY FOR SIMPLIFIED WALL 66
FIGURE 4.22 – CHROMOSOME BEST FITNESS FOR SIMPLIFIED WALL (2) 66
FIGURE 4.23 – CHROMOSOME CLASSIFICATION ERROR FOR SIMPLIFIED WALL (2) 67
FIGURE 4.24 – AVERAGE RULE COMPLEXITY FOR SIMPLIFIED WALL (2) 67
FIGURE 4.25 – EVOLUTION DETAILS FOR SIMPLIFIED WALL 68
FIGURE 4.26 – EVOLUTION DETAILS FOR SIMPLIFIED WALL (2) 68
FIGURE 4.27 – EXTRACTED DISCRETE 12-RULE KHEPERA TRAJECTORY 69
FIGURE 4.28 – ORIGINAL 12 RULES EXTRACTED FROM THE *SIMPLE* ANN 71
FIGURE 4.29 – 12 RULES ARE RE-EXTRACTED FROM THE ORIGINAL 12 RULES 71
FIGURE 4.30 – RE-EXTRACTED DISCRETE 12-RULE TRAJECTORY 73

Chapter 1

Introduction

It is well known that autonomous robotic reactive behaviors can be synthesized. However the topics of generalization ability, behavior performance scalability, and behavior transparency are less known. The investigation of these three topics constitutes the main objective of this thesis. Behavior synthesis, including generalization ability and performance scalability, is investigated in Chapter 3. Behavior transparency is investigated in Chapter 4. Finally, Chapter 5 provides concluding remarks and recommendations.

This chapter provides an introduction to behavior synthesis and behavior transparency; the remainder of this chapter is organized in sections. Section 1.1 provides an overview of behavior synthesis with regards to Valentino Braitenberg. In this section, the concept of behavior synthesis and behavior transparency is introduced. Section 1.2 discusses the prior art of behavior synthesis seen in the literature. Finally, Section 1.3 discusses the prior art of rule extraction.

1.1 Inspiration from Braitenberg

Valentino Braitenberg presents fourteen vehicle designs in his famous book "Vehicles: Experiments in Synthetic Psychology" [1]. These vehicle designs are shown to synthetically

produce autonomous behaviors with increasing complexity. Braitenberg's vehicle designs are inspiring because he starts with comprehensive vehicle designs and provides notable accounts for their resulting autonomous behaviors. The more advance vehicle designs become increasingly less comprehensive, yet Braitenberg's illuminating words are convincing enough to provide inspiration that such vehicle designs may be feasible. The proposed vehicle designs are quite theoretical and conceptual, and no implementation or experimentation is offered by Braitenberg. Exploration of this void continues to be challenging research task undertaken by many who have received inspiration from Braitenberg's famous vehicle designs.

An overview of Braitenberg's first five vehicle designs is presented in this introduction. Although the fifth vehicle design will be the focus in this thesis, it is important to examine the first four in order to observe the progression in behavior complexity and the decrease in behavior transparency. Each vehicle design section contains a brief description, which even includes some of Braitenberg's diagrams for verbatim. Some simple experimentation is conducted using the mobile robot, Khepera. (See section 2.1 for an overview of Khepera). In some aspects, the vehicle designs are extrapolated and/or manipulated for appropriate implementation on Khepera. It is with hope that these implementations do not lose the spirit of the vehicle designs presented by Valentino Braitenberg.

1.1.1 Vehicle 1 – Getting Around

Vehicle 1 is very simple—it contains one sensor and one motor. The force exerted by the motor is proportional to the absolute temperature measured from the sensor.

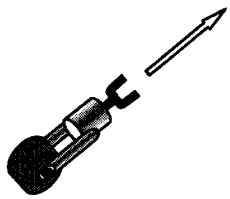


Figure 1.1 – Vehicle 1 contains one motor and one temperature sensor with a fixed connection between them [1].

Braitenberg asserts that this vehicle would display a preference for cooler temperatures since the vehicle would speed up to exit warmer environments while slowing down when cooler environments are reached. The rudimentary vehicle design produces a rudimentary behavior allowing vehicle 1 to "get around".

1.1.2 Vehicle 2 – Fear and Aggression

This vehicle is equipped with two sensors providing a differential input, and two motors providing differential drive. The forces exerted by the motors are proportional to the light intensity measured from the sensors. Braitenberg asserts that the behaviors of fear and aggression can be synthetically produced with the fixed connections between sensors and motors seen in Figure 1.2.

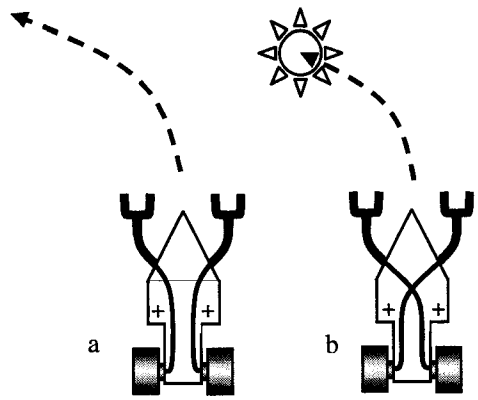


Figure 1.2 – Vehicle 2 displaying fear (left) and aggression (right) for a light source [1].

The behavior depicted in Figure 1.2 can be reproduced with the mobile robot, Khepera. Khepera's sensor readings from *i1* and *i2* can be averaged to form the left sensor while sensors *i3* and *i4* are averaged to form the right sensor. The left and right virtual sensors measure a differential infrared reflection from a nearby object. The motors are set to speeds proportional to the perceived infrared reflection from a nearby object. The proportionality is biased slightly such that the minimum speed attainable is 0.008cm/s. Khepera is then placed in close presence to an infrared reflecting object. The object is placed exactly 12 cm north and 5 cm west of Khepera's initial position. The resulting trajectory and motor speeds for the "fear" and "aggression" vehicles are recorded in Figure 1.3.

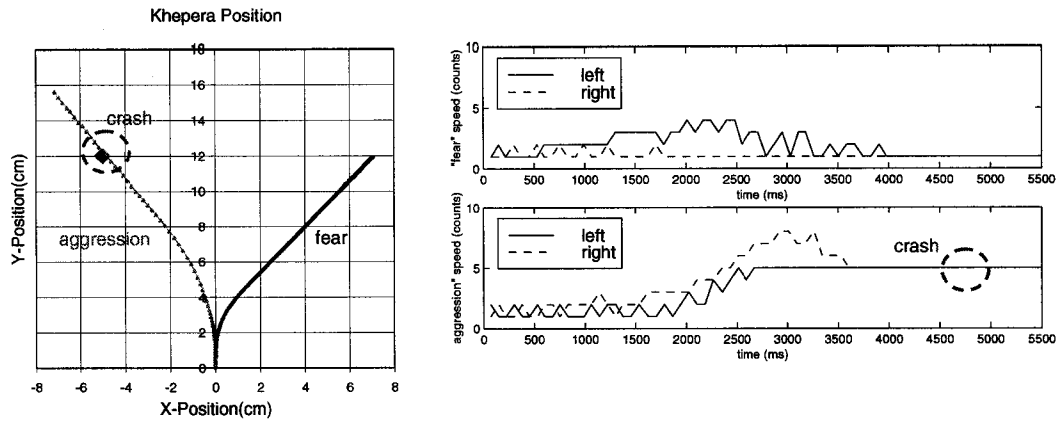


Figure 1.3 – Fear and aggression trajectories (left) and differential motor speeds (top right and bottom right). The aggressive vehicle accelerates toward the object and crashes into it while the fearsome vehicle veers away.

The trivial connections between sensors and actuators result in synthetic behaviors resembling "fear" and "aggression" towards the nearby object.

1.1.3 Vehicle 3 – Love

This vehicle design consists of two or more sensors, all in differential configurations. The sensory capabilities are widely varied and include light intensity, temperature, and oxygen concentration. The forces exerted by the motors are a function of the sensory input. Valentino Braitenberg asserts that the behavior "love" can be synthetically produced using the fixed connections seen in Figure 1.4.

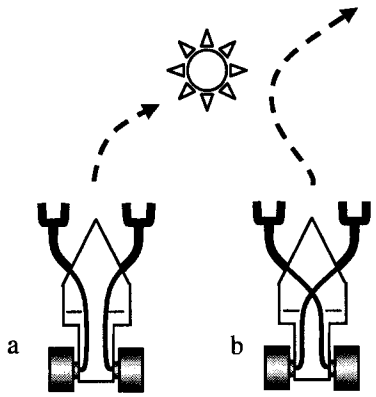


Figure 1.4 – The inhibitory sensors of Vehicle 3 attracts the vehicles to the light source: design (a) rests in close proximity to the light, and design (b) approaches the light but soon veers away [1].

Vehicle designs (a) and (b) display synthetic behaviors, which Valentino Braitenberg labels as "love". Vehicle design (b), however, will soon veer away from the light source. This behavior could be termed "promiscuous". Khepera is used to reproduce the behavior seen in Figure 1.4. The implementation is similar to the previous vehicle. Khepera's sensors *i1* and *i2* can be averaged to form the left sensor while sensors *i3* and *i4* are averaged to form the right sensor. The two virtual sensors measure a differential infrared reflection from a nearby

object. The wide range of sensors described by Valentino Braitenberg cannot be easily implemented at this time since Khepera is only equipped with one type of sensor. The motors speeds are inhibited by IR reflection from a nearby object. Khepera is placed in close presence to an infrared reflecting object. The object is placed exactly 20 cm north and 10 cm east of Khepera's initial position. Khepera's resulting trajectory speed and for the two vehicle designs are recorded in Figure 1.5.

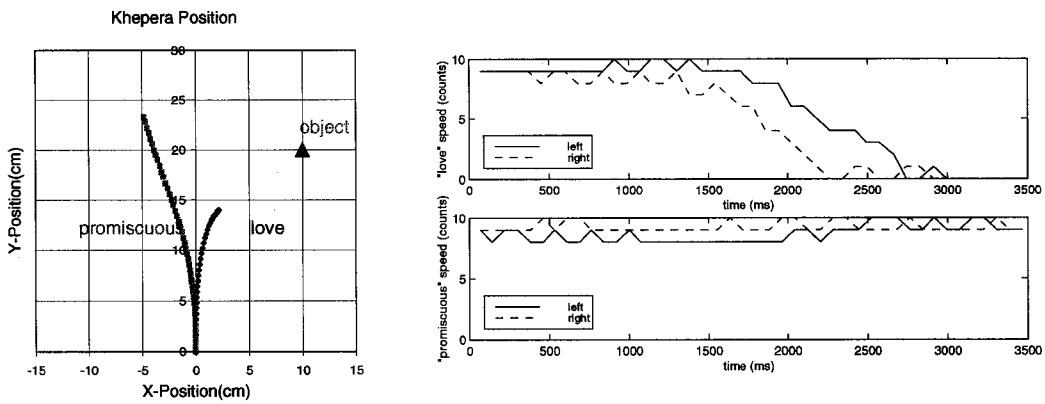


Figure 1.5 – Love and Promiscuous trajectories (left) and differential motor speeds (right). The "love" design steers towards the object, slows down, and eventually stops in front of the object. The "promiscuous" design drives towards the object but soon veers away from it.

Braitenberg's third vehicle design, much like the first two, consists of trivial connections between sensors and actuators in order to create a synthetic behavior. The synthetic behaviors seen from the first three vehicle designs are quite predictable. This behavior transparency is lost with the design complications introduced by vehicle 4.

1.1.4 Vehicle 4 – Values and Special Tastes

The fourth vehicle is similar to the previous vehicle because it also consists of two or more sensors in a differential configuration and two motors providing differential drive. The distinguishing modification to this vehicle design is that the force exerted by each motor is a non-linear function of the sensory stimulation intensity. As a result, the synthetic behavior becomes more unpredictable. This is demonstrated by implementing Khepera with the non-linear function seen in Figure 1.6.

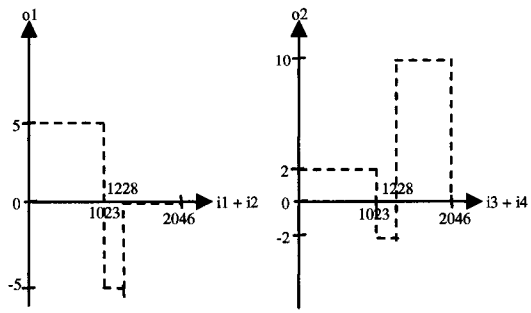


Figure 1.6 – Arbitrary non-linear functions are implemented for the left and right motors using the left and right plots, respectively.

Khepera is placed in close presence to an infrared light reflecting object. The object is placed 17.5 cm east and 5 cm south of Khepera's initial position. Khepera's resulting trajectory is recorded in Figure 1.7.

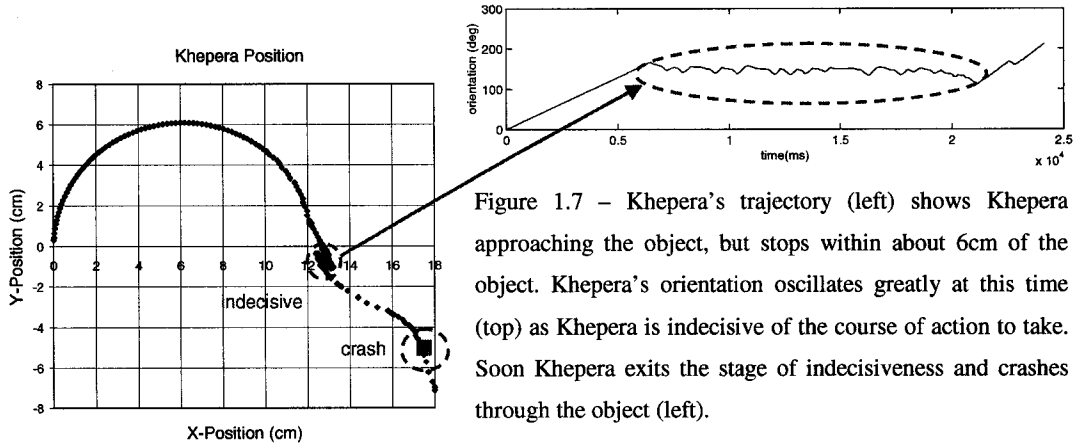


Figure 1.7 – Khepera's trajectory (left) shows Khepera approaching the object, but stops within about 6cm of the object. Khepera's orientation oscillates greatly at this time (top) as Khepera is indecisive of the course of action to take. Soon Khepera exits the stage of indecisiveness and crashes through the object (left).

Khepera remains indecisive for about 15 seconds before proceeding on a collision course with the object. The indecisiveness is caused by the drop from +5 to -5 and +2 to -2 in the functions shown in Figure 1.6. Consider the speed of the left motor, which is a function of the sensor (*i1+i2*). When (*i1+i2*) is low, the left side of Khepera moves forward with a speed of 5. If this movement is towards the object, then (*i1+i2*) will increase. If (*i1+i2*) increases past 1023 while remaining less than 1228, then the left motor speed is promptly changed to -5. If this causes movement away from the object, then (*i1+i2*) will decrease. A similar cyclical pattern is seen with the right motor. The oscillatory cycle is seen to be unstable as (*i3+i4*) eventually reaches levels in excess of 1228, which causes the right motor to attain speeds of 10. At this point, Khepera has exited the orientation oscillation stage.

It is extremely difficult to predict Khepera's behavior when its reactive controller is an arbitrary non-linear function. It is surprising that the indecisive behavior reported in Figure 1.7 can be produced by utilizing the simple reactive controller functions seen in Figure 1.6. Vehicle four distinguishes itself from the previous three vehicle designs in two main respects:

1. there are numerous design degrees of freedom, and
2. the resulting behavior for each design can be difficult to predict.

Evidently, these two observations seem to apply to the remaining ten Braitenberg vehicle designs.

1.1.5 Vehicle 5 – Logic

This vehicle design introduces logical networks consisting of units resembling McCulloch and Pitts neurons. These logical networks are similar to the generalized modern artificial neural networks (ANNs), which are widely used today. The study of the "logic" vehicle design is interesting since complex behaviors can be synthesized. However, much like the fourth vehicle design, Braitenberg is not able to supplement the vehicle design with a viable methodology for synthesizing autonomous behavior. Instead, the vehicle designs remain theoretical and without sufficient guidelines for practical implementation. As a result, Braitenberg presents only two simple examples despite the numerous possible hierarchies and applications. Braitenberg is able to explain these two examples in a theoretical sense, but leaves the practical implementation up to the reader's imagination.

1.1.6 Conclusion

There are two main observations that can be stated about the first five Braitenberg vehicle designs. First, each successive vehicle design is increasingly complex. The complexity of the fourth and fifth vehicle designs surpasses Braitenberg's ability to provide specific design implementation guidelines. Second, as the vehicle design complexity increases, the behavior transparency tends to decrease. It is difficult to predict or understand the behaviors exhibited by more complex designs, such as the fourth and fifth vehicle designs. Braitenberg provides inspiration to investigate behavior synthesis and behavior transparency.

1.2 Behavior Synthesis

Autonomous robotic behaviors are created by controllers, which on a robot provide a mapping between the sensory space and the actuator space. These controllers are classified

as either reactive or deliberative. Reactive controllers contain no state or memory while deliberative controllers do. The study of reactive controllers is very important since they not only serve as stand-alone controllers, but as building blocks for more complex deliberative systems as shown in [2]. Constructing reactive controllers based on artificial neural networks (ANNs) or fuzzy logic rules are proven methods for creating low-level behaviors such as obstacle avoidance and wall following.

In this thesis, we will examine a reactive ANN-based controller with memory in order to synthesis a wall following behavior. While the introduction of memory implies that the controller is deliberative, the controller architecture most closely resembles a reactive design and is hereby considered reactive. ANNs are a computing paradigm inspired by the parallel architecture of biological neurons found in animal brains and have been applied to a wide variety of problems in which algorithmic solution cannot easily be derived. These applications include, but are not limited to, disease diagnosis, business costs/sales predictions, process plant control, and robotics control. ANNs, with related learning algorithms, are able to provide the means for practical "logic" vehicle (see Section 1.1.5) design implementation. Furthermore, properly trained ANNs have been shown to be universal function approximators [3]. Therefore, they are able to approximate arbitrary non-linear functions, such as the one used to map sensors to motors in vehicle design four (see Section 1.1.4). Obviously, the first three vehicle designs could also be implemented with a properly trained ANN.

The study of ANNs is a significant research interest because of their proven ability to synthesize behavior; however, ANNs are by no means the only practical method. Numerous alternative approaches are widely used including fuzzy logic and evolutionary optimizations. The utility of one method over another method is often application dependant. Therefore, an overview of approaches is provided, as much as possible, in the context of the wall follow problem.

1.2.1 Fuzzy & Evolutionary Controllers

Implementing fuzzy rule base controller is attractive since understanding and creating fuzzy rules to govern a robots behavior is intuitive, whereas understanding an ANN is not [4]. However, successful implementations of fuzzy rules can be very difficult. Thorough knowledge of the robots sensory and actuator characteristics is often required in order to

fuzzify the antecedent (sensory values) and consequent (actuator values) variable appropriately. Methods for automatic fuzzy rule generation can be applied. For example, a training data set can be used such that the consequent of a rule is determined by the degree to which the training data satisfies its antecedent [4]. In this example, a training sample subset T_i is built for each antecedent A^i such that each member of T_i satisfies the antecedent to some degree. The consequent B^i is chosen as the output fuzzy subset for which the weighted average of outputs in set T_i has maximal membership. However, fuzzy rules learning remains limited in comparison to the ANN learning algorithms widely used—such as the backpropagation algorithm.

Genetic Algorithms (GAs) are a very popular evolutionary design method used to optimize reactive controllers. GAs have been applied to evolve fuzzy control rules [5,6,13] and ANNs [7,8,9] for autonomous robotic operation by favoring fit designs while non-fit solutions are disregarded [10]. Artificial evolution can develop controllers that exploit relevant features of the environment that were not explicitly defined in advanced [7]. The downfall extends from the difficulties involved in designing an appropriate fitness function and, therefore, difficulties in synthesizing behaviors for the robot. Constructing an appropriate fitness function for wall following is shown to be difficult in [8,11] and respectable wall following behaviors are difficult to achieve. Worse yet, the evolutionary process devour an extraordinary amount of time since evaluations of individual solutions must occupy at least several seconds of real time on the robot—unless the evolutionary process is completed with a simulator. Appropriately, controllers are evolved in simulation but tested on a real physical system in [8,9,12,13]. This saves time consuming physical wear on the robot but enables final results to be reported from a real physical system thereby validating the simulator employed.

1.2.2 ANN-Based Controller

An ANN-based controller is proposed to synthesize the wall follow behavior because it offers many advantages over the alternatives discussed in this introductory chapter. ANN learning algorithms are more powerful than those seen for fuzzy controllers and do not require the difficult derivation of fitness functions that evolutionary algorithms require. Yamada demonstrates that autonomous wall following can be accomplished with unsupervised learning using a self organizing map ANN [14]. Alternatively, supervised

learning can be used for ANN controllers to capture implicit knowledge and execute robotic behavior by applying the captured knowledge.

In this thesis, the backpropagation learning algorithm is explored for an ANN-based wall following controller since it is widely used and proven to be a very powerful supervised learning algorithm. Hybrid systems such as neuro-fuzzy networks may exploit the learning capabilities of ANN learning algorithms while allowing encapsulated knowledge to be expressed in a comprehensive manner. Nonetheless, it is decided to investigate the backpropagation learning algorithm for behavior synthesis with the further goal of exploring a method for enhancing ANN behavior transparency. Backpropagation is a generalized gradient based delta algorithm, which aims to minimize discrepancy between the ANN calculated output(s) and the target output(s) dictated by implicit knowledge [15]. Implicit knowledge can be encapsulated in a few different ways. In on-line reinforcement learning, the supervisor rewards a controller for good behavior and conversely punishes poor behavior [16,17,18]. In an off-line supervised setting, the supervisor collects data containing sensor values and corresponding actuator values (e.g. target motor speeds). Encapsulation of wall follow implicit knowledge is feasible by manually driving the robot for wall following and acquiring data to be used for training.

1.2.3 Behavior Evaluation

The objective of the ANN-based controller is squarely focused on autonomous wall following, which is hereby defined as driving alongside a wall while maintaining a constant distance to the wall. Varying design methodologies exist throughout the literature that aim to accomplish wall following and/or similar objectives. However, the implementation is usually limited to simplistic purely reactive designs with very little exploration of the wall following behavior itself. For example, Tunstel [19,20,21] demonstrates how purely reactive behaviors such as wall follow can be combined in a hierarchy with other reactive behaviors to create more complex behaviors for autonomous exploration. Tunstel research is valuable but creates a void in that the low-level reactive behaviors aren't critically explored. Aguirre and González [2] research the fusion of reactive behaviors including wall following. While more exploration is given to the reactive wall follow behavior than Tunstel, there remains much to explore. Recording robotic trajectories is a significant step towards quantitative evaluation of controllers. This is done in [22] for a simple wall following task. Numerous

researchers have implemented or used a wall follow behavior (or other low-level behaviors); however, there seems to be a lack of critical evaluation of the behavior.

The deficiency of evaluation likely extends from the lack of widely adopted analysis metrics, which Nehmzow [23] argues is vitally important to the advancement of the mobile robotics field. Qualitative observations and evaluations seem to be the norm while quantitative metrics are sparsely utilized. It seems reasonable to expect that task-oriented controllers can be quantitatively evaluated—especially for simplistic behaviors like wall following. This thesis ventures to quantitatively evaluate an autonomous robotic controller implemented for wall following in terms of its ability to generalize over numerous environments with varying wall geometries, and its performance scalability over varying trajectory velocities.

1.3 Rule Extraction

ANNs have been successfully applied to a wide variety of applications and have been widely accepted because of their proven accuracy [24]. Their downfall stems from fact that the knowledge is distributed across the weighted connections, which makes ANNs incomprehensive [25]. ANN rule extraction is a relatively new research area that attempts to unveil the knowledge embedded into the ANN connection weights. In doing so, the problem of deprived behavior transparency is addressed.

Numerous methods exist in the literature for extracting rules from ANNs; however, they often impose problem domain restrictions preventing the method from being universally adoptable. There exists a tradeoff between ANN complexity and performance, as seen in [26]. In some instances it is desirable to utilize a complex network to obtain higher performance on complex applications. The implication being that the rule extraction process becomes significantly more challenging. A complex ANN contains numerous rules of varying significance, including erroneous rules. The goal of the rule extraction chapter is to extract the simplest possible set of comprehensive rules from a trained complex ANN, regardless of architecture or ANN learning scheme used.

1.3.1 Rule Extraction Taxonomy

In order to evaluate and compare the numerous different rule extraction methods found in the literature, it seems reasonable that universal terminology, comparative criteria, and

benchmarks should be used adopted to evaluate the methods [27]. It seems reasonable to classify the rule extraction methods into the following three general categories [27]:

1. Decompositional. This approach extracts rules by directly examining the internal architecture of the ANN.
2. Black-box. This approach examines the input-output relations of the ANN in order to build rules.
3. Eclectic. This is a hybrid decompositional/black-box approach that uses the ANN architecture as well as the input-output relations of the ANN to build rules.

Regardless of the category, it seems necessary to evaluate rule extraction algorithms with a common criteria. Jane Neumann [28] proposes to evaluate rule extraction algorithms with the following four criteria: complexity of the algorithm, quality of the extracted rules, consistency of the algorithm, and applicability of the algorithm. Other authors have proposed similar taxonomy such as in [62]. The quality of the extracted rules is probably the most significant criterion, which is further broken down by Towell [29] into three categories: accuracy, fidelity, and comprehensibility. The capability to correctly classify a testing data set is termed as the *accuracy* of the extracted rules. The capability to mimic the behavior of the trained ANN is termed as the *fidelity* of the extracted rules. Lastly, the *comprehensibility* of the extracted rules considers the extent that rules are readable and understandable to humans. Neumann provides evaluation on 16 different rule extraction algorithms, many of which are tested against the benchmark data sets: the Monk's domain, Iris Plant, Wisconsin Breast Cancer, and the DNA Promoter domain.

1.3.2 Rule Extraction with Continuous Attributes

Most rule extraction methods do not support ANNs with continuous attributes. For example, BIO-RE [30], RULENEG [31], Activation space clustering [32], SUBSET [29], M-of-N [29], and [26,33] all impose a restriction making the method only applicable to ANNs with binary inputs and/or binary outputs. This can seemingly limit the domain of applications in which the method can be utilized. To get around this problem, one could partition the continuous variables into intervals to be binary encoded thereby increasing the number of network inputs and somewhat changing the architecture. This is done in [34] in part of a method that utilizes Karnaugh maps to extract rules hierarchally through a combination of dominant rules and less dominant rules or exceptions. This method exploits logical *don't cares* to extract simplified rules quite successfully. However, methods that partition the

attributes in a seemingly arbitrary manner suffer by not critically considering the effects on the consistency of the rules with the data. Partitioning with linear membership functions is performed arbitrarily in [35] but with some consideration of the implications: when replaced by a Gaussian membership function it is noted that the results improve significantly for the Iris database problem. The authors recognize this phenomenon and declare discretization of continuous attributes as a future research direction.

Methods such as the Chi-Merge [36] and its successor Chi2 [37,38] are based on the χ^2 statistic and have demonstrated great success in discretization of continuous attributes. Chi2 is a discretization algorithm used to partition continuous attributes into a minimal number of intervals such that a desired data consistency rate is preserved. In the event that the Chi2 algorithm completely merges all intervals together for a given attribute, the attribute is deemed unneeded. This is the process of feature selection. The significance of the Chi2 algorithm is its ability to perform both feature selection and discretization while preserving a minimal specified level of consistency with the data. This can be of great benefit over algorithms such as the information-theoretic algorithm [39], which accomplishes feature selection but not discretization. It seems clear that discretization is a required research direction for rule extraction in order to support applications with continuous attributes. Full-RE [30] and Neurolinear [40] rule extraction methods exploit the Chi2 algorithm to support continuous attributes and are reported to achieve high quality results in comparison to numerous other rule extraction methods [28]. Neurolinear is also reported to achieve relatively high accuracy in [41] in comparison to other methods.

1.3.3 Decompositional vs. Black-Box

Neurolinear and Full-RE, among many other methods, are examples of the decompositional approach since they extract rules by analyzing the ANN architecture. A significant implication of this approach is that the complexity of the method is strongly dependant on the complexity and composition of the ANN architecture. For example, extracted rules are said to be a one-to-one mapping of the network using the decomposition method of NeuroRule [42]. Some decompositional methods impose restrictions on the ANN architecture or learning scheme used as seen in [43] and [34]. The motivation for proprietary architecture is most notably to allow simplicity and effectiveness of extracting rules. However, this usually makes the method insufficient for existing trained ANNs and may restrict the applications that the ANN architecture can be applied to.

The black-box approach may be advantageous in scenarios where the ANN architecture is very complicated such as the *complex* ANN architecture considered in this thesis. Huan Liu proposes the X2R rule generator in combination with the Chi2 algorithm to generate concise rules from raw datasets in [44]. Alternative methods utilizing genetic algorithms have been shown to be very effective in generating accurate and comprehensive fuzzy rules on a variety of problem domains such as breast cancer diagnosis [45] and iris plant classification [46]. Genetic algorithms have been applied to decision trees with great success in [47] and have shown favorable results ahead of inductive approaches in [48] and [49]. A fuzzy rule set is naturally desirable since their linguistic variables permit ease of comprehension. However, designing the linguistic variables and their membership functions is not trivial. In [35], it is shown how different membership functions can cause distinctly different results.

1.4 Conclusion

Valentino Braitenberg demonstrates that behavior transparency tends to decrease as behavior complexity increases. His fifth vehicle design can reach a level of complexity whereby one cannot practically perceive or predict the vehicle behavior based on an inspection of the design. This phenomenon is the root of the main thesis objective. The main objective of this thesis is to explore robotic reactive behavior synthesis and methods for enhancing behavior transparency so that a synthesized behavior can be represented in a human comprehensive manner.

Several different approaches have been discussed in this section and it is decided to pursue behavior synthesis using ANNs. The learning capabilities of ANNs are the main reason for selecting them over alternative approaches. It should be noted that an ANN-based controller closely resembles the controller used in Braitenberg's fifth vehicle design. In the same way that Braitenberg's fifth vehicle design has poor behavior transparency, it can be seen that ANNs have poor behavior transparency. Therefore, a further objective of the research is to be able to represent a trained ANN in a more comprehensive manner. In particular, a universal method able to extract a comprehensive set of rules from an ANN is sought after.

Chapter 2

Experimentation Platform

This chapter provides details of the platform used for experimentation throughout this thesis. The goal is to clearly define and manage the experimentation platform such that controlled experiments can be conducted.

This remainder of this chapter is organized in sections. Section 2.1 provides an overview of Khepera and outlines the logistics of development required for experimentation. Section 2.2 presents the experimentation environment and documents the related sensory capability of Khepera. Finally, Section 2.3 provides concluding remarks.

2.1 Robotic Platform

The utilization of physical robots for experimentation is considered by some researchers to be dispensable to simulations, which attempts model the physical system as accurately as required. However, the task of modeling a physical system is challenging with difficulty increasing very rapidly as greater model accuracy is desired. Accurate models must account for physical non-linear anomalies, such as frictional forces. Experimentation on a physical system gives more credibility to results achieved by avoiding any doubts that a simulator may introduce. Despite the clear benefits associated with experimentation on a physical

robot, most researchers do not use physical robots for their research. The mechanical and electrical design and/or problems involved in using a physical robot provide a deterrent to many researchers. Commercially available mobile robots, such as Khepera, remove this deterrent by providing a completely operational mechanical and electrical robotic platform suitable for research. To some degree, robotic applications require an application specific mechanical and electrical design. Khepera has a very practical mechanical and electrical design; therefore, it is chosen as the robotic experimentation platform for this thesis. It should be noted that similar robotic platforms could be used to obtain similar results in this thesis.

2.1.1 Khepera Overview

Khepera is a miniature mobile robot built by K-Team in Switzerland. Its shape is cylindrical with a diameter of 55mm, height of 30mm, and mass of 30g. Khepera's miniature size is significant because it enables experiment environments to be scaled down in size, which is very convenient for smaller research laboratories.

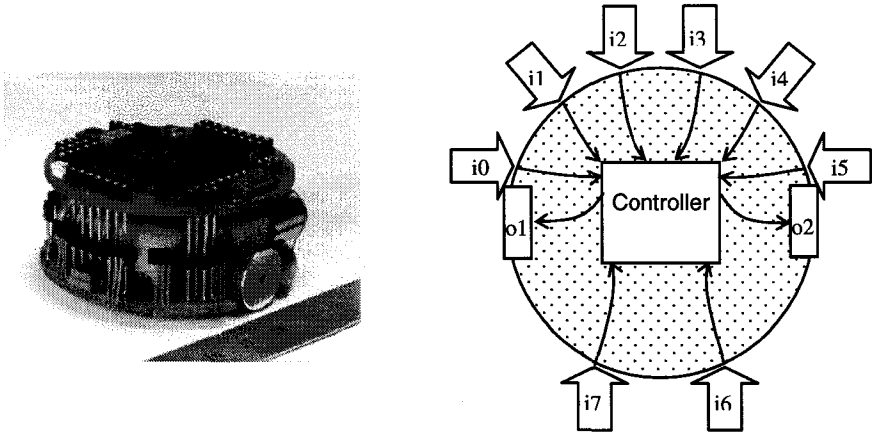


Figure 2.1 – The mobile robot Khepera beside a ruler (left). Eight proximity sensors, labeled i0 through i7, surround the robot. Khepera is equipped with two wheels with speeds o1 and o2.

Khepera possesses two wheels, each driven by its own motor. A position encoder is embedded on each motor. Khepera is able to reach a maximum speed of 1m/s with 7-bit resolution. Turning with zero radius is possible. Khepera has a total of eight infrared proximity detectors with 10-bit resolution, which serve as its sensory input. A programmable Motorola 68331 microprocessor and rechargeable NiCd batteries are embedded for autonomous operation.

2.1.2 Khepera Development Logistics

A host computer is used to control Khepera using the RS232 communications port. Development time is accelerated by employing a 3rd party RS232 serial communications library (WSC4C) from http://www.marshallsoft.com/. A class is written using C++ in order to facilitate all Khepera control and logging operations. This class is written in order to provide a simple, yet powerful, software interface to Khepera.

Khepera logging operations include tracking all sensor readings, motor speeds, position, and orientation angle throughout the duration of the experiment. The sample time for these quantities, which is likely limited by Khepera, is roughly 88ms. The data log is updated in real-time during the experiment. An example Khepera log is illustrated in Figure 2.2. Following the experiment, the data log is archived to an m-file so that data analysis can be performed offline using MATLAB.

t(ms)	i0	i1	i2	i3	i4	i5	i6	i7	al	ar	x(cm)	y(cm)	th(deg)
108	1023	1002	89	1	113	31	553	477	0	0	0	0	-90
198	1023	907	78	0	0	260	424	501	9	8	-0.008	3.41e-006	-90
288	976	920	0	216	0	584	244	522	0	0	-0.444	0.000835	-89.9
375	1023	608	229	0	204	0	467	326	9	9	-0.6	0.00229	-89.5
464	928	1016	0	124	15	249	0	320	10	7	-1.04	0.00508	-89.6

Figure 2.2 – Sample data log produced during experimentation with Khepera. From left to right the log contains a timestamp, eight proximity sensor readings, left and right motor speeds, x and y position coordinates, and an orientation angle.

The position and orientation of Khepera is computed using the readings obtained from the two motor position encoders. By defining s_1 and s_2 to be the respective left and right motor positions, and r to be Khepera's radius, then the orientation angle of Khepera can be defined as follows:

$$\theta(t) = \frac{s_1(t) - s_2(t)}{2r} \qquad [2.1]$$

The position of Khepera in a two-dimensional Cartesian plane can be described, relative to its starting point, as follows:

$$x(t) = \sum_{i=t_0}^{t} \left\{ r \cos\left(\frac{\pi}{2} - \theta(i) \right) \right\} \qquad [2.2]$$

$$y(t) = \sum_{i=t_0}^{t} \left\{ r \sin\left(\frac{\pi}{2} - \theta(i) \right) \right\} \qquad [2.3]$$

Therefore, Khepera is able to easily log its motor speeds, position trajectory, and all sensor values.

Khepera is operated by one of two controllers: an autonomous controller, or human control via teleoperation. In order to facilitate human control over Khepera, a standard 4-button game-pad is interfaced with the PC hosting Khepera. The Microsoft DirectX SDK is used to accomplish this endeavor. The developed program is flexible by enabling numerous different control devices to be used including game-pads and analog joysticks. Control of the robot can be paused, which physically stops Khepera from moving and pauses data logging. Control is resumed by re-pressing the button. A simple graphic user interface (GUI) is developed, using the Microsoft® Foundation Classes (MFC), to facilitate the manual control of Khepera. All real-time sensory readings and motor speeds are displayed in the GUI. Additionally, the status of the control device (e.g. position of joystick, status of buttons, etc.) and the status of the experiment (i.e. running, or paused) are displayed.

2.2 Khepera's Environment

A rectangular environment measuring 74.5cm x 84.5cm is built for Khepera. The walls are covered with 2.5cm wide white infrared reflective tape to enhance Khepera's perception. A smooth Plexiglas floor enables robust mobile operation without slipping. The serial communications line is suspended well above Khepera in order to prevent cord tangling. The experimentation environment is shown in Figure 2.3.

Figure 2.3 – Khepera inside its wall follow environment.

The environment is built with modularity in consideration. Additional walls, each with varying geometry, can be moved and manipulated without any reconstruction. This is essential in being able to create environments with varying wall geometries. The environment shown in Figure 2.3, and its variants created by manipulating the wall geometry, are all modeled in MATLAB. This is done so that trajectory data obtained from experimentation can be analyzed.

2.2.1 Sensory Space

Depending on the experiment, prior knowledge of the sensory space can be extremely important. While some sensory information is provided by K-team, it is beneficial to personally explore the sensory capabilities and limitations of Khepera with the infrared reflective tape used throughout the experimentation environment. Khepera's sensory range and periphery are explored on a two-dimensional plane as illustrated in Figure 2.4.

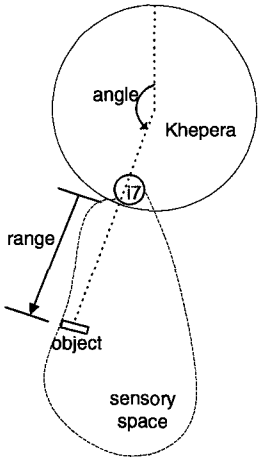


Figure 2.4 - Sensory space for sensor *i7* is generalized in the two dimensional plane with range distance and periphery angle.

Data for all eight sensors is acquired using an object comprised of the same infrared reflective tape used in the construction of the environment of Figure 2.3. It is found that all eight sensors have varying sensory space, which is probably due to variations in the infrared sensor manufacturing process. Figure 2.5 illustrates the results from sensor i7.

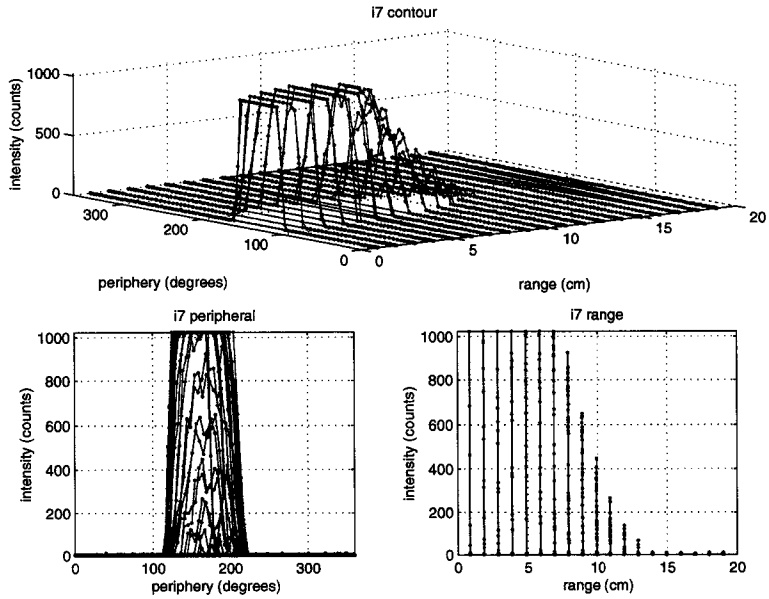


Figure 2.5 – Sensory space of infrared sensor *i7* with an ambient light level reading of 496 counts. The intensity of stimulation is plotted versus distance to an infrared reflecting paper and periphery angle to the infra-red reflecting paper (top). The side views of the contour can be seen (bottom left and bottom right).

Probably the most important observation to be made from Figure 2.5 is the limited range in which the sensor intensity is neither saturated nor zero-valued. This range roughly begins at 8cm and ends at 14cm. This operating range varies somewhat from sensors i0 through i7. The significance of operating within this range is that the distance to an object/wall can be more accurately perceived than when outside the operating range.

The contour shown in Figure 2.5 is by no means smooth and variations of over 100 units of sensory intensity have been observed. This could translate to an object range sensing error of over 1cm for experiments detecting object distances. Furthermore, the contour in Figure 2.5 varies with environment ambient light levels, which Khepera is able to measure. Consideration should be given to the ambient light levels by either keeping environment ambient light levels consistent or by calibrating the sensors accordingly. The former consideration is likely the simplest and is, therefore, chosen.

The relationship between sensor intensity and range can be shown to approximately have the form:

$$intensity = A + \frac{B}{range} \qquad [2.4]$$

This form indicates that the sensory intensity is inversely proportional to the distance to the infrared reflecting object, which intuitively makes sense. In the case of *i7*, the parameters are experimentally solved as A=-1277 counts and B=17393 counts/cm with a correlation coefficient r=0.998.

It is found that the periphery angle is approximately constant when close to the sensor. This is not the case for longer ranges such as 12cm, as shown in Figure 2.5. The sensors provide up to approximately 100 degrees of periphery sensory, which is shown in the bottom left plot of Figure 2.5. The sensory intensity decreases towards the periphery limits of approximately 130° and 230°. The relationship between sensor intensity and periphery is not known. Assuming a parabolic relationship, the following *i7* contour plot can be produced:

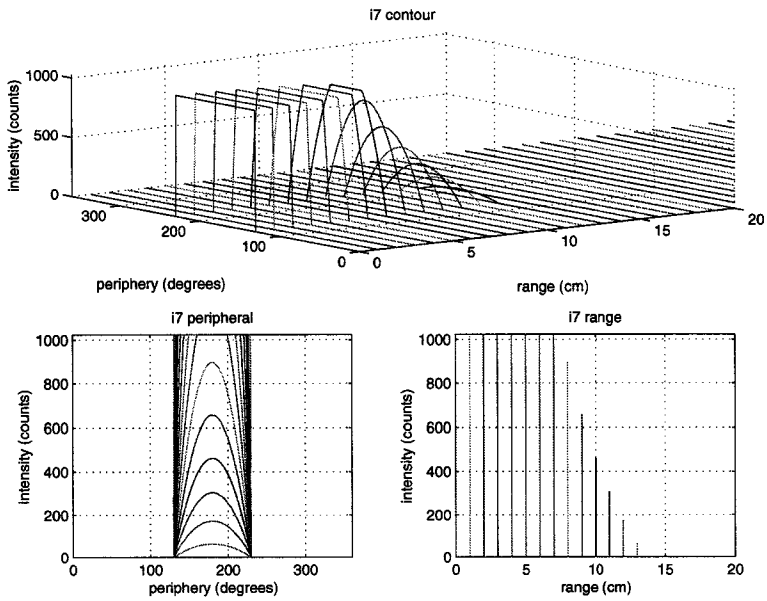


Figure 2.6 – Estimated generalized sensory characteristic function applied for *i7*. Some discrepancies can be observed from the measured contour of Figure 2.5.

Laboratory lighting conditions has a profound effect on the amount of noise seen in the sensor measurements. This phenomenon is illustrated in Figure 2.7 as four different lighting conditions are explored with varying results. For this reason, all Khepera experimentation is performed under very dark laboratory lighting conditions.

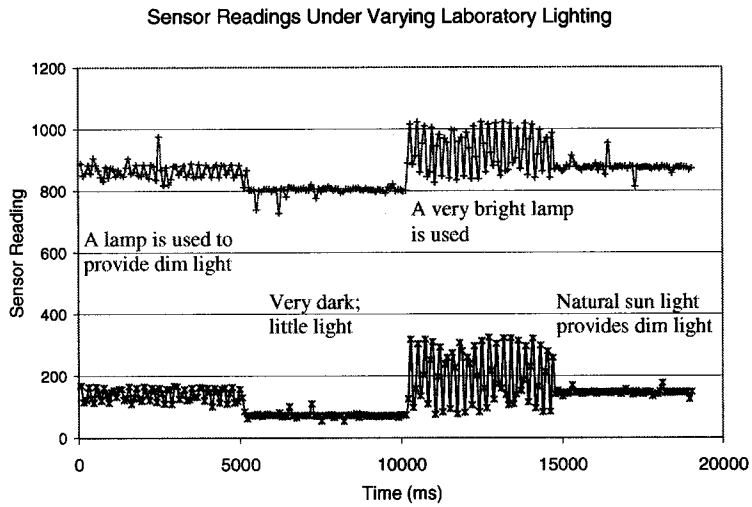


Figure 2.7 – Khepera is placed close enough to a wall such that the wall is detectable by the infrared proximity sensors. Sensor readings are taken during four different laboratory lighting conditions. Sensor noise is noticeably reduced in darker lighting conditions.

The effects of varying sensor noise levels could be explored as is done in [50]. However, noise sensitivity is marked as outside the scope of this thesis. Instead, by maintaining consistently dark laboratory lighting, the undesired effects of noise are minimized and can be assumed negligible.

2.3 Conclusion

The robotic platform and environment are chosen for experimentation. A Khepera class is written in order to facilitate control with autonomous controllers and teleoperation. The constructed environment is modular enough to allow easy manipulation. The operating range of Khepera's sensors, in the context of the environment, is identified.

Chapter 3

Behavior Synthesis

A methodology for autonomous robotic behavior synthesis is presented for the task of wall following. Implicit knowledge is captured and used to train an artificial neural network (ANN) with the backpropagation algorithm. Numerous feed forward ANN architectures are explored in terms of their ability to encapsulate the implicit knowledge and details of learning are provided. Evaluation of the trained ANN is performed on wide variety of wall geometries, which explores the generalization ability of the ANN. The ANN performance is critically evaluated against performance achievable by a human operator via teleoperation.

The remainder of this chapter is organized in sections. Section 3.1 outlines the proposed method. Section 3.3 discusses the ANN architectures that have been explored. Section 3.4 illustrates ANN training details. Section 3.5 tests the trained ANN using numerous environments with varying wall geometries in order to explore the generalization performance of the trained ANN. Section 3.6 investigates performance scaling as the output of the ANN is scaled in order to achieve control during challenging speeds. Performance comparisons between the trained ANN and manual control (via. Teleoperation) are made. Finally, section 3.7 provides conclusions for the chapter.

3.1 Proposed Method

The framework of the proposed method is summarized below in Figure 3.1.

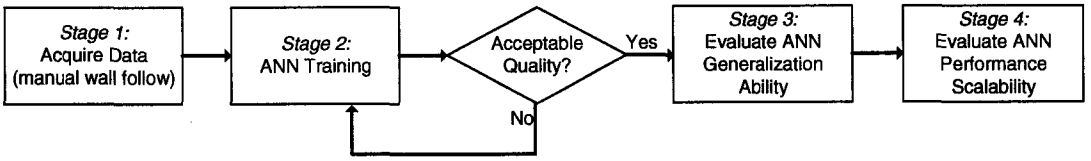


Figure 3.1 – Methodology for behavior synthesis Methodology for behavior synthesis.

It is proposed to train an ANN, using the backpropagation learning algorithm [15], to synthesize the behavior of wall following. Therefore, the objective of Stage 1 is to acquire the data required to train the ANN in off-line learning. Stage 2 explores numerous ANN architectures and learning degrees of freedom in order to best synthesize the wall follow behavior. The trained ANN is implemented on Khepera and tested on numerous wall geometries in Stage 3. Finally, Stage 4 evaluates the performance scalability of the synthesized behavior.

3.1.1 Backpropagation

The backpropagation learning algorithm is introduced here since it is a significant part of this thesis chapter. Details are very brief because backpropagation is widely reported and documented in the literature by researchers in the neurocomputing field. Backpropagation is applied to multilayer ANNs, which are often built using sigmoid neurons is shown in Figure 3.2. The notation used comes from [15], which should be consulted if further details are desired. Note that sigmoid neurons are not a must for the BP algorithm.

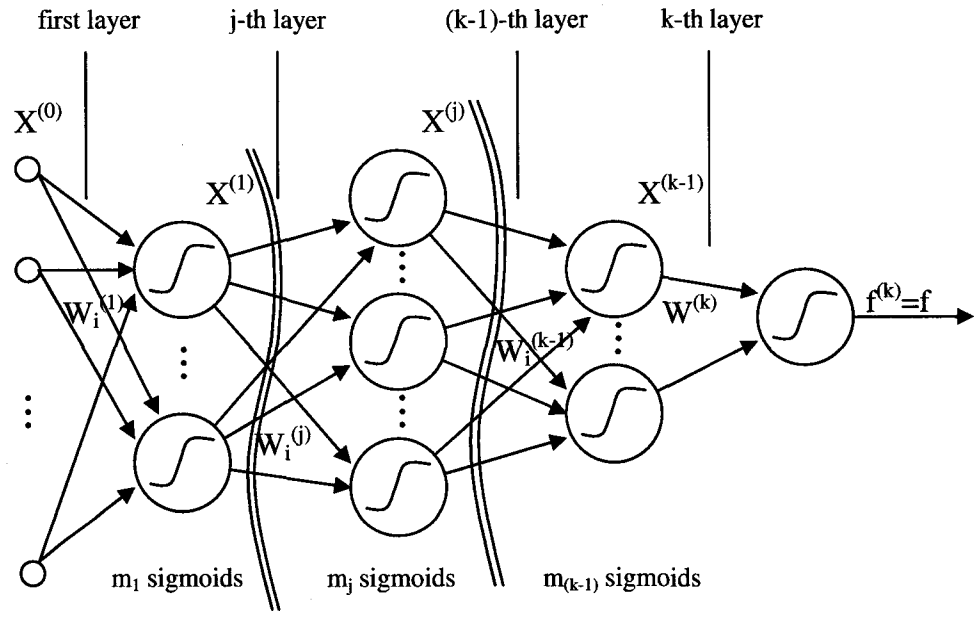


Figure 3.2 – A generalized multilayer feedforward ANN built with sigmoid neurons.

The output of each neuron in Figure 3.2 is computed using the inputs to the neuron, $\mathbf{X}^{(j-1)}$, along with the connection weights, $W_i^{(j)}$, as

$$X^{(j)} = \frac{1}{1+\exp(X^{(j-1)} \cdot W_i^{(j)})}. \qquad [3.1]$$

The backpropagation algorithm aims to reduce the discrepancy (mapping error) between the calculated output, f, and the desired/target output, d, by updating the connection weights in accordance to the gradient decent of the error function:

$$e = (d - f)^2 \qquad [3.2]$$

It can be shown that the output layer connection weights are updated as

$$W_i^{(j)} = W_i^{(j)} + c(d - f)f(1 - f)X^{(k-1)}, \qquad [3.3]$$

where c is the learning rate. Similarly, the connection weights in the intermediate layers are updated as

$$W_i^{(j)} = W_i^{(j)} + \delta_i^{(j)} X^{(j-1)}, \qquad [3.4]$$

where $\delta_i^{(j)}$ is computed recursively by

$$\delta_i^{(j)} = f_i^{(j)}\left(1 - f_i^{(j)}\right)\sum_{l=1}^{m_j+1} \delta_l^{(j+1)} w_{il}^{(j+1)}. \qquad [3.5]$$

This recursive relation implies that the connection weights are updated based on the computed error propagating backwards from the output of the ANN. This phenomenon is the reason why the algorithm is termed "backpropagation".

In off-line learning, numerous input-output pairs (or vectors) are used to train the network. This set of vectors make up the network training set. The backpropagation algorithm cycles through each vector in the training set to compute new network connection weights. In "per-pattern" learning, the connection weights are updated with each vector in the training set. Alternatively, "per-epoch" learning updates the connection weights after all vectors are used to accumulate connection weight changes. Regardless of the connection weight update scheme, the learning cycles are repeated until a desired accuracy is attained. Network accuracy is typically measured using a testing set.

During on-line learning, the connection weights are typically updated after each sample of a reinforcement signal. A reinforcement signal is provided to reward correct output and to punish incorrect output during a period in which the ANN is exposed to input. An appropriate desired/target ANN output is computed based on the reinforcement signal and learning commences with each vector sample. Learning continues until a desired level of accuracy is achieved.

Variants of the feedforward ANN seen in Figure 3.2 exists. For example, feedback connections can be introduced in order to introduce a form of memory into the ANN. The choice of ANN architecture is typically application dependant and experimentation is typically required in order to find a suitable architecture.

3.2 Acquiring Data

In Stage 1, the game-pad is used to manually drive Khepera around the environment, which is shown in Figure 2.3, for the task of wall following. The challenging aspect is controlling Khepera's distance to the walls. The set-point wall distance is chosen to be 11.5cm, which is measured from the center of Khepera to the nearest wall. This distance is chosen in order to minimize undesired saturation or under stimulation of Khepera's proximity sensors (refer to Figure 2.5 for proximity sensor space plot). A total of eight minutes of data is collected for Khepera performing wall following in each of the clockwise and counter-clockwise directions. With a sampling time of about 88ms, a total of 10881 data vectors are acquired. These vectors are used to formulate training and testing data sets using 67% and 33% of the total vectors, respectively. It is assumed that this dataset is large enough to adequately train an ANN. This assumption is validated in Section 3.3 and Section 3.4.

An important observation is made upon viewing the data set: Khepera's sensors are not able to perceive anything when driving around sharp turns such as the 135 degree turn seen in Figure 2.3 (i.e. all sensors report a reading of 0 or near 0). This leads to difficulties in wall following around corners, which has also been seen in [51]. As a direct result, if we desire to train Khepera to follow the desired trajectory, it seems reasonable to assert that memory must be introduced to the ANN in order to distinguish between left and right turns. Therefore, all time-discontinuities found in the training and testing sets are appropriately marked so that they are identified in the ANN training stage.

The training and testing data sets are organized such that they contain an equal number of data vectors obtained from Khepera moving in a clockwise and counter-clockwise orientation. The organization of the training and testing data sets is shown in Figure 3.3. A time-discontinuity is seen at the beginning of each new section. For example, the transition from the first counter-clockwise section to the clockwise section represents a time-discontinuity. The data in each of the training and testing sets are not normalized.

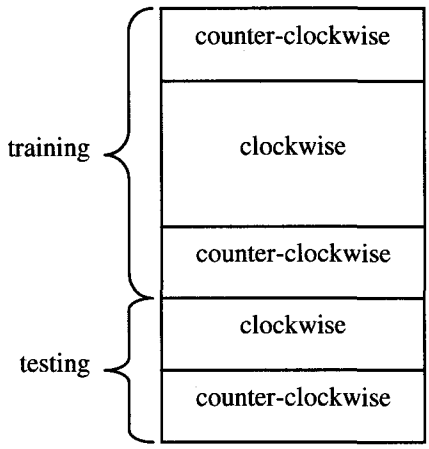


Figure 3.3 - Formulation of the training and testing data sets using the clockwise and counter-clockwise data.

3.3 ANN Architecture

A single-hidden layer backpropagation network is built using vanilla sigmoid neurons with bias. Two analog outputs are used to represent the left and right wheel velocities of the robot. Alternative controllers may output translational and steering velocity instead, which is done in part to implement a variable translational velocity controller in [52]. These two alternative approaches, however, are mathematically similar since the steering and translational velocities are linear functions of the left and right motor speeds.

ANN architecture design is a process involving trial and error. Fewer neurons are desired since it reduces training time and increases the overall generalization ability of the network, which prevents memorization. Conversely, an ANN whose architecture is too small will not be powerful enough to adequately learn the training data. With these tradeoffs in mind and plenty of trial and error, an architecture can be discovered that learns the training data and performs well on the testing data. Pruning and growing techniques can help. Pruning systematically removes weak connections from an arbitrarily large network until a sufficiently small trained network is obtained [53]. Growing starts with a tiny network and gradually increases in complexity until sufficient training performance is achieved [53].

The generalized ANN architecture, with memory, is shown in Figure 3.4. The most significant architecture degrees of freedom to explore are the number neurons in the hidden layer and the memory capacity of past inputs. Memory capacity considerations include the number of past inputs to be fed into the hidden layer and the time delay of each past input.

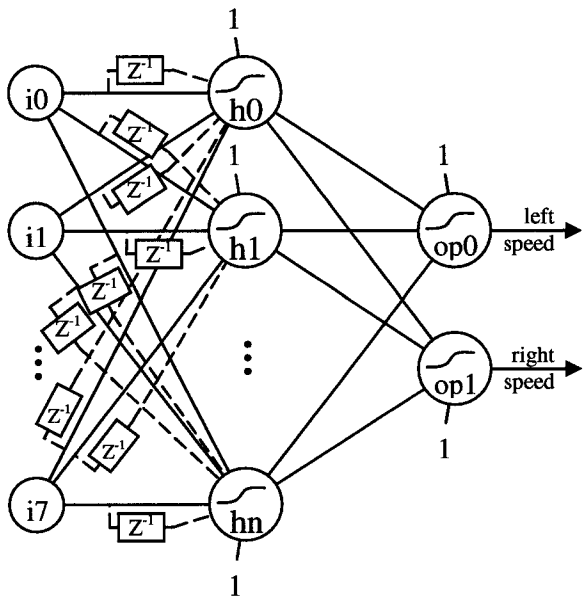


Figure 3.4 – Generalized fully connected two-layered backpropagation ANN with biased sigmoid neurons. Delay blocks permit one past input vector to be fed into the ANN. Each additional past input vector requires additional delay blocks.

By inspection of the acquired data, it is evident that Khepera requires up to 2500ms to complete a 135 degree arc. Therefore, the memory is designed to encapsulate up to 2816ms of previous sensor data, which encompasses 32 previous input vectors with the cycle time of

88ms. It is not feasible to allow all 32 past input vectors to serve as inputs to the ANN, yet it is evident that at least one past input is required. Experimentation with the number of past inputs to utilize and the number of hidden neurons is performed with careful consideration of the number of connection weights required.

A commonly used design rule is to allow at least ten times the number of training vectors as ANN connection weights [54]. Given the size of the acquired training set, the ANN architecture is limited to about 700 connection weights. For the ANN architecture of Figure 3.4, the number of connection weights is tallied as

$$n = h(ip+1)+op(h+1), \qquad [3.6]$$

where *ip* is the total number of inputs including past inputs, *h* is the number of hidden neurons, and *op* is the number of neurons in the output layer. While *op* is fixed, the other two parameters can be varied such that *n* is kept well below one tenth of the number of training vectors. With this restriction in mind, trail and error is used to obtain a suitable architecture. This is achieved by exploring numerous permutations of architecture and their associated ANN learning. The following table provides an example of ANN architectures explored:

Table 3.1 – Sample ANN Architectures Explored

Architecture	No. Layers	No. Hidden Neurons	Fully Connected?	No. Past Input Vectors	Time Between Input Vectors
(a)	2	12	yes	1	2288ms
(b)	2	18	yes	2	1408ms
(c)	3	12, 12	yes	1	2816ms
(d)	2	30	no[1]	4	704ms

Architectures with the lowest reporting performance index are deemed most accurate and appropriate for the wall follow objective. The performance index is quantified over N vectors as

$$Q = \sum_{i=1}^{N} \frac{1}{2}(left\_target_i - left\_y_i)^2 + (right\_target_i - right\_y_i)^2, \qquad [3.7]$$

where *left_target* and *right_target* are the respective left and right target motor speeds, *left_y* and *right_y* are the respective left and right ANN motor speed outputs. Note that the value for N is different for the training and testing sets. Learning is performed for the four architectures in Table 3.1 with 25 trials and the results are tabulated in Table 3.2.

[1] 15 hidden layer neurons do not have connections with the current input vector while the other 15 hidden neurons do not have connections to the past input vectors. Otherwise, the network is fully connected.

Table 3.2 – Sample ANN Architectures Performance Index

Architecture	Performance Index					
	Training			Testing		
	Q_{min}	Q_{mean}	Q_{σ}	Q_{min}	Q_{mean}	Q_{σ}
(a)	53.6	55.9	1.4	30.7	39.0	7.6
(b)	51.1	54.7	1.7	29.6	37.0	7.1
(c)	50.7	53.8	1.5	33.4	44.5	6.0
(d)	53.8	55.6	1.1	39.5	44.6	4.4

Architectures (a) and (b) have similar outcomes as reported by Table 3.2. However, after experimentation with the physical robot, it is clear that the additional past input vector is required. Without the second past input vector the robot is observed to improperly make the challenging 135 degree turns, which causes the entire trajectory to fail. The complex architecture (c) with two hidden layers does not seem to be a winner and it requires more time to train. Architecture (d) is even more complex and does not fair well. After numerous experimentation, architecture (b) is chosen since it achieves a low performance index, contains 488 connection weights, which is reasonable as per the size of the training set, and fairs well on the physical robot.

Another example of architecture experimentation is shown in Figure 3.5 and Figure 3.6. Here the number of past input vectors and the number of hidden neurons degrees of freedom are explored. Each architecture undergoes a total of ten learning trials and varying results are reported due to the randomly initialized connection weights (details of ANN learning and connection weight initialization follows in Section 3.4). It is convenient to summarize the trials by reporting the minimum and mean performance indices for each architecture.

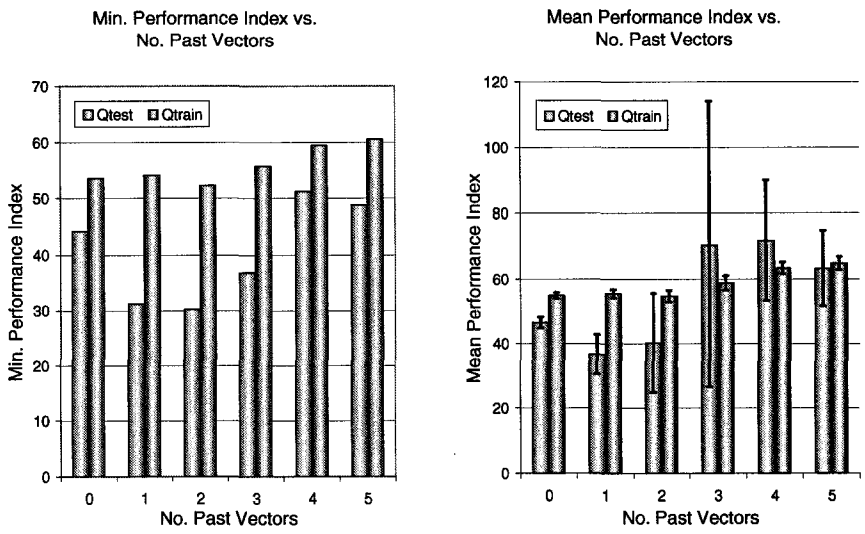


Figure 3.5 – An ANN architecture with 18 hidden neurons and 2816ms of past input duration is explored with varying number of past input vectors. The minimum testing and training set performance index is reported for 0 to 5 past input vectors with 2 past input vectors achieving the best results (left). Similarly, the mean testing and training set performance index is reported with error bars representing plus/minus one standard deviation (right).

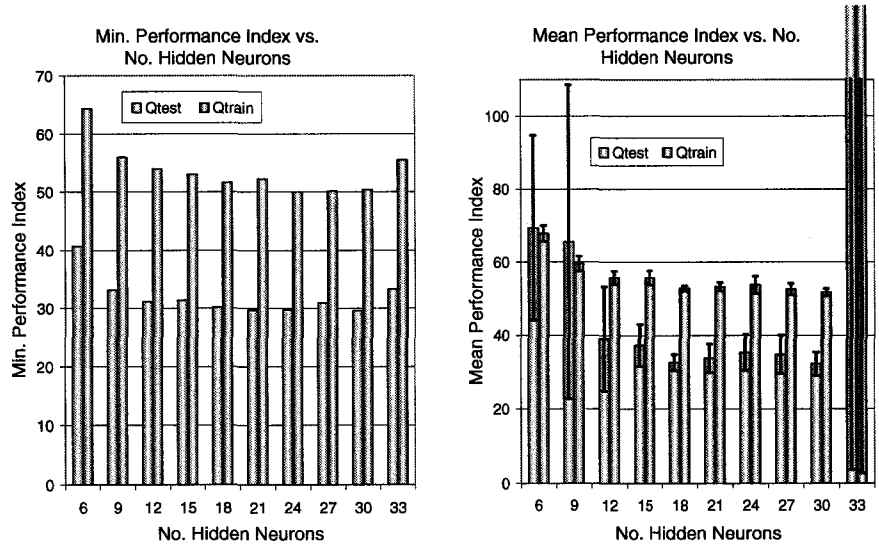


Figure 3.6 – ANN architecture with 2816ms of past input duration and 2 past input vectors is explored with varying number of hidden layer neurons in a single hidden layer network. The minimum testing and training set performance index is reported for 6 to 33 hidden neurons (left). Similarly, the mean testing and training set performance index is reported with error bars representing plus/minus one standard deviation (right). Choosing an appropriate number of hidden neurons is not so clear here since 12 to 30 hidden neurons achieve similar results.

The choice of architecture is not always clear because it is possible for different architectures to perform equally well. It is well known that a compromise exists between network performance and network complexity [55]. Architecture (b) in Table 3.1 is chosen as the compromise. Details of ANN learning follows in the next section.

3.4 ANN Learning

The backpropagation learning algorithm aims to adjust connection weights with a gradient descent of error. This generalized delta learning method is very powerful in discovering local minimums of error. However, the algorithm offers no guarantee of discovering the global error minimum and worse yet cannot give any indication on whether or not the global error minimum has been found. Therefore, numerous experimentation trials with random initial connection weights are often required in order to achieve better training and testing data modeling with little error. The backpropagation algorithm aims to minimize error calculated by the least squares error function. Alternative approaches include using the cross-entropy error, which has been shown to improve network convergence in some cases [56]. Improving the convergence is a worthwhile endeavor—especially since numerous architecture and training degrees of freedoms are to be explored. However, the simplicity and proven utility of the least squares method makes it the chosen error function for ANN learning.

In the situation where the training set is time continuous, each progressive vector may be similar to the previous vector. This could present a problem when per-pattern learning is implemented. The risk is that the ANN may learn similar vectors while poorly learning the entire training set [15]. Randomizing the order of the training data can help; however, this could be haphazard since memory is present in the system. Experimentation is performed with both "per-pattern" and per-epoch learning modes. It is decided that per-pattern learning is the better choice since it seems to achieve lower converging testing data error for the specific application at hand. Furthermore, the choice of learning rate is difficult with the per-epoch learning. Since numerous data vectors are used, the learning rate must be kept small in order to avoid diverging network error. The range of appropriate values that can be used for the small learning rate is limited and is a function of the number of training vectors. The restrictions imposed on the learning rate make per-epoch learning impractical.

The learning rate and momentum term are assigned by experimentation using the ranges 0.01 to 5.0 and 0.05 to 0.95, respectively. A high learning rate is desired for faster convergence; however, the network performance index may oscillate and diverge with the learning rate set too high. The momentum term is utilized to help the learning process skip over local minimums in hopes of discovering the global minimum of network error. Numerous values are attempted and it is discovered that a learning rate of c=1 and a momentum term of n=0.8 suffice for the application at hand.

The minimum and maximum number of algorithm iterations is set to 40 and 150, respectively. These parameters are assigned so that erroneous increases in the testing performance index are ignored during the early algorithm iterations while unnecessary late algorithm iterations are avoided. The algorithm will exit to prevent overtraining if the testing error increases by more than 1% of the lowest recorded value in the trial. Learning does not occur where time discontinuities are marked in the training data since past inputs are nonexistent. The results are widely varied between different architectures and even between different trials of the same architecture. Architecture (b) in Table 3.1 is investigated further: while rather weak, a certain correlation seems to exist between the testing error and training error as shown in Figure 3.7.

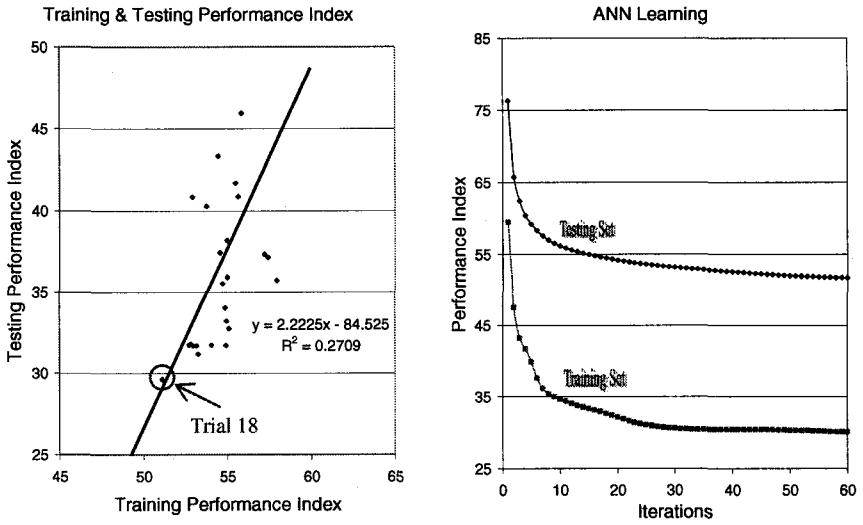


Figure 3.7 – Backpropagation algorithm is applied with 25 trails for the ANN architecture with 18 hidden layer neurons and a total of 24 inputs (current input vector plus two past input vectors). Trial 18 seems to model the data most successfully since it has achieves the lowest testing performance index (left). Testing and training performance index spanning 60 iterations (although a total of 150 iterations are completed) is plotted for trial number 18 (right).

Grayscale representation of the ANN connection weights are produced for the network before and after the training in trial number 18. This is done by arbitrarily scaling the weight connections by 32 and centering around 127, which is 50% grey. Any pixel values less than 0 or greater than 255 are cropped to 0 (black) and 255 (white), respectively. A total of 488 pixels are produced, which can roughly populate a 22x22 image as shown in Figure 3.8. Four pixels are omitted from this comparison.

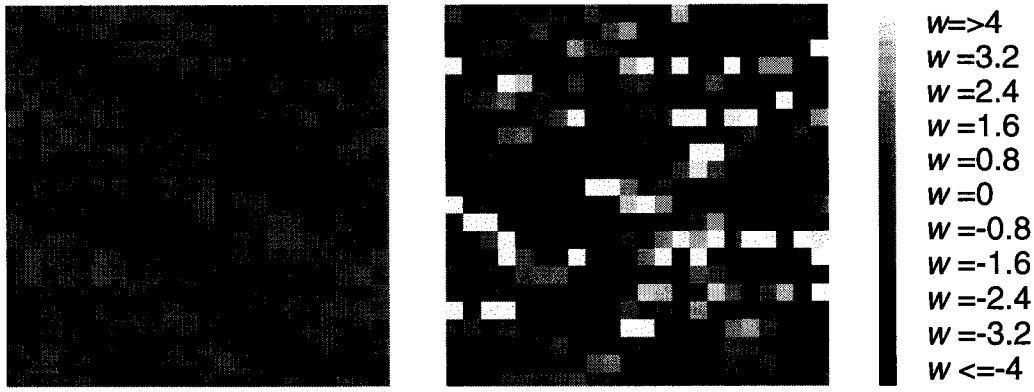


Figure 3.8 – Grayscale representation of 484 connection weights before (left) and after (right) training for trial number 18. The images indicate that the connection weights are more declared after training.

The connection weights for the two delayed input vectors have very similar distributions, as shown in Table 3.3. The connection weights associated with the input vector with zero delay has a greater distribution, which implies that the ANN output is most greatly affected by the current (i.e. zero delay) input vector.

Table 3.3 – ANN connection weights

Connections	connection weights				
	number	mean	min	max	std_dev
Input, No Delay	162	-	-	15.34	3.81
Input, One Delay	144	-	-8.82	7.71	2.73
Input, Two Delay	144	-	-8.64	7.29	2.74
Output Layer	38	-	-3.25	1.29	0.96

While there is no guarantee that the global error minimum is discovered for a given architecture, it assumed that 25 trials results in a sufficiently well trained network. This assumption it validated in the proceeding sections when performance testing and evaluation on the physical robot is presented. It is certainly a possibility that a trained ANN can produce poor results on the physical robot. In this case, it may be necessary to make revisions to previous steps completed—such as learning, architecture, or even data collection. Iterating through these stages, however, is very time consuming—especially if

the iterations include revisiting the data collection stage. This process is analogues to the waterfall model of software engineering whereby optimism is seen in the belief that previous stages are complete and (ideally) do not require any re-work.

3.5 Behavior Generalization

The trained ANN is implemented on Khepera. Implementation of the ANN requires one small consideration: initialization. In particular, the memory must be initialized appropriately depending on the initial placement of Khepera in its environment. Khepra is placed in its environment exactly 11.5cm from a wall and with parallel orientation to the wall. Resulting wall follow trajectories in both the clockwise and counter-clockwise directions are shown in Figure 3.9.

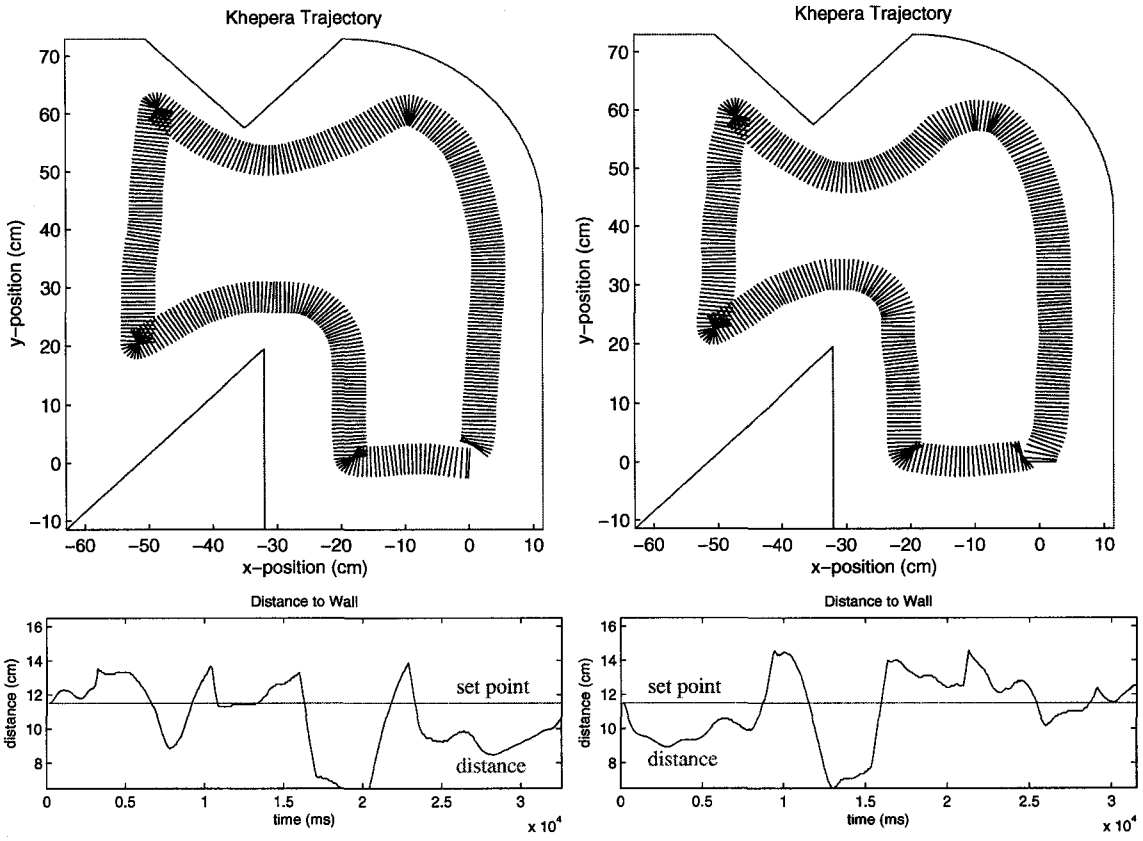


Figure 3.9 – Khepera is implemented with the trained ANN and produces trajectories in the clockwise (top left) and counter-clockwise (top right) directions for the training environment used (define as world1). Each data point is drawn as a line representing an imaginary axis connecting the two wheels. Measured minimum Euclidean distance to the wall is plotted over the duration of one complete lap in the clockwise (bottom left) and counter-clockwise (bottom right) directions along with the set point of 11.5cm. The calculated error per sample average, or mean error, is 1.8cm and 1.7cm for the respective clockwise and counterclockwise directions.

It should be noted that measurement error is prevalent in the method used to acquire trajectory data. Error is cumulative such that each successive trajectory point contains all error from previous trajectory points plus any new measurement error. Nonetheless, the acquired data can be used to quantify *short* trajectory performance with some degree of precision. Longer trajectories, such as multiple laps around an environment, would result in vast discrepancies between the measured trajectory and the actual Khepera trajectory. This is mostly attributed to wheel slipping, which is predominant when the Khepera serial communication line becomes tangled.

While all training data acquired is limited to the single environment of Figure 3.9, it is worthwhile to experiment with varying environments in order to explore the ability of the synthesized behavior to generalize. A selection of the numerous possible wall geometries are presented in this section that demonstrate the ANNs ability to generalize in different environments. For each wall geometry, experimentation is performed in both the clockwise and counter-clockwise direction, which yields similar results. Therefore, only the trajectory results from the clockwise direction are plotted.

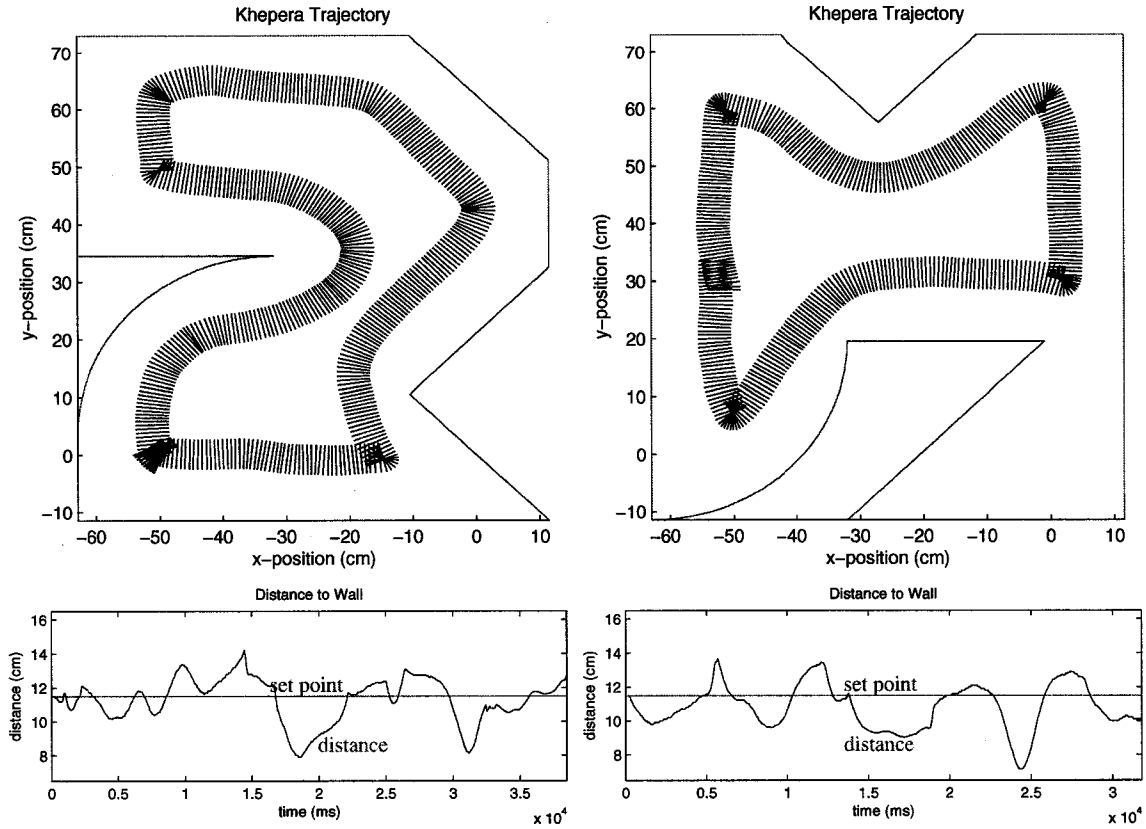


Figure 3.10 – The trained ANN is tested on world2 (left) and world3 (right) and clockwise trajectories are plotted. Mean trajectory error is 1.0cm and 1.2cm the respective world2 and world3 clockwise trajectories.

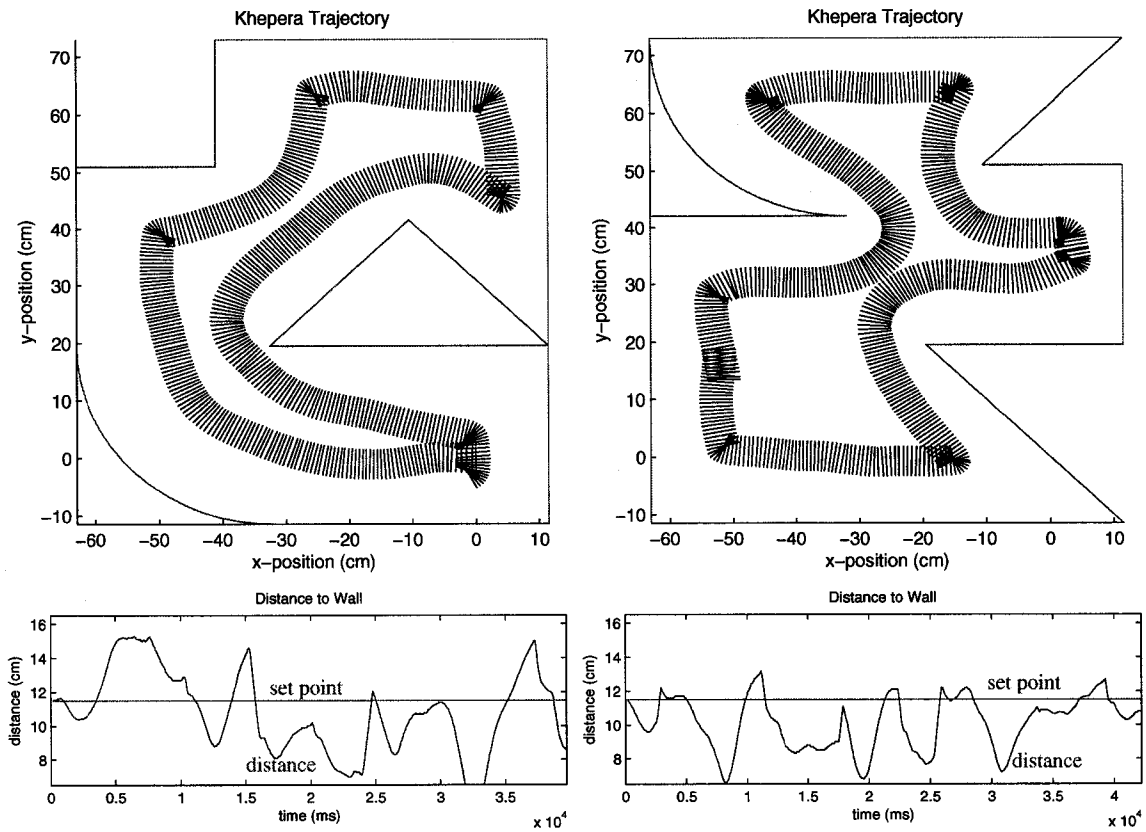


Figure 3.11 – The trained ANN is tested on world4 (left) and world5 (right) and clockwise trajectories are plotted. Mean trajectory error is 2.1cm and 1.6cm the respective world4 and world5 clockwise trajectories.

It is worth noting that Khepera would not enter the bottom right quadrant of world3, which is partially blocked by a wall and has only a 12.5cm opening. If the opening is increased enough, then Khepera enters the quadrant. Problems arise when an opening is approximately twice the wall follow distance (i.e. ~23cm) but does not widen as in the case of a corridor. In world6 shown below, Khepera is unsuccessful in completing a full rotation in either direction due to the corridor-like environment. In both cases, Khepera is initially placed near the bottom right corner. Experimentation with varying initial positions yield similar results. The corridor of world6 is significantly different than the wall geometries of world1. This difference is great enough so that the ANN generalization ability does not suffice. A cardinal rule of ANNs is that the training data must be representative of the desired behavior. World6 provides an example of this rule being broken.

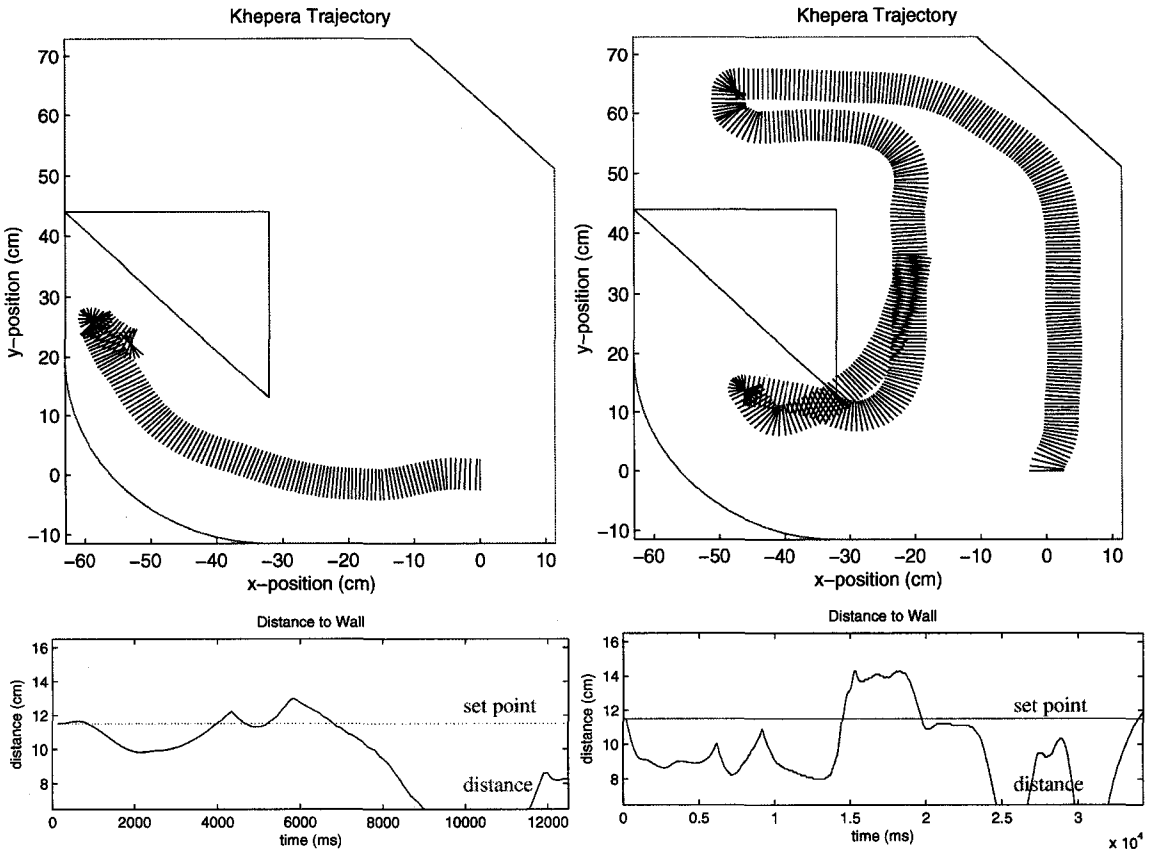


Figure 3.12 – Khepera fails to perform wall follow in world6. In the clockwise direction (left) Khepera becomes trapped while in the counterclockwise direction (right) Khepera turns around and follows the wrong side of the corridor.

The trained ANN performs very well in the simplistic circular world7. An arc wall is added to world7 such that the arc roughly coincides with the observed trajectories in world7. The new world (world8) proves to be very difficult for the trained ANN because the robot is not able to perceive the additional arc very well until it is almost touching it. The robot soon recovers and maintains a greater distance to the wall. The trained ANN is considered to fail the fall follow objective in world8 because, in some trials, the robot touches the arc wall prior to recovering to a reasonable wall distance. It is questionable whether or not a better trained ANN could adequately perform wall following on world8 because the infra-red proximity sensors limit the robots perception.

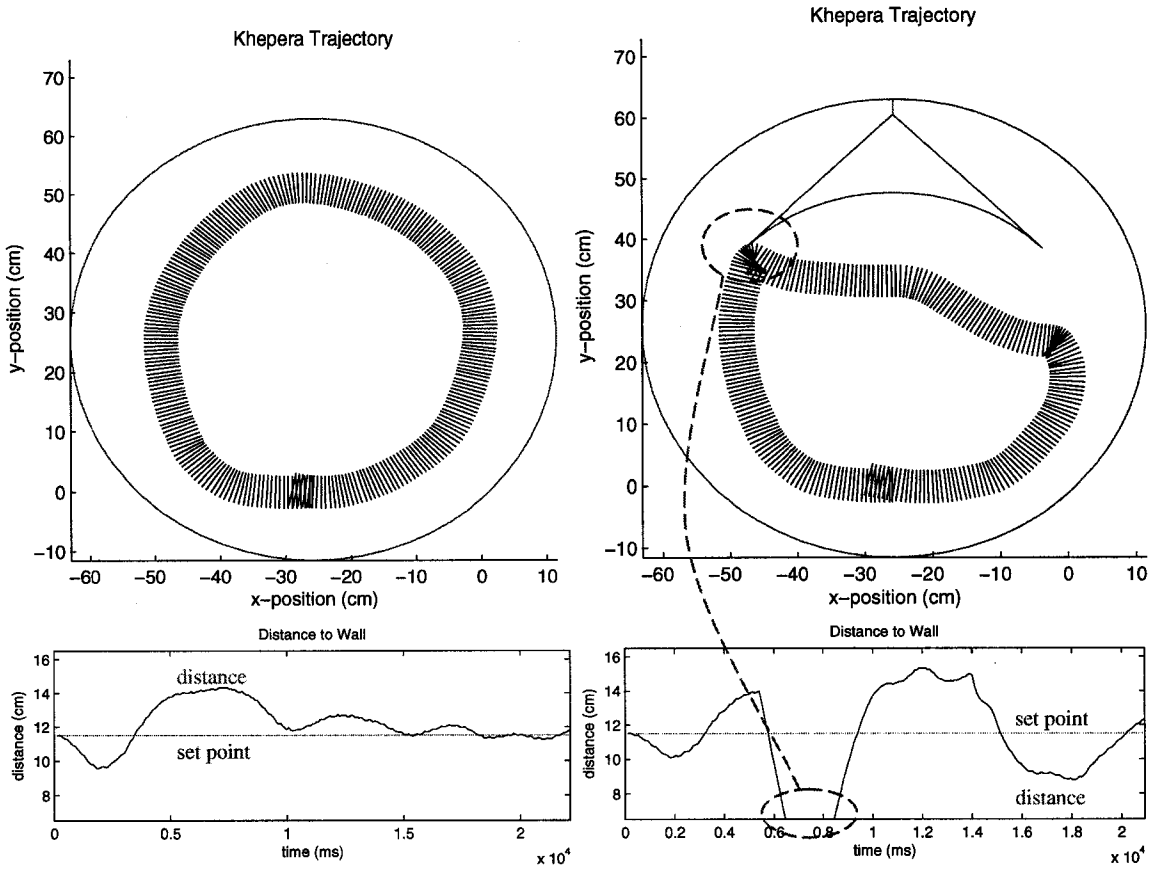


Figure 3.13 - The trained ANN is tested on world7 (left) and world8 (right) and clockwise trajectories are plotted. Mean trajectory error is 1.0cm and 2.4cm the respective world2 and world3 clockwise trajectories. An exceptionally large error is observed upon Khepera approaching the arc, which is marked in the figure.

Trajectory error between the minimum Euclidean wall distance and the set point of 11.5cm is quantized by the absolute difference between the two. The mean trajectory error, which is reported in the previous figure captions (with exception to world 6), is computed as well as error maximum, and error standard deviation for each trajectory in both clockwise and counterclockwise directions. Overall error metrics are computed for each world as an average of the metrics computed for the clockwise and counterclockwise directions. The following table and plot summarize the results:

Table 3.4 – Tabulated trajectory error

World	ANN Error (cm)								
	Clockwise			CounterClockwise			Overall ANN Error (cm)		
	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ
1	6.8	1.8	1.5	5.1	1.7	1.1	5.9	1.8	1.3
2	3.6	1.0	0.8	4.2	1.3	1.0	3.9	1.2	0.9
3	4.4	1.2	0.9	2.7	1.0	0.7	3.5	1.1	0.8
4	7.5	2.1	1.5	5.6	1.3	1.2	6.5	1.7	1.4
5	4.9	1.6	1.3	3.2	1.0	0.8	4.1	1.3	1.1
6	fail	fail	fail	fail	fail	fail	fail	fail	fail
7	2.8	1.0	0.9	3.5	1.3	1.0	3.2	1.2	0.9
8	fail	fail	fail	fail	fail	fail	fail	fail	fail

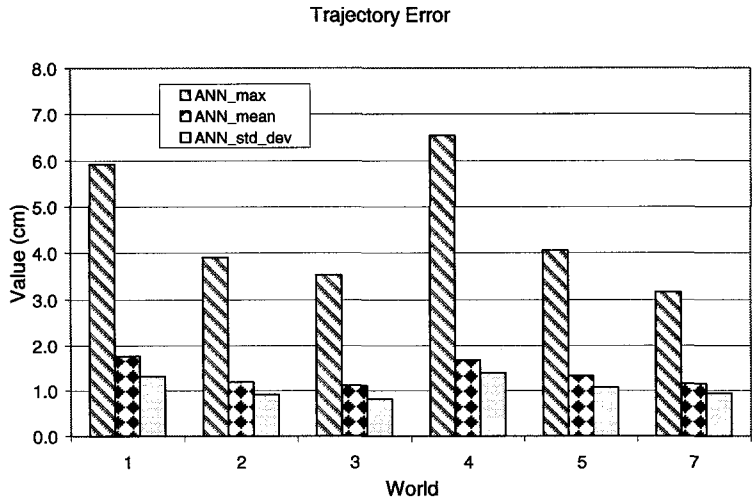


Figure 3.14 – Plotted overall trajectory error data for worlds 1 through 5 and world7 (results from world6 and world8 are not included here). Trajectory errors of worlds 2 through 5 are in the vicinity to those recorded in world 1. This suggests that the ANN is capable generalizing and has not simply memorized world 1 in the training phase.

Despite the observed shortcomings in world6 and world8, the trained ANN architecture is a feasible controller for synthesizing the wall follow objective on a wide variety of wall geometries. Utilizing a greater variety of training data (i.e. not limited to just world1) may result in improvements on a wide variety of wall geometries—especially on world6 and perhaps on world8.

3.6 Behavior Performance Scaling

When manually controlling Khepera (i.e. via gamepad) there exists limits as to how fast Khepera can be driven without loosing the desired trajectory. Any human operator is limited in terms of his/her reflexes and eye to hand coordination. An interesting comparison exists with the Khepera speed scalability between the human operator and the trained ANN. One

must be careful when scaling the output of the ANN-based controller since there is a time dependant memory element. For example, if the output of the ANN is doubled (i.e. Khepera is to have twice the speed), then the memory samples used must be only half as old as normal. With this consideration in mind, experimentation can be performed.

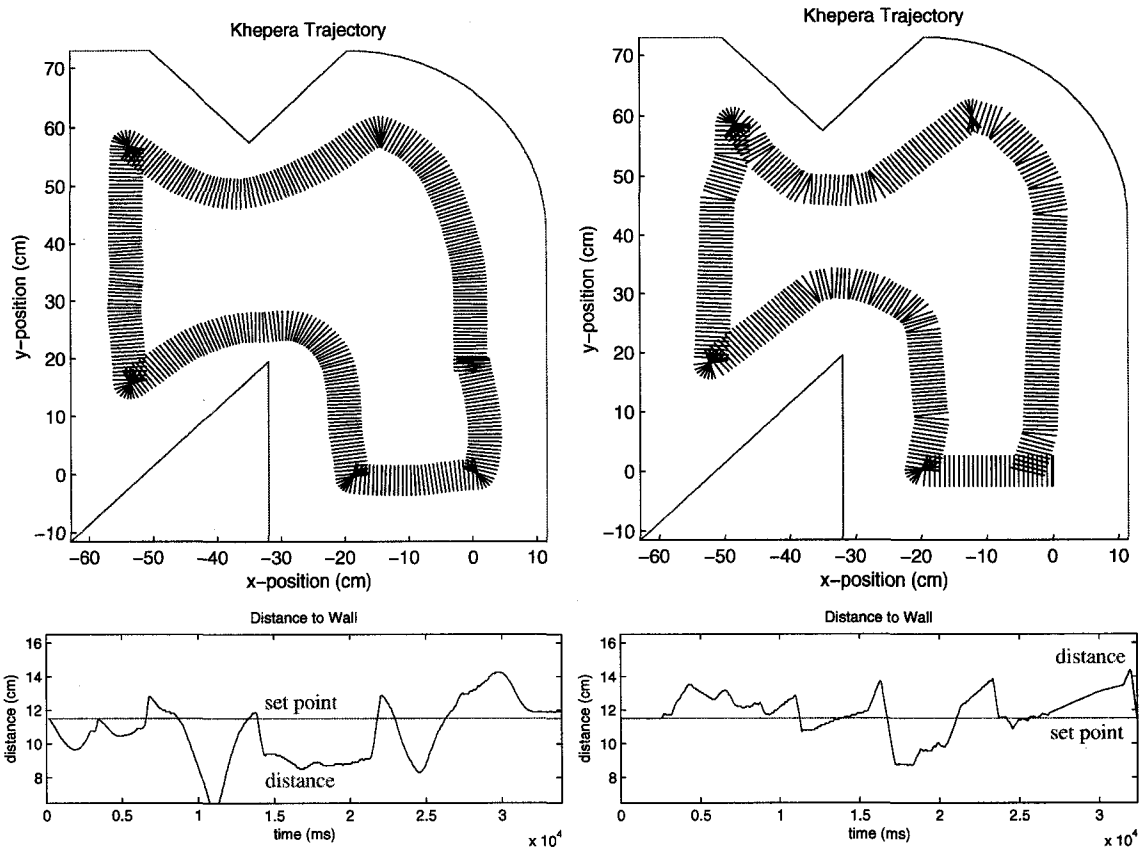


Figure 3.15 - The trained ANN with unity output scaling performs with a mean error of 1.6cm (left). Khepera is manually driven at the same speed and achieves a mean error of 1.0cm (right).

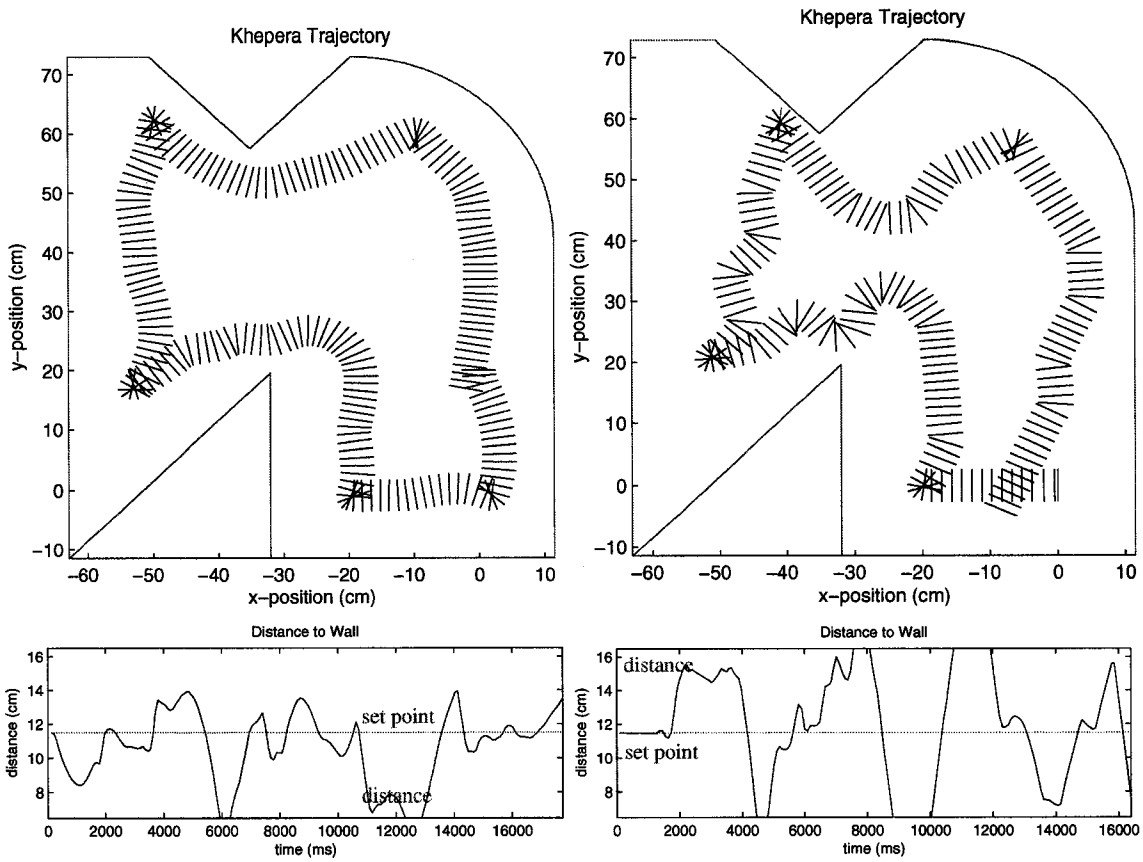


Figure 3.16 – The output of the ANN is scaled by a factor of two and still performs well with a measured mean error of 1.7cm (left). The human operator results at the higher speed are poorer with a mean error of 3.0cm (right).

Performance metrics are tabulated for the ANN and human operator over numerous speeds using world1 as a benchmark. For this benchmark, the term *fail* applies to trajectories with distance errors in excess of 13cm at any time, or fails by other means to accomplish the task (e.g. stopping, turning around, touching a wall, etc.).

Table 3.5 – Tabulated ANN and teleoperation (human) error

Speed (7.81cm/s)	Clockwise						Counter Clockwise					
	ANN Error (cm)			Teleoperation Error (cm)			ANN Error (cm)			Teleoperation Error (cm)		
	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ
0.5	5.8	1.6	1.2	2.7	1.0	0.7	3.9	1.4	1.0	4.9	1.4	1.1
1.0	5.4	1.6	1.2	2.9	1.0	0.8	2.5	0.8	0.6	7.3	2.2	1.9
1.5	5.5	1.4	1.1	3.9	1.2	0.9	6.4	2.0	1.5	8.7	2.6	2.0
2.0	5.9	1.7	1.5	8.7	3.0	2.4	8.3	2.1	1.6	10.1	2.5	2.3
2.5	12.3	2.5	2.5	fail	fail	fail	fail	fail	fail	10.0	2.5	2.5
3.0	fail	fail	fail	fail	fail	fail	fail	fail	fail	fail	fail	fail

Table 3.5 con't

Speed (7.81cm/s)	Overall (clkwise+cntclk)					
	ANN Error (cm)			Teleoperation Error (cm)		
	e_{max}	$\bar{e}$	σ	e_{max}	$\bar{e}$	σ
0.5	4.9	1.5	1.1	3.8	1.2	0.9
1.0	4.0	1.2	0.9	5.1	1.6	1.3
1.5	5.9	1.7	1.3	6.3	1.9	1.5
2.0	7.1	1.9	1.6	9.4	2.7	2.3

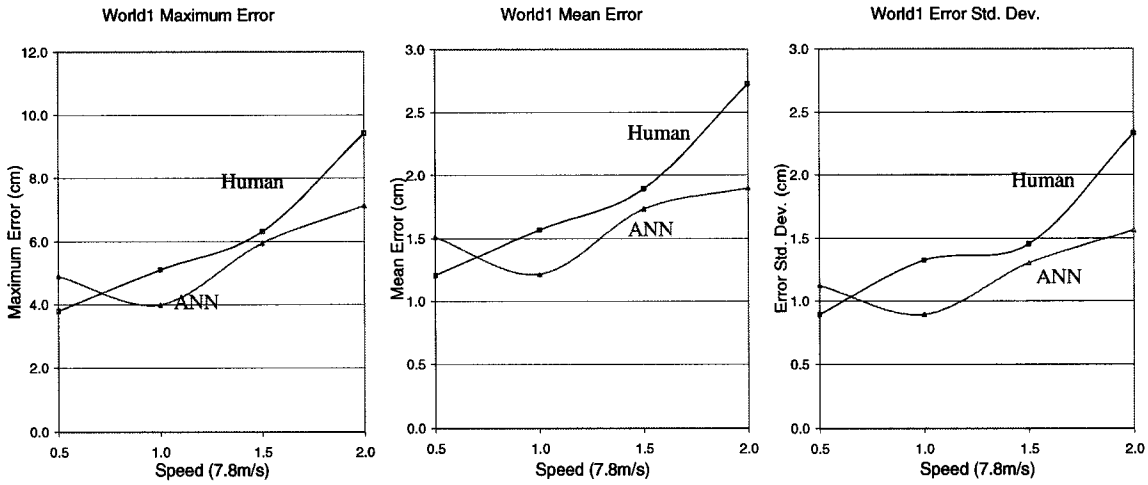


Figure 3.17 - Plotted trajectory error data contrasting the ANN performance and the human performance over a range of speeds. The ANN is able to perform equally well as the human operator—especially at higher speeds (i.e. 2x 7.8m/s).

It is imperative to emphasize that measurement error, which is largely due to slippage of Khepera's wheels, become significant at higher speeds. However, since the trajectories resulting from the ANN and human operator are acquired and evaluated by the same means, it is assumed that errors in measurements affect them equally. Therefore, it is assumed that Table 3.5 can serve as a valid benchmark in comparing the ANN and human operator. With this in mind, it is evident that the ANN can outperform the human operator at higher speeds—especially at 2.0x7.81cm/s. This may be attributed to the relatively quick cycle time of the ANN, which is 88ms. By comparison, the human operator is subjected to personal reaction time when using the game pad in addition to the 88ms cycle time of Khepera. Therefore, it seems reasonable to infer that the ANN performance is more scalable than the human operator.

More complex architectures with more layers, neurons, and past inputs can be attempted and may achieve better results than seen in this paper. However, trial and error methodology utilized for training ANNs would be extremely time consuming when exploring numerous permutations of architectures. It would be interesting to apply a genetic

algorithm (GA) to explore architectures in order to discover better fit architectures quantified by the testing set error. Naturally one must question whether or not the overhead surpasses the benefits of utilizing a GA—especially when acceptable results are quickly obtainable with the trial and error methodology used.

3.7 Conclusion

A methodology is presented that exploits encapsulated implicit knowledge into an ANN-based controller for autonomous robotic wall following. The ANN-based controller is trained using the backpropagation algorithm. Numerous quantitative measures are reported to critically evaluate the ANN-based controller. Results of this analysis indicate that the trained ANN is able to successfully generalize across a variety of different wall follow environments. Performance evaluation demonstrates that the ANN-based controller can achieve better wall follow control at higher speeds than a human operator. The proven ability to generalize and perform well makes the ANN-based controller a suitable design for the wall follow task.

Chapter 4

ANN Rule Extraction

Artificial neural networks (ANNs) are powerful computational models with proven learning and generalization ability. Their downfall resides in its black-box architecture in which encapsulated knowledge is embedded into the connection weights and is extremely difficult to perceive. It is desirable to enhance the readability of the embedded knowledge so that the knowledge can be verified by a human expert [24]. This chapter investigates the problem of extracting a comprehensive set of discrete rules from a trained complex ANN with continuous attributes (inputs) and continuous classification (outputs). Discretization and feature selection of ANN attributes is explored using the Chi2 algorithm. Additional discretization is explored on the ANN output classification space using a simple clustering algorithm. Discrete rule sets are encoded in a chromosome population and artificial evolution is simulated using a real-coded genetic algorithm (RCGA).

The remainder of this chapter is organized in sections. Section 4.1 introduces the proposed method. Section 4.2.1 reports results obtained from the iris plant problem. Section 4.2.2, 4.2.3, and 4.2.4, report results obtained when extracting rule sets from three different ANN architectures, which have all been trained for the problem of wall following with Khepera. Section 4.2.5 investigates the effects when the rule extraction method is re-applied

to a set of extracted rules. Section 4.3 provides a discussion of the approach taken and further research to be completed. Finally, Section 4.4 provides concluding remarks and summarizes the findings.

4.1 Proposed Method

The framework of the proposed method is summarized in Figure 4.1. This method is proposed for the problem of rule extraction from the ANN trained to perform wall following.

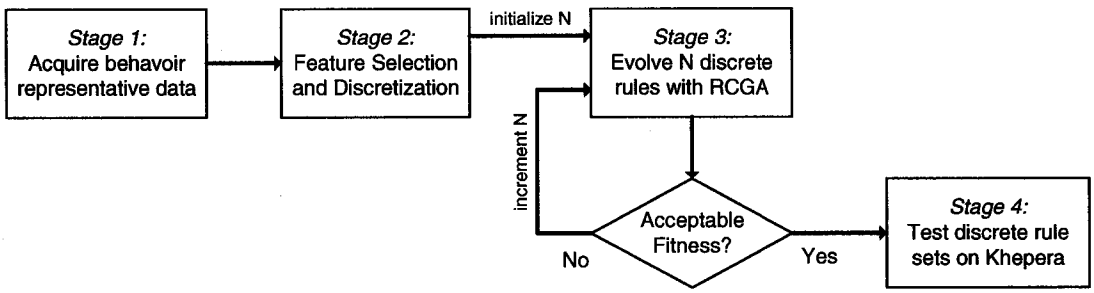


Figure 4.1 – Methodology for discrete rule set extraction Methodology for discrete rule set extraction. The number of rules in the rule set is incremented until acceptable chromosome fitness is achieved. Evolved discrete fuzzy rule sets are implemented and tested on the physical robot.

Since the methodology is to extract rules from an ANN without consideration of the architectural internals, behavioral-representative data is collected from the ANN in Stage 1. It is a hypothesis that data can be acquired to sufficiently represent the behavior of the ANN. In Stage 2, discretization and feature selection is applied to the dataset in order to reduce the search space for the RCGA.

A similar approach to the fuzzy-genetic rule extraction is taken in which a set of N discrete rules are evolved. The rule antecedents are comprised of attribute intervals with bounds generated from the Chi2 discretization algorithm. According to interval analysis theory (see [57] for a detailed overview of interval analysis), an interval can be thought of a fuzzy set with a rectangular membership function [58]. Therefore, the rule antecedents could be considered fuzzy antecedents with designed rectangular membership functions, as shown in Figure 4.2.

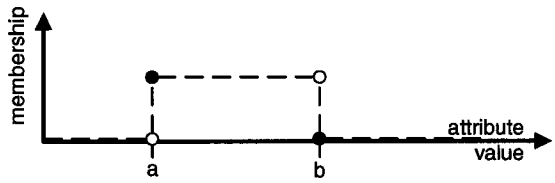


Figure 4.2 – Attribute membership function is rectangular with bounds *a* and *b* computed from the discretization process. A small modification is made from the reported rectangular membership function in [58] to set the membership at *b* to zero.

It is hoped that designing rectangular membership functions using discretization methods will help to extract *accurate* and *comprehensive* rule sets by using an evolutionary algorithm. This differs from previous methods in which fuzzy membership functions are typically not designed but rather assigned in a seemingly arbitrary manner. Although the proposed method is similar to previous fuzzy-genetic approaches, the discrete rule set is not termed fuzzy throughout the remainder of this paper since the interface between intervals and fuzzy sets remains controversial and could lead to confusion.

The rule set must be specified in such a way as to allow for expressive power while preserving simplicity in the rule set. The discrete fuzzy rule set is comprised of N rules, each with antecedents that check to see if attribute values reside inside discrete intervals derived from the Chi2 algorithm. The rule consequent represents the classification. Therefore, the discrete fuzzy rule set is based on the following rule format:

If attribute0 in $[d0_a\ d0_b)$ *and attribute1 in* $[d1_a\ d2_b)$ *and .... attributeK in* $[dK_a\ dK_b)$ *then class=*O_i

Details of the attribute discretization and the RCGA used to evolve the discrete fuzzy rule set proceed in the following two sections.

4.1.1 Discretization

Extracting comprehensive rules with accuracy and fidelity in mind is clearly very challenging when confronted with numerous continuous attributes and continuous classification. The difficulty lies in the fact that the number of rules increases exponentially with the dimensionality of the input space [46] and classification space. This provides clear motivation to reduce the dimensionality of the input and output space wherever possible. Clustering methods can be used to identify multi-dimensional antecedents, which could simplify the extracted rule set in terms of number of antecedents. However, these extracted rules with multi-dimensional antecedents would likely be incomprehensive. Instead, feature selection and discretization is explored with the Chi2 algorithm in order to reduce the input

space. A brief overview of the Chi2 discretization algorithm is presented here since it is a significant component of this thesis chapter. Further details can be found in [37].

4.1.1.1 Attributes: Chi2 Algorithm

The Chi2 algorithm is a statistically justified heuristic discretization and feature selection algorithm based on the χ^2 statistic. The algorithm starts by organizing the data into tables: one table per attribute. Within each table, the data is organized by the attribute intervals, and class frequencies are tallied. A sample table is shown in Table 4.1.

Table 4.1 – Data Sorted by Intervals

Int	Class Freq.			χ^2
4.3	1	0	0	0.2
4.4	3	0	0	0.2
4.5	1	0	0	0.2
4.6	4	0	0	0.2
4.7	2	0	0	0.2
4.8	5	0	0	2.04
4.9	4	1	1	1.78
5	8	2	0	0.381
5.1	8	1	0	0.51
7.3	0	0	1	0.2
7.4	0	0	1	0.2
7.6	0	0	1	0.2
7.7	0	0	4	0.2
7.9	0	0	1	

The algorithm calculates a χ^2 value between neighboring attribute intervals as

$$x^2 = \sum_{i=1}^{2}\sum_{j=1}^{k}\frac{(A_{ij} - E_{ij})^2}{E_{ij}}, \qquad [4.1]$$

where k is the number of classes, A_{ij} is the number of patterns in the i-th interval and j-th class, and E_{ij} is the expected frequency of A_{ij} given by

$$E_{ij} = \frac{\sum_{j=1}^{k} A_{ij} \times \sum_{i=1}^{2} A_{ij}}{\sum_{i=1}^{2} R_i}. \qquad [4.2]$$

The neighboring attribute intervals with the lowest χ^2 values are merged together. Attribute interval merging continues until further merging would increase the inconsistency rate past a preset threshold. Huan Liu and Rudy Setiono [37] demonstrate how the iris classification problem, with four continuous attributes, can be reduced to simply two attributes with four discrete values each. However, this does introduce a discretization error rate. The discretization error rate increases with the amount of attribute interval merging. This is clearly demonstrated with the Iris plant discretization example in Section 4.2.1.1.

4.1.1.2 Classification: K-Means

There is considerable motivation to discretize the classification space, in addition to attribute discretization, in order to reduce the search space for the RCGA. Therefore, a clustering algorithm is considered for classification space discretization where continuous classification exists. The K-means clustering algorithm is proposed since it is a simple method for unsupervised clustering.

The K-means clustering algorithm randomly assigns a data set into K disjoint subsets S_i. The objective is to minimize the sum of squares criterion,

$$C = \sum_{j=1}^{K} \sum_{n \in S_j} \left| x_n - \mu_j \right|^2 , \qquad [4.3]$$

where x_n is a vector representing the n-th data point and μ_j is the j-th cluster geometric centroid. In each algorithmic iteration, the data points are re-assigned to the cluster with the nearest geometric centroid. Iterations stop when no more re-assignments occur.

4.1.2 Rule Set Evolution

A real-coded genetic algorithm (RCGA) is proposed to explore the reduced search space in effort to evolve a discrete fuzzy rule set. RCGA is chosen since it posses advantages over binary-coded genetic algorithms (BCGA) for continuous variable domains, such as the wall follow problem under consideration in this paper. Refer to [59] for a detailed overview of RCGA and discussion of their benefits over BCGA.

Genetic algorithms are popular heuristic search methods for obtaining solutions to problems. They operate by encoding numerous random problem solutions into a population of chromosome. Genetic operators, such as crossover and mutation, are applied to the chromosomes population. Each chromosome is assigned a fitness, which is indicative of the utility of the solution, whereby better fit chromosomes are given better chances of survival in the selection stage. After numerous successive iterations, the chromosome population evolves to better fit solutions to the problem. Genetic algorithm details, in the context of the discrete rule extraction problem, are provided in the following sub-sections.

4.1.2.1 Chromosome Encoding

The encoding of the chromosome is a significant design stage since the chromosome models the form of the solution. Therefore, limitations introduced at this stage will have the effect of limiting the evolved solution. In the Michigan approach, each individual represents a

single rule and the set of rules is represented by the entire population. Alternatively, the Pittsburgh approach models each chromosome individual as a complete set of rules. The latter method, although computationally more expensive, allows additional optimization criteria in the fitness function for multi-objective optimization [45]. Therefore, the more flexible Pittsburgh approach is taken.

The individual chromosome encoding is shown in Figure 4.3 below.

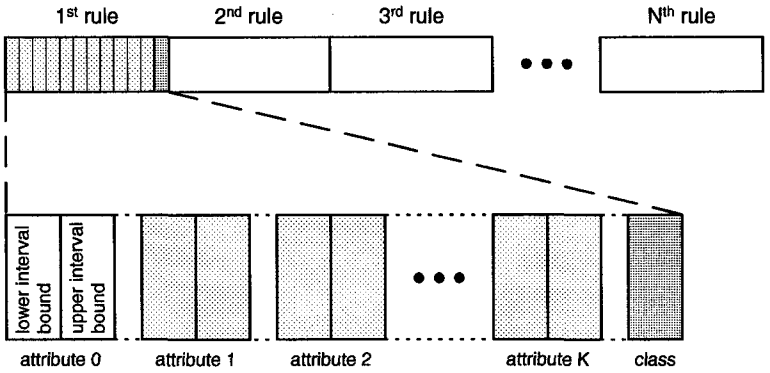


Figure 4.3 – Individual chromosome encoded with N discrete fuzzy rules. Each rule contains a lower and upper interval bound for each of the K attributes as well as a class.

The attribute interval and class gene values in Figure 4.3 are encoded from the set of attribute intervals and classes (clustered or non-clustered) generated from the discretization stage. Unique classes are marked by an integer value and used in the encoding process. A look-up table is implemented in order to decode the class back into the associated classification vector. The process of randomizing the genes (i.e. when initializing the population and when applying the mutation operator) involves randomly selecting a possible gene value, based on the gene restrictions. For sake of simplicity, each possible gene value is given equal probability of being encoded. A logical don't care is generated for attributes in which the lower interval bound is greater than or equal to the upper interval bound. When a rule does not care about one or many attributes, the rule comprehension is considered to increase since the number of antecedents is reduced.

The number of chromosome permutations for the RCGA is generalized as

$$P = (d_1 \cdot d_2 \cdots d_k)^{2N} \cdot l^N, \qquad [4.4]$$

where d is the number of attribute intervals, k is the number of attributes, l is the number of classes, and N is the number of rules. Note that it is possible for permutations to be

functionally the same (i.e. create the same rule set). It can be seen that the number of chromosome permutations decreases with fewer attribute intervals, classes, and rules.

4.1.2.2 Recombination, Selection

The double-crossover and mutation operators used in the recombination stage are illustrated below in Figure 4.4.

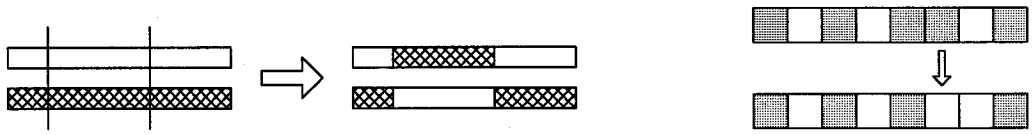


Figure 4.4 – Modified simple crossover (left) and random mutation (right) operators are used in the recombination stage.

Roulette wheel selection (proportional selection) is used for the selection stage. The individual fitness values are primarily assigned by considering the computed output of the rule set and the expected output obtained from the data. Computing the output of a fuzzy rule set involves defuzzification, which is typically performed by computing the centroid of the aggregated output fuzzy set. The output of the discrete rule set parallels this by computing the consequent arithmetic mean for all rules with true antecedents.

4.2 Experimentation

Although the primary experimentation is focused on the wall follow problem domain, it is useful to start with a much simpler pilot experiment with a well known dataset. Therefore, the iris plant dataset is used for initial experimentation. This dataset is used to test and verify correct operation of stages two and three of the methodology since the dataset is widely known and very simple. Stage one of Figure 4.1 is not used here since the iris dataset is used in place of data acquired from a trained ANN.

Subsequent discrete rule set extraction experimentation is applied to the complex trained ANN from Chapter 3 (Complex ANN), a scaled down trained ANN with fewer neurons and less memory (Moderate ANN), a simplistic trained ANN with few neurons and no memory (*Simple* ANN), and the extracted discrete rules obtained from the *Simple* ANN. The proposed method is to extract rule sets from each of these three trained ANNs with consideration of rule set *fidelity*, *accuracy*, and *comprehension*.

4.2.1 Iris Plant

The iris plant dataset is probably the most popular dataset found in the literature of pattern recognition. This dataset is available from [60]. The dataset contains four continuous attributes and three discrete classes.

4.2.1.1 Discretization

Since the dataset does not have continuous classification, class clustering is not considered and the classification look-up table is not needed. Attribute discretization, however, is performed with the Chi2 algorithm. Results of the discretization are shown in Table 4.2 and Table 4.3.

Attribute 0

Int	Class Freq.			χ^2
4.3	16	0	0	5.87
4.9	4	1	1	5.17
5	25	5	0	21.3
5.5	4	15	2	6.68
5.8	1	15	10	5.07
6.3	0	14	25	6.04
7.1	0	0	12	

(a)

Attribute 1

Int	Class Freq.			χ^2
2	0	3	1	2.36
2.3	1	6	0	9.98
2.5	0	18	18	5.8
2.9	1	7	2	4.57
3	6	8	12	1.64
3.1	12	7	12	4.57
3.4	9	1	2	1.8
3.5	6	0	0	1.9
3.6	9	0	3	1.9
3.9	6	0	0	

(b)

Attribute 2

Int	Class Freq.			χ^2
1	50	0	0	95
3	0	44	1	21.7
4.8	0	4	5	2.04
5	0	2	10	6.02
5.2	0	0	34	

(c)

Attribute 3

Int	Class Freq.			χ^2
0.1	50	0	0	104
1	0	49	5	78
1.8	0	1	45	

(d)

Table 4.2 – [left] The intervals, class frequencies, and χ^2 values for all four attributes in order (from top to bottom: (a) sepal length, (b) sepal width, (c) petal length, and (d) petal width) with the Chi2 algorithm error rate set to 0.

Attribute 0

Int	Class Freq.			χ^2
4.3	50	50	50	

(a)

Attribute 1

Int	Class Freq.			χ^2
2	50	50	50	

(b)

Attribute 2

Int	Class Freq.			χ^2
1	50	0	0	95
3	0	44	1	37.4
4.8	0	6	15	11
5.2	0	0	34	

(c)

Attribute 3

Int	Class Freq.			χ^2
0.1	50	0	0	104
1	0	49	5	78
1.8	0	1	45	

(d)

Table 4.3 – [above] The intervals, class frequencies, and χ^2 values for all four attributes in order (from top to bottom: (a) sepal length, (b) sepal width, (c) petal length, and (d) petal width) with the Chi2 algorithm error rate set to 0.03. Significantly more merging is seen—especially in the first two attributes.

By allowing even a small discretization error, the feature selection and discretization process can significantly reduce the attribute domain. This is desirable for heuristic search algorithms such as Genetic Algorithms because the search space is dramatically reduced. However, the introduction of discretization error at this stage may disallow optimal rules to be discovered. Therefore, the optimization process is left to the RCGA by choosing an error rate of 0.

4.2.1.2 Evolution

In this section, we present the results obtained when extracting discretized rules from the iris plant problem. It is imperative to mention that these extracted rules have non-continuous consequences and the output of the rule set is equal to the output of the rule(s) that classify the input without class confliction. When more than one rule classifies the input but conflict with one another, the rule set output is void. The first step is to design an appropriate fitness function.

Designing the fitness function is not trivial and may take some trial and error in order to achieve acceptable results. The chosen fitness function is

$$fitness = ramp(classification - 0.001 \times (avg\_num\_antecedants)) \quad [4.5]$$

This fitness function gives reward to correct classification percentage and gives a small penalty to containing higher average number of antecedents per rule. The penalty is introduced in effort to increase rule comprehension. The width of the attribute intervals are not considered to be an aspect of rule comprehension. The ramp function ensures that all fitness values are non-negative.

Evolution is performed with 1000 individuals with a probability of crossover and mutation of 80% and 1%, respectively, for 50 generations. A total of 150 vectors are utilized with 67% delegated as training vectors while the remaining 33% are testing vectors. The implication of dividing the dataset into training and testing sets is that when the evolved rule set is chosen, which is based on the testing set fitness, the training set performance may be quite poor. This could produce misleading results when reporting classification performance against other literature sources that do not split the dataset into training and testing sets. A summary of the results attained for one through five rules is presented in Table 4.4 below.

rules	train_fit	test_fit	train_class	test_class	complexity
1	0.32	0.35	0.32	0.35	0.00
2	0.66	0.69	0.66	0.69	1.00
3	0.92	1.00	0.92	1.00	1.00
4	0.92	1.00	0.92	1.00	1.00
5	0.92	1.00	0.92	1.00	1.00

Table 4.4 – Evolution results for 1 through 5 rules are summarized. Testing classification reaches 100% for rule sets with 3 rules or more. However, training classification remains lower at 92%.

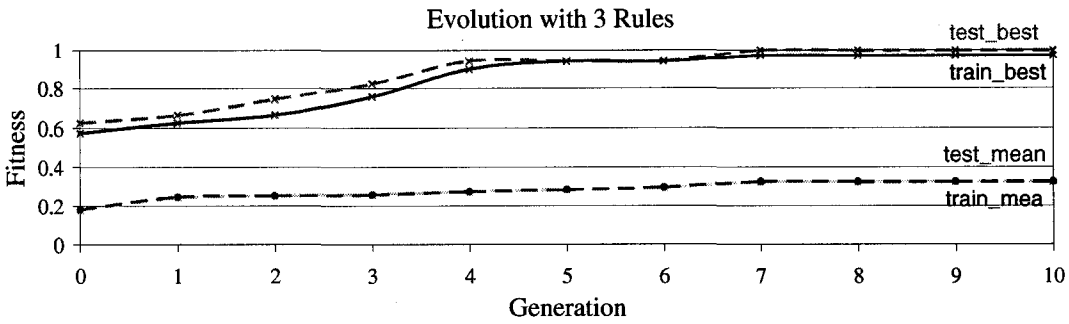


Figure 4.5 – Evolution with 3 rules. Very few generations are required to achieve the highest testing fitness when 1000 individuals are used. Most evolutions require only 10 generations.

The 3-rule set achieves a 100% classification performance with the testing set. Unfortunately, the training set performance is weaker at 92%. The 3 discrete rules are listed below in Figure 4.6.

```
If x2 in [1, 3) then y=1
If x2 in [3, 5.2) and x3 in [1, 1.8) then y=2
If x2 in [5, 11) then y=3
```

Figure 4.6 – Discrete rule set extracted with the highest testing fitness. The attributes are labeled *x0* through *x3* while the iris class is labeled *y*.

Higher performing rule sets are extracted when the dataset isn't split into training and testing sets. Examples of rule sets that achieve 96% and 97.3% classification are shown in Figure 4.7 and Figure 4.8, respectively.

```
If x3 in [0.1, 1) then y=1
If x3 in [1, 1.8) then y=2
If x3 in [1.8, 11) then y=3
```

```
If x2 in [1, 3) then y=1
If x3 in [1, 1.8) then y=2
If x3 in [1.8, 11) then y=3
```

```
If x3 in [0.1, 1) then y=1
If x2 in [3, 11) and x3 in [0.1, 1.8) then y=2
If x3 in [1.8, 11) then y=3
```

Figure 4.7 – Three examples of rule sets extracted without splitting the dataset into testing and training sets. Each rule set achieves 96% classification.

```
If x3 in [0.1, 1) then y=1
If x2 in [3, 5) and x3 in [1, 1.8) then y=2
If x2 in [5, 11) then y=3
If x3 in [1.8, 11) then y=3
```

```
If x3 in [0.1, 1) then y=1
If x2 in [3, 5.2) and x3 in [1, 1.8) then y=2
If x2 in [5.2, 11) then y=3
If x3 in [1.8, 11) then y=3
```

Figure 4.8 – Two examples of rule sets extracted without splitting the dataset into testing and training sets. Each rule set achieves 97.3% classification.

Interestingly, the classification performance does not seem to increase past 97.3% with the addition of more rules. Ishikawa is able to achieve 99.3% classification of the iris set with only three rules in [34]. It should be noted that the form of the rules differ from the form implemented here. More specifically, his antecedents are multivariate. Therefore, the proposed method for discrete rule extraction would not be able to achieve the rules that

Ishikawa was able to extract without modifying the rule form. Clearly, the design of the rule form encoded in the chromosome play a large role in setting out limitations in the rule extraction process.

4.2.2 Complex ANN

In this section, experimentation is applied to the *Complex* ANN architecture developed in Chapter 3. This ANN is termed complex since it contains more neurons and connections than the other two ANNs considered in Section 4.2.3 and Section 4.2.4.

4.2.2.1 Data Collection

Data collection is achieved by allowing Khepera with the *Complex* ANN to drive autonomously throughout the environment in Figure 2.3, in both clockwise and counterclockwise orientations. All data, namely the proximity sensor values and the motor speed values, are logged. This data is organized in a set of vectors containing all 24 required continuous attributes and 2 continuous classification outputs. A total of 717 vectors are acquired with 67% delegated as training vectors while the remaining 33% are testing vectors. It is imperative to mention that this dataset does not capture any erroneous rules, which likely exist in the ANN architecture. For example, the situation in which all proximity sensor values are saturated (i.e. Khepera is completely surrounded by walls in close proximity) is not represented in the data. Avoiding meaningless combination of input reduces the search space for the RCGA, which is done in [61]. In some applications is may be desired to extract rules for every possible permutation of current and past input. Such a challenging endeavor may be necessary for safety critical applications in which human lives depend on proper system operation [62]. This thesis, however, limits the rule extraction to *typical* operating conditions of Khepera performing wall following in the environment of Figure 2.3.

4.2.2.2 Discretization

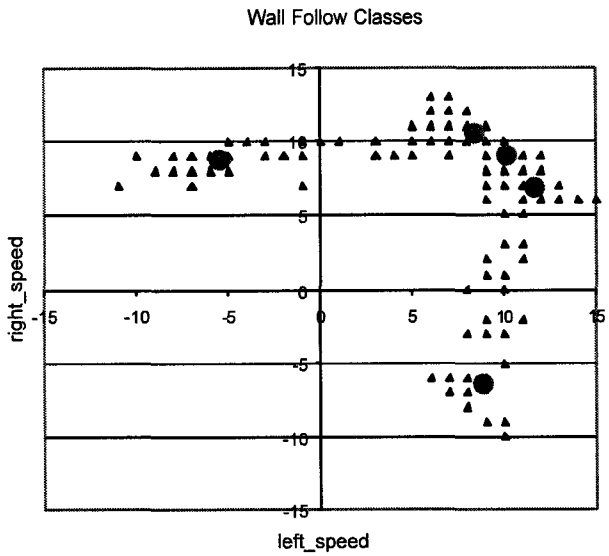
The Chi2 algorithm is applied to the complex wall follow dataset. A rather surprising and unfortunate outcome is that when the error rate is set to 0, only three attributes are discarded by the feature selection process. Fortunately, the number of intervals for each attribute is significantly reduced with ten being the greatest number of intervals in a single attribute.

Attribute	Intervals									
0	0	249	927							
1	0	8	695	700	896					
2	0	9								
3	0	22	558	798						
4	0	1	158	425						
5	0	1	7	213	432	779	953	1003		
6	0	1								
7	0	1	1023							
8	0									
9	0	1	321	703						
10	0	1								
11	0	22	84	174	228	503				
12	0	1	9	139	272	364	368	420	517	600
13	0	1	8	923						
14	0									
15	0	1								
16	0	160	1023							
17	0	1								
18	0									
19	0	1	42	63	264	701	707			
20	0	1	39	338	441	629				
21	0	1	1023							
22	0	1								
23	0	1	311							

Table 4.5 – Discretized intervals for the wall following data. Three attributes are not needed (attributes 8, 14, and 18 highlighted in grey). The number of intervals is significantly reduced from the ~1024 possible intervals.

The discretization intervals of Table 4.5 reduce the attribute space to be used in the RCGA search algorithm.

The classification space appears to have clustered regions. Therefore, by implementing a clustering algorithm, cluster centroids are identified in Figure 4.9. In this example, the K-means clustering algorithm is used to identify 5 clusters. The position of these five centroids are interesting because the represent distinct symmetrical operating states of Khepera: sharp turn left, subtle turn left, drive straight, subtle turn right, and sharp turn right.



left_speed	right_speed
8.8	-6.4
-5.4	8.7
11.6	6.7
8.3	10.5
10.1	9.0

Figure 4.9 – Wall follow classification space plotted with 5 centroids (left). Centroids are computed using the K-Means clustering algorithm and are tabulated (right).

Experimentation with and without classification clustering is performed. Results are reported in the following section.

4.2.2.3 Evolving Rule Set

Designing the individual fitness function for the wall follow problem is even more challenging than for the iris plant classification problem. Two important considerations are factored into the equation:

1. mean_squared_error: the mean squared Euclidean distance between the computed classification and the expected dassification. Refer to [63] for other similarity measures.
2. avg_num_antecedants: the average number of antecedents per rule.

These quantities are used to build the fitness function shown in Equation 2.

$$fitness = ramp\left(\frac{1}{\sum_{j=1}^{N}\sum_{i=1}^{2}(data\_class_{ji} - rule\_class_{ji})^2} - 0.001 \times avg\_num\_antecedants\right) \quad [4.6]$$

The fitness function attempts to minimize the discrepancy measurement between the expected output (from the dataset) and the actual output (from the rule set output). A small penalty term is added in effort to minimize the number of antecedents in the rule set. A ramp function ensures that the minimum chromosome fitness is non-negative.

Selecting an appropriate population size, mutation rate, and crossover rate is not trivial. It is clear from [64] that population sizes for genetic algorithms must be tailored towards the specific problem in order to achieve convergence towards the optimal solution. Alander proposes that the optimal population size is closely related to the length of the bitstring chromosome [65]. Haupt experimentally investigates optimal population sizes and mutation rates for a simple RCGA in [66] and determines that smaller populations with large mutation rates can be optimal. Nonetheless, it seems clear that experimentation with trial and error is required in order to attain acceptable results. As a starting point, the population is set to 1000 while crossover and mutation rates are set to 80% and 1%, respectively. Additional implementation details include an exit criterion is implemented such that evolution will stop when:

1. mean training fitness has not increased by more than 1% in 120 generations, or
2. mean training fitness has not increased by more than 1% in 30 generations and is within 5% of the maximum training fitness, or
3. 1000 generations have completed.

Experimentation is performed with rule sets of varying size—the number of rules is varied between one and forty. Experimentation is started without using classification space clustering. The rule sets attaining the highest testing fitness for each rule set size are recorded and summarized in Figure 4.10.

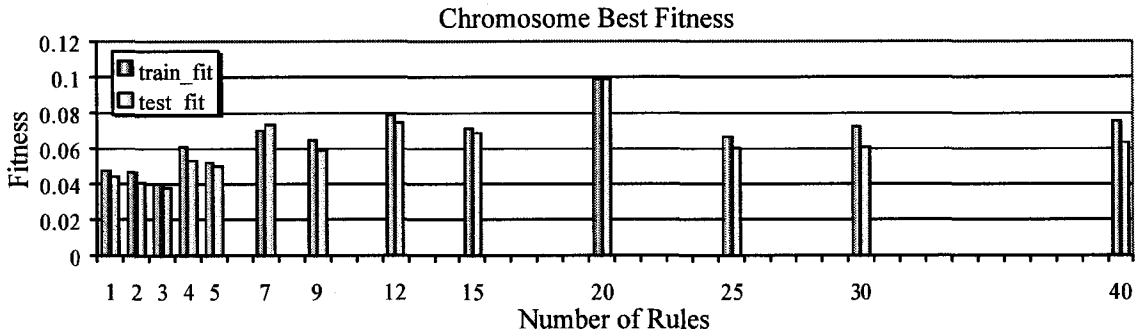


Figure 4.10 – Chromosome best fitness (both training and testing) is plotted for 1 through 40 rules without classification clustering. Experimentation with several rule sizes are omitted in order to reduce the computational cost.

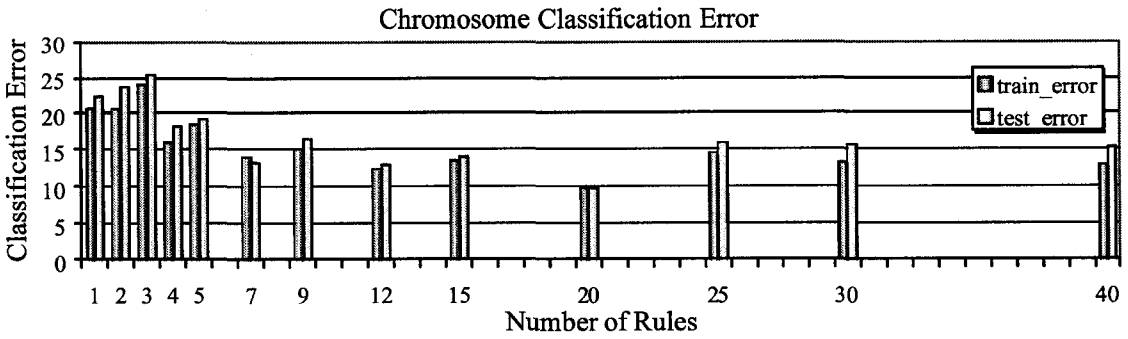


Figure 4.11 - Chromosome classification error (both training and testing) of the best fit chromosomes are plotted for 1 through 40 rules. Error tends to be reduced with additional rules. However, this relation is reportedly weak.

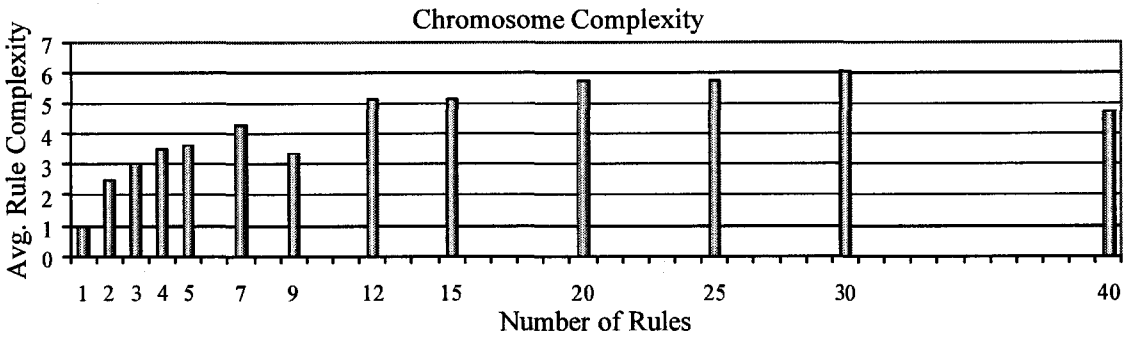


Figure 4.12 - Average rule complexity of the best fit chromosomes are plotted for 1 through 40 rules. The rule complexity is seen to increase with the number of rules.

The results are poor and the fitness values attained are quite low. Increasing the population size has the effect of increasing chromosome fitness values to higher levels. However, fitness values remain poor and detailed results are, therefore, omitted.

Attempts to discretize the classification space with the K-means clustering algorithm do not noticeably improve the fitness results, which are reported in Figure 4.13 below. This is an unfortunate result, which may be attributed to the fact that the RCGA search space is not significantly reduced by discretization of the classification space since the size of the attribute space is orders of magnitude greater than the size of the classification space. Therefore, it seems reasonable to believe that the effectiveness of classification space clustering is limited to certain problem domains where the size of the attribute space is not many orders of magnitude greater than the size of the classification space.

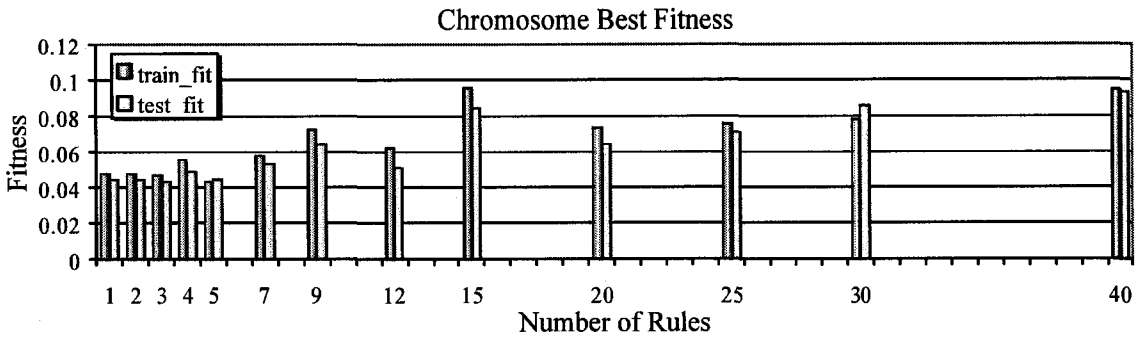


Figure 4.13 – Chromosome best fitness (both training and testing) is plotted for 1 through 40 rules. Discretization of the classification space is performed with 11 clusters. Results are not noticeably improved.

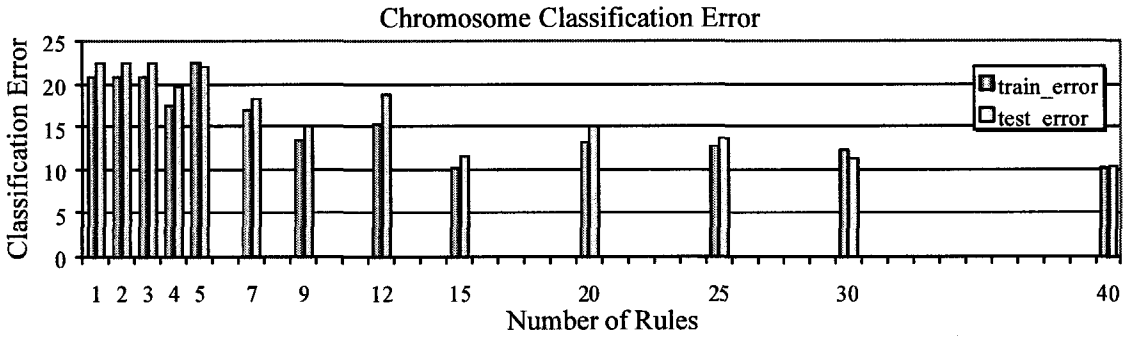


Figure 4.14 – Chromosome classification error (both training and testing) of the best fit chromosomes are plotted for 1 through 40 rules.

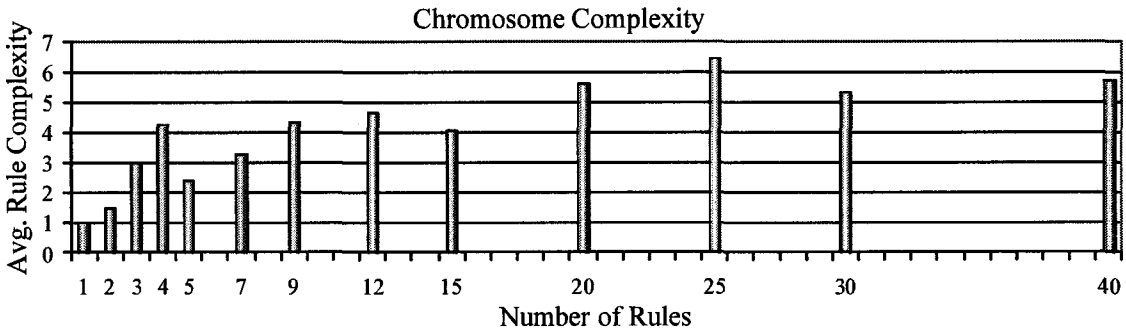


Figure 4.15 – Average rule complexity of the best fit chromosomes are plotted for 1 through 40 rules.

4.2.2.4 Testing

Thus far, the extracted rules have been tested in terms of their ability to model the data captured from the trained ANN performing wall following. The purpose of the training set is to test the *fidelity* of the rules while the testing set tests the *accuracy* of the rules. It is chosen to further test the rule *accuracy* by implementing the rule sets on Khepera and quantitatively measure the resulting wall follow trajectory. When Khepera is implemented with the each of

the evolved rule sets, some elements of the wall follow behavior are observed. However, no rule set is able to guide Khepera successfully around the environment of Figure 2.3 in both clockwise and counterclockwise orientations and most rule sets yield exceptionally poor results in both orientations and are not worth reporting. The goal in Chapter 3 was not to obtain a simple ANN architecture, but to obtain a high performing ANN architecture that could rival a human operator at the wall follow task. The fact that accurate rule sets cannot be extracted is disappointing and clearly indicates limitations to the proposed method. Experimentation could be significantly different if a simpler ANN architecture is used for the wall follow problem. This hypothesis is tested by experimenting with simpler ANN architectures in Sections 4.2.3 and 4.2.4.

4.2.3 Moderate ANN

The *Complex* ANN is modified such that only one past input vector is used instead, which reduces the number of attributes to 16. Additionally, the number of hidden layer neurons is reduced from 18 down to 10. The ANN is trained to perform wall following in a similar manner described in Chapter 3. For sake of comparison, the *Moderate* ANN achieves a testing quality index of 34 with the dataset from Chapter 3 while the *Complex* ANN achieves a testing quality index of 30. The experimentation commences in a similar manner as with the *Complex* ANN problem.

4.2.3.1 Data Collection

Behavior representative data is collected in a similar manner described in 4.2.2.1. Therefore, details are omitted.

4.2.3.2 Discretization

Attribute selection and discretization with the Chi2 algorithm is performed on the acquired data with results summarized in Table 4.5.

Attribute	Intervals					
0	0	6	669	1010		
1	0	7	511	771		
2	0	6				
3	0					
4	0	6	10	232	469	599
5	0	443				
6	0	1	6	37	445	
7	0	1	6	237	573	
8	0					
9	0	5	9	454	799	
10	0	1	6			
11	0	1	4	23	319	946
12	0	1	6	155	433	
13	0	6				
14	0	2	10			
15	0	2	5	11	211	

Table 4.6 – Discretized intervals for the *moderate* wall following data. This interval table is significantly simpler than the *complex* wall interval table seen in Table 4.5.

4.2.3.3 Evolution

Evolution is performed with results summarized in Figure 4.16, Figure 4.17, and Figure 4.18.

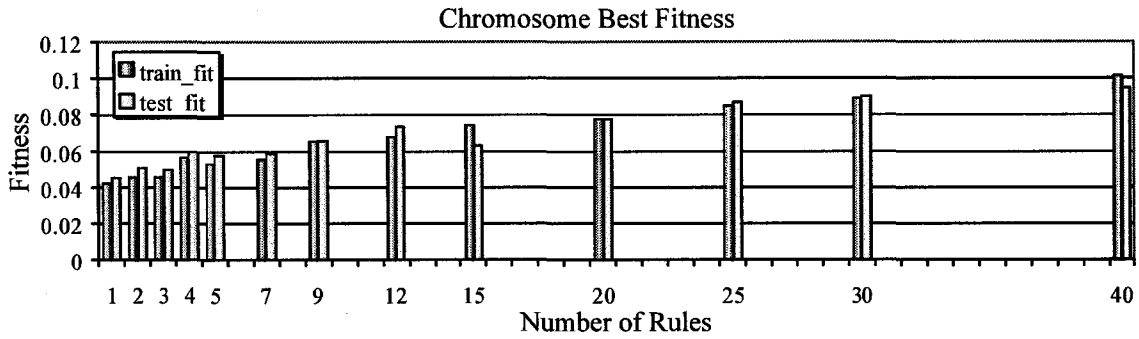


Figure 4.16 – Chromosome best fitness (both training and testing) is plotted for 1 through 40 rules.

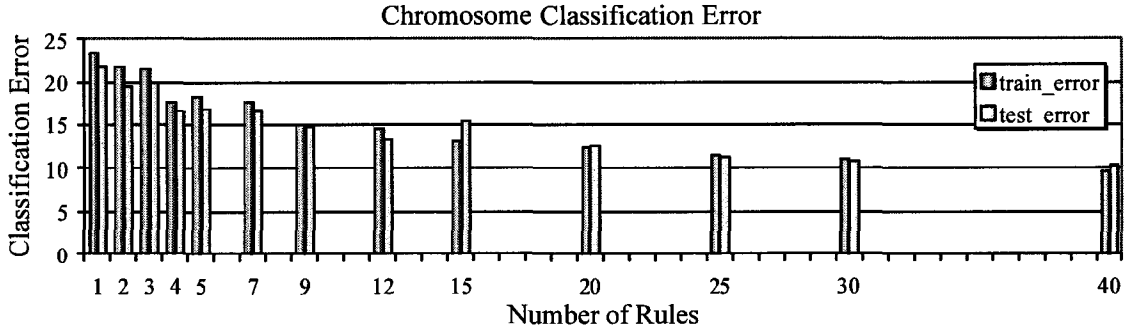


Figure 4.17 - Chromosome classification error (both training and testing) of the best fit chromosomes are plotted for 1 through 40 rules. Error tends to be reduced with additional rules.

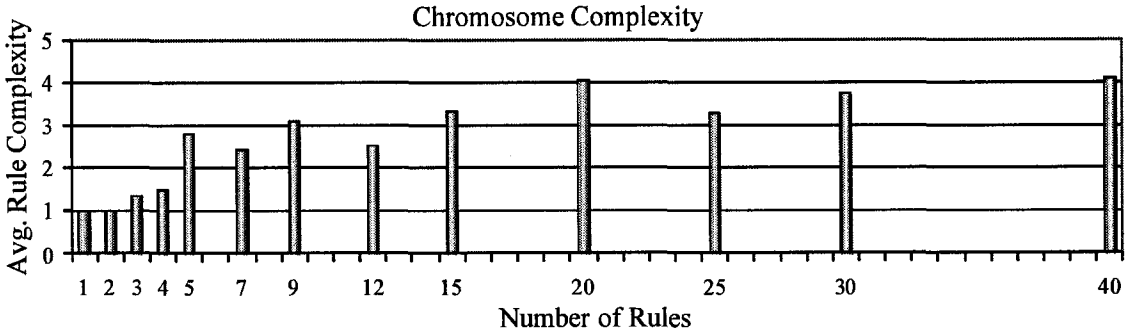


Figure 4.18 - Average rule complexity of the best fit chromosomes are plotted for 1 through 40 rules.

4.2.3.4 Testing

The evolved rule sets are implemented on Khepera and the results are poor as in the *complex* wall problem. Once again there are no set of rules which can successfully navigate Khepera through the wall follow environment in both clockwise and counterclockwise orientations. The results obtained are not worth reporting and are therefore omitted.

4.2.4 Simple ANN

The ANN is further simplified to the *Simple* ANN, which contains only 8 attributes since the memory of past inputs is completely segregated. The most significant implication of this modification is that wall following can only be implemented in one of two orientations: clockwise or counterclockwise. The complexity of the ANN is quite low: the number of hidden neurons is reduced down to 5. The *Simple* ANN is trained for wall following in the clockwise orientation of Figure 2.3.

4.2.4.1 Data Collection

Behavior representative data is collected in a similar manner described in 4.2.2.1 with one important exception. Khepera is implemented with the *Simple* ANN and data is collected with Khepera driving in only the clockwise orientation of Figure 2.3.

4.2.4.2 Discretization

Attribute selection and discretization with the Chi2 algorithm is performed on the acquired data and results are summarized in Table 4.7. It is observed that the average number of intervals per attribute is greater than in the previously seen attribute interval tables. This may be accredited to the fact that the ANN could be more discerning to the attributes since there are fewer of them.

Attribute	Intervals
0	3 727 1023
1	0 153 175 334 378 547 622 623 689 737 779 799 877 931 987 1023
2	0 6
3	0 2 5 12 69 471 638 1023
4	0 4 6 506
5	0 1 6 13
6	0 1 3 6 93
7	0 1 2 4 5 61 79 179 329 455 820

Table 4.7 – Discretized intervals for the *simplified* wall following data. This interval table is significantly simpler than the *complex* wall interval table seen in Table 4.5. However, the number of intervals per attribute is noticeably greater.

4.2.4.3 Evolution

Evolution is performed with 1000 individuals and results are summarized in Figure 4.19, Figure 4.20, and Figure 4.21.

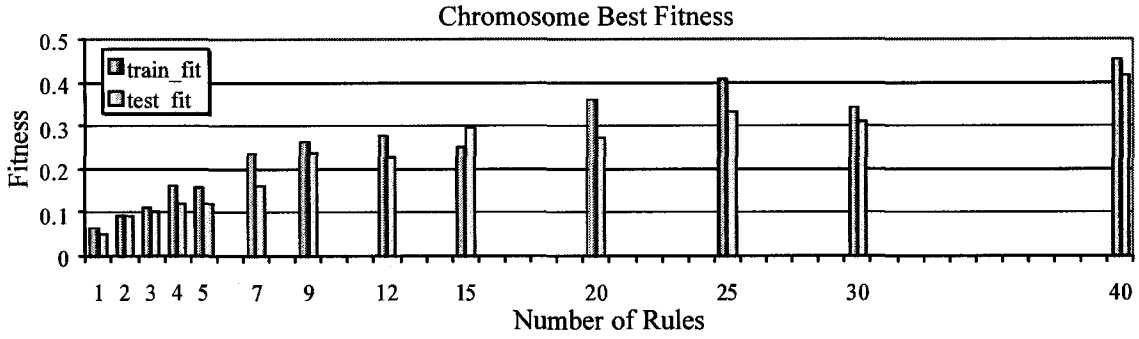


Figure 4.19 – Chromosome best fitness (both training and testing) is plotted for 1 through 40 rules.

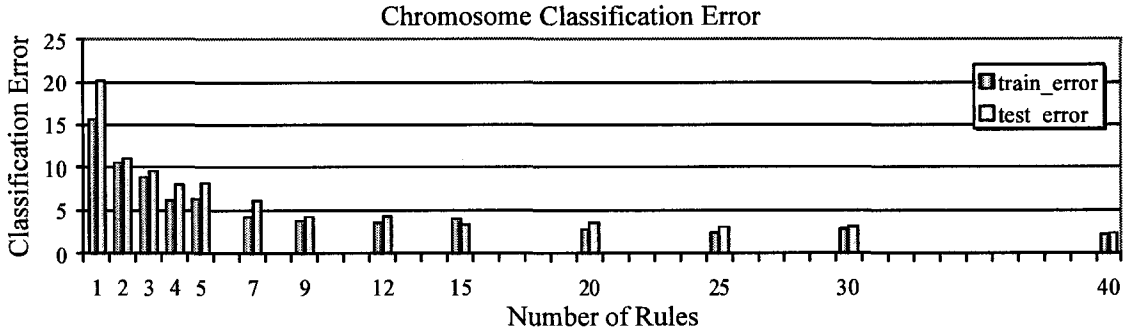


Figure 4.20 - Chromosome classification error (both training and testing) of the best fit chromosomes are plotted for 1 through 40 rules. Error tends to be reduced with additional rules. This tendency is significantly stronger than the evolution results obtained with the *complex* and *moderate* wall.

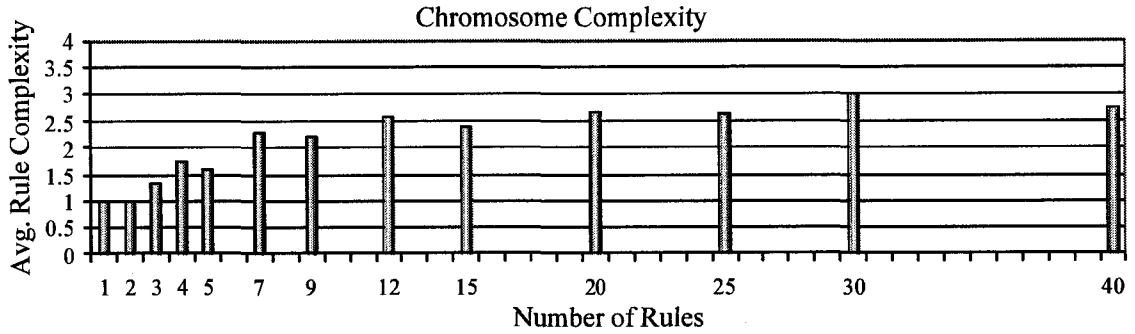


Figure 4.21 - Average rule complexity of the best fit chromosomes are plotted for 1 through 40 rules.

The chromosome fitness values attained are significantly greater in the *simple* wall problem compared to the *complex* and *moderate* wall follow problem. These improvements are significantly built upon when as the chromosome population is increased. The experiment is repeated with 5000 individuals and results are summarized in Figure 4.22, Figure 4.23, and Figure 4.24.

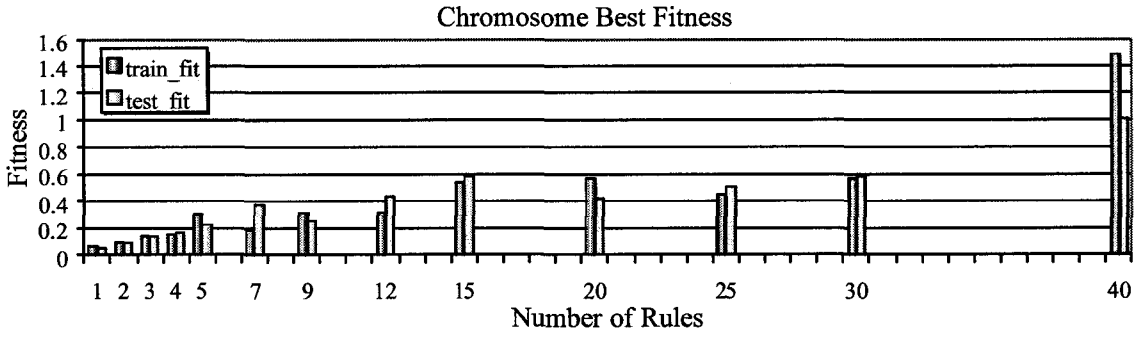


Figure 4.22 – Chromosome best fitness (both training and testing) is plotted for 1 through 40 rules.

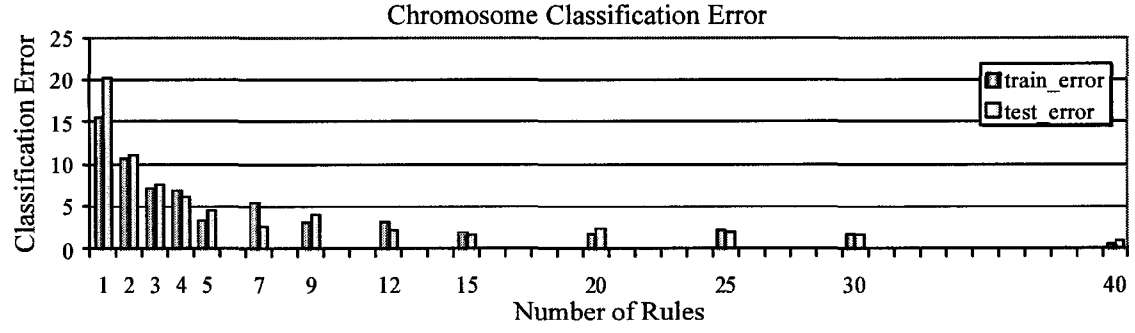


Figure 4.23 - Chromosome classification error (both training and testing) of the best fit chromosomes are plotted for 1 through 40 rules. Classification error tends to be reduced with the addition of rules.

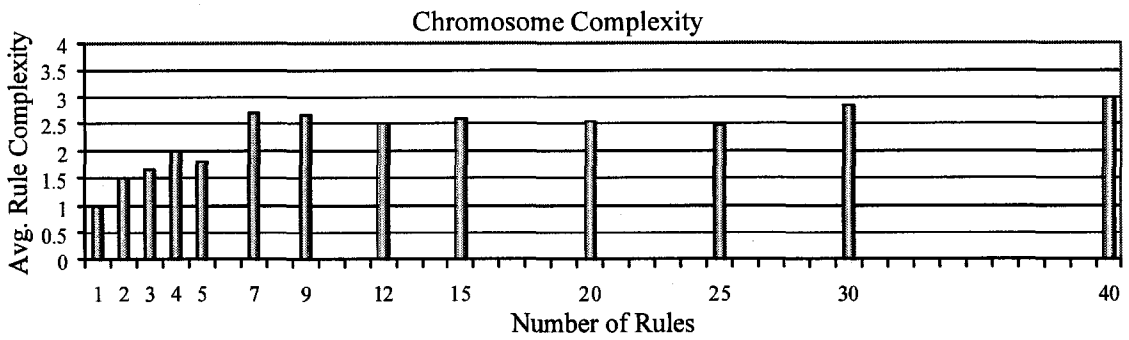


Figure 4.24 - Average rule complexity of the best fit chromosomes are plotted for 1 through 40 rules.

Over-evolution is observed in all evolution trials with 15 rules or more. In Figure 4.26, the testing fitness is seen to decrease after about 400 generations while the training fitness continues to increase. The same phenomenon is not seen in Figure 4.25 or any of the evolutions with fewer than 12 rules. Interestingly, over-evolution was not observed in the previous *complex* and *moderate* ANN problems. It is believed that memorization was not previously seen since the search space was too large to achieve any significant degree of chromosome fitness and fitness values converged towards non-optimal solutions. This is a significant shortcoming, which is believed to be caused by limiting the population size due to computational cost. Previous testing clearly indicates that the rule sets extracted from the *complex* and *moderate* ANNs are poor and are likely non-optimal.

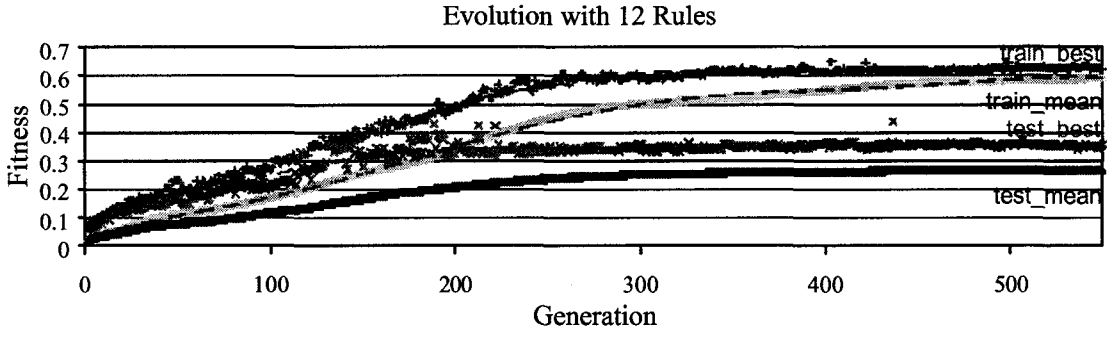


Figure 4.25 – Evolution details for 12 rules and 5000 individuals. Significantly higher fitness values are attained in comparison to the rule extraction fitness with the *complex* ANN and *moderate* ANN. Fitness values converge to levels that are believed to be near optimal.

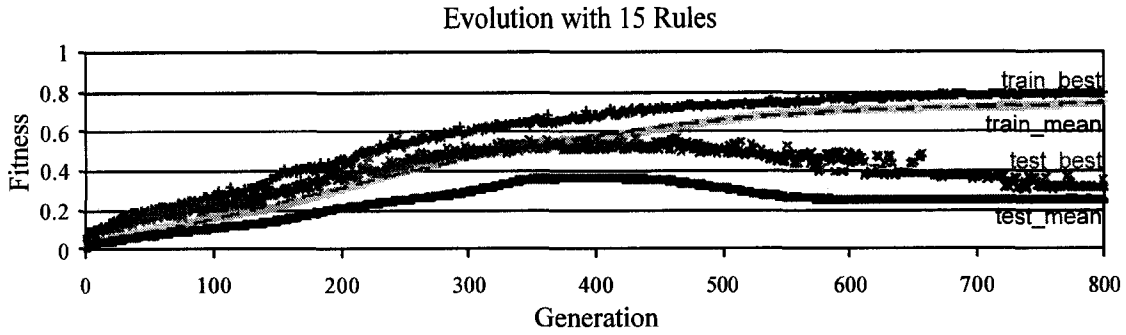


Figure 4.26 – Evolution details for 15 rules and 5000 individuals. Over-evolution is seen as the testing fitness decreases past 400 generations while the training fitness continues to increase. Fitness values do not converge.

4.2.4.4 Testing

The trajectory results obtained from the rule sets extracted from the *simple* ANN are significantly better than those extracted from the *complex* and *moderate* ANNs. Many of the extracted rule sets are able to guide Khepera successfully through the wall follow environment. In particular, all rule sets evolved with 5000 individuals and at least 9 rules are able to adequately perform wall following while 9 rules or less are inadequate. Adequate performance is defined as completing a full lap in the wall follow environment of Figure 2.3 with a maximum trajectory error of less than 10 cm. An example trajectory result is plotted below in Figure 4.27 for the case of 12 rules.

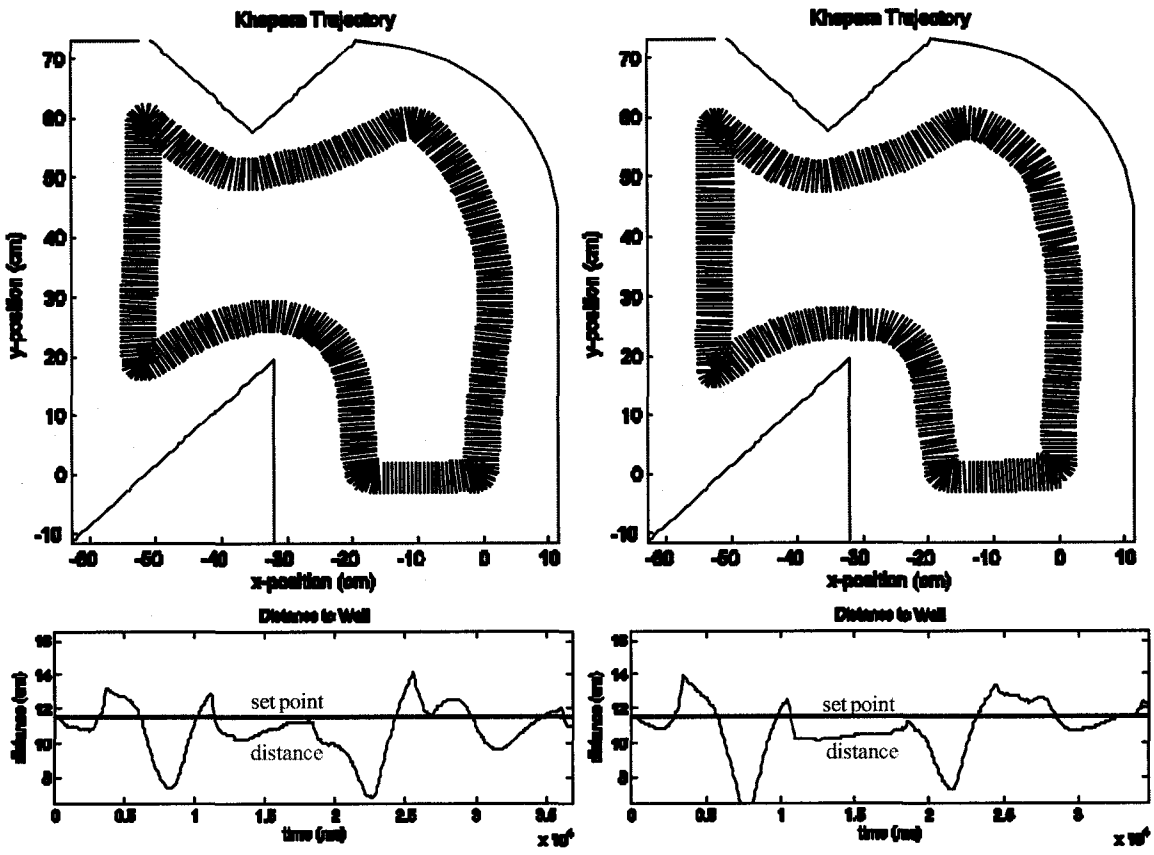


Figure 4.27 – Khepera is implemented with the 12 rules and produces a clockwise wall follow trajectory (top left) with a mean error of 1.27cm. When the *simple* ANN is implemented on Khepera a mean error of 1.25cm is achieved. The two trajectories are strikingly similar.

The 12 discrete rules yields a trajectory mean error of 1.27cm. In comparison to the trained ANN, which has a trajectory mean error of 1.25cm, the extracted rules are very similar in performance. The two trajectories performed in Figure 4.27 are highly similar, which indicates that the rule-set *accuracy* is very good. A summary of the performance obtained from each of the discrete rule sets are summarized in Table 4.8.

controller	trajectory error			
	minimum	maximum	mean	std_dev
simple ANN	0	5.69	1.27	1.19
1 rule	fail	fail	fail	fail
2 rules	fail	fail	fail	fail
3 rules	fail	fail	fail	fail
4 rules	fail	fail	fail	fail
5 rules	fail	fail	fail	fail
7 rules	fail	fail	fail	fail
9 rules	0	9.09	3.01	2.52
12 rules	0	4.63	1.25	1.1
15 rules	0	5.9	1.7	1.35
20 rules	0	5.23	1.55	1.27
25 rules	0	4.94	1.51	1.22
30 rules	0	5.83	1.85	1.45
40 rules	0	4.38	1.14	0.99

Table 4.8 – Performance of the discrete rule sets obtained from the *simple* ANN are summarized.

It is clear from Table 4.8 that a minimum number of rules must be used in order to achieve acceptable rule *accuracy*. Unfortunately, the rule set comprehension decreases with the addition of more rules. There exists a trade-off between rule set *comprehension* and rule *accuracy*.

4.2.5 Re-Extraction

In Section 4.2.4, discrete rule sets have been extracted from the *simple* ANN and rule accuracy was generally deemed acceptable. Section 4.2.5 examines the effectiveness of the proposed rule extraction method by re-applying the method to extract rules to a set of discrete rules obtained in Section 4.2.4. The motivation behind this endeavor is to compare the extracted rule set with the re-extracted rule set in terms of similarities in rule comprehension and accuracy. It is chosen arbitrarily to use the rule set with 12 rules for this study since it possesses a good compromise between rule accuracy while not employing an excessive number of rules.

4.2.5.1 Data Collection

Behavior representative data is collected in a similar manner described in 4.2.2.1 with two important exceptions: Khepera is implemented with the 12 rules instead of an ANN, and data is collected for only the clockwise orientation of Figure 2.3.

4.2.5.2 Discretization

Attribute selection and discretization with the Chi2 algorithm is performed on the acquired data with results summarized in Table 4.9.

Attribute	Intervals
0	0 1022
1	0 360 384 475 476 532 545 581 584 624 690 720 736 792 868 873 931 936 960 1023
2	0 8
3	0 11 119 438 625 648 1023
4	0 7
5	0
6	0 4
7	0 31

Table 4.9 - Discretized intervals for the *re-extraction* wall following data.

4.2.5.3 Evolution

Evolution is performed with 5000 individuals. The individual chromosome with the highest testing fitness possesses the following properties:

```
training fitness: 0.563658
testing fitness:  0.934454
training error:   1.77072
testing error:    1.0689
rule complexity:  2.16667
```

It is rather disappointing to see that the re-extracted rule set contains training and testing error of such magnitude. Ideally the error would be zero and the re-extracted rules would exactly resemble the originally extracted rules. Unfortunately this is not the case. The original and re-extracted rule sets are listed in Figure 4.29 and Figure 4.29, respectively. Upon initial inspection the rule sets appear significantly different.

If x0 in [3, 727) and x1 in [622, 1024) and x2 in [6, 1024) and x6 in [6, 1024) then y=(10,-10)
If x1 in [779, 1024) and x2 in [6, 1024) and x5 in [0, 13) and x7 in [2, 1024) then y=(10,-9)
If x3 in [638, 1024) then y=(7,-8)
If x3 in [638, 1024) then y=(9,-5)
If x1 in [547, 931) and x3 in [0, 638) then y=(10,7)
If x0 in [727, 1024) and x1 in [689, 779) and x3 in [0, 638) then y=(10,9)
If x0 in [1023, 1024) and x1 in [737, 987) and x3 in [0, 1023) and x7 in [0, 61) then y=(10,9)
If x1 in [378, 931) and x3 in [0, 1023) then y=(9,8)
If x0 in [1023, 1024) and x4 in [0, 6) then y=(9,10)
If x3 in [0, 471) and x4 in [0, 6) and x7 in [0, 179) then y=(8,10)
If x3 in [12, 1023) and x7 in [0, 61) then y=(8,11)
If x1 in [0, 623) and x3 in [0, 12) then y=(6,13)

Figure 4.28 – Original 12 rules extracted from the *simple* ANN.

If x1 in [868, 1024) and x2 in [8, 1024) and x6 in [4, 1024) then y=(10,-9)
If x3 in [625, 1024) then y=(9,-8)
If x2 in [8, 1024) and x3 in [625, 1024) and x7 in [31, 1024) then y=(9,-8)
If x2 in [8, 1024) and x3 in [119, 648) and x7 in [31, 1024) then y=(8,-7)
If x3 in [648, 1024) then y=(8,-7)
If x1 in [476, 960) and x3 in [119, 648) then y=(9,8)
If x0 in [1022, 1024) and x4 in [0, 7) then y=(9,8)
If x0 in [1022, 1024) and x3 in [0, 1023) then y=(9,8)
If x1 in [624, 868) and x3 in [0, 438) then y=(9,8)
If x1 in [736, 931) and x4 in [0, 7) then y=(9,9)
If x3 in [0, 1023) and x7 in [0, 31) then y=(7,10)
If x1 in [0, 624) and x2 in [0, 8) and x3 in [0, 11) then y=(7,11)

Figure 4.29 – 12 rules are re-extracted from the original 12 rules. These rules are marked with a '#'.

Upon closer inspection some similarities can be seen. These similarities are outlined in Table 4.10 below.

Rules	Comments
If x3 in [638, 1024) then y=(7,-8) If x3 in [638, 1024) then y=(9,-5) # If x3 in [648, 1024) then y=(8,-7) # If x3 in [625, 1024) then y=(9,-8)	Rotate right
If x1 in [0, 623) and x3 in [0, 12) then y=(6,13) # If x1 in [0, 624) and x2 in [0, 8) and x3 in [0, 11) then y=(7,11)	Bend left
If x3 in [12, 1023) and x7 in [0, 61) then y=(8,11) #If x3 in [0, 1023) and x7 in [0, 31) then y=(7,10)	Bend left
If x1 in [378, 931) and x3 in [0, 1023) then y=(9,8) # If x1 in [624, 868) and x3 in [0, 438) then y=(9,8)	Drive straight
If x0 in [1023, 1024) and x4 in [0, 6) then y=(9,10) # If x0 in [1022, 1024) and x4 in [0, 7) then y=(9,8)	Drive straight
If x1 in [547, 931) and x3 in [0, 638) then y=(10,7) # If x1 in [476, 960) and x3 in [119, 648) then y=(9,8)	Bend right
If x0 in [3, 727) and x1 in [622, 1024) and x2 in [6, 1024) and x6 in [6, 1024) then y=(10,-10) If x1 in [779, 1024) and x2 in [6, 1024) and x5 in [0, 13) and x7 in [2, 1024) then y=(10,-9) # If x1 in [868, 1024) and x2 in [8, 1024) and x6 in [4, 1024) then y=(10,-9) # If x2 in [8, 1024) and x3 in [625, 1024) and x7 in [31, 1024) then y=(9,-8) # If x2 in [8, 1024) and x3 in [119, 648) and x7 in [31, 1024) then y=(8,-7)	Misc. Rotate right
If x0 in [727, 1024) and x1 in [689, 779) and x3 in [0, 638) then y=(10,9) If x0 in [1023, 1024) and x1 in [737, 987) and x3 in [0, 1023) and x7 in [0, 61) then y=(10,9) If x3 in [0, 471) and x4 in [0, 6) and x7 in [0, 179) then y=(8,10) # If x0 in [1022, 1024) and x3 in [0, 1023) then y=(9,8) # If x1 in [736, 931) and x4 in [0, 7) then y=(9,9)	Misc. Drive Straight

Table 4.10 – The originally extracted rules and re-extracted rules are entered into a table in order to outline similarities between the two rule sets. For example, the first row contains rules that check to see if x3 is fairly high in order to sharply rotate Khepera to the right. The re-extracted rules are marked with a '#'.

The similarities between the two rule sets can be difficult to perceive as is the case in the last two rows of Table 4.10. One possible explanation could be that the rule set comprehension is poor. Alternatively, the rule set extraction method could be poor in extracting accurate rule sets. Accuracy limitations have already been observed with the *complex* and *moderate* ANN rule extraction problems. The accuracy degrading introduced by the rule extraction method is probably best investigated by quantifying and comparing trajectory error for each of the three wall follow controllers: *simple* ANN, 12 extracted rules, and 12 re-extracted rules.

4.2.5.4 Testing

Khepera is implemented with the 12 re-extracted rules and a trajectory is plotted in Figure 4.30 below. Significant performance degrading is seen as the mean trajectory error increases from 1.25cm to 1.98cm. This is mostly attributed to the poor performance seen on the right-most wall.

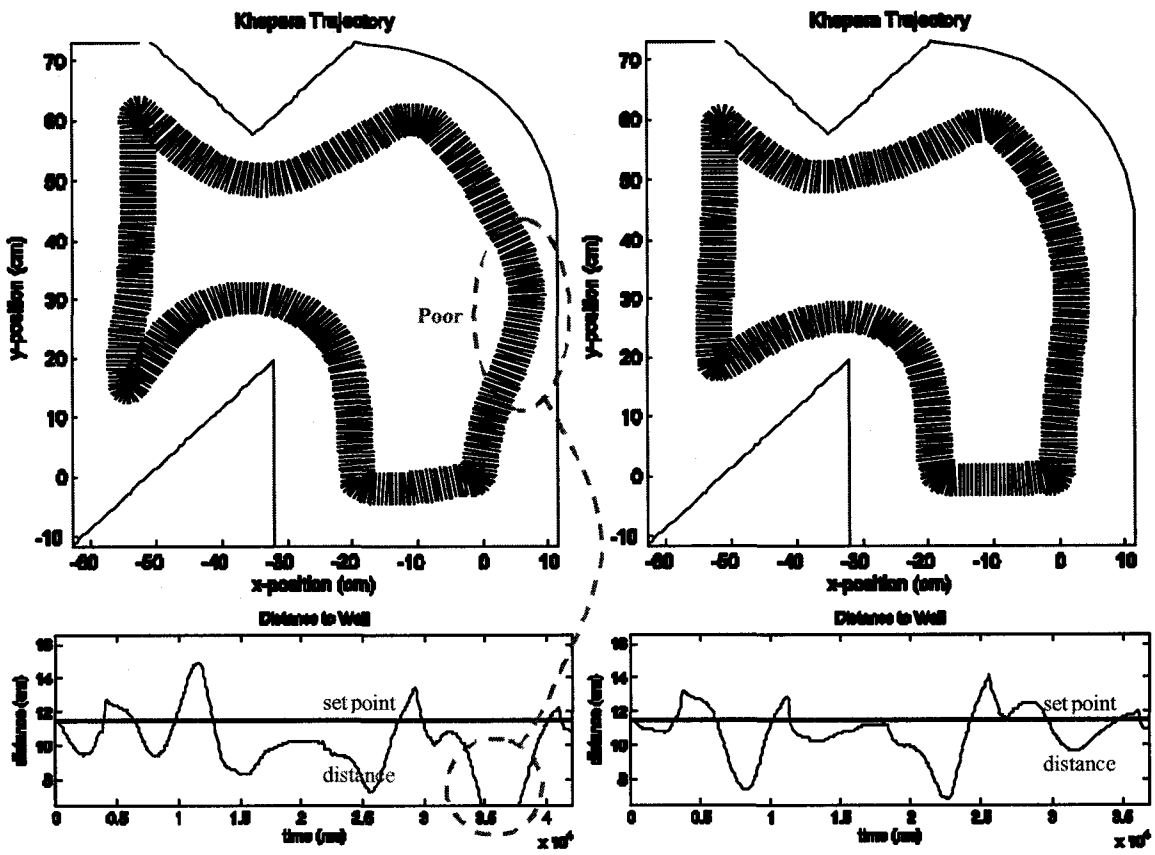


Figure 4.30 – Khepera is implemented with the 12 re-extracted rules and produces a clockwise wall follow trajectory (top left) with a mean error of 1.98cm. When the original 12 rules are implemented on Khepera a mean error of 1.25cm is achieved. Significant performance degrading is seen on the right-most wall.

In summary, the mean trajectory errors reported for the simple ANN, 12 extracted discrete rules, and 12 re-extracted discrete rules are 1.27cm, 1.25cm, and 1.98cm, respectively. It is likely that the proposed rule extraction method loses some accuracy during each translation.

4.3 Discussion

The proposed methodology for discrete rule set extraction is computationally expensive. This is mostly attributed to the compounding computational costs of undertaking the Pittsburgh chromosome approach with very challenging high-dimensionality problem domains. Consider the case of extracting 12 rules. Equation 4.4 indicates that the number of chromosome permutations is 7.40×10^{159}, 7.17×10^{217}, and 3.11×10^{320} for the *complex*, *moderate*, and *simple* ANNs, respectively. Evidently, the dimensionality of the *complex* and *moderate* ANN rule extraction problem are too great for the proposed method. The

discretization stage could not adequately reduce the search space for these two ANN problems.

There are many degrees of freedom (e.g. recombination, fitness evaluation, and chromosome encoding, etc.) of the RCGA which call for more experimentation. However, the computational cost of such an endeavor marks this infeasible. Unfortunately, computational cost restricts experimentation. The following explorations may have been considered if computational costs had not been so restricting:

1. **Greater populations**. Convergence towards non-optimal solutions is a great shortcoming of the RCGA when confronted with an exceptionally large search space. This is seen in the case of rule extraction from the *complex* and *moderate* ANNs. Increasing the population size helps to avoid this problem.
2. **Chromosome Encoding**. Only the Pittsburgh encoding method is investigated. Exploration and comparison of the Michigan and Pittsburgh approaches could be beneficial Also, different rule forms could be explored. It has been shown in Section 4.2.1 that the chosen rule form restricts the classification of irises. In such instances, a different rule form is necessary in order to achieve higher performing rule sets while preserving rule comprehension.
3. **Varying membership functions**. Alternative membership functions to the rectangular one seen in Figure 4.2 could be explored.
4. **More data**. Acquiring and utilizing additional data vectors to capture the ANN behavior may be beneficial.

The feature selection and discretization performed with the Chi2 algorithm could be modified such that the error rate associated with interval merging is computed in a manner that gives consideration to class similarity. The rationale is that the error associated with merging attribute intervals should be deemed less when the associated classes are similar as opposed to being markedly different. A measure of class similarity can be assigned based on one of the many similarity measurements outlined in [63]. This modification would make the Chi2 algorithm mores suitable for problems with continuous classification. This modification, however, is difficult to implement because it is not clear how x^2 values should be computed.

4.4 Conclusion

A methodology for extracting discrete rule sets from high dimensionality data captured from a trained ANN with both continuous attributes and classification is explored. The rule extraction methodology uses a real-coded genetic algorithm (RCGA) to evolve a discrete rule set. Some experimentation is performed with the Iris plant classification problem and the extracted rule sets are comprehensive and able to correctly classify 146 of 150 data points. Experimental results indicate that the rule sets extracted from the *simple* ANN successfully perform wall following when implemented on the mobile robot, Khepera. However, in comparison to the trained *simple* ANN, the extracted rules suffer with poorer performance. Furthermore, rules extracted from the *simple* ANN are not as comprehensive as the iris plant extracted rules. This is attributed to the fact that the wall follow problem is considerably more complicated than the iris plant classification problem in terms of attribute and classification space dimensionality. Rules cannot be practically extracted from the *complex* and *moderate* ANNs with adequate rule set accuracy due to the enormous computational cost. The proposed methodology is likely applicable to numerous problem domains independent of ANN architecture and attribute/class domain (i.e. binary, discrete, continuous, etc). However, the computational cost of the proposed method makes it inappropriate for highly complex problems. A decomposition rule extraction approach may be more appropriate in such cases.

Chapter 5

Conclusion

Braitenberg argued that behaviors could be synthetically produced in mobile robotics. Since his book publishing in 1984, numerous works can be found in the literature which validates many of his assertions. In this thesis, it is demonstrated that autonomous robotic reactive behavior can be synthesized with artificial neural networks (ANNs). Implicit knowledge is captured using a learning algorithm for an ANN-based controller for the problem of autonomous robotic wall following. Numerous quantitative measurements indicate that the trained ANN is able to successfully generalize across a variety of environments with varying wall geometry.

Performance evaluation reports that the trained ANN can achieve better wall follow trajectory control at higher speeds than a human operator. The research of behavior performance scalability is significant since real-world applications could potentially benefit from very fast reactive controllers obtained from extrapolation of controllers trained under slower operating conditions. The quantification of behavior performance scalability is a notable contribution to research and is a recommended research direction. Particular attention should be given to the controller's sampling frequency on the input. Fast and accurate reactive control requires frequent input sampling.

Braitenberg also illustrates how more complex autonomous robotic designs tend to be less transparent. This phenomenon creates a new research area of knowledge representation. It is important to represent knowledge in a comprehensive form so that the knowledge can be read and/or verified by a human expert. It is demonstrated that the implicit knowledge embedded into the trained ANN can be extracted and represented in human-comprehensive terms. A methodology for extracting discrete rule sets from a trained ANN with both continuous attributes and classification is explored. The methodology is based on attribute discretization, feature selection, and evolutionary rule-set search using a real-coded genetic algorithm. Experimentation is conducted on three different ANN architectures with varying size and complexity. Experimental results indicate that the methodology is only successful with the simplest ANN architecture. This is believed to be a result of the method's exponentially increasing heuristic search space associated with the addition of attributes in the more complex ANN architectures. Furthermore, these additional attributes tend to decrease the comprehension of the extracted rules.

The rule extraction methodology is likely applicable to numerous problem domains independent of ANN architecture and attribute/class domain (i.e. binary, discrete, continuous, etc). In this sense, the rule extraction methodology is a significant contribution to research. However, the computational cost of heuristic search makes the method inappropriate for highly complex problems. A decomposition rule extraction approach may be more appropriate in such cases. Therefore, decompositional rule extraction of ANNs with continuous attributes and continuous classification is a recommended research direction. There is a need for a universal ANN rule extraction method that would ideally be applicable to any ANN architecture and have a low order of computational cost.

Bibliography

[1] V. Braitenberg. (1984) *Vehicles: Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA.

[2] E. Aguirre, A. González, Fuzzy behaviors for mobile robot navigation: design coordination and fusion, International Journal of Approximate Reasoning 25 (2000) 255-289.

[3] T. Poggio, F.Girosi, Networks for approximation and learning, Proc. IEEE 78 (1996) 1481-1497.

[4] G. Castellano, G. Attolico, & A. Distante. Automatic generation of fuzzy rules for reactive robot controllers. Robotics and Autonomous Systems 22 (1997) 133-149.

[5] V. Matellán, C. Fernández & J. Molina, Genetic learning of fuzzy reactive controllers, Robotics and Autonomous Systems 25 (1998) 33-41.

[6] S. Lee and S. Cho (2001), Emergent Behaviors of a Fuzzy Sensory Controller Evolved by Genetic Algorithm, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 31 (2001) 919-929.

[7] D. Floreano & F. Mondada, Evolutionary neurocontrollers for autonomous mobile robots. Neural Networks 11 (1998) 1461-1478.

[8] J. Santos, R.J. Duro, J.A. Becerra, J.L Crespo, F. Bellas, Considerations in the application of evolution to the generation of robotic controllers, Information Sciences 113 (2001) 127-148.

[9] S. Nolfi, Evolving non-trivial behaviors on real robots: A garbage collecting robot, Robotics and Autonomous Systems 22 (1997) 187-198.

[10] F. Herrera, M. Lozano, J.L. Verdegay, Tackling real-coded genetic algorithms: operators and tools for behavior analysis, Artificial Intelligence Review 12 (1998) 265-319.

[11] K. Takita, Y. Kakazu, Automatic Agent Design Based on Gate Growth – Application to Wall Following Problem –, Proceedings of the 37th SICE Annual Conference (1998) 863-868.

[12] I. Harvey, P. Husbands, D. Cliff, A. Thompson, N. Jakobi, Evolutionary robotics: the Sussex approach. Robotics and Autonomous Systems 20 (1997) 205-224

[13] R. Braunstingl, J. Mujika, J. Uribe, A Wall Following Robot With A Fuzzy Logic Controller Optimised By A Genetic Algorithm, Proceedings of the 1995 International Conference of Fuzzy Systems 5 (1995) 77-82.

[14] S. Yamada, M. Murota, Unsupervised Learning to Recognizing Environments from Behavior Sequences in a Mobile Robot, Proceedings of the 1998 IEEE International Conference on Robotics & Autonmation Leuven (1998) 1871-1876.

[15] N. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[16] C. Touzet, Neural reinforcement learning for behavior synthesis, Robotics and Autonomous Systems 22 (1997) 251-281.

[17] B. Jerbic, K. Grolinger, B. Vranješ, Autonomous agent based on reinforcement learning and adaptive shadowed network, Artificial Intelligence in Engineering 13 (1999) 141-157.

[18] E.Zalama, J. Gómez, M. Paul, J. Perán, Adaptive Behavior Navigation of a Mobile Robot, IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, vol.32 no. 1 (2002) 160-169.

[19] E. Tunstel, Coordination of Distributed Fuzzy Behaviors in Mobile Robot Control, IEEE Intl. Conf. on Systems, Man, & Cybernetics, vol. 31 no. 6 (2001) 919-929.

[20] E. Tunstel, Mobile robot autonomy via hierarchical fuzzy behavior control, Proc. 6th Intl. Symp. on Robotics & Manuf. (1996) 837-842.

[21] E. Tunstel, T. Lippincott, M. Jamshidi, Behavior hierarchy for autonomous mobile robots: fuzzy-behavior modulation and evolution, Intl. Journal of Intelligent Automation & Soft Computing Vol 3 No 1 (1997) 37-50.

[22] K. Ng, M. Trivedi, A Neuro-Fuzzy Controller for Mobile Robotic Navigation and Multirobot Conveying, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, vol.28 no. 6 (1998) 829-840.

[23] U. Nehmzow, Quantitative analysis or robot—environment interaction—towards "scientific mobile robotics", Robotics and Autonomous Systems 44 (2003) 55-68.

[24] R. Setiono, J. Thong, C.S. Yap, Symbolic rule extraction from neural networks An application to identifying organizations adopting IT, Information & Management 34 (1998) 91-101.

[25] F. Maire, Rule-extraction by backpropagation of polyhedra, Neural Networks 12 (1999) 717-725.

[26] J. Alexander, M. Mozer, Template-based procedures for neural network interpretation, Neural Networks 12 (1999) 479-498.

[27] R. Andrews, J. Diederich, A. Tickle, A Survey and Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks, Knowledge Based Systems 8 (1995) 373-389.

[28] J Neumann, Classification and Evaluation of Algorithms for Rule Extraction From Artificial Neural Networks, PhD Summer Project, Division of Informatics, University of Edinburgh (1998).

[29] G. Towell, Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction. PhD thesis, Computer Science Department, University of Wisconsin, Madison (1991).

[30] I. Taha, J. Ghosh, Symbolic interpretation of artificial neural networks, IEEE transactions on Knowledge and Data Engineering 11 (1998) 448-462.

[31] E. Pop, J. Diederich, RULENEG: Rule-extraction from neural networks by step-wise negation, Technical report, Neurocomputing Research Center, Queensland University of Technology, 1994.

[32] R. Setiono, H. Liu, Understanding neural networks via rule extraction, Proceedings of the 14th International Joint Conference on Artificial Intelligence, 480-485, Montreal, Canada.

[33] R. Krishnan, G. Sivakumar, P. Bhattacharya, A search technique for rule extraction from trained neural networks, Pattern Recognition Letters 20 (1998) 273-280.

[34] M. Ishikawa, Rule extraction by successive regularization, Neural Networks 13 (2000) 1171-1183.

[35] S. Huang, H. Xing, Extract intelligible and concise fuzzy rules from neural networks, Fuzzy Sets and Systems 132 (2002) 233-243.

[36] R. Kerber, Chimerge: Discretization for numeric attributes. Proc. National Conference on Artificial Intelligence (1992), AAAI Press, pp. 123-128.

[37] H. Liu, R. Setiono, Chi2: Feature selection and discretization of numeric attributes, Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence (1995).

[38] F. Tay, L. Shen, A Modified Chi2 Algorithm for Discretization, IEEE Transactions on Knowledge and Data Engineering 14 (2002) 666-670.

[39] M. Last, A. Kandal, O. Maimon, Information-theoretic algorithm for feature selection, Pattern Recognition Letters 22 (2001) 799-811.

[40] R. Setiono, H. Liu, NeuroLinear: From neural networks to oblique decision rules, Neurocomputing 17 (1997) 1-24

[41] Y. Hayashi, R. Setiono, K. Yoshida, A comparison between two network rule extraction techniques for the diagnosis of hepatobiliary disorders, Artificial Intelligence in Medicine 20 (2000) 205-216.

[42] R. Setiono, H. Liu, Symbolic Representation of Neural Networks, IEEE Computer 29 (1996) 71-77.

[43] R. Andrews, S. Geva, RULEX & CEBP networks as the basis for a rule refinement system. In J. Hallam, editor, Hybrid Problems Hubrid Solutions, pp. 1-12. IOS Press.

[44] H. Liu and S. Tan, X2R: A Fast Rule Generator, Proceedings of IEEE International Conference on Systems, Man and Cybernetics 2 (1995) 1631-1635

[45] C. Pena-Reyes, M. Sipper, A fuzzy-genetic approach to breast cancer diagnosis, Artificial Intelligence in Medicine 17 (1999) 131-155.

[46] H. Ishibuchi, T. Nakashima, T. Murata, Three-objective genetics-based machine learning for linguistic rule extraction, Information Sciences 136 (2001) 109-133.

[47] R. Krishnan, G. Sivakumar, P. Bhattacharya, Extracting decision trees from trained neural networks, Pattern Recognition 32 (1999) 1999-2009.

[48] M. Kim, I. Han, The discovery of experts' decision rules from qualitative bankruptcy data using genetic algorithms, Expert Systems with Applications 25 (2003) 637-646.

[49] I. Jagielska, C. Matthews, T. Whitfort, An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition for classification problems, Neurocomputing 24 (1999) 37-54.

[50] M. Lee, T. Williams, Robotic Navigation using Fuzzy Spatial Inference, IEEE Proceedings of the International Conference on Systems, Man and Cybernetics (1998) 2154-2159.

[51] K. Ward, A. Zelinsky, An Exploratory Robot Controller which Adapts to Unknown Environments and Damaged Sensors, International Conference on Field and Service Robots (1997) 477-484

[52] M. Mucientes, R. Iglesias, C.V. Regueiro, A. Bugarin, S. Barro, A fuzzy temporal rule-based velocity controller for mobile robotics, Fuzzy Sets and Systems 134 (2003) 83-99.

[53] S. Huang, H. Xing, Extract intelligible and concise fuzzy rules from neural networks, Fuzzy Sets and Systems 132 (2002) 233-243.

[54] K. Mehrota, C. Mohan, S. Ranka, Elements of artificial neural networks, The MIT Press, Cambridge MA, 1997.

[55] J. Alexander, M. Mozer, Template-based procedures for neural network interpretation, Neural Networks 12 (1999) 479-498.

[56] Y. Hayashi, R. Setiono, K. Yoshida, A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders, Artificial Intelligence in Medicine 20 (2000) 205-216.

[57] G. Alefeld, G. Mayer, Interval analysis: theory and applications, Journal of Computational and Applied Mathematics 121 (2000) 421-464.

[58] W. Lodwick, K. Jamison, Special issue: interfaces between fuzzy set theory and interval analysis, Fuzzy Sets and Systems 135 (2003) 1-3.

[59] F. Herrera, M. Lozano, J.L. Verdegay, Tackling real-coded genetic algorithms: operators and tools for behavior analysis, Artificial Intelligence Review 12 (1998) 265-319.

[60] Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[61] K. Saito, R. Nakano, Medical diagnostic expert system based on PDP model, Proceedings of the IEEE International Conference on Neural Networks 1 (1998) 255-262.

[62] R. Andrews, J. Diederich, & A. Tickle, Survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge Based Systems 8 (1995) 373-389.

[63] F. Berzal, J.C. Cubero, N. Marin, D. Sánchez, Building multiway decision trees with numerical attributes, Information Sciences XXX (2003) XXX-XXX.

[64] M. Odetayo, Optimal Population Size for Genetic Algorithms: An Investigation, IEE Colloquium on Genetic Algorithms for Control Systems Engineering (1993) 2/1-2/4

[65] J. Alander, On Optimal Population size of genetic algorithms, Proceedings of CompEuro '92 (1992) 65-70.

[66] R. Haupt, Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm that Optimizes Array Factors, IEEE Antennas and Propagation Society International Symposium 2 (2000) 1034-1037.