BITS Pilani, Pilani Campus

Evolutionary Computation Optimized Neuro-Fuzzy Controller

A Neural Networks and Fuzzy Logic Project

Parag Agrawal 2013A3PS209P

Sanuj Bhatia 2013A3PS230P

Danish Abdi 2013A3PS324P

November 16, 2015

Introduction

In the field of artificial intelligence, **neuro-fuzzy** refers to **combinations of artificial neural networks and fuzzy logic**. Neuro-fuzzy was proposed by J. S. R. Jang. Neuro-fuzzy hybridization results in an intelligent system that synergizes these two techniques by combining the human-like reasoning style of fuzzy systems with the learning and connectionist structure of neural networks. Neuro-fuzzy system incorporates the human-like reasoning style of fuzzy systems through the use of fuzzy sets and a linguistic model consisting of a set of IF-THEN fuzzy rules. The main strength of neuro-fuzzy systems is that they are universal approximators with the ability to solicit interpretable IF-THEN rules.

The strength of neuro-fuzzy systems involves two contradictory requirements in fuzzy modeling: **interpretability versus accuracy**. In practice, one of the two properties prevails. The neuro-fuzzy in fuzzy modeling research field is divided into two areas: linguistic fuzzy modeling that is focused on interpretability, mainly the **Mamdani model**; and precise fuzzy modeling that is focused on accuracy, mainly the **Takagi-Sugeno-Kang (TSK) model**. We use the TSK model in this assignment.

Evolutionary computation is a subfield of artificial intelligence that can be defined by the type of algorithms it is concerned with. These algorithms, called evolutionary algorithms, are based on adopting Darwinian principles, hence the name. Technically they belong to the family of trial and error problem solvers and can be considered global optimization methods with a **meta-heuristic or stochastic optimization character**, distinguished by the use of a population of candidate solutions (rather than just iterating over one point in the search space). We in this assignment use the stochastic approach. Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. Such processes are often inspired by biological mechanisms of evolution. In particular, we use a **Genetic Algorithm** which mimics the process of natural selection.

Problem Formulation

We attempt to design a controller based on the architecture of **Neuro-Fuzzy networks**, the parameters of which we have optimized using **Genetic Algorithm**. We base our assignment on the paper ***'Tuning of a Neuro-Fuzzy Controller by Genetic Algorithm'* by Teo Lian Seng, Marzuki Bin Khalid, and Rubiyah Yusof *(IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, Vol. 29, No. 2, April 1999).***

The controller structure is based on the **radial basis function** Neural Network with Gaussian membership functions. The GA implementation incorporates **dynamic mutation and crossover rates** for faster convergence, and a **flexible position coding strategy** is used for the tuning of the Neuro-Fuzzy Logic Controller parameters to obtain near optimal solutions to our problem.

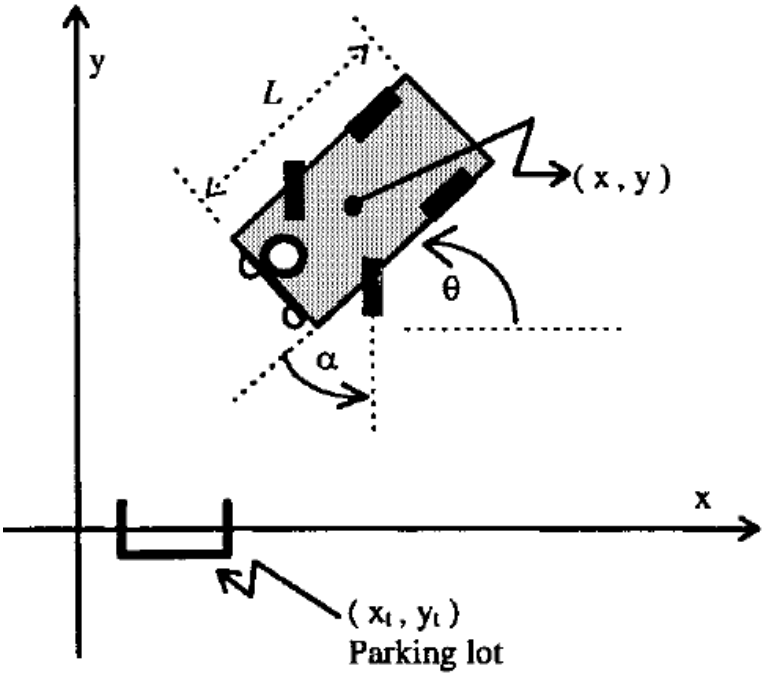
We aim to **control the trajectory of a model car** via our GA optimized controller. An **automatic car parking mechanism** has been implemented, which means that a car starting from initial position $(x_1, y1)$ travels to the final position $(x_2, y2)$ of the car parking garage using the least number of trials. Equivalently, another application is the **trajectory stabilization** of a car starting from initial position $(x_1, y1)$ (say, a parking garage) onto a predetermined line.

We use **2 inputs** to the controller, namely **error in y position** and **theta, the angle the car makes with the x-axis**, and **one output, alpha, the steering angle of the car**, sampled at instants of T=1s. Using **5 membership functions** for each of our inputs gives us 20 parameters (5 x 2 x 2) since each function has 2 parameters associated with it. A total of 5x5=25 output rules are therefore used, with each output a singleton (using the Sugeno model), which functions as the weights of the neural network. Hence, we have 20+25=**45 parameters to be optimized**. Our neural network hence consists of **2 input neurons, 25 hidden-layer neurons, and one output neuron**.

The **membership functions** are as follows: Negative Big (NB), Negative Small (NS), Zero (Z), Positive Small (PS), and Positive Big (PB).

Since we have 45 parameters, one chromosome in our GA is a 45-gene 8-bit binary encoded string, for a **total length of 45x8=360 bits**. We have used a **population size of 200**, initialized randomly, the **roulette-wheel selection** method, **two-point crossover**, and **dynamic crossover and mutation rates**. The **elitist strategy** has also been employed, which directly clones without any change the fittest member of each generation to the next. **12 trials** for every iteration were used, meaning 12 initial starting positions for the car, so that the controller is optimized for a variety of initial conditions and a larger working functionality. A **fitness function** which minimizes the total **Mean Square Error (MSE)** in each generation for all trials has been used. The **termination condition** for the program is a **maximum of 500 generations**, upon which we expect the parameters to be optimized to our problem.

The following figure shows the linear model of a car implemented, and the equations used with the model. Here, v=0.5 m/s, T=1s, L=2.6m.



$$\theta(t+1) = \theta(t) + v \times T \times \tan(\alpha)/L$$

$$y(t+1) = y(t) + v \times T \times \sin(\theta)$$

$$x(t+1) = x(t) + v \times T \times \cos(\theta)$$

Program

1. Main Script

```
clc;
close all;
clear all;

%------------------Initializing population size of 200 with random binary values------------------
popsize=200;
randompop=randi(2,popsize,360);
randompop(randompop==2)=0;
curpop=randompop;

%------------------defining x-axis as the final trajectory. Also defined the 12 trials' -------------
%------------------initial position. ---------------------------------------------------------------
fin_theta=0;
fin_y=0;
init_theta= [pi,    pi,   -pi,    -pi,   pi/2,  pi/2,   pi,  -pi,  0,   -pi/2,  pi/3,   -3*pi/4];
init_y =    [50,  -50,    50,    -50,     -1,      0,    0,    0,   0,     10,     20,        -30];
trials=12;

%-------------------------------------Normalizing the inputs to (-1,1)----------------------------------
fin_theta=fin_theta/pi;
fin_y=fin_y/50;
init_theta=init_theta/pi;
init_y=init_y/50;

%-----------------------------------------Initializing all arrays used--------------------------------------
cur_y=ones(popsize,1);
cur_theta=ones(popsize,1);
err_y=cur_y-fin_y;
fuzzified=zeros(popsize,10);
fuzzymin=zeros(5,5,popsize);
alpha=zeros(popsize,1);
aggreg=zeros(popsize,1);
sumH=zeros(popsize,1);
```

```
%------------Parameters of car: Velocity, time of sampling, and length. Maximum----------------
%------------Generations = 500-------------------------------------------------------------------
v=0.5;
t=1;
l=2.6;
Gens=0;
MaxGens=500;
PerfIndex=zeros(popsize,1);

while(Gens<=MaxGens)

   PerfIndex=PerfIndex*0;

   %-------------Decoding the chromosome to decimal values and normalizing---------------------
   [centre,width,weight]=decode(curpop,popsize);
   centre=(centre+0.5)/127.5;        %-1 to 1,0 to 1,-1 to 1
   width=width/255;
   weight=(weight+0.5)/127.5;

   for N=1:1:trials

      cur_y=(cur_y*0)+init_y(N);
      cur_theta=(cur_theta*0)+init_theta(N);

      %---------------------------We let the car run for T=300s, i.e. 300 samples----------------------
      for sample=1:1:300

         aggreg=aggreg*0;
         sumH=sumH*0;
         err_y=cur_y-fin_y;

         for i=1:1:popsize

            %-----------Flexible position coding strategy implementation. Obtain center------------
            %-----------width pairs for both inputs 1 and 2, and sort them w.r.t. centers------------
            %-----------for fuzzification of inputs variables-----------------------------------------------
            cw1=[centre(i,1:5);width(i,1:5)];
             cw2=[centre(i,6:10);width(i,6:10)];
             cw1=(sortrows(cw1.',1)).';
             cw2=(sortrows(cw2.',1)).';
```

```
  for j=1:1:5

    %--------------fuzzification of both inputs, 10 fuzzy values. 10^-8 added----------------
    %--------------to prevent division by zero-----------------------------------------------
    fuzzified(i,j)=exp(-(((cw1(1,j)-err_y(i))/(0.00000001+cw1(2,j))).^2));
    fuzzified(i,j+5)=exp(-(((cw2(1,j)-cur_theta(i))/(0.00000001+cw2(2,j))).^2));
  end
end

for i=1:1:popsize
  for j=1:1:5
    for k=1:1:5

      %-------------------Taking min (using AND) and rule firing-----------------------------
      fuzzymin(j,k,i)=min(fuzzified(i,j),fuzzified(i,k+5));
      aggreg(i)=aggreg(i) + (fuzzymin(j,k,i)*weight(i,((j-1)*5)+k));
      sumH(i)=sumH(i)+fuzzymin(j,k,i);
    end
  end
  %---------------output generated using COA aggregation method-------------------------
  alpha(i)=aggreg(i)/(0.00000001+sumH(i));
end

%------------------Updating car position after de-normalization-------------------------------
alpha=alpha*pi/3;
cur_y=cur_y*50;
cur_theta=cur_theta*pi;
for i=1:1:popsize
  cur_theta(i) = cur_theta(i) + (v*t*tan(alpha(i))/l);
  cur_y(i) = cur_y(i) + v*t*sin(cur_theta(i));
end
cur_y=cur_y/50;
cur_theta=cur_theta/pi;

err_theta=cur_theta-fin_theta;
err_y=cur_y-fin_y;
```

```
        %----------------------Evaluation of Performance index (MSE)----------------------------------
        for i=1:1:popsize
          PerfIndex(i)=PerfIndex(i) + ( (((err_theta(i)).^2)+1) * (((err_y(i)).^2)+1) *
sample/1000);
        end
      end
   end

   %---------------Function call to the GA program for creation of next generation----------------
   curpop=GA(PerfIndex,curpop,popsize,Gens,MaxGens);
   Gens=Gens+1;
end
```

2. Decode Function

```
function [ centre, width, weight ] = decode( pop,popsize )

%-------------------------------------Decodes the chromosome to decimal values--------------------
centre = zeros(popsize,10);
width = zeros(popsize,10);
weight = zeros(popsize,25);

%-----------------------1-160 are Center-width pairs. 161-360 are the weights---------------------
for i=1:1:popsize;
   for j=1:16:160;
      centre(i,floor(j/16 + 1)) =(-1)*pop(i,j)*128 + pop(i,j+1)*64 + pop(i,j+2)*32 +
pop(i,j+3)*16 + pop(i,j+4)*8 + pop(i,j+5)*4 + pop(i,j+6)*2 + pop(i,j+7);
      width(i,floor(j/16 + 1))  = pop(i,j+8)*128 + pop(i,j+9)*64 + pop(i,j+10)*32 +
pop(i,j+11)*16 + pop(i,j+12)*8 + pop(i,j+13)*4 + pop(i,j+14)*2 + pop(i,j+15);
   end

   for j=161:8:360;
      weight(i,floor((j-160)/8+1)) = (-1)*pop(i,j)*128 + pop(i,j+1)*64 + pop(i,j+2)*32 +
pop(i,j+3)*16 + pop(i,j+4)*8 + pop(i,j+5)*4 + pop(i,j+6)*2+ pop(i,j+7);
   end
end
end
```

3. Genetic Algorithm Function

```
function [ newpop ] = GA( PerfIndex,curpop,popsize,Gens,MaxGens )

rng('shuffle');
fitness=zeros(popsize,3);
totalfit=0;

%------Dynamic Crossover Rate. Evaluation of fitness using inverse of PerfIndex----------
CrossRate=exp((-0.75)*Gens/MaxGens);

for i=1:1:popsize
   fitness(i,1) = 100000/(1 + PerfIndex(i)); %f=100/(1+F);
   totalfit = totalfit + fitness(i,1);
   fitness(i,2)=i;
end

%---------Sorting fitness in ascending order, keeping indices paired with original----------
%---------individual. Find probability of each and cumulative probability for selection--------
%---------using roulette-wheel scheme---------------------------------------------------------
fitness=sortrows(fitness,1);
fitness(1,3)=fitness(1,1)/totalfit;
for i=2:1:popsize
   fitness(i,3)=fitness(i,1)/totalfit + fitness(i-1,3);
end

newpopsize=0;
newpop=zeros(popsize,360);

while(newpopsize<popsize)

 %----------------Selection of parents for crossover from the roulette-wheel----------------------
   X=rand;
   for i=1:1:popsize
      if(X<fitness(i,3))
         break;
      end;
   end;
```

```
X=rand;
for j=1:1:popsize
  if(X<fitness(j,3))
    break;
  end;
end;

%--------------if the same parent selected twice, no crossover. Selection again-----------------
if(i==j)
  continue;
end

p1=curpop(fitness(i,2),:);  %parent 1 and 2
p2=curpop(fitness(j,2),:);

%----------------------------If random number < crossover rate, crossover will happen--------
%-----------------------------otherwise parents will be cloned directly to next generation--------
if(rand<CrossRate)

  %----------------------------------Selection of two-points for crossover---------------------------
  cross1=floor(360*rand);
  cross2=floor(360*rand);

  if(cross1==0)
    cross1=1;
  end
  if(cross2==0)
    cross2=1;
  end

  %------------------------Execution of crossover between the two points-------------------------
  if(cross1>cross2)
    for i=cross2:1:cross1
      temp=p1(i);
      p1(i)=p2(i);
      p2(i)=temp;
    end
  end
```

```
    if(cross2>cross1)
      for i=cross1:1:cross2
        temp=p1(i);
        p1(i)=p2(i);
        p2(i)=temp;
      end
    end
  end
  %----------------------------Children written into next generation---------------------------------
  newpop(newpopsize+1,:)=p1;
  newpop(newpopsize+2,:)=p2;
  newpopsize=newpopsize+2;
end;

%-------------------------------------1 elite copied over directly-------------------------------------
newpop(popsize,:)=curpop(fitness(popsize,2),:);

%-----------Mutation executed gene-wise in each chromosome. Random bit flipped------------
MutRate=(exp(0.05*Gens/MaxGens))-1;
for i=1:1:(popsize-1)
  for j=1:8:360
    if(rand<=MutRate)
      X=floor(8*rand);
      if (X==8)
        X=7;
      end
      newpop(i,j+X)=1-newpop(i,j+X);
    end
  end
end
end
```

4. Optimized Network Script

```
%---------This script takes the fittest member from the execution of the Main script after 500----------
%---------Generations and puts the obtained optimized parameters into the controller. The car--------
%--------car is started from various initial positions and its x and y positions plotted and recorded---
%-------- for results and discussions. ----------------------------------------------------------------------------
clc;
close all;
clear all;
%-----------------fittest member chromosome inputted, decoded and normalized----------------
curpop=[1,0,0,0,1,1,1,1,0,0,0,1,0,0,0,0,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,0,1,0,0
,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,1,0,1,0,0,1,1,0,1,0,0,1,1,
1,0,1,1,1,1,1,0,1,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,1,0,1,0,1,1,0,1,0,0,1,0,1,0,1,1,0,0,1,0,0,1,0,0,1,1,1,
0,1,1,0,0,0,0,0,1,0,1,0,0,1,1,1,1,1,1,0,1,0,0,0,1,1,0,1,1,0,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0,1,0,0,0,1,1,1,
1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,1,0,1,0,0,1,0,1,1,1,0,1,0,0,0,1,0,1,0,0,
1,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1,0,1,1,
1,1,1,0,0,0,1,0,1,1,1,1,1,1,0,0,0,0,0,1,0,1,1,1,1,1,0,0,1,0,0,1,1,0,1,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
1,0,0,0,0,1,1,0];
[centre,width,weight]=decode(curpop,1);
centre=(centre+0.5)/127.5;        %-1 to 1,0 to 1,-1 to 1
width=width/255;
weight=(weight+0.5)/127.5;

%-------Initialize and normalize variables. A manual bias of -0.1245 applied to Input 1--------
popsize=1;
init_x=0;
fin_theta=0;
fin_y=-.1245;
init_theta=-pi;
init_y=50;

fin_theta=fin_theta/pi;
fin_y=fin_y/50;
init_theta=init_theta/pi;
init_y=init_y/50;
init_x=init_x/50;

cur_y=ones(1,301)*init_y;
cur_x=ones(1,301)*init_x;
cur_theta=ones(1,301)*init_theta;
fuzzified=zeros(1,10);
```

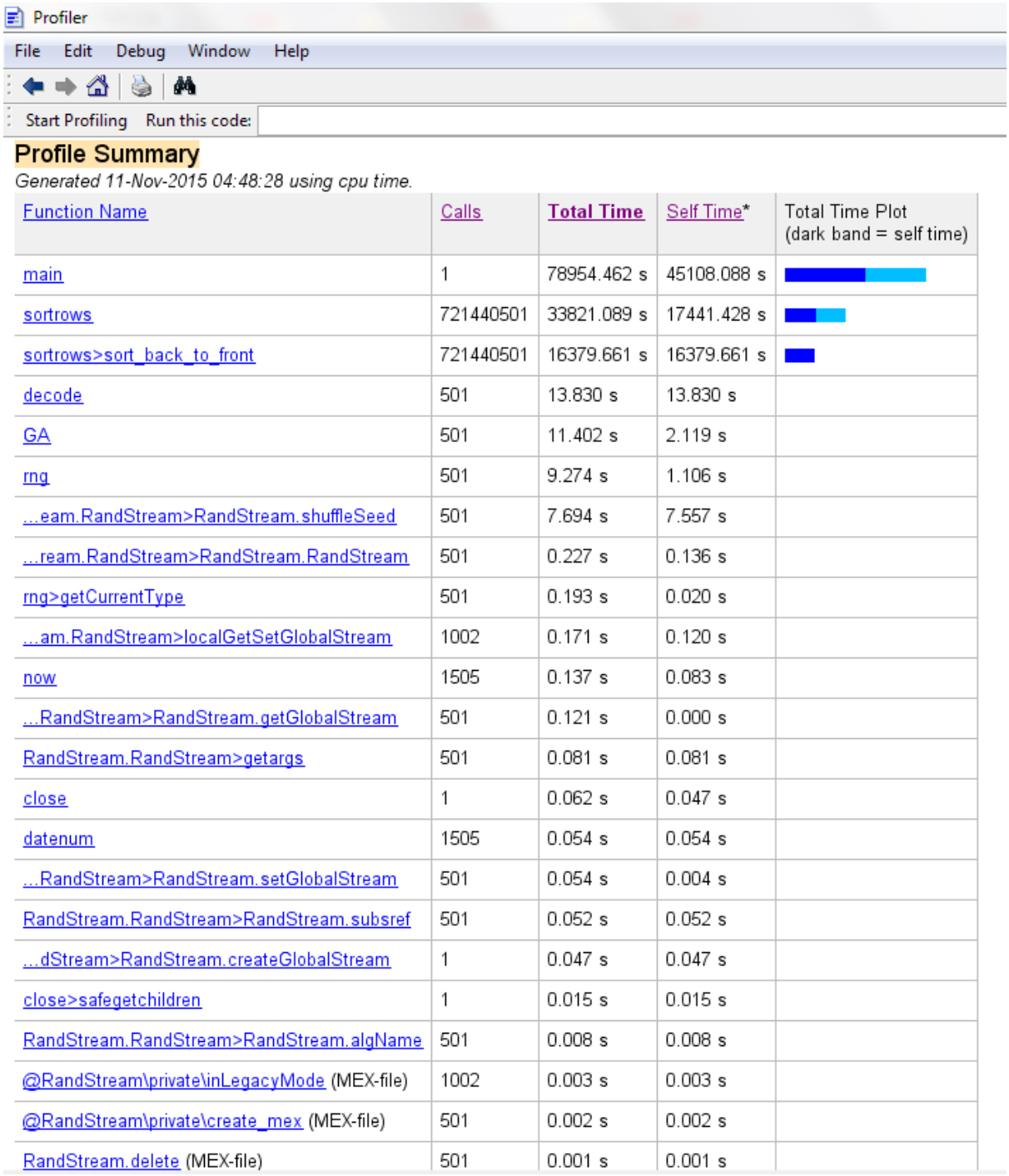
```
fuzzymin=zeros(5,5);
alpha=0;
aggreg=0;
sumH=0;
PerfIndex=0;
v=0.5;
t=1;
l=2.6;

%----------controller functioning starts in the same way as before, for T=300s----------
for sample=1:1:300

  aggreg=0;
  sumH=0;
  err_y=cur_y(sample)-fin_y;

  cw1=[centre(1:5);width(1:5)];
  cw2=[centre(6:10);width(6:10)];
  cw1=(sortrows(cw1.',1)).';
  cw2=(sortrows(cw2.',1)).';

  for j=1:1:5
    fuzzified(j)=exp(-(((cw1(1,j)-err_y)/cw1(2,j)).^2));
    fuzzified(j+5)=exp(-(((cw2(1,j)-cur_theta(sample))/cw2(2,j)).^2));
  end

  for j=1:1:5
    for k=1:1:5
      fuzzymin(j,k)=min(fuzzified(j),fuzzified(k+5));
      aggreg=aggreg + (fuzzymin(j,k)*weight(((j-1)*5)+k));
      sumH=sumH+fuzzymin(j,k);
    end
  end
  alpha=aggreg/sumH;

  alpha=alpha*pi/3;
  cur_y=cur_y*50;
  cur_x=cur_x*50;
  cur_theta=cur_theta*pi;
```

```
   cur_theta(sample+1) = cur_theta(sample) + (v*t*tan(alpha)/l);
   cur_y(sample+1) = cur_y(sample) + v*t*sin(cur_theta(sample+1));
   cur_x(sample+1) = cur_x(sample) + v*t*cos(cur_theta(sample+1));

   cur_y=cur_y/50;
   cur_x=cur_x/50;
   cur_theta=cur_theta/pi;

   err_theta=cur_theta(sample+1)-fin_theta;
   err_y=cur_y(sample+1)-fin_y;

   PerfIndex=PerfIndex + ( (((err_theta).^2)+1) * (((err_y).^2)+1) * sample/1000);
end
%-----------------End of program. Plotting X,Y coordinates of car and the axis----------------
plot(cur_x*50,cur_y*50)
hold on
hline(0)
vline(0)
hold off
```

Results and Discussions

Presented below are the results.

The Program for optimization of controller parameters was timed for its execution. It ran for approximately 22 hours on an Intel i5 Processor with 4GB RAM. Below is the profiler screenshot:

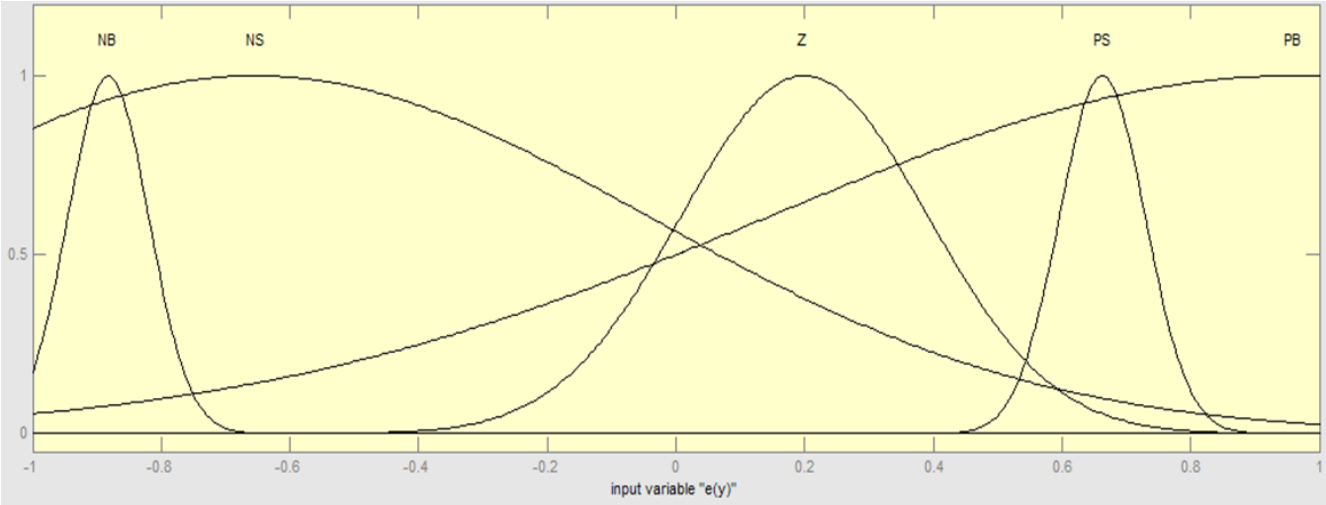


Profile Summary

Generated 11-Nov-2015 04:48:28 using cpu time.

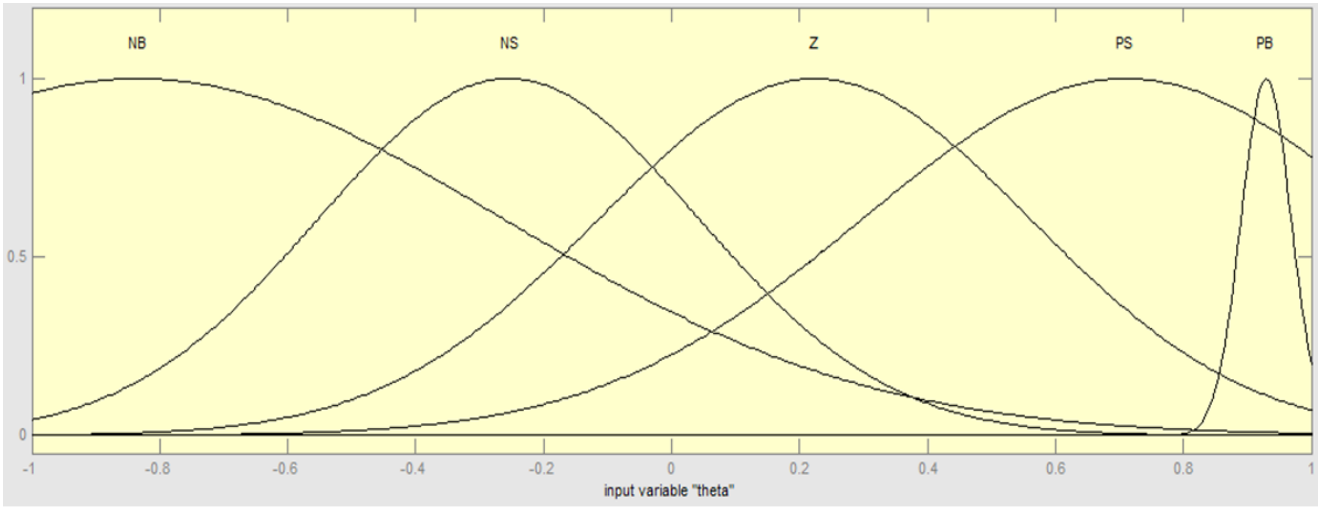
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	78954.462 s	45108.088 s	
sortrows	721440501	33821.089 s	17441.428 s	
sortrows>sort_back_to_front	721440501	16379.661 s	16379.661 s	
decode	501	13.830 s	13.830 s	
GA	501	11.402 s	2.119 s	
rng	501	9.274 s	1.106 s	
...eam.RandStream>RandStream.shuffleSeed	501	7.694 s	7.557 s	
...ream.RandStream>RandStream.RandStream	501	0.227 s	0.136 s	
rng>getCurrentType	501	0.193 s	0.020 s	
...am.RandStream>localGetSetGlobalStream	1002	0.171 s	0.120 s	
now	1505	0.137 s	0.083 s	
...RandStream>RandStream.getGlobalStream	501	0.121 s	0.000 s	
RandStream.RandStream>getargs	501	0.081 s	0.081 s	
close	1	0.062 s	0.047 s	
datenum	1505	0.054 s	0.054 s	
...RandStream>RandStream.setGlobalStream	501	0.054 s	0.004 s	
RandStream.RandStream>RandStream.subsref	501	0.052 s	0.052 s	
...dStream>RandStream.createGlobalStream	1	0.047 s	0.047 s	
close>safegetchildren	1	0.015 s	0.015 s	
RandStream.RandStream>RandStream.algName	501	0.008 s	0.008 s	
@RandStream\private\inLegacyMode (MEX-file)	1002	0.003 s	0.003 s	
@RandStream\private\create_mex (MEX-file)	501	0.002 s	0.002 s	
RandStream.delete (MEX-file)	501	0.001 s	0.001 s	

The optimized controller parameters are as follows:



Label	NB	NS	Z	PS	PB
Centre	-0.8824	-0.6549	0.2000	0.6627	0.9608
Width	0.0627	0.6118	0.1922	0.0667	0.8157

Fuzzy Membership Functions for Input 1: Error in y, e(y)



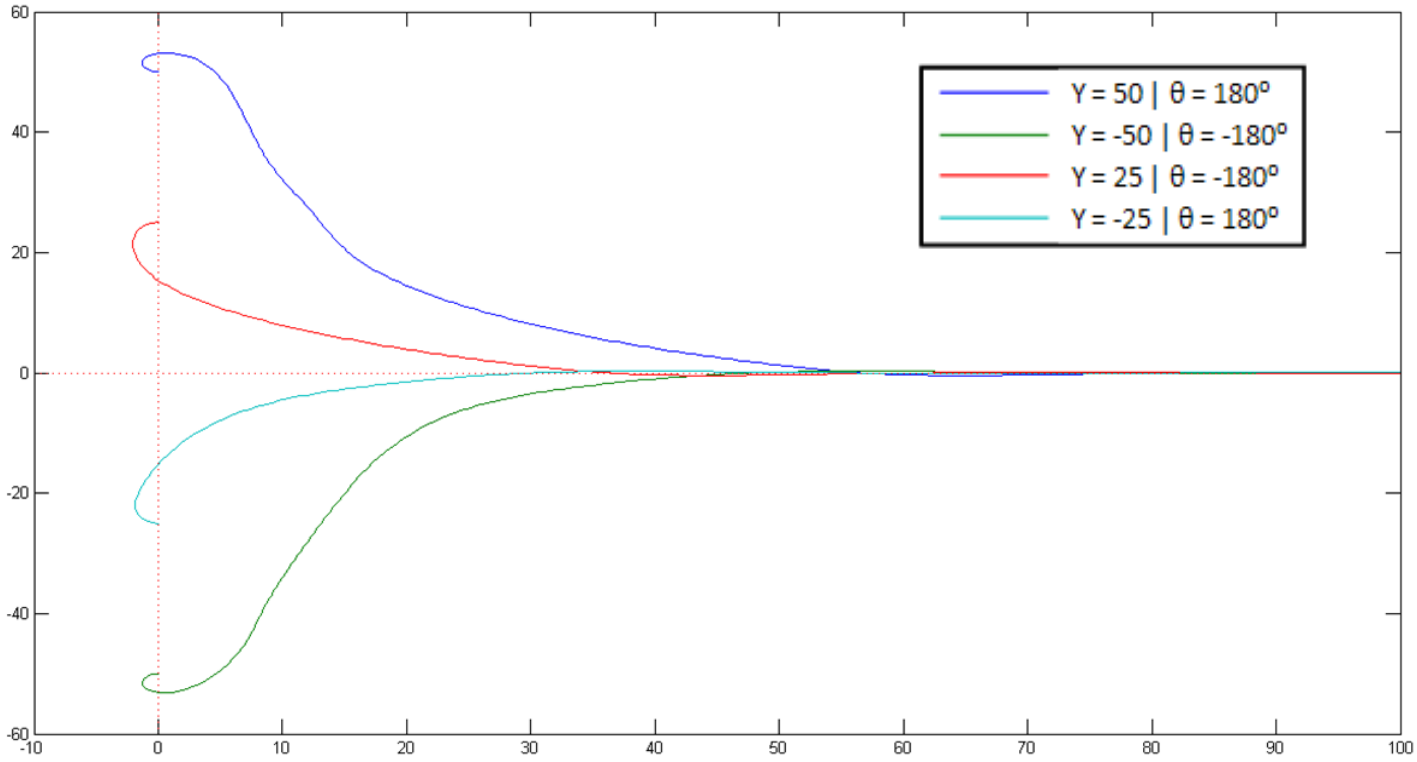
Label	NB	NS	Z	PS	PB
Centre	-0.8353	-0.2549	0.2235	0.7098	0.9294
Width	0.5725	0.2980	0.3373	0.4118	0.0392

Fuzzy Membership Functions for Input 2: Theta (θ)

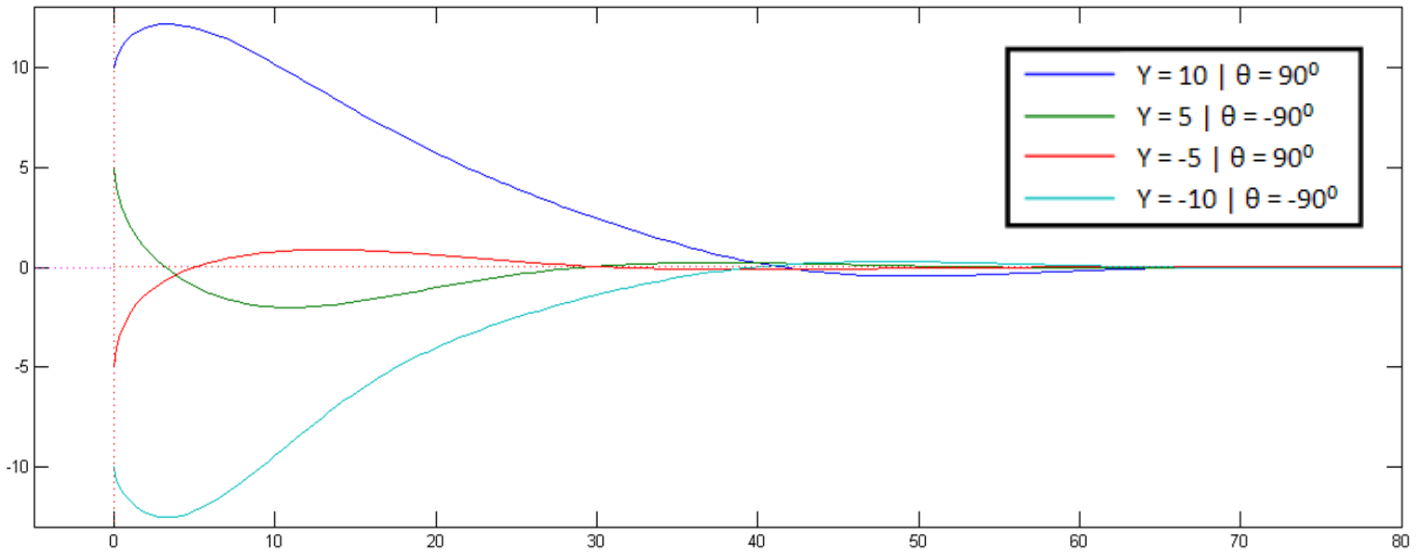
e(y)↓ θ→	NB	NS	Z	PS	PB
NB	0.984	0.216	0.671	-0.467	-0.882
NS	1.00	1.00	0.710	-0.953	-0.820
Z	0.545	0.254	-0.961	-0.969	0.255
PS	0.475	0.059	-0.255	0.184	-0.239
PB	0.977	-0.788	-0.977	-1.00	-0.953

The Weights representing the Steering Angle (α) to be controlled

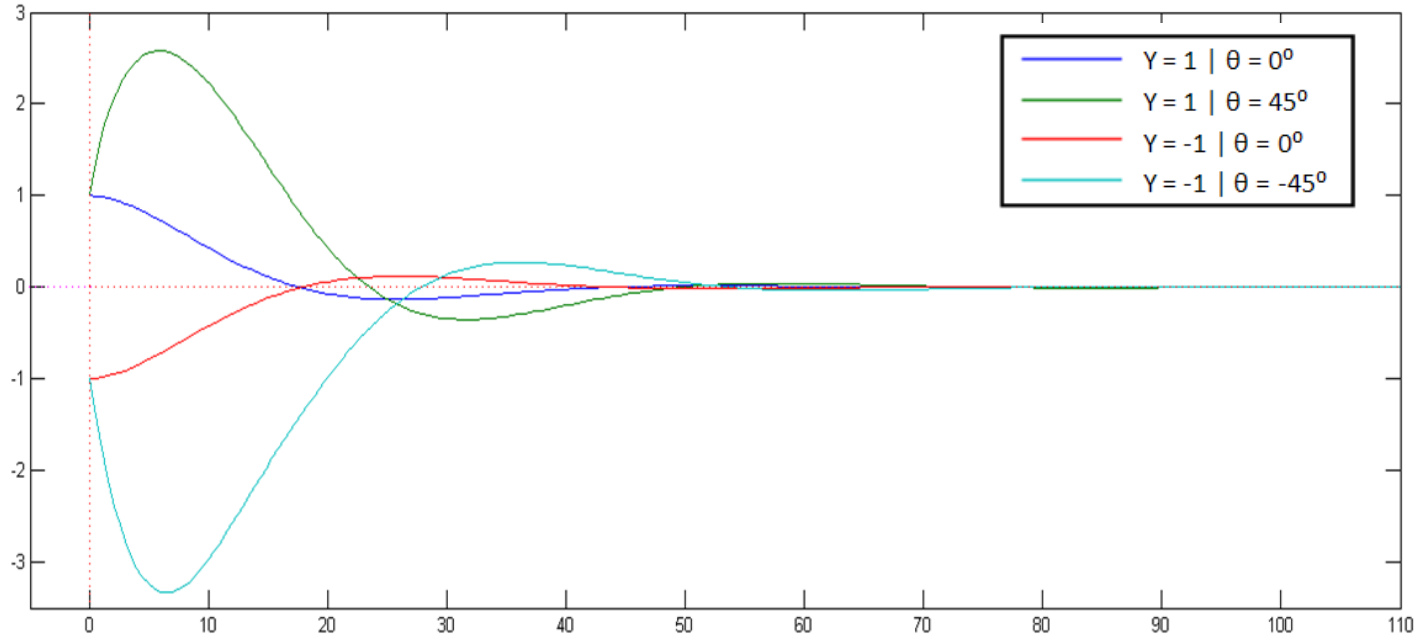
Using these results, we ran the optimized network and plotted the path of the car for a number of initial positions. The initial Y-position and the initial car angle θ are indicated in the legend associated with each graph. These are presented below.



Graph 1



Graph 2



Graph 3

As we can see, the car started from various Y and θ values sets its trajectory to x-axis well within the 300 seconds allotted to each trial. This is the **trajectory stabilization interpretation**. We can also interpret it as the car starting from an initial position and **reaching the car park** at (X, 0), given enough leeway for the car to reach X within 300 seconds. The initial condition of x position as zero is taken so, without

loss of generality, for a better representation of results, and so has final Y position and θ been taken as zero, again, **without loss of generality**.

The process of Genetic Algorithm produces subsequently better results in each generation, and we believe that **increasing the number of generations** in our program would lead to an even better solution in terms of subsequent error reduction in the position of the car. Without the availability of parallel processing techniques and multi-core CPUs, we were time-constrained to run it for only 500 generations.

The algorithm can also be improved using better parameters for **crossover and mutation rates, selection schemes, elitism and generation gap**, which can only be improved upon using experimentation. Using **more number of trials** for the error computation would also make the controller **symmetrical for negative and positive starting positions** as well as fully function for a **larger variety of initial and final conditions**.

The use of **gray-coded chromosomes instead of binary coded** ones could have resulted in faster convergence as well as avoided unintended changes during mutation. Gray-coding, however, made our program complexity even larger, and it was a compromise we made in order to use more trials and population size.

In conclusion, we were successful in making an Evolutionary Computation optimized Neuro-Fuzzy controller for the selected problem with satisfactory results, acknowledging that **there is scope for further improvement using better hardware, experimentation, and review of literature in the field.**