1. TRAINING DATA GENERATION

First we create a function called ArmsGeneration that will help us with the parts later as well. This function gets as inputs the arm lengths, the origin, the thetas and the number of samples and will provide us with the elbows points and the end locations points.

```
function [P1 P2] = ArmsGeneration(armLen ,theta, origin, samples)
  % Empty sets
  P1 = [];
  P2 =[];
  %Loop and get elbow and end points
  for i = 1:samples
    %Get the P1 and P2 values
    [p1,p2] = RevoluteForwardKinematics2D(armLen, theta(i,:), origin);
    %Store them
    P1 = [P1 ;p1];
    P2 = [P2 ;p2];
  end
end
```

1.1 Display workspace of revolute arm

This function called EndPointLocations generates 1000 samples of theta and by calling the ArmsGeneration function we created earlier we will get the P1 elbow Locations and End Points location. Also we have the plot function that demonstrate our data to a graph (*Figure 1*).

```
function [P2, theta] = EndPointLocations()
  %Number of samples
  samples = 1000;
  %Theta Samples from 0 to pi
  theta = pi.*rand(samples,2);
  %Arm Lengths  0.4
  armLen = [0.4 0.4];
  %Origin points
  origin = [0 0];
  %ArmsGeneration points locations
  [P1, P2] = ArmsGeneration(armLen,theta,origin,samples);

  figure
```

```
    hold on
    h = title('10578755: Arm EndPoint Locations')
    set(h, 'FontSize', 18);
    h = xlabel('x[m]')
    set(h, 'FontSize', 15);
    h = ylabel('y[m]')
    set(h, 'FontSize', 15);
    h = plot(origin(1),origin(2),'k*');
    set(h, 'MarkerSize', 12);
    set(h, 'LineWidth', 2);
    h = plot(P2(:,1),P2(:,2),'r*');
    set(h, 'LineWidth', 2);
    set(h, 'MarkerSize',2);
    legend('Origin','End point');
end
```

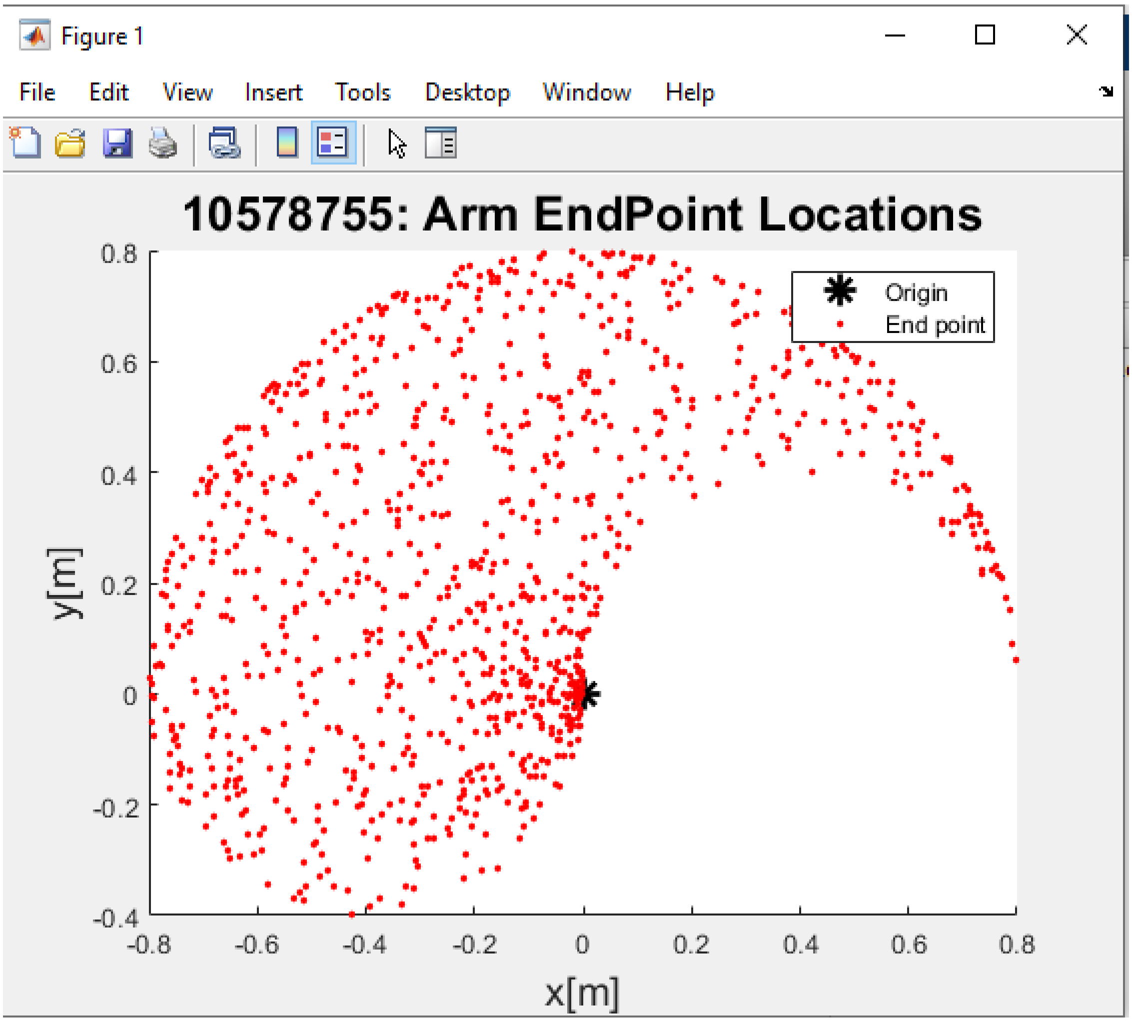

Figure 1: End points location of kinematic arm

1.2 Configurations of a revolute arm

The next function called ArmConfigurations will generate 10 samples of theta and then by calling our helper function ArmsGeneration we will get the P1 elbow Locations and P2 End Points locations. We also have the plot function that demonstrate our data to a graph (*Figure 2*). (Next Page)

```
function [origin,P1,P2] = ArmConfigurations()
    % Number of arms
  samples = 10;

  theta = pi.*rand(samples,2);
  %Arm Lenghts  0.4
  armLen = [0.4 0.4];
   %Origin points
  origin = [0 0];

  %ArmsGeneration and store the elbow locations and end points to P1 and P2
  [P1, P2] = ArmsGeneration(armLen,theta,origin,samples);

    % Filling a column with X and Y Start points Origin position
    Xstart = repmat(origin(1),samples,1);
    Ystart = repmat(origin(2),samples,1);

    %Creating Matrices to connect the lines between the origin
    Xcon = [P1(:,1) P2(:,1)];
    Ycon = [P1(:,2) P2(:,2)];
    OXcon = [Xstart P1(:,1)];
    OYcon = [Ystart P1(:,2)]
    figure
    hold on
    h = title('10578755: Arm Configurations')
    set(h, 'FontSize', 18);
    h = xlabel('x[m]')
    set(h, 'FontSize', 15);
    h = ylabel('y[m]')
    set(h, 'FontSize', 15);
    h = plot(origin(1),origin(2),'k*');
    set(h, 'MarkerSize', 14);
    set(h, 'LineWidth', 2);
    h = plot(P2(:,1),P2(:,2),'ro');
    set(h, 'MarkerSize', 5);
    set(h, 'LineWidth', 3);
    h = plot(P1(:,1),P1(:,2),'go');
    set(h, 'MarkerSize', 5);
    set(h, 'LineWidth', 3);
    h = plot(Xcon',Ycon','b-');
    set(h, 'LineWidth', 2);
    h = plot(OXcon',OYcon','b-');
    set(h, 'LineWidth', 2);
    legend('Origin','End point');
end
```

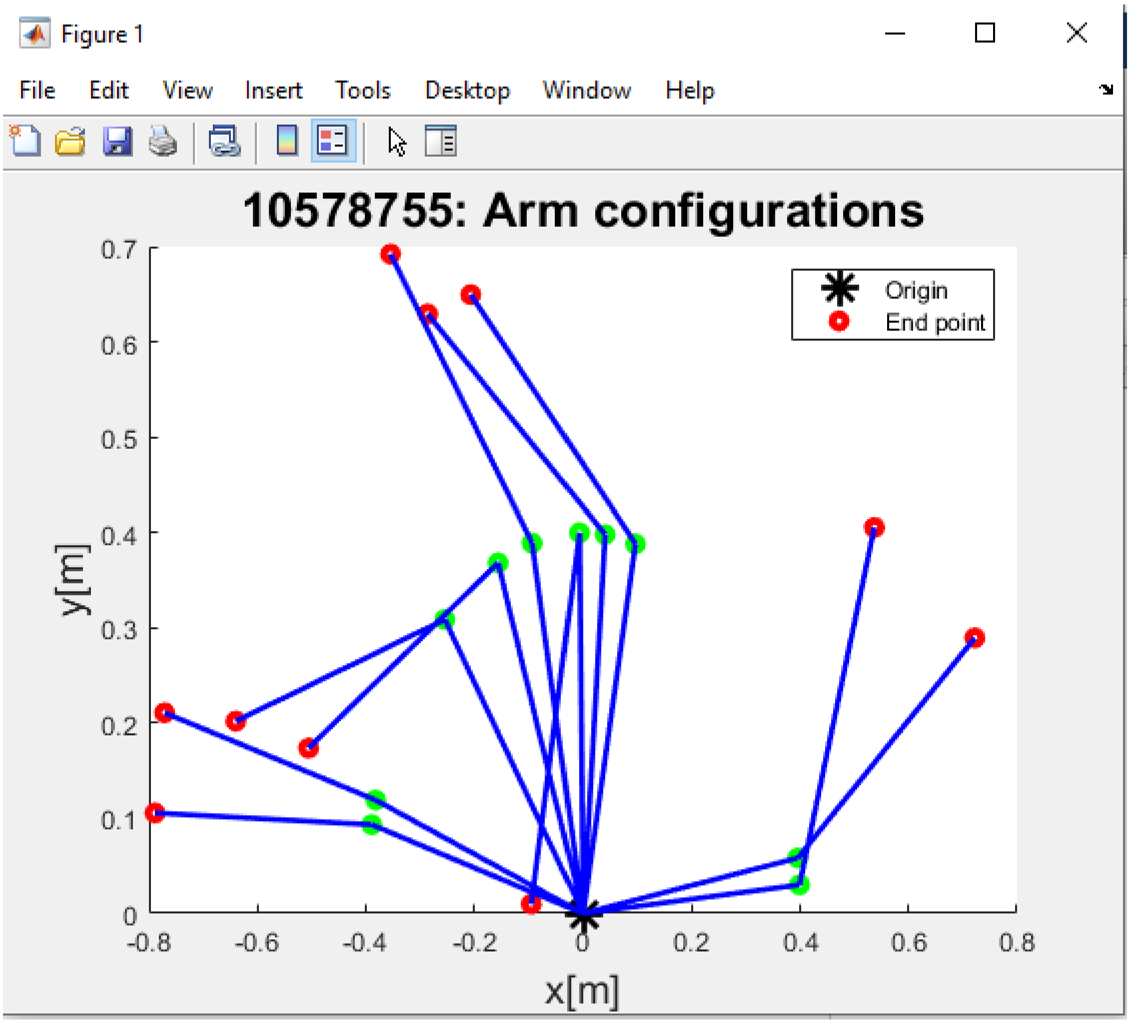

Figure 2: Arm configurations for 10 data points

2. IMPLEMENT A TWO-LAYER NETWORK

First we need a function that will provide us with the weight matrix of our multi-layer network.We also need to add bias to our matrix and to calculate the cost of the error.

```
function W = Weights(x,y)
      % W is an x*y matrix with random Values
      W = randn(x,y)
end

function a = Bias(x)
      % Add bias one to a which is the input layer
      add = ones(1,size(x,2));
      a = [x; add];
end


function e = Cost(target,prediction)
       %Calculate the cost
       e = 0.5 .* (prediction - target).^2;
 end
```

Then we create the function that returns the Sigmoid results for the activation layer and sigmoid for the hidden layers which are adding our bias row we created before.

```
 function a = Sigmoid(net)
       %Activation function for output Layer
       a = 1.0 ./ ( 1.0 + exp(-net));
end

function a = Hidden(net)
   % Activation for hidden Layers and adding a bias row
   a = Sigmoid(net);
   a = Bias(a);
end
```

2.1 Implement the network feedforward pass

Then we go on and create the feedforward pass function providing us with the prediction output and the values of the hidden layer units.

```
function [net2 , a, a2] = Prediction(data,W1,W2)
    % Input times Weights + Bias
    net1 = W1 * data;

    %Sigmoid and adding a bias
    a = Hidden(net1);

    %Matrix Multiplication
    net2 = W2 * a;

    %Sigmoid and adding bias to a2
    a2 = Hidden(net2);
end
```

2.2 Implement 2-layer network training

```
function [W1 , W2 , error] = TrainingData(data,target,hiddenUnits)
    % Train our weights
    dataWithBias = Bias(data);

    % Get some random Weights for input to hidden layer
    W1 = Weights(hiddenUnits,size(dataWithBias,1));

    % Get some random Weights + 1 adding Bias weight
    W2 =  Weights(size(target,1),hiddenUnits+1);

    error = [];

    %Our loop iterations
    repetitions = 2000;

    % Learning rate
    learningrate = 0.001;

    % Train Neural Network
    for i = 1:repetitions

        % Feedforward pass
        [o , a] = Prediction(dataWithBias,W1,W2);

        %Calculating and storing the overrall Cost function
        e = sum(Cost(target,o));
        error = [error; e];
```

```
        % Removing the bias rows
        W2hat = W2(:,1:end-1);
        %Remove bias from Hidden Layer
        aHat = a(1:end-1,:);

        %Delta rule for W1, W2
        d3 = (o - target);

        %Delta rule for W1
        d2 = W2hat' * d3 .* aHat .* (1 - aHat);

        %Calculating the errors for the weights
        dW1 = d2 * dataWithBias';
        dW2 = d3 * a';

        %Updating Weights
        W1 = W1 - (learningrate * dW1);
        W2 = W2 - (learningrate * dW2);
    end
end
```

2.3 Train Network inverse Kinematics

Running the following function will help us to train our weights and plot the error and our trained data compared with the ones we started. (Figure 3)

```
function theta = RunNetwork(W1,W2,data)
  %adds bias to matrix
  dataB = Bias(data);
  %Using the get Prediction function that actually run Neural Network
  [theta,a] = Prediction(dataB,W1,W2);
End

function [W1,W2] = OurNetwork()

  [input,target] = EndPointLocations;
  % Hidden Units
  hiddenUnits = 2;
  %Training Data
  [W1, W2, error] = TrainingData(input.',target.',hiddenUnits);

  origin = [0 0];
  armLen = [0.4 0.4];

  %call the function that will run our actual NN
  theta = RunNetwork(W1,W2,input');

  %Plot the trained data
  [P1,P2] = ArmsGeneration(armLen,theta',origin,length(theta));
```

```
    figure
    hold on
    h = title('10578755: Regenerated Via Inv and Fwd Model Endpoints')
    set(h, 'FontSize', 18);
    h = xlabel('x[m]')
    set(h, 'FontSize', 15);
    h = ylabel('y[m]')
    set(h, 'FontSize', 15);
    h = plot(origin(1),origin(2),'k*');
    set(h, 'MarkerSize', 12);
    set(h, 'LineWidth', 2);
    h = plot(P2(:,1),P2(:,2),'r*');
    set(h, 'LineWidth', 2);
    set(h, 'MarkerSize',2);
    legend('Origin','End point');

    %Plot the error
    PlotError(error);
end
```

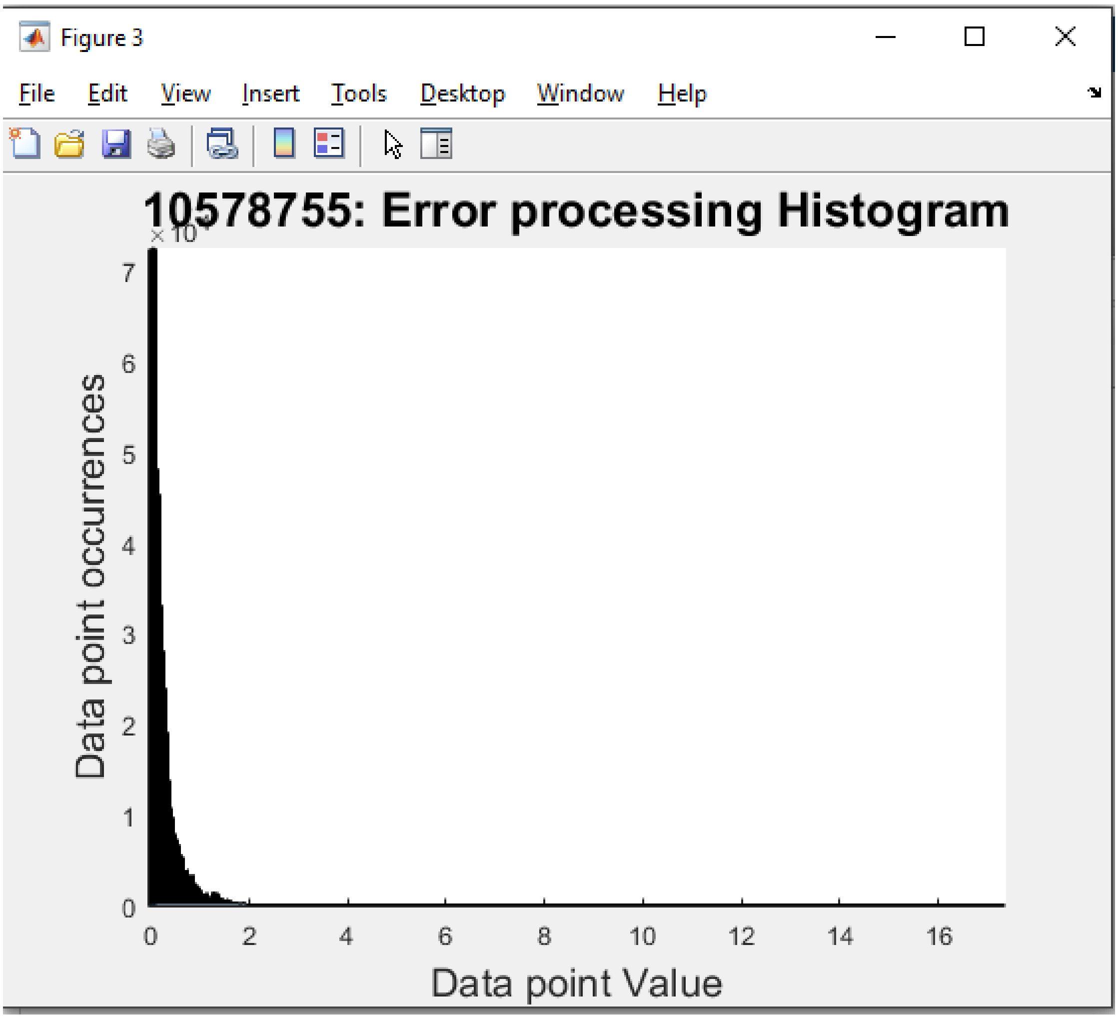

Figure 3: Our error values plotted throughout the training process

2.4 Test and interpret inverse model

After running our feedforward pass and training our data we come and plot our trained data and compare them with the random data we got at the start(Figure 4)

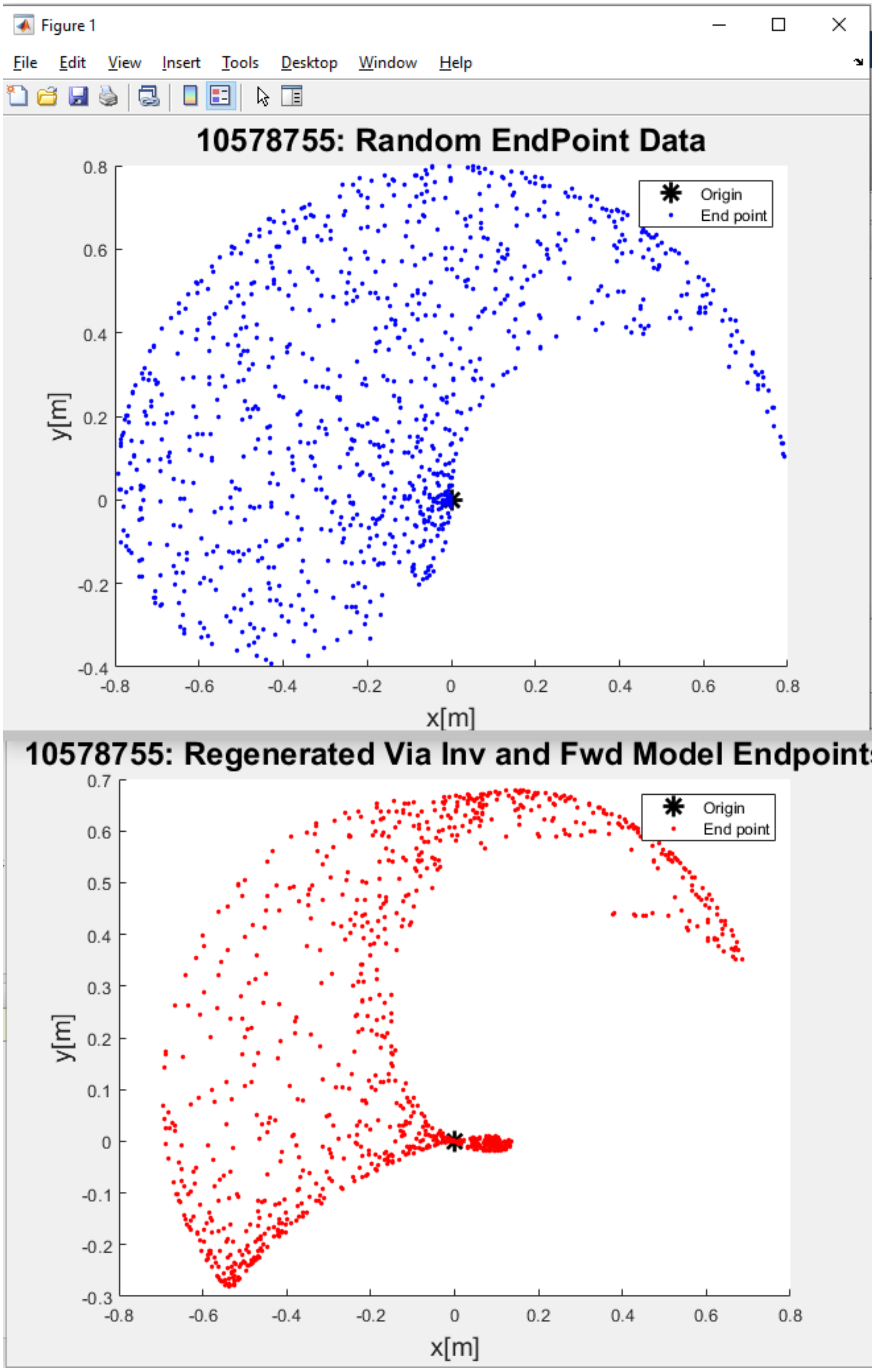

Figure 4: Comparison between the random end point data plotted at the start with the blue color in the figure and the trained end point data plotted after our Multi-layered network training with 2 hidden units.

Trained data are not uniform. They dont remain the same in all cases and times and are being distributed differently according to the hidden units, training times and learning rates you feed them or even the training datasets you use.

3. PATH THROUGH A MAZE

3.1 Random Start State

Generating 1000 starting states and plotting the histogram as shown below (*Figure 5*).

```
function startingState = RandomStatingState(f)
    startingState = [];
    for i = 1 : 1000
      b=0;
      while(b == 0)
        %Get random state
        state = randi([1 f.stateCnt],1);
        %Check if it's a normal step and not the Goal State
        if(f.stateOpen(f.stateX(state),f.stateY(state)) ~= 0 && f.stateEndID ~= state)
          b = 1;
          startingState = [state];
        end
      end
    end
end
```

```
% PRINT VALUE OF STARTING STATES AND HISTOGRAM
figure(2)
hold on
histogram(startingState,100);
h = title('10578755: Histogram Test of starting states')
set(h, 'FontSize', 18);
h = xlabel('Data point Value')
set(h, 'FontSize', 15);
h = ylabel('Data point occurrences')
set(h, 'FontSize', 15);
```

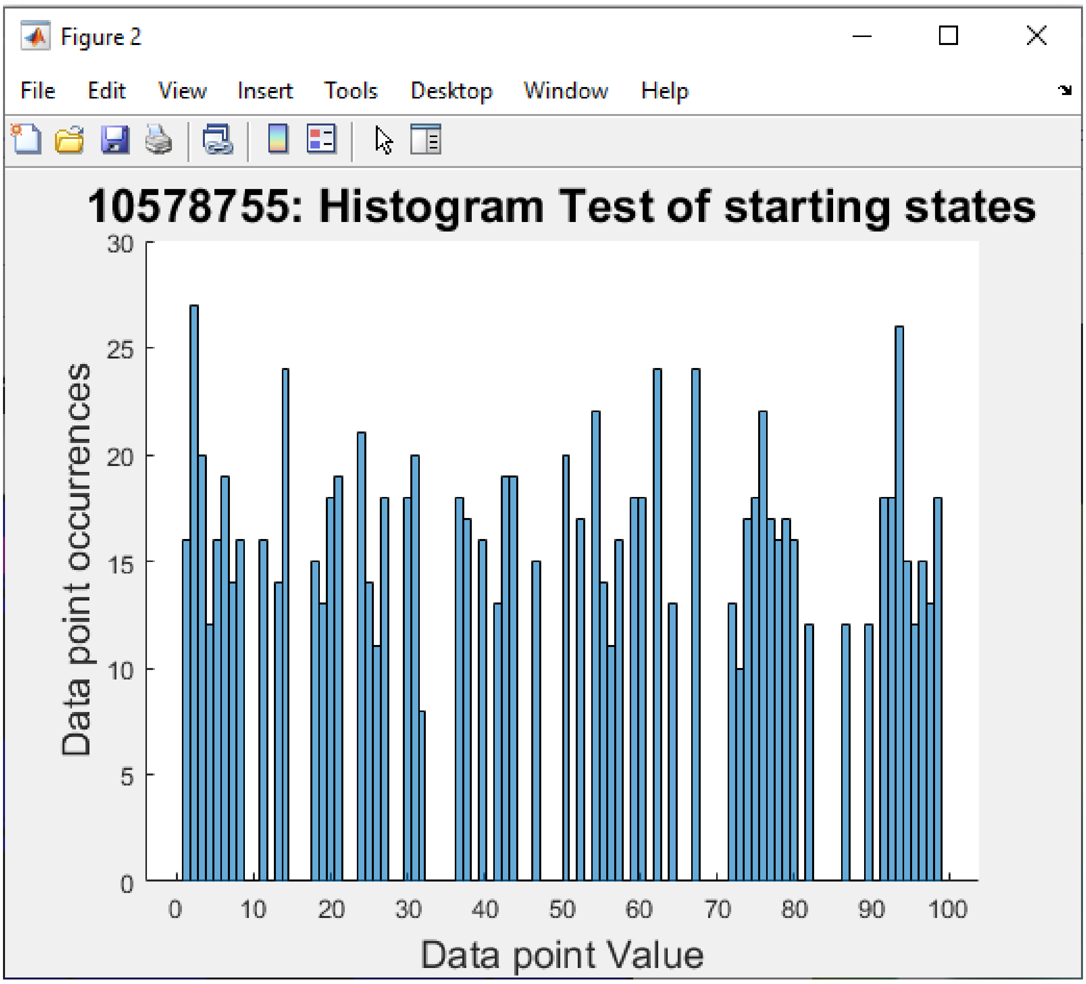

Figure 5: Histogram of 1000 randomly generated starting states plotted with 100 bins.

3.2 Build a reward function

MakeAMove Function. The function that takes a state and an action, then returns the x , y values of the next state.

```
    function [x, y] = MakeAMove(f,stateID,action)
        % A function that moves us into the maze
        x = f.stateX(stateID);
        y = f.stateY(stateID);
        % 1 = up 2 = right 3 = down 4=left
        if(action == 1)
          y = y + 1;
        elseif(action == 2)
          x = x + 1;
        elseif(action == 3)
          y = y - 1;
        elseif (action == 4)
          x = x -1 ;
       end
end
```

ValidAction Function. The function that takes x, y and checks if is a valid action.

```
function valid = ValidAction(f,x,y)
  % Check if action is valid
  valid = 0;
  if(x <= 0 || y<=0 || f.xStateCnt < x || f.yStateCnt < y)
    valid = 2;
  elseif(f.stateOpen(x,y) == 0)
    valid = 1;
  end
end
```

Reward function getting a state and an action and returning a 10 reward if the action leads to the endState, -10 if it is a Blocked location, -50 if it is out of bounds and 0 if it makes a normal step.

```
function reward = RewardFunction(f, stateID, action)
      reward = 0;
      %Getting the x and y values of action from the current stateID
      [x,y] = f.MakeAMove(stateID,action);
      valid = f.ValidAction(x,y);

      if(valid == 2)
        % Invalid Actions
        reward = -50;
      elseif(valid == 1)
        % Actions that move the agent to blockLocation
        reward = -10;
      else
        %Set the move to nextState and check Rewards
        nextStateID = f.stateNumber(x,y);
      end

      if(f.IsEndState(nextStateID) == 1)
         %Goal reward
         reward = 10;
       else
         %Step reward
         reward = 0;
       end
end
```

3.3 Generate the transition matrix

```
function f = BuildTransitionMatrix(f)
        % allocate
        f.tm = zeros(f.xStateCnt * f.yStateCnt, f.actionCnt);
        %Get
        for i = 1 : f.xStateCnt
           for j = 1 : f.yStateCnt
              for a = 1 : f.actionCnt
                %Calculate state
                state =  i + (j -1) * f.xStateCnt;
                %Make all possible actions
                [x, y] = f.MakeAMove(state,a);
                %Check if move is valid
                 if( f.ValidAction(x,y) < 2)
                    %Update TransitionMatrix
                    f.tm(state,a) = f.stateNumber(x,y);
                 end
              end
           end
   end
```

3.4 Initialize Q-values

```
function f = InitQTable(f, minVal, maxVal)
        mazeSize = f.xStateCnt * f.yStateCnt;
        % allocate
        f.QValues = zeros(mazeSize, f.actionCnt);
        %Initialize Qvalues randomly from Min to Max Val
        f.QValues = minVal + (maxVal - minVal)*rand(mazeSize,f.actionCnt);
end
```

3.5 Implement Q-learning algorithm

Trials Function that gets as inputs trails, episodes, alpha, gamma and explores the parameters and returns the Updated Q-values and the steps of each episode:

```
function [steps QValues] = Trials(f,trials,episodes,alpha,gamma,explore)
        %Getting Current Qvalues
        QValues = f.QValues;
        steps = [];
        for t = 1:trials
           %Run Episodes and get steps and new Q-values
           [step QValues] = f.Episodes(episodes,alpha,gamma,explore,QValues);
           %Save all steps from all episodes
           steps = [steps step];
        end
end
```

Episodes Function:

```
function [steps, QValues] = Episodes(f,episodes,alpha,gamma,explore,QValues)
        steps = [];
        %Get the random starting states
        randomStartinStates = f.RandomStatingState();

        for i = 1:episodes
          state = randomStartinStates(i); %Get Random start Location
          step = 0; % Set to 0
          while(f.IsEndState(state) == 0)
            %Get action by E-greedy function
            action = f.Greedy(state,explore,QValues);
            %Apply it to current state and get the next State
            nextState = f.tm(state,action);
            %Find Next State Possible actions
            [r, n_actions] = find(f.tm(nextState,:)>0);
            %Find the next MAX q value of the nextState
            max_q = max(QValues(nextState,n_actions));
            %Update Qvalues
            QValues = f.UpdateQValue(state,action,max_q,alpha,gamma,QValues);
            %Set the current state equals to the next State
            state = nextState;

            step = step + 1; %Count Steps
          end
          steps = [steps step]; %Save them
        end
    end
```

Greedy Function that exploits or explores by returning an action and updates the Q-values functions:

```
function a = Greedy(f,state,explore,QValues)
        %Find Possible actions
        [r, actions] = find(f.tm(state,:)>0);
        %Explore and Exploit
        if(rand(1) < explore)
          %Get Random action Explore
          a = actions(1,randi([1 length(actions)],1));
        else
          %Get max Qvalue Action Exploit
          [r,c] = max(QValues(state,actions));
          a = actions(c);
        end
     end
function QValues = UpdateQValue(f,state,action,max_q,alpha,gamma,QValues)
        q = QValues(state,action);
        %Q-learing Algorithm
        new_q = q + alpha * (f.RewardFunction(state,action) + gamma * max_q - q);
        %Set the new Q-value to our table
        QValues(state,action) = new_q;
end
```

3.6 Run Q-learning algorithm

```
%Set trials
trials = 100;
%Set episodes
episodes = 1000;
% Set Discount rate
gamma = 0.9;
% Set Learning rate
alpha=0.2;
%Explore possibility
explore = 0.1;
%Run Q-Learning Algorithm
Steps(:,trials) = maze.Trials(trials,episodes,alpha,gamma,explore);
 %Make the mean and SD data
 stepsMean = nanmean(Steps’);
 stepsSD = sqrt(nanvar(Steps’));
 %Send them to the function to plot the error bar
 maze.PlotSteps(stepsMean,stepsSD)
```

3.7 Exploitation of Q-values

I developed the following functions the one solving the maze and getting in a vector of the stateIDs that solves the maze and the getPathXY which terutns a matrix with x and y coordinates of the maze and the other one plotting the maze. (Figure 6)

```
function stateSolution = SolveMaze(f,QValues)
        %Set state to begin
        state = 1;
        %Initialiaze Solution
        stateSolution = state;
        %Set Explore rate 0
        explorerate = 0;

       for i = 1: f.xStateCnt * f.yStateCnt
         %Get action using Greedy function
         action = Greedy(f,state,explorerate,QValues);
         %Get the next State
         state = f.tm(state,action);
         %Save all states
         stateSolution = [stateSolution state];
         %Break if we arrived to goal state
         if(f.IsEndState(state) == 1)
            break;
         end
      end
 end
 function solutionXY = getPathXY(f,solution)
       solutionXY = [];
       for i = 1:length(solution)
          x = f.stateX(solution(i));
          y = f.stateY(solution(i));
          solutionXY = [solutionXY; (f.cursorCentre(x,y,1) + 0.01) (f.cursorCentre(x,y,2) +
          0.015)];
       end

end
function DrawMazeSolution(f,solution)
       solutionXY = f.getPathXY(solution);
       hold on
       h = plot(solutionXY(:,1),solutionXY(:,2),'mx');
       set(h, 'MarkerSize', 18);
       set(h, 'LineWidth', 5);
       h = plot(solutionXY(:,1),solutionXY(:,2),'m-', 'LineWidth', 4);
end
```

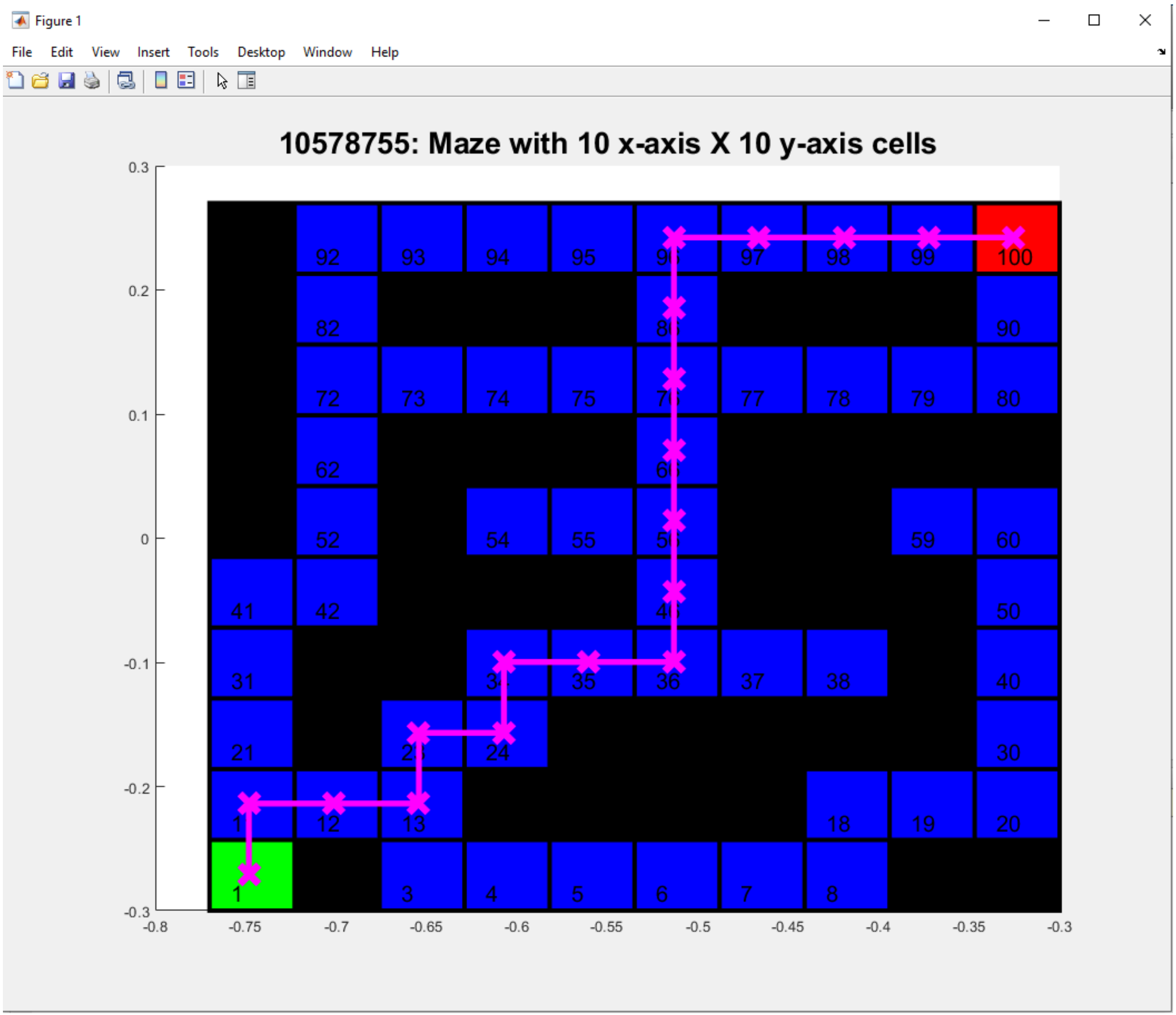

Figure 6: Pathing through the maze found by Q-learning algorithm starting from the green cell at state 1 and reached the red cell at state 100 which is the end.

4. MOVE ARM ENDPOINT THROUGH MAZE

4.1 Generate kinematic control to revolute arm

This function gets as inputs the path, origin and armLen and by getting already the trained weights runs the results in our Neural Network and generates the arms, elbows and the End Location points.

```
function [P1,P2] = ArmEndPoints(path,origin,armLen)
  %Store how many samples
  samples = length(path');
  %Getting the trained W1,W2
  [w1,w2] = OurNetwork();
  %Run neural network and provide us with the prediction theta values
  theta = RunNetwork(w1,w2,path');
  %Finally generating the Arms with our ArmsGeneration function with those thetas
  [P1,P2] = ArmsGeneration(armLen,theta',origin,length(theta));
End
```

4.2 Animated revolute arm movement

This function will plot the animation on our maze. (Figure 7)

```
function DrawMazeArm(f,solution,P1,P2,origin)
        f.DrawMaze();
        f.DrawMazeSolution(solution);
        hold on
        h = title('10578755: Animation of Revolute Arm moving along path in Maze')
        set(h, 'FontSize', 18);
        h = xlabel('Horizontal position[m]')
        set(h, 'FontSize', 15);
        h = ylabel('Vertical position [m]')
        set(h, 'FontSize', 15);
        h = plot(origin(1),origin(2),'k*');
        set(h, 'MarkerSize', 14);
        set(h, 'LineWidth', 2);

        for i = 1: samples
         A   h = plot([origin(1),P1(i,1)],[origin(2),P1(i,2)],'b-');
           set(h, 'LineWidth', 2);
           pause(0.5);
           h = plot(P1(i,1),P1(i,2),'go');
           set(h, 'MarkerSize', 5);
           set(h, 'LineWidth', 3);
           pause(0.5);
           h = plot(P2(i,1),P2(i,2),'ro');
           set(h, 'MarkerSize', 5);
```

```
            set(h, 'LineWidth', 3);
            pause(0.5)
            h = plot([P1(i,1),P2(i,1)],[P1(i,2),P2(i,2)],'b-');
            set(h, 'LineWidth', 2);
            pause(1);
        end
    end
```

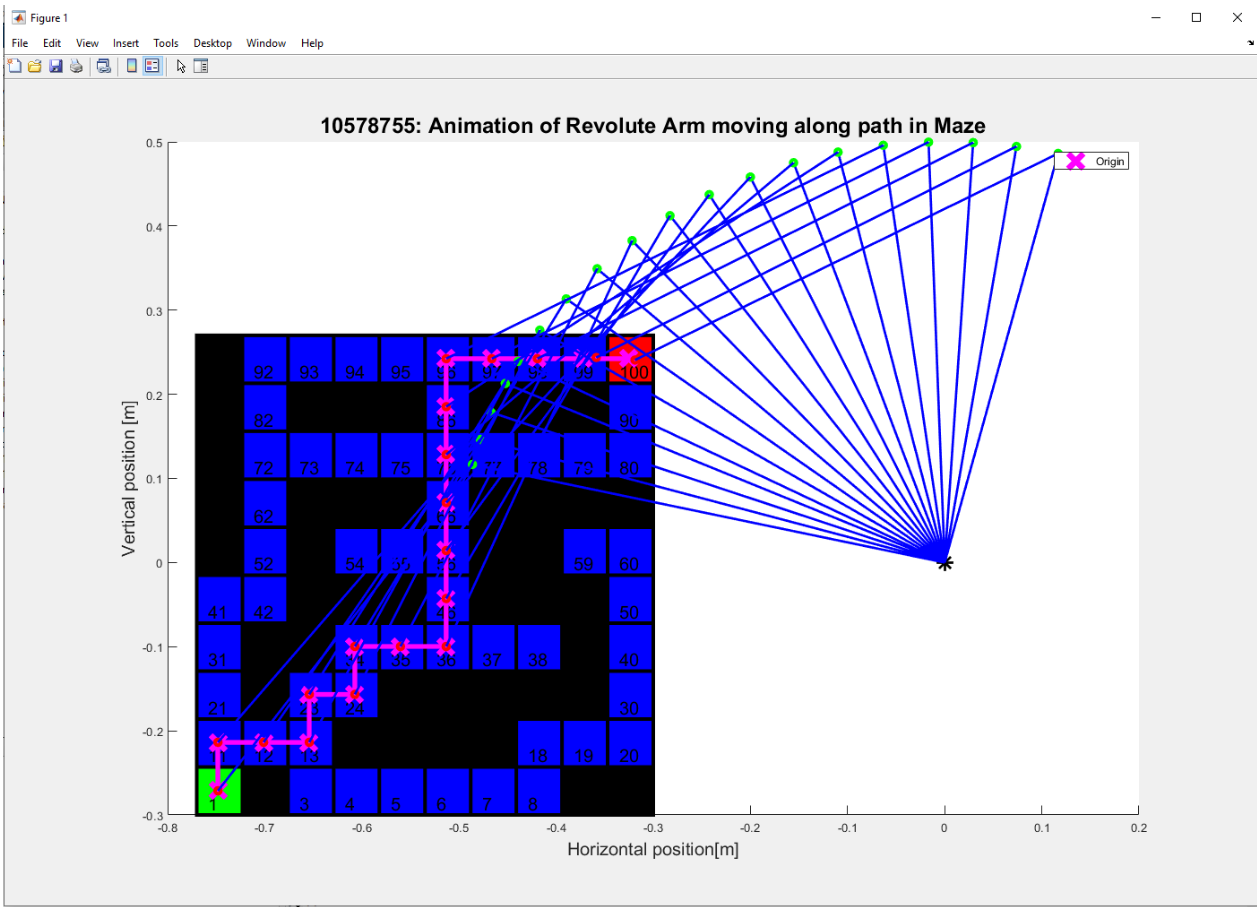

Figure 7: Screenshot of animation of 2-joint arm moving through path on maze

YOUTUBE LINK: https://www.youtube.com/watch?v=b5JOYfD4gvs