# Aint318 Coursework

Student ID 1055660

Date: 11/12/2019

ID:1055660

# Contents

# Introduction

The purpose of this coursework is to explore the implementation of reinforcement learning through QLearning, and supervised learning through Neural Networks. I will be applying these concepts to a task where I aim to move an 2D 2 joint arm through a maze.

The QLearning algorithm will implement an epsilon-greedy selection policy to find the optimal solution through an assigned maze. This path will act as coordinates for the inverse kinematics that the arm will follow.

The inverse kinematics will be calculated by the 2 layer neural network. It will use resultant arm positions as an input and the joint angles as a target, so that it can determine the relationship between the endpoints and joint angles to do the inverse of the forwards kinematics.

Once the arm network, and the maze pathfinder are both trained, they will be implemented in tandem. The arm will move through the maze, which has been scaled to its usable area.

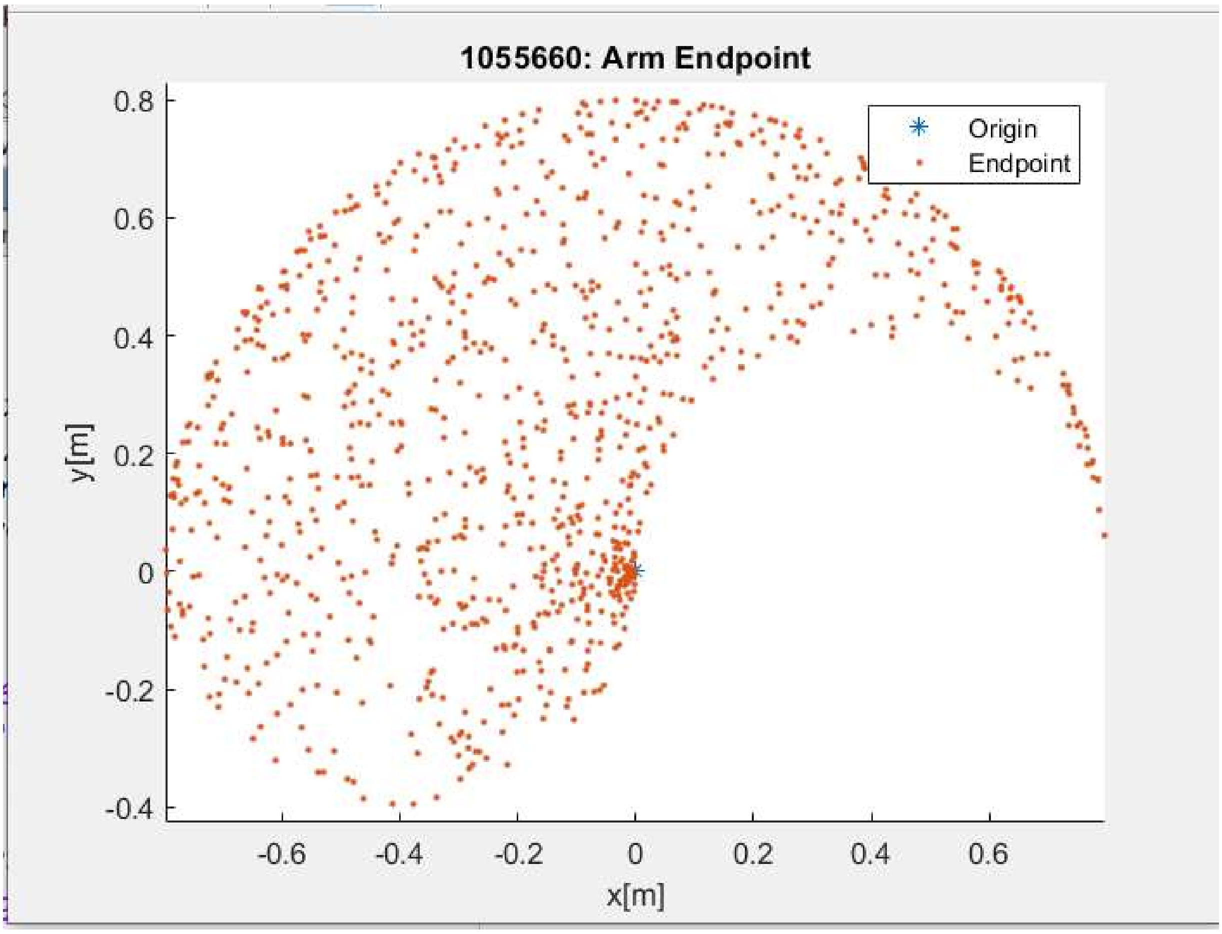
# 1. Training Data Generation:

## 1.1 Display workspace of revolute arm

Code:

```
clc
close all
clear all
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
samples = rand([2,1000])*pi; % generate 1000 random angles from 0 to pi for each joint
[P1,P2] = RevoluteForwardKinematics2D(length,samples,origin); %P1 is midpoint P2 is end point
%%Plot results:
figure
hold on
axis equal
title("1055660: Arm Endpoint")
xlabel('x[m]')
ylabel('y[m]')
plot(0,0,'*')
plot(P2(1,:),P2(2,:),'.')
legend ('Origin','Endpoint')
```

Output:




What can you say about the useful range of this arm?

There are limits to the motion of the endpoint as it moves away from the origin. It may be a good idea to scale the maze inside the main area of this range.

## 1.2. Configurations of a revolute arm

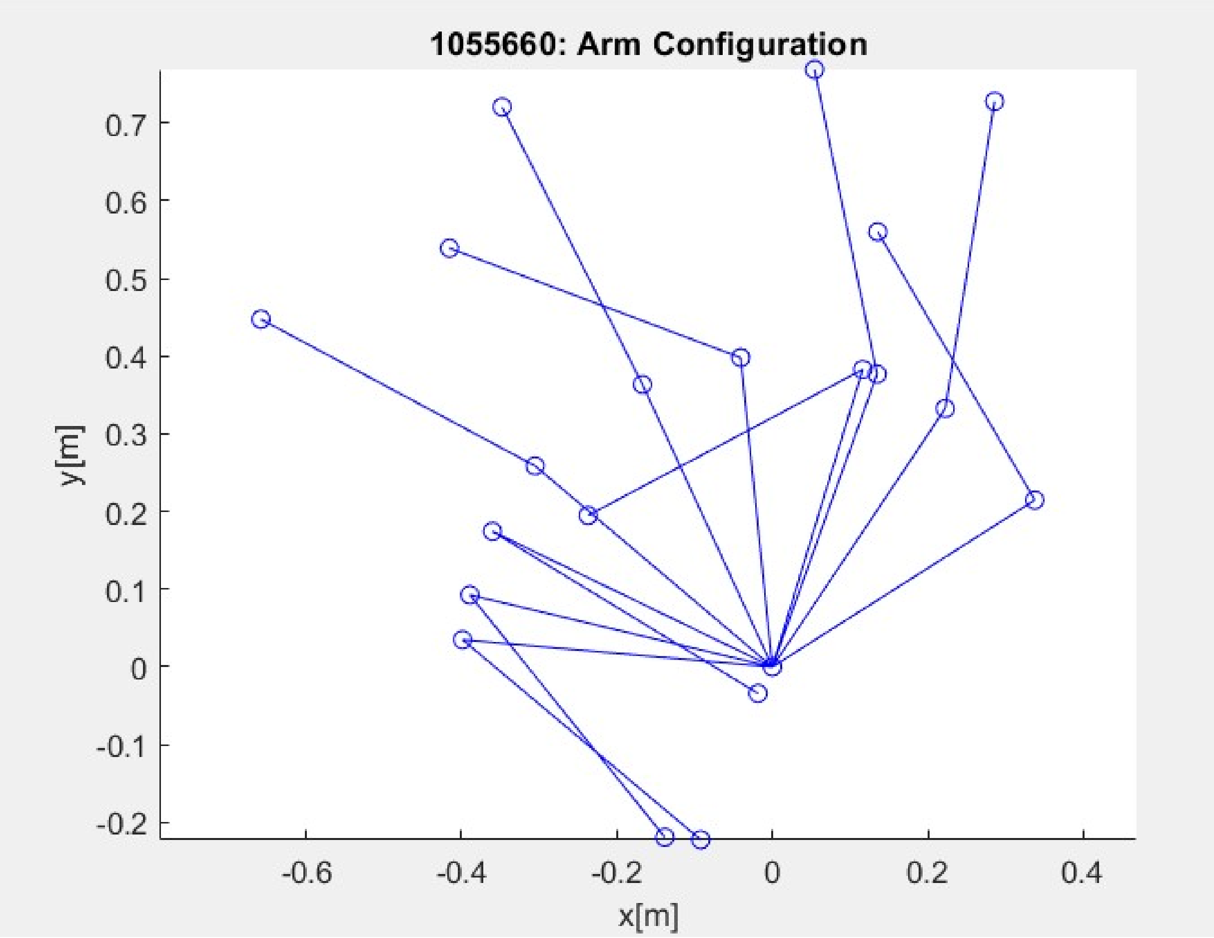
### Code

```
clc
close all
clear all
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
samples = 10;
data = rand([2,1000])*pi; % generate 1000 random angles from 0 to pi for each joint
[P1,P2] = RevoluteForwardKinematics2D(length,data,origin); %P1 is midpoint P2 is end point
figure
hold on
axis equal
title("1055660: Arm Configuration")
```

ID:1055660

```
xlabel('x[m]')
ylabel('y[m]')
for i = 1:samples
    x = [0,P1(1,i),P2(1,i)];
    y =[0,P1(2,i),P2(2,i)];
    plot(x,y,'-ob')
end
```

### Graph




# 2. Implement 2-layer network

## 2.1 Implement the network feedforward pass

### Inline function

```
function [activation,a2] = feedForwardPass(weight1, weight2, input)
input= augment(input);%augment input
net = weight1*input;
```

```
 a2 = arrayfun(@sigmoid,net);%calculate sigmoid activation for layer 1 (elementwise)
 a2Hat = augment(a2);% augment a2
activation =weight2*a2Hat;%calculate linear output activation
end
```

### internal functions

```
function [output] = augment (input)
%a function to append a row of ones to a matrix
[~,columns] = size(input);% get the number of columns of the matrix
output = [input;ones(1,columns)];%append a row of ones with an equal number of columns
end
```

```
function [output] = sigmoid(input)
%a function to calculate the sigmoid input of a single element
output = 1/(1+exp(-input));
end
```

## 2.2. Implement 2-layer network training

```
function [weight1, weight2,outError] = Training(weight1, weight2,learningRate, input, target)
%A batchwise function to tune the weights of the network and return the
%error
[O, a2]= feedForwardPass(weight1, weight2, input);% get the output and internal activations
outDelta = -(target-O); %calculate the delta term of the output layer (non-sigmoid)
weightHat = weight2(:,1:end-1);%get the weight without the bias term
backProp = (weightHat'*outDelta).*(a2.*(1-a2));% calculate the back propagation of the error
%calculate the Error Gradients for the weights
errorGradientWeight1=backProp*augment(input)';
errorGradientWeight2=outDelta*augment(a2)';
%update weights with the error
weight1 = weight1 - learningRate.*errorGradientWeight1;
weight2 = weight2 -learningRate.*errorGradientWeight2;
%calculate the error and update the array
e = outDelta.^2;
outError  = e(1,:) + e(2,:);
end
```

## 2.3 Train network inverse kinematics

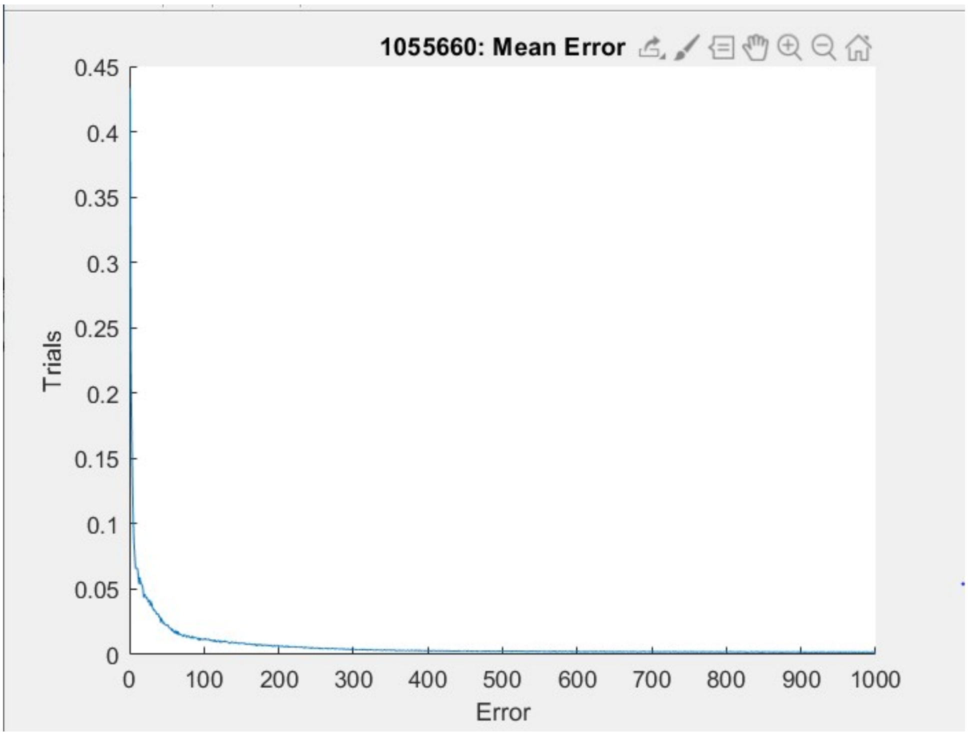
```
clc
close all
clear all
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
samples = 1000;% the number of samples used for training and testing
hidden = 12;%the number of hidden layers
input = 2;% number of system inputs
output = 2;% number of system outputs on the final layer

learningRate =0.001;
iterations = 50000;%number of times the network will be trained( a total of 50,000,000 data points
will be used to train it)
[weight1, weight2]=makeWeights(output,hidden, input);% generate the weight matrices
for i = 1:iterations
    %each iteration, a new set of training data will be generated. This
    %will help the network generalize as it won't be tuned to a specific
    %dataset
    targetData = rand([2,samples])*pi; % generate 1000 random angles from 0 to pi for each joint
    [~,inputs] = RevoluteForwardKinematics2D(length,targetData,origin);%get the endpoint positions
from the forward
    %kinematics to be used as inputs for the inverse
    [weight1, weight2,error]=Training(weight1, weight2,learningRate, inputs, targetData);%update
the trained weights and output error matrix
    meanError(i)=mean(error);% append the mean of the error matrix for future plotting
end
figure
hold on
title("1055660: Mean Error")
xlabel('Trials')
ylabel('Error')
plot(meanError)
```

### Internal Functions

```
function [weight1, weight2] = makeWeights(outputs, hidden, inputs)
%make the weight1 matrix where it is size (inputs+bias)X#ofHiddenLayers
%%inputs is the X and Y coordinates of the endpoint, the final output is
%%the angles of joints (inverse kinematics)
weight1=(2*rand(hidden,inputs+1))-1;
%make the weight2 matrix where it is size(#ofHiddenLayers+bias)XOutputs
weight2=(2*rand(outputs,hidden+1))-1;
end
```

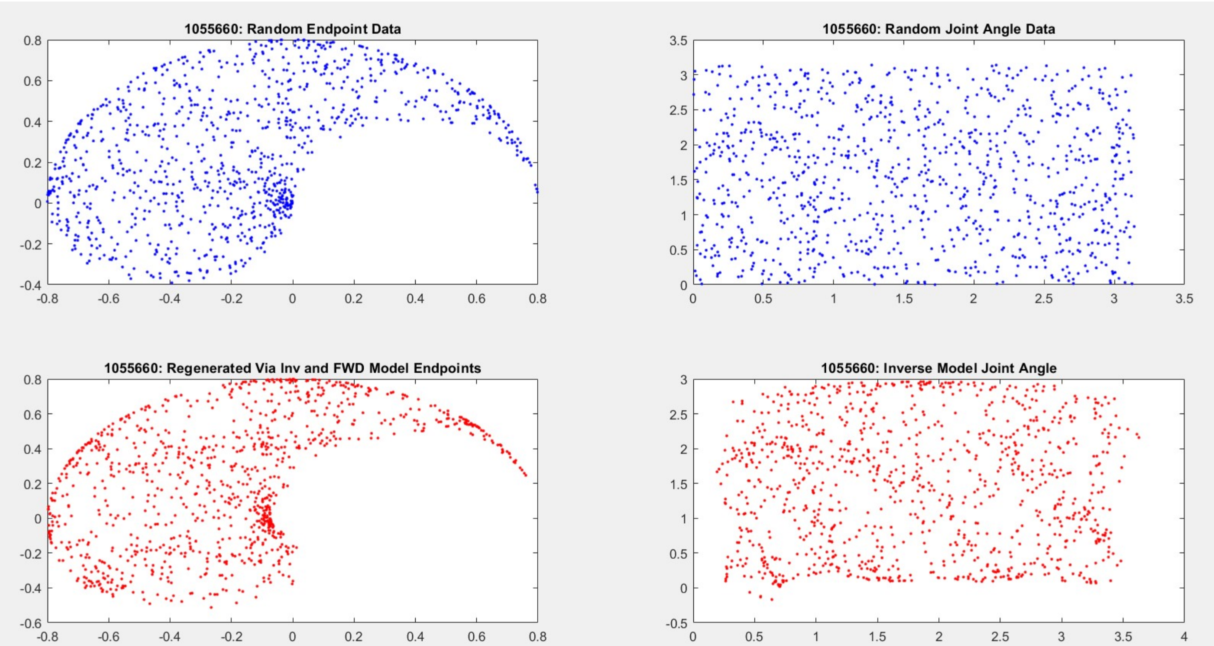
ID:1055660

## Plot




## 2.4. Test and interpret inverse model

### Neural network output



## Significance of this Plot:

This plot shows how well the neural network has been trained. As it's doing inverse kinematics, it's being shown the results of forward kinematics, and training the weights to change the input into the joint angles it's being given as a target. The regenerated graph (graph4) puts those joint angles back through forward kinematics to show the usable positions the arm can reach.

## Are there better datasets to interpret inverse model performance?

The better option would be to use a dataset of data selected only within the range of the center of the largest area of the endpoint range. If the maze is scaled to the area mentioned, then the arm will never have to reach its limits. This means that the arm will always be able to reach a given input and avoid the areas the network had more error in training. Furthermore, if you train specifically in that area, training can be faster and more thorough, as you do not need to train it on data it won't need.

## How can you make the dataset more representative of the maze task?

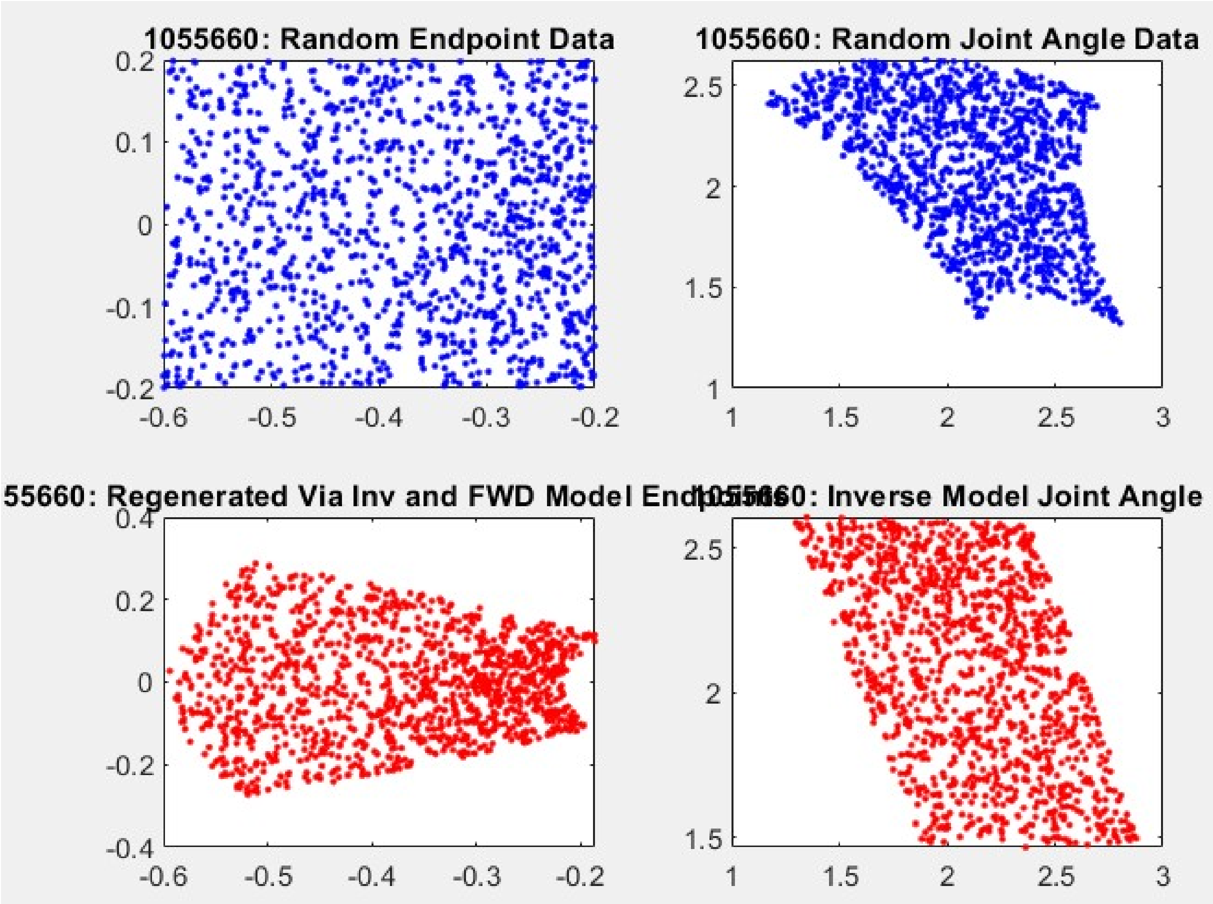
As previously stated, scaling the maze to the usable size of the arm and sampling only from that area will train the network to specifically handle that area, improving efficiency by not training the weights on useless data.

## Inline code

```
clc
close all
clear all
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
samples = 1000;% the number of samples used for training and testing
hidden = 12;%the number of hidden layers
input = 2;% number of system inputs
output = 2;% number of system outputs on the final layer

learningRate =0.001;
iterations = 50000;%number of times the network will be trained( a total of 50,000,000 data points
will be used to train it)
[weight1, weight2]=makeWeights(output,hidden, input);% generate the weight matrices
for i = 1:iterations
   %each iteration, a new set of training data will be generated. This
   %will help the network generalize as it won't be tuned to a specific
   %dataset
   targetData = rand([2,samples])*pi; % generate 1000 random angles from 0 to pi for each joint
   [~,inputs] = RevoluteForwardKinematics2D(length,targetData,origin);%get the endpoint positions
from the forward
   %kinematics to be used as inputs for the inverse
   [weight1, weight2,error]=Training(weight1, weight2,learningRate, inputs, targetData);%update
the trained weights and output error matrix
   meanError(i)=mean(error);% append the mean of the error matrix for future plotting
end
figure
```

```
hold on
title("1055660: Mean Error")
xlabel('Trials')
ylabel('Error')
plot(meanError)
%This will take the new weight matrices and test the data based off of the
%new testing data
randomAngles = rand([2,samples])*pi; % generate 1000 random angles from 0 to pi for each joint
[~,randomEndpoints] = RevoluteForwardKinematics2D(length,randomAngles,origin);%get the
endpoint positions from the forward
[networkAngles,~] = feedForwardPass(weight1, weight2, randomEndpoints);% generate a set of joint
angles
[~,networkEndpoints] = RevoluteForwardKinematics2D(length,networkAngles,origin);%generate the
endpoint of the
%arm from the inverse kinematics' joint angles
figure
hold on
subplot(221)
plot(randomEndpoints(1,:),randomEndpoints(2,:),'.b')
title("1055660: Random Endpoint Data")
subplot(222)
plot(randomAngles(1,:),randomAngles(2,:),'.b')
title("1055660: Random Joint Angle Data")
subplot(223)
plot(networkEndpoints(1,:),networkEndpoints(2,:),'.r')
title("1055660: Regenerated Via Inv and FWD Model Endpoints")
subplot(224)
plot(networkAngles(1,:),networkAngles(2,:),'.r')
title("1055660: Inverse Model Joint Angle")
```

## Inline code and plots for smaller sample size




```
clc
close all
clear all
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
samples = 10000;% the number of samples used for training and testing
hidden = 5;%the number of hidden layers
input = 2;% number of system inputs
output = 2;% number of system outputs on the final layer

minX= -0.6;
minY=-0.2;
maxX=-0.2;
maxY=0.2;

learningRate =0.0001;
iterations = 10000;%number of times the network will be trained( a total of 50,000,000 data points will be used to train it)
[weight1, weight2]=makeWeights(output,hidden, input);% generate the weight matricies
```

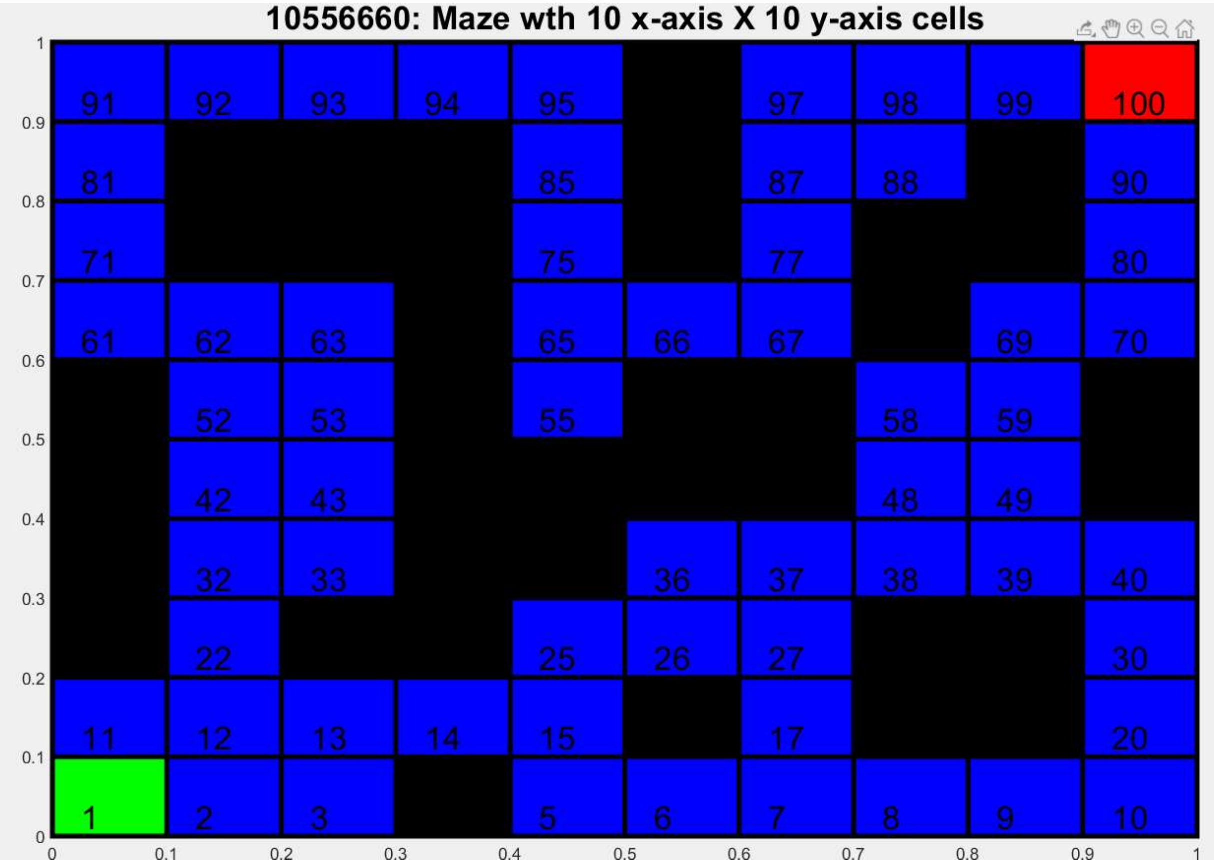
```
for i = 1:iterations
   %each iteration, a new set of training data will be generated. This
   %will help the network generalize as it wont be tuned to a specific
   %dataset
   if mod(i,500) ==0
      i
   end
  initialSamples = rand([2,samples])*pi; % generate 1000 random angles from 0 to pi for each joint
  targetData = initialSamples;
   [~,inputs] = RevoluteForwardKinematics2D(length,targetData,origin);%get the endpoint positions
from the forward
   a=find(inputs(1,:)<=maxX&inputs(1,:)>=minX);
   inputs = inputs(:,a);
   targetData = targetData(:,a);
   b=find(inputs(2,:)<=maxY&inputs(2,:)>=minY);
   inputs = inputs(:,b);
   targetData = targetData(:,b);
   %kinematics to be used as inputs for the inverse
   [weight1, weight2,error]=Training(weight1, weight2,learningRate, inputs, targetData);%update
the trained weights and output error matrix
   meanError(i)=mean(error);% append the mean of the error matrix for future plotting
end
figure
hold on
title("1055660: Mean Error")
xlabel('Trials')
ylabel('Error')
plot(meanError)
%This will take the new weight matricies and test the data based off of the
%new testing data
randomAngles = rand([2,samples])*pi; % generate 1000 random angles from 0 to pi for each joint
[~,randomEndpoints] = RevoluteForwardKinematics2D(length,randomAngles,origin);%get the
endpoint positions from the forward
a=find(randomEndpoints(1,:)<=maxX&randomEndpoints(1,:)>=minX);
randomEndpoints = randomEndpoints(:,a);
randomAngles = randomAngles(:,a);
b=find(randomEndpoints(2,:)<=maxY&randomEndpoints(2,:)>=minY);
randomEndpoints = randomEndpoints(:,b);
randomAngles = randomAngles(:,b);
[networkAngles,~] = feedForwardPass(weight1, weight2, randomEndpoints);% generate a set of joint
angles
[~,networkEndpoints] = RevoluteForwardKinematics2D(length,networkAngles,origin);%generate the
endpoint of the
%arm from the inverse kinematics' joint angles
figure
hold on
```

ID:1055660

```
subplot(221)
plot(randomEndpoints(1,:),randomEndpoints(2,:),'.b')
title("1055660: Random Endpoint Data")
subplot(222)
plot(randomAngles(1,:),randomAngles(2,:),'.b')
title("1055660: Random Joint Angle Data")
subplot(223)
plot(networkEndpoints(1,:),networkEndpoints(2,:),'.r')
title("1055660: Regenerated Via Inv and FWD Model Endpoints")
subplot(224)
plot(networkAngles(1,:),networkAngles(2,:),'.r')
title("1055660: Inverse Model Joint Angle")

save('weights.mat', 'weight1','weight2');%save weights for faster implementation
```

## 3. Path Through a Maze

### Game board:

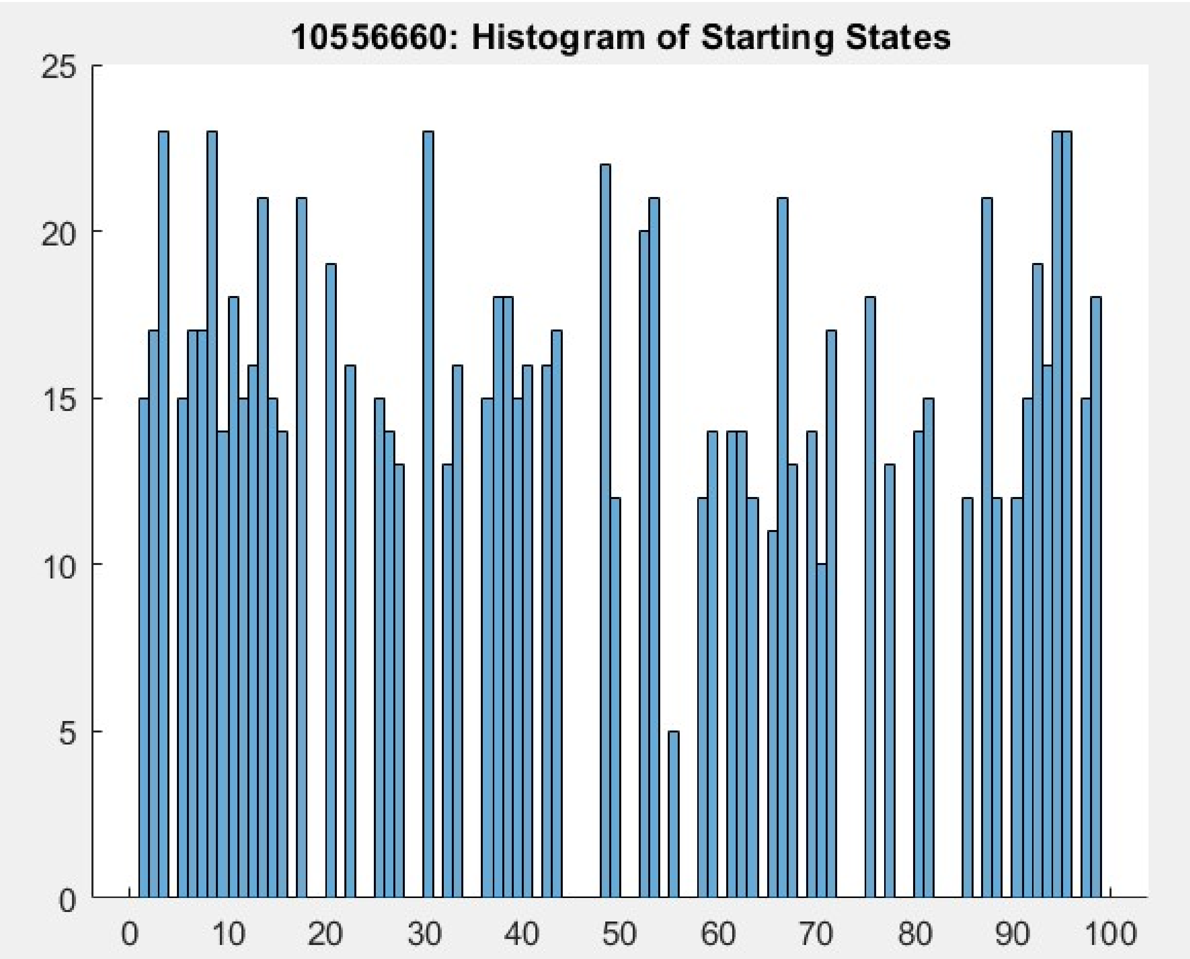


## Set Starting State and Blocked Locations

```
% specify start location in (x,y) coordinates
        startLocation=[1 1];

        % specify end location in (x,y) coordinates
        endLocation=[10 10];

        % specify blocked location in (x,y) coordinates
        f.blockedLocations = [4 1; 6 2; 8 2; 9 2; 1 3; 3 3; 4 3; 8 3; 9 3; 1 4; 4 4; 5 4; 1 5; 4 5; 5 5; 6 5;
7 5; 10 5; 1 6; 4 6; 6 6; 7 6; 10 6;...
           4 7; 8 7; 2 8; 3 8; 4 8; 6 8; 8 8; 9 8; 2 9; 3 9; 4 9; 6 9; 9 9; 6 10;];
```

## 3.1 Random start state

### Histogram:




### Comment on how the displayed state occurrences align with the maze.

Any spaces that are empty, correspond with blocked and end states. As these are all random frequency of appearance indicates nothing.

## Inline code

*Main code:*

```
% test random start
  startingIterations=1000;
  for i = 1:startingIterations
   histo(i) = maze.RandomStartingState();% add 1000 starting states to a list
  end
  figure
  hold on
  histogram(histo,[1:99])%plot the states in a histogram with whole number edges
  title("10556660: Histogram of Starting States")
```

*Random Starting State Function:*

```
    % function  computes a random starting state
    function startingState = RandomStartingState(f)
        startingState = fix(rand*98)+1;% generate a number from 1-99(leaving out end state)
        [x,y] = f.stateToCoords(startingState);% convert your state into a set of coordinates
       [rows,~] =size(f.blockedLocations); % get the number of rows of the blockedLocation matrix
       for i = 1:rows%iterate through each row of blockedLocation
        if f.blockedLocations(i,:) == [x,y]%if any rows are equal to our X,Y coordinates
            startingState = f.RandomStartingState();% then recursively call this function starting
state isn't blocked
            break
        end
       end
    end
```

*State to Coordinate Function*

```
    %function to give the coordinates of a given state
    function [x,y] = stateToCoords(f,state)
        state = state-1;%start arrays from 0
        x = mod(state,10)+1; %get the remainder from dividing by 0
        y = fix(state/10)+1;% divide by 0 without remainder

    end
```

## 3.2 Build a Reward Function

## Inline code

```
    % reward function that takes a stateID and an action
    function reward = RewardFunction(f, state, action)

       if state == 99 && action == 2 %if you are in 99 and go right
          reward = 10;
       elseif state == 90 && action == 1 %if you are in 90 and go up
          reward = 10;
       else % any other state and action
          reward = 0;
       end
```

```
    end
```

### 3.3. Generate the transition matrix (5 marks)

#### Inline Code:

*Generator Script:*

```
%% a script to generate a transition matrix autmatically and save it to a file for faster load times
blockedLocations = [4 1; 6 2; 8 2; 9 2; 1 3; 3 3; 4 3; 8 3; 9 3; 1 4; 4 4; 5 4; 1 5; 4 5; 5 5; 6 5; ...
7 5; 10 5; 1 6; 4 6; 6 6; 7 6; 10 6;4 7; 8 7; 2 8; 3 8; 4 8; 6 8; 8 8; 9 8; 2 9; 3 9; 4 9; 6 9; 9 9; 6 10;];
% a list of blocked locations to correctly transition
states = 100;
actions= 4;
transitionMatrix = zeros(states,actions);%initialize an empty matrix to be filled
[bx,~]=size(blockedLocations);%get the number of blocked locations
%a list of action numbers and their directions:
%1 = north(up) ^
%2= east(right) >
%3= south(down) v
%4= west(left) <
for a = 1:actions
   for s = 1:states%for each state and action
      [x,y] = stateToCoords(s);%get the coordinates of the states
      switch a% update next state based off the aforementioned directions
         case 1
            nextX=x;
            nextY=y+1;
         case 2
            nextX=1+x;
            nextY=y;
         case 3
            nextX=x;
            nextY=y-1;
         case 4
            nextX=x-1;
            nextY=y;
      end
      for b = 1:bx%for each blocked location
         if (nextX>10
||nextX<1||nextY>10||nextY<1||((nextX==blockedLocations(b,1))&&(nextY==blockedLocations(b,2))))
            %if move exceeds bounds, or would enter a blocked state,
            %return the original state
            nextX=x;
            nextY=y;
         end
      end
```

```
    transitionMatrix(s,a)=coordsToState(nextX,nextY); %store the state value of the next state
in the matrix
  end
end

save('transitionMatrix.mat', 'transitionMatrix');%save the matrix to file for a direct import into
the main code

%%%%%%%%%%%functions below%%%%%%%%%%%%%%
%function to give the coordinates of a given state
function [x,y] = stateToCoords(state)
  state = state-1;%start arrays from 0
  x = mod(state,10)+1; %get the remainder from dividing by 0
  y = fix(state/10)+1;% divide by 0 without remainder

end
%function to give the state from the given coordinates
function state = coordsToState(x,y)
  state = ((y-1)*10)+x;
end
```

*Import into main*

```
function f = BuildTransitionMatrix(f)

      f.tm=load('transitionMatrix.mat', 'transitionMatrix');%pull the pre-generated transition
matrix from file

    end
```

### 3.4 Initialize Q values

#### Inline code:

```
% init the q-table
function f = InitQTable(f)

    % allocate
    f.QValues = rand(f.xStateCnt * f.yStateCnt, f.actionCnt)/10;
    %initialize Q values to a random number between 0 and 0.1
end
```

### 3.5 Implement Q-Learning Algorithm

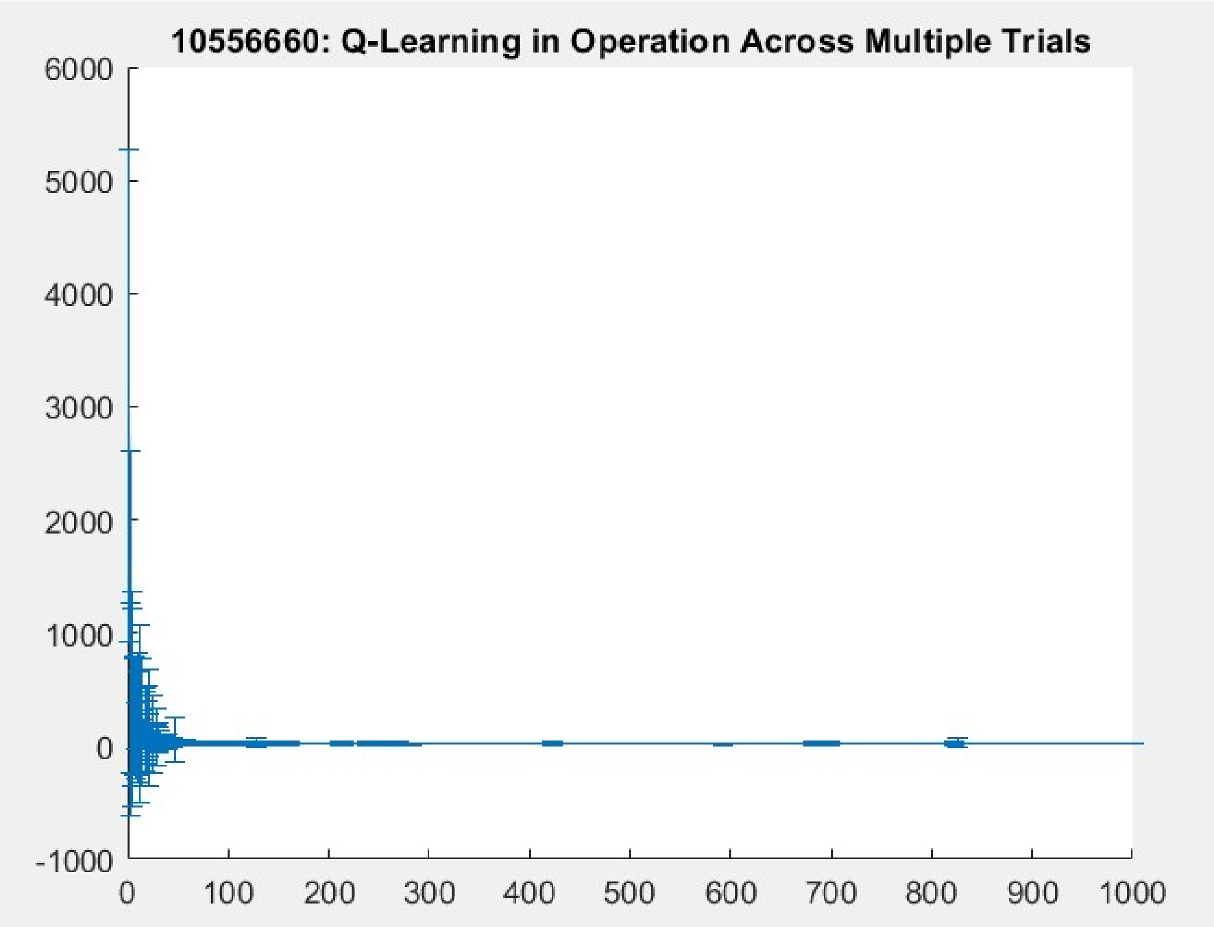
#### Inline Code:

```
trials = 100;% number of trials for this experiment
  episodes =1000;%number of episodes per trial
  explorationRate=% the rate at which the algorithm takes a random action
  temporalDiscount =
  learningRate =
  stepscat=[];
```

```
    for i=1:trials
        maze = maze.InitQTable();%each trial init a clean random Q table
        for j = 1:episodes
            step = 0;% reset the number of steps and reward values for this episode
            reward = 0;
            state = 1;%RandomStartingState();
            while reward == 0
            oldState = state;% save the old state
            step = step +1;% increment the number of steps take
            action =actionSelect(maze.QValues,state,explorationRate)%generate the action the
algorithm will take
            %using epsilon greedy
            state = maze.tm(state,action);% get the new state from the transition table
            reward = maze.RewardFunction(oldState,action);%check if the algorithm gives a reward
            maze.QValues(oldState, action)= maze.QValues(oldState, action) + learningRate...
                *(reward+temporalDiscount*max(maze.QValues(state,:))-max(maze.QValues(oldState,:)));
            %update your Qtable for the states and actions
            end
            steps(j) = step;% add the number of steps to a total for this trial
        end
        stepscat =[stepscat;steps];%concatenate each step count per trial
    end
    for i =1:episodes
        means(i)=mean(stepscat(:,i));%take the mean for all step i per episode
        STDeviations(i)=std(stepscat(:,i));%take the standard deviation for all step i per episode
    end
 %%%%%%%%%%%%%%%%%%%%%%%%%Functions%%%%%%%%%%%%%%%%%%%%%%%%%

function [action] = actionSelect (table, state,explorationRate)
    chance = rand(1);%chance to explore
    if chance < explorationRate%if exploring
        action = randperm(4,1);%give an action in whole numbers between 1 and 4
    else %if not exploring
        [~ , action] = max(table(state, :));%your action is the Highest Q value per the given state
    end
end
```

## 3.6 Run Q-Learning

### Graph:




### Observations:

The graph shows that as the algorithm is trained, the number of steps quickly drops off. The standard deviation can fluctuate a bit randomly however, that is most likely due to the random exploration rate.

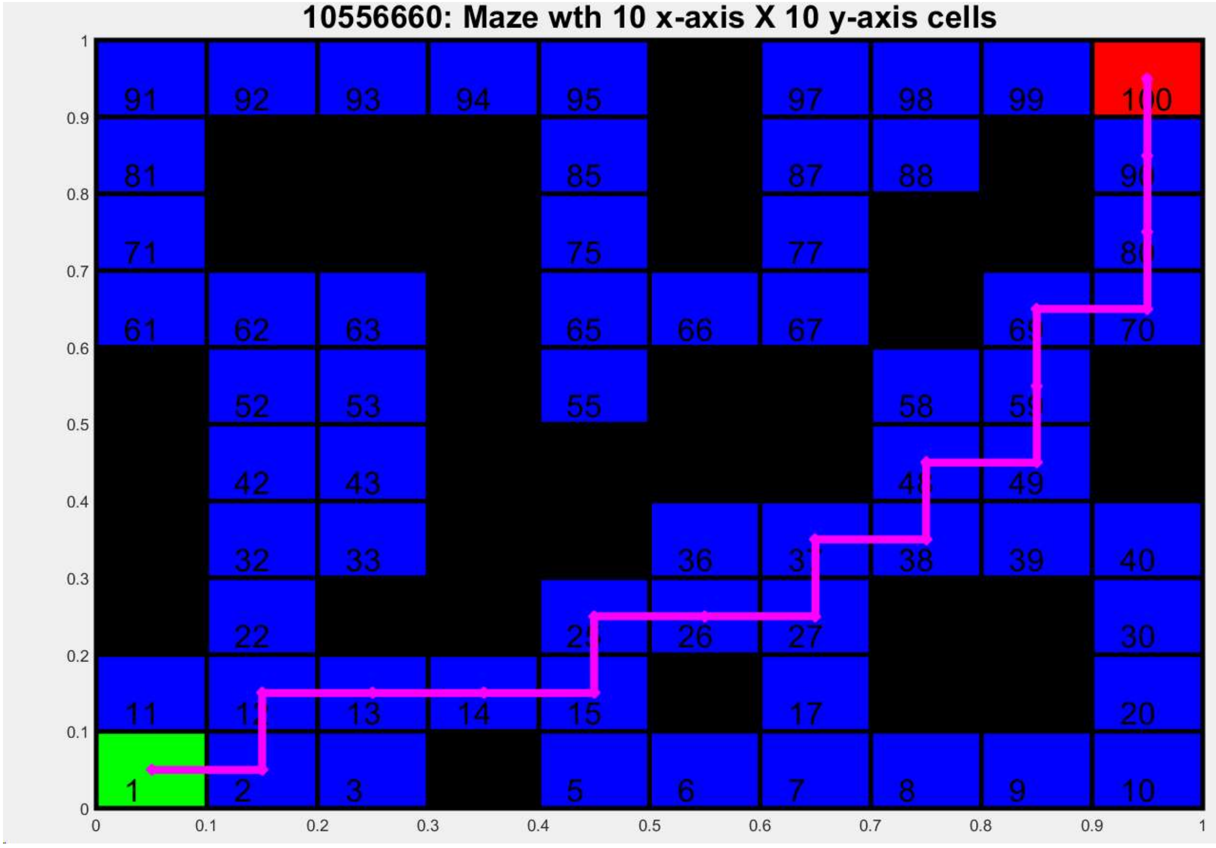
### Inline code:

```
trials = 100;% number of trials for this experiment
   episodes =1000;%number of episodes per trial
   explorationRate=0.1;% the rate at which the algorithm takes a random action
   temporalDiscount = 0.9;
   learningRate = 0.2;
   totals=zeros(episodes);
   stepscat=[];
   for i=1:trials
      maze = maze.InitQTable();%each trial init a clean random Q table
      for j = 1:episodes
         step = 0;% reset the number of steps and reward values for this episode
         reward = 0;
```

```
        state = 1;%RandomStartingState();
        while reward == 0
        oldState = state;% save the old state
        step = step +1;% increment the number of steps take
        action =actionSelect(maze.QValues,state,explorationRate)%generate the action the
algorithm will take
        %using epsilon greedy
        state = maze.tm(state,action);% get the new state from the transition table
        reward = maze.RewardFunction(oldState,action);%check if the algorithm gives a reward
        maze.QValues(oldState, action)= maze.QValues(oldState, action) + learningRate...
            *(reward+temporalDiscount*max(maze.QValues(state,:))-max(maze.QValues(oldState,:)));
        %update your Qtable for the states and actions
        end
        steps(j) = step;% add the number of steps to a total for this trial
      end
      stepscat =[stepscat;steps];%concatenate each step count per trial
   end
   for i =1:episodes
      means(i)=mean(stepscat(:,i));%take the mean for all step i per episode
      STDeviations(i)=std(stepscat(:,i));%take the standard deviation for all step i per episode
   end
   %Graph your means and standard deviations
   figure
   hold on
   errorbar(means, STDeviations)
   title("10556660: Q-Learning in Operation Across Multiple Trials")
  %%%%%%%%%%%%%%%%%%%%%%%%%%Functions%%%%%%%%%%%%%%%%%%%%%%%

function [action] = actionSelect (table, state,explorationRate)
   chance = rand(1);%chance to explore
   if chance < explorationRate%if exploring
      action = randperm(4,1);%give an action in whole numbers between 1 and 4
   else %if not exploring
      [~ , action] = max(table(state, :));%your action is the Highest Q value per the given state
   end
end
```

## 3.7. Exploitation of Q-values

### Graph




### Inline code

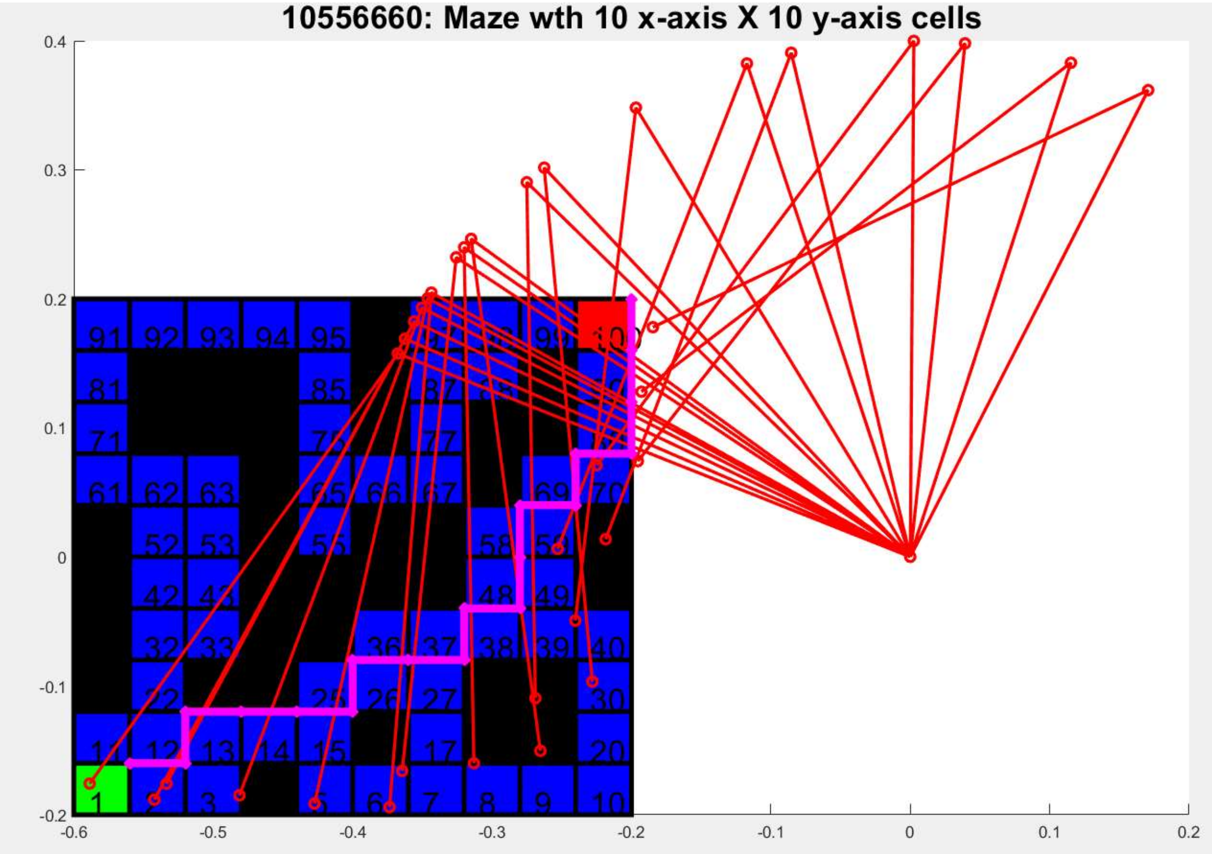
```
%%%%%%%%%%%%%No Explore run%%%%%%%%%%%%%%%%%%%%%%%%%
%reset reward, state and step number
reward = 0;
i=0;
coords=[];
state =1;
%implement the previous code with a exploration rate of 0
%and no training. Keep notes of the XY coordinates of the given states
while reward == 0
        i=i+1;
        states(i) = state;% save the old state
        step = step +1;% increment the number of steps take
        action =actionSelect(maze.QValues,state,0);%implement the epsilon Greedy with 0
exploration rate
        state = maze.tm(state,action);% get the new state from the transition table
        reward = maze.RewardFunction(states(i),action);%check if the algorithm gives a reward
        [x,y]=maze.stateToCoords(states(i));
        coords=[coords,[x;y]];
```

```
end
coords = [coords,[10;10]];% since the code doesn't record the end state, tack it on the end
coords = (coords-0.5)/10;%scale the coordinates and make them look a little nicer
plot(coords(1,:),coords(2,:), 'mx-', 'linewidth',5)% plot them
```

# 4.Move Arm Endpoint Through Maze

## 4.1. Generate kinematic control to revolute arm

### Non animated arm motion:




### Inline code:

Note: this code uses pre-generated and saved weights and Qvalues to speed up execution.

```
close all
clear all
clc

% scaled maze to the best trained area of the kinematics
minX= -0.6;
minY=-0.2;
maxX=-0.2;
maxY=0.2;
limits = [minX maxX; minY maxY;];
```

```
% build the maze
maze = CMazeMaze10x10(limits);

% draw the maze
maze.DrawMaze();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load the q-table
maze = maze.loadQvalues();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build the transition matrix
maze = maze.BuildTransitionMatrix();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%load weights from file
load('weights.mat')
%reset reward, state and step number
origin = [0;0]; %base frame of kinematics
length = [0.4,0.4]; % length of links
step =0;
reward = 0;
i=0;
coords=[];
state =1;
%implement the previous code with a exploration rate of 0
%and no training. Keep notes of the XY coordinates of the given states
while reward == 0
        i=i+1;
        states(i) = state;% save the old state
        action =actionSelect(maze.QValues,state,0);%implement the epsilon Greedy with 0
exploration rate
        state = maze.tm(state,action);% get the new state from the transition table
        reward = maze.RewardFunction(states(i),action);%check if the algorithm gives a reward
        [x,y]=maze.stateToCoords(states(i));
        coords=[coords,[x;y]];
end
coords = [coords,[10;10]];% since the code doesn't record the end state, tack it on the end
%coords = (coords-0.5);%scale the coordinates and make them look a little nicer
coords(1,:)=(coords(1,:)*((maxX-minX)/10))-(abs(minX));
coords(2,:)=(coords(2,:)*((maxY-minY)/10))-(abs(minY));
[angles,~]= feedForwardPass(weight1,weight2, coords);
[P1,P2] = RevoluteForwardKinematics2D(length,angles,origin);
for j = 1:i
   x = [origin(1),P1(1,j),P2(1,j)];
   y =[origin(2),P1(2,j),P2(2,j)];
   plot(x,y,'-or','linewidth',2)
end
```

```
plot(coords(1,:),coords(2,:), 'mx-', 'linewidth',5)
 %%%%%%%%%%%%%%%%%%%%%%%%%%Functions%%%%%%%%%%%%%%%%%%%%%%%%%%

function [action] = actionSelect (table, state,explorationRate)
   chance = rand(1);%chance to explore
   if chance < explorationRate%if exploring
      action = randperm(4,1);%give an action in whole numbers between 1 and 4
   else %if not exploring
      [~ , action] = max(table(state, :));%your action is the Highest Q value per the given state
   end
end
```

## 4.2 Animated revolute arm movement

Links: https://youtu.be/fM-I6DbB_AA

### Inline code

```
close all
clear all
clc

% scaled maze to the best trained area of the kinematics
minX= -0.6;
minY=-0.2;
maxX=-0.2;
maxY=0.2;
limits = [minX maxX; minY maxY;];

% build the maze
maze = CMazeMaze10x10(limits);

% draw the maze
maze.DrawMaze();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load the q-table
maze = maze.loadQvalues();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build the transition matrix
maze = maze.BuildTransitionMatrix();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%load weights from file
load('weights.mat')
%reset reward, state and step number
origin = [0;0]; %base frame of kinematics
```

```
length = [0.4,0.4]; % length of links
step =0;
reward = 0;
i=0;
coords=[];
state =1;
%implement the preivous code with a exploration rate of 0
%and no training. Keep notes of the XY coordinates of the given states
while reward == 0
        i=i+1;
        states(i) = state;% save the old state
        action =actionSelect(maze.QValues,state,0);%implement the epsilon Greedy with 0
exploration rate
        state = maze.tm(state,action);% get the new state from the transition table
        reward = maze.RewardFunction(states(i),action);%check if the algorithm gives a reward
        [x,y]=maze.stateToCoords(states(i));
        coords=[coords,[x;y]];
end
coords = [coords,[10;10]];% since the code doesn't record the end state, tack it on the end
coords = (coords-0.5);%scale the coordinates and make them look a little nicer
coords(1,:)=(coords(1,:)*((maxX-minX)/10))-(abs(minX));
coords(2,:)=(coords(2,:)*((maxY-minY)/10))-(abs(minY));
[angles,~]= feedForwardPass(weight1,weight2, coords);
[P1,P2] = RevoluteForwardKinematics2D(length,angles,origin);

v = VideoWriter('armEndpoint.avi');
v.FrameRate=10;
open(v);
for j = 1:i
   maze.DrawMaze();
   set(gca,"color",'k')
   xlim([minX maxX+0.2])
   ylim([minY maxY])
   hold on
   x = [origin(1),P1(1,j),P2(1,j)];
   y =[origin(2),P1(2,j),P2(2,j)];
   %axis ([minX maxX+0.2 minY maxY])
   plot(coords(1,:),coords(2,:), 'mx-', 'linewidth',5)
   plot(x,y,'-or','linewidth',2)
   %xlim([minX maxX+0.2])
   %ylim([minY maxY])
   writeVideo(v,getframe(gca));
   close(gcf)
end
close(v)
 %%%%%%%%%%%%%%%%%%%%%%%%%%Functions%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [action] = actionSelect (table, state,explorationRate)
   chance = rand(1);%chance to explore
   if chance < explorationRate%if exploring
      action = randperm(4,1);%give an action in whole numbers between 1 and 4
   else %if not exploring
      [~ , action] = max(table(state, :));%your action is the Highest Q value per the given state
   end
end
```