Mobile Robot Navigation Control in Moving Obstacle Environment using Genetic Algorithm and Artificial Neural Networks

Dr. Anupam Shukla *Dr. Ritu tiwari* *Rahul Kala*
dranupamshukla@gmail.com *rt_twr@yahoo.co.in* *rahulkalaiiitm@yahoo.com*
(Department of Information Technology)
(Indian Institute of Information Technology and Management, Gwalior)

Citation: A. Shukla, R. Tiwari, R. Kala (2009) Mobile Robot Navigation Control in Moving Obstacle Environment using Genetic Algorithms and Artificial Neural Networks, *International Journal of Artificial Intelligence and Computational Research* 1(1), 1-12.

Abstract

With the growth of technology and machines doing most of the work which used to be done by humans, the need of robots doing most of the work in future is evident. Especially it is important for many robots to synchronize between themselves and other machines. One of the most basic step in this regard is to move a robot safely from one position to the other. The problem gets even more complex, considering the fact that the environment is dynamically changing. An example of such an application may be the use of robots in industry to carry tools and other materials from one place to other. In this paper we find out an efficient path of the robot using Genetic Algorithms and Artificial Neural Networks at each instant of time. The algorithm ensures that under any circumstance, there would not be any collision of the robot with any of the dynamically changing obstacles. The mobile robot navigation control has huge industrial application. It may be used by the industry to send robots for surveys, data acquisition, doing specific work etc. The collision free movement of robot in a moving obstacle environment can be used to move robot in a world of robots. Hence it takes us closer to a fully robot control production/service system, where robots do all the work without external help. The algorithm works on a grid of definite size with known positions of obstacles. This grid may be formed by scanning the surroundings. We assume that the robot can make a limited number of moves, restricted to moving forward a unit step or turning (clockwise or anti-clockwise) a unit direction. The algorithms calculate the most efficient next move. When this algorithm was simulated, we saw the robot traveling without collision and reaching the destination. The algorithms guided the robot in a very efficient path. This was true even when the robots were placed in a highly chaotic environment.

Key Words
Robotic simulation, robotic navigation control, moving obstacle problem, Genetic algorithm, Artificial Neural Network, Back propagation

1. Introduction

Consider a situation where many robots/moving obstacles are together in a place, moving constantly just like humans move in a market. The problem is to make your robot move from the starting position to the final position [1][20]. We need to optimize the path it travels. We also need to ensure that the robot does not collide with any of the other obstacles. We need to move the robot using this navigation plan. The problem is that we have no initial idea as to what the condition of the map would be at any instant of time, as the things are changing dynamically with time.

The paper proposes the use of Genetic Algorithm and Artificial Neural Network to find out the most optimal path of the robot at every instant of time [2]. The robot tries to find a path that would most optimally take it to the goal position. Based on these results a single unit move is made. At the next unit of time again the algorithm is run to calculate the next move. Hence at every instant of time the algorithm runs and gives the next move of the robot. The robot physically moves according to these results and the move is completed. It is not ensured that every move will

make the robot closer to the goal. But at the end, if there is a path possible to reach the goal at any instant of time, the robot reaches the goal. In case it is not possible to reach the goal at all, the robot becomes stationary.

Genetic Algorithm is the first algorithm implemented. This algorithm works by the principle of trying to find the best solution by the combination of the existing solutions. At any time the available solutions may be regarded as chromosomes. This algorithm combines any two non-optimal solutions to generate a set of new solutions. Like this by continuously cross-breeding of various solutions, we may find a large number of solutions. The algorithm uses a fitness function to measure the condition of various chromosomes. The fitness function measures the effectiveness of the solution. We select the best solutions, in general, so as to get better solutions at each step. The genetic algorithm optimizes the total path travelled by the robot and makes all the movements in the same manner.

The second algorithm we have implemented is **Artificial Neural Networks**. This algorithm believes in learning from the past results. The neural network is set up. This is a special kind of network that takes the input conditions and generates the output, which is the next optimal move. The neural network consists of various neurons which are parallel processing units. Each neuron has a weight associated with it. The action of these neurons on the inputs, using various hierarchies makes it possible for the neural network to generate the output. The neural network is associated with a training phase where sample inputs are provided, whose outputs are known. Using the training phase the necessary adjustments in weights are done so that the network is ready for the test phase. In test phase the input is given and output is implemented.

The elementary model of cognition [5] includes three main cycles. Among these, the 'sensing-action' cycle is most common for mobile robots. This cycle inputs the location of the obstacles and subsequently generates the control commands for the motors to set them in motion. The second cycle passes through perception and planning states of cognition, while the third includes all possible states including sensing, acquisition, perception, planning and action [6]. Sensing here is done by ultrasonic sensors / camera or by both. Through this the robot can come to know about each and every obstacle in its vicinity. There are many algorithms for construction of the robot's world map [7].

The term Planning of Navigation [8] refers to the generation of sequences of action in order to reach a given goal state from a predefined starting state.

We consider that at any given point of time the robot is standing or moving at some position on the board. The other robots/obstacles are also positioned somewhere on the board. The robot starts scanning its environment. Using standard algorithms it forms a form of a grid, where the positions of all obstacles are known. Now the robot uses Genetic algorithm or Artificial Neural Network to get the most optimal next move. The robot uses this output to guide its motors. These motors move the robot to the next move as desired by the algorithm. Now after this move is made, the procedure repeats itself.

Section 1 is the introduction, Section 2 talks about the motivation to the problem. In Section 3 we will discuss about the simulation model which includes modeling of the problem and the algorithms. We present both the algorithms as two cases. Case I is for Genetic Algorithm, Case II is Artificial Neural Network. Then we compaire the two algorithms. In Section 4 we talk about the testing of algorithms. Section 5 is for Results. In Section 6 we give the conclusion.

2. Motivation

The problem of robot navigation control, due to its applicability, is of a great interest. We have already seen good research in various modules. A lot of work exists to model the entire problem [1]-[7]. There exist good algorithms to scan the environment and represent all the obstacles in form of a grid [3]. Also various algorithms have been proposed to plan the movement of the robot using various conditions.

The whole problem till now has been seen under separate heads of planning navigation control of static environment and planning navigation control of dynamic environment. If we come to static environment, many algorithms have been implemented and results verified [8][10][11][19]. In planning dynamic environment the steps are a little different, as the environment continuously changes.

We also have various works of research in which people have tried to solve the navigation problem using genetic algorithm [8][10][11][16]. The basic principles in all these have been to take a fixed solution length and find the solutions by using genetic operators. In this paper we propose a graphical node representation which will work for all sorts of highly chaotic conditions where the number of obstacles is very large.

Also similar work exists in neural network [6][11][15]. Here neural network has been applied mainly on static data. Here we have adopted a representation that minimizes the errors that might come due to neural noise or incomplete database.

3 Simulation Model

In this section we will discuss the way we model the whole problem and also the way we would be applying all the algorithms in this problem. The whole modeling includes modeling of the robot and environment, modeling of the algorithms and modeling of the results, so that they may be implemented.

3.1 General Assumptions

In order to visualize and implement the solution, we have made the following generalizations/assumptions[17]. Various models may be proposed to model the robot[12]. These have been made considering the practical implementation of the proposed solution.

- The entire space where the robot can move is a finite space. This space is divided in a grid of size M X N[3][13]. This space is finite and can be simulated using Genetic Algorithms or Artificial Neural Networks
- Each obstacle as well as robot can make only a unit move in a unit time called the threshold time (η) (see robotic design and assumptions).

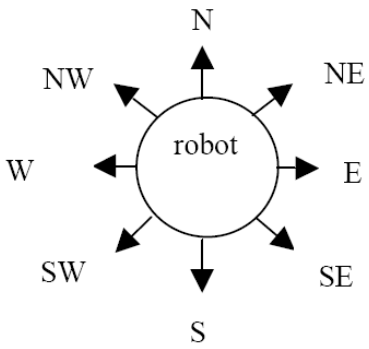
3.2 Robotic Design and Assumptions

The robot we consider here consists of two wheels. Hence it can travel forward (both wheels rotating in same direction) or rotate clockwise or anticlockwise (both wheels rotating in opposite direction) [4]. For this problem we take the following movements of the robots as valid that can be performed in a minimum threshold time. The movements have been quantized for algorithmic purposes:

- Move forward (unit step)
- Move at an angle of 45 degrees forward (unit step) from the current direction
- Move clockwise/anti-clockwise (45 degrees)
- Move clockwise/anti-clockwise (90 degrees)

It is assumed that the robot only rotates/moves in angles of a multiple of 45 degrees. Hence we can only move the robot in the directions (by rotations and forward move) North, North-East, North-West, South, South-East, South-West as given in Figure 1.

Figure 1: The Various directions the robot can move in



For algorithmic purposes the directions have been denoted as given in Table 1:

Table 1: The representation of various directions

N	0
NE	1
E	2
SE	3
S	4
SW	5
W	6
NW	7

Hence if the robot is at direction 5, it has the following possible moves:

- No motion
- Change direction to 4,3,6 or 7
- Move forward in direction 5
- Move forward in direction 6 or 4

In this paper we assume that the information of every obstacle is available and is being constantly updated in the threshold time (η) which is the unit time assumed by the algorithm.

3.3 Algorithms

We have used two different algorithms for the problem. In the subsequent sections we will discuss each one of them in two separate cases. Case I would be Genetic Algorithm. Case II would be devoted to neural networks.

3.3.1 CASE I: Genetic Algorithm

The algorithm used is the conventional Genetic algorithm to find out the goal (final position) starting from the initial position. Genetic algorithm[14][16] applied by us in this problem differs from the other genetic algorithms taking the fact that we have considered a graphical node as a chromosome, in place of taking a predetermined length bit sequence[8][10][11]. This is because we cannot predict the minimum/maximum number of moves the robot takes, or the way in which it travels. In this algorithm we use the following specifications for the functioning of the genetic algorithm.

3.3.1.1 Representation

The entire grid of MXN size is available. We know that the problem is to get the final path from the source (initial point) to the destination (final point). This consists of a series of points on the grid. Each point has three attributes associated with it. These are the x coordinate, y coordinate and the direction in which the robot is facing. Any point can thus be represented by

(x,y,d)

where x=x coordinate

y=y coordinate

d=direction in which robot is facing (between 0 to 7)

The final solution generated is in the form

$(x_1,y_1,d_1)(x_2,y_2,d_2)(x_3,y_3,d_3)\ldots\ldots\ldots\ldots\ldots\ldots(x_n,y_n,d_n)$

Each solution is stored in the solution set. We have considered three types of solution:

- **Full**: They are complete solution and fully connect source to destination
- **Left**: The start from the source but are unable to reach to the destination
- **Right**: They do not start from the source but reach the destination

For the chromosomal representation of the problem, we take each point (x,y,d) as a node on a graph. Each node can be connected to other nodes if the transition from the first node to the other node is a valid move. i.e. (0,0,0) can be

connected to (0,0,3) as we can turn from direction 0 to 3. But (0,0,0) cannot be connected to (1,1,0), as the move is not possible.

3.3.1.2 Evaluation of fitness
We measure the fitness of the chromosomes by the evaluation function. The lesser the value of the evaluation function, the more fit is the chromosome. The formula of the fitness function is given by the following

Full Solution: No of steps needed to traverse from source to destination.

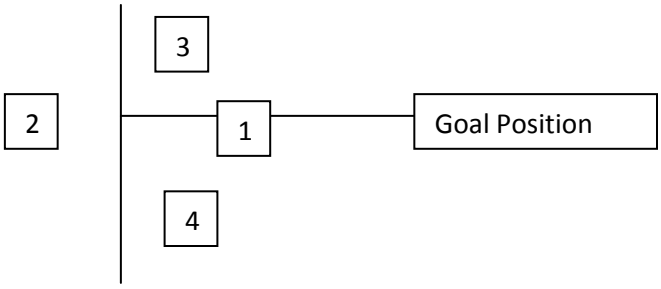
Left Solution: Number of steps needed to traverse from source to the last point traversed + (Square of the distance of this point to the final destination + R(n))

Right Solution: (Square of the distance of source point to the first point traversed + R(n)) + Number of steps needed to traverse from this point to the final solution

Here R(n) = minimum time required for the robot to rotate in its entire journey assuming no obstacles

If the direction is 2 (East) and final position is just in front, we have R(n) as the following (Refer Figure 2)
Region 1: 0
Region 2: 2
Region 3: 1
Region 4: 1

Figure 2: The values of R(n) at various places



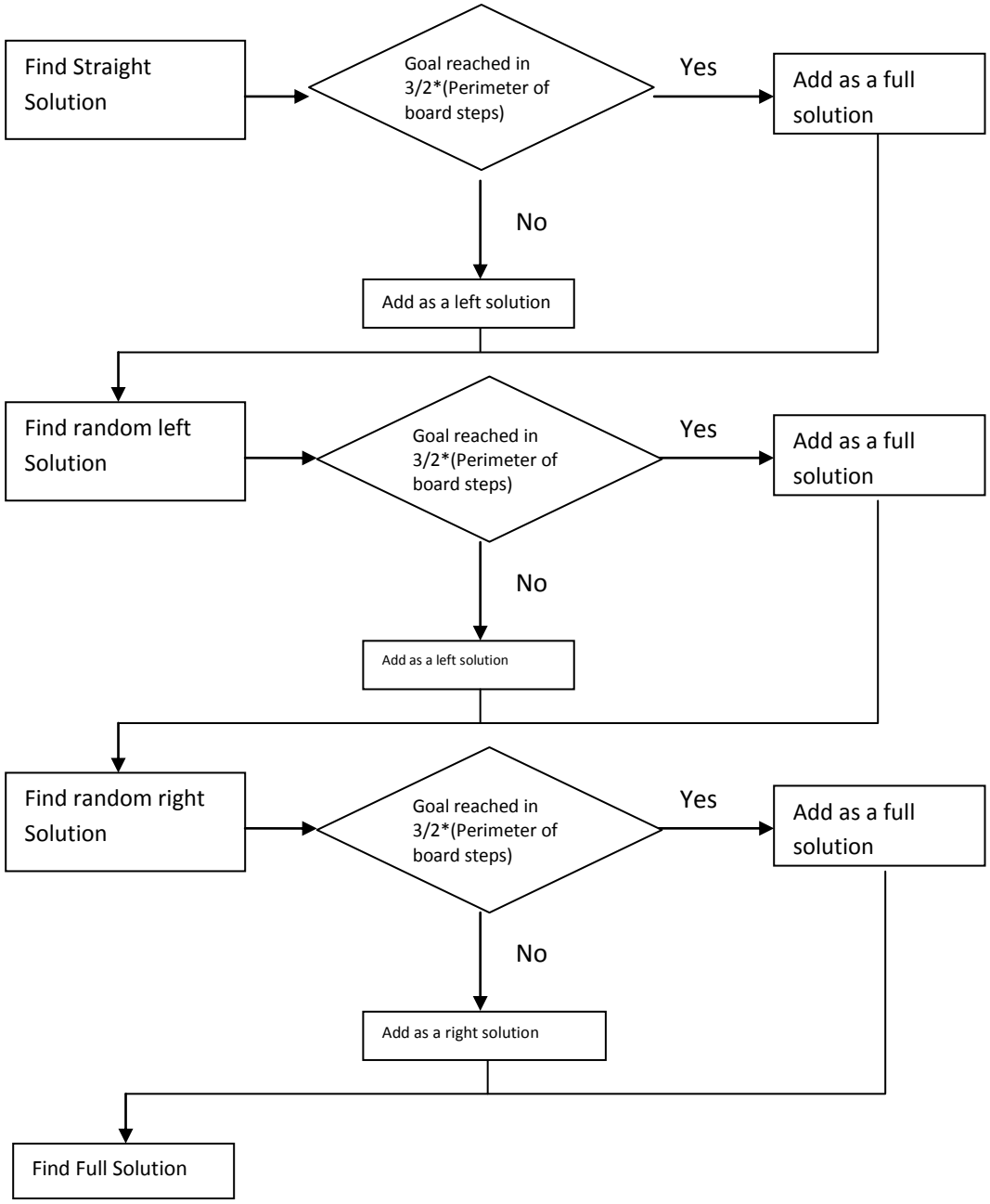
The R(n) term ensures that the number of rotations are minimum. As in the practical scenario, we need to avoid the rotations as much as possible. The lesser the number of rotations, the smoother the travel.

Hence the closer a point to the destination and traverses this in lesser steps, lesser will be the value of the function and better will be the solution

3.3.1.3 Initial Solution
The algorithm starts by trying to find an initial solution. The layout of the initial solution is explained in figure 3. The initial solution is found out by applying the following strategies one

- Find a straight path solution between source and destination. From the source march straight ahead towards the destination. Whenever travelling at any point, there is a possibility of collision, take *randomMoves* (number of random steps) (turning at random directions in between). Now repeat this process of marching towards the destination.

Figure 3: The Steps for generating initial solution for the robot



FindStraightSolution(currentPoint,destinationPoint)

Step1: for i ← 1 to noIterationsInitial
Begin
Step2: for j ← 1 to (3/2)*perimeter of board (Maximum moves per solution being tried)
Begin
Step3: if p ← destinationPoint
Step4: add p to current solution set and break
Step5: find the move m in moves which makes the robot closest to destination

Step6: if point p we reach after m is already in the current solution set then delete earlier points and add this point.
Step7: if no move takes the robot closer to target
Step 8: make randomMoves number of random moves in any direction
Step9: for each move m in these moves, if point p we reach after m is already in the current solution set then delete earlier points and add this point.
Step10: if destinationPoint is reached then add this solution set to fullSolution
Step11: else add this solution set to leftSolution

- Find random left solution between source and destination. At any point take random moves at any direction. Repeat doing this. Use direction = left in the below algorithm.

- Find random right solution between source and destination. At any point take random moves from the destination at any direction. Add every solution at the head of the final solutions sequence; so that it looks we approached the destination from a point. Repeat doing this. Use direction = right in the below algorithm.

FindRandomSolution(currentPoint,destinationPoint,direction)
Step1: p ← currentPoint
Step2: for i ← 1 to noIterationsInitial
Begin
Step3: for j ← 1 to (3/2)*perimeter of board
Begin
Step4: if p ← destinationPoint
Step5: add p to current solution set and break
Step6: moves ← make a random move
Step7: if point p we reach after move m is already in the current solution set then
Step8: delete all points from the previous point p to end
Step9: add p to the current solution set
Step10: if destinationPoint is reached then add this solution set to fullSolution
Step11: else add it to left or right solution, whichever applicable

- Find Full Solutions: Once we have sufficient entries for the left and right solution, there are chances of getting more full solutions from these data. Search all the left and right solutions obtained so far. If you find any point common between any solution of the left and right solutions, mix them to get a full solution. Hence if any of the left solution is
$(x_{11},y_{11},d_{11})(x_{12},y_{12},d_{12})$…….....$(x_i,y_i,d_i)$ $(x_{1i+1},y_{1i+1},d_{1i+1})$……...$(x_{1n},y_{1n},d_{1n})$

  and right solution is
$(x_{21},y_{21},d_{21})(x_{22},y_{22},d_{22})$……$(x_i,y_i,d_i)$ $(x_{2i+1},y_{2i+1},d_{2i+1})$……$(x_{2n},y_{2n},d_{2n})$

  The integrated full solution will be
$(x_{11},y_{11},d_{11})(x_{12},y_{12},d_{12})$……$(x_i,y_i,d_i)$ $(x_{2i+1},y_{2i+1},d_{2i+1})$……$(x_{2n},y_{2n},d_{2n})$

  We take the left part from left solution and right part from the right solution.

FindSolutionFromLeftAndRight(currentPoint,destinationPoint)
Step1: for each solution set s1 in leftSolution
Begin
Step2: for each solution set s2 in rightSolution

Begin

Step 3: for each common point p in points
Begin

Step4: sol1 ← all points in parent1 till p + rotations necessary to transform direction of p to the one of point p of parent2 + all points in parent2 after p

Step5: Add sol1 to fullSolution set

3.3.1.4 Crossover

This process means to mix two solutions already known, and to generate a solution from the mixture of the two solutions. The new solutions may be better solution than its parents. Hence two good solutions are taken for this step.

The crossover can occur between any of the following pairs of solutions

- Full Solution and Full Solution
- Left Solution and Full Solution
- Full Solution and Right Solution
- Left Solution and Right solution

In all of these we have two sets of sequences of points (solutions). One set is chosen from the left entities mentioned above and the other from the right entities of the corresponding set mentioned above. The crossover process is similar to the one used in initial process

If the left sequence selected is

$(x_{11},y_{11},d_{11})(x_{12},y_{12},d_{12})\ldots\ldots\ldots\ldots\ldots\ldots(x_i,y_i,d_i)\ (x_{1i+1},y_{1i+1},d_{1i+1})\ldots\ldots\ldots\ldots\ldots\ldots\ldots(x_{1n},y_{1n},d_{1n})$

and right sequence selected is

$(x_{21},y_{21},d_{21})(x_{22},y_{22},d_{22})\ldots\ldots\ldots\ldots(x_i,y_i,d_i)\ (x_{2i+1},y_{2i+1},d_{2i+1})\ldots\ldots\ldots\ldots\ldots\ldots(x_{2n},y_{2n},d_{2n})$

The integrated full solution will be

$(x_{11},y_{11},d_{11})(x_{12},y_{12},d_{12})\ldots\ldots\ldots\ldots\ldots(x_i,y_i,d_i)\ (x_{2i+1},y_{2i+1},d_{2i+1})\ldots\ldots\ldots\ldots\ldots\ldots\ldots(x_{2n},y_{2n},d_{2n})$

We take the left part from left sequence and right part from the right sequence.
The final solution is added to the solution set, if crossover is possible. If there are more than one common points, the crossover takes place for each of these points. All these are added to the solution set.

CrossOver(parent1,parent2)

Step 1: points ← find common points in parent1 and parent2 that have same x and y coordinate
Step 2: if points <> null
Step 3: for each common point p in points
Begin

Step4: sol1 ← all points in parent1 till p + rotations necessary to transform direction of p to the one of point p of parent2 + all points in parent2 after p

Step5: Add sol1 to fullSolution set

3.3.1.5 Mutation

In this method we try to find a random solution out of the given solutions. This is done to introduce randomness in the algorithm, so that the algorithm does not get trapped in local minima.

This process occurs with a probability of 0.02. In this process any one full solution is chosen. We take any two arbitrary points on this solution. Then we try to find out full solutions between these two points. Hence we try to find out a path between these points using algorithm similar to the algorithm used to find the initial solution. If such a path is found we replace the path between these two points with this new path. If more than one solutions are found, we replace the old path by the one with the least count of fitness function (most fit). This is added to the solution set.

Mutate()
Step 1: if no of solutions in full solution set > 0 then
Step 2: r1,r2 ← any two random points on a random solution from fullSolution set set with r1 on the left of r2
Step 3: find solutions as found in initial solution with start point as r1 and goal point as r2
Step 4: if full results generated > 0
Step 5: newSolution ← most fit solution in this generated solution set
Step 7: sol1 ← all points in solution till r1 + rotations necessary to transform direction of r1 to the one of first point of newSolution + all points in newSolution + rotations necessary to transform direction of last point in newSolution to r2 + all points in solution after r2
Step 8: add sol1 to solutionFull

3.3.1.6 Selection
The selection refers to the selection of parent chromosomes to get the child chromosome. The selection criterion we follow is

- With a probability of 2/3, select the most fit chromosome from the full solution set and any one randomly from the left/right/full solution set
- With a probability 1/3 select any random chromosome from full solution set and any one randomly from left/right/full solution set

3.3.1.7 Algorithmic Details
Another aspect of the algorithm used is its real time nature. If the Algorithm, due to excessive input or any other reason fails to produce a result in half the threshold time, then no move is made at all. This avoids the wrong calculations being made. As after this duration it is possible that the obstacles might have changed their location and the algorithm was using the old data.

The position of any node refers to its x, y coordinate and its direction.

The following is the algorithm
NavigationPlan()
Step 1: while(CurrentPosition <> FinalPosition)
Begin
Step 2: m ← getNextMove(CurrentPosition)
Step 3: moveRobot(m)

getNextMove(CurrentPosition)
Step 1: Find initial solution
Step 2: add a node n with position(n) = CurrentPosition
Step 4: for i = 1 to noIterations
Begin
Step 5: with a probability of 0.02 mutate()
Step 6: if at least 1 full solution exists
Step 7: select parent1, parent2 from the solution sets
Step 8: crossover(parent1,parent2) or crossover(parent2,parent1) whichever applicable
Step 9: else find more initial solutions and add the results to the existing solution sets
Step 10: if full solution is not empty then return first point in full solution
Step 11: if left solution is not empty return first point in left solution
Step 12: return current point

3.3.2 CASE II: Algorithm Artificial Neural Network (Back Propagation)
This algorithm tries to find out the solution by learning from the historical data. The algorithm used is the conventional ANN Back Propagation to find out the goal (final position) starting from the initial position. We know

that if we train the robot how to act in various obstacle situation, the robot can 'learn' the various actions to be taken in various situations[9][15]. In this algorithm we use the following:

No. Of Inputs: 26
No. Of Hidden Layers: 2
No. of Output Layers: 3
No. of Neurons in hidden Layers: 20, 15
Activation Function for the three layers: tansig, purelin, purelin

Range of Inputs: 0 to 1
Range of each output 0 to 1

Each position is marked as (x,y,d)
Where x=x coordinate
Y=y coordinate
D=direction (0-7)

3.3.2.1 Explanation of the Inputs
There are total 26 inputs to the neural network (I0-I25). These are as follows:

The I0 and I1 denote the rotations needed for the robot to rotate from its present direction to the direction in which the goal is present. We know that the rotation can be either -2 or +2 (180 degrees left or 180 degrees right), -1 (90 degrees left), 0 (no rotation), +1 (90 degrees right). These are represented by these 2 inputs as follows:
(I0,I1)=(0 0) represents +2 or -2
(I0,I1)=(0 1) represents +1
(I0,I1)=(1 1) represents 0
(I0,I1)=(1 0) represents -1

It should be noted that these are numbered according to gray codes so that even if one of the bits is corrupted by noise, the effect on the neural network is the minimum.

The other bits (I2 to I25) represent the condition of the map. Considering the whole map as a grid of (MXN), we take a small portion of it (5X5), with the robot at the center of the map. Each coordinate of this map is marked as 1 if the robot can move to this point in the next step, or 0 if the point does not exist or the robot cannot move to this point because of some obstacle. So we have taken a small part of the graph. If (x,y) be the coordinate of the robot, we take (x-2,y-2)……(x+2,y+2) as the graph. These are 25 points. These points, excluding the center of the graph (where our robot stands) are fed into the neural network.

The layout of the 24 inputs (I2 to I25) is given in figure 4. Here (m,n) are the coordinates of our robot. I2-I25 denotes the inputs. These are 1 or 0 depending on the obstacles.

Figure 4: The inputs for the Artificial Neural Network

I2 (m-2,n-2)	I3 (m-2,n-1)	I4 (m-2,n)	I5 (m-2,n+1)	I6 (m-2,n+2)
I7 (m-1,n-2)	I8 (m-1,n-2)	I9 (m-1,n)	I10 (m-1,n+1)	I11 (m-1,n+2)
I12 (m,n-2)	I13 (m,n-1)	Robot (mXn)	I14 (m,n+1)	I15 (m,n+2)
I16 (m+1, n-2)	I17 (m+1,n-1)	I18 (m+1,n)	I19 (m+1,n+1)	I20 (m+1,n+2)
I21 (m+2,n-2)	I22 (m+2,n-1)	I23 (m+2,n)	I24 (m+2,n+1)	I25 (m+2,n+2)

3.3.2.2 Explanation of the Outputs

There are total 3 outputs (O0,O1,O2), each can be either 0 or 1. We can make 8 combinations of outputs using these. The meanings of these outputs are as under:
(O0,O1,O2)=(0,0,0) Rotate left 90 degrees.
(O0,O1,O2)=(0,0,1) Rotate left 45 degrees.
(O0,O1,O2)=(0,1,0) Move forward.
(O0,O1,O2)=(0,1,1) Rotate left 45 degrees and move forward.
(O0,O1,O2)=(1,0,0) Do not move.
(O0,O1,O2)=(1,0,1) Rotate right 90 degrees.
(O0,O1,O2)=(1,1,0) Rotate right 45 degrees and move forward.
(O0,O1,O2)=(1,1,1) Move right 45 degrees.

It should be noted that this sequence is in gray code as well, if we arrange it in gray code sequence, the order of the moves will be 90 degrees left rotate, 45 degree left rotate, 45 degree left rotate and move forward, move forward, 45 degrees rotate right and move forward, 45 degrees rotate right, 90 degrees rotate right.

Hence there is a transition in series. This ensures that the effect of noise is minimum.

3.3.2.3 Special Constraints put in the algorithm

There were a few limitations of the algorithm; these were removed by applying some constraints to the algorithm, which ensured the smooth working of the algorithm

1) It may happen that the robot does not move or keeps turning its position continuously. Since there is no feedback in test mode, this is likely to continue indefinitely. For this we apply the constraint that if the robot sits idle for more than 5 time stamps and does not make any move (excluding rotations), we take a random move.
2) It may happen that the robot develops affinity to walk straight in a direction, hence if it is situated in the opposite direction, it may go farther and farther from the destination. For this we limit that if the robot makes 5 consecutive moves (excluding rotations) which increase its distance from the destination, we make the next one move only which decreases its distance (excluding rotations).

3.3.2.4 Procedure

The algorithm first generated test cases, so that the established neural network can be trained. These are done by generating input conditions and solving them using A* algorithm and getting the inputs and outputs.

The neural network is established and trained using these test cases. We lay more stress on the rotational parameters of the robot, i.e. the capability of the robot to rotate from its present direction to the goal direction.

After the training is over, we enter into the test case. Looking at the condition of the map, the input cases are generated and are fed into the Neural Network. The output is fetched and decoded. The same is followed and robot is moved.

The following is the algorithm
GenerateTestCases()
Step 1: initialize map and generate N obstacles at positions (xi,yi,di)
Step 2: while(noTestCaseGenerated <> noTestCasesDesired)
Begin
Step 3: Generate the robot at random position (xi,yi,di)
Step 4: I ← Input sequence
Step 5: m ← getNextRobotMoveUsingAStar(CurrentPosition)
Step 6: moveRobot(m)
Step7: O ← Output Sequence
Step 8: Add (I,O) to training data

NavigationPlan()
Step 1: n ← Generate Neural Network
Step 2: n ← TrainNeuralNetwork()
Step 3: while(CurrentPosition <> FinalPosition)
Begin
Step 4: I ← generateInputSequence()
Step 5: O ← generateOutputSequence(I)
Step6: if O gives valid move
Step 7: moveRobot(O)
Step 8: if robot not changed position in last 5 steps
Step 9: makeRandomMove()
Step10: if distance of robot decreasing in last 5 steps (excluding rotations)
Step11: move next move only if the next move decreases distance or is of equal distance

3.3.3 Comparison of Genetic and Artificial Neural Network Algorithms

We have seen the working of the algorithms. We saw both these algorithms working and guiding the robot ensuring that the robot gets to the final destination without collisions. The two algorithms used different ideologies but guided the robot well to reach the final destination.

The following are the key points of comparison between these two algorithms:

- The Genetic Algorithm takes a lot of time to generate the result as compared to Artificial Neural Network. This time is because of the optimizations it tries to make and also to generate the initial solution. The Artificial Neural Network in the test mode works very efficiently. It just calculates the result from the input weight, which is not a time consuming task. Hence in this respect the Artificial Neural Network performed better.
- The overall path traveled by the robot using genetic algorithm was less. Hence in this respect the genetic algorithm worked better. This difference was due to the optimizations made by the Genetic Algorithm that constantly tried to find the best solution. Neural Network also may get new situations. It just tried to predict the best move.
- The Artificial Neural Network Algorithm would fail if highly chaotic environment was applied. Also if a semi maze like condition was given, the neural network algorithm failed for similar reasons. This is because the neural network was designed to work in a way to reach the goal node, by overcoming obstacles. Hence it had no way to know how to react when the route was either blocked (high chaotic environment) or did not exist (maze like situation). Hence in this respect also the Genetic Algorithms were better
- The memory requirements for the Genetic Algorithms are very high as compared to Artificial Neural Network. The Neural Network just has to know its surroundings. But the Genetic Algorithm should have the complete grid from current position to goal position, stored on its memory.

Both these algorithms were highly efficient when only some obstacles were applied or no obstacle was applied. Even the path traced was very straight, which proves the efficiency of these algorithms.

Looking at the comparison, we infer that Genetic Algorithms must be applied when the conditions are highly chaotic or when the static obstacles make the grid form a maze like structure. When the obstacles come one-by-one, the robot just needs to surpass them. In such cases the Artificial Neural Network works better.

4 Testing

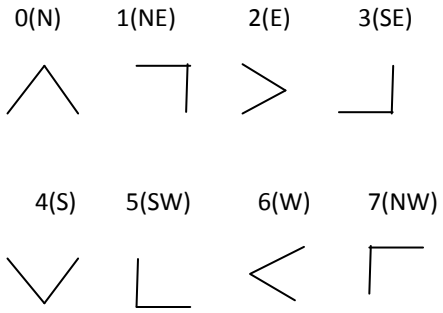
For the testing of the algorithm we made a program using Java Applets that generated n number of obstacles each controlled by an independent state. These obstacles were moved completely randomly independent of each other. These were also moved on the lines of collision free movement.

Another thread was started that moved our robot from the initial to the final position. This thread used the algorithm discussed to move the robot in its path.

All the obstacles and our robot were displayed in the applets. The directions were displayed with the head pointing towards the direction (Refer Figure 5). Here the first figure shows any robot/obstacle pointing out in the upward

(north direction). The direction number of this is 0 (as explained earlier). Similarly the subsequent figures show the robot/obstacle pointing in North-East(1), East(2), South-East(3), South(4), South-West(5), West(6) and North-West(7) respectively.

Figure 5: The representation of obstacles in various directions



An open figure denotes obstacle and a closed denotes robot.
The applet also collected the complete path followed by the robot to give a path trace of the robot.

5 Results
The algorithms were tested using the technique described earlier. In all the movement of the robot at each step was recorded using all the algorithms. The results are as follows

5.1 Genetic Algorithm
The first algorithm used was genetic algorithm. Here we had used a sample grid space on which the robot is to be moved of [100 X 100] dimension. The coordinates could vary from (0,0) to (99,99). We used total 750 obstacles, which all could move a unit step at any unit time. The threshold time was fixed to be 9 seconds. We moved the robot from position (0,0) to the goal position (99,99).

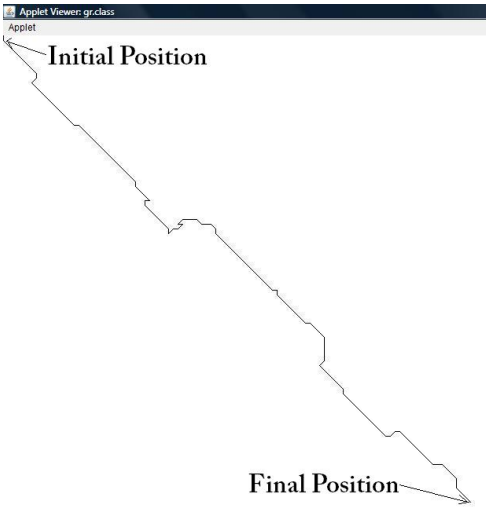
Some of the constants as mentioned in the algorithm were fixed as follows:
noIterations = 250
noIterationsInitial = 15
randomMoves = 15
maxElements = 1500

Points traced at various times are given in Table2.
The path traced by the robot is given in Figure 6 (For 4 runs)

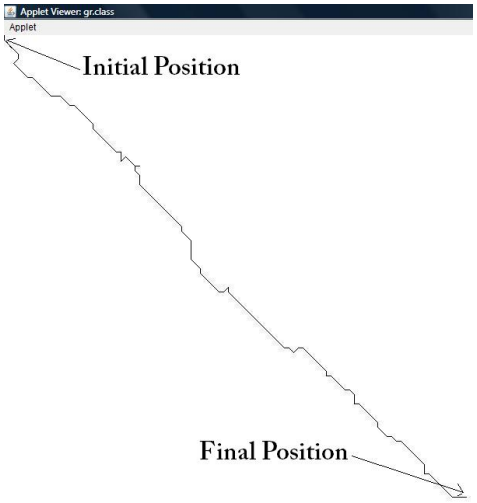
5.2 Artificial Neural Networks
A similar run was performed for the Artificial Neural Networks as well. The parameters were quite similar to the ones used in the previous algorithm. Sample grid space on which the robot is to be moved was of [100 X 100] dimension where the coordinates could vary from (1,1) to (100,100). MATLAB was used for the simulation purpose. We used total 500 obstacles, which all could move a unit step at any unit time. The threshold time was fixed to be 1 seconds. We moved the robot from position (1,1) to the goal position (99,99).

Points traced at various times are given in Table3.
The path traced by the robot is given in Figure 7 (For 4 runs)

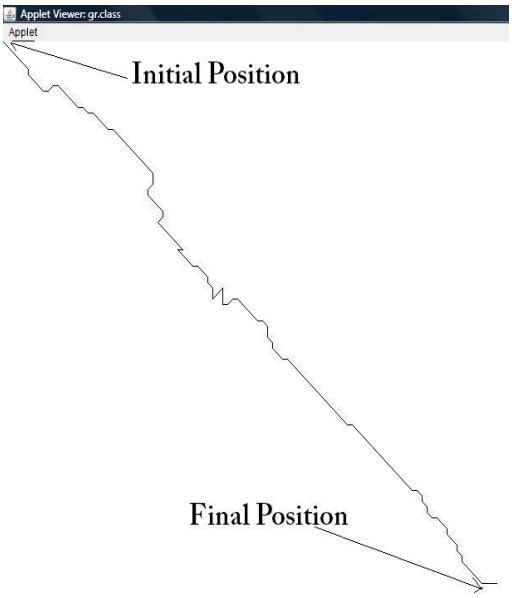
Fig 6: The path traced by the robot using genetic algorithm at various runs



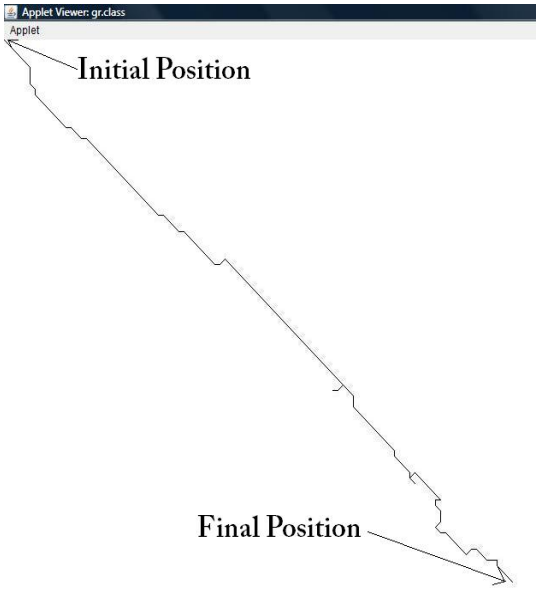
Run #1



Run #2

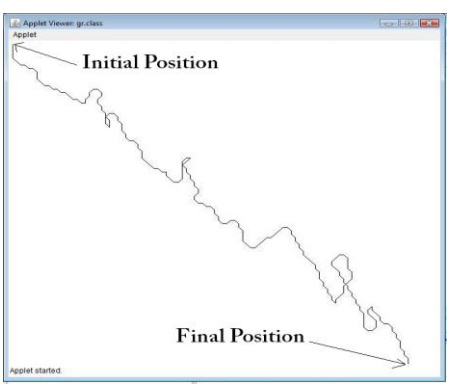


Run #3

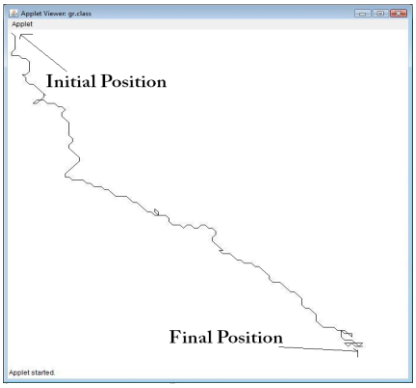


Run #4

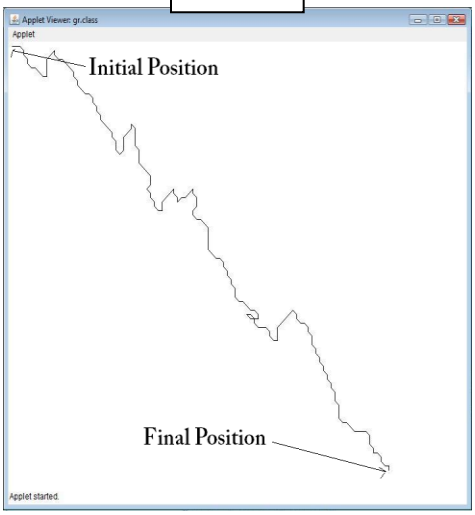
Fig 7: The path traced by the robot using ANN algorithm at various runs



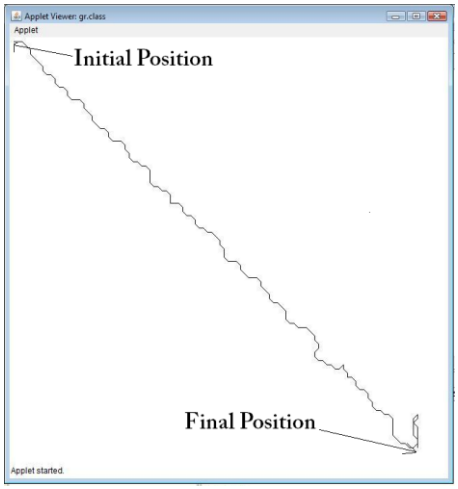
Run #1



Run #2

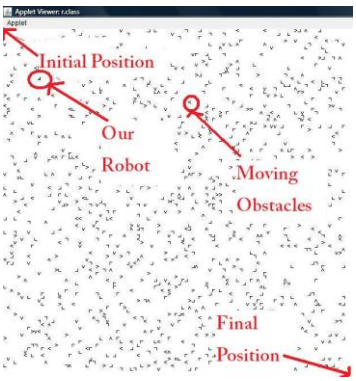


Run #3



Run #4

Figure 8: Obstacles and Robot in Motion



The condition of the board at a random time is given in figure 8
(Note: coordinate (0,0) is the top left coordinate)

The data for the consecutive runs is given below where each entry is the value at each time span:

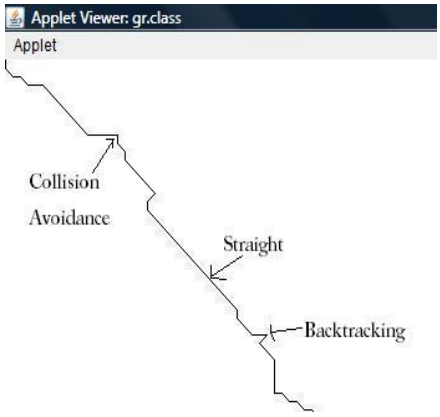
Closely watching the robot go towards its goal, we see that there is no collision on its way. Hence we have been successful in avoiding collision. Also we find looking at the path traced, that the path is optimal with respect to the conditions given.

Carefully observing the path traced we find the following kinds of paths:

1. **Stationary Phase**: In this the robot is unable to make a move. Due to the extremely chaotic conditions, it is not possible to find a solution. The robot waits for the time it gets a path. If there are a set of obstacles in very close vicinity, it may cause this problem.
2. **Straight Phase**: Here the robot moves straight towards the goal. The obstacles have no effect on its movement. If we take very few obstacles, this is what normally happens.
3. **Collision Avoidance Phase:** If an obstacle is very close to the robot, it takes a stern turn to avoid collision and keep going. This phase is known as the collision avoidance phase.
4. **Backtracking Phase:** If the robot happens to deviate a lot from its path due to the excessive number of obstacles in its close vicinity, it backtracks to its path and then continues to move further.

These are shown in figure 9.

Figure 9: The various types of paths in the motion of robot



The more chaotic the scene becomes, we see the number of straight paths decreasing and the number of stationary, collision avoidance and backtracking paths increasing. These paths are more prone to vary depending on the number of obstacles in close vicinity rather than those in the entire grid. The efficiency of the algorithm lies in the measure of more the straight paths and lesser the backtracking paths it produces.

6 Conclusions

We have been able to move the robot from initial to final position in an environment of dynamically moving obstacles without collision. The path chosen is optimum. This technique can be used to enable the movement of many robots together in a common place.

We used first genetic Algorithm and solved the navigation control problem. We saw that the algorithm does take time for the generation of initial solutions, but generates good results. The results keep getting better at each iteration. The net result is that the algorithm is able to guide the robot well, making the path collision free.

We also saw Artificial Neural Networks. We need to train the network once, which is time consuming process. But once done, the network gives a very fast response. This algorithm is very suited to collision avoidance in path when the obstacles come one by one. The network can be easily trained for such conditions.

Table 2: Position of Robot at Various Times using Genetic Algorithm

Run #1	Run #2
(x,y,dir)	(x,y,dir)
(0,0,0), (0,0,1),(1,0,2),(2,1,3), (3,2,3),(4,3,3),(5,4,3), (6,5,3),(7,6,3),(8,7,3), (8,7,1),(9,7,2),(10,6,1), (10,6,2),(11,7,3),(12,8,3), (12,8,3),(13,9,3),(14,10,3), (15,11,3),(16,12,3),(17,13,3), (17,13,5),(17,13,2),(18,14,3), (19,15,3),(19,15,7),(19,15,5), (19,16,4),(20,17,3),(21,18,3), (22,19,3),(23,20,3),(23,20,0), (23,20,6),(23,20,2),(24,21,3), (25,22,3),(26,23,3),(27,24,3), (28,25,3),(29,26,3),(30,27,3), (31,28,3),(32,28,2),(33,29,3), (34,30,3),(35,31,3),(35,31,0), (35,30,1),(36,30,2),(37,31,3), (38,32,3),(39,33,3),(39,33,4), (40,34,3),(41,35,3),(42,35,2), (42,35,4),(41,36,4),(41,37,4), (40,38,1),(40,37,7),(40,37,5), (39,38,5),(39,39,4),(39,39,0), (39,39,3),(39,40,4),(39,41,4), (40,42,3),(40,42,6),(40,42,4), (40,43,4),(40,44,4),(41,45,3), (42,45,2),(43,46,3),(44,47,3), (45,48,3),(46,49,3),(46,49,3), (47,50,3),(48,51,3),(49,52,3), (50,53,3),(51,54,3),(52,55,3), (52,55,7),(52,55,6),(52,55,2), (53,56,3),(54,57,3),(54,58,4), (54,58,3),(55,58,2),(56,59,3), (57,60,3),(58,61,3),(59,62,3), (60,63,3),(61,64,3),(61,65,4), (62,66,3),(63,67,3),(64,68,3), (65,68,0),(65,68,1),(66,68,2), (67,68,4),(67,68,0),(67,68,1), (68,68,2),(69,68,2),(69,68,3), (69,68,1),(69,68,2),(70,67,3), (71,68,3),(72,69,3),(73,70,3), (74,71,3),(75,72,3),(76,72,2), (77,73,3),(78,74,3),(79,75,3), (80,76,3),(81,77,3),(82,78,3), (83,79,3),(84,80,3),(85,81,3), (85,82,4),(85,82,6),(84,83,6), (84,83,0),(84,83,2),(84,83,7), (84,83,6),(84,83,5),(84,84,4), (85,85,3),(86,86,3),(87,87,3), (88,88,3),(89,89,3),(90,90,3), (91,91,3),(91,92,4),(91,92,3), (91,93,4),(92,94,3),(93,95,3), (94,96,3),(94,96,1),(95,96,2), (96,96,2),(97,97,3),(98,98,3), (99,99,3)	(0,0,0), (0,0,1),(1,0,2),(2,1,3), (3,2,3),(4,3,3),(5,3,1), (6,2,1),(6,2,2),(6,2,2), (7,3,3),(7,3,2),(8,4,3), (9,5,3),(9,6,4),(10,7,3), (11,8,3),(12,9,3),(13,10,3), (13,11,4),(13,12,4),(14,13,3), (15,14,2),(15,14,6),(15,14,5), (15,15,4),(16,16,3),(17,17,3), (18,18,3),(19,19,3),(20,19,2), (21,20,3),(22,21,3),(23,22,3), (24,23,3),(25,24,3),(25,25,4), (25,25,3),(25,25,1),(26,25,2), (27,25,4),(26,26,5),(26,26,6), (26,26,4),(27,27,3),(28,28,3), (28,28,7),(28,28,3),(28,29,4), (28,29,0),(28,29,1),(28,28,7), (28,28,1),(29,28,2),(29,28,7), (29,28,2),(29,28,3),(30,29,3), (31,29,2),(32,29,3),(33,30,3), (34,31,3),(35,32,3),(36,33,3), (37,34,3),(38,35,3),(39,36,3), (40,37,3),(41,38,1),(42,38,2), (43,39,3),(44,40,1),(45,40,2), (46,40,2),(47,40,0),(47,40,2), (48,40,3),(49,41,3),(50,42,3), (51,42,2),(52,43,3),(53,44,3), (54,45,3),(55,46,3),(55,47,4), (54,48,1),(55,48,3),(56,49,3), (57,50,3),(58,51,3),(59,52,3), (60,53,3),(61,54,3),(62,55,3), (63,56,3),(64,57,3),(65,58,3), (66,59,3),(67,60,3),(67,61,4), (68,62,6),(67,63,5),(67,64,4), (68,65,3),(69,66,3),(70,67,3), (71,68,3),(72,69,3),(73,69,2), (73,69,7),(73,69,5),(73,70,4), (74,71,3),(75,72,3),(76,73,3), (76,74,4),(77,75,3),(78,75,1), (79,75,2),(79,75,6),(79,75,5), (79,76,4),(80,77,3),(81,78,3), (82,79,3),(83,80,3),(84,80,2), (85,81,3),(86,82,3),(86,83,4), (87,84,3),(87,85,4),(87,85,6), (87,85,4),(88,86,3),(89,87,3), (90,88,3),(90,88,0),(90,88,2), (91,89,3),(91,89,0),(90,88,7), (90,88,5),(90,88,2),(91,89,3), (92,90,3),(93,90,2),(94,90,2), (94,90,0),(94,90,2),(94,90,3), (94,91,4),(94,91,2),(95,92,3), (96,93,3),(97,94,3),(98,95,3), (99,96,3),(99,97,4),(99,98,4), (99,99,4)

Table 3: Position of Robot at Various Times using ANN Algorithm

Run #1	Run #2
(x,y,dir)	(x,y,dir)
(1,1,1), (1,1,2),(1,1,3),(1,1,4), (1,1,5),(1,2,4),(1,3,4), (1,4,4),(1,4,5),(1,4,5),	((1,1,1), (1,1,2),(1,1,3),(1,1,4), (1,1,5),(1,1,6),(1,1,7), (1,1,0),(1,1,1),(1,1,2),

(1,5,4),(2,6,3),(2,6,2),
(3,7,3),(4,7,2),(5,8,3),
(6,8,2),(7,9,3),(7,10,4),
(7,11,4),(8,12,3),(9,13,3),
(10,13,2),(11,14,3),(12,14,2),
(13,15,3),(14,15,2),(15,16,3),
(16,17,3),(17,17,2),(18,18,3),
(19,18,2),(19,18,0),(19,17,0),
(20,16,1),(20,15,0),(21,14,1),
(22,14,2),(23,15,3),(23,16,4),
(22,17,5),(22,18,4),(23,19,3),
(23,20,4),(24,21,3),(24,22,4),
(25,23,3),(25,23,4),(25,24,4),
(25,25,4),(25,25,6),(24,24,7),
(24,23,0),(24,22,0),(25,21,1),
(26,21,2),(27,22,3),(27,23,4),
(28,24,3),(28,25,4),(28,26,4),
(29,27,3),(29,28,4),(30,29,3),
(30,29,2),(31,29,2),(32,30,3),
(32,31,4),(33,32,3),(33,33,4),
(33,34,4),(34,35,3),(35,36,3),
(36,37,3),(37,37,2),(38,38,3),
(39,38,2),(39,38,3),(39,39,4),
(40,40,3),(41,41,3),(42,41,2),
(43,40,1),(43,39,0),(43,38,0),
(43,37,0),(43,36,0),(44,35,1),
(45,34,1),(45,34,7),(44,34,6),
(43,35,5),(43,36,4),(43,36,3),
(44,37,3),(44,38,4),(45,39,3),
(45,40,4),(46,41,3),(46,42,4),
(47,43,3),(47,44,4),(46,45,5),
(46,46,4),(47,47,3),(48,47,2),
(49,48,3),(50,48,2),(51,49,3),
(51,50,4),(52,51,3),(52,52,4),
(53,53,3),(54,53,2),(55,52,1),
(56,52,2),(57,53,3),(57,54,4),
(58,55,3),(58,56,4),(58,57,4),
(58,58,4),(59,59,3),(60,60,3),
(61,60,2),(62,59,1),(63,58,1),
(64,57,1),(65,57,2),(66,56,1),
(67,55,1),(68,54,1),(69,54,2),
(70,55,3),(70,55,5),(70,56,4),
(71,57,3),(71,58,4),(72,59,3),
(72,60,4),(73,61,3),(73,62,4),
(74,63,3),(75,64,3),(76,65,3),
(76,66,4),(77,67,3),(77,68,4),
(77,69,4),(78,70,3),(78,71,4),
(79,72,3),(79,73,4),(80,74,3),
(80,75,4),(81,76,3),(81,76,1),
(81,75,0),(82,74,1),(83,73,1),
(83,72,0),(83,71,0),(84,70,1),
(85,69,1),(85,68,0),(85,67,0),
(84,66,7),(84,65,0),(84,65,7),
(84,64,0),(84,64,7),(84,63,0),
(83,62,7),(83,62,6),(82,62,6),
(81,63,5),(80,64,5),(80,65,4),
(81,66,3),(82,67,3),(82,68,4),
(83,69,3),(83,70,4),(84,71,3),
(84,72,4),(85,73,3),(85,74,4),
(86,75,3),(86,76,4),(87,77,3),
(87,77,3),(87,78,4),(88,79,3),
(88,80,4),(88,81,4),(88,82,4),
(88,83,4),(89,84,3),(90,84,2),
(91,83,1),(91,82,0),(91,81,0),
(90,80,7),(89,79,7),(89,78,0),
(90,77,1),(90,77,0),(90,77,1),
(90,77,3),(90,78,4),(91,79,3),
(92,80,3),(92,81,4),(93,82,3),
(94,82,2),(95,83,3),(95,84,4),
(96,85,3),(96,86,4),(97,87,3),
(97,88,4),(97,89,4),(98,90,3),
(98,91,4),(99,92,3),(99,93,4),
(99,94,4),(99,95,4),(98,96,5),
(98,97,4),(97,98,5),(96,99,5),
(96,99,7),(95,98,7),(95,97,0),
(96,96,1),(97,96,2),(98,97,3),
(98,98,4),(99,99,3

(1,1,0),(1,1,1),(1,1,2),
(1,1,3),(1,1,4),(2,2,3),
(2,3,4),(2,4,4),(2,5,4),
(2,6,4),(1,7,5),(1,7,4),
(1,8,4),(1,8,2),(2,8,2),
(3,8,2),(4,9,3),(5,9,2),
(6,10,3),(6,11,4),(6,12,4),
(6,13,4),(5,14,5),(5,14,6),
(4,14,6),(4,14,4),(4,14,2),
(5,15,3),(5,16,4),(5,16,2),
(6,17,3),(7,17,2),(8,18,3),
(9,19,3),(10,20,3),(11,21,3),
(11,21,2),(11,21,0),(10,20,7),
(10,20,5),(10,21,4),(9,22,5),
(8,22,6),(7,23,5),(7,23,3),
(8,23,2),(8,23,0),(8,23,1),
(9,22,1),(10,22,2),(11,23,3),
(12,23,2),(13,23,2),(14,24,3),
(15,24,2),(16,25,3),(16,25,4),
(15,26,5),(15,27,4),(15,27,2),
(15,27,2),(16,28,3),(17,29,3),
(17,30,4),(17,31,4),(17,32,4),
(17,33,4),(17,33,5),(17,33,3),
(18,34,3),(18,35,4),(17,36,5),
(17,36,3),(17,36,4),(17,36,5),
(17,37,4),(18,38,3),(19,39,3),
(19,39,4),(20,40,3),(20,41,4),
(19,42,5),(18,43,5),(17,44,5),
(17,44,4),(17,44,5),(17,44,5),
(16,45,5),(16,46,4),(16,46,3),
(17,46,2),(18,47,3),(18,47,1),
(19,47,2),(20,47,2),(21,47,2),
(22,48,3),(23,48,2),(24,49,3),
(24,49,1),(25,49,2),(25,49,1),
(26,49,2),(27,50,3),(28,50,2),
(29,51,3),(30,52,3),(31,52,2),
(32,52,2),(33,52,2),(34,53,3),
(34,53,2),(35,54,3),(36,54,2),
(37,55,3),(37,55,4),(38,56,3),
(39,57,3),(40,57,2),(41,58,3),
(42,58,2),(42,58,0),(42,57,0),
(41,56,7),(41,56,5),(41,57,4),
(42,58,3),(43,58,2),(44,58,2),
(45,59,3),(45,60,4),(45,60,5),
(45,60,4),(45,60,5),(45,60,3),
(46,61,3),(47,61,2),(47,61,0),
(47,61,1),(48,61,2),(49,62,3),
(49,62,1),(50,61,1),(51,61,2),
(52,62,3),(53,62,2),(54,61,1),
(55,61,2),(55,61,2),(56,62,3),
(56,62,1),(57,62,2),(58,63,3),
(58,64,4),(57,65,5),(57,66,4),
(58,67,3),(58,67,2),(59,68,3),
(59,68,2),(60,69,3),(60,69,1),
(60,69,7),(59,69,6),(59,69,4),
(60,70,3),(61,70,2),(62,70,2),
(63,71,3),(63,71,2),(64,72,3),
(64,72,2),(65,72,2),(66,73,3),
(67,73,2),(68,73,2),(69,74,3),
(70,74,2),(71,75,3),(72,76,3),
(72,76,2),(73,77,3),(73,78,4),
(74,79,3),(74,79,2),(74,79,3),
(74,79,1),(75,79,2),(76,79,2),
(77,80,3),(77,80,2),(78,81,3),
(79,82,3),(80,82,2),(81,83,3),
(82,84,3),(82,85,4),(82,86,4),
(83,87,3),(84,87,2),(85,88,3),
(86,89,3),(86,90,4),(86,90,2),
(86,90,3),(86,91,4),(87,92,3),
(87,92,4),(87,92,2),(88,93,3),
(89,93,2),(90,93,2),(91,93,2),
(91,93,2),(92,94,3),(92,94,1),
(93,94,2),(94,95,3),(95,95,2),
(95,95,1),(95,95,1),(96,95,2),
(96,95,3),(96,96,4),(96,96,6),
(95,95,7),(94,94,7),(93,94,6),
(93,94,4),(93,95,4),(93,96,4),
(94,97,3),(95,97,2),(96,97,2),
(97,98,3),(98,98,2),(98,98,0),
(98,98,1),(99,98,2),(99,98,2),
(99,98,4),(99,98,6),(98,99,5),

	(98,99,6),(97,99,6),(96,98,7), (95,98,6),(95,98,4),(95,98,5), (95,98,7),(94,98,6),(94,98,4), (94,98,3),(95,99,3),(95,99,4), (95,99,4),(95,99,4),(95,99,2), (96,98,1),(97,98,2),(98,99,3), (98,99,1),(98,99,2),(99,99,2)

7 References

[1] Hutchinson, S. A. and Kak, A. C., "Planning sensing strategies in a robot work cell with Multi-sensor capabilities," IEEE Trans. On Robotics and Automation,, vol.5, no.6, 1989.

[2] Rich, E. and Knight, K., Artificial Intelligence, McGraw-Hill, New York, pp. 29-98, 1991.

[3] Takahashi, O. and Schilling, R. J., "Motion planning in a plane using generalized voronoi diagrams," IEEE Trans. on Robotics and Automation, vol.5, no.2, 1989.

[4] Borenstain, J., Everett, H. R., and Feng, L., Navigating Mobile Robots: Systems and Techniques, A. K. Peters, Wellesley, 1996

[5] Matlin, W. Margaret, Cognition, Hault Sounders, printed and circulated by Prism books, India, 1996.

[6] Konar, A. and Pal, S., "Modeling cognition with fuzzy neural nets" In Fuzzy Systems Theory: Techniques and Applications, Leondes, C. T., Ed., Academic Press, New York, 1999.

[7] Pagac, D., Nebot, E. M. and Durrant. W., H., "An evidential approach to map building for autonomous robots," IEEE Trans. On Robotics and Automation, vol.14, no.2, pp. 623-629, Aug. 1998.

[8] V. Ayala-Ramirez, A. Perez-Garcia, E J. Montecillo-Puente, R.E. Sanchez-Yanez, "Path planning using genetic algorithms for mini-robotic tasks" 2004 IEEE International Conference on Systems, Man and Cybernetics

[9] Hem6 Fkezza-Buet, FrBd6ric Alexandre "Modeling prefrontal functions for robot navigation"

[10] Theodore W. Manikas, Kaveh Ashenayi, and Roger L. Wainwright, "Genetic Algorithms for Autonomous Robot Navigation", IEEE Instrumentation & Measurement Magazine December 2007

[11] DU Xin, CHEN Hua-hua, GU Wei-kang, "Neural network and genetic algorithm based global path planning in a static environment", Journal of Zhejiang University SCIENCE

[12] ZHANG Huan-cheng, ZHU Miao-liang, "Self-organized architecture for outdoor mobile robot navigation", Journal of Zhejiang University SCIENCE

[13] Peter Corke, Ron Peterson, Daniela Rus, "Networked Robots: Flying Robot Navigation using a Sensor Net", April 18, 2003

[14] Cory Quammen, "Evolutionary learning in mobile robot navigation", The ACM Student Magazine

[15] YONG-KYUN NA∗ AND SE-YOUNG OH, "Hybrid Control for Autonomous Mobile Robot Navigation Using Neural Network Based Behavior Modules and Environment Classification", 2003 Kluwer Academic Publishers. Manufactured in The Netherlands

[16] SEYYED EHSAN MAHMOUDI, ALI AKHAVAN BITAGHSIR, BEHJAT FOROUZANDEH and ALI REZA MARANDI, "A NEW GENETIC METHOD FOR MOBILE ROBOT NAVIGATION", 10th IEEE International Conference on 30 August - 2 September 2004, Miedzyzdroje, Poland, Methods and Models in Automation and Robotics

[17] Torvald Ersson and Xiaoming Hu, "Path Planning and Navigation of Mobile Robots in Unknown Environments"

[18] László Kiss, Annamária R. Várkonyi-Kóczy, "A Universal Vision-based Navigation System for Autonomous Indoor Robots"

[19] Sven Behnke, "Local Multiresolution Path Planning", Preliminary version in Proc. of 7th RoboCup Int. Symposium, Padua, Italy, 2003

[20] S. Veera Ragavan, and V. Ganapathy, "A Unified Framework for a Robust Conflict-Free Robot Navigation", PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 21 JANUARY 2007 ISSN 1307-6884

About the authors

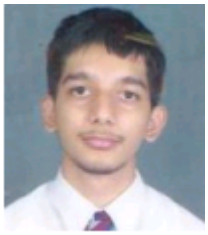
Dr. Anupam Shukla

Dr. Anupam Shukla is an Associate Professor in the IT Department of the Indian Institute of Information Technology and Management Gwalior. He has 19 years of teaching experience. His research interest includes Speech processing, Artificial Intelligence, Soft Computing and Bioinformatics. He has published around 62 papers in various National and international

journals/conferences. He is referee for 4 international journals and in the Editorial board of International journal of AI and Soft Computing. He received Young Scientist Award from Madhya Pradesh Government and Gold Medal from Jadavpur University.

Dr. Ritu Tiwari

Dr. Ritu Tiwari is an Assistant Professor in the IT Department of Indian Institute of Information Technology and Management Gwalior. Her field of research includes Biometrics, Artificial Neural Networks, Signal Processing, Robotics and Soft Computing. She has published around 20 papers in various national and international Journals/conferences. She is referee for an international journal of Digital Signal Processing by Elsevier. She has received Young Scientist Award from Chhattisgarh Council of Science & Technology in the year 2006. She also received Gold Medal in her post graduation from NIT, Raipur.



Rahul Kala

Rahul Kala is a student of 4th Year Integrated Post Graduate Course (BTech + MTech in Information Communication Technology) in Indian Institute of Information Technology and Management G walior. His fields of research are robotics, design and analysis of algorithms, artificial intelligence and soft computing. He secured 7th position in the ACM International Collegiate Programming Contest, Kanpur Regionals. He is a student member of ACM. He also secured All India 8th position in Graduates Aptitude Test in Engineeging-2008 with a percentile of 99.84.