APPENDIX C

MATLAB SIMULATION FILES

C.1 Simulation of the inverse kinematics for 2-link manipulator

```
Function Sim2L_ik();
%% DEMO PROGRAM FOR INVERSE KINEMATICS APPROXIMATION BASED RBFN.
%% Training radial basic function network for inverse kinematics of
%% See also : create_sample1, create_sample2,create_sample3
%% pre_train,invese_kine2, plot_IK1, plot_IK2, ect
%% The second version in 11/2007, Bach Dinh, HWU
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear;
l1 = 50;
l2 = 50;
Lenlink = l1+l2;
%spread = 8;
distanceOfCentres = 10;
```

```
%% Choosing centres of RBFN as uniform distribution
centreStructure = 4;% or 4
if (centreStructure==5)
   % Mode = 2 : with offset
   % Mode = 1 : without offset
   c = create_sample3(distanceOfCentres,Lenlink,2); % 5 point structures
else
   c = create_sample2(distanceOfCentres,Lenlink); % 4 point structure
end

[r_c,c_c]= size(c); %% size of centres set
%% Architecture of linear layer of RBF network.
min_max_value = ([0;1]*ones(1,c_c))';
net = network(1,1,0,1,0,1,1);
net.numInputs = 1; %% No. of input layer
net.numLayers = 1; %% No of total layers in whole network
net.biasConnect = [0] ;%% All of layers have no biases value
net.inputs{1}.range = min_max_value;
net.layers{1}.size = 2;
net.iw{1,1}= zeros(2,c_c);
w1 = c';
%b1 = 0.8326/spread;
%% FINISH BUILDING NETWORK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CREATE TRAINING EXAMPLES. THIS STEP IS VERY IMPORTANT IN BUILDING
%% KNOWLEDGE OF RBFN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Create training data based on inverse kinematics function
TrainingData = 2;
switch TrainingData
   case 1 %%Case 1: training data from forward kinematic

      %[rs,P_train,T_train] = create_sample(50,50,0,180,0,180);
      %p = [rs(3,:); rs(4,:)];
      %t = [rs(1,:); rs(2,:)];

   case 2
   %% Constraned training data distributed uniformly in the workspace
   %% The centers point are the same inputs of training examples

   if (centreStructure==5)
   %% Mode = 2 : with offset
   %% Mode = 1 : without offset
      [P_train,T_train] = create_sample3(distanceOfCentres,Lenlink,2);
   else
      [P_train,T_train] = create_sample2(distanceOfCentres,Lenlink);
   end

   case 3
   %% training data by randomly optional selection
   %% In this case, the centers of RBFN are fixed as uniformly distributed
   %% points and different from training points.
   MaxDis = 4; %6*distanceOfCentres/10;
   StoreMode = 1;
   [P_train,T_train] = create_sample5(Lenlink,c,MaxDis,StoreMode);
   %data = dlmread('TrainData10_Rand30.txt');
   %P_train = data(1:2,:);
   %T_train = data(3:4,:);

   case 4
   %% Training examples in working area selected manually
```

```
   P_train = [1 22 1 21 13 10;60 42 39 60 53 61];
   for i = 1:6
      x = p(1,i);
      y = p(2,i);
      [d1,d2]= inverse_kine2(x,y,l1,l2);
      T_train(1,i)=d1;
      T_train(2,i)=d2;
   end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TRAINING PHASE
%% IN THIS STEP THREE ALGORITHMS WOULD BE APPLIED TO TRAIN THE WEIGHTS OF
%% NET - LINEAR LAYER OF RBFNET, STRICT INTERPOLATION, INCREMENTAL TRAINING
%% AND BATCH TRAINING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TrainingPara.goal = 0.0001;
TrainingPara.epochs = 100000;
TrainingPara.lr = 0.1;

%% pre_train is the method to train RBF network in inverse kinematics
%% TrainMode = 1 : incremental method
%% TrainMode = 2 : batch method
%% TrainMode = 3 : strict interpolation method
TrainMode = 2;

% TRAINING PROCESS BY A RANGE OF SPREAD PARAMETERS
numofSpread = 0;
for Spread = 6:2:28
   numofSpread = numofSpread + 1;
   b1 = 0.8326/Spread;
   if (TrainMode==2)
   switch Spread
      case 28
         TrainingPara.lr = 0.002;
      case 26
         TrainingPara.lr = 0.003;
      case 24
         TrainingPara.lr = 0.004;
      case 22
         TrainingPara.lr = 0.006;
      case 20
         TrainingPara.lr = 0.008;
      case {16,18}
         TrainingPara.lr = 0.01;
      case 14
         TrainingPara.lr = 0.03;
      otherwise
         TrainingPara.lr = 0.05;
   end
   end


tic %% Start the timer

%% Training the linear layer of RBF network with patterns
net = pre_train(net,w1,b1,P_train,T_train,TrainingPara,TrainMode);

t = toc %% stop the timer

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TEST OF THE NETWORK PERFORMANCEAFTER TRAINING.
```

```
%% A TRAJECTORY WILL BE KEPT IN THE EDGE OF WORKSPACEWORKING AREA,
%% THE ERRORS BETWEEN DESIREDAND ACTUAL POSITION CALCULATED.(TEST DATA
2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

test2 = [2 7 12 17 22 27 32 37 42 47 52 57 62 67 72 77 82 87;
      95 94.74 94.24 93.47 92.42 91 89.45 87.5 85.21 82.56 79.5 76 72
      67.35 62 55.6 48 38];

%% This test data is close to the edge of the workspace(x^2+y^2 = 95^2)
cycle2 = 0;
perf2 = 0;
ddX2 =0;
ddY2 =0;

for i= 1:18

      cycle2 = cycle2 +1;
      x_des(cycle2)= test2(1,i);
      y_des(cycle2)= test2(2,i);
      [th1_d,th2_d] = inverse_kine2(test2(1,i),test2(2,i),l1,l2);
      hd1 = radbas(dist(w1,[test2(1,i);test2(2,i)]).*b1);
      y_sim = sim(net,hd1);
      [x_a,y_a]= forward_kine(l1,l2,y_sim(1),y_sim(2));
      th1_a(cycle2)= y_sim(1);
      th2_a(cycle2)= y_sim(2);
      deth1(cycle2) = (th1_d - y_sim(1))*180/pi;%% Error of th1
      deth2(cycle2) = (th2_d - y_sim(2))*180/pi;%% Error of th2
      x_act(cycle2) = x_a;
      y_act(cycle2) = y_a;
      dX(cycle2) = test2(1,i) - x_a; %% Error of x
      dY(cycle2) = test2(2,i) - y_a; %% Error of y
      perf2 = perf2 + (deth1(cycle2)^2 + deth2(cycle2)^2);
      ddX2 = ddX2 + abs(test2(1,i) - x_a);
      ddY2 = ddY2 + abs(test2(2,i) - y_a);
end

RMS2 = sqrt(perf2/(2*cycle2));  %%Root Mean Square error
ddX2 = ddX2/cycle2;           %%Mean Absolute error of x
ddY2 = ddY2/cycle2;           %%Mean Absolute error of y

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TEST DATA 1 WITH A 5x5mm grids AREA(FUNCTION :plot_IKF1(net,5,c,spread,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dm = 5;
cycle = 0;
perf = 0;
ddX = 0;
ddY = 0;
i = 0;
for dy = 22:dm:52
   y = dy;
   i = i+1;
   yy(i) = y;
   j = 0;
   for dx = 22:dm:52
      cycle = cycle+1;
      j = j+1;
      x = dx;
      xx(j) = x;
      hiddenoutput = radbas(dist(w1,[x;y]).*b1);
      output = sim(net,hiddenoutput);
      [x_a,y_a]= forward_kine(50,50,output(1),output(2));
```

```
      dX(i,j) = x - x_a;%% Error of x
      dY(i,j) = y - y_a;%% Error of y
      ddX = ddX + abs(x - x_a);
      ddY = ddY + abs(y - y_a);
      t1(i,j) = output(1)*180/pi;
      t2(i,j)= output(2)*180/pi;
      [d1,d2]= inverse_kine2(x,y,50,50);
      t11(i,j) = d1*180/pi;
      t22(i,j)= d2*180/pi;
      ee1(i,j) = t11(i,j)- t1(i,j);
      ee2(i,j) = t22(i,j)- t2(i,j);
      perf = perf + (ee1(i,j)^2 + ee2(i,j)^2);
   end
end
MSE = sqrt(perf/(cycle*2)); %%Root Mean Square error
ddX = ddX/cycle;          %%Mean Absolute error of x
ddY = ddY/cycle;          %%Mean Absolute error of y

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% STORE RESULTS IN FILES
%% 3 first data are belonged to test with Test Data 1 (inside workspace)
%% 3 following data are belonged to test with Test Data 2 (edge of
%% workspace)
%% the last one is training time.

Test_Result(numofSpread,:)= [MSE ddX ddY MSE2 ddX2 ddY2 t];
end
dlmwrite('Test_LMS10_C_21.txt',Test_Result);
```

C.2 Simulation of 2-D online retraining to improve the RBFN

Function **training_rbf2(Lenlink, distanceOfCentres,spread)**

```
%% See also : center_rbf2,create_sample3, pre_train,invese_kine2, plot_IK1,
%% plot_IK2, etc.
%% Bach Dinh, HWU, 11/2007

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FIRST STEP : BUILDING AND TRAINING RBF NETWORK BY STRICT INTERPOLATION
% In the problem of inaccurate data of measurement system, we assume two
% set of inaccurate patterns which we could collect.
% Mode1 : inaccurate data of angular sensor affected in the joint angles.
% Mode2 : inaccurate data of camera system affected in position
% measurements.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Create centres of hidden layer as training data based on inverse kinematics function
clc;
clear;
l1 = 50;
l2 = 50;
link = l1+l2;
spread = 10;
distanceOfCentres = 10;

[c,tc] = center_rbf2(Lenlink,distanceOfCentres);% 4 points structure
%[c,tc] = create_sample3(distanceOfCentres,Lenlink,2); % 5 points structure
[r_c,c_c]= size(c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Adding noise in training data to make a inaccurate function of network
% In mode 1: noise is added in target ouput.
noise =  10*pi*ones(2,c_c)/180;
```

```
t = tc + noise ;
% In mode 2: noise is added in centre positions.
%noise = (2*rand(2,c_c)-1)*resolution_man*5/100;
%noise =  -ones(2,c_c)*resolution_man*5/100;
%p = c + noise ;
%c = p;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Building the RBF network with the fixed centre points and linear layer
% Architecture of linear layer of RBF network.

min_max_value = ([0;1]*ones(1,c_c))';
net = network(1,1,0,1,0,1,1);
net.numInputs = 1; % No. of input layer
net.numLayers = 1; % No of total layers in whole network
net.biasConnect = [0] ;% All of layers have biases value
net.inputs{1}.range = min_max_value;
net.layers{1}.size = 2;
w1 = c';
b1 = 0.8326/spread;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot the approximated function by network as inaccurate one.
%disp('Performance of inaccurate network after pre-trained');
%plot_IKF1(net,2,distanceOfCentres,spread,1);
%disp('Press any key to continue');pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SECOND STEP : ONLINE RETRAIN RBF NETWORK TO MODIFY THE APPOXIMATION
%% FUNCTION.
%% Collect a set of accurate training data to improve the generalized
%% ability of network based Incremental LMS for the linear layer.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

trainingPat2 = [20 56 53 20 40 24 50 51 25 38; 21 19 56 54 40 25 24 49 50 37];
%trainingPat2 = [19.67 25.59 25.80 28.51  34.08 35.15 36.22 42.03 46.03 44.12 51.09 52.28;
 %              21.99 23.02 27.26 28.57 30.83 33.14 36.16 42.09 42.82 47.25 47.91 54.19];
%trainingPat2 = [19 ; 23];
goal = 0.0001;
epochs = 1; %% number of retraining loop
lr = 0.4;
K = length(trainingPat2); %% number of training points applying to retraining phase

for count = 1:epochs
  for i = 1:K
 % disp('Enter the POSITION of training pattern.');
    xd = trainingPat2(1,i);%input('X = ');
    yd = trainingPat2(2,i);%input('Y = ');
    %% Using inverse kinematics to create training pattern
    [d1,d2]= inverse_kine2(xd,yd,50,50);
    x _plot(i) = xd;
    y_plot(i) = yd;
    z_plot1(i) = -40; %% keep the lengh of z axis.
    z_plot2(i) = 80;

   hd1 = radbas(dist(w1,[xd;yd]).*b1); % input of linear network
   y = sim(net,hd1);   % get network output for this input
   e = [d1;d2] - y;      % compare with target output to get error

   if (abs(e)>= goal)
       w2_old = net.iw{1,1};
       dw = lr*e*hd1'; %% Widrow-Hoff algorithm
```

```
        w2 = w2_old + dw;
        net.iw{1,1} = w2;%% Update the weights to network
     end
    %% Plot the current training response after one online update
    % plot_IKF2(net1,[0;0],c,spread,1); % plot correct and incorrect function
    % plot_IKF2(net,[xd;yd],c,spread,2); % plot current approximate function
    %  pause;
    end % end all of training example
end % end of online training process
```

C.3 Sub-function training algorithms for 2-D simulation

Function **net = pre_train(net,w1,b1,p,t,TrainingPara,trainingMethod)**

```
%% Pre-train phase by data collected from previous.
%% Bach H. Dinh, 11/2006
%% Connect to DEMO_RBF program.

   q = length(p);
   sr = length(w1);
   goal = TrainingPara.goal ;
   epochs = TrainingPara.epochs ;
   lr = TrainingPara.lr;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Training method = 1 - incremental training the delta rule
%% Training method = 2 - batch training with LMS
%% Training method = 3 - training by strict interpolation method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

switch trainingMethod
%% TRAINING BY INCREMENTAL MODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   case 1
   for count = 1:epochs %% num of retraining loop
    for i = 1:q % Appy each example as online training( NOT RETRAIN)
     hd1 = radbas(dist(w1,p(:,i)).*b1); % input of linear network
     y = sim(net,hd1);   % get network output for this input
     e = t(:,i) - y;      % compare with target output to get error

  %% The training process just update the weights and bias of linear layer
     if (abs(e)>=goal)
        w2_old = net.iw{1,1};
        dw = lr*e*hd1'; %% Widrow-Hoff algorithm
        w2 = w2_old + dw;
        %% Update the weights to network
        net.iw{1,1} = w2;
     end
    end % of training example
   end % end of training phase
% dlmwrite('weight_ini11.txt',net.iw{1,1});%% update weight

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LMS TRAINING ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 2
     hd1 = radbas(dist(w1,p).*b1);
     % Training Performance
     net.performFcn = 'mse';

    % Learning (Adaption and Training)
     net.inputWeights{1,1}.learnFcn = 'learnwh';
```

```
    net.inputWeights{1,1}.learnParam.lr = lr;

   % Adaption
   net.adaptFcn = 'trains';

   % Training
   net.trainFcn = 'trainb';

   % Initializiaton
   net.initFcn = 'initlay';
   net.layers{1}.initFcn = 'initwb';
   net.inputWeights{1,1}.initFcn = 'initzero';
   net = init(net);
   net.trainParam.epochs = epochs;
   net.trainParam.goal = goal;

   net = train(net,hd1,t);

 %   dlmwrite('weight_ini2.txt',net.iw{1,1});%% update weight

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% STRICT INTERPOLATION METHOD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 3

%% Using interpolation matrix to calculate the weight of linear layer

   sct = length(p);
   hd1 = radbas(dist(w1,p).*b1);%*ones(1,sct)));

   w2 = t/hd1; % standard RBF network
   net.iw{1,1} = w2;
   dlmwrite('weight_ini3.txt',net.iw{1,1});%% update weight

end %% End training phase
```

C.4 Sub-function 2-D forward and inverse kinematics expressions

```
%% Forward and Inverse kinematics function of 2 link Robot
%% Ver2 , by Bach H. Dinh, 12/2005
```

Function **[p1,p2] = forward_kine(l1,l2,ang_1,ang_2)**

```
p1 = l1*cos(ang_1) + l2*cos(ang_1 + ang_2);
p2 = l1*sin(ang_1) + l2*sin(ang_1 + ang_2);
```

Function **[d1,d2] = inverse_kine2(x,y,l1,l2)**

```
c2 = (x^2 + y^2 - l1^2 - l2^2)/(2*l1*l2);
s2 = sqrt(1 - c2^2);
d2 = atan2(s2,c2);
k1 = l1 + l2*c2;
k2 = l2*s2;
d1 = atan2(y,x)- atan2(k2,k1);
```

C.5 Sub-function plot the 2-D IK functions during retraining phase

Function **[c,t] = plot_IKF2(net,data,Centres,spread, mode)**

```
%% This function plots responses of a RBF network as inverse kinematics approximated function in
workspace.
%% Version 3, Bach Dinh, 11/2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_s = data(1);
y_s = data(2);
dm = 5;
Lenlink = 100;
N = 0;

w1 = Centres';
b1 = 0.8326/spread;

%% Calculate and plot the output of network on area neiboured the training data
i = 0;
perf = 0;
ddX = 0;
ddY = 0;

for dy = 22:dm:52
    y = dy;
    i = i+1;
    yy(i)= y;
    j = 0;

   for dx = 22:dm:52
        j = j+1;
        N = N+1;
        x = dx;
        xx(j) = x;

   % Calculate the output of radial basis functions of hidden layer
        hiddenoutput = radbas(dist(w1,[x;y]).*b1);
        output = sim(net,hiddenoutput);

        % Calculate errors of the position control system
        [x_a,y_a]= forward_kine(50,50,output(1),output(2));

        dX(i,j) = x - x_a;
        dY(i,j) = y - y_a;
        ddX = ddX + abs(x - x_a);
        ddY = ddY + abs(y - y_a);
        %% Calculate errors of the network

        t1(i,j) = output(1)*180/pi;
        t2(i,j)= output(2)*180/pi;
        [d1,d2]= inverse_kine2(x,y,50,50);
        t11(i,j) = d1*180/pi;
        t22(i,j)= d2*180/pi;
        ee1(i,j) = t11(i,j)- t1(i,j);
        ee2(i,j) = t22(i,j)- t2(i,j);
        perf = perf + (ee1(i,j)^2 + ee2(i,j)^2);
    end
end

%%%disp('Mean squared error OF TEST 2 = ');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MSE = sqrt(perf/(2*N));
ddX = ddX/N;
ddY = ddY/N;
```

```
%% Plot out the function flat

if mode ==1 %% The correct function and initial function
  figure(1);
  subplot(2,1,1);%% Theta 1
  mesh(xx,yy,t1);
  hold on
  mesh(xx,yy,t11);

  subplot(2,1,2);%% Theta2
  mesh(xx,yy,t2);
  hold on
  mesh(xx,yy,t22);
end
%% Continue with current function

if mode == 2 %% The current approximate function
  disp('Root Mean squared error OF the network = ');
  disp(MSE);
  disp('Mean error X OF TEST 2 = ');
  disp(ddX);
  disp('Mean error Y OF TEST 2 = ');
  disp(ddY);
  figure(1);
  subplot(2,1,1);%% Theta1
  hold on
  mesh(xx,yy,t1);
 %plot3(x_s,y_s,-40,'*');
  hold off
  subplot(2,1,2);%% Theta2
  hold on
  mesh(xx,yy,t2);
%plot3(x_s,y_s,80,'*');
hold off
```

C.6 Simulation of the inverse kinematics for 3-link manipulator

Function **sim3L_ik()**

```
%% DEMO PROGRAM FOR INVERSE KINEMATICS APPROXIMATION BASED RBFN.
%% Training radial basic function network for inverse kinematics of 3 link manipulator.
%% See also : fkine_3,ikine_3L, pre_train3link, etc
%% The first version in 07/2007.
%% Bach H. Dinh, Heriot-Watt University
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear;
robot.l1 = 31;
robot.l2 = 45;
robot.l3 = 55;

%% Choosing centres of RBFN as uniform distribution
%spread = 8;
distanceOfCentres = 10;

c = center_3l(robot,distanceOfCentres); % 4 point structures
numofCentre = length(c); %% size of centres set
```

```
% Architecture of linear layer of RBF network.
min_max_value = ([0;1]*ones(1,numofCentre))';
net = network(1,1,0,1,0,1,1);
net.numInputs = 1; % No. of input layer
net.numLayers = 1; % No of total layers in whole network
net.biasConnect = [0] ;% All of layers have no biases value
net.inputs{1}.range = min_max_value; % num of inputs of linear layer
net.layers{1}.size = 3; % num of outputs of linear layer
net.iw{1,1}= zeros(3,numofCentre);
w1 = c';
%b1 = 0.8326/spread;

%% FINISH BUILDING NETWORK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% CREATE TRAINING EXAMPLES. THIS STEP IS VERY IMPORTANT IN BUILDING
%% KNOWLEDGE OF RBFN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Create training data based on inverse kinematics function

TrainingData = 3;
switch TrainingData
    case 1
    % Case 1: training data from forward kinematic
      [rs,P_train,T_train] = create_sample(50,50,0,180,0,180);
      p = [rs(3,:); rs(4,:)];
      t = [rs(1,:); rs(2,:)];

    case 2
    % % Constrained training data distributed uniformly through entire workspace
      [P_train,T_train]= center_3l(robot,distanceOfCentres); % 4 point structures
      data = [P_train;T_train];

    case 3
    %% training data by randomly optional selection. In this case, the centers of RBFN are fixed as
%%uniformly distributed
      MaxDis = 3*distanceOfCentres/10;%
       StoreMode = 1;
       Random patterns around centres positions
       data = randomsamp3_2(robot,c,MaxDis,StoreMode);%
   %   data = dlmread('3DDataRand_R30.txt');

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TRAINING PHASE, IN THIS STEP THREE ALGORITHMS WOULD BE APPLIED TO TRAIN
%% THE WEIGHTS OF NET - LINEAR LAYER OF RBFNET, STRICT INTERPOLATION,
%%INCREMENTAL TRAINING AND BATCH TRAINING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TrainingPara.goal = 0.0001;
TrainingPara.epochs = 500000;
TrainingPara.lr = 0.02;
TrainMode = 3;

% TRAINING PROCESS BY A RANGE OF SPREAD PARAMETERS
numofSpread = 0;

for Spread = 6:2:28
    numofSpread = numofSpread + 1;
    b1 = 0.8326/Spread;
    if (TrainMode==2)
      switch Spread
        case 28
          TrainingPara.lr = 0.0001;
```

```
        case 26
          TrainingPara.lr = 0.0003;
        case 24
          TrainingPara.lr = 0.0003;
        case 22
          TrainingPara.lr = 0.0005;
        case 20
          TrainingPara.lr = 0.0008;
        case {16,18}
          TrainingPara.lr = 0.001;
        case 14
          TrainingPara.lr = 0.003;
        case {10,12}
          TrainingPara.lr = 0.01;
        otherwise
          TrainingPara.lr = 0.05;
    end
end


tic %% Start the timer
%% pre_train is the method to train RBF network in inverse kinematics
%% TrainMode = 1 : incremental method
%% TrainMode = 2 : batch method
%% TrainMode = 3 : strict interpolation method
%% Training the linear layer of RBF network with patterns

net = pre_train3link(net,w1,b1,data,TrainingPara,TrainMode);

t = toc %% stop the timer

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TEST WITH DATA SET 2 (AT THE EDGE OF THE WORKSPACE)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Td2  = [ 92.0675   24.6694   49.0000
         90.2663   24.1868   57.0000
         87.7505   23.5127   65.0000
         84.4564   22.6300   73.0000
         80.2881   21.5131   81.0000
         75.1003   20.1231   89.0000
         68.6623   18.3980   97.0000
         60.5767   16.2315  105.0000
         78.0777   54.6706   49.0000
         76.5502   53.6010   57.0000
         74.4167   52.1071   65.0000
         71.6231   50.1511   73.0000
         68.0882   47.6759   81.0000
         63.6887   44.5953   89.0000
         58.2290   40.7724   97.0000
         51.3720   35.9711  105.0000
         54.6706   78.0777   49.0000
         53.6010   76.5502   57.0000
         52.1071   74.4167   65.0000
         50.1511   71.6231   73.0000
         47.6759   68.0882   81.0000
         44.5953   63.6887   89.0000
         40.7724   58.2290   97.0000
         35.9711   51.3720  105.0000 ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cycle2 = 0;
perf2 = 0;
```

```
ddX2 =0;
ddY2 =0;
ddZ2 = 0;

for j = 1:24

      cycle2 = cycle2 +1;
      x_des2(cycle2)= Td2(j,1);
      y_des2(cycle2)= Td2(j,2);
      z_des2(cycle2)= Td2(j,3);
      th_d = ikine_3L(robot,[Td2(j,1);Td2(j,2);Td2(j,3)]);
      hd1 = radbas(dist(w1,[Td2(j,1);Td2(j,2);Td2(j,3)]).*b1);
      th_sim = sim(net,hd1);
      P_act = fkine_3(robot,th_sim);
      deth1(cycle2) = (th_d(1) - th_sim(1))*180/pi;
      deth2(cycle2) = (th_d(2) - th_sim(2))*180/pi;
      deth3(cycle2) = (th_d(3) - th_sim(3))*180/pi;

      %% Use the actual position and the actual angles to train the linear layer
      %% to satisfy the relation actual position and angles.
      x_act2(cycle2) = P_act(1);
      y_act2(cycle2) = P_act(2);
      z_act2(cycle2) = P_act(3);
      dX2(cycle2) = Td2(j,1) - P_act(1);
      dY2(cycle2) = Td2(j,2) - P_act(2);
      dZ2(cycle2) = Td2(j,3) - P_act(3);

      perf2 = perf2 + (deth1(cycle2)^2 + deth2(cycle2)^2 + deth3(cycle2)^2);
      ddX2 = ddX2 + abs(Td2(j,1) - P_act(1));
      ddY2 = ddY2 + abs(Td2(j,2) - P_act(2));
      ddZ2 = ddZ2 + abs(Td2(j,3) - P_act(3));

end
RMS2 = sqrt(perf2/(3*cycle2));
ddX2 = ddX2/cycle2;
ddY2 = ddY2/cycle2;
ddZ2 = ddZ2/cycle2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TEST WITH TEST DATA SET 1 (INSIDE WORKSPACE)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

k = 0;
dm = 5;
perf = 0;
ddX = 0;
ddY = 0;
ddZ = 0;
x1 = 36;
y1 = 34;
z1 = 48;

for jz = 0:1:4
   kz = z1 + dm*jz;
   for jx = 0:1:4
      kx = x1 + dm*jx;
      for jy = 0:1:4
         ky = y1 + dm*jy;
         k = k +1;
         Td1(k,:)= [kx ky kz];
         % joint angles based mathematical inverse kinematics function
         th_tar = ikine_3L(robot,[kx;ky;kz]);
         HiddenOutput = radbas(dist(w1,[kx;ky;kz]).*b1);
```

```
            th_sim = sim(net,HiddenOutput);
            P_act = fkine_3(robot,th_sim);
            deth1(k) = (th_tar(1) - th_sim(1))*180/pi;
            deth2(k) = (th_tar(2) - th_sim(2))*180/pi;
            deth3(k) = (th_tar(3) - th_sim(3))*180/pi;
            %% Use the actual position and the actual angles to train the linear layer
            %% to satisfy the relation actual position and angles.
            x_act(k) = P_act(1);
            y_act(k) = P_act(2);
            z_act(k) = P_act(3);
            dX(k) = kx - P_act(1);
            dY(k) = ky - P_act(2);
            dZ(k) = kz - P_act(3);
            perf = perf + (deth1(k)^2 + deth2(k)^2 + deth3(k)^2) ;
            ddX = ddX + abs(kx - P_act(1));
            ddY = ddY + abs(ky - P_act(2));
            ddZ = ddZ + abs(kz - P_act(3));
        end
    end
end

RMS = sqrt(perf/(3*k));
ddX = ddX/k;
ddY = ddY/k;
ddZ = ddZ/k;

Test_Result(numofSpread,:)= [RMS ddX ddY ddZ RMS2 ddX2 ddY2 ddZ2 t];
end
dlmwrite('Test_str10_c_test.txt',Test_Result);
```

C.7 Simulation of 3-D online retraining to modify the RBFN

Function **net = recorrect3L_ik(robot,distanceOfCentres,spread)**

```
%% The first version in 08/2007. Connecting to train3L_ik.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FIRST STEP : BUILDING AND TRAINING RBF NETWORK BY STRICT INTERPOLATION
% In the problem of inaccurate data of measurement system, we assume two
% set of inaccurate patterns which we could collect.
% Mode1 : inaccurate data of angular sensor affected in the joint angles.
% Mode2 : inaccurate data of camera system affected in position
% measurements.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear;
robot.l1 = 20;
robot.l2 = 50;
robot.l3 = 50;
spread = 12;
distanceOfCentres = 10;

%% Create centres of hidden layer as training data based on inverse kinematics function
%% Using inverse kinematics function to calculate the target output according to inputs as centres
[c,t] = center_3l(robot,distanceOfCentres); % 4 point structures
numofCentre = length(c); %% size of centres set

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Adding noise in training data to make a inaccurate function of network
% In mode 1: noise is added in target ouput.
% In mode 2: noise is added in centre positions.
NoiseMode = 1;
```

```
switch NoiseMode
   case 1
   noise =  10*pi*ones(3,numofCentre)/180;
   t = t + noise ;
   disp('Noise in joint angles.')
   case 2
   noise = (2*rand(3,numofCentre)-1)*2;
 %   noise =  -ones(2,c_c)*resolution_man*5/100;
   c = c + noise ;
   disp('Noise in position measurement.')
   otherwise
   disp('Noiseless in network.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Building the RBF network with the fixed centre points and linear layer
% Architecture of linear layer of RBF network.
min_max_value = ([0;1]*ones(1,numofCentre))';
net = network(1,1,0,1,0,1,1);
net.numInputs = 1; % No. of input layer
net.numLayers = 1; % No of total layers in whole network
net.biasConnect = [0] ;% All of layers have no biases value
net.inputs{1}.range = min_max_value; % num of inputs of linear layer
net.layers{1}.size = 3; % num of outputs of linear layer
net.iw{1,1}= zeros(3,numofCentre);
w1 = c';
b1 = 0.8326/spread;

%% Using interpolation matrix to calculate the weight of linear layer
IniateMode = 1;
switch IniateMode
   case 1
      %matrix_b1 = ones(numofCentre,1)*b1;
      HiddenOutput = radbas(dist(w1,c).*b1); %(matrix_b1*ones(1,numofCentre)));
      w2 = t/HiddenOutput;
      net.iw{1,1} = w2;
      disp('Initiate from beginning with strict interpolation by incorrect data.');
   case 2
      net.iw{1,1} = dlmread('weight_re32.txt');%% Load weights from file
      disp('Load weights from file.');
end
%plot3_IKF(net,robot,c,spread);pause;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot the approximated function by network as inaccurate one.
%disp('Performance of inaccurate network after pre-trained');
%plot_IKF1(net,2,distanceOfCentres,spread,1);
%disp('Press any key to continue');pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SECOND STEP : ONLINE RETRAIN RBF NETWORK TO MODIFY THE APPOXIMATION
%% FUNCTION.
%% Collect a set of accurate training data to improve the generalized
%% ability of network based Incremental LMS for the linear layer.
%% RetrainMode = 1 : trainned by manually selected patterns (each step)
%% RetrainMode = 2 : trainned by selected patterns (mass training)
%% RetrainMode = 3 : trainned by random patterns (mass training)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RetrainMode = 2
switch RetrainMode
   case 1
      disp('PERFORMANCE OF NETWORK BEFORE RECORRECT PHASE');
      plot3_IKF(net,robot,c,spread);
```

```
    pause;
    goal = 0.001;
    lr = 0.2;
    enterPat = input('Press 1 for entering data Or anynumber to quit.');
    while ( enterPat == 1)%% number of training patterns applying to retraining phase
       disp('The POSITION of training pattern.');
       xd = input('X = ');
       yd = input('Y = ');
       zd = input('Z = ');

       if ((xd^2 + yd^2 + (zd-z_offset)^2)<=(len^2))
       %% Using inverse kinematics to create training pattern
       th_tar = ikine_3L(robot,[xd;yd;zd]);
       HiddenOutput = radbas(dist(w1,[xd;yd;zd]).*b1);
       th_sim = sim(net,HiddenOutput);% get network output for this input
       e = th_tar - th_sim;      % compare with target output to get error

       if (abs(e)>= goal)% update one step only
          w2_old = net.iw{1,1};
          dw = lr*e*HiddenOutput'; %% Widrow-Hoff algorithm
          w2 = w2_old + dw;
          net.iw{1,1} = w2;%% Update the weights to network
       end
       %% Plot the current training response after one online update
       plot3_IKF(net,robot,c,spread);
       else
         disp('Your position entered as an outside of workspace.');
       end
       enterPat = input('Press 1 for entering data Or anynumber to quit.');
    end
    saveFile = input('Press 1 for saving weights Or anynumber to refuse.');
    if (saveFile==1)
       disp('Store weights in file weight_re31.txt');
       dlmwrite('weight_re31.txt',net.iw{1,1});
    else
       disp('Not store data.');
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% End of case 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 2
 %     disp('PERFORMANCE OF NETWORK BEFORE RECORRECT PHASE');
 %     plot3_IKF(net,robot,c,spread);
 %     pause;

    Pat5 = [34 34 34 34 34 58 58 58 58 58 46 46 46 46 46;
          35 61 61 35 48 35 61 61 35 48 35 61 61 35 48;
          48 48 74 74 62 48 48 74 74 62 48 48 74 74 62];


    prePat = Pat5;
    numofPat = length(prePat);
    goal = 0.0001;
    epochs = 1;
    lr = 0.1;
    k = 0;
    for count = 1:epochs
       trainingPat = randintrlv(prePat,count);
       %goal = s_goal*1/(count^10);
       for i = 1:numofPat %% number of training patterns applying to retraining phase
  %     disp('The POSITION of training pattern.');
       xd = trainingPat(1,i);%input('X = ');
```

```
        yd = trainingPat(2,i);%input('Y = ');
        zd = trainingPat(3,i);%input('Y = ');
        %% Using inverse kinematics to create training pattern
        k = i + (count-1)*numofPat;
        th_tar = ikine_3L(robot,[xd;yd;zd]);
        x_plot(k) = xd;
        y_plot(k) = yd;
        z_plot(k) = zd;
        HiddenOutput = radbas(dist(w1,[xd;yd;zd]).*b1);
        th_sim = sim(net,HiddenOutput);% get network output for this input
        e = th_tar - th_sim;      % compare with target output to get error
        %% The training process just update the weights and bias of linear layer

        if (abs(e)>= goal)
           w2_old = net.iw{1,1};
           dw = lr*e*HiddenOutput'; %% Widrow-Hoff algorithm
           w2 = w2_old + dw;
           net.iw{1,1} = w2;%% Update the weights to network
        end
        %% Plot the current training response after one online update
        plot3_IKF(net,robot,c,spread);
        disp('Trained times =')
        k
        pause;
        end % of training example
    %      plot3_IKF(net,robot,c,spread);
    %      pause;
      end
      plot3_IKF(net,robot,c,spread);
%       saveFile = input('Press 1 for saving weights Or anynumber to refuse.');
%       if (saveFile==1)
%          disp('Store weights in file weight_re32.txt');
%          dlmwrite('weight_re32.txt',net.iw{1,1});
%       else
%          disp('Not store data.');
%       end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% End of case 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 3
      % data = dlmread('randompattern_2.txt');
      % trainingPat = data(1:3,:);
      numofPat = length(trainingPat);
      goal = 0.00001;
      lr = 0.5;
      for i = 1:numofPat %% number of training patterns applying to retraining phase
        disp('The POSITION of training pattern.');
        xd = trainingPat(1,i)%input('X = ');
        yd = trainingPat(2,i)%input('Y = ');
        zd = trainingPat(3,i)%input('Y = ');
        %% Using inverse kinematics to create training pattern
        th_tar = ikine_3L(robot,[xd;yd;zd]);
        x_plot(i) = xd;
        y_plot(i) = yd;
        z_plot(i) = zd; %% keep the lengh of z axis.
        HiddenOutput = radbas(dist(w1,[xd;yd;zd]).*b1);
        th_sim = sim(net,HiddenOutput);% get network output for this input
        e = th_tar - th_sim;      % compare with target output to get error
        %% The training process just update the weights and bias of linear layer
        w2_old = net.iw{1,1};
        if (abs(e)>= goal)
           dw = lr*e*HiddenOutput'; %% Widrow-Hoff algorithm
```

```
            w2 = w2_old + dw;
            net.iw{1,1} = w2;%% Update the weights to network
         end
         %% Plot the current training response after one online update
         plot3_IKF(net,robot,c,spread);
         pause;
      end % of training example
      saveFile = input('Press 1 for saving weights Or anynumber to refuse.');
      if (saveFile==1)
         disp('Store weights in file weight_re33.txt');
         dlmwrite('weight_re33.txt',net.iw{1,1});
      else
         disp('Not store data.');
      end
end
disp('Complete.')
```

C.8 Sub-function training algorithms for 3-D simulation

Function **net = pre_train3link(net,w1,b1,traindata,TrainingPara,trainingMethod)**

```
%% Pre-train3link is training phase by data collected from previous.
%% Connect to sim3L_ik program.
%% See also :
%% Bach H. Dinh, 07/2007

q = length(traindata); % num of patterns
sr = length(w1);
goal = TrainingPara.goal ;
epochs = TrainingPara.epochs ;
lr = TrainingPara.lr;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Training method = 1 - incremental training by Delata rule
%% Training method = 2 - batch training with LMS
%% Training method = 3 - training by strict interpolation method

switch trainingMethod
   % case 1 : Training phase with incremental method
  case 1
  %   train_secl = traindata';
     p = traindata(1:3,:);%changed
     t = traindata(4:6,:);%changed
     for count = 1:epochs % number of retraining loop
%        da_train = randintrlv(train_secl,count);
%        p = da_train(:,1:3)';
   %     t = da_train(:,4:6)';

        for i = 1:q % Appy each example as online training( NOT RETRAIN)
         hd1 = radbas(dist(w1,p(:,i)).*b1); % input of linear network
         y = sim(net,hd1);   % get network output for this input
         e = t(:,i) - y;      % compare with target output to get error
         %% The training process just update the weights and bias of linear layer
         for i=1:3
            if (abs(e(i))<=goal)
               e(i) = 0;
            end
         end

         w2_old = net.iw{1,1};
         dw = lr*e*hd1'; %% Widrow-Hoff algorithm
```

```
            w2 = w2_old + dw;
            %% Update the weights to network
            net.iw{1,1} = w2;
           end % of training example
       end % end of training phase
        dlmwrite('weight_ini31.txt',net.iw{1,1});%% write weights to file
    %case 2 : training phase with batch training method LMS
    case 2
        p = traindata(1:3,:);
        t = traindata(4:6,:);
        HiddenOutput = radbas(dist(w1,p).*b1);

        % Performance
        net.performFcn = 'mse';

        % Learning (Adaption and Training)
        net.inputWeights{1,1}.learnFcn = 'learnwh';
        net.inputWeights{1,1}.learnParam.lr = learnRate;

        % Adaption
        net.adaptFcn = 'trains';

        % Training
        net.trainFcn = 'trainb';

        % Initializiaton
        net.initFcn = 'initlay';
        net.layers{1}.initFcn = 'initwb';
        net.inputWeights{1,1}.initFcn = 'initzero';
        net = init(net);

        net.trainParam.epochs = epochs;
        net.trainParam.goal = goal;
        net = train(net,HiddenOutput,t);

       dlmwrite('weight_ini32.txt',net.iw{1,1});%% write weights to file

    case 3
%% Using interpolation matrix to calculate the weight of linear layer
    p = traindata(1:3,:);
    t = traindata(4:6,:);
    hd1 = radbas(dist(w1,p).*b1);
    w2 = t/hd1;
    net.iw{1,1} = w2;
    dlmwrite('weight_ini33.txt',net.iw{1,1});%% write weights to file

end
```

C.9 Sub-function 3-D forward and inverse kinematics expressions

```
%% Forward and Inverse kinematics for 3-link manipulator
%% 07/2007 by Bach H.Dinh, HWU
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function **P = fkine_3(robot,q)**

```
q1 = q(1);
q2 = q(2)+ pi/2;
q3 = q(3);

l1 = robot.l1; %  20;
l2 = robot.l2; %  50;
l3 = robot.l3; %  50;

A1 = [cos(q1) 0 sin(q1) 0;
   sin(q1) 0 -cos(q1) 0;
   0 1 0 l1;
   0 0 0 1];
A2 = [cos(q2) -sin(q2) 0 l2*cos(q2);
   sin(q2) cos(q2) 0 l2*sin(q2);
   0 0 1 0;
   0 0 0 1];
A3 = [cos(q3) -sin(q3) 0 l3*cos(q3);
   sin(q3) cos(q3) 0 l3*sin(q3);
   0 0 1 0;
   0 0 0 1];

T = A1*A2*A3;
P = [T(1,4);T(2,4);T(3,4)];
```

Function **th = ikine_3L(robot,P)**

```
%% Lengths of links
l1 = robot.l1; %  20;
l2 = robot.l2; %  50;
l3 = robot.l3; %  50;

%% Position of manipulator
Px = P(1);
Py = P(2);
Pz = P(3);

%% Inverse kinematics of Algebraic approach
th1 = atan(Py/Px);
K1 = cos(th1)*Px + sin(th1)*Py ;
K2 = Pz - l1;
%th2 = atan2(K2,K1) + atan2((sqrt(4*l2^2*(K1^2+K2^2) - (K1^2+K2^2+l2^2-
l3^2)^2)),(K1^2+K2^2+l2^2-l3^2));
%th3 = atan2(K2 - l2*sin(th2), K1 - l2*cos(th2))-th2;
%dth2 = th2 - pi/2;
%th = [th1;dth2;th3];

%% Inverse kinematics of Geometric approach
K1 = sqrt(Px^2 + Py^2) ;
th2 = - atan2(K1,K2) + atan2((sqrt(4*l2^2*(K1^2+K2^2) - (K1^2+K2^2+l2^2-
l3^2)^2)),(K1^2+K2^2+l2^2-l3^2));
th3 = atan2(-K1 - l2*sin(th2),K2 - l2*cos(th2))- th2;
th = [th1;th2;th3];
```

C.10 Sub-function plot 3-D IK functions during the retraining phase

Function **plot3_IKF(net,robot,Centres,spread)**

```
%% This function plots responses of a RBF network as 3D inverse kinematics
%% approximated function in workspace.
%% Version 1, Bach Dinh, 08/2007
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Building the hidden layer based on distance of centres and spread
numofCentre = length(Centres);
w1 = Centres';
b1 = 0.8326/spread;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate and plot the output of network on area neiboured the training
%% data
k = 0;
dm = 5;
perf = 0;
ddX = 0;
ddY = 0;
ddZ = 0;
x1 = 36;
y1 = 38;
z1 = 51;
for jz = 0:1:4
   kz = z1 + dm*jz;
   for jx = 0:1:4
      kx = x1 + dm*jx;
      for jy = 0:1:4
         ky = y1 + dm*jy;
         k = k +1;
         x_des(k)= kx;
         y_des(k)= ky;
         z_des(k)= kz;
         % joint angles based mathematical inverse kinematics function
         th_tar = ikine_3L(robot,[kx;ky;kz]);
         HiddenOutput = radbas(dist(w1,[kx;ky;kz]).*b1);
         th_sim = sim(net,HiddenOutput);
         P_act = fkine_3(robot,th_sim);
         deth1(k) = (th_tar(1) - th_sim(1))*180/pi;
         deth2(k) = (th_tar(2) - th_sim(2))*180/pi;
         deth3(k) = (th_tar(3) - th_sim(3))*180/pi;
         %% Use the actual position and the actual angles to train the linear layer
         %% to satisfy the relation actual position and angles.
         x_act(k) = P_act(1);
         y_act(k) = P_act(2);
         z_act(k) = P_act(3);
         dX(k) = kx - P_act(1);
         dY(k) = ky - P_act(2);
         dZ(k) = kz - P_act(3);
         perf = perf + (deth1(k)^2 + deth2(k)^2 + deth3(k)^2) ;
         ddX = ddX + abs(kx - P_act(1));
         ddY = ddY + abs(ky - P_act(2));
         ddZ = ddZ + abs(kz - P_act(3));
      end
   end
end
mse = sqrt(perf/(3*k));
ddX = ddX/k;
ddY = ddY/k;
ddZ = ddZ/k;
```

```
disp('NETWORK PERFPRMANCE ');
disp('Root Mean square error of the network performance = ');
disp(mse);
disp('ERRORS IN X ');
disp(ddX);
disp('ERRORS IN Y ');
disp(ddY);
disp('ERRORS IN Z ');
disp(ddZ);

figure(1)
subplot(3,1,1);
plot(1:k,x_des,'-k',1:k,x_act,'b');

subplot(3,1,2);
plot(1:k,y_des,'-k',1:k,y_act,'r');

subplot(3,1,3);
plot(1:k,z_des,'-k',1:k,z_act,'g');

figure(2)
subplot(3,1,1);
bar(1:k,dX,'b');

subplot(3,1,2);
bar(1:k,dY,'r');

subplot(3,1,3);
bar(1:k,dZ,'g');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(3)
subplot(3,1,1);
bar(1:k,deth1,'b');
subplot(3,1,2);
bar(1:k,deth2,'g');
subplot(3,1,3);
bar(1:k,deth3,'r');

figure(4);
plot3(x_des,y_des,z_des,'r*');
```