THE IMPROVEMENT OF ROBOTIC PATHWAY NAVIGATION AND OBSTACLE AVOIDANCE BY SIMULATION AND STRATEGIC GENETIC ALGORITHMIC MANIPULATION

A thesis submitted to the

Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of Industrial Engineering
of the College of Engineering

1999

by

Christopher J. Deters

B.S., University of Cincinnati, 1994

Committee Chair: Dr. Ernest L. Hall

Abstract

This thesis investigates the use of strategic genetic algorithmic manipulation, using basic theory without the use of binary coding or bit string representation, to improve the performance of an autonomous mobile robot whose actions and behaviors are governed by a stimulus-response rule base. The general theory is reviewed and then applied to a simple autonomous robotic model, pathway, and obstacles. The robotic model's goal is to successfully maneuver down a specified pathway, maintaining the pathway constraints while avoiding obstacles positioned in the pathway. A simulation of the simple autonomous robotic model is employed using the pathway and obstacle model's positions on a Cartesian coordinate system to provide input signals or stimuli by which the robot then reacts or responds. The initial stimulus triggers a sequence of bidding and selection for specific response rules, testing the performance of these rules, evaluating the testing, assigning a payoff for the stimulus-response rules utilized, followed by simple genetic manipulation of these stimulus-response rules then repeating the sequence. This process is then iterated repeatedly leading to a more affective (improved) set of stimulus-response rules governing its actions and behaviors.

This process was developed as a strategic plan that employs the idea of treating the set of stimulus-response rules as a unified set that relies on each other rather than individuals. The results of applying this theory enabled the robotic simulation model to improve its performance from traversing approximately 10% of the prescribed course to completing a successful run of 100% of the course.

The significance of this research is to provide a simple, yet affective method that may ultimately lead to the development of an autonomous mobile robot model that is more robust than the original model. Although this is a specific case, this research can then be translated and utilized into providing a simple/uncomplicated means to improve pathway navigation and obstacle avoidance situations in such areas as: Industrial material handling (AGVs), tool pathway improvement, hazardous waste handling, military applications such as unmanned weapons delivery vehicles, and in the automotive industry as automated guidance around road hazards.

Copyright Notice

Acknowledgments

I wish to express my sincere gratitude to my advisor Professor Ernest L. Hall for his guidance, input and continuous support throughout my graduate studies and particularly with the preparation of this thesis.

I would also like to express my thanks to the entire Industrial Engineering department and staff at the University of Cincinnati for assisting me in returning to academia with my graduate studies. In particular, I would like to thank my thesis committee members, Drs. S. Anand and A. Houshmand for their time and support throughout my time at the University of Cincinnati.

I would also like to acknowledge the many works of Alan C. Schultz and John J. Grefenstette along with their associates whose work not only was the initial influence for this thesis topic but also provided vast information and great insight into this subject. Also, The Navy Center for Applied Research in Artificial Intelligence for providing numerous technical papers in a very timely manner.

I would also like to acknowledge my friends and employers, Jake Berry, Charlie Hernandez, and Dale Skjerseth for their consideration in work scheduling while I pursued this endeavor.

Finally, I wish to thank my family and friends for their support throughout the years and wish to dedicate this work to my parents Donald and Madge Deters who are no longer with me to see this work completed.

Table of Contents

List of Figures

List of Symbols

R_u = range to upper boundary

R_o = range to obstacle

β = bearing angle

θ = heading angle

# = either a distance or angle value where appropriate

r = radius of a circle

x = any position on the a-axis

h = x-axis position for the center of a circle

k = y-axis position for the center of a circle

$y_{upper\ bound}$ = the y-axis distance from the center of the robot to the upper boundary

y_{robot} = y-axis position of the robot

x_{robot} = x-axis position of the robot

$x_{obstacle}$= x-axis position of an obstacle

y_{robot}= y-axis position of the robot

$y_{obstacle}$= y-axis position of an obstacle

x_{new} = new x-axis position

$x_{initial}$ = initial x-axis position

y_{new} = new y-axis position

$y_{initial}$ = initial y-axis position

S_e = episode strength

x_{rmax} = maximum distance on the x-axis the robot traveled

N = number of movements performed by the robot

S_p = present strength

S_e = episode strength

S_u = updated strength

l = sensitivity factor

Δ = strength difference between episode and present strength

Chapter 1

Introduction

1.1 Overview

Darwinian evolution is embodied by the theory of generate, test, and adapt. This theory provides a strategy that can quickly identify and capitalize on regularities in the environment, according to John Holland [1]. This concept, that Holland studied, was then applied to develop a general purpose search algorithm using the theory of generate, test, and adapt (natural population genetics) to evolve solutions to problems.

Since Holland's [1] introduction of using genetic algorithms as a search method to provide solutions or improved solutions to problems, there has been many studies that use this type of search method to improve and/or try to optimize all sorts of situations including that of machine learning behavior. In many cases of machine learning, the actions or behaviors of the machine are governed by a set of stimulus-response rules. The rules are applied in a case specific manner triggered by environmental stimuli. The case that is examined by this thesis is one of a simple autonomous mobile robot whose actions and behaviors are controlled exactly as described above. It is the author's intention to explore the use of genetic algorithmic theory to provide an improved performance of the robotic model examined.

1.2 Research Objectives

The objective of this thesis is to provide a method by which an improved set of stimulus-response rules governing the actions and behavior of a simple pathway navigating/obstacle avoiding autonomous mobile robot. The method used to enhance the performance employs a simulation model, represented and tested through Matlab$^{©}$ software [2,3], combined with simple genetic algorithmic manipulation of the stimulus-response rules. The simple genetic algorithmic manipulation uses basic theory without the use of standard binary coding representation. Instead, a numeric model using conditional statements is used. The performance characteristics are determined by the length of the course navigated by the robot while maintaining it's prescribed boundaries and the duration of the performance. The actual simulation is not intended to be an all encompassing replica of an autonomous mobile robot, but a tool used to approximate its behavior to analyze and judge how altering the various stimulus-response rules affect its performance. The simulation is just a simplified instrument used in conjunction with the application of genetic algorithms to achieve the research objective. John J. Grefenstette [4] states that ideally, a simulation model should adequately reflect all the important aspects of the studied environment. In practice, however, the validation of the simulation model compared to the target environment is often quite difficult. As a result, conceivably the most important aspect of simulation-assisted learning is how well the results of the learning process convey to the target environment that may not conform exactly to the simulation model.

1.3 Background on Genetic Algorithms

Genetic algorithms can be described as a computational evolutionary procedure that utilizes genetic operators coupled with fitness evaluation and selection to positively adapt (gain knowledge) initial arithmetic model parameters of a simulation model into a significantly improved model whose parameters are genetic offspring of the original. Genetic algorithms (GA's) are general purpose adaptive search techniques derived from principles of natural population genetics[1]. The basic concept is one where there exists a population of possible solutions to a specific problem. The population then evolves over time through competition (survival of the fittest) and variation (genetic operations) leading to an "improved" population. Competition, in this case, can be defined as where individuals in a population vie for usage against each other. Competition leads to an individual's utility, which is evaluated to determine whether or not it is useful to the entire population and if not, it is to leave. Variation is carried out through genetic operations. These operations yield new members to be considered for entrance into the population. Genetic operations produce these new members (offspring) by utilizing a piece or pieces of existing members of the population (parents) combined with a piece or pieces that have been genetically altered in some fashion. A basic model or paradigm of genetic algorithms can be seen in Figure 1.

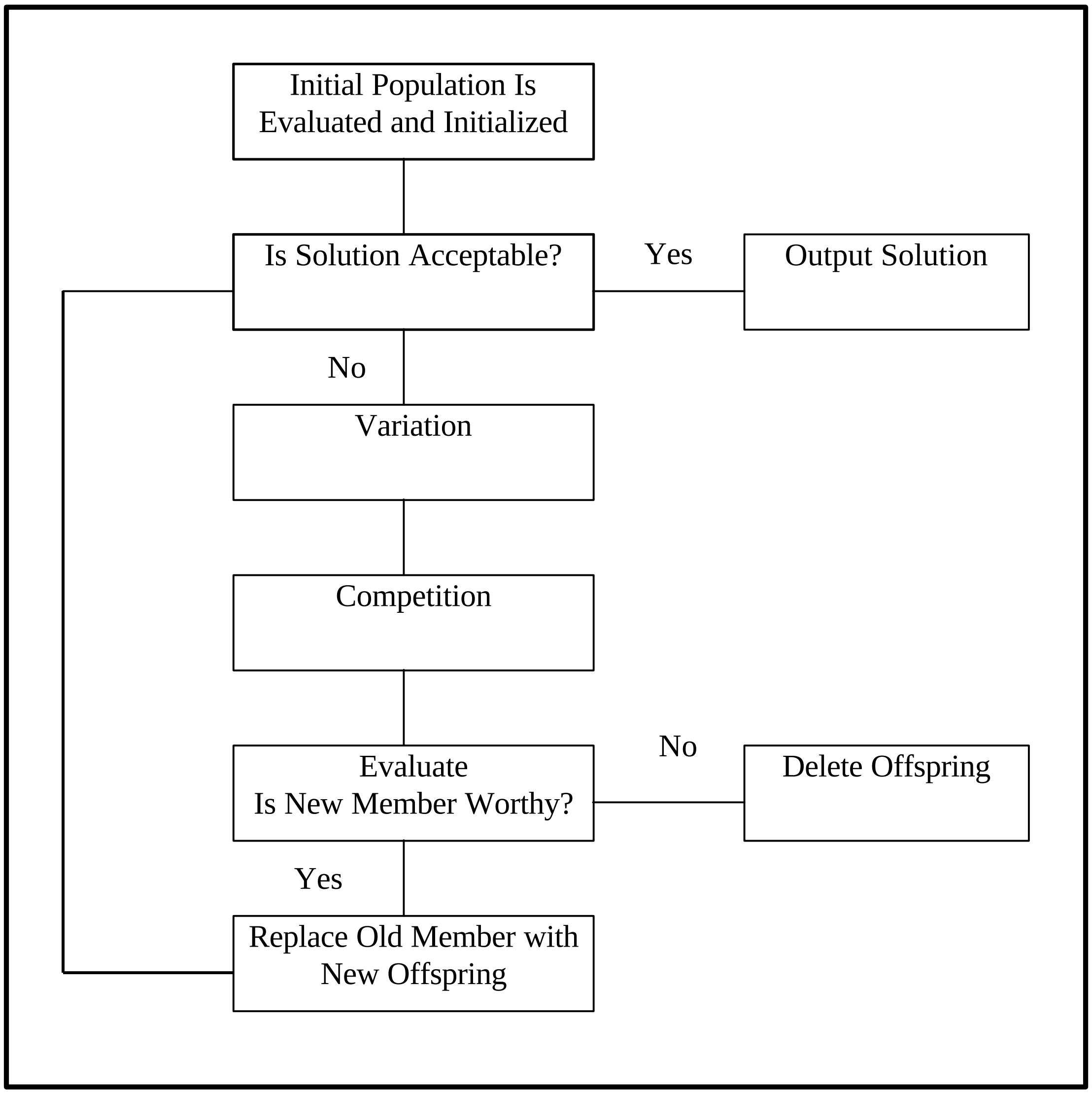

Initial Population Is Evaluated and Initialized
Is Solution Acceptable?
Yes
Output Solution
No
Variation
Competition
Evaluate Is New Member Worthy?
No
Delete Offspring
Yes
Replace Old Member with New Offspring

Figure 1. Paradigm of basic genetic algorithms.

Since Holland's [1] introduction of genetic algorithms as a search method for solutions to a problem, they have also been used a method for optimization. Genetic algorithms do provide a robust adaptive search method but do not always provide for an "optimal" solution. When speaking of genetic algorithms as an optimization package, there is a subtle important difference in semantics that needs to be addressed. Genetic algorithms are not function optimizers, but can lead to function optimization. This is best recognized as there may be infinite solutions to a problem which the optimal point may never be investigated or there are many or even no solutions to the problem [5]. Keeping this in mind, genetic algorithms coupled with fitness evaluation and selection do provide for a robust adaptive system that may or may not arrive at a global optimum, but will inherently lead to an improved solution whether optimal or not.

James Z. Tu [6] states that genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest of population members with a structured yet randomized information exchange to form a search algorithm. In every generation a new population is created using bits and pieces of the fittest of the old with an occasional new part tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

Lawrence Davis [7] states that for a genetic algorithm to solve a problem it must have the following five components.

1. a chromosomal representation of solutions to the problem,
2. a way to create an initial population of solutions,

3. an evaluation function that plays the role of the environment, rating solutions in terms of their “fitness,"
4. genetic operators that alter the composition of children during reproduction, and
5. values for the parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.)

Davis [7] states that in all of Holland’s work, and in the work of many of his students, chromosomes are bit strings - lists of 0’s and 1’s. These bit strings have been shown to be capable of usefully encoding a wide variety of information, and they have been shown to be affective representation mechanisms in unexpected domains (function optimization, for example). While bit strings seems to be the most popular and widely used method of chromosomal representation of solutions to a problem, there have been other representations with industrial applications of genetic algorithms. Examples of these other representations include ordered lists (for bin packing), embedded lists (for factory scheduling problems), variable-element lists (for semiconductor layout), and the representations used by Grefenstette [4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20]. These alternative representations are coupled with genetic operators that are not those traditionally utilized with bit strings as proposed by Holland.

1.3.1 Representation

This thesis uses one of these alternative representations instead of bit strings, namely the representations studied by the above mentioned Grefenstette and Alan C. Schultz [14, 15, 16, 17, 18, 19]. Even though the representations may be alternative, the previously mentioned five components that a genetic algorithm must have in order to solve a problem still pertains. The representation is used to identify each chromosome or member of the population of solutions to the task or problem. Each chromosome is composed of a variety of genes that determine the value of variables in the chromosome uniquely identifying it. The genes used in this thesis are in the form of distances (ranges) and angles. The genes then make up the chromosomes that are represented as a stimulus-response rule written in a Boolean algebraic form. The stimulus-response rules are in the form of an antecedent, a condition or stimulus, which is a sensor reading falling into a prescribed region or domain that triggers a consequence, an action or response. All stimulus-response rules (chromosomes) with their associated genes take the form of:

$$\text{if } \# \leq R_u < \# \text{ and } \# \leq R_o < \# \text{ and } \# \leq \beta < \# \text{ then } \theta = \# \qquad (1)$$

where,

$$R_u = \text{range to upper boundary} \qquad (2)$$

$$R_o = \text{range to obstacle} \qquad (3)$$

$$\beta = \text{bearing angle} \qquad (4)$$

$$\theta = \text{heading angle} \qquad (5)$$

$$\# = \text{either a distance or angle value where appropriate} \qquad (6)$$

The meaning of the genes and chromosomes are described completely in Chapter 3.

1.3.2 Initialization

In order to satisfy Davis's [7] second component of a genetic algorithm, that of a way to create an initial population of solutions to the task or problem, a starting point is needed. If there is no information available on possible solutions then one way to initialize a population is by purely creating it randomly. Another initialization method is known as seeding. Seeding is generating a population through already known solutions. Additionally, a third method, that is somewhat a subset of seeding is that of applying known heuristics to some of the population then randomly forming the rest. This third method could also include what is thought of as an educated guess. An educated guess is not necessarily applying already known solutions nor is it a random act. The initialization process used in this thesis is that of the third method. Initialization takes place by using an educated guess to form some parts of the original population while the genetic operations will form the rest. Section 2.6 discusses over and under constrained initial conditions that directly applies to initialization and Appendix B is a table depicting the initial stimulus-response rules utilized in this thesis.

1.3.3 Evaluation Function

The third component of a genetic algorithm is that of evaluation. Once an initial population has been formed, there needs to be a means to determine its "fitness". This also is true for each subsequent generation of the population. The evaluation is comprised of the objective function coupled with its constraints (Chapter 3). The objective function in this thesis is for the autonomous mobile robot model to successfully navigate the course model while maintaining boundaries, avoiding collision with obstacles, and other constraints set forth. The simulation provides the means for determining the degree of success in achieving the objective function which then the fitness is determined.

1.3.4 Derivation of Episode Strength

The episode strength, which can be also thought of as the payoff, is defined as an estimate of strength or the amount of utility the “active” stimulus-response rules (rules that were used in the sequence) have contributed to the particular episode tested. Every stimulus-response rule begins with an initial strength that is then updated at the end of each episode that increases the simulation performance for every rule that has been used or “fired." The payoff for each rule(s) utilized in an episode will be equal no matter how many times a rule(s) has fired. This can be thought of as an unbiased profit sharing model in which all rules that fire share equally in the payoff.

The primary goal of the robot is to successfully navigate the length of the course or pathway without coming into contact with the obstacles. The pathway is set up in a manner such that the degree of success can be measured by the aggregate distance traveled along the x-axis. Because of the structure in which the strength updating takes place, the episode strength is desired to take the form of a number with a value between zero and one (except in special cases to be described later). Since the pathway is defined to be 100 unit lengths along the x-axis, the shortest possible distance traveled by the robot will also be 100 unit lengths. This may not be achievable because of the contour of the pathway and the need to avoid obstacles but it does provide a defined minimum length that cannot be breached. The episode strength can then be defined as:

$$S_e = \text{episode strength} \tag{7}$$

$$x_{rmax} = \text{maximum distance on the x-axis the robot traveled} \tag{8}$$

$$S_e = x_{rmax} / 100 \tag{9}$$

There is a special case in which the value of the episode strength may be larger than then value of one. This situation occurs if the robot does successfully navigate the entire length of the course in which a secondary goal applies in which more utility is provided by a more efficient route taken. In other words, if more than one successful run is made, the run that took the fewest steps to achieve the goal is obviously more desirable than one that took more. Therefore, if a successful run is completed the strength will be given by:

$$S_e = \left({}^{100}\!/\!{}_{100} \right) + 100 / N \tag{10}$$

$$S_e = 1 + 100 / N \tag{11}$$

where

$$N = \text{number of movements performed by the robot} \tag{12}$$

1.3.5 Derivation of Stimulus-Response Rule Strength Update

Once a simulation run or episode is completed with an increase in performance and the episode strengths have been assigned, the stimulus-response rules are then updated by the following manner:

$$S_p = \text{present strength} \tag{13}$$

$$S_e = \text{episode strength} \tag{14}$$

$$S_u = \text{updated strength} \tag{15}$$

$$l = \text{sensitivity factor} \tag{16}$$

$$\Delta = \text{strength difference between episode and present strength} \tag{17}$$

By subtracting the present strength from the episode strength the difference between the two is found.

$$\Delta = S_e - S_p \tag{18}$$

The difference Δ will eventually be taken into account in the strength updating algorithm but before it is used it will be multiplied by a sensitivity factor l .

$$l\ \Delta = \text{sensitivity multiplied by the difference in strengths} \tag{19}$$

The sensitivity factor l has a value between 0 and 1. This will be explained later. By adding the difference in strengths multiplied by the sensitivity factor to the present strength an altered present strength that will increase incrementally directly correlated to the sensitivity factor.

$$S_p + l\ \Delta = \text{altered present strength} \qquad (20)$$

Then by adding the present strength to the altered present strength and dividing by two the mean is found which is then adopted as the updated strength.

$$S_u = (S_p + S_p + l\ \Delta) / 2 \qquad (21)$$

Which can also be seen as:

$$S_u = (S_p + S_p + l\ (S_e - S_p)) / 2 \qquad (22)$$

Note: The sensitivity factor l is utilized in a manner such that the strength update will only be modified in small increments so that the stimulus-response rule strengths will not by overly affected by a single episode performance .

It can be seen that if the value of l is equal to 1 then the update strength will just be the mean of the present strength and the present strength plus the strength difference.

$$\text{If } l = 1 \text{ then } S_u = (S_p + S_p + (S_e - S_p)) /2 \qquad (23)$$

It also can be seen that if the value of l is equal to 0 then the updated strength will never change and be equal to the present strength.

$$\text{If } l = 0 \text{ then } S_u = 2\ S_p / 2 \text{ or } S_p \qquad (24)$$

1.3.6 Genetic Operators

The fourth component of genetic algorithms are genetic operators. Genetic operators are functions that alter chromosomes or members of the population by transforming a gene or genes of those members they are applied to thus creating an “offspring”. Offspring therefore have some of the same genes as their parent and some may differ greatly. It is important to note that a genetic operator may have the capability of cloning a parent therefore creating an offspring identical to the parent.

The genetic operators applied in this thesis will be those of crossover, single gene mutation, all gene mutation, creep, and delete. Crossover is a genetic operation which creates an offspring by combining segments (genes) from one parent member (chromosome) with segments of another parent. When combining segments of two parents to create a new offspring, it should be obvious that that these segments should come from higher performing chromosomes. These new combinations from existing high performing genes may help improve the solution population. It should be noted, since the crossover operation creates an offspring by recombining genes that are already present in the population, crossover is limited in the exploration of the solution population to areas that already exist. This also limits the search to a finite number of possible solutions. The genetic operators of mutation and creep overcome this limitation.

Davis [7] states that the ability of a genetic algorithm to maintain a search depends on the continued ability of the crossover operator to perform effectively. What happens in practice is that if left unchecked, with only the best parents being crossed over, the population becomes more similar with the crossover operation becoming a replication operation rather than an operation that creates new and varied offspring through recombination. Diversity through crossover would then be minimized or lost.

The manner in which crossover is utilized in this thesis is to randomly recombine gene pair of the stimulus-response rule with the highest strength value with a gene pair of the stimulus-response rule randomly chosen from the top 25% of the highest strength values. The value of 25% is purely subjective. This limits the ability of lower performing genes to be considered for recombination while still encouraging variety. Since the stimulus-response rules are written in the form as seen in Equation 1, an example of crossover may take the form of:

A parent represented as,

$$\text{if } A \leq R_u < B \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \text{ then } \theta = M \tag{25}$$

and another parent represented as,

$$\text{if } G \leq R_u < H \text{ and } I \leq R_o < J \text{ and } K \leq \beta < L \text{ then } \theta = N \tag{26}$$

a possible offspring may be

$$\text{if } A \leq R_u < B \text{ and } C \leq R_o < D \text{ and } K \leq \beta < L \text{ then } \theta = M \tag{27}$$

or

$$\text{if } A \leq R_u < B \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \text{ then } \theta = N \tag{28}$$

By definition, used in this thesis, crossover will randomly recombine only gene pairs of the antecedent or the single gene of the consequence such as:

in the case of the antecedent

$$A \leq R_u < B \tag{29}$$

or

$$E \leq \beta < F \tag{30}$$

in the case of the consequence

$$M \text{ or } N \tag{31}$$

Single gene mutation creates new stimulus-response rules by generating random changes in a gene of an existing stimulus-response rule. These random changes then provide for the introduction of new genes or members into the population. Single gene mutation can alter any parameter or gene in the stimulus-response rule set, whether condition or action to any legal value for that attribute. Single gene mutate is randomly applied to any rule in the solution population except for a null rule. Examples of single gene mutation may take the form of:

In the case of the antecedent, a sensor reading may fall into a range (R_u) of 50 units to 100 units that is then randomly mutated to a range of 50 units to 75 units as follows:

$$\text{if } 50 \leq R_u < 100 \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \tag{32}$$

to

$$\text{if } 50 \leq R_u < 75 \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \tag{33}$$

Or in the case of the consequence a turning action may be initially $\theta = 60°$ which is then mutated to -45° as follows:

$$\theta = 60° \tag{34}$$

to

$$\theta = -45° \tag{35}$$

As long as these are legal values for the specified attributes.

Creep is very similar to mutation, in fact, it comprises a subset of mutation. Where mutation can replace any stimulus-response rule value with any legal value for that attribute, creep is limited or restricted in the sense that only small changes of legal values are made. This creates a more localized change or acts as “fine tuning". Examples of creep may take the form of:

In the case of the antecedent from

$$\text{if } 50 \leq R_u < 100 \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \tag{36}$$

to

$$\text{if } 50 \leq R_u < 95 \text{ and } C \leq R_o < D \text{ and } E \leq \beta < F \tag{37}$$

Or in the case of the consequence from

$$\theta = 30° \tag{38}$$

to

$$\theta = 32° \tag{39}$$

The concept of “stall” needs to be addressed. Stall is the condition where no further improvement of the performance by the existing solution population can be achieved even though an optimum solution may not have been attained. This may be due to the loss of diversity as previously mentioned by Davis [7]. With this in mind, the solution population may be reaching a local optimum. Because the stimulus-response rules are written in the form as seen by Equation 1, consisting of three antecedent parameter pair and one single consequence parameter, their genetic operations interaction may lead to stall. Every gene in the stimulus-response rule does have opportunity to single gene mutate into any other possible legal value for that particular parameter. A problem arises because the parameters are independent of each other, as far as their function is concerned; they are dependent upon each other to enter into the solution population. Their acceptance criteria is based upon how well the entire rule, not the individual genes, perform with the existing solution population.

With the method utilized in this study, genetic operators are used to create new rules (offspring) that after being tested through simulation are constantly modifying rules. The offspring will bear a resemblance to its parents, which if they are accepted into the solution population will then also be considered parents. In breeding or line breeding may then become a problem. With the rule's parameters being dependent upon each other to enter the solution population, genetic variation may then be limited or restricted. Just as in the animal kingdom, the passing down and recombination of "good" genes is dependent upon the quality and variation of those genes. If in breeding takes place for too long a period, a population may then become vulnerable to simple diseases, birth defects, and lowered survivability. In order to keep the population healthy; the introduction of "new blood" is necessary to provide for genetic variation.

An all gene mutation will provide the new blood for this study. All gene mutation will be defined as creating a new offspring, which will eventually become a potential parent if it enters the solution population, where each gene is created randomly through its single gene mutation operation. In the case of the gene pairs, there will be two new genes created. One gene value will be larger than the other unless the extremely rare case of equal values is created. Because the gene pairs are ordered by values, the larger value created will take the place of the larger valued gene and the same is obviously true for the smaller valued gene. All gene mutation is randomly applied to any rule in the solution population. The new offspring will then be tested to see if it will enter the solution population. This will alleviate the problem of in breeding and provide genetic variation, which in turn, leads to a more robust and adaptive solution population.

Delete is mostly self explanatory. It is where a stimulus-response rule or chromosome leaves the population. This thesis uses delete in the following manner. Once a rule enters the set of a stimulus-response rule base or population, it will remain there until it is deemed by evaluation that it is providing the least amount of utility to the entire population. Once this point is reached, it will be scheduled for genetic operations to act with a new chromosome being created to take its place which then leads to it being deleted from the population. Delete will be triggered by the strength value associated with the specific rule which is also defined as the amount of utility provided by the rule to the population. It is important to note that when a chromosome is deleted from the population that information that it contained is now lost. There is, however, a possibility that the information lost may be regained by the other genetic operators creating a new offspring that may be the same as the deleted chromosome. Therefore, information that is deleted does have the possibility to reenter the solution population.

1.3.7 Parameter Values

The fifth and final component of a genetic algorithm is that of values for the parameters that the genetic algorithm uses. While effective values of the parameters used in the running of genetic algorithms have been studied intensively for bit string representations, they have been less intensively studied for other representations [7]. One parameter value is that of population size or the number of chromosomes in the solution population. It goes without saying, the larger the population size the larger the computational time is needed to evaluate the population. This thesis uses a completely subjective population size of 25.

Another parameter is that of mutation rate. In this thesis, a single gene mutation operation will take place after five consecutive crossover operations. This leads to a mutation rate of approximately 25%. There is an exception to this rate which occurs when the mutated rule increases the success of the objective function. The mutated chromosome then undergoes the creep operation and then is tested and evaluated in the simulation again. This is to provide for the possibility of fine tuning or local hill climbing as described previously by creep. Also, the all gene mutation function will be triggered when there is no increase in performance in 25 consecutive genetic operations.

Summarizing Davis' [7] five components of a genetic algorithm as utilized in this thesis, the genetic algorithms are represented as Boolean algebraic expressions (chromosomes) that are comprised of various numerical values of distances and angles (genes). The initial population is created by known heuristics, educated guesses, and genetic operations producing the rest. The chromosomes are evaluated and ranked for their fitness in the population which then leads to genetic operations being performed on the population. This leads to diversified and possibly new information entering the population providing alternative solutions in which to search for an improved solution population. The genetic operations are governed by parameters set forth by strategy in designing the simulation and operations. The actual genetic algorithm utilized in this thesis can be seen in Figure 6 (Section 3.4).

1.4 Approach

The simulation consists of both a numerical and graphical representation of an autonomous mobile robot, its motion/position, pathway to be navigated by the robot, and obstacles in that pathway to be avoided with each element maintaining its own physical constraints. The graphical representation is to provide aide in visualization. The autonomous mobile robot's behaviors, i.e. motion and direction are portrayed as a series of stimulus-response rules which govern its decision making process. The robot's sensors provide the stimulus in which triggers a response of the robot's locomotion and navigation systems. In this case, the sensors will not be simulated but the sensors' output that is then applied as the input for the stimulus-response rule base will be simulated. Artificial randomly generated "noise" will be introduced to these sensor outputs as to provide a more realistic model that will be discussed later (Section 2.5).

The goal of the robot simulation is to successfully navigate the prescribed pathway while staying within the pathway guidelines and avoiding collision with the obstacles in that pathway. The approach being used is that of initially testing the performance of the autonomous mobile robot through the use of a simulation. The advantages of using a simulation versus manual testing should be quite evident. The time needed for a simulated run is vastly reduced versus the actual operation of a vehicle. This becomes even more pronounced when looking at the possibility of tens of thousands or even millions of iterations of the process which a computer can handle readily. Also, wear and tear on a vehicle is reduced and of course the damage to a vehicle due to inappropriate stimulus-response rules or mistakes is greatly reduced.

The initial simulation testing ceases, as will be the case in all future cases, when either the robot traverses a pathway guideline, travels retrogressively (in a negative direction on the x-axis, see Sections 3.1 and 3.1.2), collides with an obstacle, or successfully finishes the prescribed course. This will constitute a single episode. The performance is then recorded and evaluated with a payoff then assigned to each of the stimulus-response rule(s) that were utilized. Payoff will be defined as an estimate of strength or the amount of utility the "active" stimulus-response rules, rules that were used in the sequence, have contributed to the particular episode tested. Every stimulus-response rule begins with an initial strength that is then updated at the end of each episode for every rule that has been employed or "fired." The payoff for each rule(s) utilized in an episode will be equal in value no matter how many times a rule(s) has fired. This is as an unbiased profit sharing model in which all rules that fire share equally in the payoff. In example, a rule that fired only once will be assigned the same payoff as a rule that may have fired hundreds of times. The rationale behind this thinking can be thought of similarly to a chain is only as strong as its weakest link, whereas the stimulus-response rule base is a set of rules dependent upon each other to achieve a single goal. A rule may only be used very infrequently but may be the only rule that is pertinent to the situation which, if not used, will cause the entire sequence of events to fail. This rule interdependence is therefore a tactical plan which improves and possibly maximizes the robot's performance [4].

Once the stimulus-response rule(s) have had their strengths updated they will then be sorted by ascending strength values. By doing so, the manner in which the stimulus-response rules vie for stimulus (the bidding process) is then greatly reduced to simply associating the first stimulus-response rule whose antecedent corresponds with the set of parameters given by the stimulus. Since the stimulus-response rules are in an ascending order of strengths, once a rule is associated with the given stimulus it is not necessarily its final rule. If there is another rule with a higher strength value that also fits the stimulus, it will overshadow the first. This process is iterated throughout the ordered stimulus-response rule population which then leads to the rule with the highest strength value that conforms to the stimulus is applied to the simulation. If there is no rule that conforms to the stimulus then a default rule of the heading = 0 is applied. There is no need for any complicated bidding process or conflict resolution as the ascending sorting of strengths leads to a LIFO (last in first out) queuing operation when the stimulus-rules match the environmental input.

The application of the simple genetic algorithmic manipulation is applied in this logic:

1. Initialize the solution population.
2. Simulate the task.
3. Evaluate and record the results.
4. Did the simulation achieve the objective function? If yes then stop, if no continue.
5. Assign and update strengths to the stimulus-response rule(s) that participated. If any of the offspring rules were temporary and they achieved a higher strength value than the rule with the lowest value then permanently replace them. If they did not, replace the original rule with the lowest strength value.

6. Sort and arrange stimulus-response rules by strengths.
7. If there has been 25 consecutive genetic operations with no increase in strength then all genes mutate.
8. Did mutation or creep operations cause an increase in strength? If yes go to line # 11 (creep), if no continue.
9. Has the crossover operation been utilized the last four consecutive operations? If yes go to line # 13 (crossover), if no continue.
10. Randomly mutate any parameter of the stimulus-response rule with the highest strength value to create a new offspring that will temporarily replace (delete) the rule with the lowest strength value.
11. Return to line #2 (simulation).
12. Creep the highest strength rule to create new offspring and temporarily replace the rule with the lowest strength value. If the creep operation caused increased strength values, creep the same parameter again of the same rule. This is the same as local hill climbing.
13. Return to line #2 (simulation).
14. Crossover the rules with the highest and fifth highest strength value to create new offspring and temporarily replace the rule with the lowest strength value.
15. Return to line #2 (simulation).

A block diagram of the logic for the simple genetic algorithm used in this thesis can be seen in Figure 2.

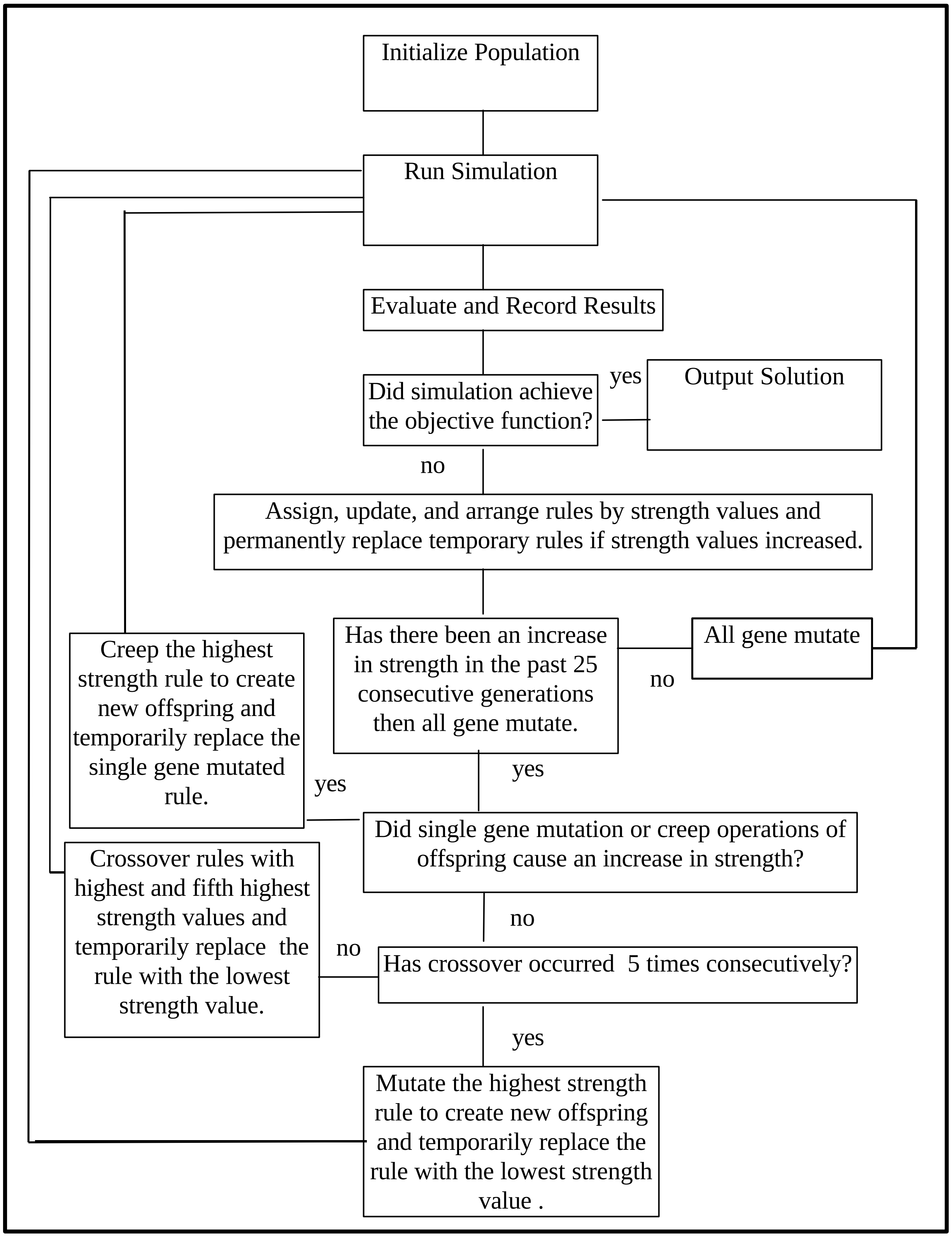

Initialize Population
Run Simulation
Evaluate and Record Results
Did simulation achieve the objective function?
yes
Output Solution
no
Assign, update, and arrange rules by strength values and permanently replace temporary rules if strength values increased.
Has there been an increase in strength in the past 25 consecutive generations then all gene mutate.
no
All gene mutate
Creep the highest strength rule to create new offspring and temporarily replace the single gene mutated rule.
yes
yes
Did single gene mutation or creep operations of offspring cause an increase in strength?
no
Crossover rules with highest and fifth highest strength values and temporarily replace the rule with the lowest strength value.
no
Has crossover occurred 5 times consecutively?
yes
Mutate the highest strength rule to create new offspring and temporarily replace the rule with the lowest strength value .

Figure 2. A block diagram of the logic for the strategic genetic algorithm used in this thesis.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

There have been many research investigations in the field of robotics of how genetic algorithms can be used to learn and improve complex behaviors. This thesis focuses on behaviors that are controlled by stimulus-response rule bases. These can be characterized as a dynamic system whose decision making agents interact with either a discrete time based system, position based system, or a combination of the two. In a 1990 Navy Center for Applied Research in Artificial Intelligence (NCARAI) report authored by Grefenstette, Ramsey, and Schultz [4] a sequential decision task is described as system in a specific state at the beginning of each time step whose agents observe the current state and selects one of a finite set of actions based upon decision (stimulus-response) rules. As a result, the system then enters a new state and the process then repeats.

This process was extended to a series of NCARAI reports and research by Alan C. Schultz, John J. Grefenstette [13,17, 19] and others in which an autonomous robot named RoboShepherd was to learn various complex behaviors through the use of genetic algorithms. Their research with RoboShepherd dealt mainly with pathway navigation and obstacle detection that then lead to a series of complex behaviors of the robot. These papers were the main stimulus of this thesis. This research then ushered in the development of the SAMUEL Learning System, a machine learning program that uses genetic algorithms and other competition-based heuristics to solve sequential decision problems [20]. Although this system was not used in this thesis, it did provide quite a bit of useful background and insight which did influence this thesis.

There are also numerous other NCARAI reports [6, 8, 11, 12, 13, 14, 15, 16, 17, 20] with related subjects using the SAMUEL Learning System that have also been of great interest. A few of the topics reviewed are: genetic algorithms and function optimization, how “noise” affects simulation-assisted learning, and learning sequential decision rules using simulation models and competition. These are just a few of topics examined in this thesis.

2.2 Learning Sequential Decision Rules Using Simulation Models

In a 1990 paper with the above title written by Grefenstette, Ramsey, and Schultz [4] deals with learning tactical decision rules from a simple flight simulator. Their study focuses what is called the evasive maneuvers problem. The problem, simply stated, is that of determining the best course of action for a plane to avoid being hit by an approaching missile. The missile tracks the position of the plane and steers toward its anticipated position for interception. It is the goal of the plane to maneuver in such a manner, using information about the missile from its sensors, to avoid interception. The planes maneuver’s are governed by a set of stimulus-response rules in which the planes sensors provide the stimulus in which triggers the response. They describe the sequential decision tasks by the following. A decision making agent interacts with a discrete-time dynamical system in an iterative fashion. At the beginning of each time step, the system is in some state. The agent observes a representation of the current state and selects one of a finite set of actions, based on the agent's decision rules. As a result, the dynamical system enters a new state and returns a (perhaps null) payoff. This cycle repeats indefinitely. The objective is to find a set of decision rules that maximizes the expected payoff.

This report utilizes the application of genetic algorithms at a level of a tactical plan. This is to state the stimulus-response rules applied were thought of as an entire set or plan comprised of the utilized rules, as opposed to each rule acting alone. This approach was determined to be best suited for reactive planning compared to projective planning. Therefore, it is very well suited for a constantly changing environment.

This report also touches on the fact that a dynamic system has quite a large state space, in fact, infinite. This is reduced and controlled by the granularity, discreteness, of both the input and output of the sensors that still may provide for millions of individual stimulus-response states. To be useful, the stimulus-response rules learned by the system must therefore have a reasonable amount of generality.

Also briefly touched on in this report is the thought of delayed payoff. This is where the payoff, or the amount of utility provided by the stimulus-response rules, is not determined until the end of an episode. This obviously fits very well with the tactical plan described earlier. It is stated that if payoff was immediately assigned after each decision then other methods for learning could be applied. However, for complex problems a reliable critic and or method of evaluation would probably approximate the manual knowledge engineering effort required to create an optimal set of stimulus-response rules. This requires quite a bit of human analytical effort that is not the intention of the study.

And finally the concept of noise is briefly introduced. They basically state the obvious that sensors are realistically going to have some type of "noise" associated with them in which may provide incorrect or imprecise information concerning the current state of the system. This obviously complicates the learning task. The concept of noise and how to deal with it is also researched on another paper by the same authors titled *Simulation-assisted Learning by Competition: Effects of Noise Differences Between Training Model and Target Environment* [15], which is addressed in section 2.5.

2.3 RoboShepherd

A series of studies previously mentioned have been performed by Alan C. Schultz, John J. Grefenstette, and others using a Nomad 200 autonomous robot to learn complex behavior patterns. The task performed is that of shepherding where one robot, RoboShepherd, tries to guide another robot, the sheep, into a specific target without coming into contact with it. In these studies, a complex behavior is controlled by a set of stimulus-response rules that governs RoboShepherd's actions. The sheep reacts to the shepherd by moving away from it. Otherwise its movement is random. Both robots use an array of 16 sonar sensors, tactile sensors, and a structured light range finder to provide input for the stimulus. Both robots are controlled by a set of stimulus-response rules but only the shepherd's rules are learned.

In a 1996 paper by Alan C. Schultz, John J. Grefenstette, and William Adams [32] titled *RoboShepherd: Learning a Complex Behavior* the authors report on how using the SAMUEL Learning System in conjunction with simulation, the RoboShepherd was able to successfully maneuver the sheep into the pasture 86% of the time. The set of stimulus-response rules that provided this success rate was provided by the 250th generation of the

SAMUEL Learning System coupled with the simulation. This set of stimulus-response rules was then applied to the actual robots for testing. The actual robots succeeded 67% of the time. The lower success rate could be explained by the shepherd losing track of the sheep and communication failures. Neither of these conditions were possible in the simulation. If these instances were removed the success rate was observed to be 73%.

The RoboShepherd task and how it was attacked using the SAMUEL Learning System in conjunction with simulation provided the impetus for this thesis. The shepherd utilized sensors to locate objects, the sheep, maintain pathway guidelines, and locate a target, the pasture, coupled with stimulus-response rules in order to achieve its objective of guiding the sheep to its pasture. Likewise the goals of the autonomous robot in this thesis are to successfully maintain pathway guidelines and locate obstacles in that pathway. In somewhat the complement of finding the sheep and guiding it to the pasture, the autonomous robot in this thesis wants to locate obstacles and stay away or guide themselves around them. Similar logic would still apply.

2.4 The SAMUEL Learning System

The SAMUEL Learning System is a machine learning program that uses genetic algorithms and competition-based heuristics to solve sequential decision problems. The system actively explores the space of alternative decision policies in simulation, and modifies its candidate policies based on this experience [20]. SAMUEL stands for Strategy Acquisition Method Using Evolutionary Learning. This software was explored to be utilized in this thesis. Although SAMUEL has the capability to provide precisely what is needed to fulfill the objectives of the autonomous robot utilized in this thesis, it was found to be far to complicated to meet the thesis objective of using simple genetic algorithmic manipulation and therefore not used. SAMUEL did, however, provide a solid logic base from which to work. SAMUEL is designed for sequential decision tasks in which the feedback or payoff occurs at the end of an episode that may consist of several stimulus-response rules being applied. Therefore, general rules for genetic algorithmic manipulation used in this thesis could be drawn using a loose fitting model of SAMUEL.

2.5 Noise

The concept of noise needs to be addressed. Noise, in this case, can be defined as the spurious data that is associated with an imperfect world. This can come from imperfections in the environment and/or imperfections in the hardware or sensing devices themselves. In a 1990 study by Connie Loggia Ramsey, Alan C. Schultz and John J. Grefenstette [15] titled *Simulation-assisted Learning by Competition: Effects of Noise Differences Between Training Model and Target Environment*, it was shown that the effect of learning tactical plans without noise then testing the plans in a noisy environment, and learning tactical plans with noise added then testing the plans in a noisy environment, it was empirically shown that using tactical plans with noise added, which more closely matches the target environment, provides best results. It follows that the closer the match between simulation and the target environment the more accurate the results. With the absence of a perfect match, it is better to have a less regular (noisy) training environment for the simulation than a more regular (less noisy) training environment.

2.6 Over and Under Constrained Initial Conditions

A learning system should be able to use to its advantage knowledge that is available. With this in mind, is it possible to over and/or under constrain the initial parameters (initial stimulus-response rules). In a 1992 paper written by John Grefenstette [12] titled *Learning Decision Strategies with Genetic Algorithms* demonstrated through experimentation that using genetic algorithms with a sequential decision learning task with little or no initial constraints have a very weak impact on the task. There is insufficient positive experiences on which to build. On the opposite side of the spectrum, if there are too many or too strict of initial constraints, this may limit the behavior beyond what is desired. What was determined was that a hybrid approach was best. This approach furnishes the sequential decision learning task with enough initial constraints (stimulus-response rules) to function at a minimal level of competence. The system then refines these initial constraints to improve on their performance.

Chapter 3

METHODOLOGY

3.1 Simulation Model

The simulation model in this study will be based upon a two dimensional graphical and numerical right hand Cartesian coordinate system performed with Matlab© software. All the simulation attributes and parameters, including constraints are defined by simple geometric shapes and functions. The reason for the simplicity of these components was to avoid any complicated kinematics that would greatly increase the complexity of the simulation. As stated in the introduction, the goal of this thesis is to show that the application of the theory introduced will aide in an improved set of stimulus-response rules governing the actions or behavior of an autonomous robot, not to provide an all encompassing precise detailed simulation of the robot and its environment. Also the robot is assumed to move at a constant speed and therefore any movement will be treated as movement in a specified heading, θ, for a unit distance.

3.1.1 Autonomous Robot Model

The robot model (Figure 3.) used in the simulation is represented as a circle whose center is located at (h, k) and whose radius, r, can be defined as:

$$r^2 = (x - h)^2 + (y - k)^2 \qquad (40)$$

or

$$r = \pm\sqrt{(x - h)^2 + (y - k)^2} \tag{41}$$

This also can be shown in the form of:

$$y = \pm\sqrt{r^2 - (x-h)^2} + k \tag{42}$$

r will be defined as:

$$r = 1 \tag{43}$$

Therefore:

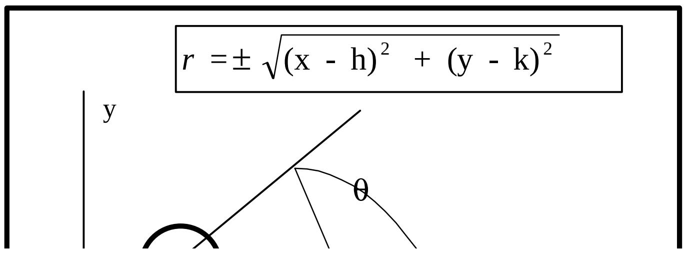
$$y = \pm\sqrt{1-(x-h)^2} + k \tag{44}$$

The motion of the robot will simply be that of the center point of the circle represented as a single point moving with a uniform acceleration along a straight line at a heading of q for a unit distance on a Cartesian coordinate system. With this in mind the location of the center point of the robot after each unit move can be determined as:

$$x = \cos q \tag{45}$$

and

$$y = \sin q \tag{46}$$



3.1.2 Pathway Model

As stated previously, since the robot is assumed to move with a constant speed at unit increments, the length of the course or pathway is set in unit lengths. The pathway will be considered 100 units in length and be bounded by two parallel polynomial functions (Equations 47 and 48). Polynomial functions were utilized as pathway model boundaries as to provide a continuous boundary that can be mathematically and graphically defined and evaluated at any point along the pathway. Although the polynomial functions look complex, they are simply the product of polynomial regression to subjective data points modeling the desired pathway.

The upper bound is defined by the function,

$$y = 1.365552747075397\text{e-}008\ x^6 - 3.854053894435499\text{e-}006\ x^5 + 3.984636770991189\text{e-}004\ x^4 - 1.834868905217691\text{e-}002\ x^3 + 3.588291378474007\text{e-}001\ x^2 - 1.965681703914941\text{e+}000\ x + 2.519654741393028\text{e+}001 \quad (47)$$

and the lower bound is defined by the horizontal line,

$$y = 1.365552747075397e\text{-}008\, x^6 - 3.854053894435499e\text{-}006\, x^5 + 3.984636770991189e\text{-}004\, x^4 - 1.834868905217691e\text{-}002\, x^3 + 3.588291378474007e\text{-}001\, x^2 - 1.965681703914941e{+}000\, x + 1.019654741393028e{+}001 \quad (48)$$

The pathway model can by seen in Figure 4.

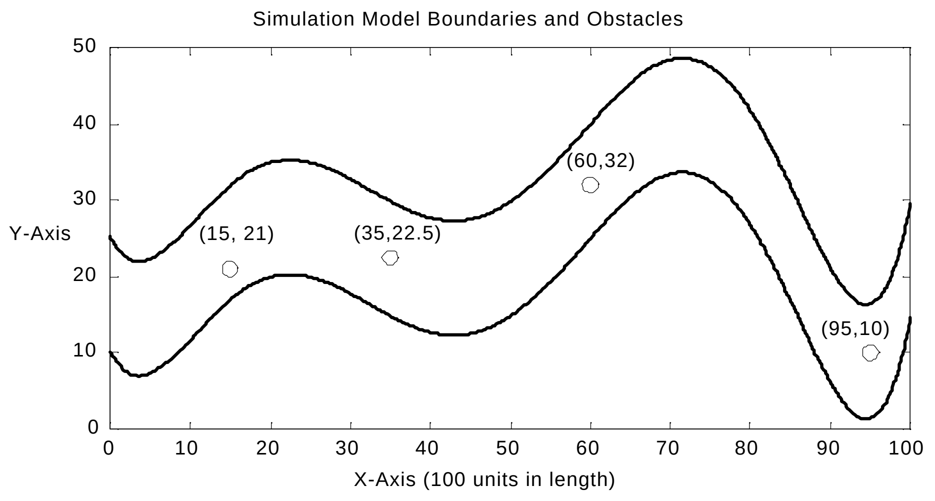


Figure 4. Simulation pathway and obstacle model.

3.1.3 Obstacle Model

The obstacles are modeled as circles with a radius of one unit similar to the robot model (see Equations 40 through 44). These circles are not dynamic in any way and therefore once defined they are immovable. There are four obstacles positioned on the course in such a manner so as to force the robot model to maneuver around them as it traverses the pathway. The centers of these obstacles are located at the following coordinates:

$$(15,21)$$

$$(35,22.5)$$

$$(60,32)$$

and

$$(95,10)$$

A graphical representation of the entire model is shown in Appendix A.

3.2 Mechanics

For the simulation, only the range from the upper bound, ascertained at a 90° angle, or perpendicular from the x-axis, is determined and/or needed since the course upper and lower boundaries are symmetric and parallel. If the range from the lower bound is established by subtracting the range from the upper bound from the width of the pathway. All ranges and angles are determined from the center points of the robot and obstacles. All angles, robot heading and bearing to obstacles, are measured from the x-axis intersecting the center point of the position of the robot model. These can be seen in Figure 5.

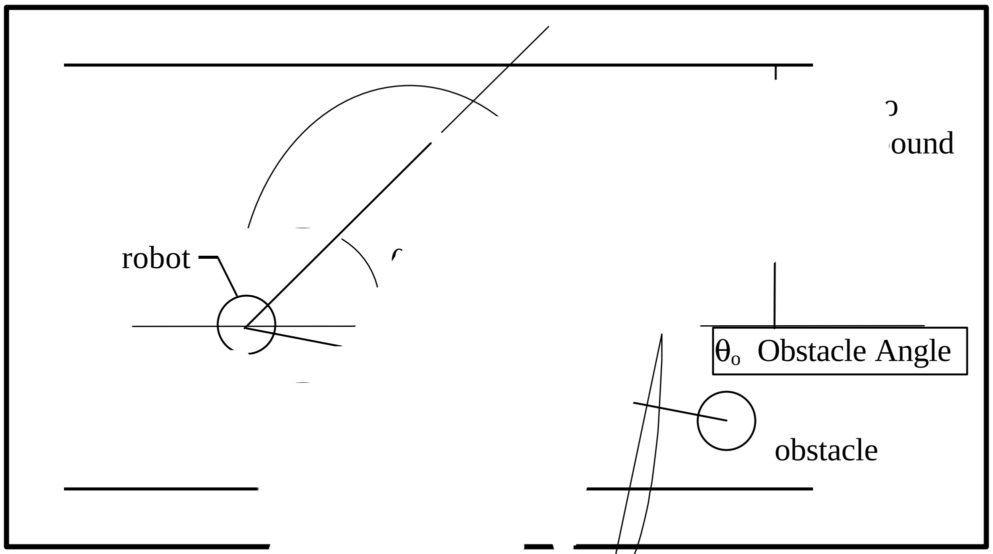


Figure 5. Graphica robot heading, bearing to obstacles, and range.

The range to the upp by subtracting the y-axis value for the upper bound at the x-axis position of the del center from the y-axis value of the robot model center.

$$\text{range to upper bound} = y_{\text{upper bound}} - y_{\text{robot}} \quad (49)$$

The range from the robot to the obstacles is found by taking the square root of the difference between x-axis center point positions squared then added to the difference between the y-axis center points squared (Equation 50).

$$\text{range to obstacle} = \sqrt{\left(x_{robot} - x_{obstacle}\right)^2 + \left(y_{robot} - y_{obstacle}\right)^2} \quad (50)$$

The cosine of the bearing to an obstacle, β, is determined subtracting the y-axis value of the center point of the obstacle from the y-axis value of the robot then dividing by the range to the obstacle (Equations 51 and 52).

$$\sin \beta = \left(y_{obstacle} - y_{robot}\right) \big/ \text{range to obstacle} \qquad (51)$$

or

$$\beta = \text{Asin}(\left(y_{obstacle} - y_{robot}\right) \big/ \text{range to obstacle}) \qquad (52)$$

It is important to note that the obstacle bearing angle (β) is found by subtracting the previous, or initial, robot heading angle (θ_i) from the angle the obstacle makes with the robot's position parallel to the x-axis (θ_o), as can be seen in Equation 53 and Figure 5. The initial robot heading and obstacle bearing angle is based off a line parallel with the x-axis going through the center of the robot's initial position. After the initial movement, the robot heading and obstacle bearing angle is based off the robots previous heading.

$$\beta = \theta_o - \theta_i \qquad (53)$$

The task of the robot is to successfully maneuver the length of the pathway without crossing an upper or lower bound while not coming into contact with an obstacle. The simulation initially starts with all positions defined (initial conditions and constraints) after which a response to a new position with a unit length is performed. Each subsequent response or action is then based upon the stimulus or conditions at that new position. This is to say that the new x-axis position for the robot model is equal to the initial x-axis position added to the cosine of the heading angle. The new y-axis position for the robot is equal to the initial y-axis position added to the sine of the of the heading angle. Keep in mind that the distance moved in the heading direction is equal to one unit distance.

$$x_{new} = x_{initial} + \cos\theta \qquad (54)$$

and

$$y_{new} = y_{initial} + \sin\theta \qquad (55)$$

As each step is incremented, the new value then becomes the initial value for the following step.

3.3 Initial Conditions and Constraints

Other than the upper and lower pathway boundaries and obstacle positions, both which have been defined in sections 3.1.2 and 3.1.3 respectively, the initial position of the robot model needs to be defined. The robot model initially starts every episode with its center point located at (2,15). This is a position that is centered on the pathway and is located such that no part of its circumference (its radius defined as 1 unit length) is touching a point where $x = 0$. This should not make any difference in any of the equations, but to be on the safe side the possibility is eliminated. Also, as stated earlier, every movement accomplished by the robot model will have a unit length in the angle of the heading.

The heading is defined to be constrained between or equal to the values of ± 90° (±1.5708 radians). This is to say that any value between -90° and +90° is valid.

The end of an episode has been defined as whenever the robot travels retrogressively, intersects with a pathway boundary, contacts an obstacle or successfully executes the course. For the robot to travel retrogressively, it simply has to have an x-axis value less than the previous x-axis value. This would mean that the robot traveled in a negative x-axis direction.

For the robot to end an episode by intersecting a pathway, all that is needed is that the difference between the upper boundary y-axis value at the center point of the robot and the y-axis center point value for the robot is less than or equal to one or the difference is greater than or equal to 9. Again, recall that the radius of the robot is defined as 1 unit length and the pathway width, perpendicular to the x-axis, has been defined as 15 unit lengths.

The situation when the robot comes in contact with an obstacle arises when the range of the obstacle is less than or equal to 2 unit lengths. Both the radius of the robot and the radii of the obstacles have been defined as being 1 unit length, therefore contact occurs when the range of the obstacle is less than or equal to 2 unit lengths. There is one slight flaw that occurs with this constraint. Whenever the center point of the robot has passed, the center point of the obstacle in a positive x-axis direction, the range is then negative and therefore less than 2 unit lengths. The robot can actually follow a heading that causes contact with an obstacle after the center points have passed. Therefore the determination of obstacle contact is established by calculating the absolute value of the obstacle range and setting that less than or equal to 2 unit lengths.

All the stimulus-response rules are written using Boolean algebraic operators in the form of:

$$\text{if } \# \le R_u < \# \text{ and } \# \le R_o < \# \text{ and } \# \le \beta < \# \text{ then } \theta = \# \tag{56}$$

where,

$$R_u = \text{range to upper boundary} \tag{57}$$

$$R_o = \text{range to obstacle} \tag{58}$$

$$\beta = \text{bearing angle} \tag{59}$$

$$\theta = \text{heading angle} \tag{60}$$

They too are constrained. Which means the genetic operators of mutate and creep are also constrained. As stated earlier, the upper boundary range has to fall in between a value of 1 and 14 unit lengths. Anything else would constitute the end of an episode. Therefore when mutating these range values, they too must fall between a value of 1 and 14. The same holds true for the creep operator which also has the stipulation, by definition for this simulation, that it not change any value more than ± 5 % for any single operation. The obstacle range is constrained to values between 2 and 60 unit lengths. By definition of this simulation, any range further than 60 unit lengths is to be ignored and anything less than 2 unit lengths is either in contact with the obstacle or the robot has successfully maneuvered past the obstacle(s). The latter case takes into account a negative value for the range where the robot has passed the obstacle(s). Again, when mutating the ranges must fall between 2 and 60 unit lengths and the same holds true for the creep operator. The heading and bearing angles are defined to be ± 90° and ± 180° (3.1417 and 6.2832 radians) respectively, which also applies to their mutate operators and again the same applies to the creep operators. The initial conditions for the stimulus-response rules can be seen in Appendix B.

The initial strength values are determined by a “cold” execution of the simulation with the initial stimulus-response rules in the order as seen in Appendix B. There episode strengths are then determined. The rules are then sorted in an ascending order to provide the initialized solution population for the rest of the simulation.

3.4 Simulation Strategy

The goal of the simulation is to improve the performance of the autonomous mobile robot model by genetically altering its set of stimulus-response rules. The strategy employed is to be constantly providing an improved set of stimulus-response rules (if not constantly improving at least equal to) through the use of simulation and application of genetic algorithms. The simulation is used to test and evaluate the changes that the application of genetic algorithms create. The genetic algorithm used in this thesis which is comprised of conditional statements can be seen in Figure 6.

Using a LIFO (last in first out) queuing system for the application of the stimulus-response rules in combination with the evaluation method, episode strength assignment, rule updating, and ascending sorting of the rules by strength values creates a strategic plan. The strategic plan also considers the solution population as a whole. Not as individuals. Therefore, the solution population is only as strong as its weakest link.

There are three separate occasions in the simulation and genetic operations phase in which random numbers are generated. The first is introducing random noise into the system, the second is a choosing function, and the third is the mutating and creep functions. The choosing function is one where which specific rules and parameters are randomly chosen for genetic operations. The third function randomly creates new values for the mutate and creep operations. In order to minimize any correlation between these random numbers each function has its own random number generator. The random number generators are uniform random number generators provided by Matlab$^{©}$, Excel, and Visual Basic respectively.

1. Initialize the population.
2. Run the simulation.
3. Evaluate the results of the simulation.
4. If the simulation achieved the objective function then output the solution.
5. Assign and update strengths of the rules.
6. If there has been 25 consecutive generations without improvement then all gene mutate, go to line #2.
7. If the temporary rule achieved a higher strength than the one it replaced then the replacement is permanent.
8. If the temporary rule did not achieve a higher strength than the one it replaced then the original rule is returned to the population.
9. Arrange rules by decreasing strength.
10. If the operations of single gene mutate or creep caused the offspring to have a greater strength value than the rule it temporarily replaced then perform the creep subroutine.
11. If step 10 did not cause a higher strength value and if the crossover operation has not been the last 5 consecutive genetic operations then perform the crossover subroutine.
12. If step 10 did not cause the offspring to have a greater strength value and the crossover subroutine was not the last 5 consecutive genetic operations then perform the single gene mutate subroutine.
13. Return to simulation, line #2.

*Note: Steps 2 - 9 are omitted for the first genetic algorithm application due to initialization.

Figure 6. Genetic algorithm comprised of conditional statements used in this thesis.

3.4.1 Derivation and Pseudocode for Genetic Algorithm

The population first needs to be initialized. A stated in Sections 1.3 and 1.3.2, the initial solution population will be created using what is best described as an educated guess to seed part of the initial population. The genetic operations will fill in the rest. Once the initial population is formed, strength values need to be assigned to each individual rule. The way this is performed is by executing the simulation with these rules and then assigning all the rules a strength value. Sort and arrange the rules in a ascending order.

Now that the population is fully initialized and a simulation and evaluation has already taken place, the rule with the highest strength value and a randomly chosen rule in the top 25% of the strength values will be chosen for the crossover operation. This is accomplished by:

- Count the number of rules that are not null (rule count).
- integer (rule count * 0.25) = number of rules in the top 25%.
- 1 / number of rules in the top 25% - 1. This assigns each rule in the top 25% a zone where it can be chosen. One is subtracted as the rule with the highest strength is providing the crossover gene pair.
- Generate a random number between 0 and 1.
- The rule which coincides with the random number will be chosen

The crossover operation will be executed as follows:

- A random number will be generated between the values of 0 and 1 (inclusive).

- There are 4 possible parameters per rule (3 pair of antecedent genes and 1 consequence gene), as stated in Section 1.3.4, that may be recombined. By definition used in this thesis, the value of the random number generated will determine a single parameter of the first rule to take the place of the same parameter in the chosen rule.
  - ∗ If $0 \leq$ the random number is > 0.25 then the first parameter is chosen.
  - ∗ If $0.25 \leq$ the random number is > 0.50 then the second parameter is chosen.
  - ∗ If $0.50 \leq$ the random number is > 0.75 then the third parameter is chosen.
  - ∗ If $0.75 \leq$ the random number is > 1.0 then the fourth parameter is chosen.

The new offspring created will now temporarily be assigned the position of the highest strength rule and the 25th rule (the rule with the lowest strength value) is temporarily deleted. The simulation is then executed with the new offspring temporarily in place. The solution population is then evaluated to see if the objective function has been reached. The objective function may be either a successful completion of the task or the maximum number of generations may have been reached. If the objective function has not been reached, the rules then have their strengths assigned and updated. If the temporary rules have a higher strength value than the rule they replaced temporarily, the change then becomes permanent. If not, the original rule is replaced back into the population.

Now that the genetic algorithm has been set is motion it will continue various subroutines until the objective function has been reached. The determination of which subroutine that will be utilized is again triggered by conditional statements. If there has been 25 or more consecutive generations with no improvement of performance then all gene mutate. This will be discussed later. If the crossover operation has not consecutively triggered 5 times in the last 5 genetic operations then it will trigger again. If the crossover operation has triggered 5 consecutive times in the last 5 genetic operations and the creep operation was not the last operation then the mutate operation will take place. The mutation operation will be executed as follows:

- Any rule can be chosen for mutation by random number generation.
- Each rule is assigned an range of 0.04 from 0 to 1 from which to be chosen.
- Generate a random number.
- The random number will choose the rule to mutate.
- Another random number will be generated between the values of 0 and 1 (inclusive).

There are 7 possible parameters per rule (6 antecedent genes and 1 consequence gene), as stated in Section 1.3.6, that may be mutated. Once mutated it will temporarily take the position of the rule with the highest value and temporarily delete the rule with the lowest strength value. The determination of which gene to mutate is executed in the following manner.

- A random number is generated with a value between 0 and 1 (inclusive). The random value will then determine which gene to mutate.
    - ∗ If $0 \leq$ the random number < 0.1429 then mutate the first gene.
    - ∗ If $0.1429 \leq$ the random number < 0.2857 then mutate the second gene.
    - ∗ If $0.2857 \leq$ the random number < 0.4286 then mutate the third gene.
    - ∗ If $0.4286 \leq$ the random number < 0.5714 then mutate the fourth gene.
    - ∗ If $0.5714 \leq$ the random number < 0.7143 then mutate the fifth gene.
    - ∗ If $0.7143 \leq$ the random number < 0.8571 then mutate the sixth gene.
    - ∗ If $0.8571 \leq$ the random number < 1.000 then mutate the seventh gene.

Once the determination of which gene is chosen, the actual mutation is performed in the following manner:

- Mutate the first gene. The first gene is the lower value of the upper bound range. It can have a value between one and nine as long as its value is less than the higher upper bound range.
- Generate a random number.
- The mutated first gene = (random number * 13) + 1
- If the mutated first gene = 1 or if mutated first value = 14 then mutate again.
- If mutated first gene $\geq$ second gene mutate again.
- Mutate the second gene. The second gene is the higher value of the upper bound range. It can have a value between 1 and 14 as long as its value is greater than the lower upper bound range.
- Generate a random number.
- The mutated second gene = (random number * 13) + 1.
- If mutated second gene = 1 or if mutated second gene = 14 then mutate again
- If mutated second gene $\leq$ first gene mutate again.
- Mutate the third gene. The third gene is the lower value of the obstacle range. It can have a value between 2 and 60 as long as its value is less than the higher obstacle range.
- Generate a random number.
- The mutated third gene = (random number * 58) + 2.
- If mutated third gene = 2 or if mutated third gene = 60 then mutate again.

- If mutated third gene ≥ fourth gene mutate again.
- Mutate the fourth gene. The fourth gene is the higher value of the obstacle range. It can have a value between 2 and 60 as long as its value is greater than the lower obstacle range.
- Generate a random number.
- The mutated fourth gene = (random number * 58) + 2.
- If the mutated fourth gene = 2 or if mutated fourth gene = 60 then mutate again.
- If the mutated fourth gene ≤ third gene mutate again.
- Mutate the fifth gene. The fifth gene is the lower value of the obstacle bearing. It can have a value between +180° (3.1416 radians) and -180° (-3.1416 radians) as long as its value is less than the higher obstacle bearing.
- Generate a random number.
- The mutated fifth gene = 2 * 3.1416 (0.5 - random number).
- If mutated fifth gene ≥ sixth gene then mutate again.
- Mutate the sixth gene. The sixth gene is the higher value of the obstacle bearing. It can have a value between +180° (3.1416 radians) and -180° (-3.1416 radians) as long as its value is greater than the lower obstacle bearing.
- Generate a random number.
- The mutated gene value = 2 * 3.1416 (0.5 - random number).
- If the mutated sixth gene ≤ fifth gene then mutate again.
- Mutate the seventh gene. The seventh gene is the value of the robot heading. It can have a value between +90° (1.5708 radians) and -90° (-1.5708 radians).

- Generate a random number.
- The mutated seventh gene = 3.1416 (0.5 - random number).

The new offspring created will now temporarily take the place of the rule with the lowest strength value. The simulation is then executed with the new offspring temporarily in place. The solution population is then evaluated to see if the objective function has been reached. The objective function may be either a successful completion of the task or the maximum number of generations may have been reached. If the objective function has not been reached, the rules then have their strengths assigned and updated**.** If the temporary rules have a higher strength value than the rule they replaced temporarily, the change then becomes permanent. If not, the original rule is replaced back into the population.

If the genetic operation is not in the crossover subroutine, less than 5 consecutive crossovers, and the single gene mutation did create an offspring of higher strength then this triggers the creep subroutine. If not, the single gene mutation operation is applied.

The creep subroutine is exactly identical to the mutate subroutine except that the values of the genes are only permitted to change by 10% of their total allowable range. This is also by definition in this thesis. The creep subroutine repeats itself on the same gene when the subroutine is invoked until no further increase in strength is realized. When it does repeat, it does not temporarily replace the rule with the lowest strength value but instead temporarily replaces itself. This is to ensure local hill climbing and to avoid many rules being created with similar genes such as the problem that may occur with crossover. Creep is performed as:

gene * (0.5-rand(1))/5

The all gene mutation operation is identical to the single gene mutation with the exception that all the genes of a randomly chosen rule are mutated. In the case of the gene pairs, there will be two new genes created. One gene value will be larger than the other unless the extremely rare case of equal values is created. Because the gene pairs are ordered by values, the larger value created will take the place of the larger valued gene and the same is obviously true for the smaller valued gene.

Chapter 4

Results

4.1 Initial Simulation

To begin the initial simulation the initial rule base had to be created. The concept of under and over constrained initial rules became apparent quickly. Very general rules were first applied with them quickly being identified as being under constrained. Because of the generality, a single rule selfishly controlled the actions of the simulation and would not allow any other rule to participate. This can be seen in Figure 7.

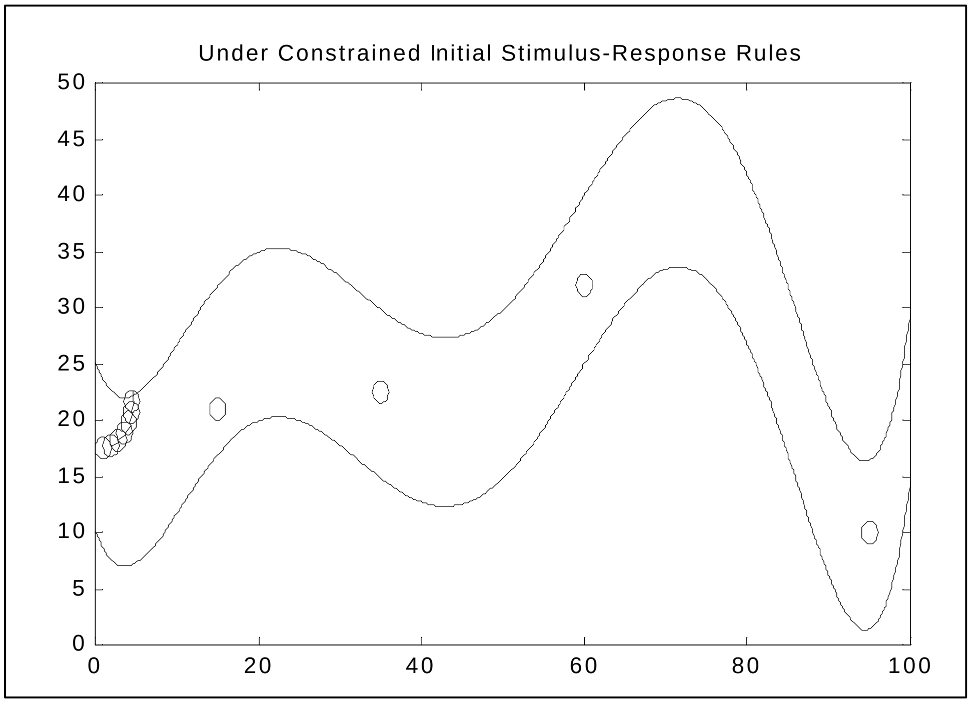


Figure 7. Under constrained initial stimulus-response rules.

The concept of being over constrained also was very evident. When applying very stringent stimulus-response rules, none would meet every parameter and therefore the default rule of heading = 0 would be the only rule applied. Therefore the robot simulation would only travel in a straight line.

After coming to grips with over and under constrained stimulus-response rules, an initial solution population was obtained (see Appendix B). The robot traveled 10.9409 units in the x-axis direction until crossing the lower boundary. This can be seen in Figure 8.

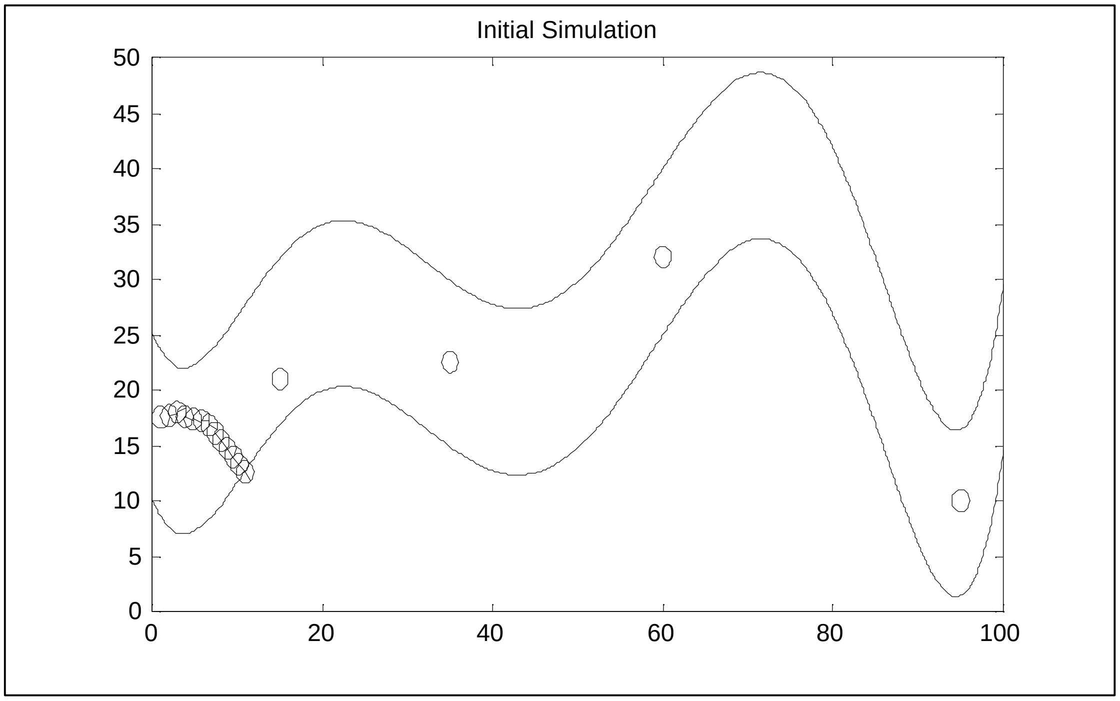


Figure 8. Initial simulation.

4.2 Results of Simulation and Genetic Algorithm Application

The application of the simulation and genetic algorithm described in this thesis did provide for an improved set of stimulus-response rules governing the actions and behaviors of the robot model in its modeled environment. In fact, the robot model was able to successfully navigate the entire length of the course while maintaining its boundaries while avoiding the obstacles in its pathway. The successful trial can be seen in Figure 9.

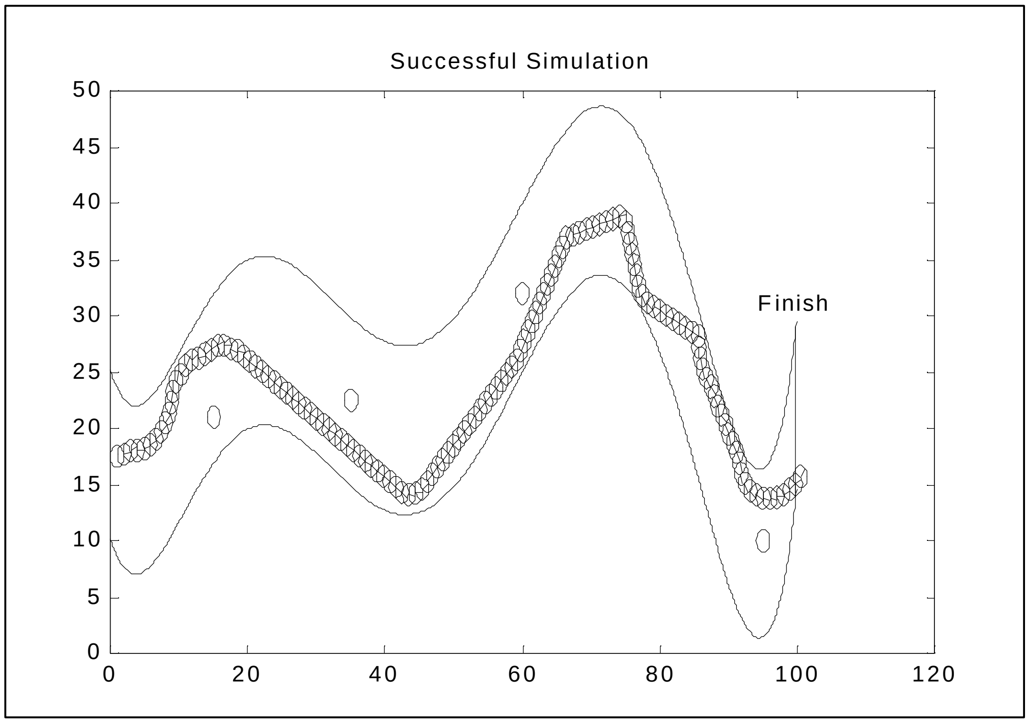


Figure 9. Successful simulation.

The successful simulation was achieved after 201 application of the genetic algorithm. Appendix C shows a few examples of the progression from start to finish of the entire process.

The single gene mutate operation was applied 12.44% of the total genetic operations. The all gene mutate operation was applied 17.41% of the total genetic operations. And the creep operator was applied 1.5% of the total genetic operations.

This leads to an overall mutation rate of 31.35%. The intended mutation rate was 25% plus the addition of creep operation therefore this rate is close to what was intended.

Figure 10 depicts the success, the distance traveled on the x-axis, of the robot versus the number of generations. The erratic movement or lower “spikes” are attributed to the fact that the offspring are temporarily given the position of highest strength value when tested with the simulation. If the offspring are not beneficial to the solution population then they may offer a very poor solution. They are then deleted from the solution population if they do not provide any improvement in performance. Also, by looking at Figure 10, this search method may be approaching an asymptote. This cannot be certain as the simulation was ended without the payoff giving credit for successfully completing the course was not applied.

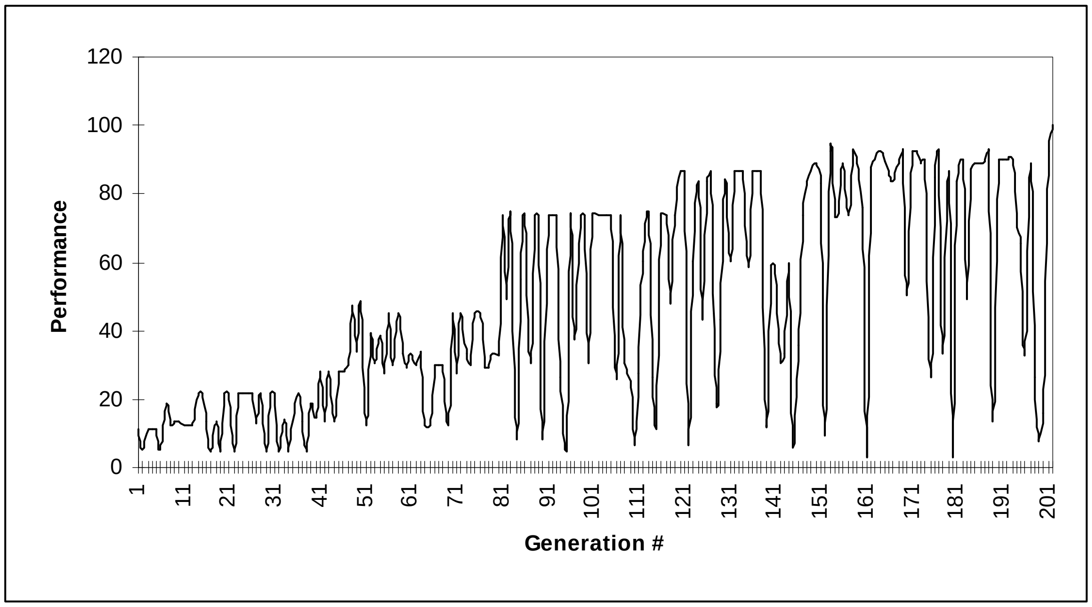


Figure 10. Performance vs. number of generations.

Chapter 5

Conclusions

5.1 Critique of Results

The application of the strategic application of the genetic algorithm combined with the computer simulation did provide a robust method for improving a solution population for the robotic and environmental model. The successful completion of the course did occur sooner than what was expected, but could be explained by the possibly the simplicity of the models or a combination of the simplicity with the event of being in the right place at the right time, as far as random generation of populations is concerned.

The successful solution is by no means an optimized solution. It just provided a solution population that did provide for a successful trial. It may or may not be possible for an optimized solution to be approached if the iterative process would be permitted to continue. Further testing and refinement would be necessary.

5.2 Direction of Future Work

As just stated, further testing and refinement would be the next step. It was recognized that some stimulus-response rules overlapped and were even subsets of other rules. The concept of fuzzy logic can easily be brought in to play with the decision of which decision rules to apply. It certainly applies to sets or subsets where there is partial sharing of members.

The goal of this thesis was to show that the theory involved did work. Now that it has been shown, in this case, some of the simplifications that were used can then be transformed into more complex and more true to situation characteristics. Also, the development of multiple testing compared to each other could be done in order to attach some statistical significance to the results.

The field of genetic algorithms is relatively new and therefore has room to expand. With the continued research in this type of theory, not only will the knowledge of this specific case increase but genetic machine based learning will increase leading to higher performance of all types of machines.

References

1. Holland, John H. 1975. Adaptation in Natural and Artificial Systems, The University of Michigan Press.

2. Matlab 5.3 Student Version. The MathWorks, Inc. Natick, MA.

3. Palm III, William J., 1999. Matlab© for Engineering Applications, McGraw-Hill, Boston.

4. Holland, John H. 1975. Adaptation in Natural and Artificial Systems, The University of Michigan Press.

5. Grefenstette, J. J., Ramsey, C. L. and Schultz, A. C. 1990. Learning Sequential Decision Rules Using Simulation Models and Competition, *Machine Learning* 5(4), 355-381. NCARAI Report AIC-90-010.

6. Tu, James Z. 1992. *Genetic Algorithms in Machine Learning and Optimization*, Ph.D. Dissertation, University of Cincinnati Department of Mechanical, Industrial and Nuclear Engineering. Cincinnati, Ohio.

7. Davis, Lawrence. 1987. Genetic Algorithms and Simulated Annealing. Pitman, London.

8. Grefenstette, J. J. 1989. Incremental learning of control strategies with genetic algorithms. *Proceeding of the Sixth International Workshop on Machine Learning*, Ithaca, NY: Morgan Kaufman, 340-344. NCARAI Report AIC-89-006.

9. Grefenstette, J. J. 1989. Learning rules from simulation models, *Proceedings of the 1989 International Association of Knowledge Engineers Conference*, Washington, DC: IAKE, 117-122. NCARAI Report AIC-89-008.

10. Grefenstette, J. J. 1990. (near a i) Genetic Algorithms and Their Applications. The Encyclopedia of Computer Science and Technology, Volume 21, Akent and J. G. Williams (editors), Marcel Dekker. NCARAI Report AIC-90-006.

11. Grefenstette, J. J. 1991. (near a i) Strategy acquisition with genetic algorithms. In *The Genetic Algorithms Handbook*, L. Davis (ed.), Boston: Van Nostrand Reinhold, 186-201. NCARAI Report AIC-91-013.

12. Grefenstette, J. J. 1992. Learning decision strategies with genetic algorithms. Invited paper, *Proceedings of the International Workshop on Analogical and Inductive*

Inference, Lecture Notes in Artificial Intelligence 642, Springer-Verlag, 35-50. NCARAI Report AIC-92-010.

13. Grefenstette, J. J. 1996. Genetic Learning for Adaptation in Autonomous Robots. In *Robotics and Manufacturing: Recent Trends in Research and Applications, Vol. 6,* M. Jamshidi, F. Pin and P. Dauchez (Eds.), Proceedings of the Sixth International Symposium on Robotics and Manufacturing, May 1996, ASME Press: New York, 1996, pp. 265-270. NCARAI Report AIC-96-022.

14. Gordon, Diana F., Alan C. Schultz, John J. Grefenstette, James Ballas, and Manuel A. Perez. 1994. User's Guide to the Navigation and Collision Avoidance Task. Navy Center for Artificial Intelligence (NCAI), Naval Research Laboratory, Washington DC, AIC-94-013, Internal Report.

15. Ramsey, C. L., Schultz, A. C. and Grefenstette, J. J. 1990. Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. *Proceedings Seventh International Conference on Machine Learning*, Sam Mateo, CA: Morgan Kaufmann, 211-215. NCARAI Report AIC-90-011.

16. Schultz, A. C. and Grefenstette, J. J. 1992. Using a genetic algorithm to learn behaviors for autonomous vehicles. *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation and Control Conference,* Hilton Head, SC, August 1992, AIAA, 739-749. NCARAI Report AIC-92-009.

17. Schultz, A. C., Grefenstette, J. J. and Adams, W. L. 1996. RoboShepherd: Learning a Complex Behavior. *Proceedings of the Robots and Learning Workshop (RoboLearn '96),* May 1996, Key West, Florida, pp. 105-113. NCARAI Report AIC-96-030.

18. Schultz, A. C., Grefenstette, J. J. and De Jong, K. A. 1993. Adaptive testing of controllers for autonomous vehicles. *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology,* June 1992, Washington: IEEE, 158-164. NCARAI Report AIC-92-004.

19. Schultz, A. C. 1994. Learning Robot Behaviors Using Genetic Algorithms. In *Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications, v1,* Mohammed Jamshidi and Charles Nguyen (Eds.). *Proceedings of the First World Automation Congress (WAC '94) and Fifth International Symposium on Robotics and Manufacturing (ISRAM '94),* 607-612, TSI Press: Albuquerque, NM. NCARAI Report AIC-94-003.

20. Grefenstette, John J. 1997. The User's Guide to SAMUEL-97: An Evolutionary Learning System. Navy Center for Applied Research in Artificial Intelligence. Code 5514. Naval Research Laboratory, Washington DC.

21. Dasgupta, D. and Z. Michalewicz. 1997. Evolutionary Algorithms in Engineering Applications. Springer-Verlag, New York.

22. Fujimura, Kikuo. 1991. Motion Planning in Dynamic Environments, Springer-Verlag, Tokyo.

23. Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company Inc., Reading, MA.

24. Ahrikencheikh, Cherif and Ali Seireg. 1994. Optimized-Motion Planning: Theory and Implementation, John Wiley & Sons, New York.

25. Agre, P. E. & Chapman, D. 1987. Pengi: An implementation of a theory of activity. *Proceedings Sixth National Conference on Artificial Intelligence*. Pp. 268-272.

26. Li, Zexiang and J. F. Canny. 1993. Nonholonomic Motion Planning, Kluwer Academic Publishers, Boston, MA.

27. Lilly, Kathryn W. 1993. Efficient Dynamic Simulation of Robotic Mechanisms, Kluwer Academic Publishers, Boston, MA.

28. Michalewicz, Zbigniew. 1996. Genetic Algorithms + Data Structures = Evolution Programs, Third, Revised and Extended Edition. Springer-Verlag, New York.

29. Shoup, Terry E., and Farrokh Mistree. 1987. Optimization Methods with Applications for Personal Computers, Prentice-Hall, Inc. Englewood Cliffs, New Jersey.

30. Whitley, L. Darrell. 1993. Foundations of Genetic Algorithms•2, Morgan Kaufman Publishers, San Mateo, California.

31. Zalzala, A. M. S. and A. S. Morris. 1996. Neural Networks for Robotic Control, Ellis Horwood, London.

Appendix A

Graphical Representation of Entire Simulation Model

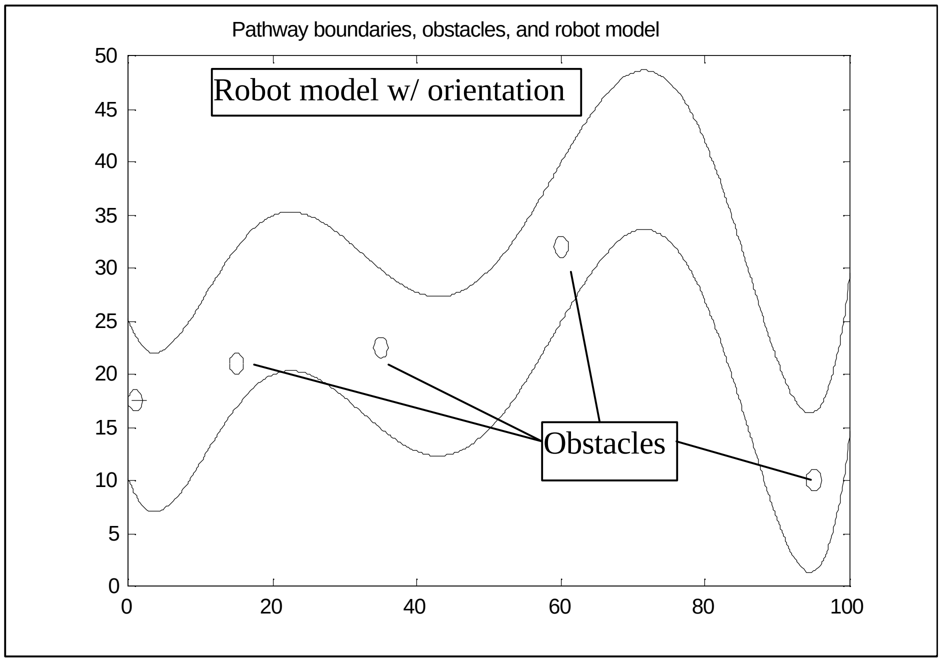


Figure 11. Graphical Representation of Entire Simulation Model

Appendix B

Initial Stimulus-Response Rules

Strength	Lower R_u	Upper R_u	Lower R_o	Upper R_o	Lower β	Upper β	θ
0	1	7	2	10	0	1.5708	-0.3491
0	1	7.5	10	20	0	1.5708	0.1745
0	7.5	14	2	5	-0.7854	0	1.5708
0	1	7.5	2	5	0	0.7854	-1.5708
0	1	7.5	20	60	-0.7854	0	0
10.9409	7.5	14	2	10	-1.5708	0	0.1745
10.9409	7.5	14	10	20	-1.5708	0	0.3491
10.9409	1	7.5	10	20	-0.7854	0	-0.7854
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Appendix C

Examples of Simulation Progression

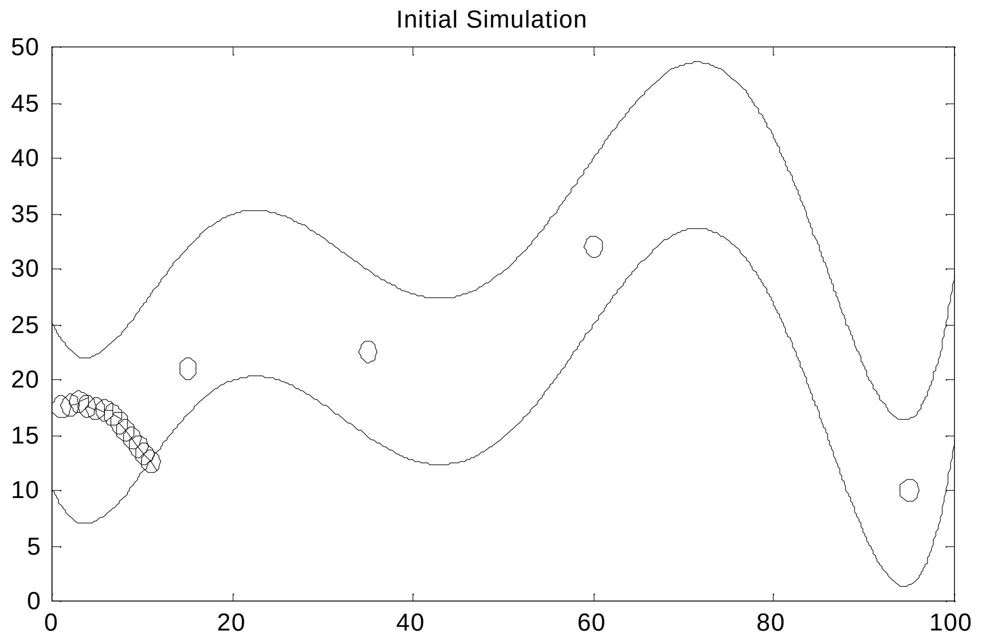


Figure 12.

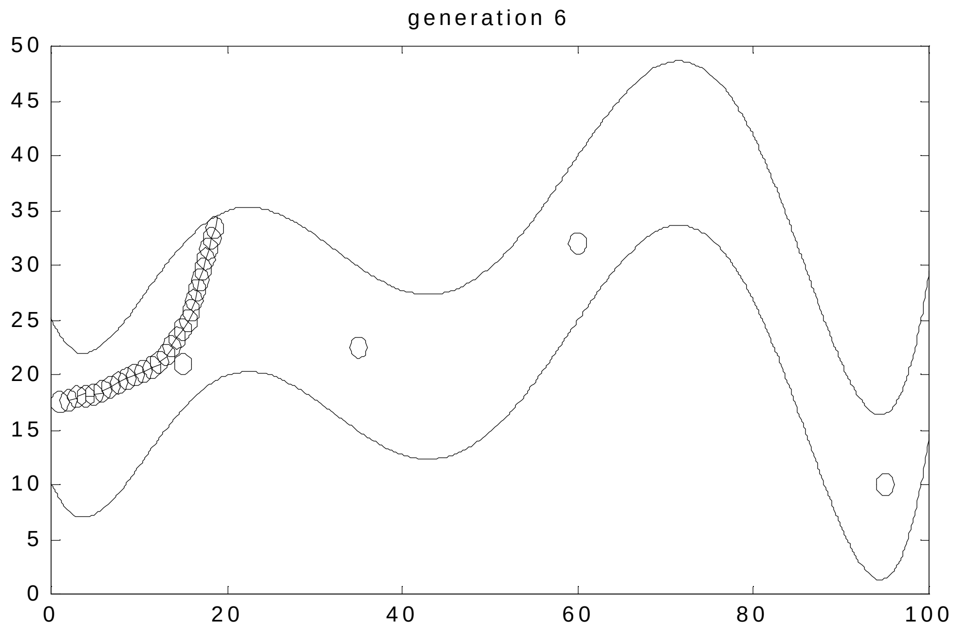


Figure 13.

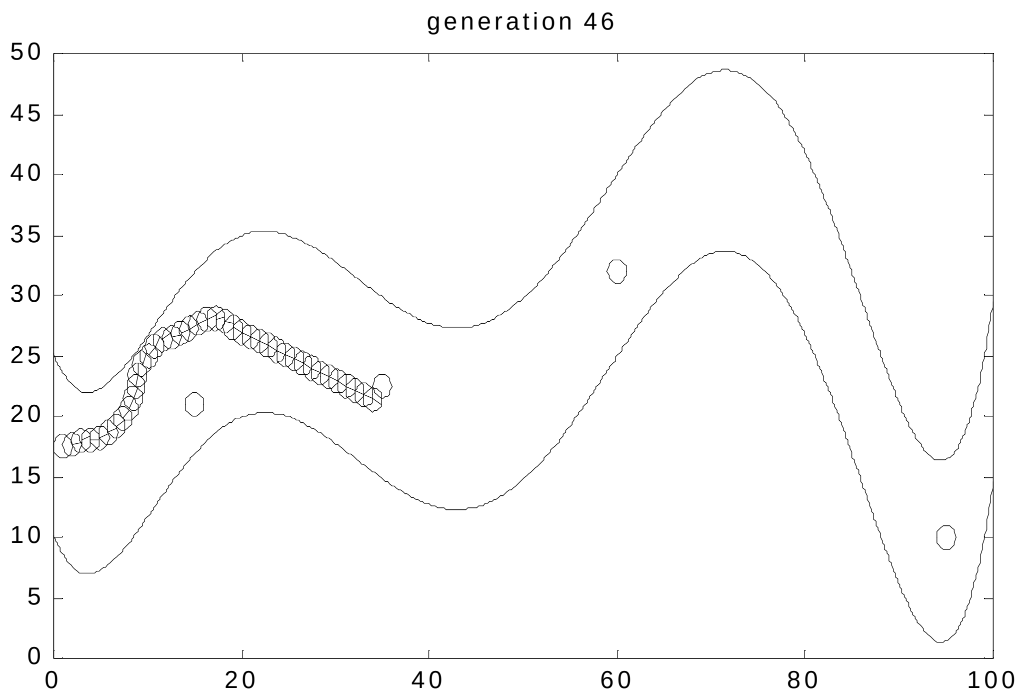


Figure 14.

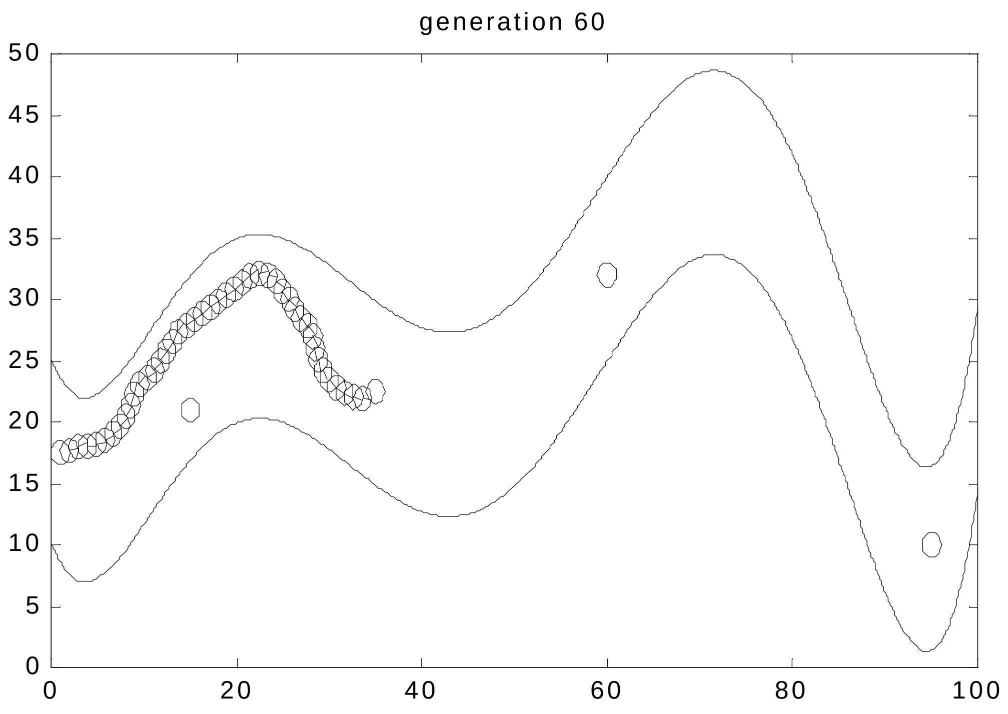


Figure 15.

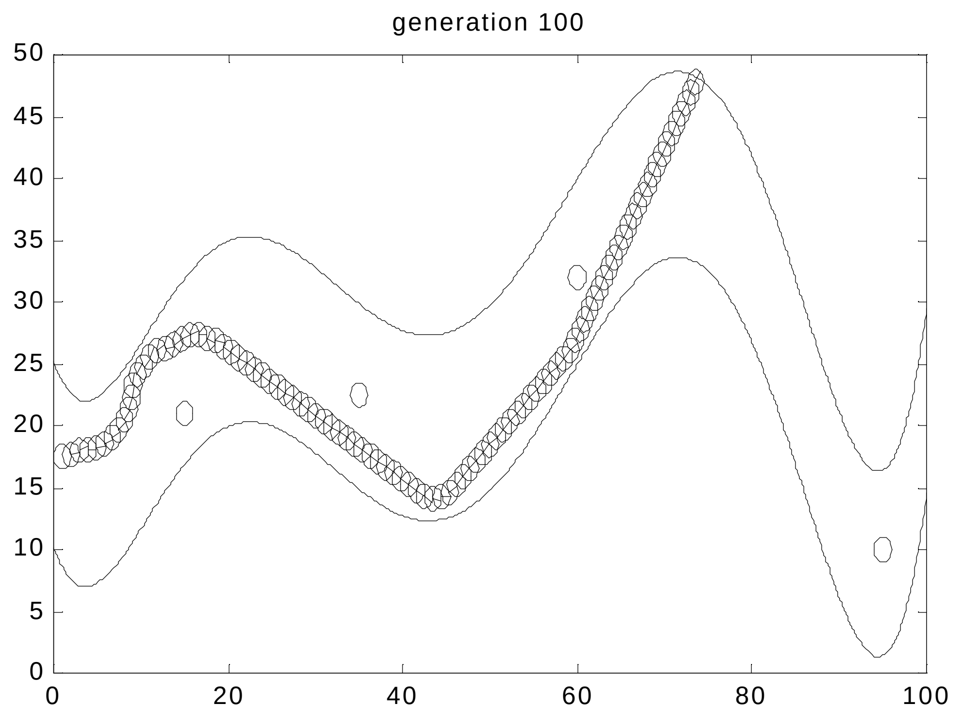


Figure 16.

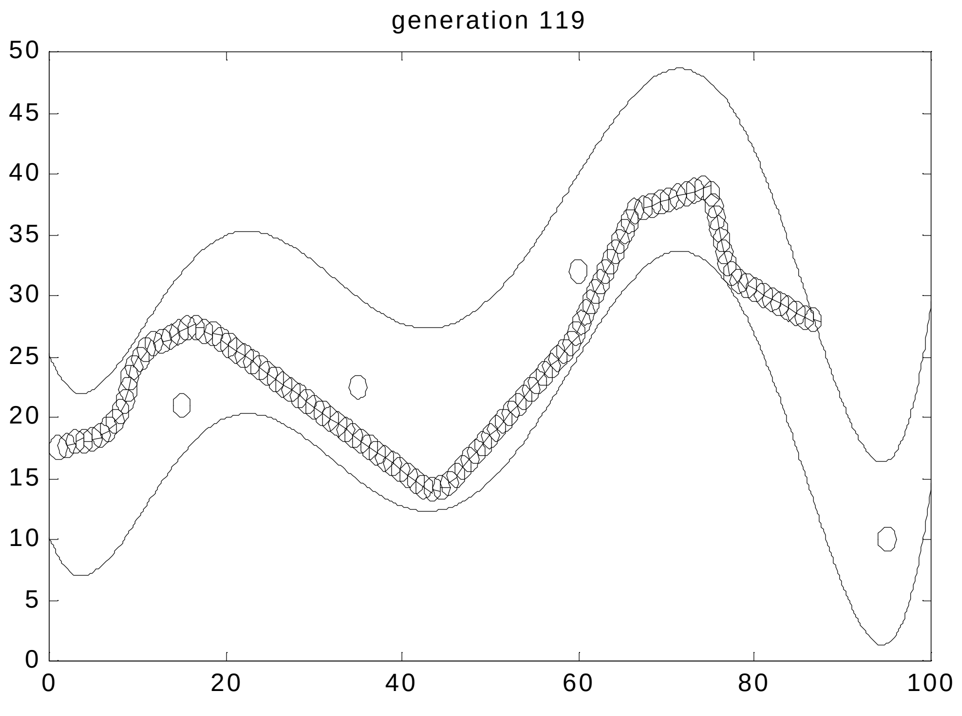


Figure 17.

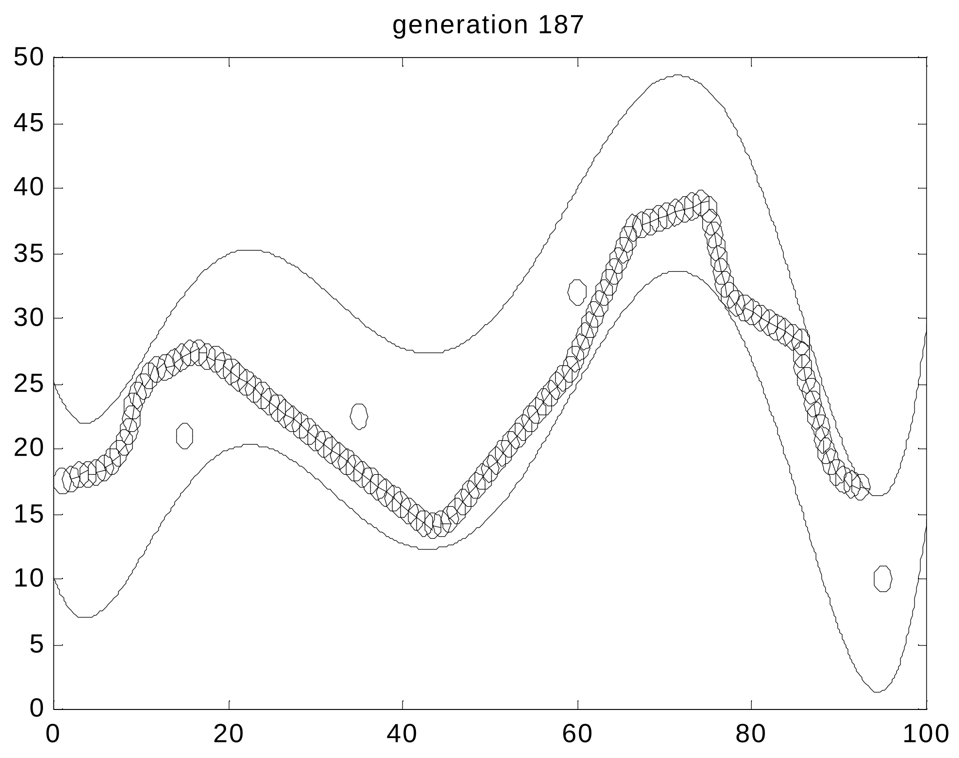


Figure 18.

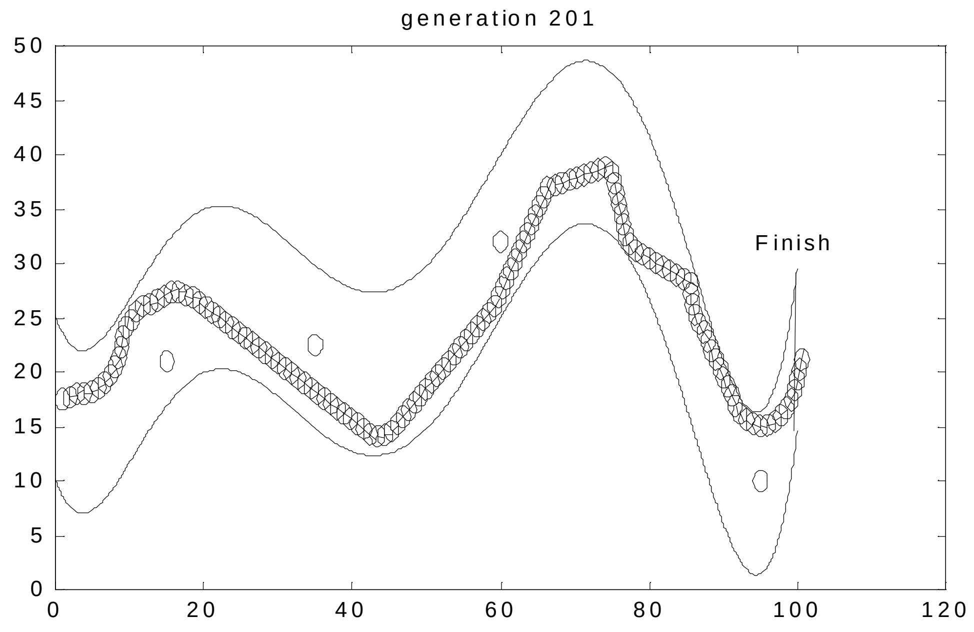


Figure 19.

Detailed Procedure of the Computer Simulation

The simulation is composed of two components, the testing phase and the manipulation of stimulus-response rules by genetic operations phase. The testing phase initially begins which then leads to the genetic operations phase. This then becomes a sequential procedure referring from one to the other and back again. This is the coding for the simulation written with Matlab©.

Testing Phase

```
1.  axis equal
2.  % plots the lower and upper pathway boundaries;
3.  [0:.1:100];
4.  y1=[1.365552747075397e-008,-3.854053894435499e-006,3.984636770991189e-
    004,-1.834868905217691e-002,3.588291378474007e-001,-
    1.965681703914941e+000,1.019654741393028e+001];
5.  y2=[1.365552747075397e-008,-3.854053894435499e-006,3.984636770991189e-
    004,-1.834868905217691e-002,3.588291378474007e-001,-
    1.965681703914941e+000,2.519654741393028e+001];
6.  b1=polyval(y1,x);
7.  b2=polyval(y2,x);
8.  plot(x,b1,'m',x,b2,'m');
9.  hold;
10. %plots the first obstacle;
```

```
[14:.1:16];
theta1=atan2((21+sqrt(1-(x-15).^2)),x);
theta2=atan2((21-sqrt(1-(x-15).^2)),x);
rho1=sqrt(x.^2+(21+sqrt(1-(x-15).^2)).^2);
rho2=sqrt(x.^2+(21-sqrt(1-(x-15).^2)).^2);
[x,y3]=pol2cart(theta1,rho1);
[x,y4]=pol2cart(theta2,rho2);
plot(x,y3,'r',x,y4,'r');
%plots the second obstacle;
[34:.1:36];
theta3=atan2((22.5+sqrt(1-(x-35).^2)),x);
theta4=atan2((22.5-sqrt(1-(x-35).^2)),x);
rho3=sqrt(x.^2+(22.5+sqrt(1-(x-35).^2)).^2);
rho4=sqrt(x.^2+(22.5-sqrt(1-(x-35).^2)).^2);
[x,y5]=pol2cart(theta3,rho3);
[x,y6]=pol2cart(theta4,rho4);
plot(x,y5,'r',x,y6,'r');
% plots the third obstacle;
[59:.1:61];
theta5=atan2((32+sqrt(1-(x-60).^2)),x);
theta6=atan2((32-sqrt(1-(x-60).^2)),x);
rho5=sqrt(x.^2+(32+sqrt(1-(x-60).^2)).^2);
rho6=sqrt(x.^2+(32-sqrt(1-(x-60).^2)).^2);
```

```
[x,y7]=pol2cart(theta5,rho5);
[x,y8]=pol2cart(theta6,rho6);
plot(x,y7,'r',x,y8,'r');
% plots the fourth obstacle;
[94:.1:96];
theta7=atan2((10+sqrt(1-(x-95).^2)),x);
theta8=atan2((10-sqrt(1-(x-95).^2)),x);
rho7=sqrt(x.^2+(10+sqrt(1-(x-95).^2)).^2);
rho8=sqrt(x.^2+(10-sqrt(1-(x-95).^2)).^2);
[x,y9]=pol2cart(theta7,rho7);
[x,y10]=pol2cart(theta8,rho8);
plot(x,y9,'r',x,y10,'r');
% plots the initial position of the robot;
[0:.1:2];
theta7=atan2((17.5+sqrt(1-(x-1).^2)),x);
theta8=atan2((17.5-sqrt(1-(x-1).^2)),x);
rho7=sqrt(x.^2+(17.5+sqrt(1-(x-1).^2)).^2);
rho8=sqrt(x.^2+(17.5-sqrt(1-(x-1).^2)).^2);
[x,y11]=pol2cart(theta7,rho7);
[x,y12]=pol2cart(theta8,rho8);
plot(x,y11,'b',x,y12,'b');
% initial robot coordinates;
xrobot=1;
```

```
yrobot=17.5;
%initial obstacle coordinates;
xobstacle1=15;
yobstacle1=21;
xobstacle2=35;
yobstacle2=22.5;
xobstacle3=60;
yobstacle3=32;
xobstacle4=95;
yobstacle4=10;
% dummy obstacle;
xobstacle5=150;
yobstacle5=20;
% termination counter to avoid going into an endless loop;
terminationcounter=0;
% Initializes robot x-axis new position;
xrobotnew=1;
% Initializes robot y-axis new position;
yrobotnew=17.5;
% Initializes robot heading;
headinginitial=0;
% Initializes obstacle bearing angles;
thetaorientation=0;
```

```
% Initialize stimulus-response rule firing counters;
rule1=0;
rule2=0;
rule3=0;
rule4=0;
rule5=0;
rule6=0;
rule7=0;
rule8=0;
rule9=0;
rule10=0;
rule11=0;
rule12=0;
rule13=0;
rule14=0;
rule15=0;
rule16=0;
rule17=0;
rule18=0;
rule19=0;
rule20=0;
rule21=0;
rule22=0;
```

```
rule23=0;
rule24=0;
rule25=0;
rule26=0;
% Starts an indefinite loop that stops when the condition of x>=100 is satisfied;
% At the point where x>=100 the robot has successfully navigated the course;
while xrobot<100;
    % Sets default heading to 0 if no rule applies;
        heading=0;
    % Starts episode counter;
    terminationcounter=terminationcounter+1;
    % Stops simulation if more than 300 episodes have occurred;
    if termination counter>=300;
            break;
    end;
    % Determines if the robot has traveled retrogressively;
    if xrobotnew<xrobot;
            break;
    end;
% If not then both xrobot and yrobot are iterated to their new coordinates
    respectively;
    xrobot=xrobotnew;
    yrobot=yrobotnew;
```

```
% Defines the range from the robot to each obstacle;
rangeob1=sqrt((xrobot-xobstacle1).^2+(yrobot-yobstacle1).^2);
rangeob2=sqrt((xrobot-xobstacle2).^2+(yrobot-yobstacle2).^2);
rangeob3=sqrt((xrobot-xobstacle3).^2+(yrobot-yobstacle3).^2);
rangeob4=sqrt((xrobot-xobstacle4).^2+(yrobot-yobstacle4).^2);
rangeob5=sqrt((xrobot-xobstacle5).^2+(yrobot-yobstacle5).^2);
% Alters the value of the range by ±2.5%, therefore noise is added;
rangewithnoiseob1=rangeob1+((0.5-rand(1))/20)*rangeob1;
rangewithnoiseob2=rangeob2+((0.5-rand(1))/20)*rangeob2;
rangewithnoiseob3=rangeob3+((0.5-rand(1))/20)*rangeob3;
rangewithnoiseob4=rangeob4+((0.5-rand(1))/20)*rangeob4;
rangewithnoiseob5=rangeob5+((0.5-rand(1))/20)*rangeob5;
% Defines the x-axis bearing angle between the robot and each obstacle;
xaxisbearingob1=asin((yobstacle1-yrobot)/rangewithnoiseob1);
xaxisbearingob2=asin((yobstacle2-yrobot)/rangewithnoiseob2);
xaxisbearingob3=asin((yobstacle3-yrobot)/rangewithnoiseob3);
xaxisbearingob4=asin((yobstacle4-yrobot)/rangewithnoiseob4);
xaxisbearingob5=asin((yobstacle5-yrobot)/rangewithnoiseob5);
% Defines bearing angle off previous heading;
bearingob1=thetaorientation-xaxisbearingob1;
bearingob2=thetaorientation-xaxisbearingob1;
bearingob3=thetaorientation-xaxisbearingob1;
bearingob4=thetaorientation-xaxisbearingob1;
```

```
      148.bearingob5=thetaorientation-xaxisbearingob1;
    149.% Defines the range of the robot from the upper boundary;
150.individualvaluesupper=1.365552747075397e-008*xrobot.^6-3.854053894435499e-
    006*xrobot.^5+3.984636770991189e-004*xrobot.^4-1.834868905217691e-
    002*xrobot.^3+3.588291378474007e-001*xrobot.^2-
    1.965681703914941e+000*xrobot+2.519654741393028e+001;
    151.robotrangetoupperbound=individualvaluesupper-yrobot;
    152.% Determines if the robot has crossed a pathway boundary;
    153.if robotrangetoupperbound<=1;
            154.break;
    155.end;
    156.if robotrangetoupperbound>=14;
            157.break;
    158.end;
159.% Determines if the robot has collided with any of the obstacles;
    160.if abs(rangeob1)<=2;
            161.break;
    162.end;
    163.if abs(rangeob2)<=2;
            164.break;
    165.end;
    166.if abs(rangeob3)<=2;
            167.break;
    168.end;
    169.if abs(rangeob4)<=2
            170.break
    171.end
172.% Determines if the robot has passed the first obstacle;
    173.if (0<=xrobot)&(xrobot<=15)
174.% Determines which stimulus-response rule to apply.  Each rule is composed of seven
    separate genes (a-f) and there are 25 rules total permitted in the solution population.
    Therefore every gene letter will have 25 values, i.e. a1, a2, a3,… a25;
175.if
    ((genea1<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb1))&((genec
    1<=rangeob1)&(rangeob1<=gened1))&((genee1<=bearingob1)&(bearingob1<=genef
    1))
                176.heading=geneg1
                177.rule1=rule1+1
        178.end;
179.if
    ((genea2<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb2))&((genec
    2<=rangeob1)&(rangeob1<=gened2))&((genee2<=bearingob1)&(bearingob1<=genef
    2))
                180.heading=geneg2
                181.rule2=rule2+1
        182.end;
```

```
183.if
    ((genea3<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb3))&((genec
    3<=rangeob1)&(rangeob1<=gened3))&((genee3<=bearingob1)&(bearingob1<=genef
    3))
                184.heading=geneg3
                185.rule3=rule3+1
        186.end;
187.if
    ((genea4<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb4))&((genec
    4<=rangeob1)&(rangeob1<=gened4))&((genee4<=bearingob1)&(bearingob1<=genef
    4))
                188.heading=geneg4
                189.rule4=rule4+1
        190.end;
191.if
    ((genea5<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb5))&((genec
    5<=rangeob1)&(rangeob1<=gened5))&((genee5<=bearingob1)&(bearingob1<=genef
    5))
                192.heading=geneg5
                193.rule5=rule5+1
        194.end;
195.if
    ((genea6<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb6))&((genec
    6<=rangeob1)&(rangeob1<=gened6))&((genee6<=bearingob1)&(bearingob1<=genef
    6))
                196.heading=geneg6
                197.rule6=rule6+1
        198.end;
199.if
    ((genea7<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb7))&((genec
    7<=rangeob1)&(rangeob1<=gened7))&((genee7<=bearingob1)&(bearingob1<=genef
    7))
                200.heading=geneg7
                201.rule7=rule7+1
        202.end;
203.if
    ((genea8<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb8))&((genec
    8<=rangeob1)&(rangeob1<=gened8))&((genee8<=bearingob1)&(bearingob1<=genef
    8))
                204.heading=geneg8
                205.rule8=rule8+1
            206.end;
207.if
    ((genea9<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb9))&((genec
    9<=rangeob1)&(rangeob1<=gened9))&((genee9<=bearingob1)&(bearingob1<=genef
    9))
```

```
                208.heading=geneg9
                209.rule9=rule9+1
            210.end;
211.if
    ((genea10<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb10))&((gen
    ec10<=rangeob1)&(rangeob1<=gened10))&((genee10<=bearingob1)&(bearingob1<=
    genef10))
                212.heading=geneg10
                213.rule10=rule10+1
            214.end;
215.if
    ((genea11<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb11))&((gen
    ec11<=rangeob1)&(rangeob1<=gened11))&((genee11<=bearingob1)&(bearingob1<=
    genef11))
                216.heading=geneg11
                217.rule11=rule11+1
            218.end;
219.if
    ((genea12<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb12))&((gen
    ec12<=rangeob1)&(rangeob1<=gened12))&((genee12<=bearingob1)&(bearingob1<=
    genef12))
                220.heading=geneg12
                221.rule12=rule12+1
            222.end;
223.if
    ((genea13<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb13))&((gen
    ec13<=rangeob1)&(rangeob1<=gened13))&((genee13<=bearingob1)&(bearingob1<=
    genef13))
                224.heading=geneg13
                225.rule13=rule13+1
            226.end;
227.if
    ((genea14<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb14))&((gen
    ec14<=rangeob1)&(rangeob1<=gened14))&((genee14<=bearingob1)&(bearingob1<=
    genef14))
                228.heading=geneg14
                229.rule14=rule14+1
            230.end;
231.if
    ((genea15<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb15))&((gen
    ec15<=rangeob1)&(rangeob1<=gened15))&((genee15<=bearingob1)&(bearingob1<=
    genef15))
                232.heading=geneg15
                233.rule15=rule15+1
            234.end;
```

```
235.if
    ((genea16<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb16))&((gen
    ec16<=rangeob1)&(rangeob1<=gened16))&((genee16<=bearingob1)&(bearingob1<=
    genef16))
              236.heading=geneg16
              237.rule16=rule16+1
          238.end;
239.if
    ((genea17<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb17))&((gen
    ec17<=rangeob1)&(rangeob1<=gened17))&((genee17<=bearingob1)&(bearingob1<=
    genef17))
              240.heading=geneg17
              241.rule17=rule17+1
          242.end;
243.if
    ((genea18<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb18))&((gen
    ec18<=rangeob1)&(rangeob1<=gened18))&((genee18<=bearingob1)&(bearingob1<=
    genef18))
              244.heading=geneg18
              245.rule18=rule18+1
          246.end;
247.if
    ((genea19<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb19))&((gen
    ec19<=rangeob1)&(rangeob1<=gened19))&((genee19<=bearingob1)&(bearingob1<=
    genef19))
                248.heading=geneg19
                249.rule19=rule19+1
            250.end;
251.if
    ((genea20<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb20))&((gen
    ec20<=rangeob1)&(rangeob1<=gened20))&((genee20<=bearingob1)&(bearingob1<=
    genef20))
                252.heading=geneg20
                253.rule20=rule20+1
            254.end;
255.if
    ((genea21<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb21))&((gen
    ec21<=rangeob1)&(rangeob1<=gened21))&((genee21<=bearingob1)&(bearingob1<=
    genef21))
                256.heading=geneg21
                257.rule21=rule21+1
            258.end;
259.if
    ((genea22<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb22))&((gen
    ec22<=rangeob1)&(rangeob1<=gened22))&((genee22<=bearingob1)&(bearingob1<=
    genef22))
```

```
            260.heading=geneg22
            261.rule22=rule22+1
        262.end;
263.if
    ((genea23<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb23))&((gen
    ec23<=rangeob1)&(rangeob1<=gened23))&((genee23<=bearingob1)&(bearingob1<=
    genef23))
            264.heading=geneg23
            265.rule23=rule23+1
        266.end;
267.if
    ((genea24<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb24))&((gen
    ec24<=rangeob1)&(rangeob1<=gened24))&((genee24<=bearingob1)&(bearingob1<=
    genef24))
            268.heading=geneg24
            269.rule24=rule24+1
        270.end;
271.if
    ((genea25<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb25))&((gen
    ec25<=rangeob1)&(rangeob1<=gened25))&((genee25<=bearingob1)&(bearingob1<=
    genef25))
            272.heading=geneg25
            273.rule25=rule25+1
            274.end;
        275.end;
    276.% Determines if the robot has passed the second obstacle;
        277.if (15<xrobot)&(xrobot<=35);
    278.% Determines which stimulus-response rule to apply;
    279.if
        ((genea1<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb1))&((genec
        1<=rangeob2)&(rangeob1<=gened2))&((genee1<=bearingob2)&(bearingob2<=genef
        1))
                280.heading=geneg1
                281.rule1=rule1+1
            282.end;
    283.if
        ((genea2<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb2))&((genec
        2<=rangeob2)&(rangeob2<=gened2))&((genee2<=bearingob2)&(bearingob2<=genef
        2))
                284.heading=geneg2
                285.rule2=rule2+1
            286.end;
    287.if
        ((genea3<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb3))&((genec
        3<=rangeob2)&(rangeob2<=gened3))&((genee3<=bearingob2)&(bearingob2<=genef
        3))
```

```
                288.heading=geneg3
                289.rule3=rule3+1
        290.end;
291.if
    ((genea4<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb4))&((genec
    4<=rangeob2)&(rangeob2<=gened4))&((genee4<=bearingob2)&(bearingob2<=genef
    4))
                292.heading=geneg4
                293.rule4=rule4+1
        294.end;
295.if
    ((genea5<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb5))&((genec
    5<=rangeob2)&(rangeob2<=gened5))&((genee5<=bearingob2)&(bearingob2<=genef
    5))
                296.heading=geneg5
                297.rule5=rule5+1
        298.end;
299.if
    ((genea6<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb6))&((genec
    6<=rangeob2)&(rangeob2<=gened6))&((genee6<=bearingob2)&(bearingob2<=genef
    6))
                300.heading=geneg6
                301.rule6=rule6+1
    302.end;
303.if
  ((genea7<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb7))&((genec
  7<=rangeob2)&(rangeob2<=gened7))&((genee7<=bearingob2)&(bearingob2<=genef
  7))
        304.heading=geneg7
        305.rule7=rule7+1
    306.end;
307.if
  ((genea8<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb8))&((genec
  8<=rangeob2)&(rangeob2<=gened8))&((genee8<=bearingob2)&(bearingob2<=genef
  8))
        308.heading=geneg8
        309.rule8=rule8+1
      310.end;
311.if
  ((genea9<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb9))&((genec
  9<=rangeob2)&(rangeob2<=gened9))&((genee9<=bearingob2)&(bearingob2<=genef
  9))
        312.heading=geneg9
        313.rule9=rule9+1
      314.end;
```

```
315.if
    ((genea10<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb10))&((gen
    ec10<=rangeob2)&(rangeob2<=gened10))&((genee10<=bearingob2)&(bearingob2<=
    genef10))
                316.heading=geneg10
                317.rule10=rule10+1
            318.end;
319.if
    ((genea11<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb11))&((gen
    ec11<=rangeob2)&(rangeob2<=gened11))&((genee11<=bearingob2)&(bearingob2<=
    genef11))
                320.heading=geneg11
                321.rule11=rule11+1
            322.end;
323.if
    ((genea12<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb12))&((gen
    ec12<=rangeob2)&(rangeob2<=gened12))&((genee12<=bearingob2)&(bearingob2<=
    genef12))
                324.heading=geneg12
                325.rule12=rule12+1
            326.end;
327.if
    ((genea13<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb13))&((gen
    ec13<=rangeob2)&(rangeob2<=gened13))&((genee13<=bearingob2)&(bearingob2<=
    genef13))
                328.heading=geneg13
                329.rule13=rule13+1
            330.end;
331.if
    ((genea14<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb14))&((gen
    ec14<=rangeob2)&(rangeob2<=gened14))&((genee14<=bearingob2)&(bearingob2<=
    genef14))
                332.heading=geneg14
                333.rule14=rule14+1
            334.end;
335.if
    ((genea15<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb15))&((gen
    ec15<=rangeob2)&(rangeob2<=gened15))&((genee15<=bearingob2)&(bearingob2<=
    genef15))
                336.heading=geneg15
                337.rule15=rule15+1
            338.end;
339.if
    ((genea16<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb16))&((gen
    ec16<=rangeob2)&(rangeob2<=gened16))&((genee16<=bearingob2)&(bearingob2<=
    genef16))
```

```
            340.heading=geneg16
            341.rule16=rule16+1
        342.end;
343.if
    ((genea17<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb17))&((gen
    ec17<=rangeob2)&(rangeob2<=gened17))&((genee17<=bearingob2)&(bearingob2<=
    genef17))
            344.heading=geneg17
            345.rule17=rule17+1
        346.end;
347.if
    ((genea18<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb18))&((gen
    ec18<=rangeob2)&(rangeob2<=gened18))&((genee18<=bearingob2)&(bearingob2<=
    genef18))
            348.heading=geneg18
            349.rule18=rule18+1
        350.end;
351.if
    ((genea19<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb19))&((gen
    ec19<=rangeob2)&(rangeob2<=gened19))&((genee19<=bearingob2)&(bearingob2<=
    genef19))
            352.heading=geneg19
            353.rule19=rule19+1
            354.end;
355.if
    ((genea20<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb20))&((gen
    ec20<=rangeob2)&(rangeob2<=gened20))&((genee20<=bearingob2)&(bearingob2<=
    genef20))
                356.heading=geneg20
                357.rule20=rule20+1
            358.end;
359.if
    ((genea21<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb21))&((gen
    ec21<=rangeob2)&(rangeob2<=gened21))&((genee21<=bearingob2)&(bearingob2<=
    genef21))
                360.heading=geneg21
                361.rule21=rule21+1
            362.end;
363.if
    ((genea22<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb22))&((gen
    ec22<=rangeob2)&(rangeob2<=gened22))&((genee22<=bearingob2)&(bearingob2<=
    genef22))
                364.heading=geneg22
                365.rule22=rule22+1
            366.end;
```

```
367.if
    ((genea23<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb23))&((gen
    ec23<=rangeob2)&(rangeob2<=gened23))&((genee23<=bearingob2)&(bearingob2<=
    genef23))
            368.heading=geneg23
            369.rule23=rule23+1
        370.end;
371.if
    ((genea24<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb24))&((gen
    ec24<=rangeob2)&(rangeob2<=gened24))&((genee24<=bearingob2)&(bearingob2<=
    genef24))
            372.heading=geneg24
            373.rule24=rule24+1
        374.end;
375.if
    ((genea25<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb25))&((gen
    ec25<=rangeob2)&(rangeob2<=gened25))&((genee25<=bearingob2)&(bearingob2<=
    genef25))
            376.heading=geneg25
            377.rule25=rule25+1
        378.end;
    379.end;
380.% Determines if the robot has passed the third obstacle;
```

```
  381.if (35<xrobot)&(xrobot<=60);
382.% Determines which stimulus-response rule to apply;
383.if
  ((genea1<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb1))&((genec
  1<=rangeob3)&(rangeob3<=gened1))&((genee1<=bearingob3)&(bearingob3<=genef
  1))
            384.heading=geneg1
            385.rule1=rule1+1
     386.end;
387.if
  ((genea2<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb2))&((genec
  2<=rangeob3)&(rangeob3<=gened2))&((genee2<=bearingob3)&(bearingob3<=genef
  2))
            388.heading=geneg2
            389.rule2=rule2+1
     390.end;
391.if
  ((genea3<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb3))&((genec
  3<=rangeob3)&(rangeob3<=gened3))&((genee3<=bearingob3)&(bearingob3<=genef
  3))
            392.heading=geneg3
            393.rule3=rule3+1
     394.end;
```

```
395.if
    ((genea4<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb4))&((genec
    4<=rangeob3)&(rangeob3<=gened4))&((genee4<=bearingob3)&(bearingob3<=genef
    4))
                396.heading=geneg4
                397.rule4=rule4+1
        398.end;
399.if
    ((genea5<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb5))&((genec
    5<=rangeob3)&(rangeob3<=gened5))&((genee5<=bearingob3)&(bearingob3<=genef
    5))
                400.heading=geneg5
                401.rule5=rule5+1
        402.end;
403.if
    ((genea6<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb6))&((genec
    6<=rangeob3)&(rangeob3<=gened6))&((genee6<=bearingob3)&(bearingob3<=genef
    6))
                404.heading=geneg6
                405.rule6=rule6+1
        406.end;
407.if
    ((genea7<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb7))&((genec
    7<=rangeob3)&(rangeob3<=gened7))&((genee7<=bearingob3)&(bearingob3<=genef
    7))
                408.heading=geneg7
                409.rule7=rule7+1
        410.end;
411.if
    ((genea8<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb8))&((genec
    8<=rangeob3)&(rangeob3<=gened8))&((genee8<=bearingob3)&(bearingob3<=genef
    8))
                412.heading=geneg8
                413.rule8=rule8+1
            414.end;
415.if
    ((genea9<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb9))&((genec
    9<=rangeob3)&(rangeob3<=gened9))&((genee9<=bearingob3)&(bearingob3<=genef
    9))
                416.heading=geneg9
                417.rule9=rule9+1
            418.end;
419.if
    ((genea10<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb10))&((gen
    ec10<=rangeob3)&(rangeob3<=gened10))&((genee10<=bearingob3)&(bearingob3<=
    genef10))
```

```
            420.heading=geneg10
            421.rule10=rule10+1
        422.end;
423.if
    ((genea11<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb11))&((gen
    ec11<=rangeob3)&(rangeob3<=gened11))&((genee11<=bearingob3)&(bearingob3<=
    genef11))
            424.heading=geneg11
            425.rule11=rule11+1
        426.end;
427.if
    ((genea12<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb12))&((gen
    ec12<=rangeob3)&(rangeob3<=gened12))&((genee12<=bearingob3)&(bearingob3<=
    genef12))
            428.heading=geneg12
            429.rule12=rule12+1
        430.end;
431.if
    ((genea13<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb13))&((gen
    ec13<=rangeob3)&(rangeob3<=gened13))&((genee13<=bearingob3)&(bearingob3<=
    genef13))
            432.heading=geneg13
            433.rule13=rule13+1
            434.end;
435.if
    ((genea14<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb14))&((gen
    ec14<=rangeob3)&(rangeob3<=gened14))&((genee14<=bearingob3)&(bearingob3<=
    genef14))
                436.heading=geneg14
                437.rule14=rule14+1
            438.end;
439.if
    ((genea15<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb15))&((gen
    ec15<=rangeob3)&(rangeob3<=gened15))&((genee15<=bearingob3)&(bearingob3<=
    genef15))
                440.heading=geneg15
                441.rule15=rule15+1
            442.end;
443.if
    ((genea16<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb16))&((gen
    ec16<=rangeob3)&(rangeob3<=gened16))&((genee16<=bearingob3)&(bearingob3<=
    genef16))
                444.heading=geneg16
                445.rule16=rule16+1
            446.end;
```

```
447.if
    ((genea17<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb17))&((gen
    ec17<=rangeob3)&(rangeob3<=gened17))&((genee17<=bearingob3)&(bearingob3<=
    genef17))
                448.heading=geneg17
                449.rule17=rule17+1
            450.end;
451.if
    ((genea18<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb18))&((gen
    ec18<=rangeob3)&(rangeob3<=gened18))&((genee18<=bearingob3)&(bearingob3<=
    genef18))
                452.heading=geneg18
                453.rule18=rule18+1
            454.end;
455.if
    ((genea19<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb19))&((gen
    ec19<=rangeob3)&(rangeob3<=gened19))&((genee19<=bearingob3)&(bearingob3<=
    genef19))
                456.heading=geneg19
                457.rule19=rule19+1
            458.end;
459.if
    ((genea20<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb20))&((gen
    ec20<=rangeob3)&(rangeob3<=gened20))&((genee20<=bearingob3)&(bearingob3<=
    genef20))
            460.heading=geneg20
            461.rule20=rule20+1
        462.end;
463.if
    ((genea21<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb21))&((gen
    ec21<=rangeob3)&(rangeob3<=gened21))&((genee21<=bearingob3)&(bearingob3<=
    genef21))
            464.heading=geneg21
            465.rule21=rule21+1
        466.end;
467.if
    ((genea22<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb22))&((gen
    ec22<=rangeob3)&(rangeob3<=gened22))&((genee22<=bearingob3)&(bearingob3<=
    genef22))
            468.heading=geneg22
            469.rule22=rule22+1
        470.end;
471.if
    ((genea23<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb23))&((gen
    ec23<=rangeob3)&(rangeob3<=gened23))&((genee23<=bearingob3)&(bearingob3<=
    genef23))
```

```
                heading=geneg23
                rule23=rule23+1
            end;
    if
        ((genea24<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb24))&((gen
        ec24<=rangeob3)&(rangeob3<=gened24))&((genee24<=bearingob3)&(bearingob3<=
        genef24))
                heading=geneg24
                rule24=rule24+1
            end;
    if
        ((genea25<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb25))&((gen
        ec25<=rangeob3)&(rangeob3<=gened25))&((genee25<=bearingob3)&(bearingob3<=
        genef25))
                heading=geneg25
                rule25=rule25+1
            end;
        end;
    % Determines if the robot has passed the fourth obstacle;
        if (60<=xrobot)&(xrobot<=95);
    % Determines which stimulus-response rule to apply;
    if
        ((genea1<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb1))&((genec
    1<=rangeob4)&(rangeob4<=gened1))&((genee1<=bearingob4)&(bearingob4<=genef
    1))
                488.heading=geneg1
                489.rule1=rule1+1
        490.end;
491.if
    ((genea2<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb2))&((genec
    2<=rangeob4)&(rangeob4<=gened2))&((genee2<=bearingob4)&(bearingob4<=genef
    2))
                492.heading=geneg2
                493.rule2=rule2+1
        494.end;
495.if
    ((genea3<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb3))&((genec
    3<=rangeob4)&(rangeob4<=gened3))&((genee3<=bearingob4)&(bearingob4<=genef
    3))
                496.heading=geneg3
                497.rule3=rule3+1
        498.end;
499.if
    ((genea4<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb4))&((genec
    4<=rangeob4)&(rangeob4<=gened4))&((genee4<=bearingob4)&(bearingob4<=genef
    4))
```

```
            500.heading=geneg4
            501.rule4=rule4+1
        502.end;
503.if
  ((genea5<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb5))&((genec
  5<=rangeob4)&(rangeob4<=gened5))&((genee5<=bearingob4)&(bearingob4<=genef
  5))
            504.heading=geneg5
            505.rule5=rule5+1
        506.end;
507.if
  ((genea6<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb6))&((genec
  6<=rangeob4)&(rangeob4<=gened6))&((genee6<=bearingob4)&(bearingob4<=genef
  6))
            508.heading=geneg6
            509.rule6=rule6+1
        510.end;
511.if
  ((genea7<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb7))&((genec
  7<=rangeob4)&(rangeob4<=gened7))&((genee7<=bearingob4)&(bearingob4<=genef
  7))
            512.heading=geneg7
            513.rule7=rule7+1
        514.end;
515.if
    ((genea8<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb8))&((genec
    8<=rangeob4)&(rangeob4<=gened8))&((genee8<=bearingob4)&(bearingob4<=genef
    8))
                516.heading=geneg8
                517.rule8=rule8+1
            518.end;
519.if
    ((genea9<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb9))&((genec
    9<=rangeob4)&(rangeob4<=gened9))&((genee9<=bearingob4)&(bearingob4<=genef
    9))
                520.heading=geneg9
                521.rule9=rule9+1
            522.end;
523.if
    ((genea10<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb10))&((gen
    ec10<=rangeob4)&(rangeob4<=gened10))&((genee10<=bearingob4)&(bearingob4<=
    genef10))
                524.heading=geneg10
                525.rule10=rule10+1
            526.end;
```

```
527.if
    ((genea11<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb11))&((gen
    ec11<=rangeob4)&(rangeob4<=gened11))&((genee11<=bearingob4)&(bearingob4<=
    genef11))
                528.heading=geneg11
                529.rule11=rule11+1
            530.end;
531.if
    ((genea12<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb12))&((gen
    ec12<=rangeob4)&(rangeob4<=gened12))&((genee12<=bearingob4)&(bearingob4<=
    genef12))
                532.heading=geneg12
                533.rule12=rule12+1
            534.end;
535.if
    ((genea13<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb13))&((gen
    ec13<=rangeob4)&(rangeob4<=gened13))&((genee13<=bearingob4)&(bearingob4<=
    genef13))
                536.heading=geneg13
                537.rule13=rule13+1
            538.end;
539.if
    ((genea14<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb14))&((gen
    ec14<=rangeob4)&(rangeob4<=gened14))&((genee14<=bearingob4)&(bearingob4<=
    genef14))
                540.heading=geneg14
                541.rule14=rule14+1
            542.end;
543.if
    ((genea15<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb15))&((gen
    ec15<=rangeob4)&(rangeob4<=gened15))&((genee15<=bearingob4)&(bearingob4<=
    genef15))
                544.heading=geneg15
                545.rule15=rule15+1
            546.end;
547.if
    ((genea16<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb16))&((gen
    ec16<=rangeob4)&(rangeob4<=gened16))&((genee16<=bearingob4)&(bearingob4<=
    genef16))
                548.heading=geneg16
                549.rule16=rule16+1
            550.end;
551.if
    ((genea17<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb17))&((gen
    ec17<=rangeob4)&(rangeob4<=gened17))&((genee17<=bearingob4)&(bearingob4<=
    genef17))
```

```
            552.heading=geneg17
            553.rule17=rule17+1
        554.end;
555.if
    ((genea18<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb18))&((gen
    ec18<=rangeob4)&(rangeob4<=gened18))&((genee18<=bearingob4)&(bearingob4<=
    genef18))
            556.heading=geneg18
            557.rule18=rule18+1
        558.end;
559.if
    ((genea19<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb19))&((gen
    ec19<=rangeob4)&(rangeob4<=gened19))&((genee19<=bearingob4)&(bearingob4<=
    genef19))
            560.heading=geneg19
            561.rule19=rule19+1
        562.end;
563.if
    ((genea20<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb20))&((gen
    ec20<=rangeob4)&(rangeob4<=gened20))&((genee20<=bearingob4)&(bearingob4<=
    genef20))
            564.heading=geneg20
            565.rule20=rule20+1
            566.end;
567.if
    ((genea21<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb21))&((gen
    ec21<=rangeob4)&(rangeob4<=gened21))&((genee21<=bearingob4)&(bearingob4<=
    genef21))
                568.heading=geneg21
                569.rule21=rule21+1
            570.end;
571.if
    ((genea22<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb22))&((gen
    ec22<=rangeob4)&(rangeob4<=gened22))&((genee22<=bearingob4)&(bearingob4<=
    genef22))
                572.heading=geneg22
                573.rule22=rule22+1
            574.end;
575.if
    ((genea23<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb23))&((gen
    ec23<=rangeob4)&(rangeob4<=gened23))&((genee23<=bearingob4)&(bearingob4<=
    genef23))
                576.heading=geneg23
                577.rule23=rule23+1
            578.end;
```

```
579.if
    ((genea24<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb24))&((gen
    ec24<=rangeob4)&(rangeob4<=gened24))&((genee24<=bearingob4)&(bearingob4<=
    genef24))
                580.heading=geneg24
                581.rule24=rule24+1
            582.end;
583.if
    ((genea25<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb25))&((gen
    ec25<=rangeob4)&(rangeob4<=gened25))&((genee25<=bearingob4)&(bearingob4<=
    genef25))
                584.heading=geneg25
                585.rule25=rule25+1
            586.end;
    587.end;
588.% Dummy statements in order to provide values for the dummy obstacle;
    589.if (95<xrobot)&(xrobot<=160);
590.% Determines which stimulus-response rule to apply;
591.if
    ((genea1<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb1))&((genec
    1<=rangeob5)&(rangeob5<=gened1))&((genee1<=bearingob5)&(bearingob5<=genef
    1))
                592.heading=geneg1
            593.rule1=rule1+1
        594.end;
595.if
    ((genea2<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb2))&((genec
    2<=rangeob5)&(rangeob5<=gened2))&((genee2<=bearingob5)&(bearingob5<=genef
    2))
            596.heading=geneg2
            597.rule2=rule2+1
        598.end;
599.if
    ((genea3<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb3))&((genec
    3<=rangeob5)&(rangeob5<=gened3))&((genee3<=bearingob5)&(bearingob5<=genef
    3))
            600.heading=geneg3
            601.rule3=rule3+1
        602.end;
603.if
    ((genea4<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb4))&((genec
    4<=rangeob5)&(rangeob5<=gened4))&((genee4<=bearingob5)&(bearingob5<=genef
    4))
            604.heading=geneg4
            605.rule4=rule4+1
        606.end;
```

```
607.if
   ((genea5<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb5))&((genec
   5<=rangeob5)&(rangeob5<=gened5))&((genee5<=bearingob5)&(bearingob5<=genef
   5))
            608.heading=geneg5
            609.rule5=rule5+1
      610.end;
611.if
   ((genea6<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb6))&((genec
   6<=rangeob5)&(rangeob5<=gened6))&((genee6<=bearingob5)&(bearingob5<=genef
   6))
            612.heading=geneg6
            613.rule6=rule6+1
      614.end;
615.if
   ((genea7<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb7))&((genec
   7<=rangeob5)&(rangeob5<=gened7))&((genee7<=bearingob5)&(bearingob5<=genef
   7))
            616.heading=geneg7
            617.rule7=rule7+1
      618.end;
619.if
   ((genea8<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb8))&((genec
    8<=rangeob5)&(rangeob5<=gened8))&((genee8<=bearingob5)&(bearingob5<=genef
    8))
                620.heading=geneg8
                621.rule8=rule8+1
            622.end;
623.if
    ((genea9<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb9))&((genec
    9<=rangeob5)&(rangeob5<=gened9))&((genee9<=bearingob5)&(bearingob5<=genef
    9))
                624.heading=geneg9
                625.rule9=rule9+1
            626.end;
627.if
    ((genea10<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb10))&((gen
    ec10<=rangeob5)&(rangeob5<=gened10))&((genee10<=bearingob5)&(bearingob5<=
    genef10))
                628.heading=geneg10
                629.rule10=rule10+1
            630.end;
631.if
    ((genea11<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb11))&((gen
    ec11<=rangeob5)&(rangeob5<=gened11))&((genee11<=bearingob5)&(bearingob5<=
    genef11))
```

```
            632.heading=geneg11
            633.rule11=rule11+1
        634.end;
635.if
    ((genea12<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb12))&((gen
    ec12<=rangeob5)&(rangeob5<=gened12))&((genee12<=bearingob5)&(bearingob5<=
    genef12))
            636.heading=geneg12
            637.rule12=rule12+1
        638.end;
639.if
    ((genea13<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb13))&((gen
    ec13<=rangeob5)&(rangeob5<=gened13))&((genee13<=bearingob5)&(bearingob5<=
    genef13))
            640.heading=geneg13
            641.rule13=rule13+1
        642.end;
643.if
    ((genea14<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb14))&((gen
    ec14<=rangeob5)&(rangeob5<=gened14))&((genee14<=bearingob5)&(bearingob5<=
    genef14))
            644.heading=geneg14
            645.rule14=rule14+1
            646.end;
647.if
    ((genea15<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb15))&((gen
    ec15<=rangeob5)&(rangeob5<=gened15))&((genee15<=bearingob5)&(bearingob5<=
    genef15))
                648.heading=geneg15
                649.rule15=rule15+1
            650.end;
651.if
    ((genea16<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb16))&((gen
    ec16<=rangeob5)&(rangeob5<=gened16))&((genee16<=bearingob5)&(bearingob5<=
    genef16))
                652.heading=geneg16
                653.rule16=rule16+1
            654.end;
655.if
    ((genea17<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb17))&((gen
    ec17<=rangeob5)&(rangeob5<=gened17))&((genee17<=bearingob5)&(bearingob5<=
    genef17))
                656.heading=geneg17
                657.rule17=rule17+1
            658.end;
```

```
659.if
    ((genea18<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb18))&((gen
    ec18<=rangeob5)&(rangeob5<=gened18))&((genee18<=bearingob5)&(bearingob5<=
    genef18))
                660.heading=geneg18
                661.rule18=rule18+1
            662.end;
663.if
    ((genea19<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb19))&((gen
    ec19<=rangeob5)&(rangeob5<=gened19))&((genee19<=bearingob5)&(bearingob5<=
    genef19))
                664.heading=geneg19
                665.rule19=rule19+1
            666.end;
667.if
    ((genea20<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb20))&((gen
    ec20<=rangeob5)&(rangeob5<=gened20))&((genee20<=bearingob5)&(bearingob5<=
    genef20))
                668.heading=geneg20
                669.rule20=rule20+1
            670.end;
671.if
    ((genea21<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb21))&((gen
    ec21<=rangeob5)&(rangeob5<=gened21))&((genee21<=bearingob5)&(bearingob5<=
    genef21))
                672.heading=geneg21
                673.rule21=rule21+1
            674.end;
675.if
    ((genea22<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb22))&((gen
    ec22<=rangeob5)&(rangeob5<=gened22))&((genee22<=bearingob5)&(bearingob5<=
    genef22))
                676.heading=geneg22
                677.rule22=rule22+1
            678.end;
679.if
    ((genea23<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb23))&((gen
    ec23<=rangeob5)&(rangeob5<=gened23))&((genee23<=bearingob5)&(bearingob5<=
    genef23))
                680.heading=geneg23
                681.rule23=rule23+1
            682.end;
683.if
    ((genea24<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb24))&((gen
    ec24<=rangeob5)&(rangeob5<=gened24))&((genee24<=bearingob5)&(bearingob5<=
    genef24))
```

```
            heading=geneg24
            rule24=rule24+1
        end;
if
    ((genea25<=robotrangetoupperbound)&(robotrangetoupperbound<=geneb25))&((gen
    ec25<=rangeob5)&(rangeob5<=gened25))&((genee25<=bearingob5)&(bearingob5<=
    genef25))
            heading=geneg25
            rule25=rule25+1
        end;
    end;
% Heading is equal to the initial heading plus the new heading;
    headingnew=headinginitial+heading;
% The initial heading is then incremented to the new heading;
    headinginitial=headingnew;
% Robot moves 1 unit distance in the heading provided by the stimulus-response
    rules;
    xrobotnew=xrobot+cos(headingnew);
    yrobotnew=yrobot+sin(headingnew);
% Plots new robot position and heading;
    robotcenter=xrobotnew;
    robotcenterleft=robotcenter-1;
    robotcenterright=robotcenter+1;
```

```
[robotcenterleft:.1:robotcenterright];
theta7=atan2((yrobotnew+sqrt(1-(x-xrobotnew).^2)),x);
theta8=atan2((yrobotnew-sqrt(1-(x-xrobotnew).^2)),x);
rho7=sqrt(x.^2+(yrobotnew+sqrt(1-(x-xrobotnew).^2)).^2);
rho8=sqrt(x.^2+(yrobotnew-sqrt(1-(x-xrobotnew).^2)).^2);
[x,y11]=pol2cart(theta7,rho7);
[x,y12]=pol2cart(theta8,rho8);
plot(x,y11,'b',x,y12,'b');
xorient=[xrobotnew xrobotnew+cos(headingnew)];
    yorient=[yrobotnew yrobotnew+sin(headingnew)];
    thetaorientation=atan(sin(headingnew)/cos(headingnew));
plot(xorient,yorient,'g');
rule26=terminationcounter-rule1-rule2-rule3-rule4-rule5-rule6-rule7-rule8-rule9-rule10-rule11-rule12-rule13-rule14-rule15-rule16-rule17-rule18-rule19-rule20-rule21-rule22-rule23-rule24-rule25;
xrobot
    yrobot
    robotrangetoupperbound
    rangeob1
    bearingob1
    heading
    thetaorientation
    % increments the obstacle bearing to the initial obstacle bearing;
```

```
        initialbearingob1=bearingob1;
        initialbearingob2=bearingob2;
        initialbearingob3=bearingob3;
        initialbearingob4=bearingob4;
        initialbearingob5=bearingob5;
end;
xrobot
terminationcounter
rule1
rule2
rule3
rule4
rule5
rule6
rule7
rule8
rule9
rule10
rule11
rule12
rule13
rule14
rule15
```

747.rule16

748.rule17

749.rule18

750.rule19

751.rule20

752.rule21

753.rule22

754.rule23

755.rule24

756.rule25

757.rule26

Key Words Index