Project Report on: STUDY ON DYNAMIC ROBOT PATH PLANNING

26th November, 2017

Study Project Report (EEE/INSTR F266)

Under the guidance of

Dr. Surekha Bhanot
Professor
BITS Pilani

Submitted By:

Arjun Gupta
Msc. (Hons) Mathematics and B.E. EEE 4th year student
2014B4A30754P
BITS Pilani, Pilani Campus

Rasal Kumar
Msc. (Hons) Mathematics and B.E. ENI 4th year student
2014B4A80801P
BITS Pilani, Pilani Campus

Acknowledgment

We would like to express my gratitude to my project in-charge Dr. Surekha Bhanot. We were quite eager to learn about Dynamic Path Planning and we are extremely grateful for the opportunity provided to us to work under her guidance on the topic. Her constant guidance, insights and regular discussions have been extremely valuable. She has provided us with a wonderful research experience, which has shaped our approach to tackle research problems especially on path planning. We will be eternally grateful for the guidance provided.

Contents

Abstract

The path planning for mobile bots is the issue of most concern in the field of vehicle control. The purpose of a path planner is to compute a path from the starting point to the destination point. The main concern for the path planner is to find a collision free path from start point to end point. In this report a path planning method based on artificial neural network is proposed. Data for the training has been collected using ultrasonic and inertial measurement unit. The neural network has been trained using back propagation algorithm on the data obtained. The performance of the algorithm in terms of distance from obstacles and steering angle is evaluated using MATLAB environment. Results and plots evidently show that the bot traces a desired path avoiding obstacles to reach the destination.

1 Introduction

The fourth industrial revolution is setting in the demand and the need for automation is increasing every day. Many industries can be taken as an example which have taken advantage of advancement in robotics by deploying mobile robots for their operations and research. Not only in industries but mobile robots have an application in rescue operations as well. For example, iRobot's packbots were the first of its kind used for site investigation after the 9/11 attack at U.S. World Trade Center. Such requirements make the problem of path planning for robots more complex as the complexity of surrounding is continuously varying with time. Thus, it is important to develop algorithms that cater to the problem of path planning and motion control of a robot in such dynamic environment.

Many studies have been conducted in this field to improve the accuracy and beat the human accuracy. Since we aim to beat the human accuracy to achieve the desired tasks, the inspiration is taken from the human brain and an attempt is made by researchers to replicate it in the form of mathematical network which is termed as Neural Network. In this study, we propose a model which uses a neural network (NN) trained by backpropagation algorithm[1] to move along a path as guided by the output of the NN. The NN maps the data received from the ultrasonic sensors to the output as heading angle, which is used to steer the bot, thus preventing the bot from any collision. A suitable algorithm is proposed to interpret the data received by the sensors and then finally move along the obtained optimum path. A state-space model based on the bicycle model for designing the dynamics of a vehicle is developed and ACO algorithm can be incorporated in this model to search a global collision free path which is the shortest routine through known static obstacles. Results of simulation experiments are shared in the report and are discussed in detail.

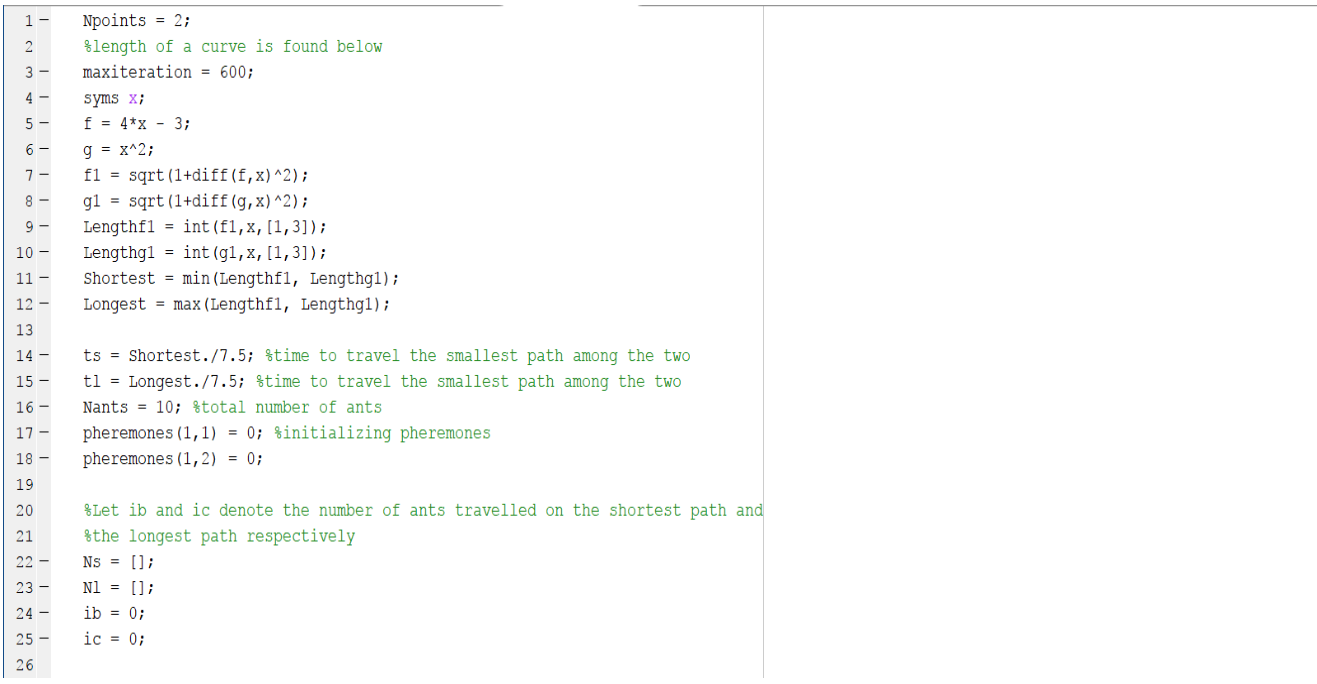
The report is structured as; section 2 discusses the ACO algorithm briefly and the MATLAB results obtained to support the theory. In section 3 a brief introduction of Artificial Neural Network and later Backpropagation algorithm is given. In section 4 we have discussed the dynamic vehicle model we have built using mathematical models for path planning controller and in section 5 we have discussed our model and the results obtained. Finally we conclude in section 6 and propose the future expectations.

2 Ant Colony Optimization

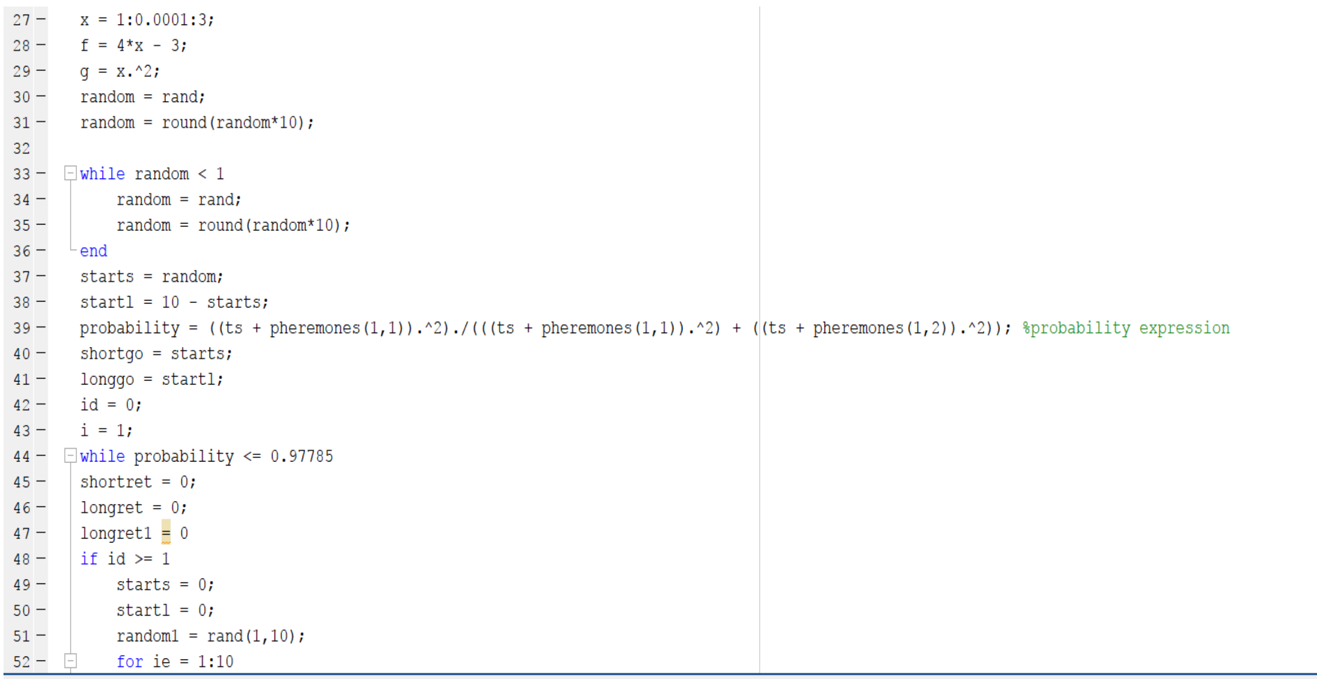
The Ant Colony Optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs [2]. Ant colony optimization (ACO) is developed after understanding and observing the foraging behavior of some ant colonies. These ants deposit pheromone on the ground in order to mark the favorable path that must be followed by other members of the colony. Ant colony optimization exploits a similar mechanism for solving optimization problems. Various experiments have found that the in the long run colony of ants converge to the shortest possible path thus resulting in global optimum path. In our model we aim to find the global optimum path by using ACO if possible. Initially to understand the basic behavior of ant colony we used MATLAB as a platform to obtain results. We built two tracks following the curve equations: 1. $y_1 = x$ and 2. $y_2 = x^2$. The probability of the decision taken by the ant at the junction is determined by the formula:

$$Probability = \frac{t_s + \phi(1,1)^2}{(t_s + \phi(1,1)^2) + (t_s + \phi(1,2)^2)} \tag{1}$$

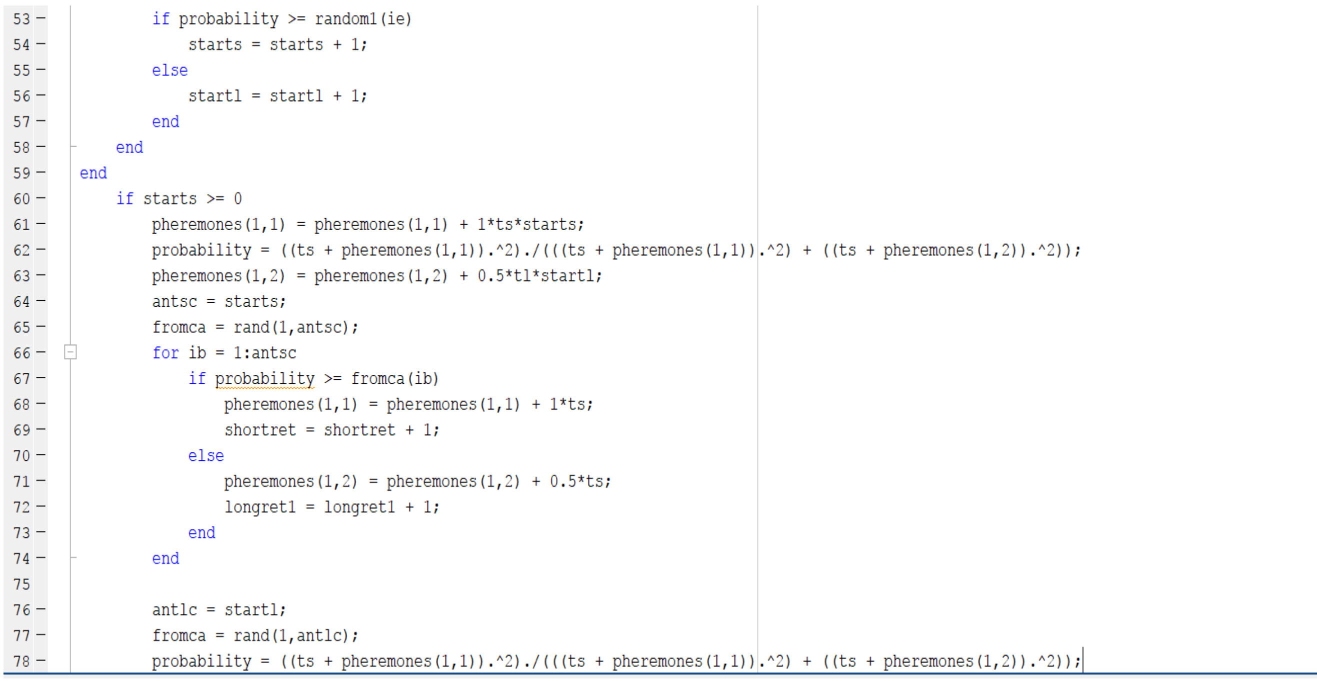
where t_s is the time taken along the shortest path, $\phi(1,1)$ denotes the pheromone level on the shortest path and $\phi(1,2)$ denotes pheromone level on the longer path. In our code we have taken 10 ants and then written a code in the MATLAB to verify the well established theory. The results obtained show that ants are initiallly random distributed among the two paths and as the number of iterations increases all the ants converge to the shorter path. Here, it is easy to determine visibly the longer path but in general it is not easy to determine the longer path thus the path to which all the ants converge finally is the shorter path. MATLAB code is shared in the figures below and the screenshots of some part of the result obtained show in fig 2, 3.



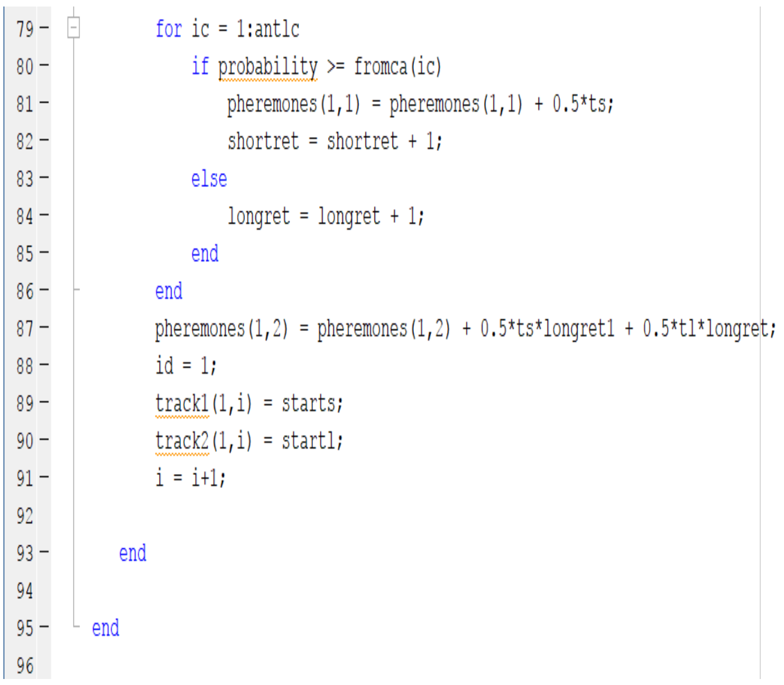
```matlab
Npoints = 2;
%length of a curve is found below
maxiteration = 600;
syms x;
f = 4*x - 3;
g = x^2;
f1 = sqrt(1+diff(f,x)^2);
g1 = sqrt(1+diff(g,x)^2);
Lengthf1 = int(f1,x,[1,3]);
Lengthg1 = int(g1,x,[1,3]);
Shortest = min(Lengthf1, Lengthg1);
Longest = max(Lengthf1, Lengthg1);

ts = Shortest./7.5; %time to travel the smallest path among the two
tl = Longest./7.5; %time to travel the smallest path among the two
Nants = 10; %total number of ants
pheremones(1,1) = 0; %initializing pheremones
pheremones(1,2) = 0;

%Let ib and ic denote the number of ants travelled on the shortest path and
%the longest path respectively
Ns = [];
Nl = [];
ib = 0;
ic = 0;

```



```matlab
x = 1:0.0001:3;
f = 4*x - 3;
g = x.^2;
random = rand;
random = round(random*10);

while random < 1
    random = rand;
    random = round(random*10);
end
starts = random;
startl = 10 - starts;
probability = ((ts + pheremones(1,1)).^2)./(((ts + pheremones(1,1)).^2) + ((ts + pheremones(1,2)).^2)); %probability expression
shortgo = starts;
longgo = startl;
id = 0;
i = 1;
while probability <= 0.97785
shortret = 0;
longret = 0;
longret1 = 0
if id >= 1
    starts = 0;
    startl = 0;
    random1 = rand(1,10);
    for ie = 1:10
```

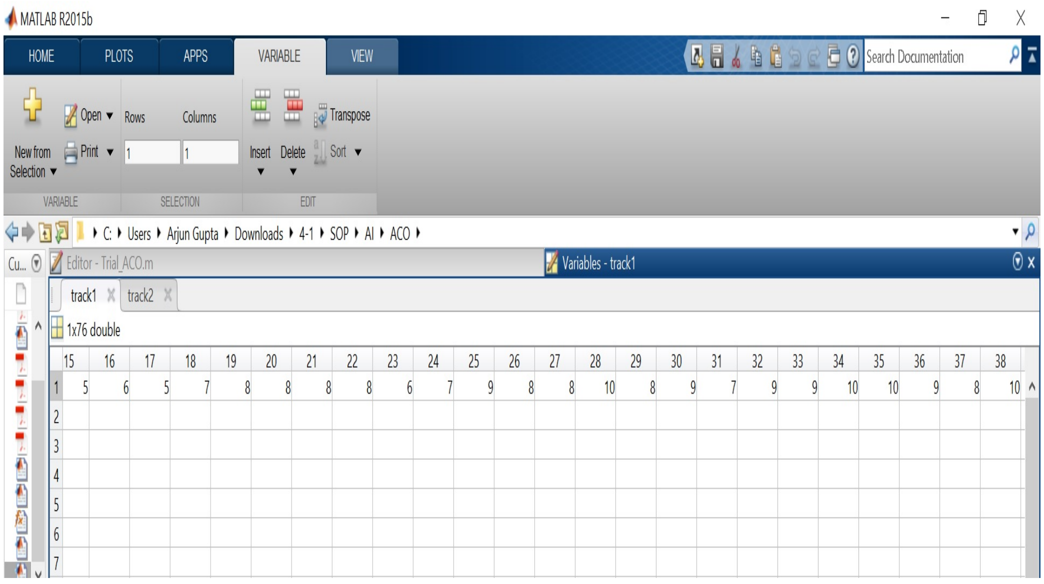


```
        if probability >= random1(ie)
            starts = starts + 1;
        else
            start1 = start1 + 1;
        end
    end
end
    if starts >= 0
        pheremones(1,1) = pheremones(1,1) + 1*ts*starts;
        probability = ((ts + pheremones(1,1)).^2)./(((ts + pheremones(1,1)).^2) + ((ts + pheremones(1,2)).^2));
        pheremones(1,2) = pheremones(1,2) + 0.5*t1*start1;
        antsc = starts;
        fromca = rand(1,antsc);
        for ib = 1:antsc
            if probability >= fromca(ib)
                pheremones(1,1) = pheremones(1,1) + 1*ts;
                shortret = shortret + 1;
            else
                pheremones(1,2) = pheremones(1,2) + 0.5*ts;
                longret1 = longret1 + 1;
            end
        end

        antlc = start1;
        fromca = rand(1,antlc);
        probability = ((ts + pheremones(1,1)).^2)./(((ts + pheremones(1,1)).^2) + ((ts + pheremones(1,2)).^2));
```



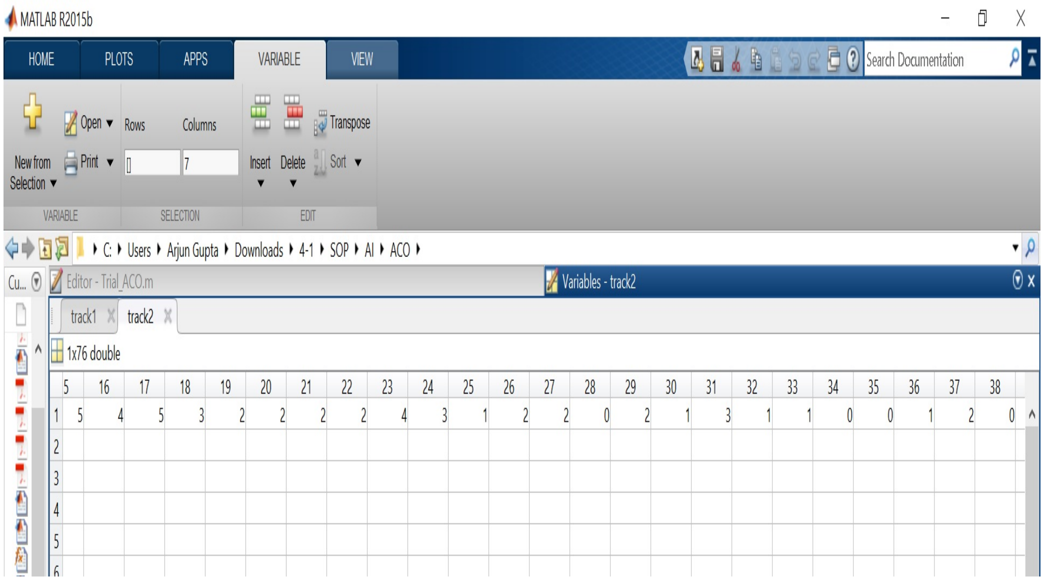
```
        for ic = 1:antlc
            if probability >= fromca(ic)
                pheremones(1,1) = pheremones(1,1) + 0.5*ts;
                shortret = shortret + 1;
            else
                longret = longret + 1;
            end
        end
        pheremones(1,2) = pheremones(1,2) + 0.5*ts*longret1 + 0.5*t1*longret;
        id = 1;
        track1(1,i) = starts;
        track2(1,i) = start1;
        i = i+1;

    end

end

```

Figure 1: The figures above show the different excerpts of MATLAB code sequentially



	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
1	5	6	5	7	8	8	8	8	6	7	9	8	8	10	8	9	7	9	9	10	10	9	8	10
2																								
3																								
4																								
5																								
6																								
7																								

Figure 2: Number of ants traveling on track 1 in different iterations



	5	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
1	5	4	5	3	2	2	2	2	4	3	1	2	2	0	2	1	3	1	1	0	0	1	2	0
2																								
3																								
4																								
5																								
6																								

Figure 3: Number of ants traveling on track 2 in different iterations

As all the ants finally start to converge on the shorter path i.e. $y_1 = x$ in our case. The result obtain verifies the the theoretical assertions claimed by the several studies.

3 Artificial Neural Network

The analogy of Artificial Neural Network (ANN) is based on biological neural networks which are by far the most efficient computing system. Similar to human brain the ANNs are a result of interconnection between the small computational units termed as nodes or neurons. Every neuron in the system is connected to some other neuron within the same system via connection link. All the operations performed in the neural network are based on the information associated with input signal and the importance of the signal is determined by the weight associated with it. The weights imparted by each neuron at different layers creates a unique map for every input to an the output signal/signals. For a neuron to fire, it is requisite that the input to the neuron crosses a certain threshold value which is determined using the activation function for that layer [3]. Refer fig. 4 for the basic architecture of ANN. As per the network topology the ANN can be classified as:

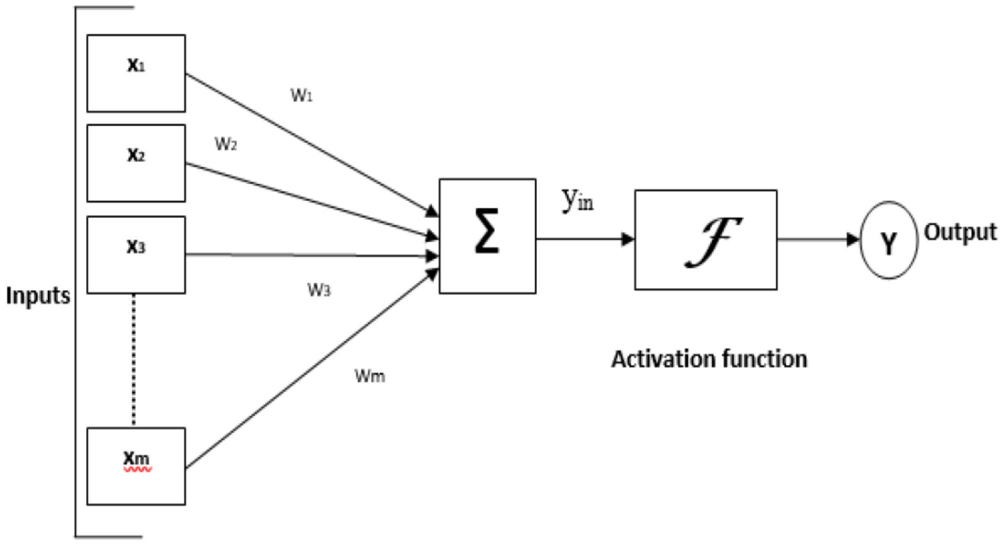


Figure 4: Basic Model of ANN

3.1 Feedforward Network

As the name suggests it is a network where signal flows in a single direction i.e. from input to output which means that there is no feedback loop in the network. Every node in a layer is connected to a neuron from the previous layer. Based on the number of layers in the network it is divided into two categories: Single Layer Feedforward Network(fig.5) and Multilayer Feedforward Network(fig.7). In Multi Layered Feedforward Network the layers between the input layer and the output layer are termed as *hidden layers*.

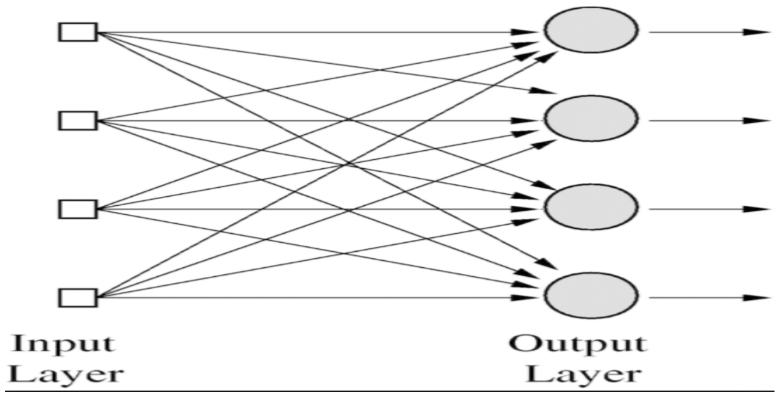


Figure 5: Architecture of Single Layer Feedforward Network

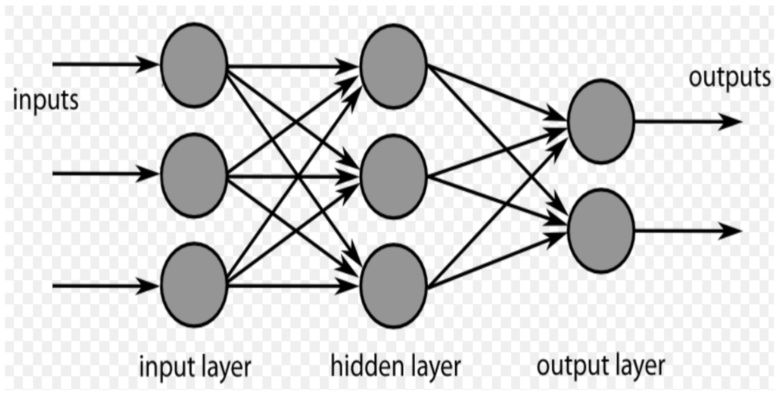


Figure 6: Architecture of Multilayer Feedforward Network

3.2 Feedback Network

Unlike the feedforward networks the signals in this topology travels in both the directions through loops (not necessarily feedback loops). Thus the system keeps changing till it reaches a state of equilibrium. Recurrent networks is one example of such topology. The following figure summarizes all the classifications with their figures.

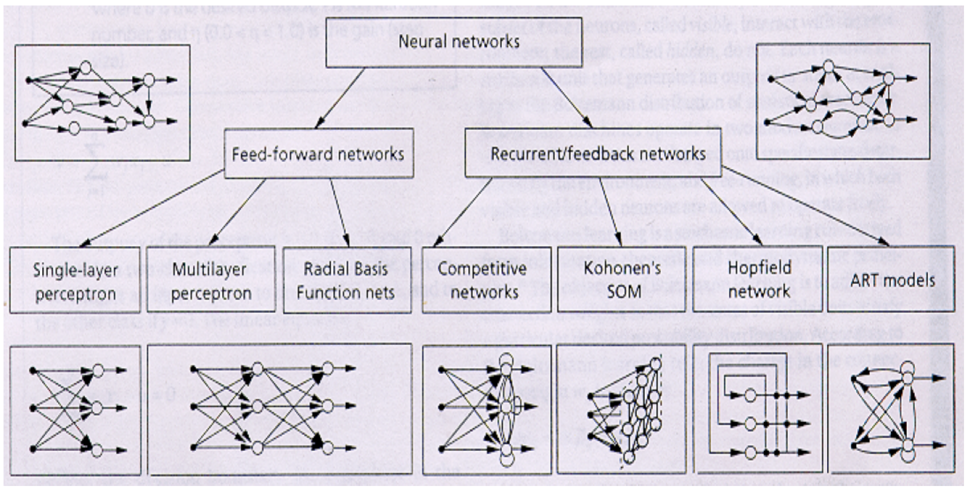


Figure 7: Classification Based on Topology

3.3 Back Propagation Neural Networks

Back Propagation Neural (BPN) is an example of multilayer neural network which is composed of an input layer, hidden layers (atleast 1) and the output layer. At the output layer, the output is compared with the desired output and the error is back propagated towards the input layer. The weights of each layer are updated corresponding to the error that is being back propagated. Refer fig. 8, it shows a BPN with three interconnected layers which have weights on them. In a system it is not preferred to change the threshold value as the stability of the system gets affected thus we add bias to the hidden layers and the output layer. The bias has weight 1 always.

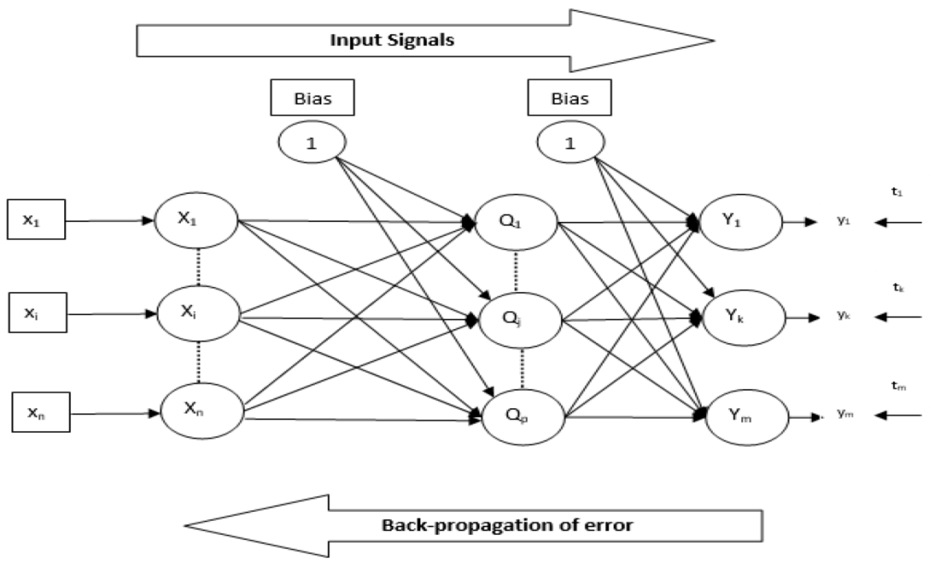


Figure 8: Architecture of BPNN

4 Proposed Model and Results

4.1 Dynamic Vehicle Model

The Vehicle Dynamics of the vehicle are modeled based on a bicycle model also known as 'Single-track' model [4]. This model utilizes the heading angle as input and gives out the x-y trajectory of the CoG of the vehicle. This model is divided into two parts :

4.1.1 The Vehicle Dynamics

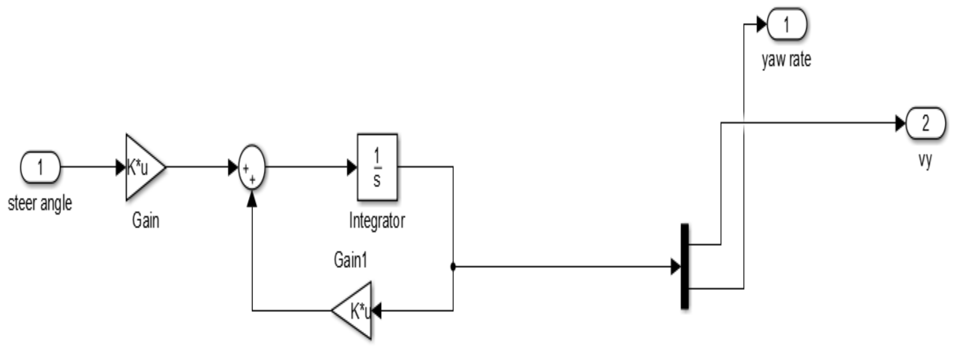


Figure 9: Vehicle Dynamics

This block diagram is based on the equation mentioned below:

$$\begin{bmatrix} \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{-C_{\alpha f}-C_{\alpha r}}{mv_x} & \frac{-a_1C_{\alpha f}+bC_{\alpha r}}{mv_x} \\ \frac{-aC_{\alpha f}+bC_{\alpha r}}{I_zv_x} & \frac{-a^2C_{\alpha f}-b^2C_{\alpha r}}{I_zv_x} \end{bmatrix} \begin{bmatrix} v_y \\ r \end{bmatrix} + \begin{bmatrix} \frac{C_{\alpha f}}{m} \\ \frac{aC_{\alpha f}}{I_z} \end{bmatrix} \delta$$

This equation takes the steering angle as input and gives out the yaw rate($\dot{r}$) and the lateral velocity as output depending upon the different parameters of the vehicle. For example the mass of the car(m), the distance of CoG from the front(a) and rear(b) axles, friction coefficients($C_{\alpha r}$ and $C_{\alpha f}$) of the tyres and yaw inertia(I).

4.1.2 The Path Planner

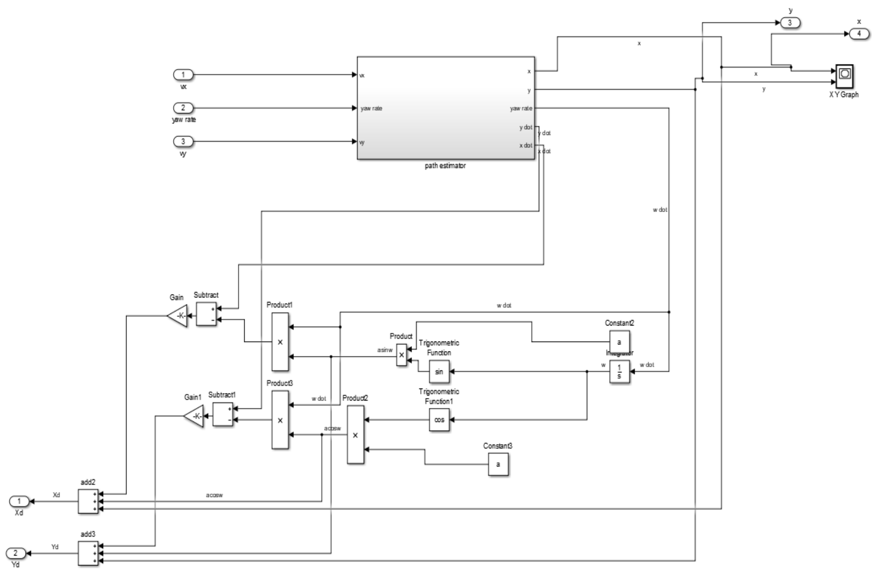


Figure 10: Path Planner

This block diagram is the one that plans the trajectory of the CoG of the vehicle based on the outputs from the previous diagram. This simulink model takes in the lateral velocity(v_x), yaw rate and the longitudnal velocity of the car as input and gives out the X and Y trajectory of the car's CoG and the trajectory of the preview point. The preview point(X_d,Y_d) represents the point on the road as seen by the driver. This point can be varied based on the type of vehicle and the driver experience. The following equations were used in making this diagram:

$$\begin{bmatrix} r = \dot{\psi} \\ \dot{x}_g = v_x cos(\theta) - v_y sin(\theta) \\ \dot{y}_g = v_x sin(\theta) + v_y cos(\theta) \end{bmatrix}$$

$$X_d = X(t) + acos(\psi) + \frac{d}{v_x}(\dot{X} - a\dot{\psi}sin(\psi)) \tag{2}$$

$$Y_d = Y(t) + asin(\psi) + \frac{d}{v_x}(\dot{Y} + a\dot{\psi}cos(\psi)) \tag{3}$$

4.2 Dataset Collection

In our model we have trained our neural network using back propagation algorithm. The working of path planning controller is verified using the data obtained from the thesis report [5]. After successful verification of the model, a new dataset was collected for training the neural network using three HC-SR04 ultrasonic sensors (as shown in fig.11) mounted on a small radio controlled car. For target data (i.e. heading angle in our case) a 6-DOF IMU (inertia measurement unit) was used. The ultrasonic sensors and the IMU unit were interfaced using Arduino and the dataset obtained was processed using MATLAB. This processed data was then further used to train the neural network.

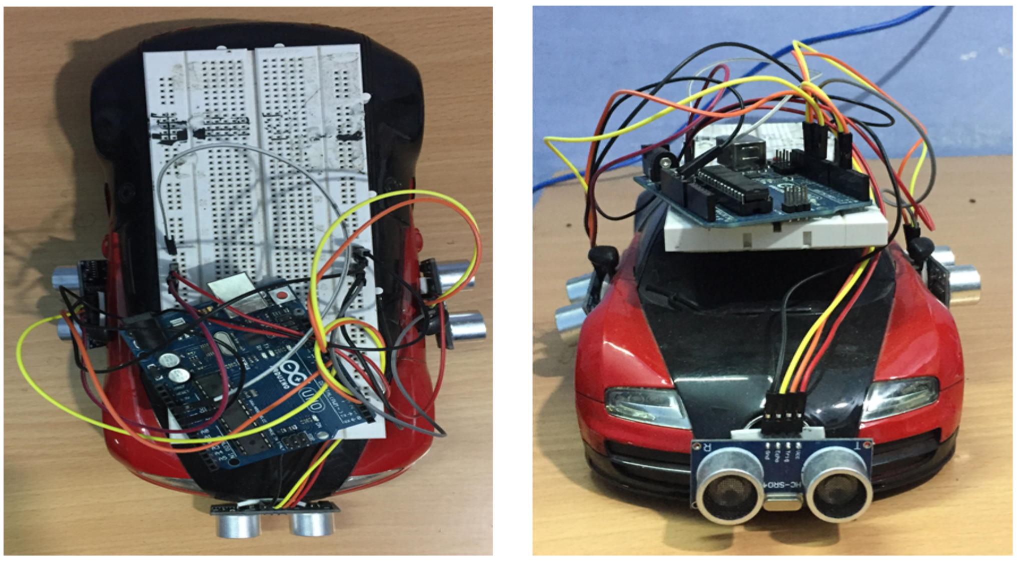


Figure 11: Ultrasonic Sensor Mounted on a Radio Controlled Car

4.3 Training the Neural Network

Training of neural network is done using the parameter values as shown in the fig 13. Number of layers for the neural network is obtained by using different number of layers and then choosing the one with optimum result. The structure of our neural network is shown in fig 14. The training process interim screenshot is shown in fig 15. The result obtained after training the neural net on the chosen data set are shown in fig 16. The data contains 4 columns where first three contain the lateral left, right and front distance of vehicle from the obstacles (as shown in fig. 12) as obtained from the sensors. The first three columns are given as the input and the fourth column contains the heading angle which is set as the target data for the neural network. We used the same data we used for training to find the trajectory of the bot.

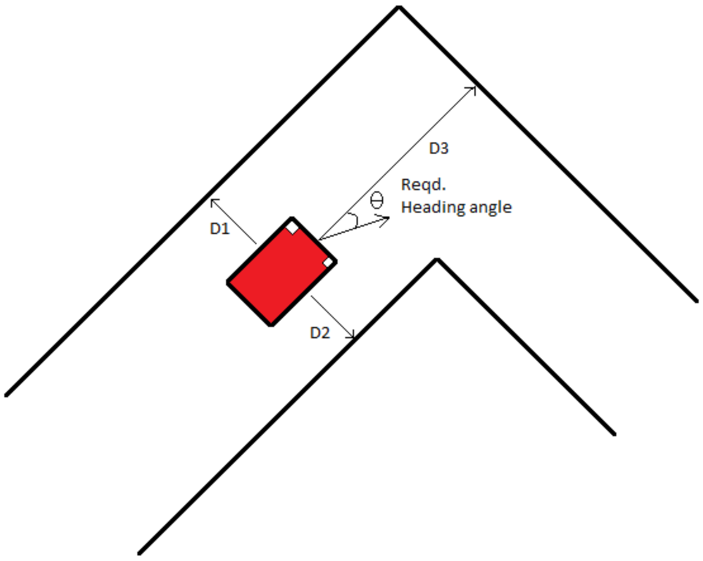
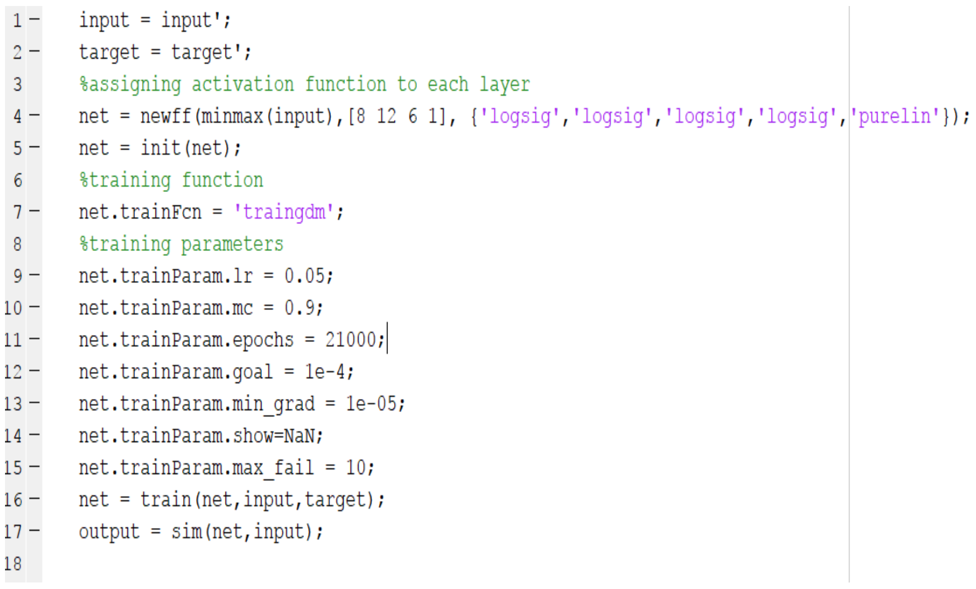


Figure 12: Diagram for Lateral Distances and Heading Angle



```
input = input';
target = target';
%assigning activation function to each layer
net = newff(minmax(input),[8 12 6 1], {'logsig','logsig','logsig','logsig','purelin'});
net = init(net);
%training function
net.trainFcn = 'traingdm';
%training parameters
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 21000;
net.trainParam.goal = 1e-4;
net.trainParam.min_grad = 1e-05;
net.trainParam.show=NaN;
net.trainParam.max_fail = 10;
net = train(net,input,target);
output = sim(net,input);
```

Figure 13: Parameters for Neural Network

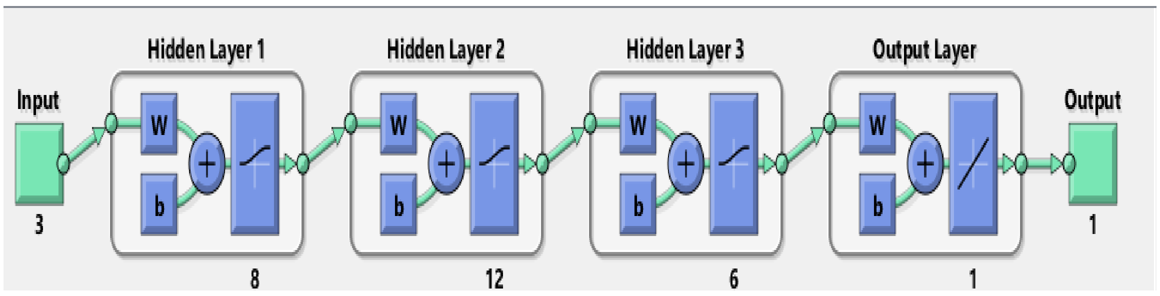
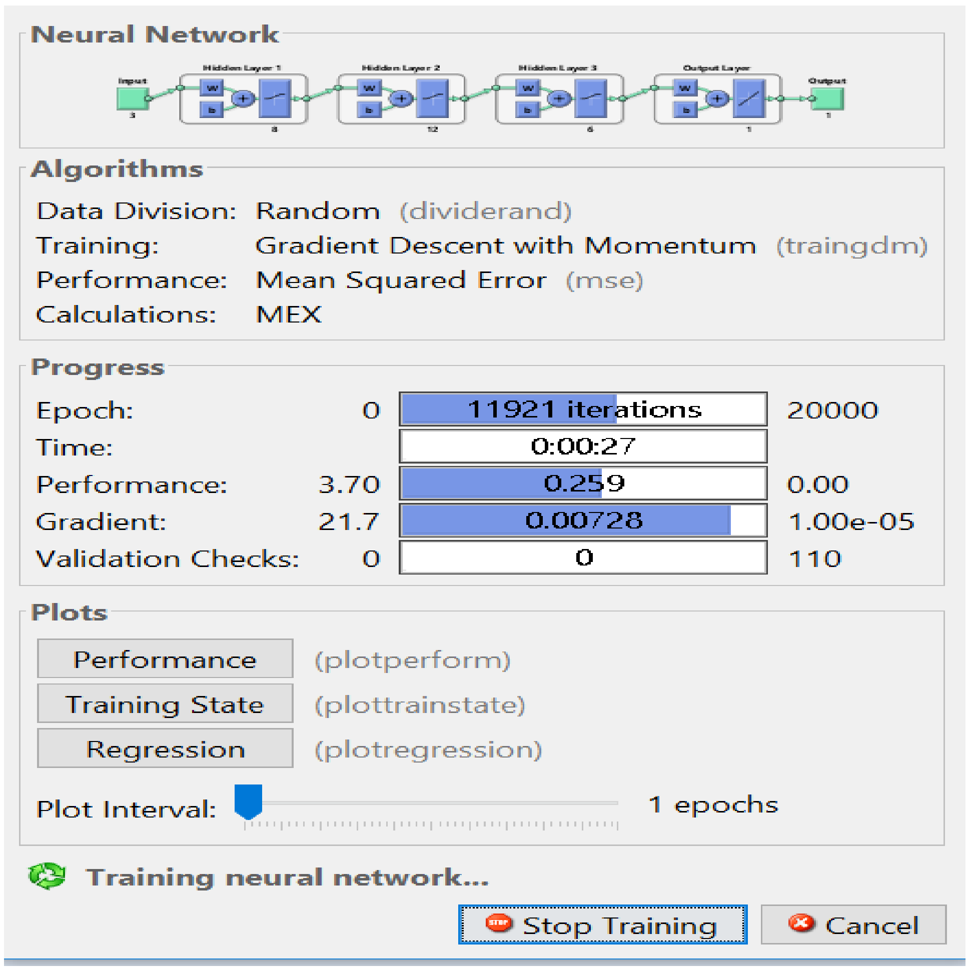


Figure 14: Neural Network for the model



Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Gradient Descent with Momentum (traingdm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

Epoch:	0	11921 iterations	20000
Time:		0:00:27	
Performance:	3.70	0.259	0.00
Gradient:	21.7	0.00728	1.00e-05
Validation Checks:	0	0	110

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

Training neural network...

Figure 15: Neural Network Training Interim

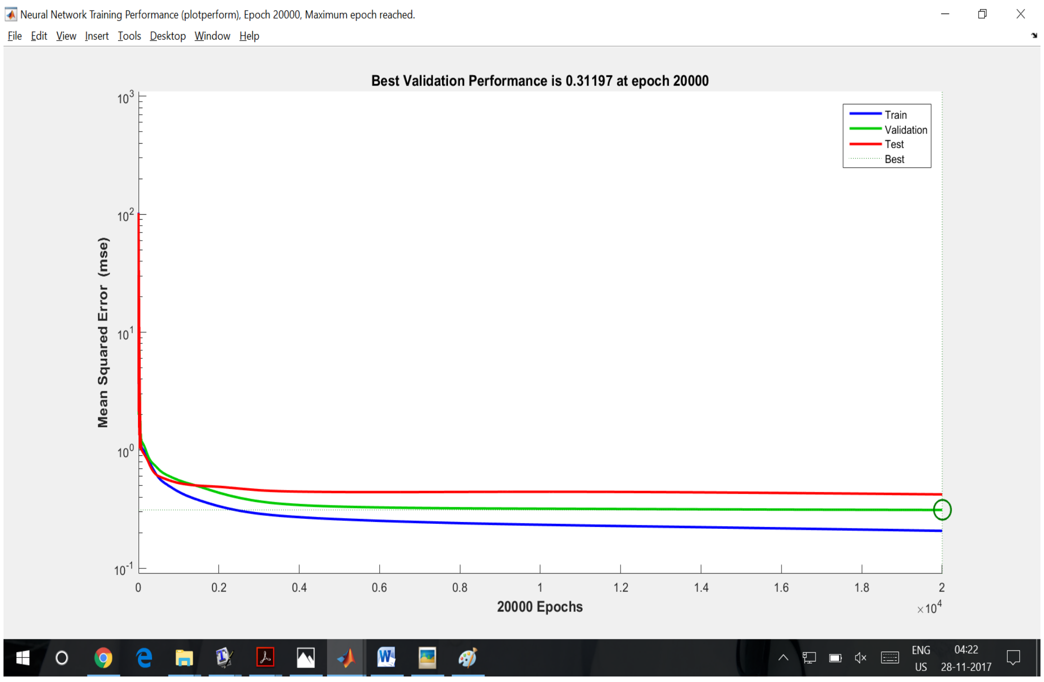


Figure 16: Training Results

4.4 Trajectory Traced

In fig.17 the path shown with red colour is the desired trajectory and the trajectory with blue colour depicts the trajectory traced by the bot. It can be clearly observed that the bot doesn't follow the exact trajectory as expected but show deviations around the corner. Such error can occur due to non-ideal setup and noise in the data.

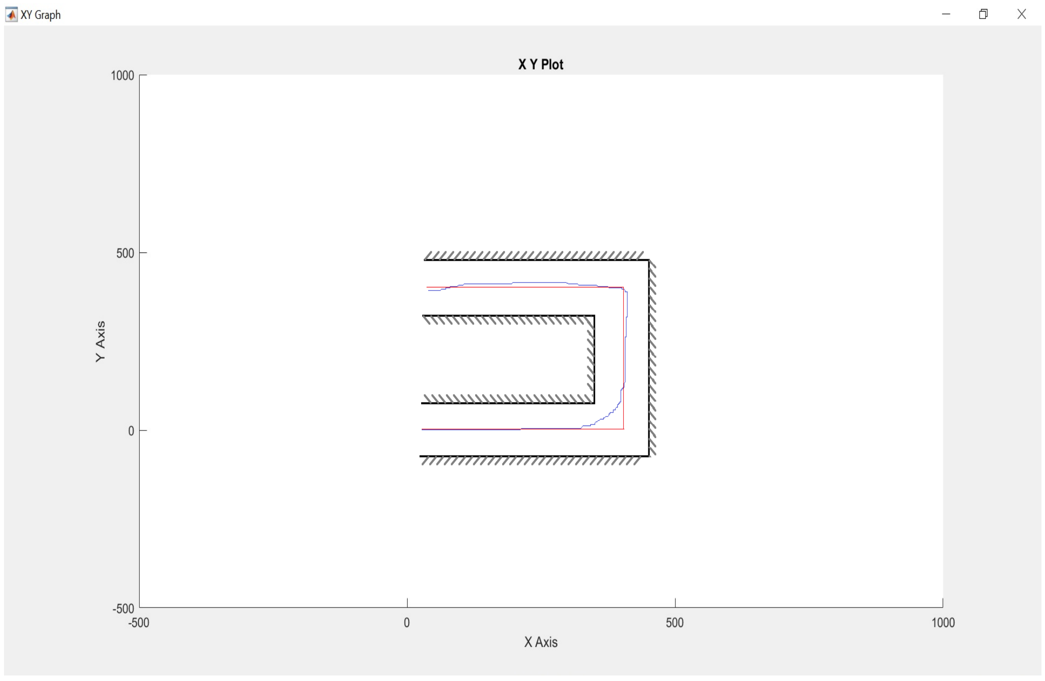


Figure 17: Trajectory as traced by the bot

5 Conclusion

Robot navigation is governed and managed by a neural network working on the data obtained. We have learned about the neural networks and used back propagation algorithm to train the network. Number of layers in the network are 4. The output of our neural network gives the heading angle to the bot. The trained neural network was successfully implemented and the bot was able to trace the desired path, avoiding obstacles, to reach the final destination. Due to non-ideal setup of the prototype a small amount of noise was observed in the data obtained from the sensors. As a result of which the bot movement was not as accurate as expected at a few instances. In an attempt to improve the accuracy of the results obtained from the noisy data we aim to implement Q-learning or Recurrent learning after learning their application feasibility on our model.

References

[1] Wahidin Wahab. Autonomous mobile robot navigation using a dual artificial neural network. pages 1–6, 2009.

[2] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[3] Christopher M Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995.

[4] Reza N Jazar. Vehicle dynamics. *Theory and Applications. Riverdale, NY: Springer Science+ Business Media*, 2008.

[5] Anshu J Behera Rohan Kumar Sabat. Automated steering design using neural network. 2011.