# Faculty of Science and Engineering

Coursework– 2022/23 Academic Year

**Module Code: AINT351Z**

**Module Title: Machine Learning**

**Module Leader:**

**School: School of Engineering, Computing and Mathematics**

**DEADLINE FOR SUBMISSION:**

## IMPORTANT NOTICE

Submit a zip file containing your PDF report and all your MATLAB code to the DLE by Thursday 15th December 2022 (4pm)

You will receive feedback within 21 working days

## Report

**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 sentences of explanation as required per task (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 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 or your OneDrive. Please **do not** submit more than the single PDF file for the coursework.

## MATLAB code submission

Make sure you write a main program script for each task you demonstrate, named as requested in the corresponding assessment sections. Each script must run from the MatlabCodeAndReportxxx directory. Make sure any functions needed for your scripts to run are included in that directory. If the code does not run, you will lose marks.

## Submit you PDF report and MATLAB code in a single zip file.

To do so, please put your PDF report and all MATLAB code in a directory names as follows:

MatlabCodeAndReportxxx

where xxx is your student ID. Then generate a zip file and upload it to the DLE.

## MODULE AIMS

- Introduce the area of machine learning (ML), covering unsupervised, supervised and reinforcement learning from Bayesian perspectives.
- Give students the theory behind a range of learning techniques and how to apply these to build representations of data in systems that make decisions and predictions.
- Enable students to analyse real datasets and control real-time systems.

## ASSESSED LEARNING OUTCOMES:

At the end of the module the learner will be expected to be able to:

1. Have an understanding in the basic concepts in unsupervised, supervised and reinforcement learning.
2. Demonstrate the ability to implement and apply machine learning techniques to make decisions and predictions on real data sets and real-time control scenarios.

## MARKING SCHEME

**<30% - Fail**
Computational implementation in MATLAB is poor or missing.
Results from MATLAB solutions to tasks are incorrect or are missing.
Descriptions and discussions are missing.
Discussion lacks critical insights.

**30% – 49% Unsatisfactory**
Computational implementation in MATLAB could be better.
Some results from MATLAB solutions are incorrect or are missing and others could be better.
Descriptions and discussions attempt to address issues.
Discussion demonstrates little critical insights.

**50% – 59% Good**
Computational implementation in MATLAB is good.
Most results from MATLAB solutions are present and generate appropriate results.
Descriptions and discussions adequately address issues.
Discussion demonstrates some critical insights.

**60% – 69% Very Good**
Computational implementation in MATLAB is very good
Most results from MATLAB solutions are present and work well.
Descriptions and discussions clearly address issues.
Discussion demonstrates significant critical insights.

**70% – 85% Excellent**
Computational implementation in MATLAB is excellent.
Results from MATLAB solutions are excellent.
Descriptions and discussions are precise and concisely address issues.
Discussion demonstrates widespread critical insights.

**>85% Outstanding**
Computational implementation in MATLAB is outstanding.
Results from MATLAB solutions are outstanding.
Descriptions and discussions meticulously and concisely address issues.
Discussion demonstrates extensive critical insights.

## 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 a 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 AINT351Coursework2022.zip.

- There are 4 script files that you must fill in to perform the coursework tasks.
- There is also a maze class containing both useful functions to build and draw a maze. It has several empty MATLAB functions that you will also need to fill-in with your code during this assignment.
- These and any other functions you develop must be included in the zip file upload to the DLE

| Script | Description |
|---|---|
| Main_P1_GenerateArmData2022.m | Generate arm data to train inverse model |
| Main_P2_TrainNeuralNetwork2022.m | Train a neural network to implement inverse model |
| Main_ P3_RunGridworld2022.m | Find path through maze. Some code is included in the script to help you use the supplied maze class and develop a Q-learning algorithm quickly. |
| Main_P4_MoveArmViaMaze2022.m | Move arm through maze |

## 1. Training data generation

### 1.1. Display workspace of the revolute arm

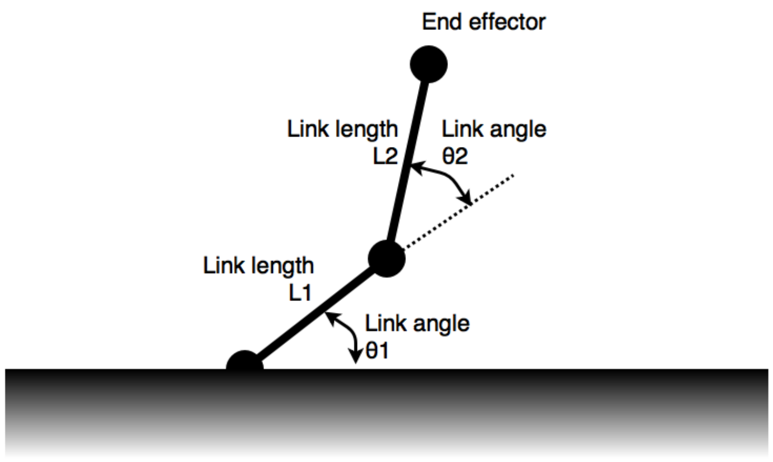

Figure 1. Plan view of a revolute 2-joint arm. The lower link has a length L1 and the upper link length L2. The position of the end effector is affected by both link angles $\theta_1$ and $\theta_2$.

You should put all the code you develop to perform this task in the MATLAB script Main_P1_GenerateArmData2022.m. Calling this function should generate the testing and training data and generate the requested endpoint plot. It is recommended that the script saves the generated data to a file using the MATLAB save command, so you can then use the data in subsequent tasks by reading it in with the MATLAB load command.

You have been supplied with a MATLAB function RevoluteForwardKinematics2D that calculated the forward kinematics of a 2 joint revolute arm shown in Fig. 1. The function 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 = [$\theta_1$ $\theta_2$];

origin = [$x_o$ $y_o$];

p2 = [$x_{p2}$ $y_{p2}$];

p1 = [$x_{p1}$ $y_{p1}$];

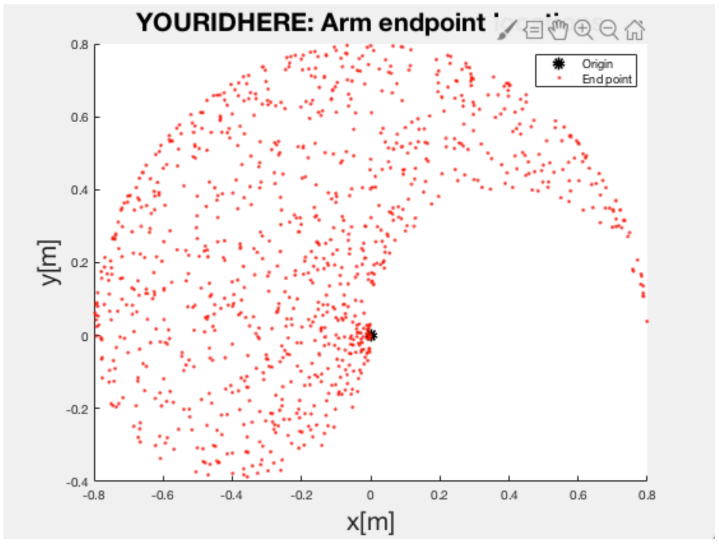

Figure 2. Endpoint locations (red dots) for 1000 data points when the arm angles are randomly distributed between 0 to $\pi$ radians. Origin of the arm (its shoulder joint) also shown as the black dot at (0,0).

To show the range of the endpoint when the arm angles move between 0 to $\pi$ radians.

- Set the arm lengths to 0.4m and the arm base origin to (0, 0).
- Generate two separate sets of 1000 uniformly distributed set of angle data points in the array theta over the range 0 to $\pi$ radians using the MATLAB rand command and run the RevoluteForwardKinematics2D function with the specified parameters to generate the corresponding endpoint locations. One set of data will later be used to train a neural network and the other to test it.
- Plot the endpoint positions and the arm origin position too.
- This should generate a plot like that shown in Fig. 2.
- Discuss the useful range of this arm.

**[3 marks]**

# 2. Implement 2-layer network

You are now required to build a multi-layer perceptron in MATLAB from first principles, with 2 layers, with a sigmoid hidden layer and linear output layer. It will be used to first learn and then implement the robot arm's inverse kinematics.

You should put all your code for training and testing the neural network into the script Main_ P2_TrainNeuralNetwork2022.m. This single script should run all the requested tasks and generate all the requested output figures. It is recommended you read-in the training data generated in the previous section using the MATLAB load function.

## 2.1. Implement 2-layer network training

- You will first need to implement the 2-layer network feedforward pass (with $n_h$ hidden units that can be specified as a parameter, and a linear output). Given the weight matrix, this will generate the network output activation for a given input vector.
- Then implement backpropagation to train all the weights.
- Make sure you initialize the weights in the network appropriately before running the iterative training.

**[20 marks]**

## 2.2. Train network inverse kinematics

Now you are required to train your network to perform the inverse kinematics of the planar arm using the 2-layer neural network you developed above. using the dataset, you generated in section 1.

- Train your linear output 2-layer network with the **augmented** end effect positions as input, and the two joint angles as outputs.
- Plot the error as training proceeds.

**[10 marks]**

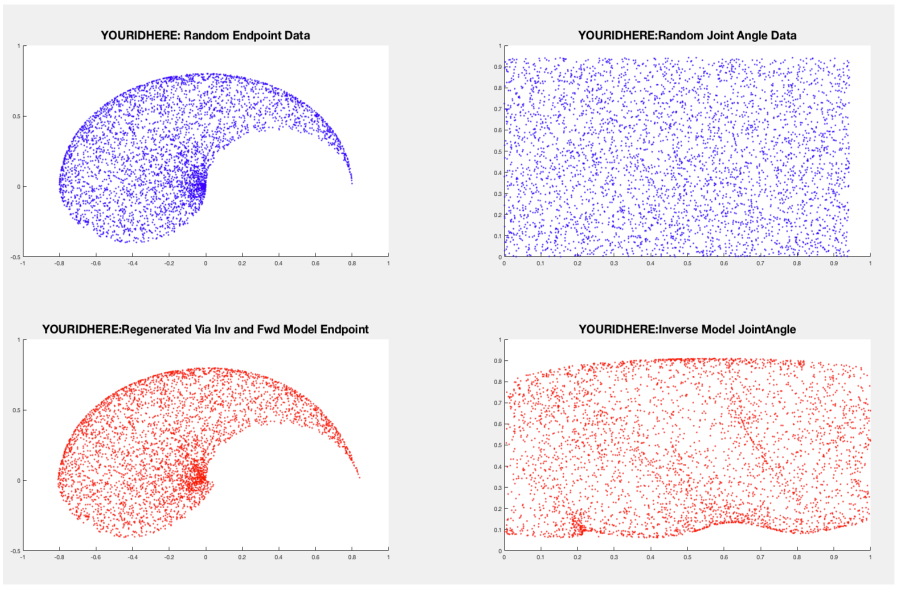

Figure 3. Training workspace of the endpoint of 2-link arm (left panel) is first mapped to joint angles using the 2-link arm inverse model (**here shown scaled between 0 and 1)**. These angles are then used with the forward model of the 2-link arm to show how well the inverse model operates. Ideally all points should map back to themselves.

## 2.3. Test and improve the inverse model

Test your inverse model looking at how the 2-link arm endpoint workspace is mapped back to itself. Do so as follows:

- Using your testing dataset run a forward network pass on the position data to generate predicted arm angles.
- Then run the forward kinematics function of these predicted angles and generate the 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. 3.
- Discuss the significance of these plots.
- Experiment with different numbers of hidden units, training times and learning rates.
- Also experiment with different training data sets.

**[13 marks]**

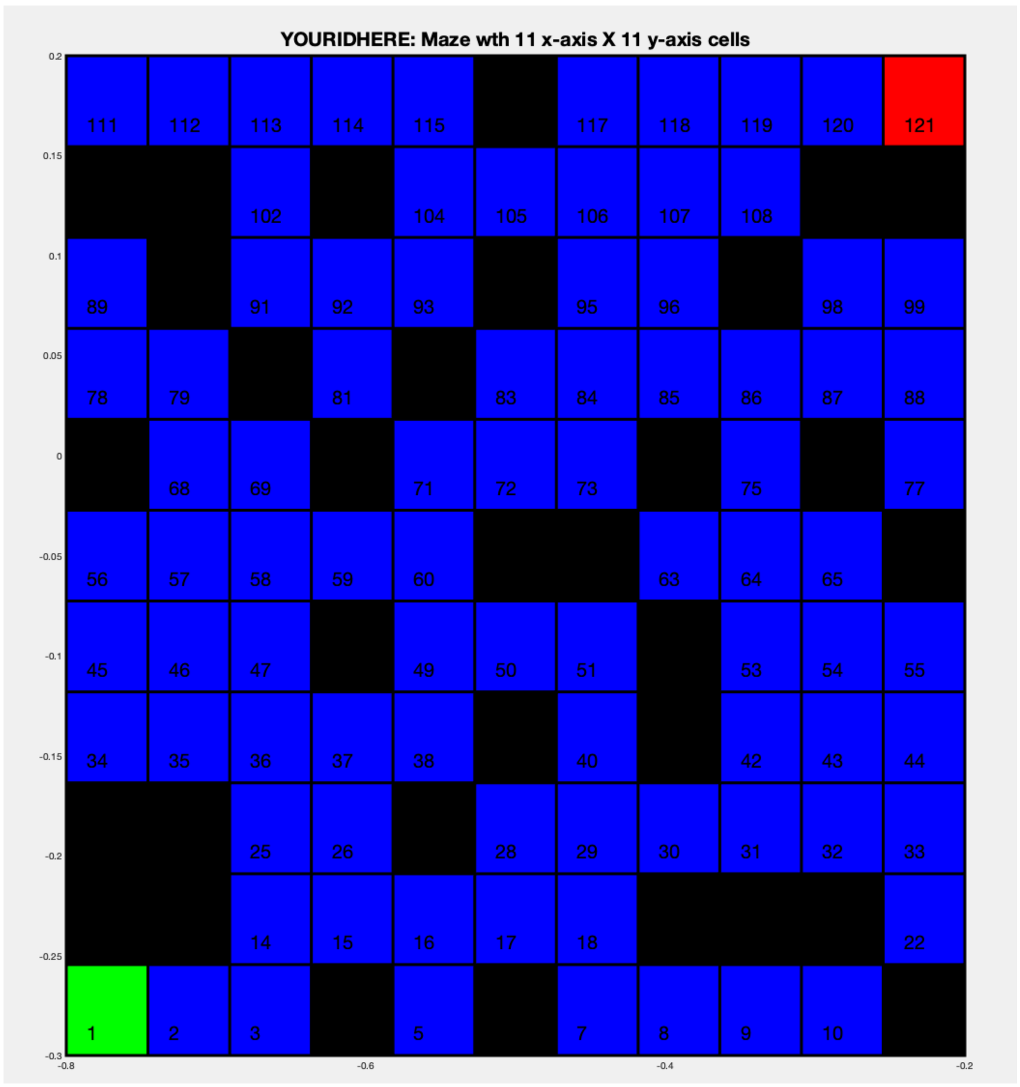

Figure 4. 11 x 11 grid maze with specified start state shown in green (no reward). The goal states shown in red and has a reward of 10. The blue states are allowed states with no reward associated with them. The black cells are blocked states and cannot be entered. An action towards a blocked or out of cell state results in the same state the current state

## 3. Path through a maze

You should put all your code for solving the maze using Q-learning into the MATLAB script Main_P3_RunGridworld2022.m. This script that provides the first steps you need in maze generation that will be used with your Q-Learning algorithm. Running this single script should run all the requested tasks and generate all the requested output figures.

Now you will tackle a movement planning problem. You are provided with a simple 2-dimentional grid representing a maze in Fig. 5. **You must use and implement this particular maze!** You are required to develop a Q-learning algorithm from first principles in MATLAB 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. 4.

- The states are numbered 1 to 121 as indicated.
- The goal of the Q-learning algorithm is finding the optimal path from any 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.

## 3.1. Random start state

- 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 looking something like the one shown in Fig 5.
- Comment on the histogram.

**[3 marks]**

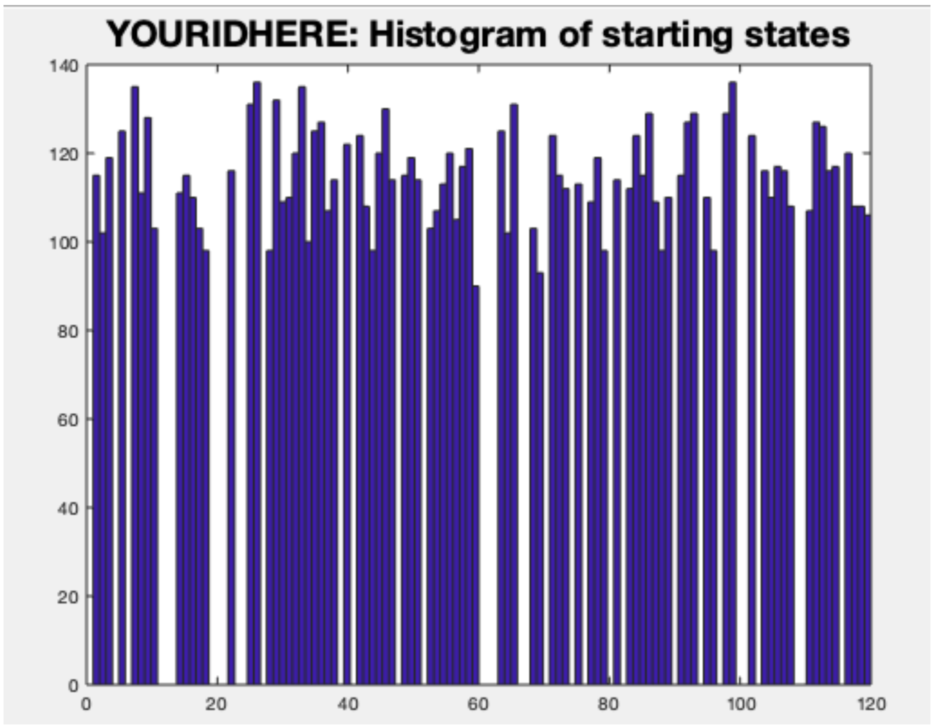

Figure 5. Histogram of 1000 randomly generated starting states, plotted with 121 bins corresponding to one of the maze's 121 states.

## 3.2. Build a reward function

- Specify the reward function for the maze.
- Explain how your reward function works.

**[3 marks]**

## 3.3. Generate the transition matrix

- Specify a transition matrix for the maze.
- You should generate it automatically after you represent the maze appropriately using a matrix.

**[3 marks]**

## 3.4. Initialize Q-values

- Initialize the Q-values matrix to suitable numbers.
- Explain your solution.

**[3 marks]**

## 3.5. Implement Q-learning algorithm

- Build the Q-table update function
- To implement Q-learning you will also need to
  - an outer loop to run 100 trials.
  - 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.
- Record the number of steps needed in an episode since it is indicative of how good Q-learning policy is becoming, and can be used as an indication of algorithm performance on the training data.

**[15 marks]**

## 3.6. Run Q-learning

- 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.
- Note you only need to run over 100 trials only to show the variance across complete trials – just a single trial is enough to actually solve the maze.
- 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. You should get a ploy like that shown in Fig. 6. Describe what you find.

**[8 marks]**

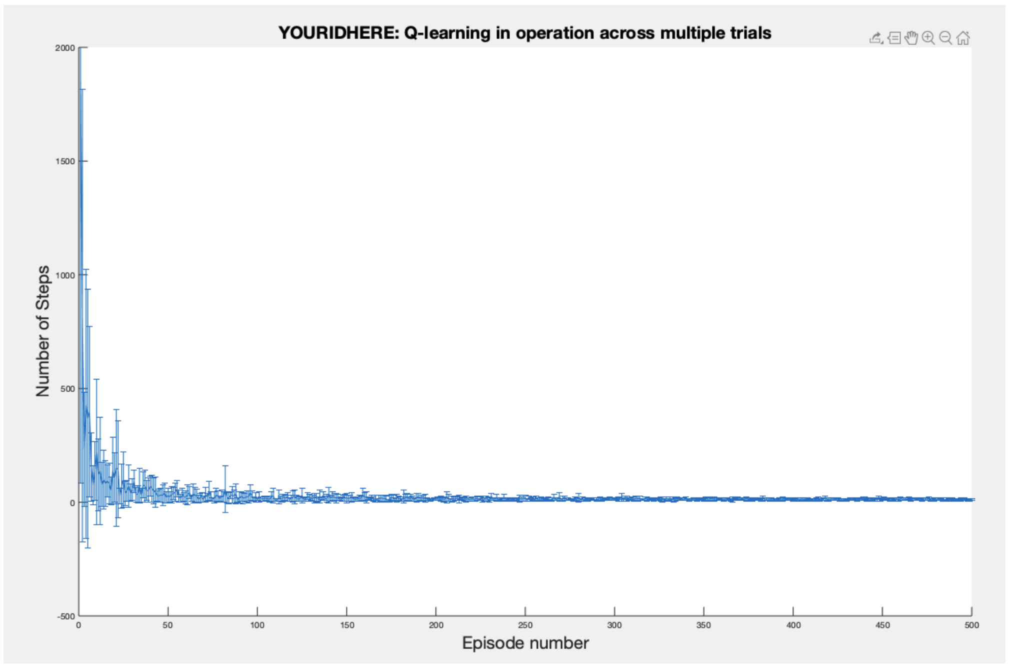

Figure 6. Typical example of mean and SD of steps plotted against episode.

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

- Record the Q-table at the end of a training trial.
- Write an exploitation function (or use your Є -greedy function with a zero-exploration probability Є) that makes use of the Q-values and performs 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. 7.

**[3 marks]**

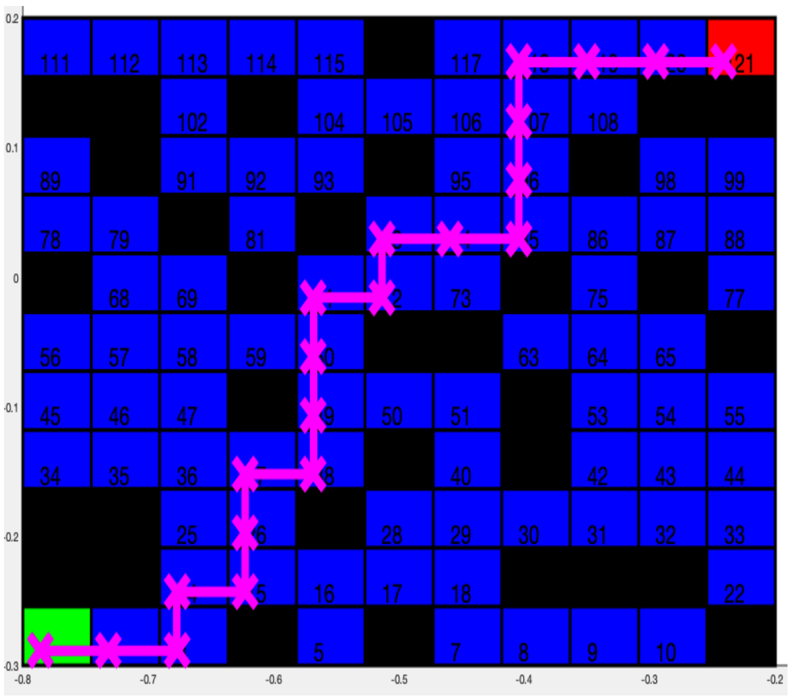

Figure 7. Path through the maze found by Q-learning algorithm. The path (shown in magenta starts) from the green cell at state 1 and finally reaches the red destination cell at state 121.

## 4. Move arm endpoint through maze

You should put all your code for kinematic arm control into the MATLAB script Main_ P4_MoveArmViaMaze2022.m This single script should run all the requested tasks and generate all the requested output figures and arm animation.

### 4.1. Generate kinematic control to revolute arm

- 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.
- Ensure you have scaled the maze appropriately so that it fits into the workspace of your revolute arm!

- Use the forward kinematic function with these angles as input to calculate the arm elbow and endpoint positions.
- Plot the endpoint trajectory of the arm on to the maze path you are following to demonstrate how well the inverse model works.
- Notice your MLP inverse model will not be perfect, but try to get it to work as well as you can.

**[10 marks]**

## 4.2. Animated revolute arm movement

- 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 or OneDrive in your report.

**[6 marks]**

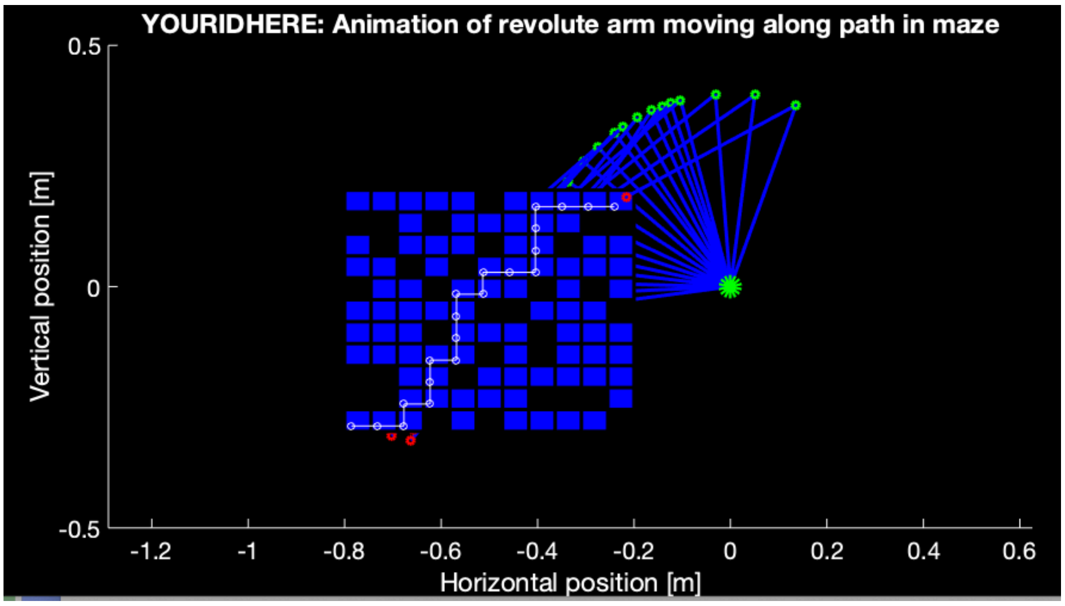

Figure 8. Screenshot of animation of 2-joint arm moving through path on maze. The base of the arm is represented by the large green dot and is located at the coordinates (0,0). The elbow joint is represented by the small green circle. The arm endpoint is given by the red dot and the arm links are shown in blue. Notice that the maze has been scaled to fit into workspace of the robotic arm.