2012

Distributed Biogeography Based Optimization for Mobile Robots

Arpit Shah
Cleveland State University

DISTRIBUTED BIOGEOGRAPHY BASED OPTIMIZATION FOR MOBILE ROBOTS

Arpit Shah

Bachelor of Science in Electronics and Communication Engineering

HNGU North Gujarat University

May, 2009

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

May, 2012

DISTRIBUTED BIOGROGRAPHY BASED OPTIMIZATION FOR MOBILE ROBOTS

ARPIT SHAH

ABSTRACT

I present hardware testing of an evolutionary algorithm (EA) known as distributed biogeography based optimization (DBBO). DBBO is an extended version of biogeography based optimization (BBO). Typically, EAs require a central computer to control the evaluation of candidate solutions to some optimization problem, and to control the sharing of information between those candidate solutions. DBBO, however, does not require a centralized unit to control individuals. Individuals independently run the EA and find a solution to a given optimization problem. Both BBO and DBBO are based on the theory of biogeography, which describes how organisms are distributed geographically in nature. I have compared the performance of BBO and DBBO by using fourteen benchmark functions that are commonly used to evaluate the performance of optimization algorithms. I perform both hardware and simulation experiments. Wall-following robots are used as hardware to implement the DBBO algorithm. Robots use two different controllers to maintain a constant distance from the wall: one is a proportional integral derivative (PID) controller and the other is a fuzzy controller. DBBO optimizes the performance of the robots with respect to the control parameters. During simulation experiments I used different EA mutation rates; different staring points for the robots; and different wheel bases. I have also done T-tests to analyze the statistical significance of performance differences and robustness tests to analyze the performance of the algorithms in the face of environmental changes. The results show that centralized BBO gives better optimization results than distributed BBO. DBBO gives less optimal solutions but it removes the necessity of centralized control. The results also show that the fuzzy controller performs better than the PID controller.

This thesis has been approved for the

Department of **ELECTRICAL AND COMPUTER ENGINEERING**

and the College of Graduate Studies by

Thesis Committee Chairperson, Dr. Dan Simon

Department/Date

Dr. Lili Dong

Department/Date

Dr. Chansu Yu

Department/Date

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

$\boldsymbol{\mu}$	emigration rate
$\boldsymbol{\lambda}$	immigration rate
$\boldsymbol{P}$	population size
$\boldsymbol{f(P_i)}$	fitness of the *i*-th individual
$\boldsymbol{W}$	worst fitness value
$\boldsymbol{B}$	best fitness value
$\boldsymbol{\delta(A)}$	membership function of fuzzy set *A*
$\boldsymbol{d_1}$	distance from the wall measured by sensor 1
$\boldsymbol{d_2}$	distance from the wall measured by sensor 2
$\boldsymbol{d_{\mathrm{b}}}$	difference between d_1 and d_2
$\boldsymbol{y_{\mathrm{ref}}}$	constant distance robot has to maintain from wall
$\boldsymbol{\theta}$	angle
$\boldsymbol{e(t)}$	tracking error
$\boldsymbol{e_{current}}$	current error
$\boldsymbol{e_{old}}$	previous error
$\boldsymbol{k_1}$	constant

$\boldsymbol{k_2}$	constant
Cost	cost value calculated by robots
$\boldsymbol{r}$	rise time
$\boldsymbol{k_p}$	proportional constant
$\boldsymbol{k_i}$	integral constant
$\boldsymbol{k_d}$	derivative constant
$\boldsymbol{\alpha}$	membership function of error input
$\boldsymbol{\beta}$	membership function of delta error input
$\boldsymbol{\gamma}$	membership function of motor voltage output
$\boldsymbol{N}$	number of candidates
$\boldsymbol{V_{left}}$	pulse width modulation cycles for left wheel
$\boldsymbol{V_{right}}$	pulse width modulation cycles for right wheel
$\boldsymbol{V_{ref}}$	reference pulse width modulation cycles
$\boldsymbol{U}$	uniform distribution of random numbers

ACRONYMS

BBO	biogeography based optimization
bps	bits per second
DBBO	distributed biogeography based optimization
dBM	decibels in milliwatts
DC	direct current
EA	evolutionary algorithm
EC	evolutionary computing
EP	evolutionary programming
ES	evolutionary strategy
GA	genetic algorithm
GP	genetic programming
HSI	habitat suitability index
IR	infrared
LCD	liquid crystal display
MF	membership function
PCB	printed circuit board

PID	proportional integral derivative
PWM	pulse width modulation
SIV	suitability index variable

CHAPTER I

INTRODUCTION

1.1 Introduction to evolutionary algorithms

One of the main functions of engineers is to try to find different methods to optimize or solve very complex and difficult problems. The evolutionary algorithm (EA) is one of the most attractive methods to solve different problems, such as generating adaptive genetic operators [14], evolutionary path planning for autonomous under water vehicles [1], and smart highway systems to solve traffic problems [2]. The main advantage of EAs over analytical optimization algorithms is their flexibility and adaptability.

The EA is a robust search and optimization method. The idea of EAs originated in the 1950s based on research by Bremermann [4]; Friedberg [10], [9]; Box [3]; and others. At that time EAs were not used widely by scientists due to a lack of powerful computers. After a few decades, as computers became more available, more research was done in this field. EA functionality is inspired by biological evolution, such as reproduction, mutation, recombination, and selection, as shown in Figure 1 [8].

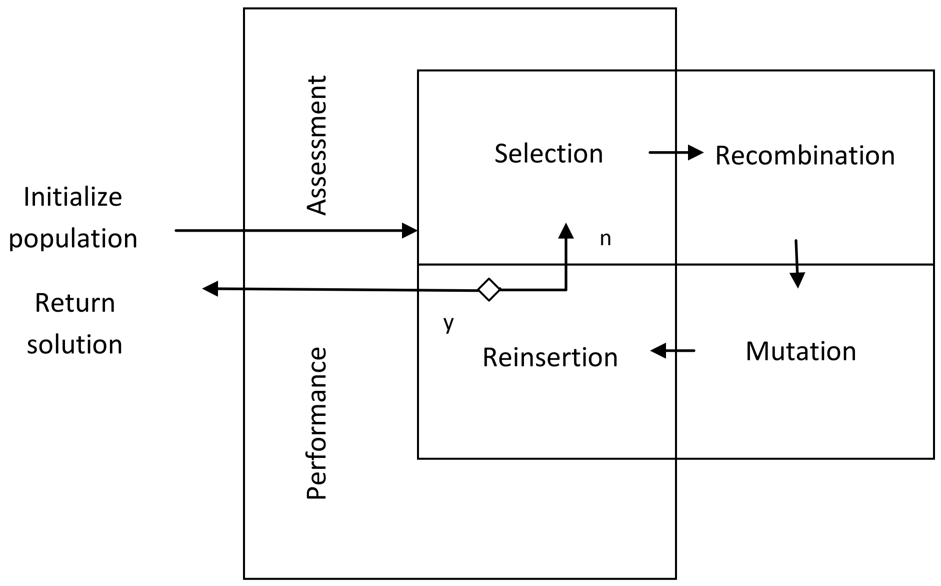


Figure 1: Block diagram of EA

Figure 1 shows the basic block diagram of EAs. The first step is to initialize the population. Each individual in the population has a candidate solution to the problem. Then the central unit assesses these solutions by assigning fitness values to these solutions according to their effectiveness in solving the problem. A good solution has high fitness and a bad solution has low fitness. EAs will sort these individuals according to their fitness level. Evolution is done by various operators, like recombination and mutation, as I mentioned previously. Recombination is also known as crossover. Crossover exchanges the genes of the parent individuals to generate a new child. Mutation randomly selects new solutions within a given search space and generates a new child. At the end of each generation, the central unit sorts all these new solutions by their fitness values, and after several generations the EA evolves a better solution. EAs are a subset of evolutionary computing (EC) and has mainly four approaches: evolution strategies (ES), evolutionary programming (EP), genetic algorithm (GA) and genetic programming (GP). I discuss these four approaches in the following paragraphs.

The ES was proposed by three students who were working on the experiment of minimizing drag in wind tunnel [30], [40]. Their algorithm, also called "cybernetic solution path," has two characteristics. The first is mutation, in which the system randomly changes the variables in each generation, and the second is selection, in which, if the new solution from these variable changes is not good, then the system will keep the old variables, and otherwise it will keep the new ones.

Fogel introduced EP in his book *Artificial Intelligence Through Simulated Evolution* in 1962 [12], [11]. According to Fogel, artificial intelligence is the combined ability to predict the environment of any system, and to predict the response of that system for the predicted environment with respect to any specific independent variable. EP mutation is different from ES. During mutation EP does not change its variables, but instead regenerates new variables and selects the candidate according to fitness level.

The third EA approach is the GA. In 1975 Holland published an article called *Adaption in Natural and Artificial Systems* [16]. The algorithm introduced by him was very helpful in the development of GAs. GA behavior is governed by a fundamental theorem called the schema theorem. In GA the candidate is referred as a chromosome, which is analogous to DNA. According to the schema theorem, the first step is single point crossover of individual chromosomes which are in the same class, and sorting these chromosomes by fitness. After several generations fitness will increase exponentially.

The last EA approach is GP. As I discussed, the previous EA approaches were used to get the solution of real world problems. GP is an approach to improve computer programs. GP was first introduced by Cramer in 1985 [5]. He generated an algorithm which develops simple sequential programs. He used GP to generate different functions which are well defined and have two inputs and one output. Another application of GP is in circuit design. After we set the requirements and list of components, GP generates the size, placement of component and routing of the circuit [18].

In this thesis I am going to explain a new EA which is inspired from the science of biogeography and which is called biogeography based optimization (BBO).

1.2 Biogeography based optimization

The term biogeography is the study of the geographical distribution of plants and animal life. It was first introduced by Charles Darwin in the nineteenth century [6]. In the 1960s MacArthur and Edward Wilson created biogeography models from their studies of island biogeography [24]. Their main focus was to study the distribution of species among islands. They introduced mathematical models of the migration of species. These models explain how species migrate from one island to other, how new species come to islands, and how they become extinct. Here the word 'island' refers to a habitat, which is a geographically isolated region.

BBO was developed as a mathematical model of biogeography to optimize solutions for different problems. It has been applied to satellite image classification [27], aircraft engine sensor selection [33], antenna design [39], optimization of different power system problems [28], [29], groundwater detection [17], mechanical gear train design [35] and neuro-fuzzy system training for biomedical applications [25]. Recent research in the area of BBO has focused on putting it on a firm theoretical and mathematical foundation, including the derivation of dynamic system models [38] and Markov models [36, 37] that describe its behavior.

1.3 Problem statement

In this thesis, I am going to explain BBO and extend it to distributed learning. It also gives experimental and simulation results. BBO and most other EAs use a central computer to gather data from each individual for the algorithm. But in the real world there are a few systems (for example peer to peer networking) in which it becomes very hard to have a central computer which communicates with all the individuals. So we have developed distributed BBO which does not require a central computer. DBBO's control and communication is distributed between different BBO individuals rather than coordinated by a central computer.

Distributed BBO is based on distributed learning. It is a theory which developed to explain how the human mind learns [44]. The research says that the human mental capability is not only centralized inside the mind, but outside social interaction also has an effect on human mental capability [7]. Distributed learning explains how environmental influences prompt humans to solve problems. For example, a human child who wants to learn how to walk does not learn only by himself. He looks around him and sees how other humans walk and learns better. In general, humans as a team perform tasks more accurately than alone. For example, companies dealing with big projects usually choose a team of employees to work together rather than a single employee, so that each employee communicates, shares their ideas, and achieves the goal quickly and more accurately. I applied this distributed learning to a swarm of robots. Like humans, these robots also perform tasks, learn, and solve problems by communicating with each other.

This thesis also discusses fuzzy logic control (FLC) which is based on fuzzy logic. FLC has become a very interesting area for research. There are many applications of FLC, like elevator control [8], fuzzy memory devices [19], and water quality control [43],[45], which encourages the development of FLC. In this paper, I am going to compare the performance of FLC with traditional proportional-integral-derivative (PID) controller for mobile robot control.

I used these two controllers in wall-following robots. The main objective of the robots is to maintain a constant distance from the wall. I am choosing two parameters for the PID controller, which is the proportional term and the derivative term, and fifteen (15) parameters for the fuzzy controller, to control a wall-following robot. I applied DBBO to these controller parameters to tune these parameters to get the best performance from the robots.

1.4 Contribution of thesis

As I discussed in the previous section, this thesis explains BBO and distributed BBO. The main contribution of this paper is the physical and simulation experiments for the DBBO algorithm. I have used four wall-following robots for the physical experiments and applied DBBO to improve their control performance. I have used two different controllers for the robots, and tuned them using DBBO and compared their performance. I have also done several simulations with different mutation rates, different starting points, and different wheel bases to tune the controllers using BBO and DBBO in order to compare their results. Moreover, I ran T-tests and robustness tests and discuss the results.

1.5 Structure of thesis

Chapter II gives an overview of the BBO algorithm and explains BBO terminology. The first section explains the traditional centralized BBO in which a central unit gathers all the data from each individual component and applies BBO. The second section explains distributed BBO, which describes how each individual component independently applies BBO instead of through a central unit. Finally, the third section explains the definition of benchmark functions and the results of the BBO and DBBO applied to fourteen benchmark functions.

Chapter III discusses fuzzy logic. The first section explains the definition of fuzzy logic. As the complexity of the system increases, it becomes very hard to make decisions regarding system behavior. So, Dr. Zadeh introduced a decision-making scheme known as the fuzzy logic controller (FLC) [47]. The last section explains two types of FLCs: the Mamdani model and the Takagi and Sugeno model.

Chapter IV of this thesis explains the hardware and control systems used in the robots. The first section gives a brief introduction of the parts used by the wall-following robots and the reasons for choosing those components. The four wall-following robots' main objective is to maintain a constant distance from the wall. The second section explains the robots' sensing and implementation of the control. I used two control systems: the PID controller and the fuzzy controller. The third section explains the first controller, which is the PID controller. It uses the proportional and derivative term to reduce error. The fourth section explains the fuzzy controller and its membership functions. It uses five membership functions for each input and output. The controller uses fuzzy if-then rules for decision making. The last section explains the software of the robot. It describes the flowchart of the whole cycle of the experiment.

Chapter V contains the results. I have done several experiments including T-tests and robustness tests. It explains the simulation experiments and their results for both the PID and fuzzy controllers, and also explains hardware experiment results for both controllers. It compares the results of BBO and DBBO with a different number of peers. It also compares the two controllers' performances.

Finally Chapter VI concludes the thesis and also suggests future work.

CHAPTER II

BIOGEOGRAPHY-BASED OPTIMIZATION

Chapter II gives a brief introduction about the evolutionary algorithm called bio-geography-based optimization (BBO). As I discuss in Chapter I, BBO is based on biogeography. This chapter explains the different terms of biogeography. The first section explains the algorithm of centralized BBO. Then I introduce distributed BBO, and explain the differences between BBO and DBBO. Finally, the third section explains the benchmark functions and compares the results of the BBO and the DBBO algorithms.

The BBO algorithm is created to optimize the solution of problems on the basis of the theory of biogeography. According to island biogeography, in nature species migrate from one island to another and create better habitats. Different islands have different environmental factors. These environmental factors are called suitability index variables (SIVs). A habitat that is more suitable for species has a high habitat suitability index

(HSI). Migration depends on the habitat's HSI. High-HSI habitats have more species, so their emigration rates will increase and low-HSI habitats have fewer species, so their immigration rates will increase.

BBO is the application of this concept to the engineering field. In an engineering field, an island is replaced with a candidate solution or an individual; HSI indicates the fitness or cost of individuals, and SIV indicates a solution feature or independent variable. Thus, an individual that has a good solution has a high fitness and a high emigration probability. An individual that has a low fitness has a high immigration probability, so it is more likely to accept a solution feature from surrounding individuals to try to better optimize the problem [33].

There are mainly three processes in BBO: migration, mutation, and elitism. These processes allow candidates to share information and save the best solution for the next generation. Migration is the most important process. Migration allows the candidates to emigrate or immigrate data from other candidates at each generation. Mutation is the same process used in other evolutionary algorithms. Each generation, each candidate randomly generates new solution features. Elitism finds the best solution at each generation, and replaces the worst solutions each generation with the best ones from the previous generation.

2.1 Centralized BBO

Centralized BBO is the original BBO algorithm. According to centralized BBO, each BBO individual sends information to a central unit. A central unit gets information, performs the BBO algorithm, sends back new solution features and creates better solutions. As explained above, migration depends on a habitat's HSI. Here, habitat is analogous to a solution and HSI is analogous to the fitness of the solution. The fitness of the solution determines the emigration rate (μ) and immigration rate (λ) as follows.

$$\mu_i = \frac{f(P_i) - min_k f(P_k)}{max_k f(P_k) - min_k f(P_k)} \tag{1}$$

$$\lambda_i = 1 - \mu_i \tag{2}$$

where $f(P_i)$ is the fitness of the *i*-th individual, and P represents the population size. As shown in Figure 2, the candidate with the worst solution has the lowest fitness, so it has both high immigration probability and low emigration probability. The candidate with the best solution has a high fitness level, so it has a high emigration probability and a low immigration probability.

The migration rates are scaled between 0 and 1. Thus, for the best solution we set the immigration rate λ_i = 0 and emigration rate μ_i = 1, and for the worst solution emigration rate μ_i = 0 and immigration rate λ_i = 1. If all the individuals have the same fitness then according to Equation 1 the denominator becomes the zero and migration rates become infinite. So for this case we set both immigration rate and emigration rate equal to 0.5.

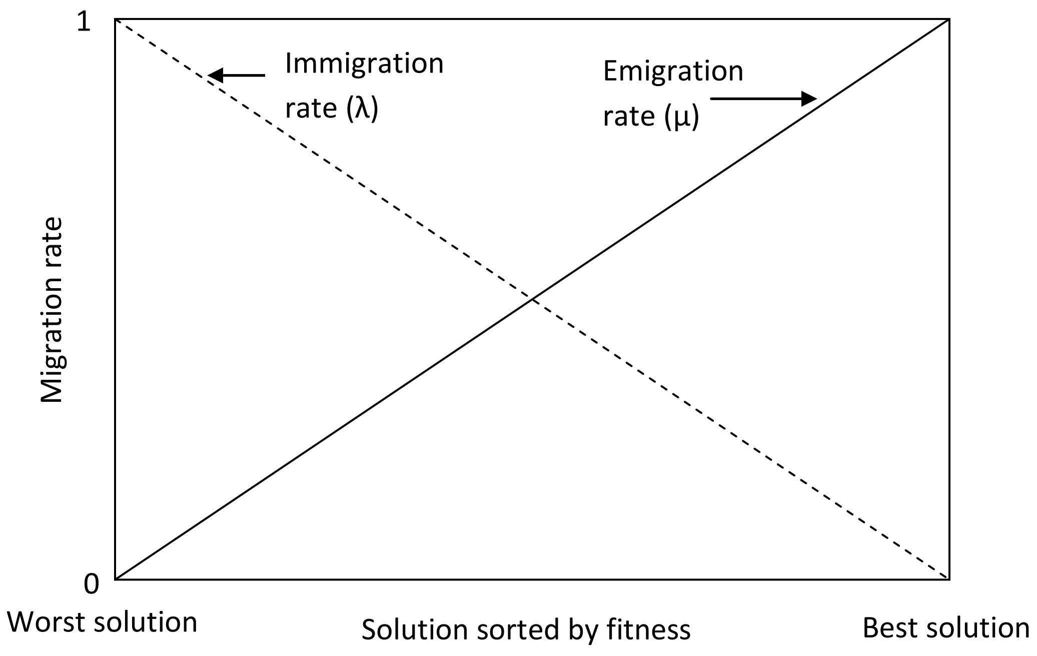


Figure 2: Migration rates as a function of BBO solution fitness

The immigration of the solution feature of the individual *x* to the individual *y* is probabilistically selected from the rest of the population based on their emigration rates as follows.

$$\text{Probability of emigration from } y = \frac{\mu(y)}{\sum_{i=1}^{N} \mu(x)} \text{ for all } y \qquad (3)$$

where *N* is the total number of candidates in the population.

After calculating the migration rates for the individual, BBO will perform mutation. According to the mutation probability BBO randomly generates new parameters from the search space. The population size and the mutation probability are user defined variables that depend on the problem. The user also has to define the number of generations in the BBO, or some other termination criteria for the optimization process. The last step is elitism in which the central unit will keep the best solutions from the previous generation, and use them to replace the worst solutions of the current generation. The BBO algorithm is shown in Figure 3.

Figure 3: Basic description of the BBO algorithm for one generation.

```
For each candidate problem solution $P_i$
    Calculate immigration probability $\lambda_i$ and emigration probability $\mu_i$ (see Figure 2)
        $\mu_i \in [0, 1]$ is proportional to the fitness of $P_i$, and $\lambda_i = 1 - \mu_i$
Next candidate solution: $i \leftarrow i+1$
For each candidate problem solution $P_i$
    For each solution variable $v$ in $P_i$
        Use immigration probability $\lambda_i$ to decide whether to immigrate to $P_i$
        If immigrating to $P_i$
            Use Equation 3 to select $P_k$ for emigration
            $P_k$ emigrates data to $P_i$ : $P_i(v) \leftarrow P_k(v)$
        End immigration
    Next solution variable
    Mutate $P_i$ probabilistically based on mutation probability
Next candidate solution: $i \leftarrow i+1$
```

2.2 Distributed BBO

Distributed BBO is an extension of the BBO algorithm. BBO uses a central control unit which collects data and applies the BBO algorithm. In contrast, DBBO does not use a central unit. Each individual in the system will independently apply the BBO algorithm. So the main advantage of this algorithm is that it does not require a central unit. Individuals (in our experiment, mobile robots) randomly select other individuals and start communication as shown in Figure 4.

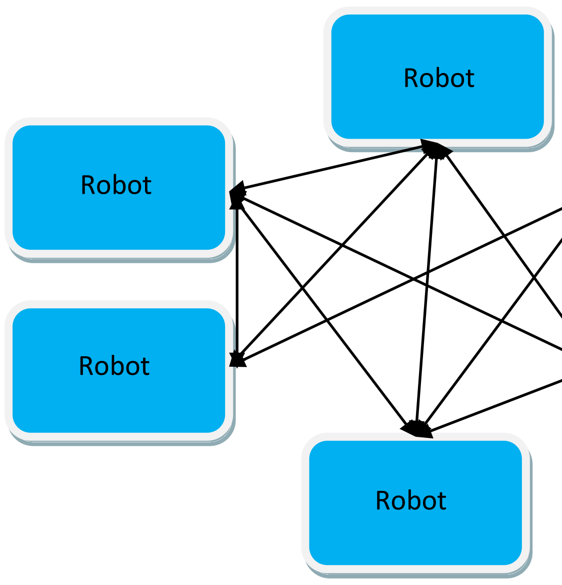


Figure 4: Random robot communication

In this thesis I use mobile robots to illustrate the DBBO algorithm. DBBO has similar characteristics to BBO. However, DBBO does not have elitism. In DBBO each individual randomly communicates with other ones. It is like peer to peer communication. Therefore an individual does not know the best solution of the entire population.

Figure 5: Basic description of the DBBO algorithm for one generation [32].

```
Select $m$ peers $\{P_i\}$ for communication with each other
Revise each peer's best and worst fitness estimates. For each $i$,
    $\text{MinEst}_i = \min_{k \in I} \{\text{MinEst}_k\}$ and $\text{MaxEst}_i = \max_{k \in I} \{\text{MaxEst}_k\}$, where
    $I$ is the set of all peers of robot $i$
Calculate each peer's likelihood to immigrate, $\lambda$, and emigrate, $\mu$:
      $\mu_i \in [0, 1]$ is proportional to the fitness of $P_i$ relative to its peers, and $\lambda_i = 1 - \mu_i$
For each peer $P_i$
    For each solution variable $v$
        Use immigration probability $\lambda_i$ to decide whether to immigrate to $P_i$
        If immigrating to $P_i$
            Use Equation 3 to select $P_k$ for emigration, where $N$ is replaced with $m$
            $P_k$ emigrates data to $P_i$: $P_i(v) \leftarrow P_k(v)$
        End immigration
    Next solution variable
    Mutate $P_i$ according to mutation probability
Next peer
```

The DBBO algorithm is shown in Figure 5. It seems the same as the BBO algorithm but there are some changes in the DBBO algorithm. For example in BBO the migration probability is given by whole population as indicated in Equation 1 but in DBBO it is determined by the members in the particular communicating group because an individual does not have the minimum and maximum fitness values of the population. So it uses the fitness values of those with whom it has already communicated to estimate the minimum and maximum fitness values of the entire population. Suppose we have a population indicated by the set *P* and the communicating group of individuals at a given time is denoted as *C*. Then *C* is the subset of *P*. The *j*-th individual's estimated best and worst fitness values of the entire population are denoted by B_j and W_j (denoted as MinEst_j and MaxEst_j in Figure 5). Now the *j*-th individual communicates with other individuals in group *C* and updates its estimations as follows.

$$W_j = \min\left[W_j, \min_i f(C_i)\right] \tag{4}$$

$$B_j = \max\, [B_j, \max_i f(C_i)] \tag{5}$$

where the minimization and maximization are taken over the *j*-th individual's peer group. Now the migration probabilities of the *j*-th individual are calculated as follows:

$$\mu_j = \frac{f(C_j) - W_j}{B_j - W_j} \tag{6}$$

$$\lambda_j = 1 - \mu_j \tag{7}$$

If the values of B_j and W_j are equal then the immigration and emigration rates are set equal to 0.5. After calculating the migration rates and performing migration, the DBBO individuals perform mutation, which is the same as in centralized BBO. In the next generation individuals form different groups and communicate with different individuals of the population.

2.3 Benchmark functions

In the field of evolutionary computation, lots of algorithms exist. Now to compare the performance of these algorithms we have to apply these algorithms on some predefined problems. These problems are called benchmark functions. BBO was used for optimization for the first time in 2008. Dr. Simon applied BBO as well as other algorithms to fourteen benchmark functions to compare their results. The results show that BBO outperformed the other EAs on ten benchmark functions out of fourteen [33]. The details of these benchmark functions are given in Appendix C.

Recall that Chapter II explained the two different versions of BBO. One is centralized and the other is distributed. It described how an individual calculates the migration rates, and performs mutation and elitism. The basic difference between these two is that in centralized BBO the central unit implements the BBO algorithm, while in distributed BBO each individual implements the BBO algorithm. The other difference is that DBBO does not implement elitism.

In this paper, I am going to compare the performance of the DBBO algorithm having a different number of peers (2, 4, and 6) with the BBO algorithm. I have used the same fourteen benchmark functions. To analyze the performance of the algorithms, I set the population size to 50, the function evolution limit to 500, the number of independent variables (problem dimension) to 10, and the mutation probability to 1%, and ran 100 Monte Carlo simulations. Table 1 show the average cost values after 100 Monte Carlo runs and the standard deviation.

Table 1: Average cost values of benchmark functions and standard deviation.

Benchmark Functions	BBO	DBBO/2	DBBO/4	DBBO/6
Ackley	**12.62±1.5**	13.33±2.66	12.95±2.3	13.20±2.15
Fletcher	**3.5E+4±1.E4**	1.2E+5±4.3E4	1.2E+5±4.1E4	1.2E+5±5.1E4
Griewank	**16.29±5.81**	57.65±22.49	56.18±21.03	59.32±18.86
Penalty #1	**3.7E+5±4.8E+5**	7.3E+6±7.5E+6	9.4E+6±8.1E+6	9.4E+6±10.7E+6
Penalty #2	**2.1E+6±2.3E+6**	2.5E+7±2.1E+7	2.9E+7±2.0E+7	3.5E+7±2.5E+7
Quartic	**0.24±0.15**	1.32±0.96	1.71±1.08	1.79±1.04
Rastrigin	**30.25±7.16**	74.88±14.01	73.28±13.51	71.18±13.56
Rosenbrook	**119.90±43.50**	4.0E+2±1.9E+2	4.2E+2±1.8E+2	4.4E+2±1.9E+2
Schwefel 1.2	**2.2E+3±7.7E2**	4.0E+3±1.3E+3	4.8E+3±1.4E+3	4.7E+3±1.1E+3
Schwefel 2.21	38.06±6.39	**29.16±9.78**	36.49±8.57	37.11±9.28
Schwefel 2.22	**6.30±1.81**	44.81±52.34	88.49±2.56E2	130.14±5.1E2
Schwefel 2.26	**9.2E+2±2.2E+2**	1.8E+3±3.5E+2	1.8E+3±3.3E+2	1.8E+3±3.0E+2
Sphere	**4.68±1.84**	16.57±6.63	16.22±5.45	17.06±6.27
Step	**1.8E+3±7.0E+2**	6.2E+3±2.4E3	6.4E+3±1.8E+3	6.0E+3±2.2E+3

According to Table 1 BBO outperformed DBBO in thirteen out of fourteen benchmark functions. But DBBO performed good in one benchmark function. So, in general, BBO is better than DBBO, but DBBO does not require a central unit.

CHAPTER III

FUZZY LOGIC

Chapter III introduces fuzzy logic. It is different from traditional logic. The first section explains the concept of fuzzy logic and gives definitions. The second section explains the use of fuzzy logic in decision making schemes, and the basic structure of FLCs. The third section summarizes two FLC models, which are Mamdani and TSK, and gives a comparison of them.

3.1 Concept of fuzzy logic

Robots use a fuzzy controller to maintain a constant distance from the wall. The fuzzy controller is based on fuzzy logic. Fuzzy logic is different from crisp logic. Sets in crisp logic are fixed and exact. In contrast, sets in fuzzy logic are approximate as shown in Figure 6. For example, set membership in binary logic contains only two values, either logic 1 or logic 0 (true or false), but set membership in fuzzy logic contains an infinite number of values between 0 and 1.

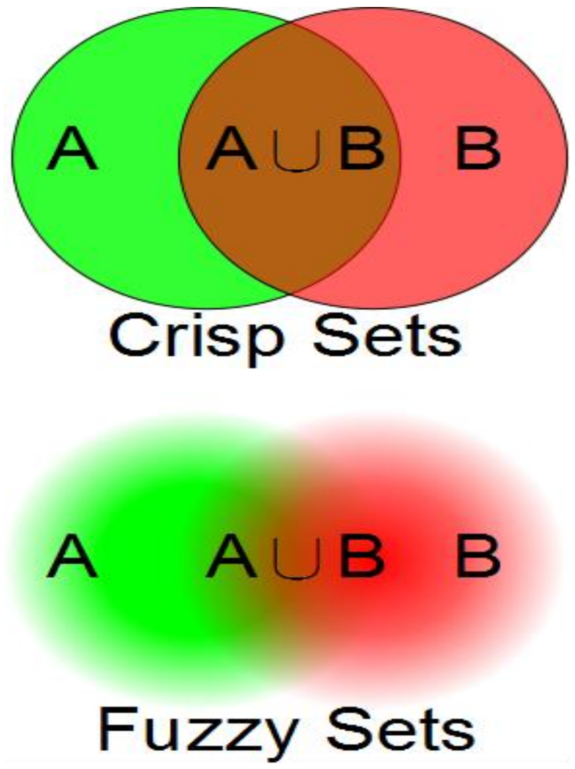


Figure 6: Representation of crisp sets and fuzzy sets

Fuzzy logic describes partial truth which ranges between complete truth and complete falsehood. So, in general, fuzzy set A in the universe U having a membership function δ_A which take values in the interval [0,1] is defined as follows:

$$A = \{(x, \delta_A(x)) | x \in U\} \quad (8)$$

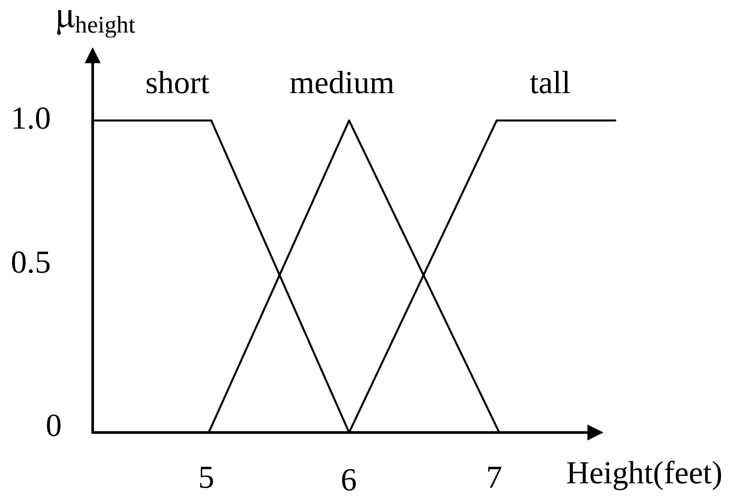


Figure 7: Graphical representation of fuzzy height

To explain fuzzy sets, I will use the example of men of differing heights and how those men can be classified. As shown in Figure 7, the heights of the men can be classified in three groups: short, medium, and tall. In this example, height is referred to as a linguistic variable, the universe have the values between the range [3 8] in units of feet, and the three groups are described by membership functions. So, as shown in Figure 7, men having height below 5 feet are 100% in the ‘short’ group, men having height between 5 feet to 6 feet are in both the short and the ‘medium’ group with different levels of membership, men between 6 feet and 7 feet are in both the ‘medium and the ‘tall’ group with different levels of membership, and men having height above 7 feet are 100% in the ‘tall’ group.

3.2 Fuzzy logic controller

Zadeh introduced a fuzzy decision making scheme on the basis of fuzzy logic [47]. He mapped input and output variables by fuzzy adjectives to indicate values like ‘high,’ ‘hot,’ and ‘small’. These adjectives are associated with membership functions. He uses these adjectives to relate different variables and make rules—for example, “If the temperature is high, turn on the compressor of the air conditioner.” The basic block diagram of the fuzzy logic controller is shown in Figure 8.

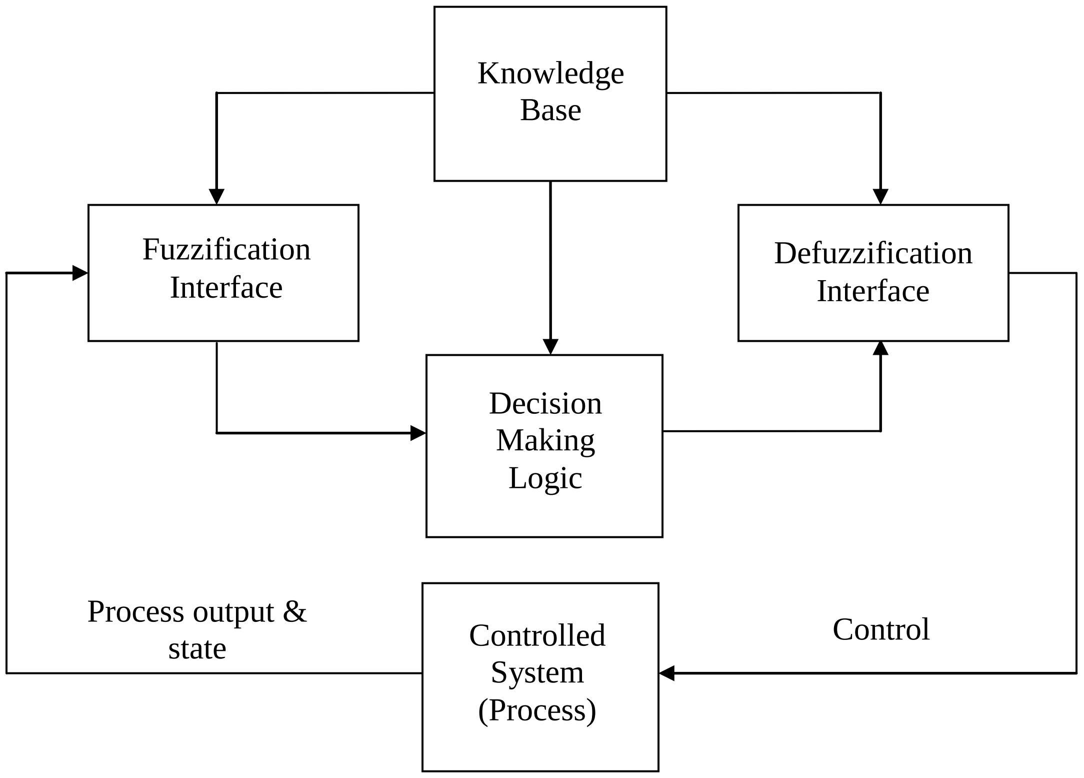


Figure 8: Block diagram of fuzzy logic controller

According to Figure 8, the FLC is divided into four basic components: fuzzification interface, knowledge base (KB), decision making logic, and defuzzification interface. The fuzzification interface includes the measurement of input values and by applying converts input values into linguistic variables. The knowledge base includes the knowledge of control goals and fuzzy rules. The decision making logic is the main part of FLC. It is like a human mind which also makes decisions according to fuzzy rules to obtain outputs. The last component is the defuzzification interface which gathers all the fuzzy outputs from the decision making logic component, combines them, and defuzzifes them to obtain a crisp, non-fuzzy, numerical control action [21].

3.3 Different models of fuzzy logic controllers

The Mamdani Model

Mamdani and Assilia used Zadeh's decision-making scheme to control systems [47]. Their controller is divided mainly into steps: First we get the measurements or readings from the system. Second we apply those measurement values to predefined fuzzy if-then rules and generate fuzzy outputs. These outputs are useful for decision-making but to control the system we need a crisp output, which leads us to the third step. The third step is to average all fuzzy outputs and defuzzify the average to obtain a crisp numerical output. Fuzzy if-than rules are some of the control parameters and are defined as follows.

$$\begin{gathered}\text{If } x_1 = A_1, x_2 = A_2, \ldots\ldots \text{ and } x_n = A_n \\ \text{Then } y = B\end{gathered} \tag{9}$$

where $x = \{x_1, x_2, \ldots, x_n\}$ and y are inputs and output respectively, and $A = \{A_1, A_2, \ldots, A_n\}$ and *B* are linguistic values, or fuzzy sets. Each system may have different fuzzy if-then rules [23]. An example of the practical application of this system is the temperature control of a freezer. The freezer's temperature sensors will sense the temperatures inside the freezer, which are the inputs of the system. Then there are defined linguistic values like "too hot," "hot," "good," "cool," and "too cool." According to the readings the fuzzy rules will apply and generate the output. In this case the output is how much time the compressor will stay on. Finally, we average all those fuzzy outputs to make one decision that will turn the compressor on or off.

This model provides a user-friendly representation of rules, but there are a few drawbacks of the Mamdani model. The first drawback is that when we use this model for systems which have a high number of inputs and outputs it requires a lot of computation. The second drawback is that it is very hard to get optimal solutions for systems by using this model because there are so many tuning parameters. Another limitation is that if the model does not cover all input combinations, then it may not be able to find an optimum solution.

The Takagi-Sugeno Model

To overcome these limitations, Takagi and Sugeno introduced a different fuzzy model [42], [34], call the Takagi-Sugeno model, the Takagi-Sugeno-Kang model, or the TSK model. They made a few changes in the fuzzy if-then rules. These are defined as follows:

$$\begin{gathered}\text{If } x_1 = A_1, x_2 = A_2, \dots\dots \text{ and } x_n = A_n \\ \text{Then } y = c_0 + c_1x_1 + c_2x_2 + \cdots \dots + c_nx_n\end{gathered} \tag{10}$$

where x_i, y, and A_i are the same as defined above for the Mamdani model, and c_i are weighting parameters. This representation of the rule contains more information than the Mamdani model, and therefore it requires fewer rules. For complex and multi-dimensional systems, Takagi and Sugeno’s fuzzy model is generally better than the Mamdani fuzzy model. The other advantage of Takagi and Sugeno’s fuzzy model is that it naturally combines the outputs of local models in a smooth way to get a combined output. This model provides more accurate solutions than the Mamdani model [41]. But for problems with fewer inputs and outputs, the Mamdani model is good because it is intuitive to define the rules. In this paper, for robot control we have used the Mamdani fuzzy model.

3.4 Summary

This chapter has given the definition of fuzzy logic and its terminology. It has also differentiated fuzzy logic from traditional crisp logic. The second section explained FLCs. The last section explained the Mamdani and TSK models and compared the models. To control systems having a smaller number of input and outputs, the Mamdani model is preferred; but for systems having more inputs and outputs, the TSK model is preferred. In general, fuzzy logic is often more robust than traditional and analytically obtained control systems; we can also adjust the fuzzy if-then rules to make the controller more accurate.

CHAPTER IV

MOBILE ROBOT CONTROL

Chapter IV gives a brief introduction of the overall control system of the wall-following robots. It also explains the hardware and the software of the robots. The first section explains the PCB layout and the different parts of the robots. The second section explains how the robots detect the wall. I have used two different controllers, the PID and the fuzzy controller, to maintain a constant distance from the wall. The third and the fourth sections describe the different parameters and constants used for these two controllers. The last section describes the hardware experimentand how we use BBO to optimize the controllers.

4.1 Hardware

The brain of the robot is the microcontroller PIC18F4520. It controls radio communication and synchronizes the sensors and motors. The PCB layout shown in Figure 9 is used in each robot.

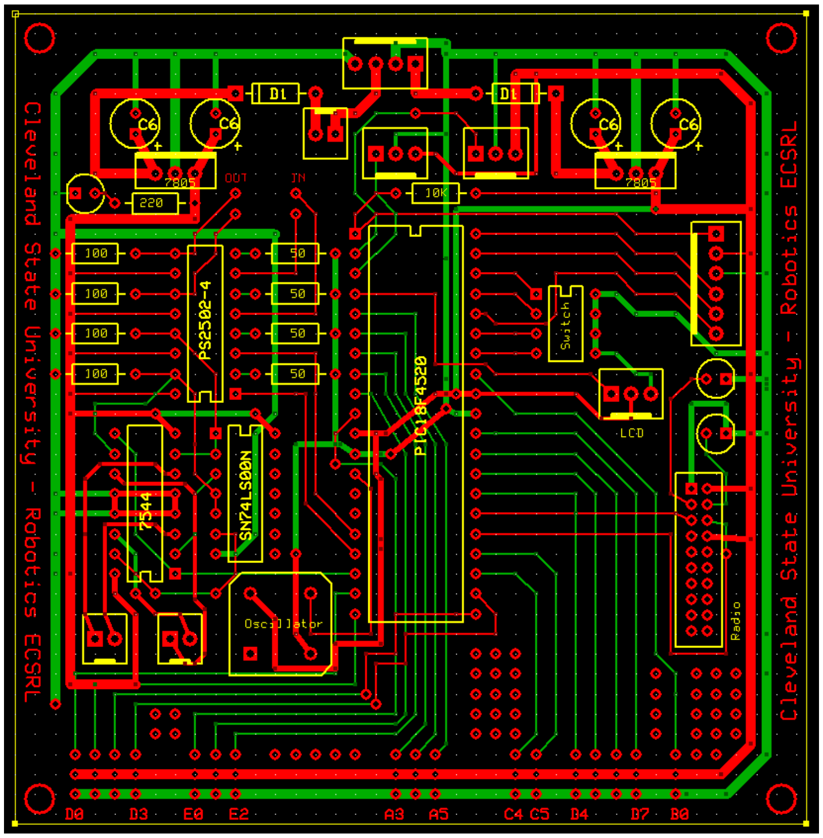


Figure 9: PCB layout used in robot [22]

Each robot contains two infrared (IR) sensors to get the distance from the wall, and two DC motors, one for each of the two rear wheels. Initially I used ultrasonic sensors, but because the DC motors make more noise affecting the sensors, I switched to infrared sensors. The effective range of each sensor is 10 centimeters to 80 centimeters and the typical response time is 39 milliseconds. In this experiment robots try to maintain a distance of 60 centimeters from the wall, which is well with the IR sensors' range. A photograph of the IR sensor is shown in Figure 10.

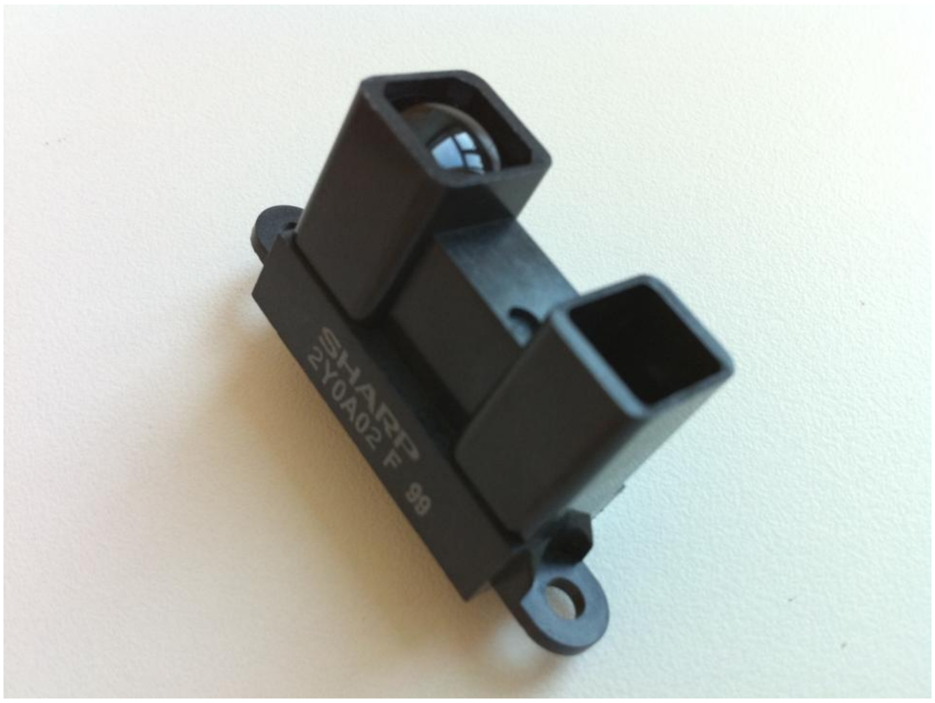


Figure 10: Photograph of IR sensor

The robots also include a MaxStream 9Xtend wireless radio to communicate with other robots and with the computer. The maximum outdoor RF line-of-sight of this radio is advertised as 40 miles. The receiver sensitivity is around –110 dBM. It has up to 1 Watt of power output, which is comparatively high for indoor use requirements. The 9Xtend has two different data rates: one is 9600 bps and other is 115200 bps. I have used the 9600 bps data rate for this experiment. One advantage of this radio is that it has low power consumption. Figure 11 shows a photograph of the MaxStrem 9Xtend wireless radio.

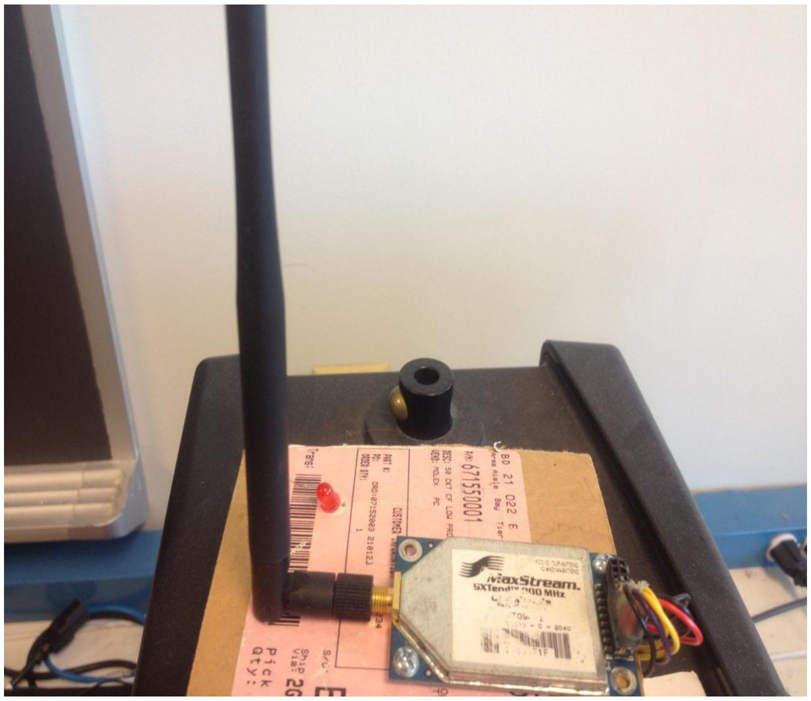


Figure 11: Photograph of MaxStrem 9Xtend wireless radio

There is one liquid crystal display (LCD) mounted on the robot, which will display robot status information. There are two packs of eight AA batteries; each provides 9 volts direct current (DC) supply, one for the motors and one for the digital electronics. There are two voltage regulators (7805). Each regulator generates a constant 5 V for the motor, microcontroller, and the rest of the components. The output current from the microcontroller pin is not large enough to run the motors. Thus, we used the SN754410NE quad H bridge to drive the motors. The actual robot looks like the one shown in Figure 12.

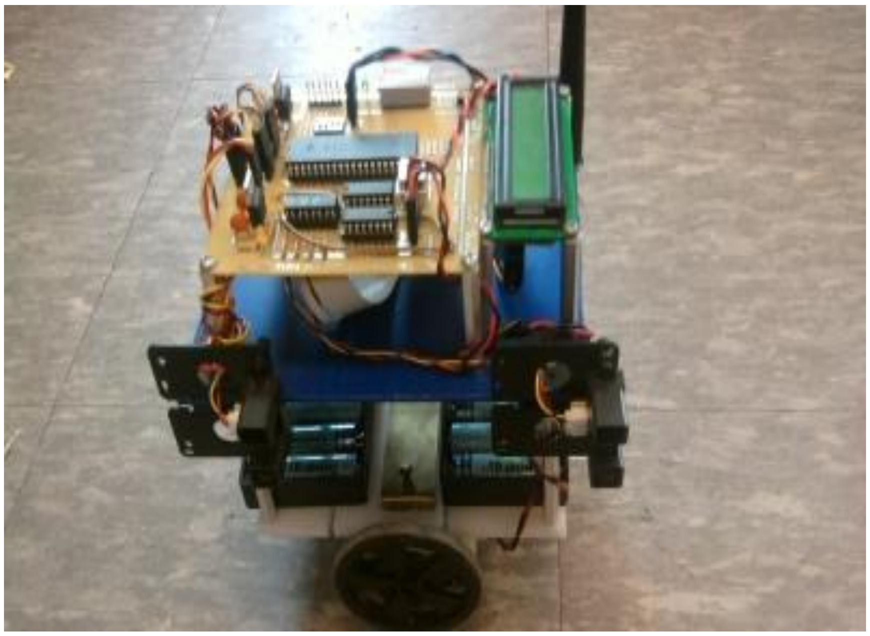


Figure 12: Photograph of the robot. Two IR sensors are seen on the left and right sides of the robot. The wireless radio antenna is seen at the back right side of the photo. The large chip in the middle of the PCB is the PIC18 microcontroller.

4.2 Implementation of control

The main function of each robot is to detect the wall and maintain a constant distance from the wall. Figure 13 shows a block diagram of the system. In each cycle the robots measure the distance from the wall, subtract measured distance from desired distance, and apply the controller to determine the desired steering direction and run the motors. The control parameters are optimized by the DBBO algorithm.

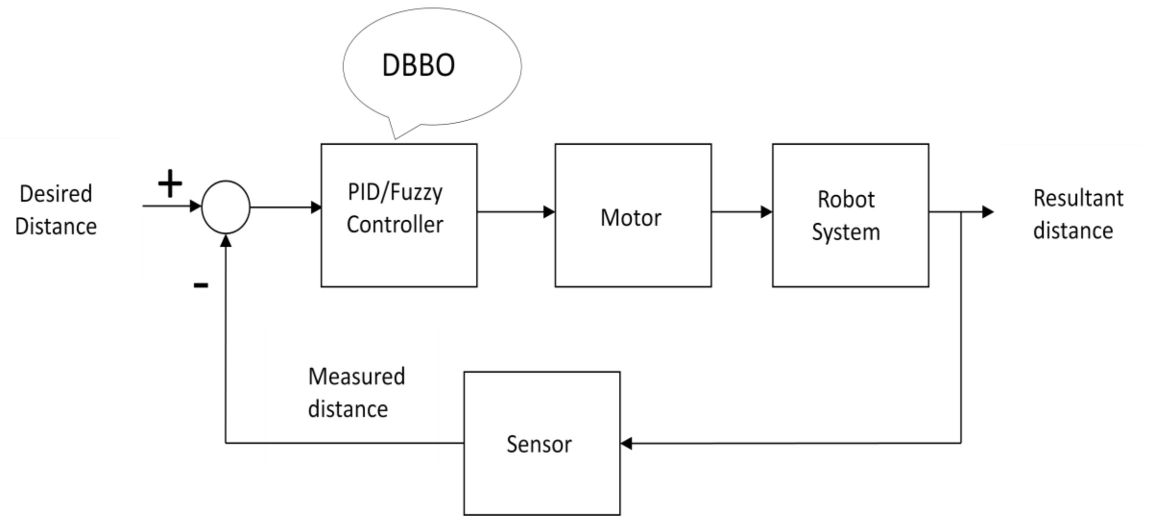


Figure 13: Block diagram of the robot system

The first step is to find the angle between wall and robot; in order to do so, the robot must find the distance from the difference of the readings of IR sensors (d_2–d_1) and use that measurement as one of the legs of a right angle triangle as shown in Figure 14. Then the robot must take the constant known distance (d_b) between the two sensors as the second leg of the triangle. Next, by using the Pythagorean equation, the angle is found as follows:

$$\theta = \tan^{-1}\frac{d_2 - d_1}{d_b} \tag{11}$$

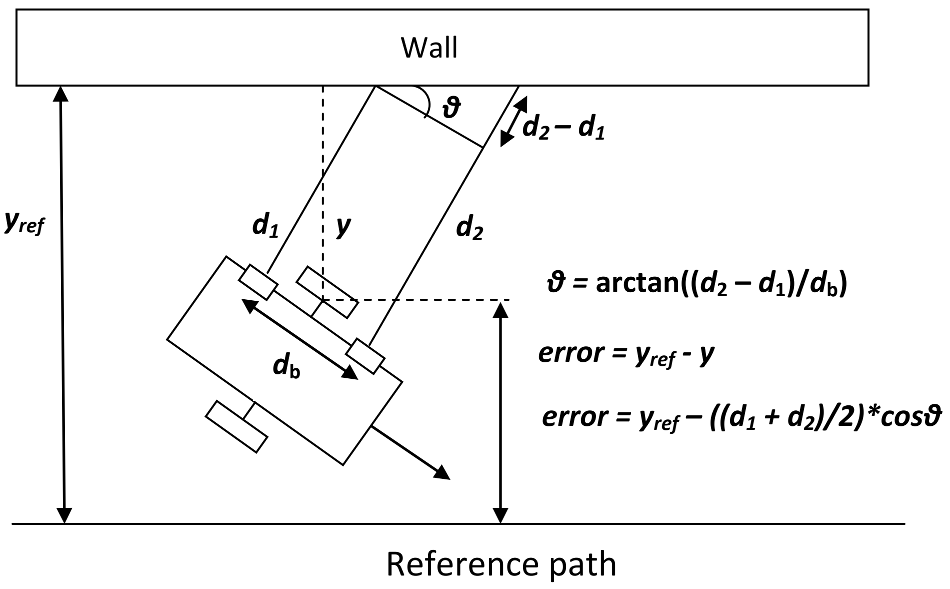


Figure 14: Detection of wall [22]

This angle shows which direction the robot is moving in. Then, the robot calculates the error as follows:

$$e(t) = y_{ref} - \left(\frac{d_2+d_1}{2} \times \cos\theta\right) \tag{12}$$

where y_{ref} is the constant distance that the robot tries to maintain from the wall. Then, we approximate the derivative of the error by taking the difference between the previous error and current error. Now, the robot takes tracking errors and derivative errors as the input to the controller, and the motor voltage correction value as the controller output. This analog voltage is used to set the PWM duty cycles for the motors. The resolution of the PWM values is an 8-bit unsigned integer, which ranges from 0 to 255. Therefore, the PWM resolution is 1/255 of 5 V. According to the output and the analog voltage of the controller, the PWM duty cycles will set for the left and right motors, which will control their speeds.

After finishing the wall-following task for a preset period of time (20 seconds in our case), each robot calculates a cost function depending on the tracking error as follows:

$$Cost = k_1 \int |e(t)|dt + k_2 r \tag{13}$$

where k_1 and k_2 are two constants which are used as weighting parameters. The value of $k_1 = 1$ and $k_2 = 5$ (determined through experimental trial and error to balance the two components of Equation 13), and $e(t)$ is the tracking error and r is the rise time. Rise time is defined as the time taken by the robot to reach 95% of the reference tracking distance. The lower limit of the integral is equal to the rise time.

4.3 Proportional integral derivative controller

PID is a very well-known algorithm for control systems. The block diagram of a PID controller is shown in Figure 15. There are mainly three parameters of the PID controller: proportional term k_p, integral term k_i, and derivative term k_d. The proportional term determines the amount of output signal according to the current error. As the value of k_p increases, the response time of the control system decreases. The integral term is proportional to both magnitude and time duration of the error. It speeds up the response and also reduces the steady state error. The last term of the PID controller is the derivative term k_d, which decreases the overshoot caused by the k_p and k_i terms.

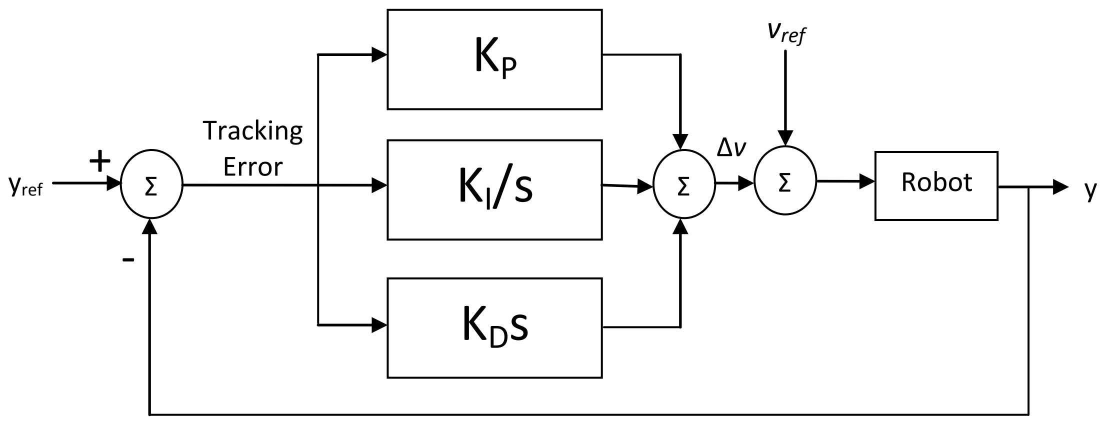


Figure 15: Block diagram of PID controller

As mentioned above, the inputs of the controller are the error and the delta error, and the output is the voltage correction for the motors. For the PID controller, the motor voltages are calculated as follows:

$$V_{left} = V_{ref} + 0.5\left(k_p e_{current} + k_d(e_{current} - e_{old})\right)$$
$$V_{right} = V_{ref} - 0.5(k_p e_{current} + k_d(e_{current} - e_{old})) \tag{14}$$

where V_{left} and V_{right} are the PWM duty cycle values of the left and right motors respectively, and $V_{ref}$is the reference PWM duty cycle, which is 240 for the reference speed of the motors. The range of values for the 8-bit unsigned integer is between 0 and 255. The values k_p and k_d are the parameters of the PID controller, and $e_{current}$ and e_{old} represent the current and previous error values.

In this experiment, I use only the proportional and derivative terms of the PID controller. Those two parameters are tuned by the DBBO algorithm. Each generation, each robot exchanges PID parameters with each other, applies the BBO algorithm, and generates new parameters, which decreases the error as well as the cost value of Equation 13.

4.4 Fuzzy controller

The second controller is the fuzzy controller. As I mention in Chapter III, there are several models for the FLC. In this experiment, we use the Mamdani model. The first step for the FLC is to define membership functions. So for the robot control we have defined five triangular membership functions (MF) which are: large positive (LP), small positive (SP), zero (Z), small negative (SN), and large negative (LN) as shown in Figure 16. So, each input and output of the system is mapped to five MFs. Now, according to the model, the system has *n* membership functions to describe each variable and therefore the fuzzy set description must have a total of *n* break points. So as we have defined five membership functions, each input and output variable has five break points as shown in Table 2. The DBBO algorithm will modify the shape of the MFs by modifying these break points.

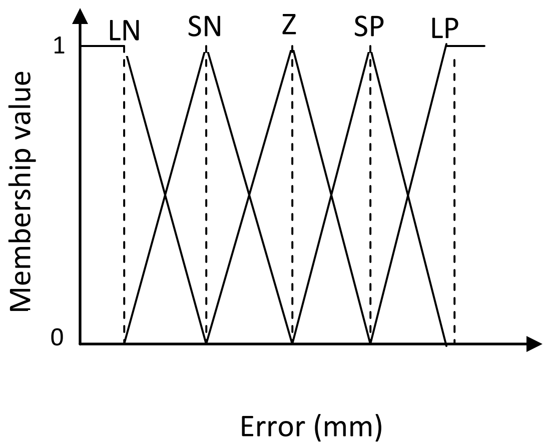


Figure 16: Representation of membership functions [22]

Table 2: Range of each fuzzy parameter

DBBO domain		Break Points				
		LN	SN	Z	SP	LP
Variables	Error (mm)	[–1000, –250]	[–250, 0]	[0, 0]	[0, 250]	[250, 1000]
	ΔError (mm)	[–100, –25]	[–25, 0]	[0, 0]	[0, 25]	[25, 100]
	ΔMotor voltage	[–100, –25]	[–25, 0]	[0, 0]	[0, 25]	[25, 100]

The second step is to define fuzzy if-then rules for the FLC. We have used the maximum and minimum of the parameters to generate the fuzzy if-then rules. Each intersection between the pair of input MFs will represent one rule. As mentioned previously, there are two inputs, error and delta error, and one output, which is the delta voltage of the motor. There are a total of 25 fuzzy if-then rules for this system as shown in Table 3.

To these fuzzy rules, we use minimum inference, which works as follows. FLC identifies the output MF γ and calculates its MF as the minimum of the input MFs. For example, suppose the error of the robot is between [–1000, –250] so it belongs to the fuzzy set LN, which indicates the robot is far from the reference line, and Δerror is between [0, 25] so it belongs to the fuzzy set SP, which indicates that it is getting closer to the reference line. Then according to Table 3 the fuzzy rule for output motor voltage is SN which includes values that are between [–25, 0], which indicates that the robot should maintain its direction with a slight decrease of motor voltage on the left wheel to try to reach to the reference line.

Table 3: Rule table for fuzzy if-then rules (LP = large positive, SP = small positive, Z = zero, SN = small negative, and LN = large negative)

Rule Table		Error				
		LN	SN	Z	SP	LP
ΔError	LN	LN	LN	LN	SN	Z
	SN	LN	LN	SN	Z	SP
	Z	LN	SN	Z	SP	LP
	SP	SN	Z	SP	LP	LP
	LP	Z	SP	LP	LP	LP

The third step of the Mamdani FLC is to convert the fuzzy output into a crisp numerical output. So for example if error has a 'small negative' membership function and Δerror has a 'zero' membership function as shown in Figure 17, then the output voltage is assigned to the 'small negative' membership function with a membership that is equal to the minimum of the memberships of the error and Δerror. Now the robot finds all minimums of the output and calculates the centroid of the fuzzy output as follows [21].

$$\text{Centroid} = \sum_{i=1}^{N} \min(\alpha_i, \beta_i) \times \frac{(\alpha_i + \beta_i)}{2} \tag{15}$$

where α_i represents the i-th MF of the error input, β_i represents the i-th MF of the derivative error, and N is the total number of fuzzy rules.

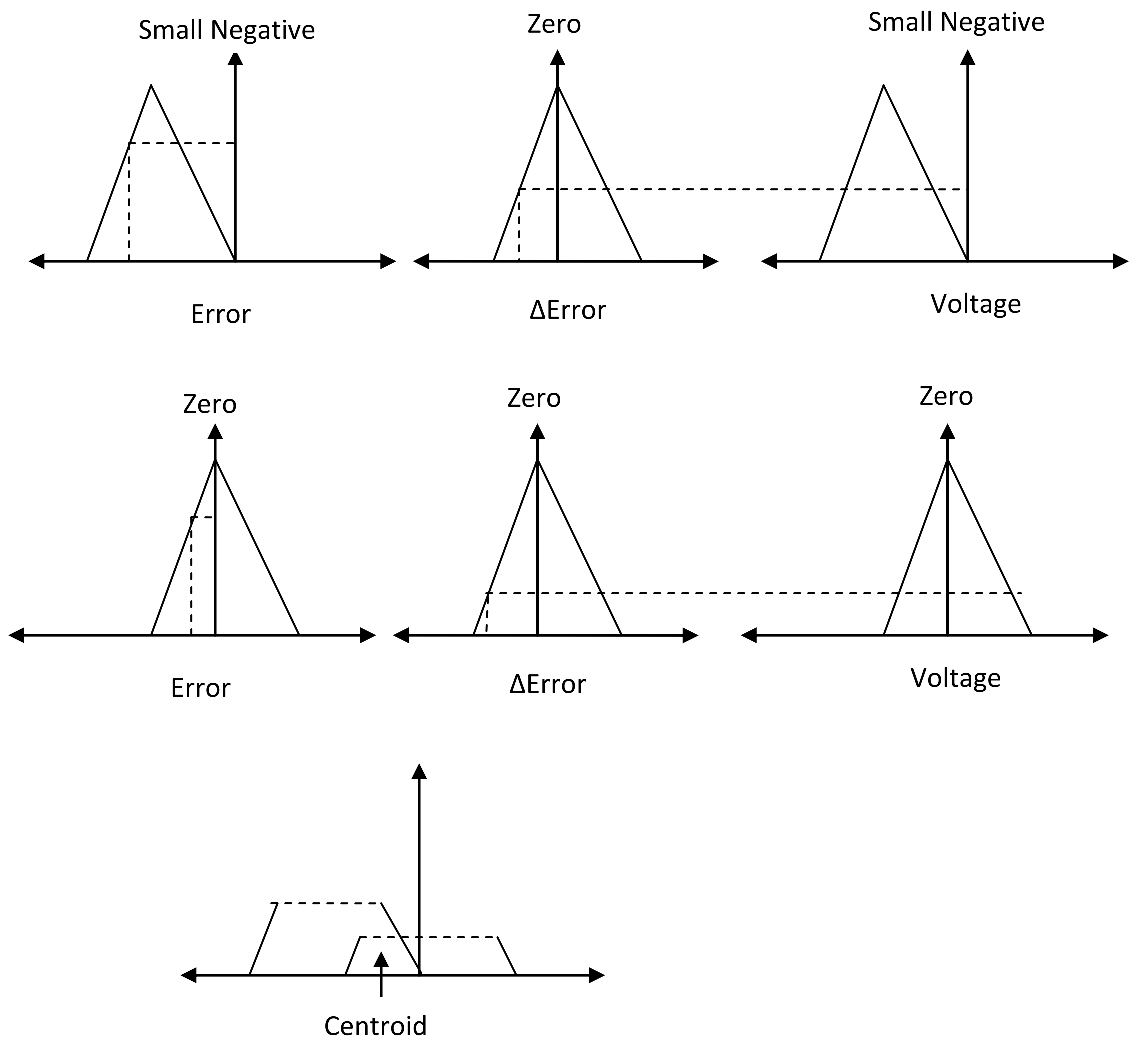


Figure 17: Defuzzification of fuzzy output

Now calculate the defuzzified output as shown below [21].

$$\text{Defuzzified output} = \frac{\text{Centroid}}{\sum_{i=1}^{N} \min(\alpha_i, \beta_i)} \tag{16}$$

Finally calculate the voltage for both motors as follows:

$$\begin{aligned} V_{left} &= V_{ref} + D \\ V_{right} &= V_{ref} - D \end{aligned} \tag{17}$$

where *D* is the defuzzified output. So in general, for each sample time (0.1 seconds in our experiments) during wall following, robots use the algorithm shown in Figure 18 to maintain a desired distance from the wall. DBBO is used to tune the break points of the fuzzy MFs and thus optimize the controller.

Figure 18: Pseudo code for fuzzy logic controller

```
Get input values
Map input values into MFs
For each MF, compute $(\alpha, \beta)$ as MF values
For each $(\alpha, \beta)$ combination
    If $\alpha \neq 0$ and $\beta \neq 0$
    Identify rule and output MF $\gamma$ – see Table 3
end
Calculate centroid $C$ – see equation 15
Calculate defuzzified output  – see equation 16
```

4.5 Experimental procedure

This section explains how the wall following controller and the DBBO algorithm run in each robot. The whole experiment is controlled by the external desktop computer. The user gives commands from the computer using a radio and gets the resultant data from the robots. Each robot has its own unique robot id. The flow of the software that is programmed in each robot is shown in Figure 19.

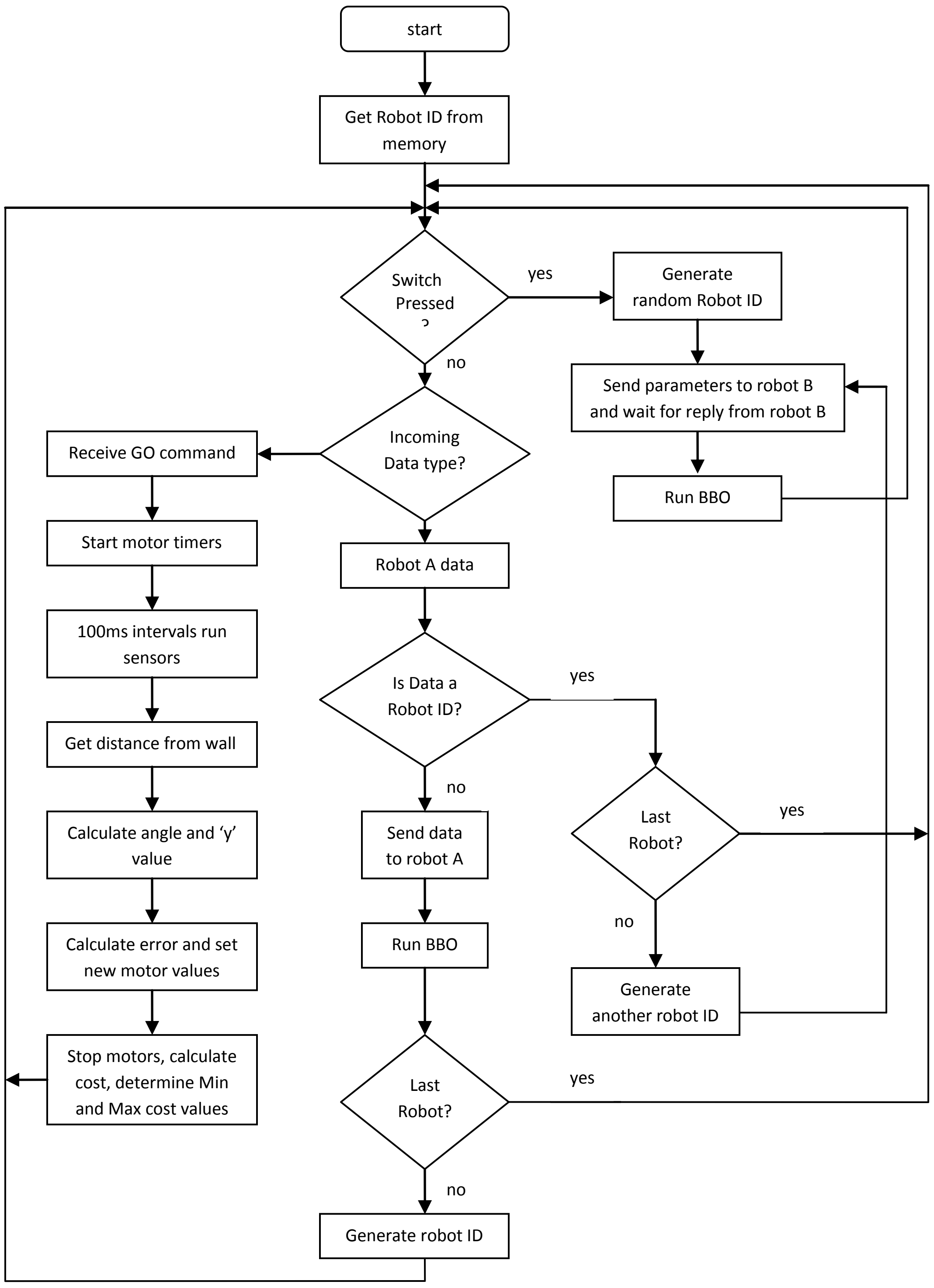


Figure 19: Flowchart of the system

The experiment will start when the user sends the command “Begin Run” to the robots. All robots start running and following the wall according to their control algorithm. During every run, each robot will take 200 distance readings with the IR sensor. Each reading is taken every 1/10 second. After taking readings, the microcontroller calculates the angle and distance *y* from the wall. It also calculates the error and derivative of error. The controller determines the desired motor voltage variation as its output. After the 20-second wall following sequence, the robot goes in idle mode. Then the user has to press a switch on one of the robots to start the DBBO algorithm.

As soon as the user presses a switch on any of the robots, they start to communicate with each other. As shown in Figure 19, first robot A, whose switch is pressed, generates a random robot ID, which we call robot B, and sends its own control parameters and its maximum cost estimate, minimum cost estimate, and most recent cost value to robot B. Then it waits for the response from robot B. As soon as robot B gets data from robot A, it sends its own control parameters, maximum cost estimate, minimum cost estimate, and most recent cost value back to robot A. Now both robots have data from each other. So both robots will perform the BBO algorithm independently and set their own new control parameters. Then robot B will generate a new robot ID other than its own and robot A’s, and send the list of the robot IDs which have already been used to the new robot ID. Then the receiver robot will perform the same sequence of operations as explained above.

All robots communicate with each other according to the above description and perform the BBO algorithm independently. As soon as communication completes, the user will send the “Get Data” command to each robot simultaneously from the desktop computer. Each robot will send its tracking data, its new control parameters, and its maximum estimated, minimum estimated, and most recent cost values to the computer. The MATLAB® application on the computer will create a new data sheet in Microsoft Excel and save all data. It also plots the tracking data and membership functions if the fuzzy controller is being used. After that, a new DBBO generation will start.

The PIC® controller synchronizes all the components. I have used the Code Composer Studio™ compiler to program the PIC. The timer0 module of the PIC is used to take the sensor readings every 100 ms and find *y* and tracking error. Two serial ports are used: one is for the LCD connection and the other is for radio communication. Two PWM modules are used to generate the PWM signals for the two DC motors.

CHAPTER V

RESULTS

In this chapter, I am going to explain the results I obtained during my experiments. I have done simulation as well as hardware experiments to analyze the performance of the DBBO algorithm. The first section explains the results I obtained from the MATLAB simulation. It also compares the results from the PID and fuzzy controllers. The second section explains the hardware experiment results. As I mentioned in previous sections, I have used wall-following robots, applied DBBO to them, and analyzed the performance. I have used two different controllers (PID and fuzzy) and compared the results.

5.1 Simulation results

Simulation results were generated in MATLAB. I wrote a robot program in MATLAB which simulates the robot's function of maintaining a constant distance from the wall. I have set the population size at 50 and mutation rate at 1%. I ran BBO and

DBBO with 500 function evaluations (that is, a total of 500 robot simulations). I chose the simulation parameters as above to make the simulation similar to the robot hardware experiment. I have used two different controllers (PID and fuzzy) in my simulation experiment as explained below.

5.1.1 DBBO using proportional integral derivative controller

I have done 100 Monte Carlo simulations for DBBO using a PID controller. I ran DBBO using 2, 4 and 6 peers. The range of the k_p and k_d parameters is between [0 1] and [0 10] respectively, ranges were set empirically. During the first generation, the computer randomly generates the parameters from the given range.

Table 4 shows the minimum, maximum, and average costs, and the standard deviation of the cost for the robots. The minimum is the best cost achieved over all generations after 100 Monte Carlo simulations, the maximum is the worst of the 100 best costs achieved by the 100 simulations, and the average is the mean of the best costs achieved by the 100 simulations. The readings show that there is not a big difference among the different algorithms; still the average cost of BBO is the lowest compared to DBBO with different numbers of peers, while DBBO/6 has the lowest minimum cost compared to the other algorithms. The standard deviation shows that the BBO algorithm has less fluctuation and is thus more consistent than the others. So overall BBO outperforms the DBBO, but DBBO still has good results.

Table 4: 100 Monte Carlo simulation results for PID controller

	BBO	DBBO/2	DBBO/4	DBBO/6
Minimum Cost	7.48	7.23	7.30	**7.16**
Maximum Cost	**7.99**	8.12	8.07	8.10
Average Cost	**7.68**	7.78	7.77	7.76
Standard Deviation	**0.119**	0.169	0.147	0.193

The above result is not enough to numerically differentiate between the performance of the BBO and the DBBO algorithm, so I have used T-tests. I obtained T-test results for BBO and DBBO as shown in Table 5. The numbers in the table show the probabilities that the differences between two different experiments are due solely to random fluctuations, and are not due to fundamental differences between the algorithms. If the result is less than 0.05, then we conclude that those two algorithms are different, while if the result is greater than 0.05, then we conclude that there is not enough numerical evidence to conclude that those two algorithms are different. The results show that the BBO and the DBBO algorithms are different, but DBBO with different numbers of peers are not different.

Table 5: T-test results (probabilities) for the PID controller

	BBO	DBBO/2	DBBO/4	DBBO/6
BBO	1	**1.25E–06**	**2.68E–06**	**0.00036**
DBBO/2	**1.25E–06**	1	0.5971	0.4084
DBBO/4	**2.68E–06**	0.5971	1	0.7003
DBBO/6	**0.00036**	0.4084	0.7003	1

5.1.2 DBBO using fuzzy controller

I have also done 100 Monte Carlo simulations for DBBO with different numbers of peers using the fuzzy controller. Similar to the PID controller, the fuzzy controller also randomly generates its control parameters during the first generation from the given range. The minimum values and maximum values for these parameters are shown in Table 6. These ranges were determined empirically.

Table 6: Minimum and maximum values of fuzzy membership function breakpoints used in mobile robots

Input/Output	Membership function	Minimum value	Maximum value
Error (mm)	Large negative (LN)	–1000	–500
	Small negative (SN)	–500	0
	Zero(Z)	–0.0001	0.0001
	Small positive (SP)	0	500
	Large positive (LP)	500	1000
ΔError (mm)	Large negative (LN)	–100	–25
	Small negative (SN)	–25	0
	Zero(Z)	–0.0001	0.0001
	Small positive (SP)	0	25
	Large positive (LP)	25	100
ΔVoltage (PWM counts)	Large negative (LN)	–100	–25
	Small negative (SN)	–25	0
	Zero(Z)	–0.0001	0.0001
	Small positive (SP)	0	25
	Large positive (LP)	25	100

Table 7 shows the results of the simulation experiments. The table shows that BBO has the lowest minimum cost and average cost compared to the DBBO algorithms, but DBBO/4 has the lowest standard deviation. Among DBBO algorithms, DBBO/6 has the lowest minimum cost and lowest average cost. The standard deviations for the fuzzy controller are higher than the PID controller.

Table 7: 100 Monte Carlo simulation results using fuzzy controller

	BBO	DBBO2	DBBO4	DBBO6
Minimum Cost	**5.65**	5.81	5.97	5.78
Maximum Cost	**7.74**	9.19	7.94	8.48
Average Cost	**6.56**	7.13	6.92	6.96
Standard Deviation	0.48	0.75	**0.43**	0.55

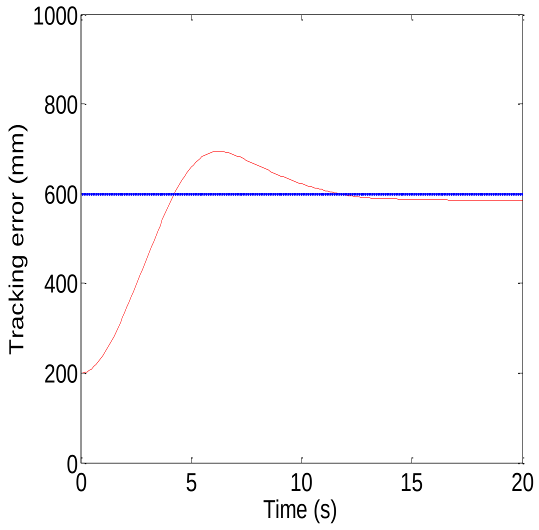
I have performed T-tests on the fuzzy control tuning results. According to Table 8, there is a statistically significant difference between the results of the BBO algorithm and DBBO.

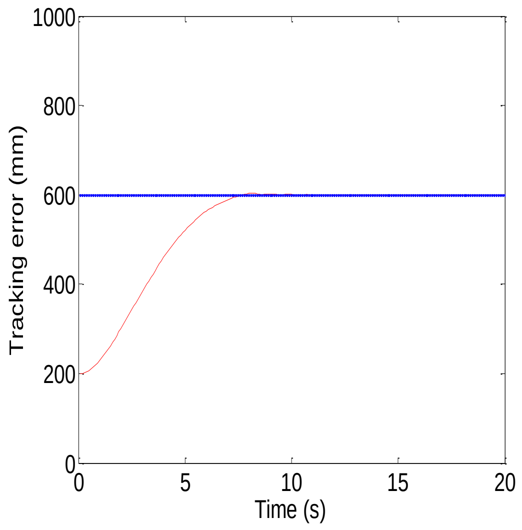
Table 8: T-test results for the fuzzy controller

	BBO	DBBO/2	DBBO/4	DBBO/6
BBO	1	**1.96E-05**	**0.0001**	**0.0002**
DBBO/2	**1.96E–05**	1	0.0926	0.2000
DBBO/4	**0.0001**	0.0926	1	0.6995
DBBO/6	**0.0002**	0.2000	0.6995	1

5.1.2(a) Different starting points

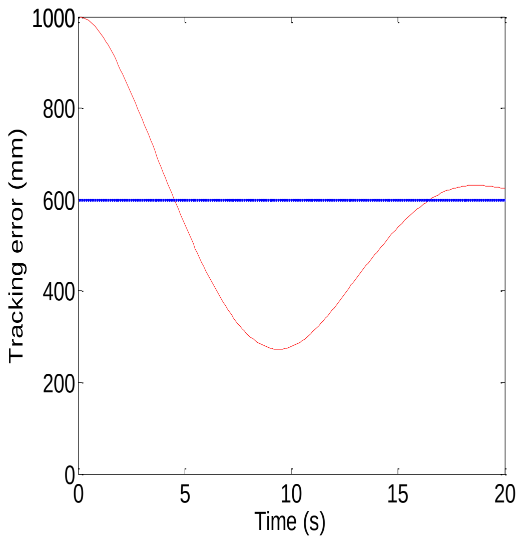
I have done several additional simulation experiments on the fuzzy controller using BBO and DBBO with different number of peers. First, I have run each algorithm for 100 generations and collected the tracking error data. During this experiment, I have chosen several starting points. Figure 20 shows the tracking error responses of the best individual at the 1st generation and the best individual at the 100th generation when the robots start 200 mm from the wall, and Figure 21 shows the tracking error responses when the robots start 1000 mm from the wall. The graph shows that during the first generation the controller takes more time to reach the reference distance compared to the 100th generation. This illustrates the effectiveness of BBO.

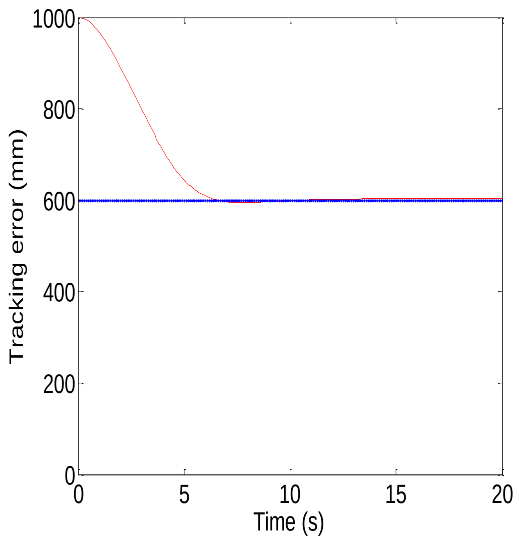




(a) (b)

Figure 20: Tracking error response of the fuzzy controller when the robots start point 200 mm from the wall. (a) best BBO individual at 1st generation (b) best BBO individual at 100th generation





(a) (b)

Figure 21: Tracking error response of the fuzzy controller when the robots start 1000 mm from the wall. (a) best BBO individual at 1st generation (b) best BBO individual at 100th generation

Figure 22(a) shows the five membership functions of the best robot at the first generation, and Figure 22(b) shows the five membership functions at the 100th generation, when the starting point is 200 mm from the wall. Similarly, Figure 23 shows the five membership functions at the first and the 100th generation when the starting point is 1000 mm from the wall. From Figure 22 you can see that the 3rd membership function corresponds to positive error for the robot whose starting point is 200 mm, and the 3rd membership function corresponds to negative error for the robot whose starting point is 1000 mm. Figures 22 and 23 show how the membership functions change from the 1st to the 100th generation, which indicates that after every generation DBBO adjusts the controller in order to improve its response.

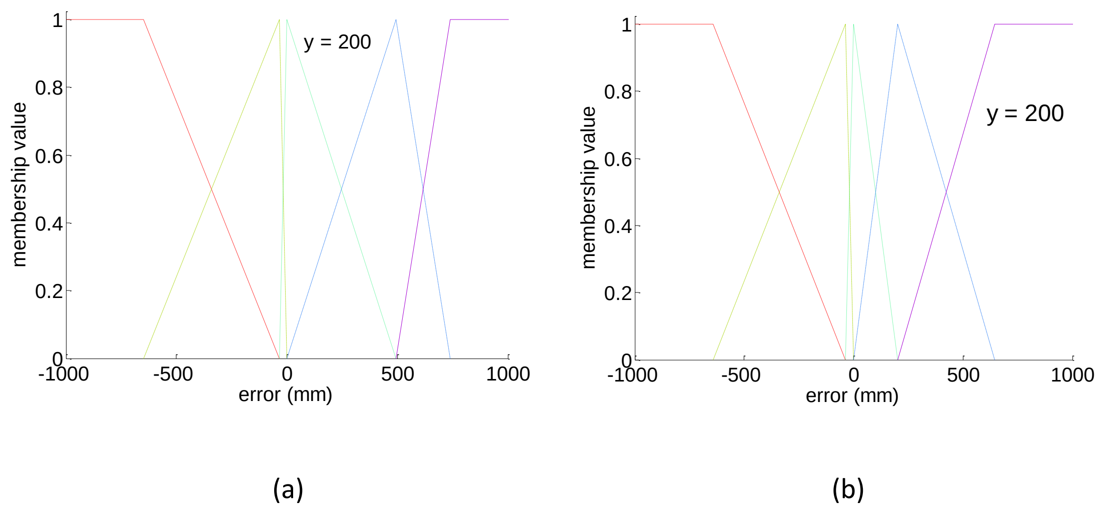
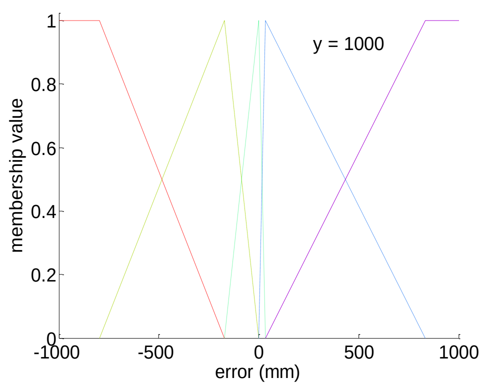
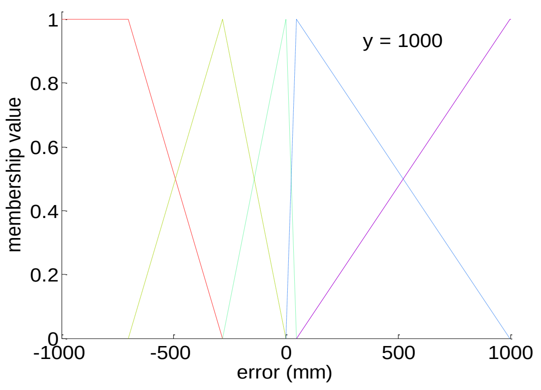


Figure 22: Fuzzy controller membership function of robots starting at 200 mm (a) best BBO individual at 1st generation (b) best BBO individual at 100th generation



(a)



(b)

Figure 23: Fuzzy controller membership function of robots starting at 1000 mm (a) best BBO individual at 1st generation (b) best BBO individual at 100th generation

Figure 24 shows the average cost values of the robots at different starting points from the wall when different algorithms were used for optimization. The reference distance for the robots to maintain from the wall is 600 mm. I have used 200 mm, 400 mm, 600 mm, 800 mm, and 1000 mm starting points to optimize the controllers and find the cost values. The cost value at the starting point 600 mm is close to zero because, as I mention above, the reference line which a robot has to follow is 600 mm. So, for this case, the robot is already on the reference line, therefore, it does not have to fluctuate. The graph shows that the cost values at starting points 200 mm and 1000 mm are similar to each other, and the cost values at starting points 400 mm and 800 mm are similar to each other. This implies that if the distance from the starting point to the reference line is equal, then the cost values will be equal, which indicates that the performance of the robots is symmetric.

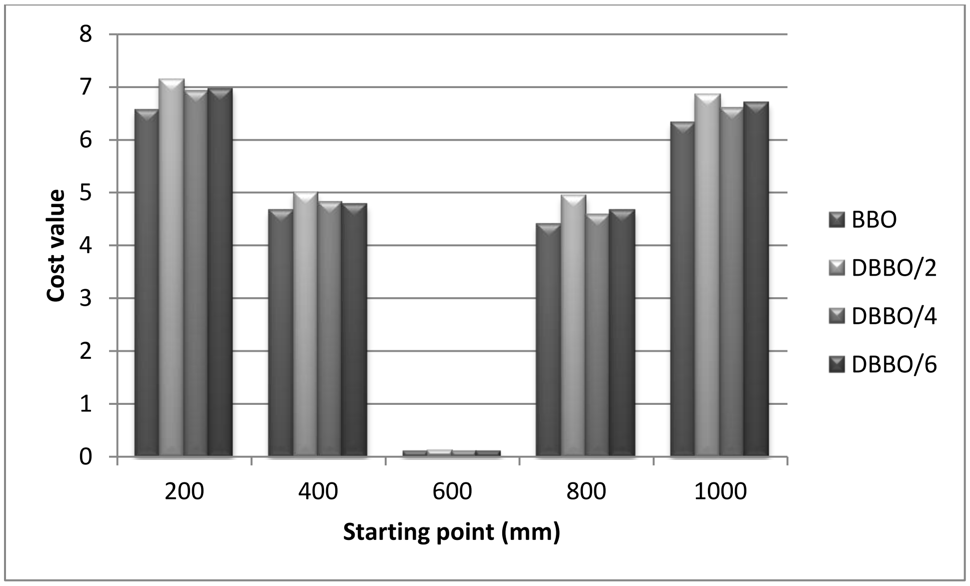


Figure 24: Average cost values at different starting points

5.1.2(b) Different mutation rates

Secondly, I have done experiments by using different mutation rates. Table 9 shows the average cost values at 1% and 10% mutation rates for BBO and DBBO with different numbers of peers. The readings show that there is a difference in cost values for all algorithms as the mutation rate changes, but DBBO/2 has the highest change. So, from the readings, we can say that the mutation rate makes a significant difference in cost values for a small number of peers. According to the mutation rate, a candidate solution generates new random solutions and replaces them with the duplicate solutions during each generation. When we apply DBBO/2, only two candidates at a time communicate and exchange their controller parameters. So there is a greater probability of having duplicate solutions (that is, duplicate controllers) in the robot population. Now, as I mentioned earlier, as the mutation rate goes higher, the probability of replacing the duplicate solutions with new solutions is high, so for a high mutation rate, the population has fewer duplicate solutions, which provides the potential to reduce the best cost value of the robot population. Algorithms other than DBBO/2 include more robots communicating with each other, which results in fewer duplicate solutions, so increasing the mutation rate for those algorithms does not make a significant change in performance.

Table 9: Average cost at different mutation rates

Mutation rate	1%	10%
BBO	4.04	4.36
DBBO/2	**6.20**	**4.94**
DBBO/4	4.49	4.76
DBBO/6	4.46	4.90

5.1.2(c) Different robot wheel base lengths

Up to this point, I have used only one wheel base length for the robot simulations but to study the control algorithm in more detail, I have to use different wheel base lengths for the robots. Therefore, I have used different wheel base lengths of the robots and tuned the fuzzy controllers using different BBO and DBBO algorithms. For the previous experiments the wheel base was 185 mm. So for this experiment, I used wheel base lengths of 175 mm, 180 mm, 190 mm, 195 mm, and 200 mm. I have done 50 Monte Carlo simulations and obtained the average cost values of the best costs from 50 simulations as shown in Table 10.

Table 10: Average cost for different wheel base lengths

Length (mm)	BBO	DBBO/2	DBBO/4	DBBO/6
175	**6.48**	6.90	6.75	6.80
180	**6.53**	6.96	6.75	6.74
185	**6.56**	7.13	6.92	6.96
190	**6.65**	7.28	6.96	6.94
195	**6.71**	7.04	7.03	7.05
200	**6.90**	7.30	7.23	7.19

The results of Table 10 indicate that as the robot wheel base increases the cost value also increases. The reason is that the mathematical model of the robot dynamics includes the following equation for the derivative of the robot angle:

$$\dot{\theta} = (v_r - v_l)/L \tag{18}$$

where v_r is the velocity of right wheel, v_l is the velocity of left wheel, and L is the length of the wheel base. According to Equation 18, as the robot wheel base length increases, $\dot{\theta}$ decreases, which indicates that robots having a short wheel base can change their angle more rapidly compared to robots having a larger wheel base, and this makes the robot more controllable.

5.1.2(d) Robustness tests

I have also performed robustness tests for the BBO and the DBBO algorithms, where robustness is defined as follows: "*The robustness/ruggedness of an analytical procedure is a measure of its capacity to remain unaffected by small, but deliberate variations in method parameters and provides an indication of its reliability during normal usage*" [13]. We can divide robustness into two categories. The first is the degree of reproducibility when changing external conditions like analytical equipment, analyst, laboratory, etc. These are called inter-laboratory trials. The second is the degree of reproducibility when changing the experimental parameters like temperature, experimental time, etc. This is called an intra-laboratory study. In this thesis I use the second category of robustness. I vary experimental parameters [31].

As I mentioned in Chapter I, to apply the DBBO algorithm, I have used four wall-following robots. So for robustness tests I have taken the wheel base of these robots as a variable. I train the robot controller having a one wheel base, use these control parameters in other robots with different wheel bases, and examine the cost values for the robots. Figure 25, 26, 27, and 28 show the percentage cost deviations calculated as follows

$$\text{Percentage cost deviation}$$
$$= \left(\frac{cost\ value\ of\ untrained\ robot - cost\ value\ of\ trained\ robot}{Cost\ value\ of\ trained\ robot}\right)100 \quad (19)$$

The horizontal axis represents the wheel bases for which the robot is trained; I then used the optimized control parameters on robots with different wheel bases and obtained the new cost values. The vertical axis represents the percentage difference between the cost values of the trained robot and the other robots with different wheel bases. I used BBO, DBBO/2, DBBO/4, and DBBO/6 for the robustness tests. The details of these tests are shown in Appendix A

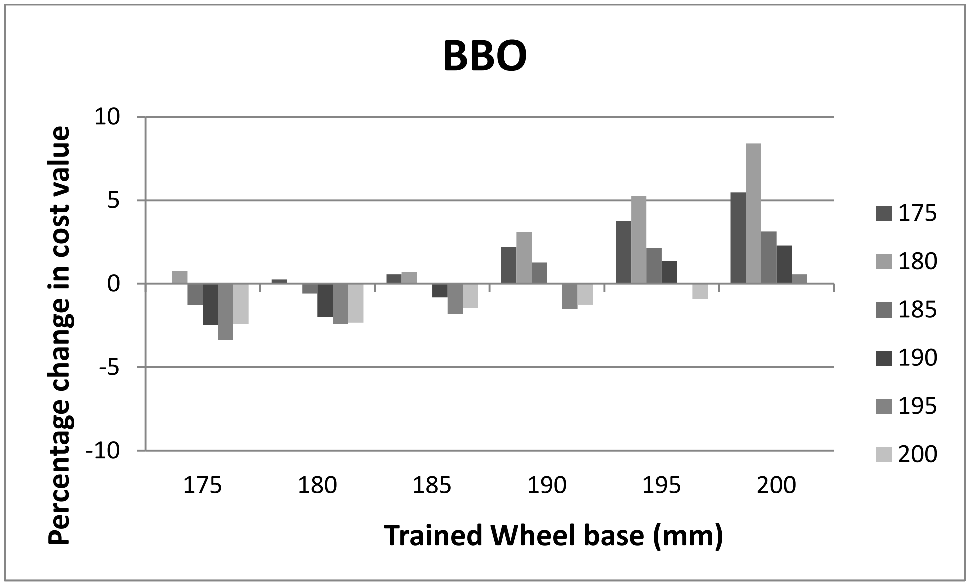


Figure 25: Robustness test variation in cost values of robots with different wheel bases using BBO. Each cluster of bars on the chart represents the percentage cost deviations when robots with different wheel bases use the control parameters that are optimized using the wheel base shown on the horizontal axis.

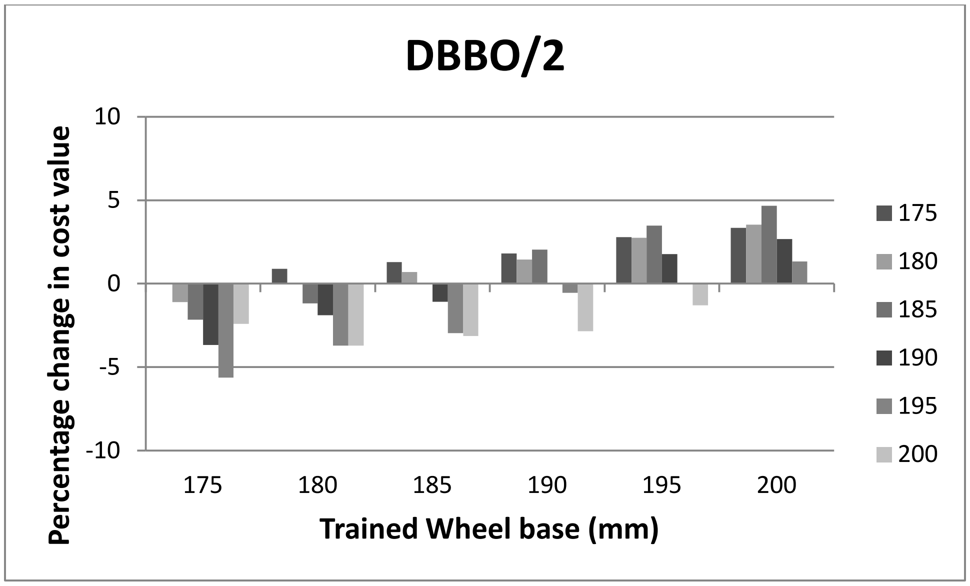


Figure 26: Robustness test variation in cost values of robots with different wheel bases using DBBO/2. Each cluster of bars on the chart represents the percentage cost deviations when robots with different wheel bases use the control parameters that are optimized using the wheel base shown on the horizontal axis.

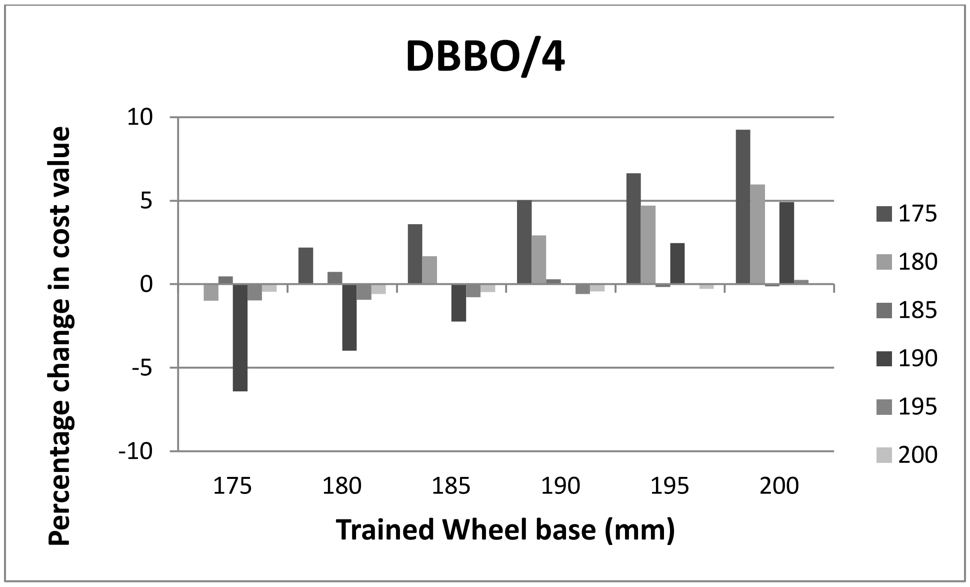


Figure 27: Robustness test variation in cost values of robots with different wheel bases using DBBO/4. Each cluster of bars on the chart represents the percentage cost deviations when robots with different wheel bases use the control parameters that are optimized using the wheel base shown on the horizontal axis.

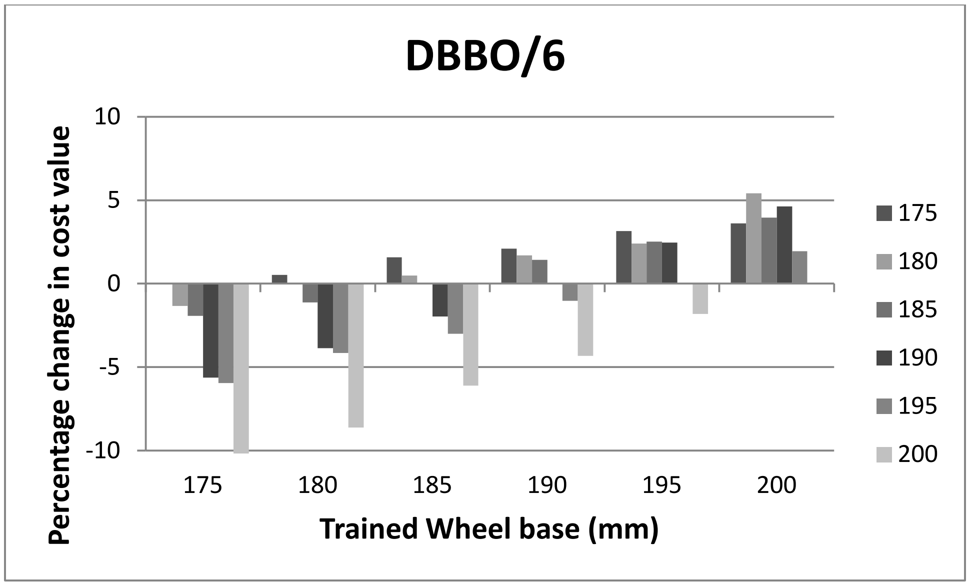


Figure 28: Robustness test variation in cost values of robots with different wheel bases using DBBO/6. Each cluster of bars on the chart represents the percentage cost deviations when robots with different wheel bases use the control parameters that are optimized using the wheel base shown on the horizontal axis.

Figure 25, 26, 27, and 28 indicate that there is no more than 10% variation in the performance of the robots when the wheel base changes in the range 175 mm to 200 mm. Among the four algorithms, the worst-case variation is the smallest for DBBO/2 (less than 6%), while the average variation is the smallest for BBO.As I mentioned previously, the cost value increases when the wheel base increases. Thus, when I trained a robot with a 185 mm wheel base, and use those parameters for a smaller wheel base, it gives me a positive difference. If I use the parameters for a larger wheel base it gives me negative difference. In general, DBBO is robust. I ran T-tests on the robustness results, and the results show that although the changes in robot cost values are small they are also statistically significant. The details of the T-test results are in Appendix B.

5.2 Experimental results

I have used four wall-following robots and applied DBBO/2 to their control parameters to improve the performance of the robots. During each experimental cycle, the robots start 200 mm from the wall, follow the wall for 20 seconds, and then calculate the cost. For the hardware experiment, I have made a minor change in the definition of the cost function. During the simulation experiment, the cost value is calculated by taking the integral of error after the robot reaches the rise time (see Equation 13). During the hardware experiment the cost value is calculated by taking the integral of error from the starting point. Therefore, the cost value in the simulation result is small compared to the hardware results. After each BBO or DBBO generation, two robots randomly communicate with each other and exchange control parameters and cost values. As mentioned in the previous section, I have used two different controllers (PID and fuzzy) to maintain the reference distance from the wall.

5.2.1 DBBO using proportional integral derivative controller

At the start of the first generation, each robot randomly generates two PID control parameters k_p and k_d. (Recall that we are not using integral control in these experiments.) The range of the k_p parameter is [0 1] and that of the k_d parameter is [0 10]. Table 11 shows the k_p and k_d values at the 1st and 8th generations of each robot.

Table 11: PID parameters of the DBBO/2 optimized robots at the 1st and 8th generations

	1st generation	8th generation
Robot 1	k_p= 0.93, k_d = 4.26	k_p= 0.82, k_d = 9.03
Robot 2	k_p= 0.07, k_d = 6.36	k_p= 0.07, k_d = 3.41
Robot 3	k_p= 0.18, k_d = 2.45	k_p= 0.67, k_d = 4.32
Robot 4	k_p= 0.12, k_d = 2.21	k_p= 0.02, k_d = 2.03

During every generation, each robot generates control parameters using DBBO, tracks the wall, and calculates the cost value. Then, two robots randomly communicate with each other, run one generation of DBBO, and adjust their controllers. Thus, as shown in Figure 29, after several generations the cost value decreases, which indicates that the tracking error also decreases. Figure 29 shows the best cost value among the four robots at each generation.

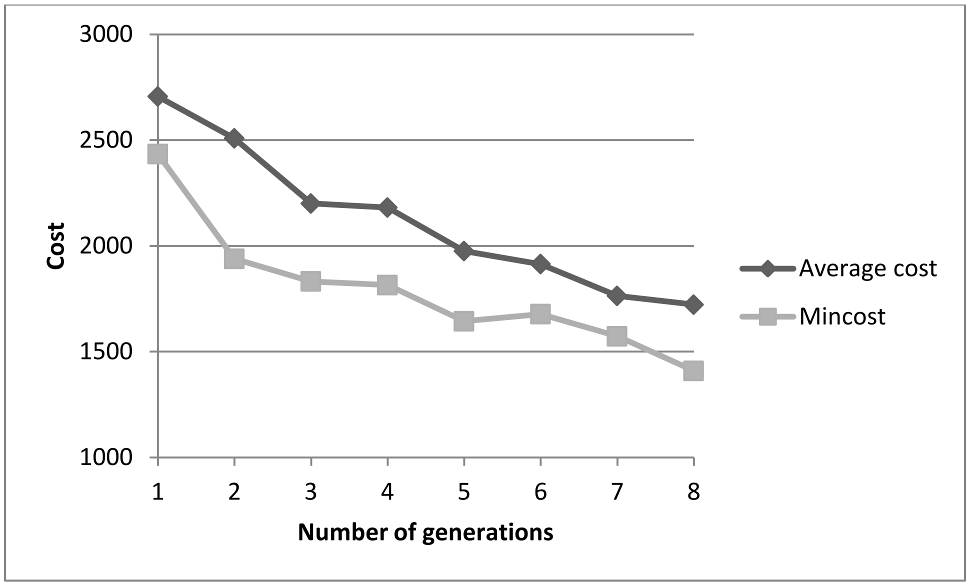


Figure 29: Graph of experimental cost values vs. number of generations for DBBO/2

5.2.2 DBBO using fuzzy controller

At the start of the first generation, each robot is assigned fifteen (15) random fuzzy controller parameters and starts to follow the wall. Each input/output has five membership functions. Figures 30, 31, and 32 show the plots of the membership functions of the best controller at the 1st generation and at the 10th generation. According to the plots, there is a change in the shape of the membership functions. Figure 32 shows that the plots of the membership function of output delta voltage are more compressed in the 10th generation compared to the 1st generation, which indicates that the range of the membership function decreased. The reason is that as the amount of fluctuation of the robot path decreases, the error and Δerror decreases, decreasing the required voltage variation, and the robots more smoothly follow the wall. The input membership functions (error and Δerror) in Figures 30 and 31 have smaller changes compared to the output membership function (voltage), which indicates that the output membership function has a greater effect on robot performance than the input membership functions. Table 12 shows the fuzzy parameters for the best robot at the 1st generation and at the 10th generation.

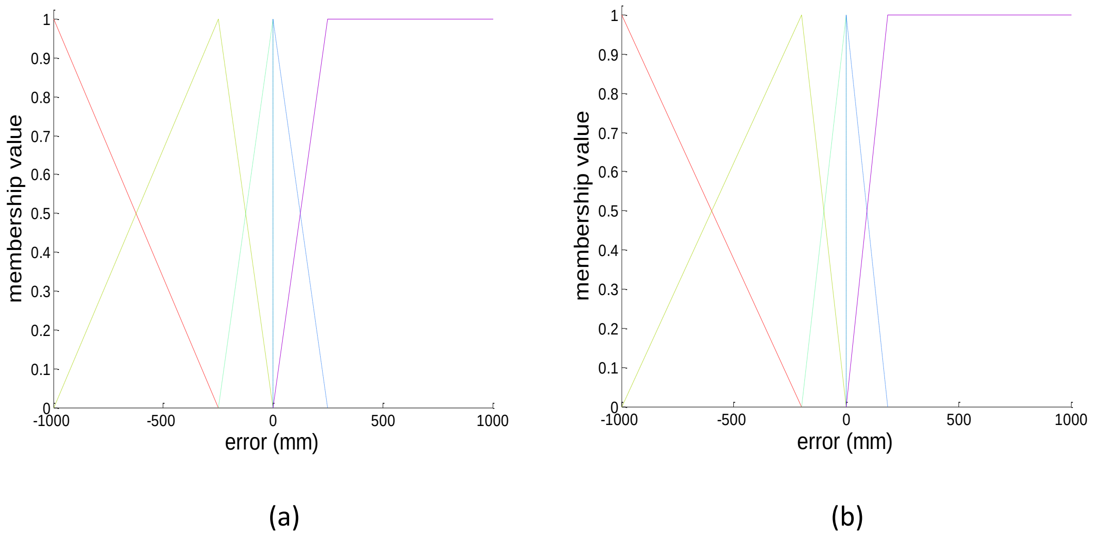
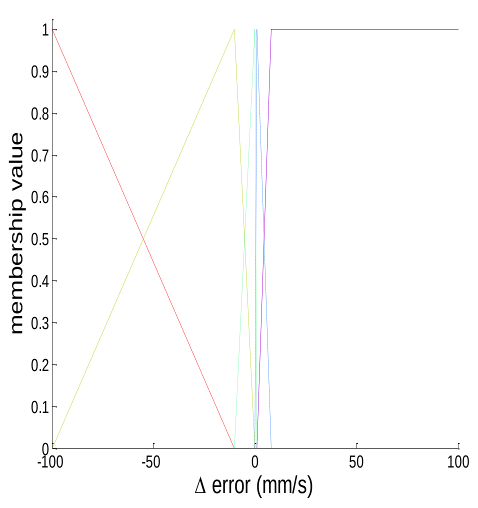
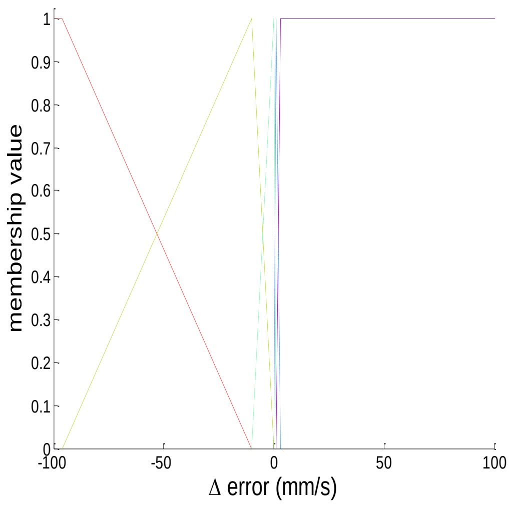


Figure 30: Fuzzy controller membership function of error (a) best DBBO/2 individual at 1st generation (b) best DBBO/2 individual at 10th generation

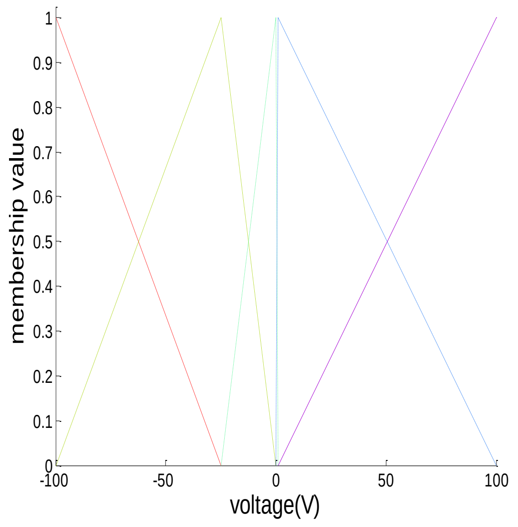


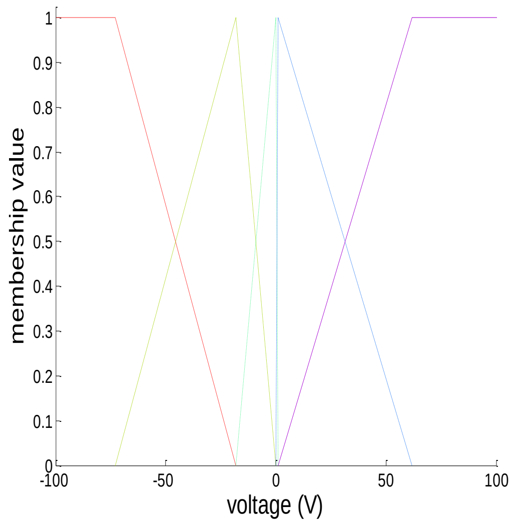


(a)

(b)

Figure 31: Fuzzy controller membership function of Δerror (a) best DBBO/2 individual at 1st generation (b) best DBBO/2 individual at 10th generation





(a)

(b)

Figure 32: Fuzzy controller membership function of output delta voltage (a) best DBBO/2 individual at 1st generation (b) best DBBO/2 individual at 10th generation

Table 12: Fuzzy parameters at 1st and 10th generations of DBBO/2

		1st generation	10th generation
Error	Large negative (LN)	–1000	–1000
	Small negative (SN)	–250	–200
	Zero (Z)	0	0
	Small positive (SP)	1	1
	Large positive (LP)	250	184
∆Error	Large negative (LN)	–100	–96
	Small negative (SN)	–10	–10
	Zero (Z)	0	0
	Small positive (SP)	1	1
	Large positive (LP)	8	3
∆Voltage	Large negative (LN)	–100	–73
	Small negative (SN)	–25	–18
	Zero (Z)	0	0
	Small positive (SP)	1	1
	Large positive (LP)	100	62

After every generation each robot exchanges its control parameters with other robots and applies the DBBO algorithm. As a result, after ten generations the fluctuation of the robots' path is decreased. The minimum cost values and the average cost values are also decreased after several generations. Figure 33 shows the minimum cost values and average cost values for each generation. So by applying the DBBO algorithm, robots improve their performance and more smoothly follow the wall.

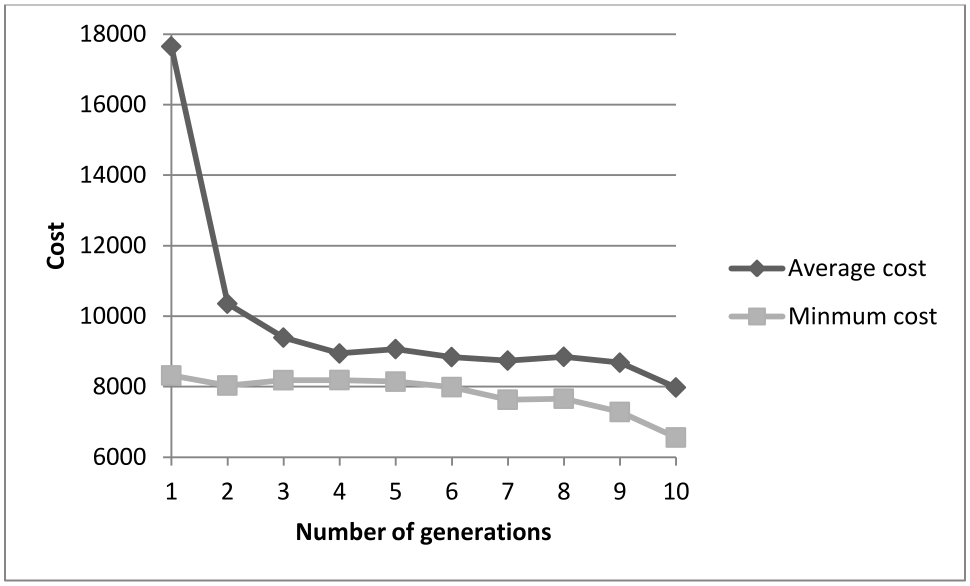


Figure 33: Graph of cost values vs. number of generations

Summary

This section summarized different experiments and their results. I have done several simulation tests by using different starting points, different mutation rates, and different wheel bases. I have also performed T-tests and robustness tests. The results show that DBBO is symmetric and robust, that there is a statistically significant difference between BBO and DBBO, and that different mutation rates do not make a significant difference in BBO and DBBO performance except for DBBO with only two interacting peers. I have also done hardware testing by using wall-following robots and optimizing their controllers. The results show that for both the PID and the fuzzy controller the cost values of the robots decrease generation by generation.

CHAPTER VI

CONCLUSION AND FUTURE WORK

6.1 Conclusion

I have introduced a distributed BBO algorithm, which is an evolutionary optimization algorithm. I have applied DBBO to a group of wall-following robots. These robots maintain a constant distance from the wall using the PID and the fuzzy controller. They communicate with one another to exchange their parameters and apply the DBBO algorithm each generation. The results show that after several generations, the tracking error of the robots decrease, and they smoothly follow the wall. Thus, DBBO removes the necessity of a centralized computing unit for optimization, and individual robots improve their performance by communicating with one another.

I have used fourteen benchmark functions and applied BBO and DBBO with different numbers of peers (2, 4, and 6). The results show that the BBO algorithm outperforms the DBBO algorithm in thirteen benchmark functions out of fourteen, but DBBO with 2 peers obtains the lowest average cost for the Schwefel 2.21 function.

I have performed simulation experiments using MATLAB. I have applied BBO and DBBO with different numbers of peers. The results show that BBO performs better than DBBO. Still, DBBO can be used for optimization because it does reduce the cost values after several generations. The main advantage of the DBBO is that it does not require a central unit to control the optimization process.

I have also used a fuzzy controller, which is based on fuzzy logic, and used BBO and DBBO algorithms to tune the fuzzy controller. The results compare with a traditional PID controller. The results show that a PID controller has a lower standard deviation compared to a fuzzy controller, while a fuzzy controller has lower minimum cost values compared to a PID controller. Fuzzy controllers can more quickly respond to system errors compared to PID controllers. The fuzzy controller is a very flexible controller. We can change the membership function parameters and if-then rules depending on our system.

I have used different starting points for the robots on both sides of the reference line and tuned their controllers accordingly. The results show that the robots whose starting points are at an equal distance from the reference line have equal cost values. These results show that the robot controllers and the DBBO algorithm are symmetric.

I have also used different mutation rates for the BBO and DBBO algorithms. The results do not show a significant difference in the cost values for the BBO and DBBO algorithms except for DBBO/2. The cost values obtained using DBBO/2 at mutation rates of 1% and 10% changed significantly. The reason is that in the DBBO/2 algorithm, only two candidates communicate at a time. So the mutation rate makes a significant difference compared to DBBO/4 and DBBO/6. Thus, as the number of peers increases, the effect of mutation rate decreases.

6.2 Future work

In future I want to continue this work by adding more robots. According to the DBBO algorithm, as the population size increases, more individuals can interact with one another. They can share their parameters and improve performance in fewer generations. I also want to increase the number of generations so that we can more clearly see the improvement due to DBBO.

In this paper I have compared DBBO with BBO. I want to compare the results of DBBO with other evolutionary algorithms, such as a genetic algorithm or other distributed algorithms. It would also be interesting to use DBBO to optimize other real-world systems such as a swarm of nuclear power reactors, peer-to-peer networking, and different constrained problems. I want to use theoretical Markov modeling and dynamic system modeling for DBBO.

I have discussed in this paper two different models of a fuzzy controller; one is Mamdani, and the second is Takagi and Sugeno model. I have used the Mamdani model of a fuzzy controller. For the next step, we can also use the Takagi and Sugeno model and compare the results of both models.

REFERENCES

[1] A. Alvarez, A. Caiti, and R. Onken, “Evolutionary Path Planning for Autonomous Underwater Vehicles in a Variable Ocean,” *IEEE Journal of Oceanic Engineering,* vol. 29, no. 2, pp. 418−429, 2004.

[2] S. Baluja, R. Sukthankar, and J. Hancock, “Prototyping Intelligent Vehicle Modules Using Evolutionary Algorithms,” *Evolutionary Algorithms in Engineering Applications,* pp. 241−258, New York, Springer, 2001.

[3] G. Box, “Evolutionary Operation: A Method for Increasing Industrial Productivity,” *Journal of the Royal Statistical Society,* vol. 6, no. 2, pp. 81−101, June 1957.

[4] H. Bremermann, “Optimization through Evolution and Recombination,” in *Self-Organizing Systems,* Washington, DC: Spartan, 1962.

[5] N. Cramer, “A Representation for the Adaptive Generation of Simple Sequential Programs,” in *Proceedings of the First International Conference on Genetic Algorithms*, pp. 183−187, 1985.

[6] C. Darwin, “The Origin of Species,” New York: Gramercy, 1995.

[7] G. Fischer, “Distributed Intelligence: Extending the Power of the Unaided, Individual Human Mind,” *Advance Visual Interfaces Conference* pp. 7−14, 2006.

[8] P. Flaming, and R. Purshouse, “Evolutionary Algorithm in Control System Engineering: A Survey,” *Control Engineering Practice,* vol. 10, pp. 1223−1241, 2002.

[9] R. Friedberg, B. Dunham, and J. H. North, “A Learning Machine: Part II,” *International Business Machines Journal,* vol. 3, no. 7, pp. 282−287, July 1959.

[10] R. Friedberg, “A Learning Machine: Part I,” *International Business Machines Journal*, vol. 2, no. 1, pp. 2–13, Jan. 1958.

[11] L. Fogel, A. Owens, M. Walsh, *Artificial intelligence through simulated evolution*, John Wiley & Sons, 1966.

[12] L. Fogel, “Autonomous Automata,” *Industrial Research*, vol. 4, no. 2, pp. 14–19, 1962.

[13] Y. Heyden, A. Nijhuis, J. Smeyers-Verbeke, B. Vandeginste, and D. Massart, “Guidance for Robustness/Ruggedness Tests in Method Validation,” *Journal of Pharmaceutical and Biomedical Analysis*, vol. 24, no. 5–6, pp. 723–753, March 2001.

[14] F. Herrera, and M. Lozano, “Adaptive Genetic Operators Based on Coevolution with Fuzzy Behaviors,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 2, pp. 149–165, 2001.

[15] G. Hill, “Algorithm 395: Students T-distribution," *Communications of the ACM*, vol. 13, pp. 617–619, October 1970.

[16] J. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[17] H. Kundra, A. Kaur, and V. Panchal, “An Integrated Approach to Biogeography Based Optimization with Case Based Reasoning for Retrieving Groundwater Possibility,” *8th Annual Asian Conference and Exhibition on Geospatial Information, Technology, and Applications*, 2009.

[18] J. Koza, F. Bennett III, D. Andre, M. Keane, and F. Dunlap, “Automated Synthesis of Analog Electrical Circuits by means of Genetic Programming,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 109–128, 2002.

[19] M. Kinoshita, T. Fukuzaki, T. Satoh, and M. Miyake, “An Automatic Operation Method for Control Rods in BWR Plants,” in *Procedure Specialists’ Meeting on In-Core Instrumentation and Reactor Core Assessment*, Cadarache, France, 1988.

[20] J. Kennedy, and R. Eberhart, *Swarm Intelligence,* Morgan Kaufmann Publishers, 2001.

[21] C. Lee, “Fuzzy Logic in Control System: Fuzzy Logic Controller—Part I”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 404–418, March 1990.

[22] P. Lozovyy, G. Thomas, and D. Simon, “Biogeography-Based Optimization for Robot Controller Tuning,” *Computational Modeling and Simulation of Intellect: Current State and Future Perspectives* (B. Igelnik, editor) IGI Global, pp. 162−181, 2011.

[23] H. Mamdani, and S. Assilian, “An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller,” *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1−13, 1975.

[24] R. MacArthur, and E. Wilson, “The Theory of Biogeography,” Princeton, NJ: Princeton Univ. Press, 1967.

[25] M. Ovreiu, and D. Simon, “Biogeography-Based Optimization of Neuro-Fuzzy System Parameters for Diagnosis of Cardiac Disease,” *Genetic and Evolutionary Computation Conference*, pp. 1235−1242, 2010.

[26] E. Parker, “Distributed Intelligence: Overview of the Field and its Application in Multi-robot Systems,” *Journal of Physical Agents*, vol. 2, pp. 5−14, 2008.

[27] V. Panchal, P. Singh, N. Kaur, and H. Kundra, “Biogeography-Based Satellite Image Classification,” *International Journal of Computer Science and Information Security*, vol. 6, pp. 269‒274, 2009.

[28] R. Rarick, D. Simon, F. Villaseca, and B. Vyakaranam, “Biogeography-Based Optimization and the Solution of the Power Flow Problem,” *IEEE Conference on Systems, Man, and Cybernetics*, pp. 1003‒1008, 2009.

[29] P. Roy, S. Ghoshal, and S. Thakur, “Biogeography-Based Optimization for Economic Load Dispatch Problems,” *Electric Power Components and Systems*, vol. 38, pp. 166‒181, 2010.

[30] I. Rechenberg, “Cybernetic Solution Path of an Experimental Problem,” *Library Translation*, vol. 1122, 1964.

[31] L. Rodr´ıguez, R. Garc´ıa, A. Campan˜a, and J. Sendra, “A new Approach to a Complete Robustness Test of Experimental Nominal Conditions of Chemical Testing Procedures for Internal Analytical Quality Assessment,” *Chemometrics and Intelligent Laboratory Systems*, vol. 41, no.1, pp. 57‒68, July 1998.

[32] C. Scheidegger, A. Shah, and D. Simon, “Distributed Learning with Biogeography-Based Optimization,” *Industrial, Engineering and Other Applications of Applied Intelligent Systems Conference,* Syracuse, New York, pp. 203‒215, 2011.

[33] D. Simon, ”Biogeography-Based Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702–713, 2008.

[34] M. Sugeno, and T. Yasukawa, “A Fuzzy-logic-Based Approach to Quantitative Modeling,” *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 1, 1993.

[35] V. Savsani, R. Rao, and D. Vakharia, “Discrete Optimization of a Gear Train using Biogeography-Based Optimization Technique,” *International Journal of Design Engineering*, vol. 2, pp. 205–223, 2009.

[36] D. Simon, M. Ergezer, D. Du, and R. Rarick, “Markov Models for Biogeography-Based Optimization,” *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 41, pp. 299–306, 2011.

[37] D. Simon, M. Ergezer, and D. Du, “Population Distributions in Biogeography-Based Optimization Algorithms with Elitism,” *IEEE Conference on Systems, Man, and Cybernetics*, pp. 1017–1022, 2009.

[38] D. Simon, “A Dynamic System Model of Biogeography-Based Optimization”, *Applied Soft Computing*, vol.11, no. 8, pp. 5652–5661, December 2011.

[39] U. Singh, H. Singla, and T. Kamal, “Design of Yagi-Uda Antenna using Biogeography Based Optimization,” *IEEE Transactions on Antennas and Propagation*, vol. 58, no. 10, pp. 3375–3379, 2010.

[40] H. Schwefel, “Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stromungstechnik,” Master’s thesis, 1965.

[41] M. Sugeno, and T. Yasukawa, “A Fuzzylogic Based Approach to Qualitative Modeling,” *IEEE Transactions on Fuzzy Systems,* vol. 1, pp. 7–29, 1993.

[42] T. Takagi, and M. Sugeno, “Fuzzy Identification of Systems and its Application to Modeling and Control,” *IEEE Transactions on Systems, Man and Cybernetics* vol. 15, pp. 116–132, 1985.

[43] M. Togai, and S. Chiu, “A Fuzzy Accelerator and a Programming Environment for Real-time Fuzzy Control,” in *Interoperability for Enterprise Software and Applications Conference*, pp. 147–151, Tokyo, Japan, 1987.

[44] K. Valavanis, G. Saridis, “Intelligent Robotic Systems: Theory, Design and Application,” *Kluwer Acadamic*, Boston, 1992.

[45] O. Yagishita, O. Itoh, and M. Sugeno, “Application of Fuzzy Reasoning to the Water Purification Process,” in *Industrial Application of Fuzzy Control*, M. Sugeno, Ed. Amsterdam: North-Holland, pp. 19–40, 1985.

[46] X. Yao, Y. Liu, and G. Lin, “Evolutionary Programming made Faster,” *IEEE Transactions Evolutionary Computation*, vol. 3, pp. 82–102, July 1999.

[47] L. Zadeh, “Fuzzy Sets,” *Information and Control*, vol. 8, no. 3, pp. 338–353, June 1965.

APPENDICES

APPENDIX A

Robustness test results

This section gives detailed results of robustness test as I discussed in Chapter V. I have used the robot wheel base length as the independent variable. Table 12(a) shows the average cost values of robots for different wheel bases, where first column gives the wheel base for which the robot is trained, and the row gives the cost values of the trained robot when it has a different wheel base. Table 12(b) shows the percentage change of the cost value. Similarly, Table 13, Table 14, and Table 15 show the values when using DBBO/2, DBBO/4, and DBBO/6. The values indicate that the percentage change in the cost values of the robots having different wheel bases is not more that 10%, which indicates that the change in robot wheel base does not strongly affect its cost value. Therefore, the BBO and the DBBO algorithms are robust with respect to the robots' wheel base length.

Table 12 (a): Average cost values of robots with fuzzy controller using BBO for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	5.58	5.59	5.61	5.70	5.79	5.88
180	5.62	5.58	5.62	5.75	5.88	6.05
185	5.63	5.67	5.71	5.78	5.83	5.88
190	5.47	5.50	5.57	5.61	5.69	5.74
195	5.53	5.58	5.62	5.64	5.72	5.75
200	5.58	5.58	5.63	5.64	5.66	5.72

Table 12 (b): Percentage difference of cost value of robots with fuzzy controller using BBO for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	0	0.25	0.56	2.19	3.74	5.47
180	0.77	0	0.69	3.09	5.26	8.40
185	–1.27	–0.59	0	1.27	2.15	3.12
190	–2.49	–2.00	–0.80	0	1.37	2.29
195	–3.37	–2.42	–1.81	–1.50	0	0.56
200	–2.39	–2.33	–1.45	–1.26	–0.91	0

Table 13 (a): Average cost values of robot with fuzzy controller using DBBO/2 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	5.89	5.94	5.97	6.00	6.06	6.09
180	5.98	6.05	6.09	6.14	6.21	6.26
185	5.53	5.58	5.65	5.77	5.85	5.91
190	5.51	5.61	5.66	5.72	5.82	5.87
195	5.57	5.68	5.73	5.87	5.90	5.98
200	5.46	5.38	5.42	5.43	5.52	5.59

Table 13 (b): Percentage difference of cost value of robots with fuzzy controller using DBBO/2 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	0	0.89	1.29	1.80	2.78	3.34
180	–1.10	0	0.69	1.44	2.74	3.53
185	–2.16	–1.18	0	2.04	3.48	4.66
190	–3.66	–1.89	–1.08	0	1.77	2.67
195	–5.62	–3.71	–2.95	–0.54	0	1.32
200	–2.39	–3.70	–3.14	–2.85	–1.29	0

Table 14 (a): Average cost values of robot with fuzzy controller using DBBO/4 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	5.94	6.07	6.16	6.24	6.34	6.49
180	5.65	5.70	5.80	5.87	5.97	6.05
185	6.15	6.17	6.12	6.14	6.11	6.11
190	5.64	5.78	5.89	6.02	6.17	6.32
195	6.16	6.16	6.17	6.18	6.22	6.23
200	6.78	6.77	6.77	6.78	6.79	6.81

Table 14 (b): Percentage difference of cost value of robots with fuzzy controller using DBBO/4 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	0	2.18	3.59	5.03	6.64	9.23
180	–0.98	0	1.67	2.91	4.70	5.97
185	0.47	0.73	0	0.30	–0.16	–0.12
190	–6.40	–3.97	–2.24	0	2.45	4.91
195	–0.96	–0.93	–0.78	–0.59	0	0.25
200	–0.45	–0.59	–0.47	–0.44	–0.27	0

Table 15 (a): Average cost values of robot with fuzzy controller using DBBO/6 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	6.47	6.51	6.57	6.61	6.67	6.70
180	6.67	6.76	6.79	6.87	6.92	7.12
185	6.28	6.33	6.40	6.49	6.56	6.65
190	6.94	7.08	7.21	7.36	7.54	7.70
195	7.16	7.29	7.38	7.53	7.61	7.76
200	7.04	7.17	7.37	7.51	7.71	7.85

Table 15 (b): Percentage difference of cost value of robots with fuzzy controller using DBBO/6 for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	0	0.52	1.57	2.09	3.14	3.60
180	–1.32	0	0.48	1.69	2.41	5.41
185	–1.92	–1.13	0	1.42	2.52	3.95
190	–5.63	–3.85	–1.97	0	2.46	4.63
195	–5.95	–4.15	–2.99	–1.03	0	1.94
200	–10.34	–8.61	–6.11	–4.31	–1.80	0

Appendix B

T-test results

The T-test method was invented by William Sealy Gosset in 1908 [20]. This method is also known as the Student's T-test and is used to determine if there is a statistically significant difference between the results of two sets of experiments. According to this method we first calculate X_1 and X_2 as follows:

$$X_1 = \frac{1}{N_1}\sum_{i=1}^{N_1} x_{1i}$$

$$X_2 = \frac{1}{N_2}\sum_{i=1}^{N_2} x_{2i} \tag{20}$$

where X_1 is the average value of the group 1 experimental results, N_1 is the number of values in group 1, x_{1i} is the *i*-th data point of group 1, X_2 is the average value of the group 2 experimental results, N_2 is the number of values in group 2, and x_{2i} is the *j*-th data point of group 2. Then we calculate the standard deviations S_1 and S_2, as follows.

$$S_1 = \sqrt{\frac{\sum_{i=1}^{N_1}(x_{1i} - X_1)^2}{N_1}}$$

$$S_2 = \sqrt{\frac{\sum_{i=1}^{N_2}(x_{2i} - X_2)^2}{N_2}} \tag{21}$$

Now calculate S_t as shown below.

$$S_t = \sqrt{S_1^2 + S_2^2} \tag{22}$$

Finally the T-test value is calculated as follows:

$$\text{T-test value} = \frac{(X_1 - X_2)}{S_t} \tag{23}$$

Now we calculate the degree of freedom as

$$\text{Degree of freedom} = N_1 + N_2 - 2 \tag{24}$$

By using the degree of freedom and the T-test value we find the probability according to [15] that the difference between two sets of results are due solely to random variation. If this probability is more than 0.05 we conclude that there is not a statistically significant difference between the two groups, and if the probability is less than 0.05 we conclude that there is a statistically significant difference between the two groups. I have done the T-test for robustness test results and the resultss are shown in Table 5, 6, 7, and 8. The values in the tables are very low, which indicates that the difference between different wheel bases is not due to random fluctuation. The difference is rather due to the wheel base length.

I have done the T-test on the results of the robustness tests as I mentioned in Chapter V. Table 16, 17, 18, and 19 show the T-test results. The numbers in the tables show the probabilities that two sets of results came from the same probability distribution. This is another way of saying that the differences between two sets of numbers are simply due to random variations rather than systematic differences between the underlying experiments. The values in the table are very small which indicates that there is a statistically significant difference between robots having different wheel bases, and this change is not due to random variation.

Table 16: T-test results for BBO (probabilities) for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	1	1.01E–11	6.66E–13	2.37E–27	2.82E–31	5.46E–39
180	1.19E–15	1	5.61E–19	1.13E–28	1.94E–32	5.84E–33
185	6.53E–21	8.90E–15	1	8.41E–23	2.42E–23	1.05E–21
190	7.01E–33	4.90E–28	3.43E–24	1	3.40E–27	1.50E–29
195	2.20E–16	1.28E–16	8.38E–19	1.05E–12	1	2.24E–12
200	4.11E–24	1.21E–23	3.69E–17	7.47E–18	2.70E–14	1

Table 17: T-test results for DBBO/2 (probabilities) for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	1	5.82E–30	6.41E–27	4.36E–36	1.93E–31	5.87E–41
180	3.76E–29	1	8.07E–26	6.14E–32	5.59E–35	1.94E–37
185	3.87E–26	1.79E–22	1	9.51E–27	1.47E–27	3.07E–28
190	3.75E–35	2.24E–30	3.71E–26	1	1.67E–25	7.88E–33
195	3.93E–29	3.94E–24	3.10E–22	4.87E–10	1	9.42E–19
200	2.68E–22	3.92E–18	1.42E–21	1.49E–19	2.96E–18	1

Table 18: T-test results for DBBO/4 (probabilities) for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	1	9.09E–36	1.20E–35	4.81E–29	1.89E–44	1.14E–43
180	5.30E–23	1	1.54E–30	1.50E–38	6.78E–25	1.34E–44
185	7.04E–07	2.51E–08	1	8.06E–06	5.37E–03	2.45E–02
190	2.29E–40	3.72E–25	1.24E–31	1	7.98E–29	1.98E–34
195	5.89E–23	5.76E–22	1.00E–21	1.66E–18	1	6.60E–10
200	1.78E–21	1.06E–21	2.21E–21	1.01E–20	2.31E–16	1

Table 19: T-test results for DBBO/6 (probabilities) for wheel base lengths 175–200 mm. The rows show trained wheel base lengths and the columns show experimental wheel base lengths

	175	180	185	190	195	200
175	1	1.15E–14	1.03E–25	3.95E–26	4.94E–29	1.74E–28
180	4.42E–20	1	1.06E–23	2.83E–26	1.18E–28	1.33E–31
185	3.86E–23	3.09E–07	1	1.85E–20	2.31E–25	2.04E–25
190	8.08E–34	2.11E–35	2.01E–26	1	7.00E–33	1.12E–37
195	1.43E–36	8.78E–34	4.53E–24	1.53E–23	1	1.77E–27
200	1.05E–41	4.39E–38	2.98E–37	2.22E–32	1.36E–27	1

Appendix C

Benchmark functions

I have used fourteen benchmark functions to compare the performance of the BBO and DBBO having different numbers of peers (2, 4, and 6) as I mentioned in Chapter II. Here I have given the details of those fourteen benchmark functions [46].

1. Ackley

- Number of variables : n
- Definition : $f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{-\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$
- Search domain : $-30 < x_i < 30, i \in [1, n]$
- Global minimum : $f(x) = 0$

2. Fletcher

- Number of variables : n
- $U[c, d]$ = random number uniformly distributed on the domain $[c, d]$
- Random numbers : a_{ij}, b_{ij}
- Definition :

$a_{ij} \sim U[-100,100], i, j \in [1, n]$

$b_{ij} \sim U[-100,100]$

$\alpha_{ij} \sim U[-\pi, \pi]$

$A_i = \sum_{j=1}^{n}\left(a_{ij}\sin(\alpha_j) + b_{ij}\cos(\alpha_j)\right)$

$B_i = \sum_{j=1}^{n}(a_{ij}\sin(x_j) + b_{ij}\cos(x_j))$

- Search domain : $-\pi < x_i < \pi, i \in [1, n]$
- Global minimum : $f(x) = 0$

3. **Griewank**

- Number of variables : n
- Definition : $f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
- Search domain : $-600 < x_i < 600, i \in [1, n]$
- Global minimum : $f(x) = 0$

4. **Penalty #1**

- Number of variables : n
- Definition :

$$u_i = \begin{Bmatrix} 0, x_i \in [-10,10] \\ 100(|x_i| - 10)^4, \quad \text{otherwise} \end{Bmatrix}$$

$$y_i = 1 + \frac{(x_i + 1)}{4}$$

$$f(x) = \sum_{i=1}^{n} u_i + \frac{10(sin^2(\pi y_1) + (y_n - 1))\pi}{30} + \sum_{i=1}^{n-1} \frac{(y_i - 1)^2(1 + 10sin^2(\pi y_{i-1}))\pi}{30}$$

- Search domain : $-50 < x_i < 50, i \in [1, n]$
- Global minimum : $f(x) = 0$

5. **Penalty #2**

- Number of variables : n
- Definition :

$$u_i = \begin{Bmatrix} 0, x_i \in [-10,10] \\ 100(|x_i| - 5)^4, \quad \text{otherwise} \end{Bmatrix}$$

$f(x) =$

$\sum_{i=1}^{n} u_i + 0.1\left(sin^2(3\pi x_1)(x_n - 1)^2\left(1 + sin^2(2\pi x_n)\right)\right) + \sum_{i=1}^{n-1} 0.1(x_i -$

$1)^2\left(1 + sin^2(3\pi x_{i+1})\right)$

- Search domain : $-50 < x_i < 50, i \in [1, n]$
- Global minimum : $f(x) = 0$

6. Quartic

- Number of variables : *n*
- Definition : $f(x) = \sum_{i=1}^{n} ix_i^4$
- Search domain : $-1.28 < x_i < 1.28, i \in [1, n]$
- Global minimum : $f(x) = 0$

7. Rastrigin

- Number of variables : *n*
- Definition : $f(x) = 10n + \sum_{i=1}^{n}({x_i}^2 - 10\cos(2\pi x_i))$
- Search domain : $-5.12 < x_i < 5.12, i \in [1, n]$
- Global minimum : $f(x) = 0$

8. Rosenbrock

- Number of variables : *n*
- Definition : $f(x) = \sum_{i=1}^{n-1} 100(x_i - x_{i+1}^2)^2 + (1 - x_i)^2$
- Search domain : $-2.048 < x_i < 2.048, i \in [1, n]$
- Global minimum : $f(x) = 0$

9. Schefel 1.2

- Number of variables : *n*

- Definition : $f(x) = \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$
- Search domain : $-65.536 < x_i < 65.536, i \in [1, n]$
- Global minimum : $f(x) = 0$

10. Schefel 2.21

- Number of variables : n
- Definition : $f(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$
- Search domain : $-100 < x_i < 100, i \in [1, n]$
- Global minimum : $f(x) = 0$

11. Schefel 2.22

- Number of variables : n
- Definition : $f(x) = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i|$
- Search domain : $-10 < x_i < 10, i \in [1, n]$
- Global minimum : $f(x) = 0$

12. Schefel 2.26

- Number of variables : n
- Definition : $f(x) = \max(|x_i|)$
- Search domain : $-512 < x_i < 512, i \in [1, n]$
- Global minimum : $f(x) = 0$

13. Sphere

- Number of variables : n
- Definition : $f(x) = \sum_{i=1}^{n} x_i^2$
- Search domain : $-5.12 < x_i < 5.12, i \in [1, n]$
- Global minimum : $f(x) = 0$

14. Step

- Number of variables : n
- Definition : $f(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$
- Search domain : $-200 < x_i < 200, i \in [1, n]$
- Global minimum : $f(x) = 0$