Robot Motion Planning with Neuro-Genetic-Fuzzy Approach in Dynamic Environment

Thongam Khelchandra and Jie Huang

Department of Computer and Information Systems,
The University of Aizu, Aizu-Wakamatsu 965-8580,

thongam@gmail.com, j-huang@u-aizu.ac.jp

Abstract

To find an optimal path for robots in an environment that is only partially known and continuously changing is a difficult problem. This paper presents a new method for generating a collision-free near-optimal path for an autonomous mobile robot in a dynamic environment containing moving and static obstacles using neural network and fuzzy logic with genetic algorithm. The mobile robot selects a collision-free local path using the neural network (ANN). A supervised learning-rule is used for the neural network using the gradient descent method. The fuzzy logic system with genetic algorithm comes into play when finding a local path by neural network becomes impossible. Fuzzy logic (FL) is used to avoid collisions when all the paths are blocked by obstacles. Genetic Algorithm (GA) is used as optimizer to find optimal locations along the obstacle-free directions and positions by selecting a set of fuzzy rules for the fuzzy logic system from a large rule base. Experimental results show that the method is efficient and gives near-optimal path reaching the target position of the mobile robot.

Keyword: Mobile robots, dynamic environment, motion planning, artificial neural network, fuzzy logic, genetic algorithm, obstacle avoidance, collision-free path, supervised learning, static obstacles.

I. Introduction

The basic motion planning problem is to find a collision-free path for a robot (a rigid or articulated object) among static obstacles of different shape and sizes [1, 2]. Several extension of the basic motion planning problem have been studied later, for instance, obstacles are moving [3], kinematic and dynamic constraints limit robot motions, optimized trajectories, multiple robots co-ordination. An efficient randomized planner for kinodynamic motion planning in the presence of moving obstacles with known trajectories have been presented [4]. Some researchers proposed an approach for anytime path planning and re-planning in partially known dynamic environments where the future behaviour of the moving obstacles is not known precisely [5]. Solving the problem of path planning by computational geometry seems to be unfeasible when implemented in real time. To tackle this problem, many researchers started using artificial neural networks [6 – 13], fuzzy systems [14], evolutionary algorithms [15] and hybrid techniques such as ANN-FL [16], GA-FL [17 – 18], GA-ANN [19]. The combination of these tools is to get advantage from both of them and to eliminate their individual limitations thereby making the whole process of finding the path very efficient. The spiking neural network [20] has been used for path planning in partially known and changing environment [21]. In their process a new optimal path is found by *path length limitation* and *cutting noise*, when a path becomes sub-optimal by a moving target point or obstacles.

In this paper, we propose a method of building path for an autonomous mobile robot using a combination of multilayer perceptron neural network and fuzzy logic system with genetic algorithm in a continuously changing dynamic environment. We only know the maximum velocity of each obstacles and the position of the obstacles that can be sense by the robot sensor from a specific robot position in the workspace. The future behaviour of the obstacles is not known and we developed a method to avoid the robot from the moving obstacles. The neural network (ANN) is use to avoid the moving obstacles initially and choose a path from a set of n paths if it is possible otherwise the fuzzy logic (FL) system with combination of genetic algorithm (GA) comes into play to avoid from hitting

the obstacles and move ahead towards the destination. Fuzzy logic is used for obstacle avoidance when neural network is unable to choose a path due to blockage of obstacles in all the set of directions. GA is used to make the fuzzy logic system more efficient by finding optimal locations from the collision free positions. Results show that the combination of these features is computationally efficient by helping each other to eliminate their individual limitations.

The paper is organized as follows: Section II gives the preliminaries to prepare for the rest of the work. We describe how the path is computed using ANN in Section III. Collision avoidance using combined fuzzy logic (FL) system and GA is explained in Section IV. Simulation results and its evaluation are presented in Section V and VI respectively. Finally the conclusion is given in Section VII.

II. Preliminaries

A. ***Problem Definition***

The formal definition of our motion planning problem is as follows. Let R be a robot moving in a two dimensional workspace W. The configuration of an arbitrary object is a specification of the position of every point on it relative to a fixed reference frame. Let F_R and F_W be Cartesian frame embedded in R and W respectively. So, a configuration of R is a specification of the position (x, y) and orientation θ of F_A with respect to F_W. We use the notation R(x) to refer to robot R configured at x in W. Let there be n moving obstacles O1, O2,….,On which state at time t is denoted by O(t). All the obstacles have a maximal velocity given by v1, v2,….,vn. The robot has a maximal velocity V which is larger than each of the maximal velocities of the obstacles. We do not assume any knowledge of the velocities and directions of motion of the moving obstacles, other than that they have a maximal velocity. Let Ω be a list of adjacent points that will represent the path of the robot. Let s, g ∈ Ω be the start and goal configuration of the robot respectively and let t_0 be the start time. The task is to compute a path $\Pi : [t_0,T] \rightarrow \Omega$, such that $\Pi(t_0) = s$ and $\Pi(T) = g$ and there is no collision with the moving

obstacles and the robot, i,e $\forall(t \in [t_0,T] :: R(\Pi(t)) \cap O(t) = \varphi)$. T is the arrival time of the robot R at its goal configuration.

B. *Artificial Neural Network (ANN)*

Our main objective is to find a collision free path of a robot system from a given initial position to some goal position. The environment is a 2 dimensional space with obstacles.

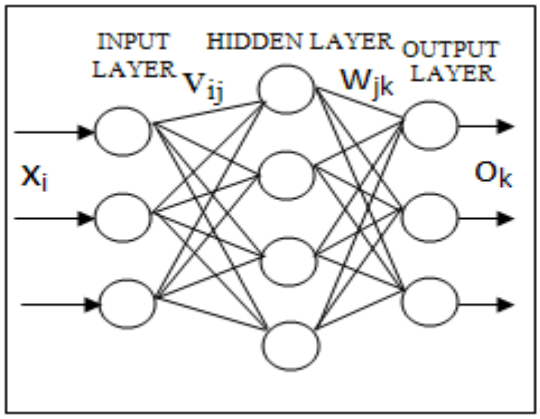

Figure 1. Structure of MLP network

The neural network that we used is the Multilayer perceptron (MLP). The structure of the neural network is given in Figure 1. The network contains three layers: input, hidden and output. The layers are connected by synaptic weights. The learning of the network is realized by back-propagation (BP) algorithm. The BP algorithm is based on the error-correction principle. The parameters that are used during the learning process are given:

x_i : The ith input

y_j: The output of the j^{th} hidden neuron

o_k : The output of the k^{th} output neuron

d_k : The desired output

v_{ij}: The weight from the i^{th} input to the j^{th} hidden neuron

w_{jk}: The weight from the j^{th} hidden neuron to the k^{th} output neuron

η : The learning rate

There are I inputs, J hidden neurons and K output neurons. The weights of the hidden layer is updated using the equation

$$v_{ij}(n+1) = v_{ij}(n) + \eta * \delta_j(n) * x_i(n) \quad (1)$$

δ_j is the error signal produced by the j[th] hidden neuron.

The weights of the output layer is updated using the equation

$$w_{jk}(n+1) = w_{jk}(n) + \eta * \delta_k(n) * y_j(n) \quad (2)$$

δ_k is the error signal produced by the k[th] output neuron.

$$\delta_k(n+1) = (d_k - o_k) * (1 - o_k) * o_k \quad (3)$$

Using δ_k , we can calculate δ_j as follows

$$\delta_j^{'} = \sum_{k=0}^{K} \delta_k * w_{jk} \quad (4)$$

$$\delta_j = (1 - y_j) * y_j * \delta_j^{'} \quad (5)$$

So, for any input, we find the output of the hidden neurons, and then the output of the output layer neurons. The outputs in each layer are computed using the sigmoid function. The weights of each hidden and output layer neurons are updated using the above equations. The error signals of the hidden neurons are back propagated from the output layer to the hidden layer. This process is repeated for the next input-output pattern and so on until the error is below a prespecified threshold. We used the minimization of the squared error cost function:

$$E = \frac{1}{2}\sum_{k=0}^{K}(d_k - o_k)^2 \quad (6)$$

C. *Fuzzy Logic System*

Fuzzy logic is a problem solving methodology that takes imprecise inputs and outputs crisp information.

The basic steps in Fuzzy system are:

1. Fuzzification

a) First we define the input and output variables of the fuzzy system.

b) Create the fuzzy rules of the rule base.

2. Rule Evaluation

a) Find the degree of membership of the linguistic values of the input variables using a membership function.

3. Defuzzification

a) Calculate the weights of the set of fuzzy rules using the degree of membership.

b) Calculate the crisp or actual output by using the weights.

D. Genetic Algorithms

A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems.

The basic steps in GA are:

1. Initialisation

a) In the beginning, the GA randomly creates a population of complete bit strings (solutions). 1 in the bit string indicates the presence of rule and 0 indicates the absence of rule.

b) Give the definition of the fitness.

2. Evaluation

a) For each solution (bit string), value of fitness is computed of the bits with value 1. We then find the total fitness of the bit string by summing up the fitness of each bit with value 1.

3. Selection

a) Choose a set of bit string whose fitness value is greater than some specific number.

4. Reproduction

a) The population is modified using three operators namely selection, crossover and mutation.

b) This process (Evaluation, Selection, Reproduction) is repeated for many generations and we choose a set of bit string whose fitness value is greater than some specific number.

E. Integrated Fuzzy Logic System and Genetic Algorithm

1. Fuzzification of fuzzy system

   We define the input and output variables and then we create the fuzzy rules of the rule base of the fuzzy system.

2. Initialisation of GA

   Encode the rules into one bit string of 0's and 1's by randomly creating these bit strings (solutions). We then define the fitness equation. 1 in the bit string indicates the presence of rule and is decoded into fuzzy rules.

3. Evaluation, Selection and Reproduction of GA

   The value of fitness is computed for each bit string and then we choose a set of bit string whose fitness value is greater than some specific number. The population is modified and the process is repeated for many generations. With this, we finally find an optimal set of rules for the fuzzy system which is the 1's in a bit string with largest fitness value.

4. Rule Evaluation and Defuzzification procedure of fuzzy system.

We find the degree of membership of the linguistic values of the input variables and then calculate the weights of the set of fuzzy rules. Finally we calculate the crisp or actual output by using the weights.

III. The Motion Planning Algorithm

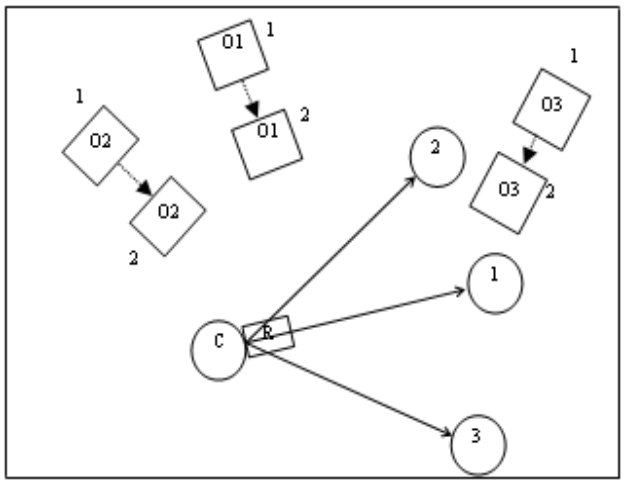

Figure 2. Robot workspace with dynamic obstacles

In Figure 2, R is the robot and c is the current position of the robot. The path or direction C1, C2 and C3 means the path from C to 1, C to 2 and C to 3 respectively as in Figure 2. The robot tries to choose a direction from the three directions C1, C2 and C3. O1, O2 and O3 are moving obstacles with some velocities V1, V2 and V3. The robot can gather local information about the obstacles within its sensing range. By recording the positions of obstacles in two successive time intervals, the speed and directions of the obstacles can be determined. The neural network will learn a set of target outputs (desired outputs) for a given set of input patterns. The input pattern is described by the time taken by obstacles to reach the robot paths. Using the neural network, the robot will choose a path from the set of paths to move ahead towards the goal position.

Initially a direction is chosen which is perpendicular to the goal position. The robot always tries to move straight to the goal position if possible from any position in the workspace. Some other directions are also chosen above and below this perpendicular direction for the robot to move ahead. At each step of neural network processing, the robot moves ahead a distance of D unit (which is a constant) in one of the direction from the set of directions.

Definition 1: *Critical obstacle*

The obstacle reaching a path taking minimum time but greater than a value *d* is chosen as the *critical obstacle* for the path for that motion.

Definition 2: *Critical path*

If an obstacle is the critical obstacle for a path, then that path becomes the *critical path* for that robot motion.

A. Velocities of the moving obstacles

As in Figure 2, the obstacles O1, O2 and O3 are at initial position indicated by label 1. We calculate the positions of the obstacles at their initial positions at time t1. At next successive time interval t2, the positions of the obstacles (indicated by label 2 in Figure 2) is calculated again. So, we know the displacement and time of the moving obstacles. With this information we calculate the velocities of the obstacles. The obstacles move with a constant velocity. The velocities of the obstacles can be different or same.

At time t1 the obstacles O1, O2 and O3 are at position (x1, y1), (x2, y2) and (x3, y3) respectively. At time t2 they are at position (a1, b1), (a2, b2) and (a3, b3). With these, we calculate the velocities V1, V2 and V3 of the obstacles O1, O2 and O3.

B. Choosing path by the neural network

We will choose one *critical obstacle* for each path for a robot motion. Let the critical obstacle for the path C1, C2 and C3 be O3, O1 and O1. The distance of the critical obstacles O3 and O1 from its initial position to its critical robot path be d1, d2 and d3. The time taken by the critical obstacles to reach its critical robot path will form the input pattern to the neural network. According to Figure 3, the input pattern will be the set $\{t_{o3c1}, t_{o1c2}, t_{o1c3}\}$. The desired output pattern will be {1, 1, 1} if a motion is possible in the path C1, C2 and C3. If a motion is not possible in any of the path C1 or C2 or C3, then we assigned a value of 0 according to the path where motion is not possible. The value 0 will indicate that a motion is not possible in that path.

The actual output is determined using the sigmoid function. During the learning process, both the weights of the connections from the input-layer I to the hidden-layer J, v_{ij} and those from the hidden-layer J to the output-layer K, w_{jk} should be adjusted to minimize the difference between the actual output and the desired output pattern. This is done using the weight-change equations for v_{ij} and w_{jk} given in Section II, B.

Finally the robot chooses a direction associated with the output value of 1 in the output neuron layer. The robot will always choose the middle path perpendicular to the destination if the output value associated with the middle path is 1. If the output value associated with the middle path is 0 and that with left and right path is 1, then the robot will choose the path from left and right path. In this case the robot will choose the one (left or right path) which is closer to the global destination. In this way we get the near-optimal robot path reaching the destination position. The robot chooses the path which operates first if the paths (without middle path) to the destination have the same length whose output value is 1. In this way the neural network is trained so that the robot thus chooses a direction from the set of directions and moves D unit ahead.

C. *Velocity calculation of the robot to avoid collision*

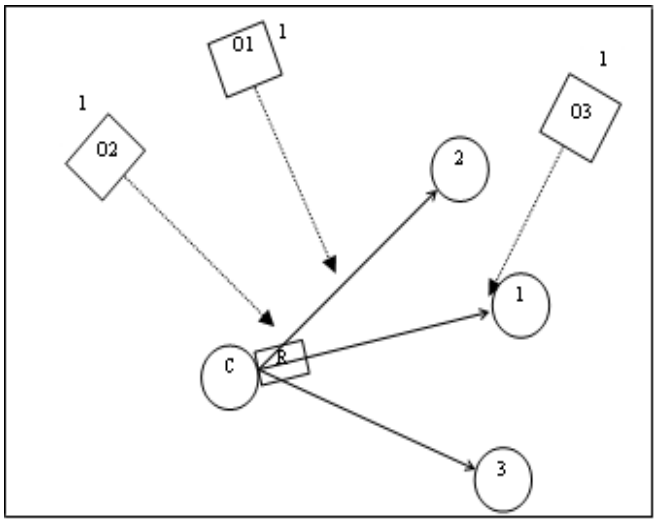

Figure 3. Obstacles reaching robot's path

After choosing a path by the neural network, we compute a velocity Vr1 for the robot that will avoid the moving obstacles and move to reach a local destination labeled as 1, 2, 3. According to Figure 3, the input pattern is the set $\{t_{o3c1}, t_{o1c2}, t_{o1c3}\}$ which is the time taken by the critical obstacles to reach its critical robot path. The distance from C to 1 or 2 or 3 as in Figure 3 is D. The value of Vr1 is calculated as $Vr1 = D / t_r$ where t_r is the time taken by the mobile robot to reach the local destination. The three conditions to calculate the time taken t_r with the above input pattern example are as follows:

1. If path C1 is chosen, then $t_r < t_{o3c1}$ and $t_r > d$.
2. If path C2 is chosen, then $t_r < t_{o1c2}$ and $t_r > d$.
3. If path C3 is chosen, then $t_r < t_{o1c3}$ and $t_r > d$.

According to this the robot moves ahead with velocity Vr1 in the path chosen by the neural network.

IV. Integrated Fuzzy Logic System and Genetic Algorithm for Obstacle avoidance

If all the directions of the robot from a particular position are blocked by static obstacles, or by dynamic obstacles coming straight in the path of the robot, then we apply the fuzzy logic. The blockage by moving obstacle coming straight in the path of the robot is like a dead-lock situation. The same logic is applied for static or dynamic obstacles or combination of both. As in Figure 4, when the robot is at position C, all the three paths are blocked by obstacles and cannot go ahead further. In such situation the fuzzy logic and GA is applied.

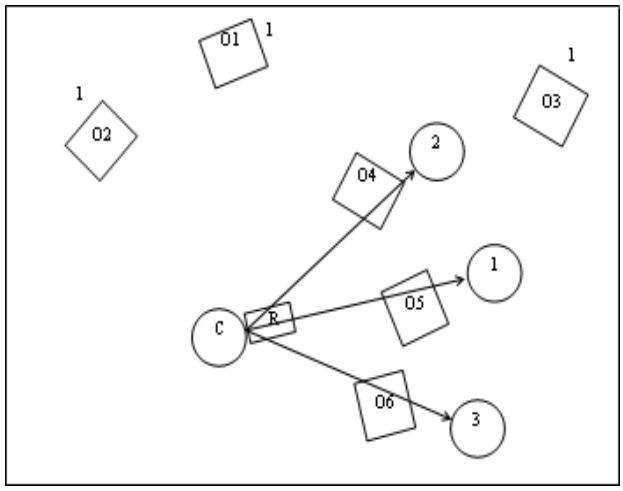

Figure 4. Situation that use fuzzy logic and genetic algorithm

A. *The input and output variables*

In Figure 4, O1, O2 and O3 are dynamic obstacles and O4, O5 and O6 are static obstacles. The path or direction C1, C2 and C3 means the path from C to 1, C to 2 and C to 3 respectively as in Figure 4. The dynamic obstacles are at initial position as indicated by label 1 and are heading towards the direction of the arrow. First we define the input and output variables of the fuzzy system. The input variables are:

1. *angle* with linguistic values *small (S), large (L).*

This variable *angle* is the angle between line joining the robot centre and local destination of the middle path and the line joining the robot centre and extreme edge of the obstacle blocking the middle path.

2. *distance* with linguistic values *near (N), far (F).*

This variable *distance* is the distance between the robot and the obstacle blocking the middle path.

3. *left_obstacle_dist* with linguistic values *near (N), far (F).*

This variable is the distance (in y-axis) from the middle obstacle to the nearest obstacle on the left side blocking the path.

4. *right_obstacle_dist* with linguistic values *near (N), far (F).*

This variable is the distance (in y-axis) from the middle obstacle to the nearest obstacle on the

right side blocking the path.

5. *time1* with linguistic values *small (S), big (B).*

This variable is the time taken by the critical obstacle on reaching path C2.

6. *time2* with linguistic values *small (S), big (B).*

This variable is the time taken by the critical obstacle on reaching path C3.

The output variable is:

1. *adjustment angle* with linguistic values *normal_left (NL), big_left (BL), normal_right (NR), big_right (BR).*

This variable *adjustment angle* is the adjustment angle of the robot on left or right side of the middle obstacle to avoid possible collisions with obstacles.

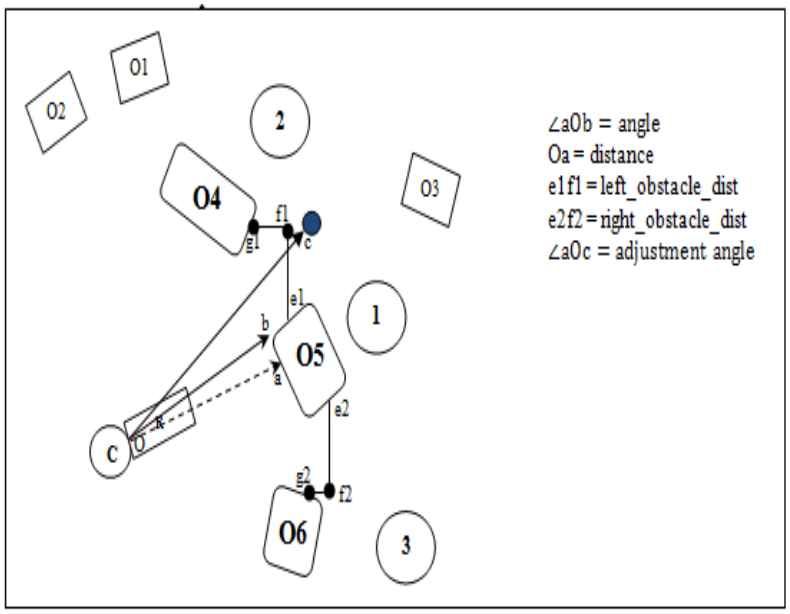

Figure 5. Avoiding obstacles

In Figure 5, the robot R is blocked by obstacles O5, O4 and O6 in three directions with local destination positions 1, 2 and 3. The robot avoids the obstacles when moving from position C. In Figure 5, angle, distance, left_obstacle_dist, right_obstacle_dist, time1 and time2 (not labeled in the figure) are the input variables and adjustment angle is the output variable. From the figure we can know that ∠a0b is the input variable angle, Oa is the distance, e1f1 is the left_obstacle_dist,

e2f2 is the right_obstacle_dist and ∠aOc is the adjustment angle. The rule based consists of 64 rules of the form:

if(angle==S) and (distance==F) and (left_obstacle_dist==F) and (right_obstacle_dist==F) and (time1==B) and (time2==S) then adjustment angle = NL

B. *Operation with Genetic Algorithm*

We encode the rules into one bit string of 0's and 1's. One bit string contains 64 bits. The coded bit strings of the fuzzy rules may look like as follows:

1001001010001000011100011110001100110001111110000011110011000011

0000001010001111111100000000001100111111100000001111100000001111

.

.

1000001010001000000000011111111001011100000001110000000000011000

1 in the bit string indicates the presence of rule and 0 indicates the absence of rule. All the fuzzy rules are defined before the GA operation begins. The 1's in a bit string are decoded into fuzzy rules (according to its position in the array) as follows:

if(*angle == S*) and (*distance == N*) and (*left_obstacle_dist == N*) and (*right_obstacle_dist == N*) and (*time1 == S*) and (*time2 == S*) then ***adjustment angle*** = *BL*

.

.

if(*angle == L*) and (*distance == F*) and (*left_obstacle_dist == F*) and (*right_obstacle_dist == F*) and (*time1 == B*) and (*time2 == B*) then ***adjustment angle*** = *NR*

In the beginning, the GA randomly creates a population of complete bit strings. Each of these bit string is decoded into fuzzy rules. Each solution (bit string) is assigned a value of fitness. After each solution in the population is assigned a fitness value, the population is modified using three operators namely selection, crossover and mutation. This process is repeated for

many generations and we choose a set of fuzzy rules whose fitness value is greater than some specific number.

The fitness value, fit is given by the equation:

$$fit = \frac{fit1}{fit2} \tag{7}$$

The value of $fit1$ equals 1 if the path in the direction of the adjustment angle is obstacle free up to D unit ahead otherwise the value is 0 indicating blockage by obstacles.

$$fit2 = \sum_{i=0}^{1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{8}$$

$fit2$ is the distance from the current position of the robot to the global destination through the collision avoided path. (x_0, y_0), (x_1, y_1), (x_2, y_2) are the co-ordinates of the current position, the local destination and the global destination of the robot. This makes the path optimized for minimum distance to reach the destination. For a bit string, we calculate the fitness of the bits with value 1. We then find the total fitness of the bit string by summing up the fitness of each bit with value 1. We then choose the bit string with largest fitness value and the 1's in the string becomes the optimal set of rules.

After the GA finds an optimal set of rules for the fuzzy logic system, the rule evaluation and defuzzification procedure starts to get the crisp value of the adjustment angle.

C. *The degree of membership*

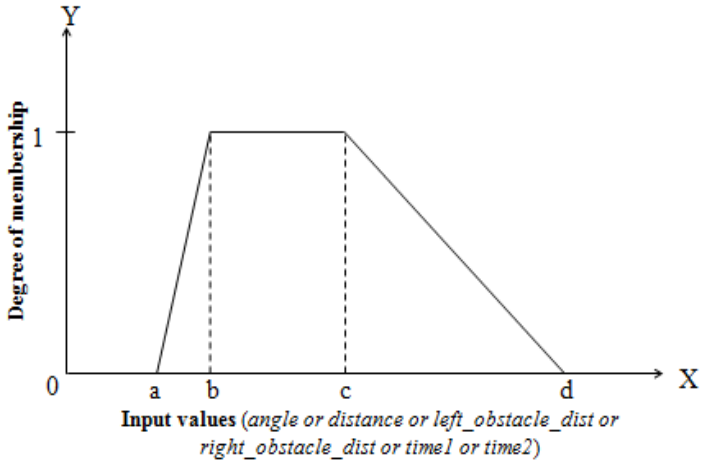

Figure 6. Trapezoidal membership function

We need to find the degree of membership of the linguistic values of the input variables of the fuzzy system in the range of 0 to 1. We use the trapezoidal membership function to find the degree of membership for the input variables with max-min composition. As shown in Figure 6, *a* to *d* is the range of values for a variable of a particular linguistic value. We take two points *b* and *c* inside the range for calculation.

$$\left.\begin{aligned} T1 &= (input - a)/(b - a) \\ T2 &= (d - input)/(d - c) \\ T1 &= Min\big(T1, Min(1, T2)\big) \\ T &= Max(T1, 0) \end{aligned}\right\} \quad (9)$$

In equation 9, input is the value of the input variable we get when the robot motion starts. The function Max and Min finds the maximum and minimum value respectively between their parameters. The final value T gives the required degree of membership of the input variable.

D. *The actual output of the Fuzzy System*

To find the crisp or actual output which is the adjustment angle, we calculate the weights of the set of rules using the degree of membership. A weight of a rule is calculated by multiplying the membership values of the input values content in that rule. The weight w of a particular rule may look like:

w = angle_D(angle_input) * distance_D(distance_input) * left_obs_dist_D(left_obs_dist input) * right_obs_dist_D(right_obs_dist input) * time1_D(time1 input) * time2_D(time2 input) (10)

The function angle_D( ), distance_D( ), left_obs_dist( ), right_obs_dist( ), time1_D( ), time2_D( ) will give the value of degree of membership. Finally, we can calculate the crisp output by using the equation:

$$Crisp\ Output = [(w_1 * v_1) + (w_2 * v_2) + \cdots + (w_n * v_n)] \ / \ [w_1 + w_2 + \cdots + w_n] \quad (11)$$

$v_1, v_2 \ldots v_n$ are the preset values determined by us. $w_1, w_2 \ldots w_n$ are the weights of the rules. n is the number of rules of the fuzzy system.

V. Simulation Results

In Figure 7, 8, 9, 10, 11, and 12, I is the initial position and G is the goal position of the robot. R is the mobile robot and the small dots are the positions the robot can reach. The black rectangular objects are the obstacles. Obstacles labeled with 1, 2 and 3 are moving obstacles and we called it O1, O2 and O3 respectively. Obstacle labeled with 4 is a static obstacle and we called it O4. Other static obstacles are not labeled. C1, C2 and C3 mean the middle, left and right path respectively. The x and y axis of the robot workspace ranges from -120 to 120. The horizontal axis is the x axis and the vertical one is the y axis. O1 is at the initial position with x axis parameter x1=-75, x2=-65 and y axis parameter y1=0, y2=15. O2 is at the initial position with x axis parameter x1=-30, x2=-40 and y axis parameter y1=10, y2=0. O3 is at the initial position with x axis parameter x1=-75, x2=-65 and y axis parameter y1=-65, y2=-50. The initial position I of the robot is x=-100, y=-50. The goal position G of the robot is x=50, y=65.

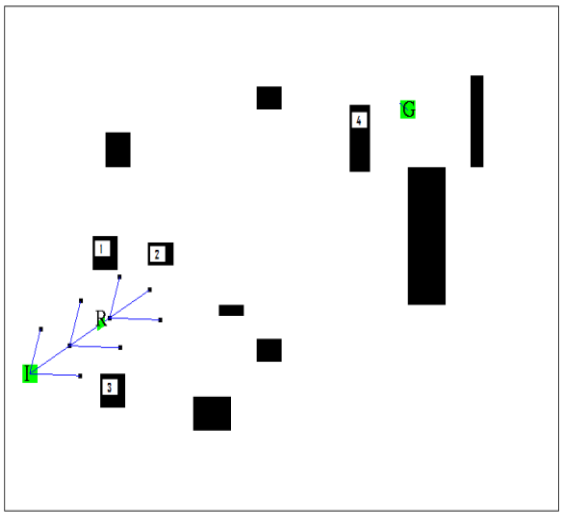

Figure 7. Showing paths for the third motion

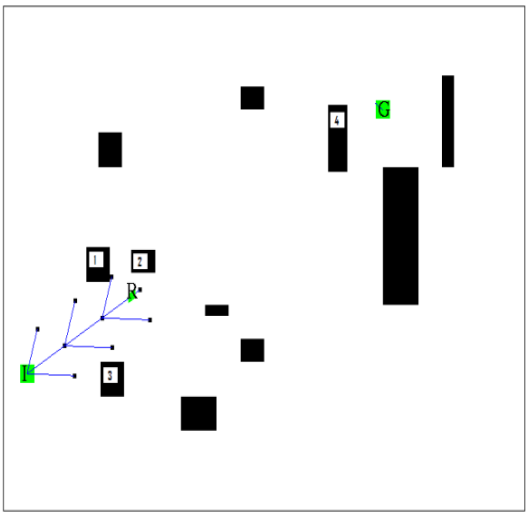

Figure 8. Third motion of the robot

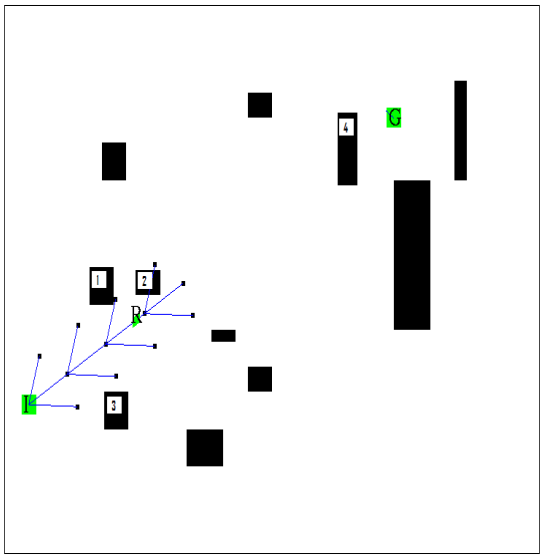

Figure 9. Ready for fourth motion

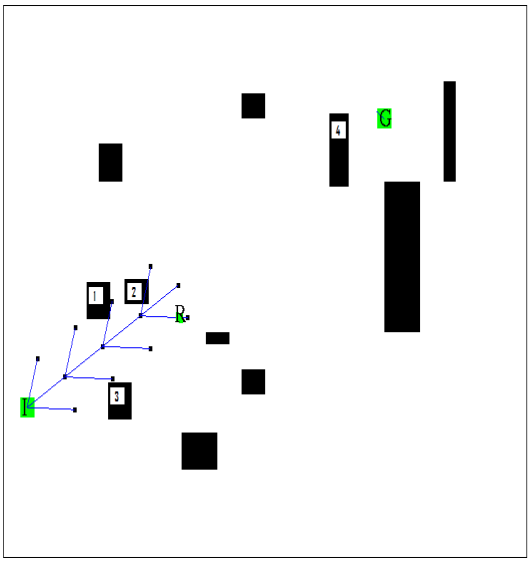

Figure 10. Fourth motion of the robot.

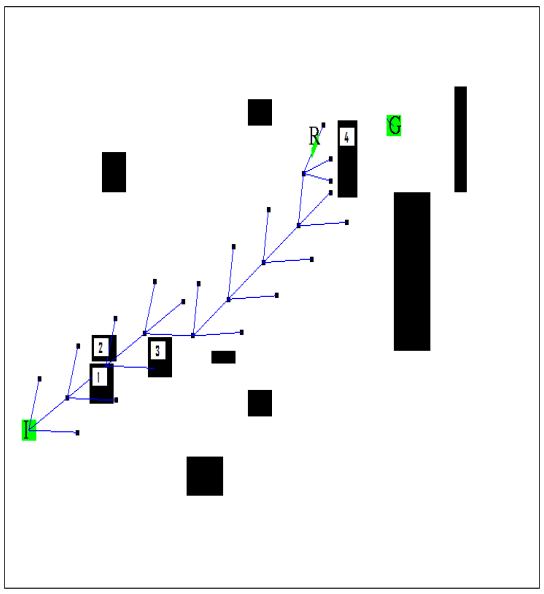

Figure 11. Robot avoiding O4 using fuzzy logic and genetic algorithm

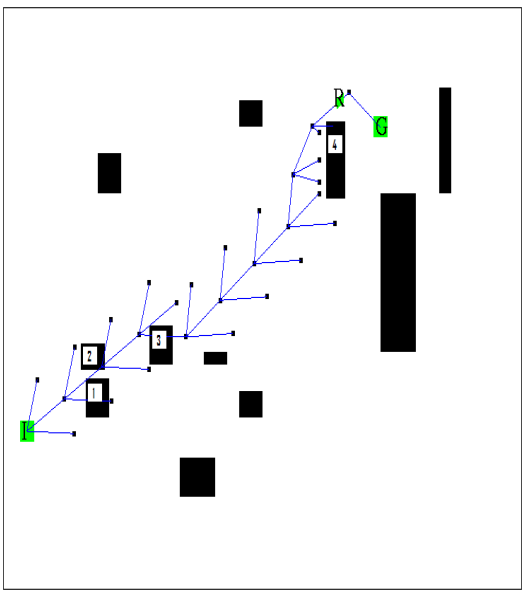

Figure 12. Robot reaching the goal position

O1 covers a distance of 5 units in the negative y direction in 1 second. O2 moves a distance of 3 units in negative x direction and 3 units in negative y direction in 1 second. O3 moves a distance of 3 units in positive x direction and 5 units in positive y direction in 1 second. The obstacle O1 moves with a velocity with 1.3 mm/sec, O2 moves with a velocity of 1.1 mm/sec and O3 moves with a velocity of 1.4 mm/sec in some direction.

VI. Evaluation of Simulation Results

We will choose one *critical obstacle* for each path for every robot motion. The time taken by the critical obstacles to reach its critical robot path will form the input pattern to the neural network for that particular robot motion. Since we have three paths, the input pattern will contain three elements.

In Figure 7, the robot is ready for the third motion. For the third motion, O1 reach C1, C2 and C3 at time 4sec, 2sec and 5sec respectively. This means t_{o1c1}=4sec, t_{o1c2}=2sec and t_{o1c3}=5sec. O2 reach C1, C2 and C3 at time t_{o2c1}=3sec, t_{o2c2}=4sec and t_{o2c3}=6sec respectively. O3 reach C1, C2 and C3 at time t_{o3c1}=5sec, t_{o3c2}=infinite sec and t_{o3c3}=4sec respectively. The infinite sec for t_{o3c2} means the obstacle O3 never touches the path C2 during the motion. So for the third motion, O2 is the critical obstacle for C1, O1 is the critical obstacle for C2 and O3 is the critical obstacle for C3. $\{t_{o2c1},t_{o1c2},t_{o3c3}\}$ will form the input set for the third motion of the robot. The robot chooses C1 for the motion as shown in Figure 8.

In Figure 9, for the fourth motion, O1 reach C1, C2 and C3 at time t_{o1c1}=infinite sec, t_{o1c2}=infinite sec and t_{o1c3}=infinite sec respectively. O2 reach C1, C2 and C3 at time t_{o2c1}=2 sec, t_{o2c2}=0 sec and t_{o2c3}=4sec respectively. O3 reach C1, C2 and C3 at time t_{o3c1}=6sec, t_{o3c2}=infinite sec and t_{o3c3}=5sec respectively. So for the fourth motion, O2 is the critical obstacle for C1, O2 is the critical obstacle for C2 and O2 is the critical obstacle for C3. $\{t_{o2c1},t_{o2c2},t_{o2c3}\}$ will form the input set for the fourth motion of the robot. The robot chooses C3 for the motion avoiding the collision with O2 as shown in Figure 10.

In Figure 11, the robot hits an obstacle (O4) in all the three directions at the ninth motion. O4 is a static obstacle. The obstacle is avoided using the fuzzy logic technique and genetic algorithm. The values of the input variables of the fuzzy system of the robot ninth motion shown in Figure 11 are: 1. angle = S 2. distance = F 3. left_obstacle_dist = N 4. right_obstacle_dist = N 5. time1 = B 6. time2 = B.

Then the GA randomly creates a population of complete bit strings (solutions). 1 in the bit string indicates the presence of rule and 0 indicates the absence of rule. For a bit string, we calculate the fitness of the bits with value 1 and then find the total fitness of the string by summing up the fitness. After many generations, we then choose the bit string with largest fitness value and the 1's in the string becomes the optimal set of rules.

We then find the degree of membership of the linguistic values of the input variables present in the rules. The degree of membership for variable *angle* with linguistic value: (1) small = 0.000000 (2) medium = 0.386667 (3) large = 0.226667. The degree of membership for variable *distance* with linguistic value: (1) near = 0.000000 (2) far = 0.750000 (3) very far = 0.500000. The degree of membership for variable *left_obstacle_dist* with linguistic value: (1) near = 1.000000 (2) far = 0.500000. The degree of membership for variable *right_obstacle_dist* with linguistic value: (1) near = 0.666667 (2) far = 0.750000. The degree of membership for variable *time1* with linguistic value: (1) small = 0.000000 (2) big = 1.000000. The degree of membership for variable *time2* with linguistic value: (1) small = 0.000000 (2) big = 1.000000.

We then calculate the weights of the set of rules using the degree of membership. Finally, using the weights, we find the actual output which is the adjustment angle to avoid the obstacle. The value of the output variable, *adjustment angle* we got is normal_left (NL). Finally, the robot reaches the goal in a collision free path as shown in Figure 12.

VII. Conclusion

In this paper, we have described a new method for solving motion planning problem in dynamic environment using neural network, fuzzy logic and genetic algorithm. The neural network is trained to choose a path from a set of *n* paths for the robot to move ahead towards the goal position. A supervised learning-rule is used for neural network using the gradient descent method. The combined fuzzy logic and genetic algorithm is used to avoid collisions when all the *n* paths are

blocked by static or dynamic obstacles or combination of both. Simulation results show that the method is efficient and gives near-optimal path reaching the target position of the mobile robot.

References

[1] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," in Internat. J. Robot. Res., vol. 10, no. 6, 1991, pp. 628–649.

[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, "Principles of Robot Motion: Theory, Algorithms, and Implementations," A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England (2005).

[3] Jur van den Berg and Mark Overmars, "Roadmap-based Motion Planning in Dynamic Environments," in IEEE Transactions on Robotics, vol. 21, no. 5, 2005, pp. 885–897.

[4] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," in International Journal of Robotics Research, vol. 21, no. 3, 2002, pp. 233–255.

[5] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in Proc. IEEE Int. Conf. on Robotics and Automation, 2006, pp. 2366–2371.

[6] P. Martin and A.P. del Pobil, "Application of artificial neural networks to the robot path planning problem," in Transactions on Information and communications Technologies, WIT press, vol 6, 1994, ISSN 1743–3517.

[7] Danica Janglová, "Neural Networks in Mobile Robot Motion," in International Journal of Advanced Robotic Systems, vol. 1, no. 1, 2004, pp. 15–22.

[8] Ulrich Roth, Marc Walker, Arne Hilmann, and Heinrich Klar, "Dynamic Path Planning with Spiking Neural Networks," in Proceedings of the International Work-Conference on Artificial and Natural Neural Networks, 1997, pp. 1355–1363.

[9] Ross Graham, Hugh McCabe, and Stephen Sheridan, "Neural Networks for Real-timePathfinding in Computer Games," in Proceedings of ITB Research Conference, 2004.

[10] Youshen Xia and Jun Wang, "A Discrete-Time Recurrent Neural Network for Shortest-Path Routing," in IEEE TRANSACTIONS ON AUTOMATIC CONTROL, vol. 45, no. 11, 2000.

[11] Panrasee Ritthipravat, Kenji Nakayama, "Obstacle Avoidance by Using Modified Hopfield Neural Network," in IC-AI, USA, 2002.

[12] C. L. Philip Chen and Farid Ahmed, "Polynomial-Neural-Networks-Based Mobile Robot Path Planning," in Applications of Artificial Intelligence, 1993.

[13] R.Glasius, A.Komoda, S.Gielen, "Neural network dynamics for path planning and obstacle avoidance," in Neural Networks(Neural netw.), 1995.

[14] V. Kanakakis, K. P. Valavanis, and N. C. Tsourveloudis, "Fuzzy-Logic Based Navigation of Underwater Vehicles," in Journal of Intelligent and Robotic Systems, vol. 40, 2004, pp. 45–88.

[15] Kamran H. Sedighi, Kaveh Ashenayi, Theodore W. Manikas, Roger L. Wainwright, Heng-Ming Tai, "Autonomous Local Path-Planning for a Mobile Robot Using a Genetic Algorithm," in Proceedings of the IEEE Congress on Evolutionary Computation 2004 (CEC'04), Portland Oregon, 2004, pp. 1338–1344.

[16] Thongam Khelchandra and Jie Huang, "Neuro-Fuzzy Approach for Optimal Motion Planning of Mobile Robot," in International Conference on Systemics, Cybernetics and Informatics (ICSCI), 2009.

[17] S S Roy and D K Pratihar, "A Genetic-Fuzzy Approach for Optimal Path-planning of a Robotic Manipulator among Static Obstacles," in Institution of Engineers (I) Jl, 84, 2003, pp. 15–22.

[18] Masoud Mohammadian and Russel J. Stonier, “Fuzzy Logic and Genetic Algorithms for Intelligent Control and Obstacle Avoidance,” in Complexity International ISSN 1320-0682, 1996.

[19] DU Xin, CHEN Hua-hua, GU Wei-kang, “Neural network and genetic algorithm based global path planning in a static environment,” in Journal of Zhejiang University SCIENCE, 2004, ISSN 1009-3095.

[20] Amir Reza Saffari Azar Alamdari, “Unknown Environment Representation for Mobile Robot Using Spiking Neural Networks,” in Transactions on Engineering, Computing and Technology, vol. 6, 2005, pp. 49–52.

[21] Dmitry V. Lebedev, Jochen J. Steil, and Helge J. Ritter, “The dynamic wave expansion neural network model for robot motion planning in time-varying environments,” in Neural Networks (Neural netw.), vol. 18, 2005, pp. 267–285.

Thongam Khelchandra received his M.S. degree in Computer Science and Engineering from The University of Aizu, Japan. Currently he is pursuing his Ph.D at the same university. His main research interest includes applications of neural networks, evolutionary algorithms, and fuzzy system mainly in the field of robotics.

Jie Huang received B. Eng. degree from Nagoya Institute of Technology, Japan, in 1985. He received M. Eng. and D. Eng. degrees from Nagoya University, in 1987 and 1991, respectively. He is currently working as an associate professor at the University of Aizu, in the Human Interface Lab, School of Computer Science and Engineering. Dr. Huang is a member of RSJ (The Robotics Society of Japan), SICE (The Society of Instrument and Control Engineers), IEICE (The Institute of Electronics, Information and Communication Engineers), ASJ (The Acoustical Society of Japan), ASA (The Acoustical Society of America), and IEEE. His main research interests are in Spatial Sound Processing, Human Audition, Auditory Scene Analysis, and Robot Sensing Systems.