AINT351 – MACHINE LEARNING

10555972 | Computer Science | 12th December 2019

Contents

[1. Training Data Generation 3](#_Toc25680144)

[1.1. Display Workspace of Revolute Arm 3](#_Toc25680145)

[1.2. Configurations of a Revolute Arm 3](#_Toc25680146)

[2. Implement a 2-Layer Network 3](#_Toc25680147)

[2.1. Implement the Network Feedforward Pass 3](#_Toc25680148)

[2.2. Implement 2-Layer Network Training 3](#_Toc25680149)

[2.3. Train Network Inverse Kinematics 3](#_Toc25680150)

[2.4. Test and Interpret Inverse Model 3](#_Toc25680151)

[3. Path Through a Maze Using Q-Learning 3](#_Toc25680152)

[3.1. Generate Random Start State 3](#_Toc25680153)

[3.2. Build a Reward Function 3](#_Toc25680154)

[3.3. Generate a Transition Matrix 3](#_Toc25680155)

[3.4. Initialize Q-Values 3](#_Toc25680156)

[3.5. Implement Q-Learning Algorithm 3](#_Toc25680157)

[3.6. Run Q-Learning 3](#_Toc25680158)

[3.7. Exploitation of Q-Values 4](#_Toc25680159)

[4. Move arm Endpoint Through Maze 4](#_Toc25680160)

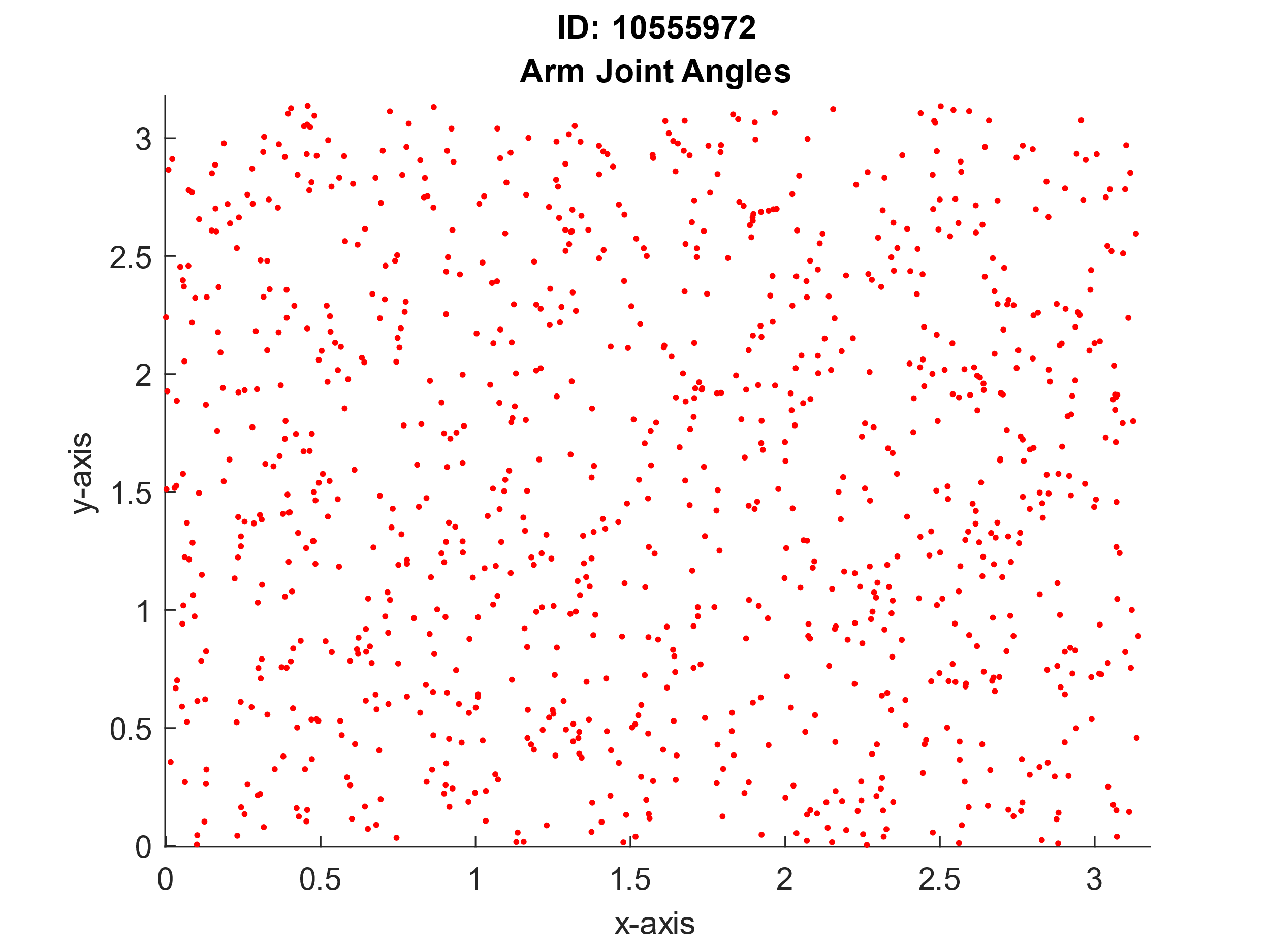
[4.1. Generate Kinematic Control to Revolute Arm 4](#_Toc25680161)

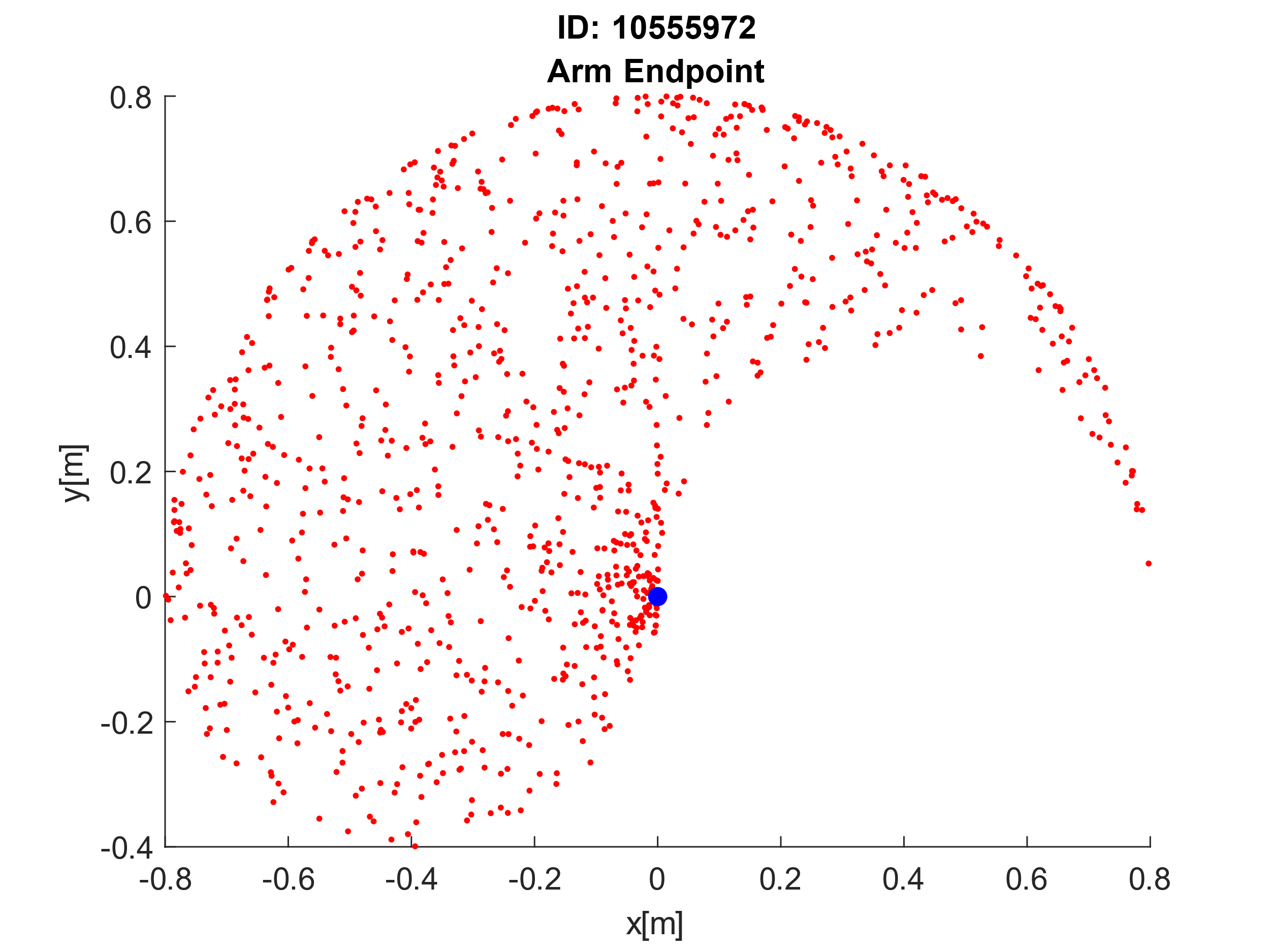
[4.2. Animated Revolute Arm Movement 4](#_Toc25680162)

# Training Data Generation

The provided Revolute Forward Kinematics 2D function is used to be able to output the arm end points by using the arms joint angles, the length of the arms and the base origin of the arms. The end points that are generated from this function will then later be used to train a neural network.

## Display Workspace of Revolute Arm

To display the workspace of the revolute arm I generated a random dataset between the values of 0 and π. The dataset had uniform distribution and contained 2x1000 samples. This dataset contains the angles that will be passed through the forward kinematics to calculate the end points and show the workspace of the arm. I set the parameters of the Revolute Forward Kinematics function to use, the arm lengths for before and after the elbow were set to 0.4 and the base origin coordinates was set to (0, 0). Passing in these values and the joint angles previously generated the function produces the correspdoning end points

****

% Defining variables

armLength = [0.4;0.4];

baseOrigin = [0, 0];

samples = 1000;

% Generating 2 x samples between 0 - pi

angles = pi \* rand(2,samples);

% Run angles through forward kinematics

[P1, P2] = RevoluteForwardKinematics2D(armLength, angles, baseOrigin);

% Plot randomly generated angles

figure

hold on

title({'ID: 10555972', 'Arm Joint Angles'});

xlabel('x-axis');

ylabel('y-axis');

plot(angles(1,:), angles(2,:), 'r.');

% Plot end points

figure

hold on

title({'ID: 10555972', 'Arm Endpoint'});

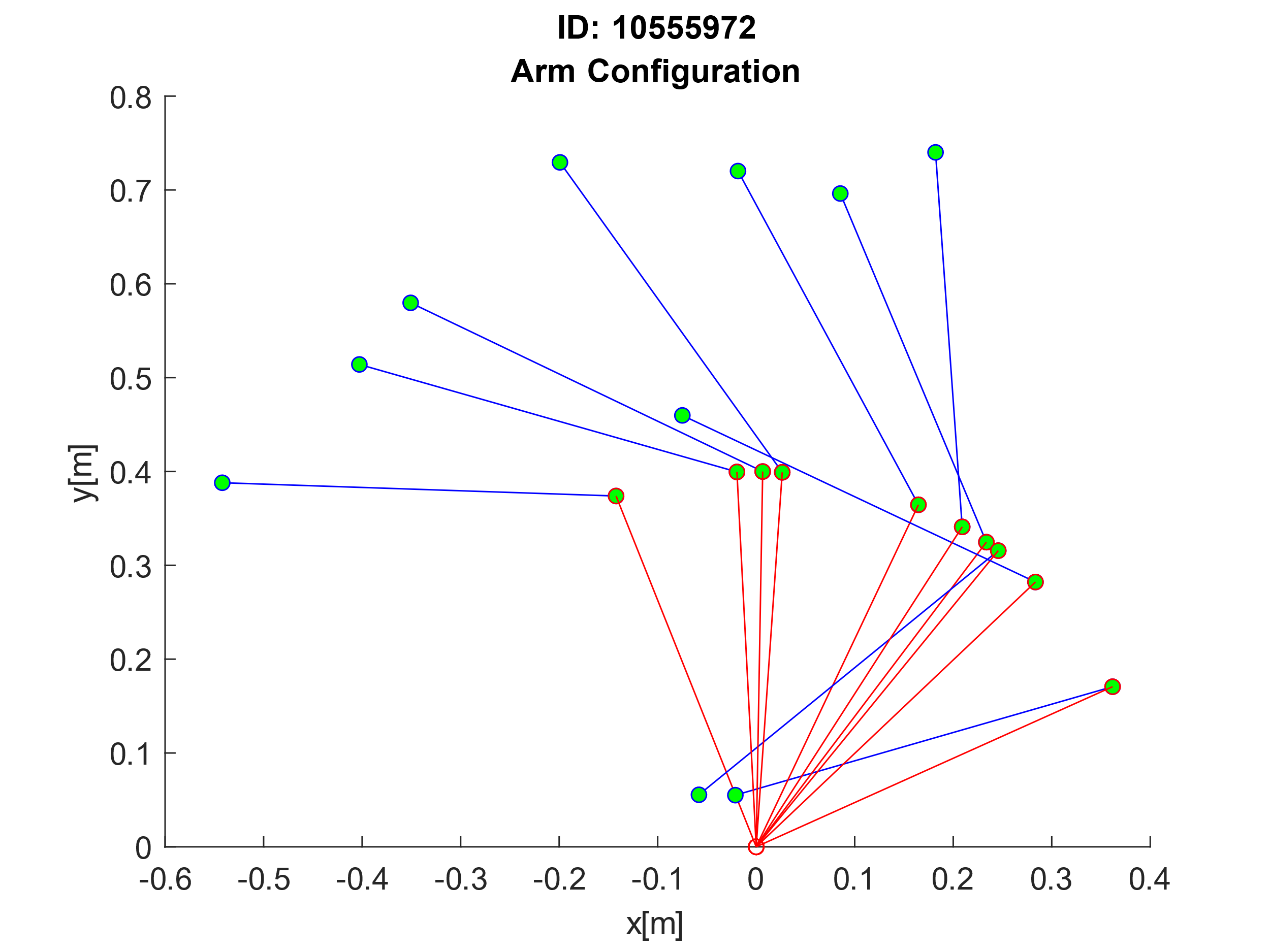
xlabel('x[m]');

ylabel('y[m]');

plot(P2(1,:), P2(2,:), 'r.')

plot(baseOrigin(1), baseOrigin(2), 'b.', 'MarkerSize', 20);

## Configurations of a Revolute Arm



% Plot 10 arm configurations

figure

title({'ID: 10555972', 'Arm Configuration'});

xlabel('x[m]');

ylabel('y[m]');

for i = 1:10

hold on

% Plotting from elbow to end of arm

plot([P1(1,i) P2(1,i)],[P1(2,i) P2(2,i)], 'b-o', 'MarkerSize', 5, 'MarkerFaceColor', 'green');

% Plotting from origin to elbow

plot([P1(1,i) baseOrigin(1)], [P1(2,i) baseOrigin(2)], 'r-o', 'MarkerSize', 5);

end

# Implement a 2-Layer Network

Draw network, 2 input nodes, 3 hidden nodes, 2 output nodes

Speak about scaling data & why you haven’t done it

## Implement the Network Feedforward Pass

## Implement 2-Layer Network Training

## Train Network Inverse Kinematics

## Test and Interpret Inverse Model

# Path Through a Maze Using Q-Learning

## Generate Random Start State

## Build a Reward Function

## Generate a Transition Matrix

## Initialize Q-Values

## Implement Q-Learning Algorithm

## Run Q-Learning

## Exploitation of Q-Values

# Move arm Endpoint Through Maze

## Generate Kinematic Control to Revolute Arm

## Animated Revolute Arm Movement