AINT351 Machine Learning 2019 Dr Ian Howard
Coursework: Planning and control of a 2-D revolute arm

IMPORTANT NOTICE

Submit this report to the DLE by Thursday 12th December 2019 (4pm)
You will receive feedback within 20 working days

Please only submit your report for this coursework in PDF format. You **must** include your student number in a header on every page of the report and **also include** your student number **in the document filename**! Please use the report template file if possible.

This coursework should contain a few pages of explanation as required (although it can be more than this is necessary) and a set of images showing the plots requested by the individual parts of the practical exercises, as well as embedded equations and Matlab code that you developed to implement your solutions.

The report ***must*** be a **stand-alone document**. Please embed images in-line in the report.

In order to show video an animation of the final maze movement task, please use Internet links to videos that you have been uploaded to YouTube. Please ***do not*** submit more than the single PDF file for the coursework.

Overview

This coursework consists of five tasks that will result in a simple kinematic controller that will operate a simulated 2-joint revolute arm and trace a path through a maze.

1. Data generation for learning the control of the arm. This task requires using an Matlab forward kinematic model of the arm.

2. Implement the feedforward and training passes of a 2-layer neural network.

3. Learn the inverse kinematic model of the arm using a 2-layer NN.

4. Calculate the trajectory through a maze through reinforcement learning.

5. Use the NN-based inverse kinematic model of the arm learned in Task 3 on to the trajectory generated by the RL-based trajectory planner in Task 4.

- First download and expand the file AINT351Coursework 2019.zip.
- There is a main Matlab script that provides the first steps you need in maze generation that will be used with your Q-Learning algorithm.
- There is a maze class containing both useful functions to build and draw a maze. It also has several empty Matlab functions that you will also need to fill-in with your code during this assignment.
- There is an empty function to generate the forward kinematics that you will need to fill-in with your code during this assignment.

1. Training data generation (marks 10)

You will first use a function that implements the forward kinematics for the specified 2-joint revolute arm. This involves setting its joint angles and calculating the location of its endpoint. The forward kinematic model will be use it to generate a suitable dataset to train a mapping in the opposite direction - namely from end effector position to arm joint angles.

Consider the two-joint revolute arm shown in Fig. 1. Given we know the lengths of the links, the endpoint of the arm is determined by the arm's joint angles. The end effector position in (x,y) extrinsic cartesian coordinates is given by

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

The relationship between the input and angles and the output end effector position is known as the forward kinematics of the arm. Thus, given the input angle we can use the forward kinematics to tell us the endpoint of the arm.

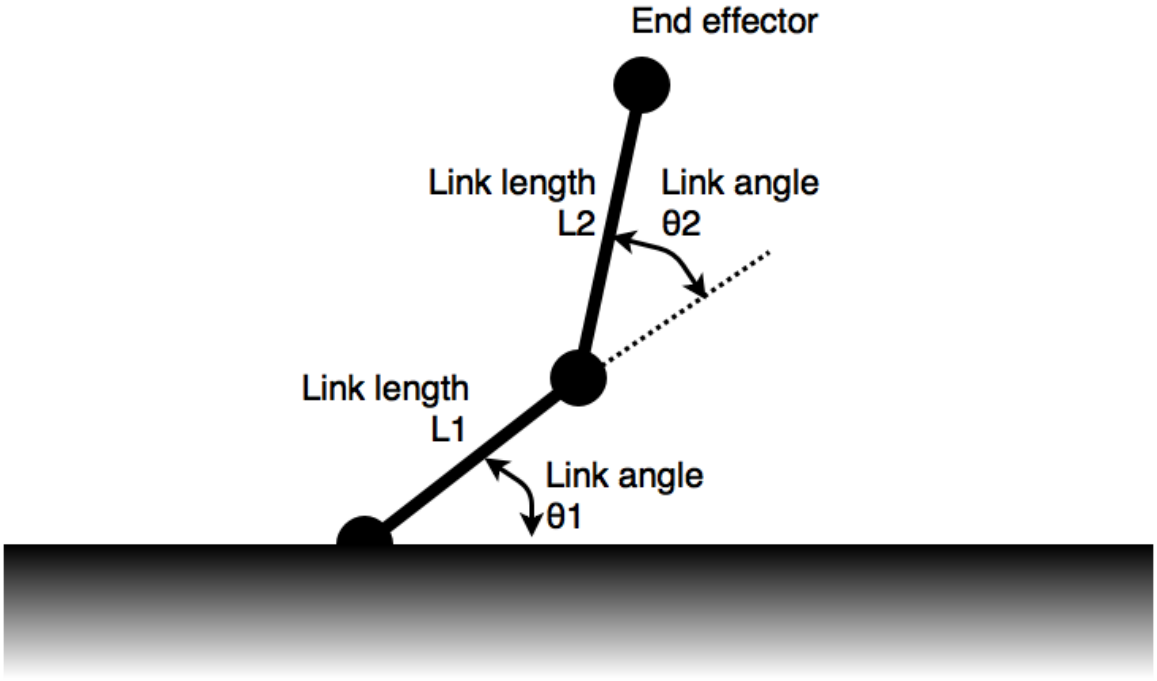

Figure 1. Plan view of a revolute 2-joint arm. The lower link has a length L1 and the upper link length L2. The position of the end effector is affected by both link angles θ_1 and θ_2.

- You have been supplied with a Matlab function RevoluteForwardKinematics2D, with parameters shown below, that takes arm lengths in the array L, the input angles in the array theta the location of the arm base in the array origin.
- The function returns the location of the end point p2 (x_{p2}, y_{p2}) and the elbow joint p1 (x_{p1}, y_{p1})

[p1 p2] = RevoluteForwardKinematics2D(armLen, theta, origin)

Where

armLen = [L_1 L_2];
theta = [θ_1 θ_2];
origin = [x_o y_o];
p2 = [x_{p2} y_{p2}];
p1 = [x_{p1} y_{p1}];

1.1. Display workspace of revolute arm (5 marks)

- We wish first to show the range of the endpoint when the arm angles move between 0 to π radians.
- Set the arm lengths to 0.4m and the arm base origin to (0, 0).
- Generate 1000 uniformly distributed set of training angle data points in the array theta over the range 0 to π radians using the Matlab rand command
- Also generate a testing dataset of 1000 samples
- Run he RevoluteForwardKinematics2D function with the specified parameters to generate the corresponding endpoint locations.
- Plot the endpoint positions and the arm origin position too.
- This should generate a plot like that shown below in Fig. 2.

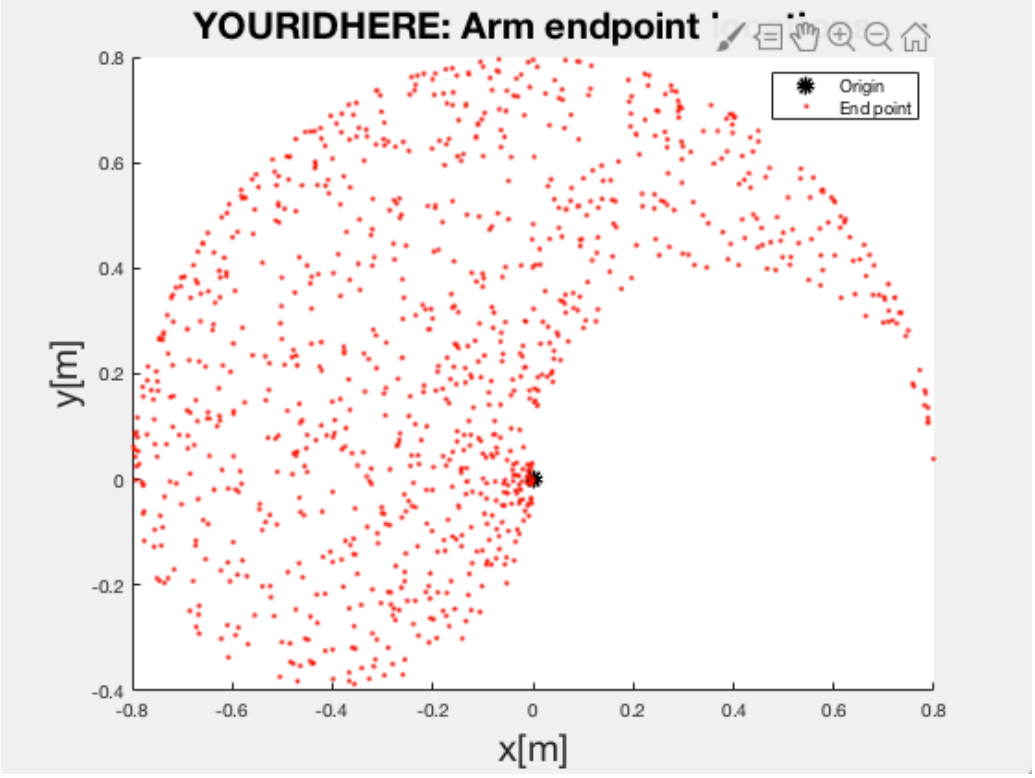

Figure 2. Endpoint locations (red dots) for 1000 data points when the arm angles are randomly distributed between 0 to π radians. Origin of the arm (its shoulder joint) also shown as the black dot at (0,0).

- What can you say about the useful range of this arm?
- Include your plot and commented Matlab solution code embedded in the report document

1.2. Configurations of a revolute arm (5 marks)

- To illustrate arm configurations, keeping other parameters as before, now generate just 10 uniformly distributed set of angle data points between 0 – π radians using the Matlab rand command
- Again, run the RevoluteForwardKinematics2D function with the specified parameters to generate the corresponding elbow and endpoint locations.
- Plot the elbow and endpoint locations and the arm sections between them.
- This should generate a plot like that shown in Fig. 3.
- Include your plot and commented Matlab solution code embedded in the report document

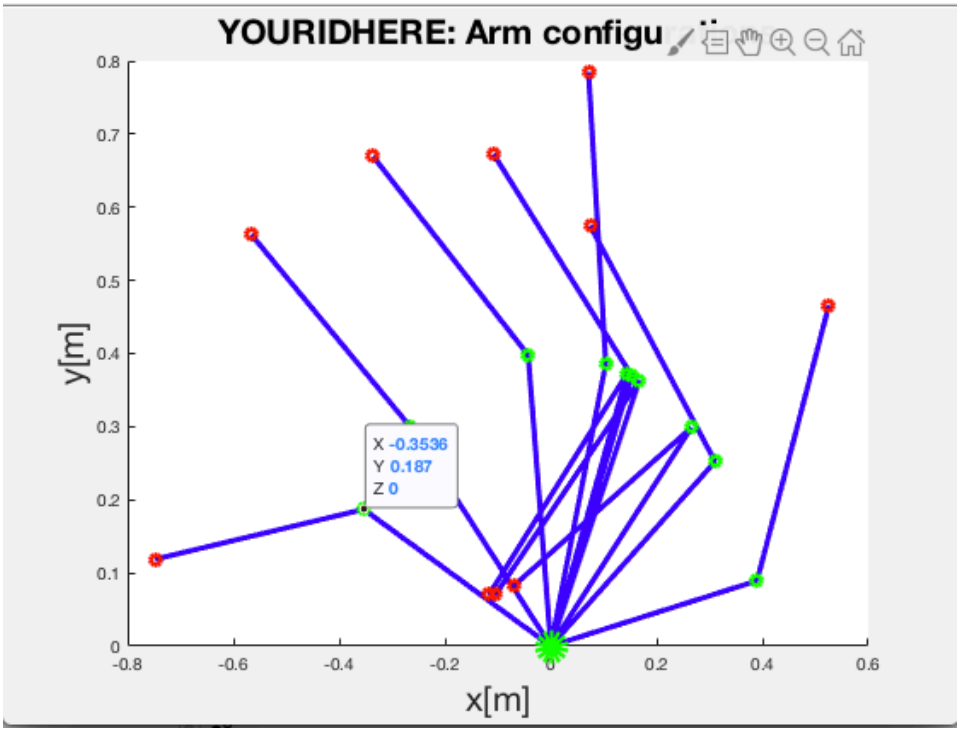

Figure 3. Arm configurations for 10 data points when the arm angles are randomly distributed between 0 to π radians. Endpoint locations are red, elbow is green, and the arm is in blue. Origin of the arm also shown at (0,0) in black.

2. Implement 2-layer network (marks 80)

You are now required to build a multi-later perceptron with 2 layers perceptron, with a sigmoid hidden layer and linear output layer, which will be used to learn and implement the robot arm's inverse kinematics

- Consider a 2- layer network with n_h nodes in its hidden layer with a sigmoidal non-linearity at their outputs, and **a single linear output.** The network schematic is depicted in Fig.4 below.
- The network has 2 inputs and fully connected weights in layers 1 and 2 specified by the weight matrixW_1 and W_2.
- To elegantly realize a bias term for the network, we augment the network input and weight vectors appropriately.

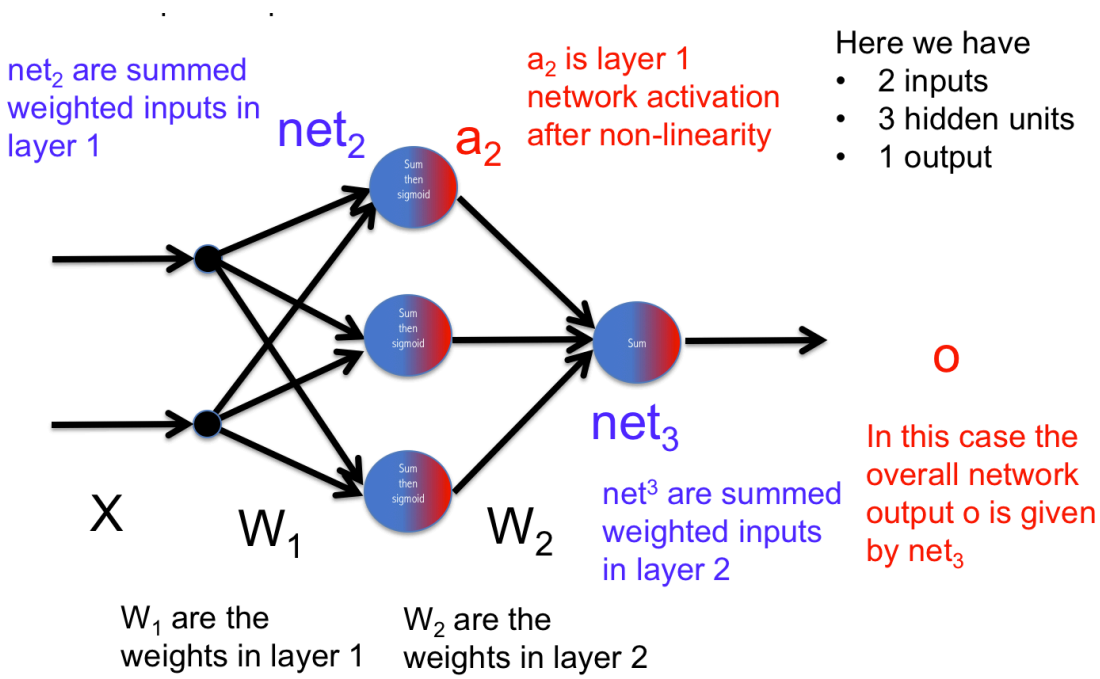

Figure 4. Two-layer network

2.1. Implement the network feedforward pass (10 marks)

- Implement the 2-layer network feedforward pass (with n_h hidden units and a linear output) in Matlab.

- Given the weight matrix, this will generate the output activation for a given input vector.
- Include commented Matlab inline in the report document

2.2. Implement 2-layer network training (30 marks)

- Implement the generalized delta rule in Matlab to train a network
- Include your Matlab code inline in the report document

2.3. Train network inverse kinematics (15 marks)

In the case of our simple 2-join revolute arm, it is possible to derive an analytic expression to calculate the inverse kinematics. However, as the number of degrees of freedom of the arm increase, this becomes increasingly difficult to derive. Here we will train network to perform this task.

Running the forward kinematics of the robotics arm results in a dataset with input joint angle and their corresponding end effector positions. This dataset is therefore ideal to train a supervised regression algorithm to map between the two in ***either*** direction.

This next part of this work is to learn the inverse kinematics using the 2-layer neural network you developed in the first part of this assignment.

- Train your linear output 2-layer network with the end effect positions as input and joint angles as outputs.
- To do so you will need two networks with a single linear output each. Alternatively, you can use a single network with two outputs.
- Plot the error as training proceeds.

2.4. Test and interpret inverse model (25 marks)

Test your inverse model looking at how the *2-link arm* endpoint workspace is mapped back to itself

- To do so test the network by running a forward pass on position data to generate predicted arm angles
- This could be the training data but ideally should be another testing set

- Then run the forward kinematics of predicted angles and get corresponding end points
- Plot the joint angles and endpoints and compare with those you started with. You should get something like the plots shown in Fig. 5. Discuss the significance of these plots.
- Is there better datasets to interpret inverse model performance?

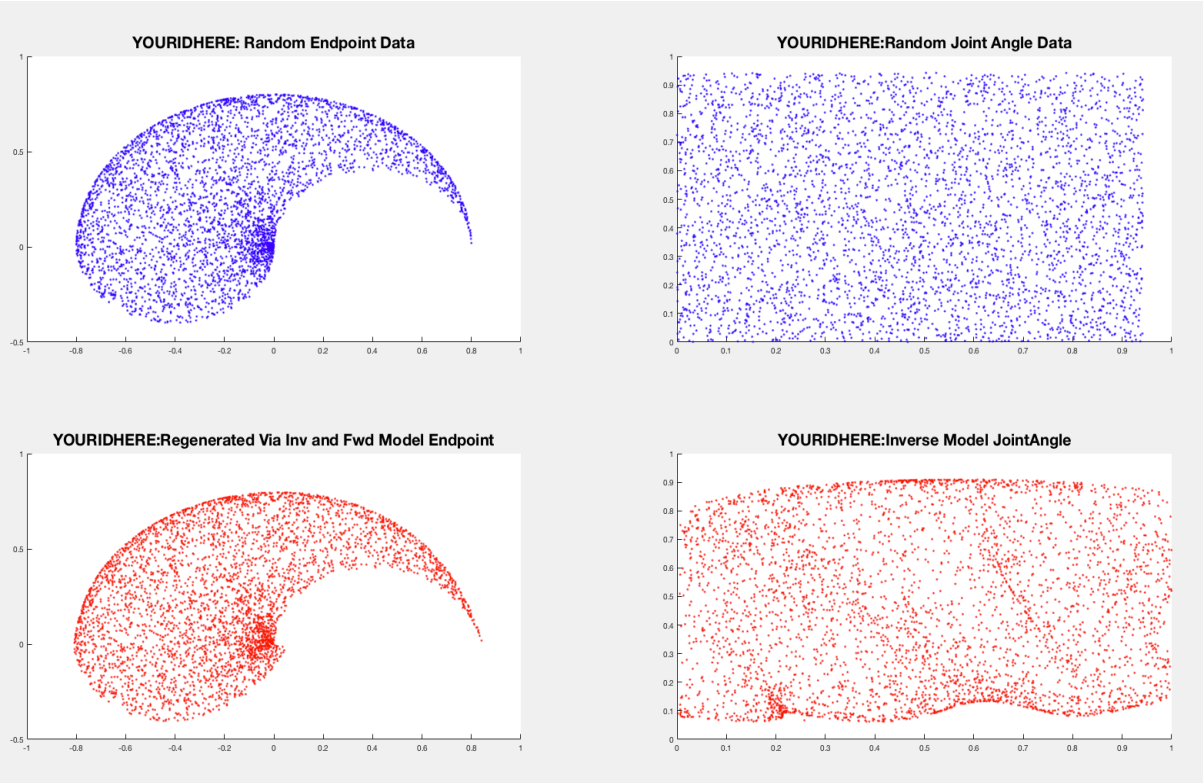

Figure 5. Training workspace of the endpoint of 2-link arm (left panel) is first mapped to the joint angles using the 2-link arm inverse model. These angles are then used with the forward model of the 2-link arm to show how well the inverse model operates. Ideally all points should map back to themselves.

- Experiment with different numbers of hidden units, training times and learning rates.
- Also experiment with different training data sets.
- You should look at the effect of randomizing the order of presentation of patterns.
- How can you make the dataset more representative of the maze task?
- Also, consider how the data points in extrinsic space are distributed. Are they uniform? If not, how can you make them more uniform?

3. Path through a maze (marks 75)

Now you will tackle a movement planning problem. You are provided with a simple 2-dimentional grid representing a maze in Fig. 6.

You must use and implement this particular maze!

You are required to develop a Q-learning algorithm from first principles that can generate a path through a maze. After this task has been achieved, you will then use the path trajectory to generate the control angles for the 2-joint arm and generate an animation of it moving between the start and goal states.

We now consider the maze you must use which is shown in Fig. 6.

- The states are numbered 1 to 100 as indicated.
- The goal of the Q-learning algorithm is finding the optimal path from a given goal state to the red goal state.
- At the goal there is a reward of 10, all other states the reward is 0.
- The green state is one possible start state, that we will make use of later.
- The black cells are blocked states that cannot be entered.
- In general, an action from a state can involve moving north, south, east of west, or staying put if it would lead to a blocked state or edge.

You are now required to implement Q-learning for the maze shown in Fig. 6.
Note that during training:

- A step is the execution of a single action of the Q-learning algorithm. That is, from its current state, it executes a greedy action with a small random element, often the moving to a new state, and updating a Q-value.
- An episode starts from a random starting state and runs for many steps until the algorithm reaches the goal state of the maze, at which point the episode terminates.
- A trial starts with naïve Q-values and involves running the Q-learning algorithm for a given number of episodes, updating the same Q-value function across episodes.
- An experiment involves running a given number of trials.

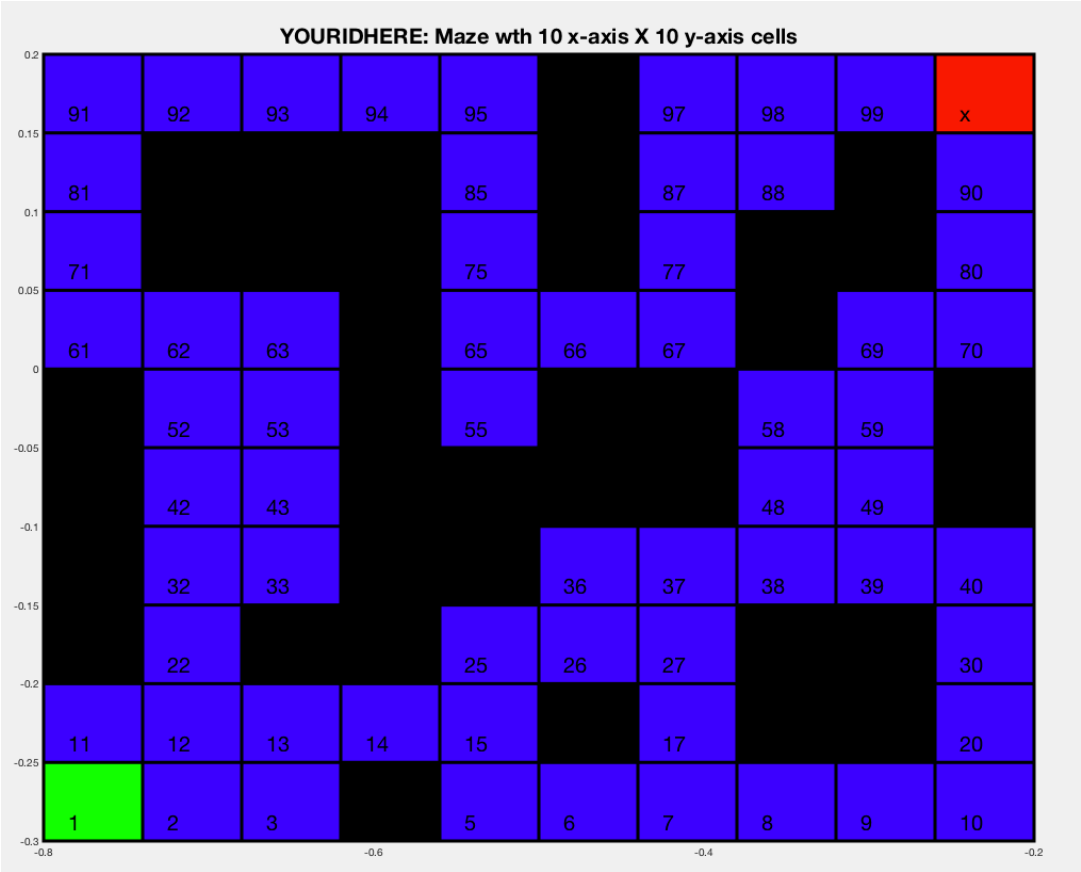

Figure 6. 10 x 10 grid maze with specified start state shown in green (no reward). The goal states shown in red and has a reward of 10. The blue states are allowed states with no reward associated with them. The black cells are blocked states and cannot be entered. An action towards a blocked or out of cell state results in the same state the current state.

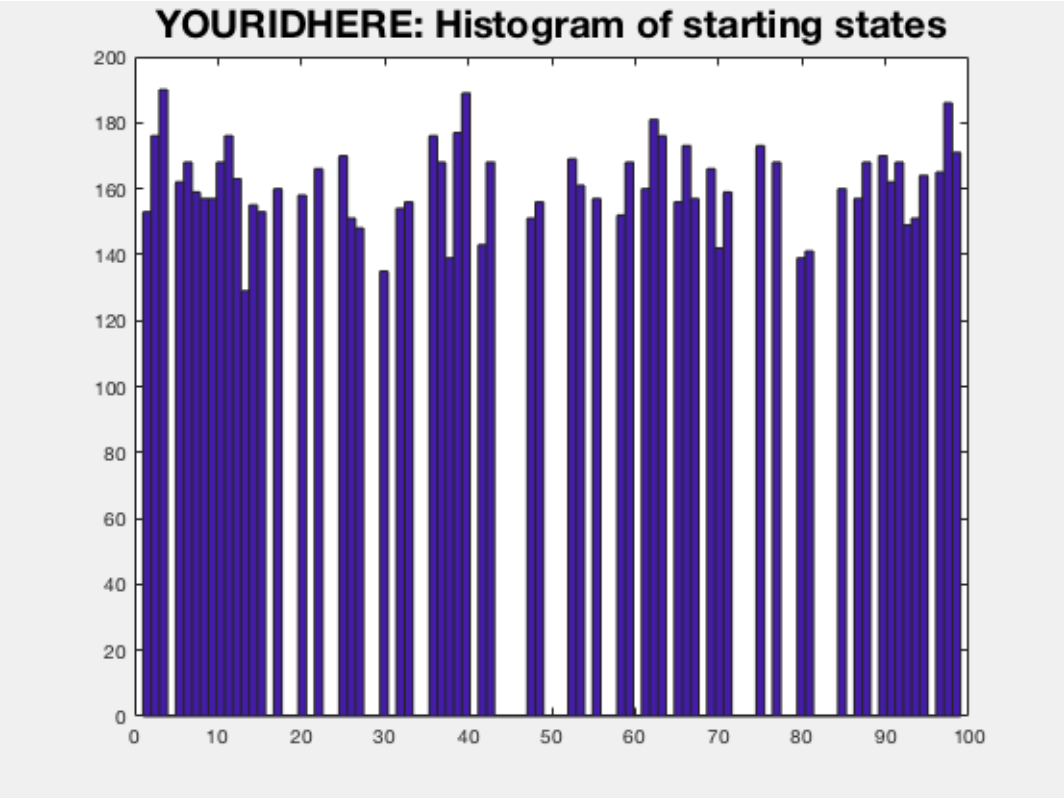

Figure 7. Histogram of 1000 randomly generated starting states, plotted with 100 bins corresponding to one of the maze's 100 states.

3.1. Random start state (5 marks)

- Write a function to generate a random starting state in the maze. NB: The start state may not be a blocked state or the goal state.
- Generate 1000 starting states and plot a histogram using the Matlab histogram function to check your function is working. You should end up with a histogram like the one shown in Fig 7.
- Comment on how the displayed state occurrences align with the maze.

3.2. Build a reward function (5 marks)

- Specify the reward function for the maze
- Tip: States under an action that lead to movement into the goal states lead to reward

3.3. Generate the transition matrix (5 marks)

- Specify a transition matrix for the maze.
- **Tip**: It's easy to generate it automatically after you represent the maze appropriately using a matrix.

3.4. Initialize Q-values (5 marks)

- Initialize the Q-values matrix to sensible numbers.

3.5. Implement Q-learning algorithm (30 marks)

To implement Q-learning you will also need to

- In this assignment your algorithm will need an outer loop to run 100 trials.
- In this assignment within a trial, you will need a loop to run 1000 episodes.
- Within each episode you will need a loop to run for a given number of states until the goal state is reached.
- The number of steps needed in an episode is indicative of how good the Q-learning policy is becoming, so it can be used as an indication of algorithm performance on the training data.

3.6. Run Q-learning (15 marks)

- Run the Q-learning algorithm using:
  An exploration rate of 0.1
  A temporal discount rate gamma of 0.9
  A learning rate alpha of 0.2.
- Analyse the performance of your Q -learning algorithm on the maze by running an experiment with 100 trials of 1000 episodes
- Generate an array containing the means and standard deviations of the number of steps required to complete each episode
- Plot the mean and standard deviation across trials of the steps taken against episodes. Describe what you find.
- You should end up with a plot like the one shown in Fig.8.

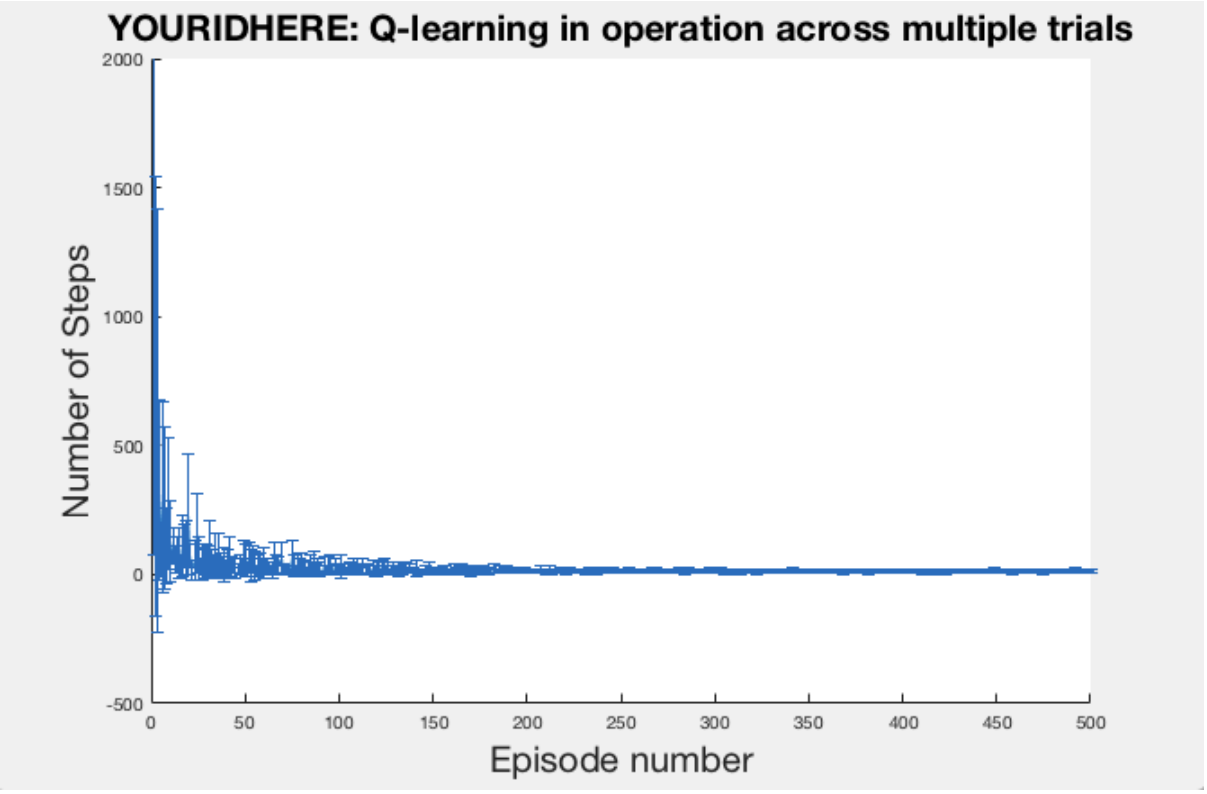

Figure 8. Mean and SD of steps plotted against episode.

We now wish to make use of the learned Q-values to compute the optimal path through the maze for any give stating point.

3.7. Exploitation of Q-values (10 marks)

- Record the Q-table at the end of a training trial.
- Write an exploitation function that makes use of the Q-values and makes greedy action selection ***without exploration.***
- Select the starting state as the green state shown on the maze
- Record the visited states for this episode.
- Convert the state into a 2-D coordinate that you can plot out in a matrix.
- This should take the form of a 2xN matrix where the first dimension relates to the (x,y) coordinates of the data points, and the second to the N of steps in the episode
- Try to plot out the path over a drawing of the maze, like as shown in Fig. 9

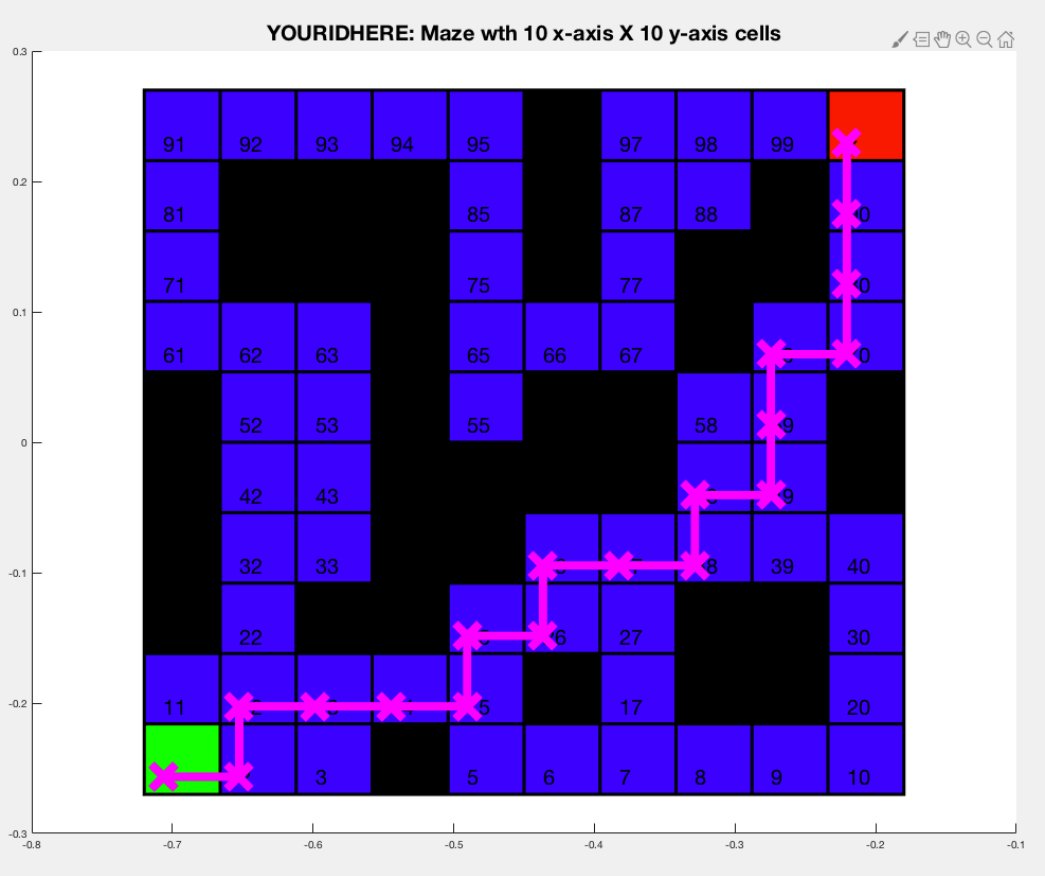

Figure 9. Path through the maze found by Q-learning algorithm. The path (shown in magenta starts) from the green cell at state 1 and finally reaches the red destination cell at state 100.

4. Move arm endpoint through maze (marks 35)

4.1. Generate kinematic control to revolute arm (15 marks)

- Finally use the maze path to specify the endpoint trajectory of the 2-joint revolute arm.
- Use the inverse kinematic neural network you trained earlier to generate the arm's joint angles.
- Tip: ensure you have scaled the maze so that it fits into the workspace of the revolute arm!

- Use the forward kinematic function with these angles as input to calculate the arm elbow and endpoint positions.

4.2. Animated revolute arm movement (20 marks)

- Generate an animation of the endpoint of the revolute arm moving through the maze. Also draw the arm as well.
- Produce a video of your results and put a link to the video uploaded to YouTube in your report.
- Tip A screenshot of my implementation of this animation is shown in Fig. 10.

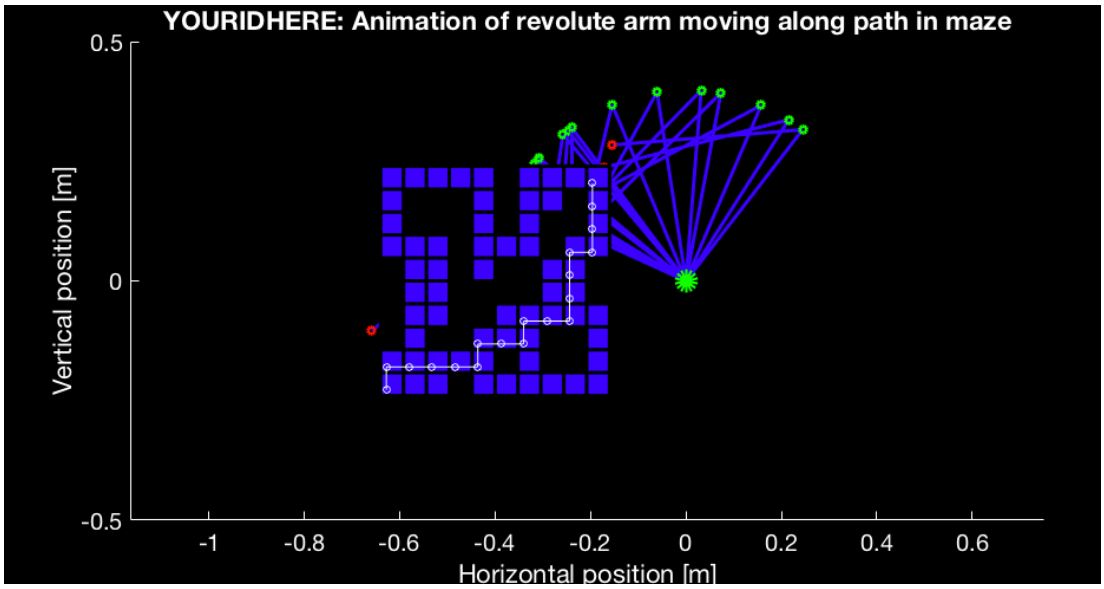

Figure 10. Screenshot of animation of 2-joint arm moving through path on maze. The base of the arm is represented by the large green dot and is located at the coordinates (0,0). The elbow joint is represented by the small green circle. The arm endpoint is given by the red dot and the arm links are shown in blue. Notice that the maze has been scaled to fit into workspace of the robotic arm.