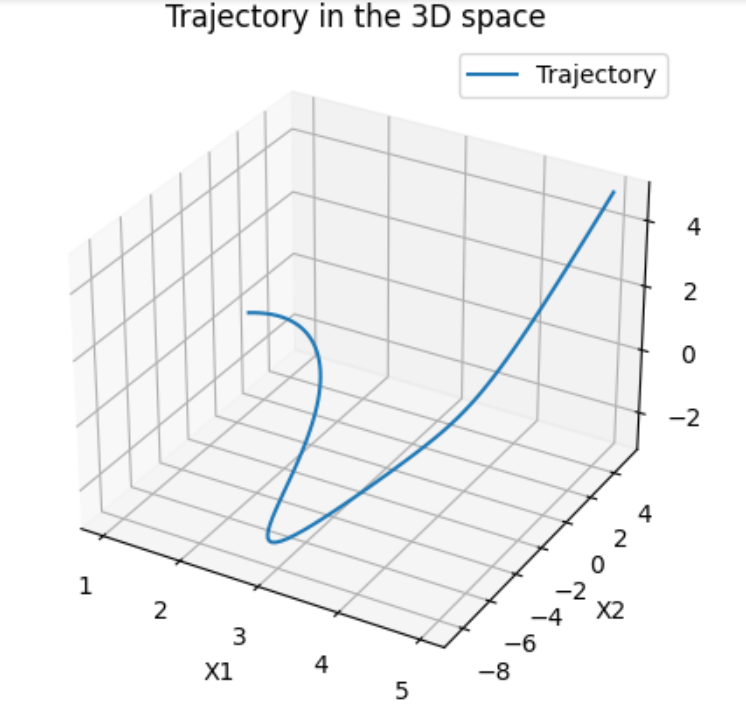
**Answer to the Question no. – 1(di)**

**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question1(di).ipynb*

|  |  |
| --- | --- |
| import numpy as np  import matplotlib.pyplot as plt  from sympy import symbols, lambdify  from mpl\_toolkits.mplot3d import Axes3D  # Variables for coefficients  coef1\_sym = symbols('coef1')  coef2\_sym = symbols('coef2')  coef3\_sym = symbols('coef3')  coef4\_sym = symbols('coef4')  coef5\_sym = symbols('coef5')  coef6\_sym = symbols('coef6')  coef7\_sym = symbols('coef7')  coef8\_sym = symbols('coef8')  # Basis matrix  t = 10  basis\_matrix = np.array([[1, 0, 0, 0, 0, 0, 0, 0],  [0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0],  [1, t, t\*\*2, t\*\*3, 0, 0, 0, 0],  [0, 1, 2 \* t, 3 \* t\*\*2, 0, 0, 0, 0],  [0, 0, 0, 0, 1, t, t\*\*2, t\*\*3],  [0, 0, 0, 0, 0, 1, 2 \* t, 3 \* t\*\*2]])  print(basis\_matrix)  # Matrix multiplication using pseudo-inverse  arr = np.array([[1],  [1],  [0],  [1],  [5],  [1],  [5],  [5]])  mul = np.linalg.pinv(basis\_matrix)  solutions = np.dot(mul, arr)  print(solutions)  # Coefficients as dictionary  coefficients\_dict = {  coef1\_sym: solutions[0, 0],  coef2\_sym: solutions[1, 0],  coef3\_sym: solutions[2, 0],  coef4\_sym: solutions[3, 0],  coef5\_sym: solutions[4, 0],  coef6\_sym: solutions[5, 0],  coef7\_sym: solutions[6, 0],  coef8\_sym: solutions[7, 0]  }  # alpha values as a list  alpha\_values = [1.0000000e+00, 1.0000000e+00, -1.8000000e-01, 1.2000000e-02,  -1.0185019e-13, 1.0000000e+00, -5.5000000e-01, 5.0000000e-02]  # Generate t values  T = np.linspace(0, 10, 100)  # Functions of z1\_dot & z2\_dot  def z1\_dot(T):  return alpha\_values[1] + 2 \* alpha\_values[2] \* T + 3 \* alpha\_values[3] \* (T \*\* 2)  def z2\_dot(T):  return alpha\_values[5] + 2 \* alpha\_values[6] \* T + 3 \* alpha\_values[7] \* (T \*\* 2)  # Value of X2  x1 = z1\_dot(T)  x3 = z2\_dot(T)  def x2(T):  return x3 / x1  X2 = x2(T)  # Values of X1 & X3  X1 = alpha\_values[0] + alpha\_values[1] \* T + alpha\_values[2] \* (T \*\* 2) + alpha\_values[3] \* (T \*\* 3)  X3 = alpha\_values[4] + alpha\_values[5] \* T + alpha\_values[6] \* (T \*\* 2) + alpha\_values[7] \* (T \*\* 3)  # 3D plot  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  ax.plot(X1, X2, X3, label='Trajectory')  ax.set\_xlabel('X1')  ax.set\_ylabel('X2')  ax.set\_zlabel('X3')  ax.set\_title('Trajectory in the 3D space')  ax.legend()  plt.show() |  |
|  |  |

**Output:**

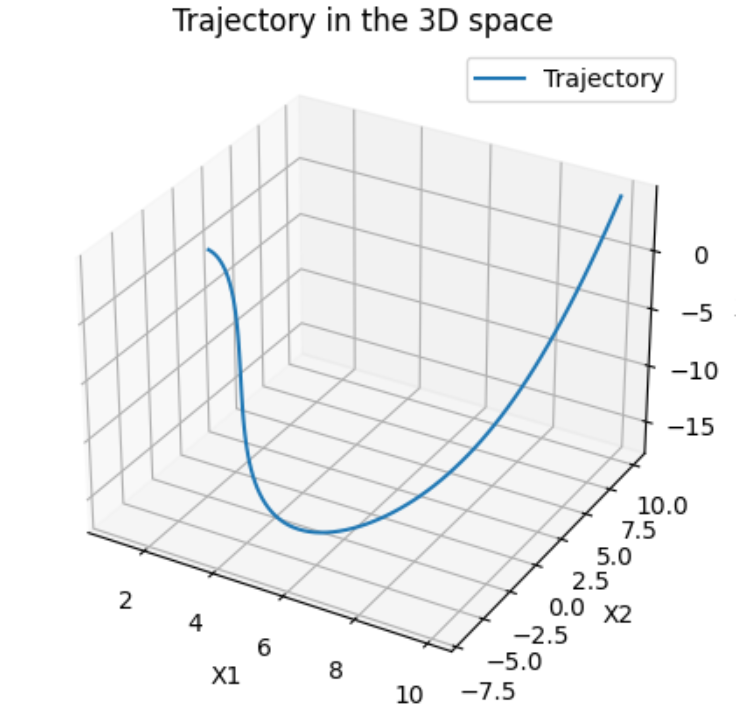
****

**Answer to the Question no. – 1(dii)**

**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question1(dii).ipynb*)

|  |  |
| --- | --- |
| import numpy as np  import matplotlib.pyplot as plt  from sympy import symbols, lambdify  from mpl\_toolkits.mplot3d import Axes3D  # Variables for coefficients  coef1\_sym = symbols('coef1')  coef2\_sym = symbols('coef2')  coef3\_sym = symbols('coef3')  coef4\_sym = symbols('coef4')  coef5\_sym = symbols('coef5')  coef6\_sym = symbols('coef6')  coef7\_sym = symbols('coef7')  coef8\_sym = symbols('coef8')  # Basis matrix  t = 15  basis\_matrix = np.array([[1, 0, 0, 0, 0, 0, 0, 0],  [0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0],  [1, t, t\*\*2, t\*\*3, 0, 0, 0, 0],  [0, 1, 2 \* t, 3 \* t\*\*2, 0, 0, 0, 0],  [0, 0, 0, 0, 1, t, t\*\*2, t\*\*3],  [0, 0, 0, 0, 0, 1, 2 \* t, 3 \* t\*\*2]])  print(basis\_matrix)  # Matrix multiplication using pseudo-inverse  arr = np.array([[1],  [1],  [0],  [1],  [10],  [1],  [5],  [10]])  mul = np.linalg.pinv(basis\_matrix)  solutions = np.dot(mul, arr)  print(solutions)  # Coefficients as dictionary  coefficients\_dict = {  coef1\_sym: solutions[0, 0],  coef2\_sym: solutions[1, 0],  coef3\_sym: solutions[2, 0],  coef4\_sym: solutions[3, 0],  coef5\_sym: solutions[4, 0],  coef6\_sym: solutions[5, 0],  coef7\_sym: solutions[6, 0],  coef8\_sym: solutions[7, 0]  }  # alpha values as a list  alpha\_values = [1.00000000e+00, 1.00000000e+00, -8.00000000e-02, 3.55555556e-03,  4.14285794e-13, 1.00000000e+00, -7.33333333e-01, 4.59259259e-02]  # Generate t values  T = np.linspace(0, 15, 100)  # Functions of z1\_dot & z2\_dot  def z1\_dot(T):  return alpha\_values[1] + 2 \* alpha\_values[2] \* T + 3 \* alpha\_values[3] \* (T \*\* 2)  def z2\_dot(T):  return alpha\_values[5] + 2 \* alpha\_values[6] \* T + 3 \* alpha\_values[7] \* (T \*\* 2)  # Value of X2  x1 = z1\_dot(T)  x3 = z2\_dot(T)  def x2(T):  return x3 / x1  X2 = x2(T)  # Values of X1 & X3  X1 = alpha\_values[0] + alpha\_values[1] \* T + alpha\_values[2] \* (T \*\* 2) + alpha\_values[3] \* (T \*\* 3)  X3 = alpha\_values[4] + alpha\_values[5] \* T + alpha\_values[6] \* (T \*\* 2) + alpha\_values[7] \* (T \*\* 3)  # 3D plot  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  ax.plot(X1, X2, X3, label='Trajectory')  ax.set\_xlabel('X1')  ax.set\_ylabel('X2')  ax.set\_zlabel('X3')  ax.set\_title('Trajectory in the 3D space')  ax.legend()  plt.show() |  |

**Output:**

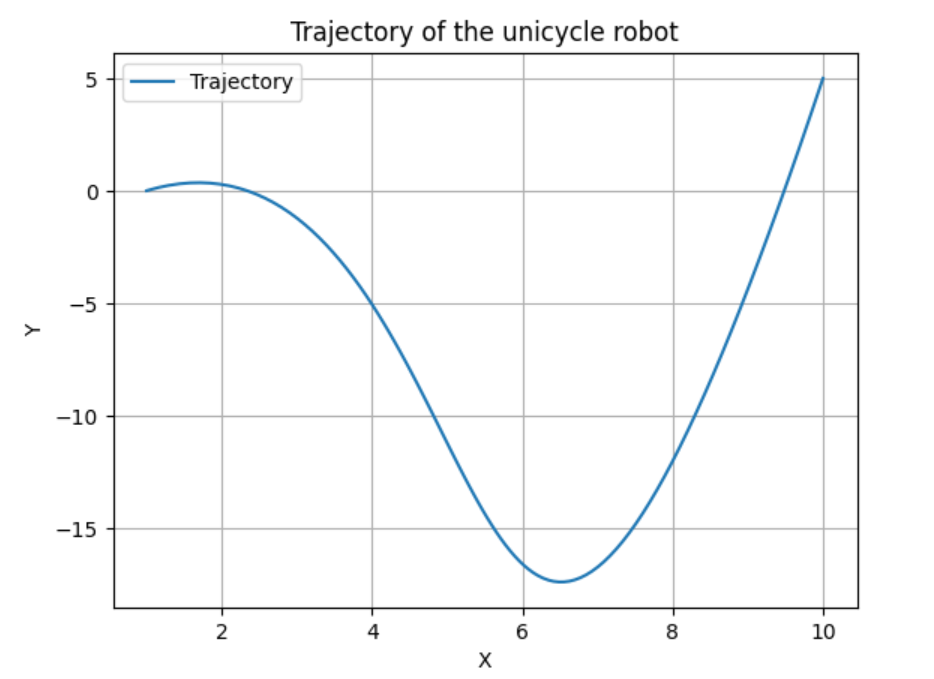
****

**Answer to the Question no. – 2(b)**

**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question2(b).ipynb*

|  |  |
| --- | --- |
| import numpy as np  import matplotlib.pyplot as plt  from sympy import symbols, lambdify  # Variables for coefficients  coef1\_sym = symbols('coef1')  coef2\_sym = symbols('coef2')  coef3\_sym = symbols('coef3')  coef4\_sym = symbols('coef4')  coef5\_sym = symbols('coef5')  coef6\_sym = symbols('coef6')  coef7\_sym = symbols('coef7')  coef8\_sym = symbols('coef8')  # Basis matrix  t = 15  basis\_matrix = np.array([[1, 0, 0, 0, 0, 0, 0, 0],  [0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0],  [1, t, t\*\*2, t\*\*3, 0, 0, 0, 0],  [0, 1, 2 \* t, 3 \* t\*\*2, 0, 0, 0, 0],  [0, 0, 0, 0, 1, t, t\*\*2, t\*\*3],  [0, 0, 0, 0, 0, 1, 2 \* t, 3 \* t\*\*2]])  print(basis\_matrix)  # Matrix multiplication using pseudo-inverse  arr = np.array([[0],  [0],  [0.5],  [-1.6],  [5],  [5],  [0.5],  [-1.6]])  mul = np.linalg.pinv(basis\_matrix)  solutions = np.dot(mul, arr)  print(solutions)  # Coefficients as dictionary  coefficients\_dict = {  coef1\_sym: solutions[0, 0],  coef2\_sym: solutions[1, 0],  coef3\_sym: solutions[2, 0],  coef4\_sym: solutions[3, 0],  coef5\_sym: solutions[4, 0],  coef6\_sym: solutions[5, 0],  coef7\_sym: solutions[6, 0],  coef8\_sym: solutions[7, 0]  }  # alpha values as a list  alpha\_values = [1.00000000e+00, 1.00000000e+00, -8.00000000e-02, 3.55555556e-03,  4.14285794e-13, 1.00000000e+00, -7.33333333e-01, 4.59259259e-02]  # Generate T values  T = np.linspace(0, 15, 100)  # Values of X1 & X3  X1 = alpha\_values[0] + alpha\_values[1] \* T + alpha\_values[2] \* (T \*\* 2) + alpha\_values[3] \* (T \*\* 3)  X3 = alpha\_values[4] + alpha\_values[5] \* T + alpha\_values[6] \* (T \*\* 2) + alpha\_values[7] \* (T \*\* 3)  # plot  plt.figure()  plt.plot(X1, X3, label='Trajectory')  plt.xlabel('X')  plt.ylabel('Y')  plt.title('Trajectory of the unicycle robot')  plt.legend()  plt.grid(True)  plt.show() |  |

**Output:**

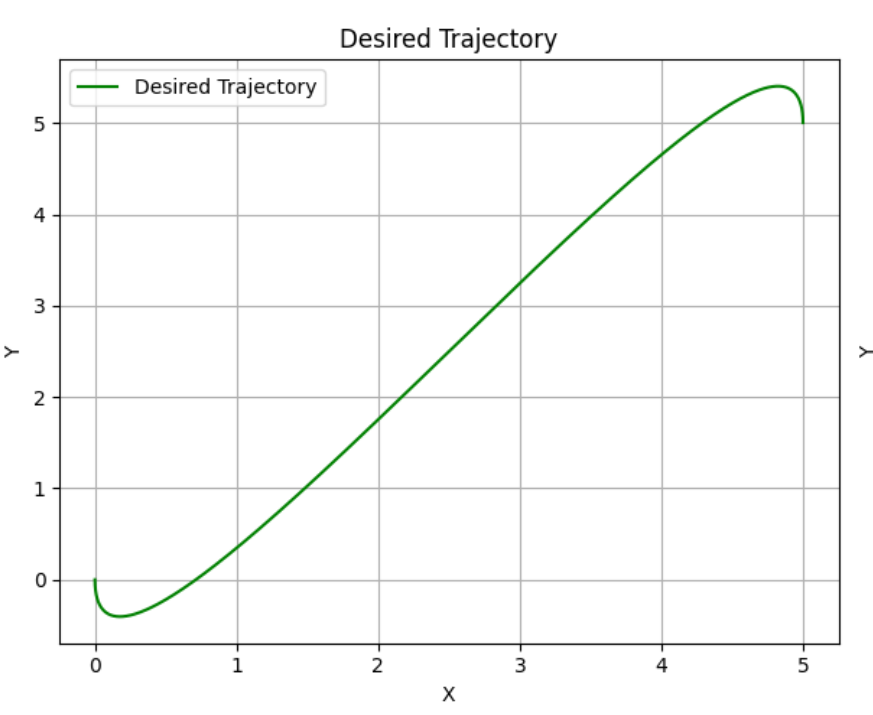
****

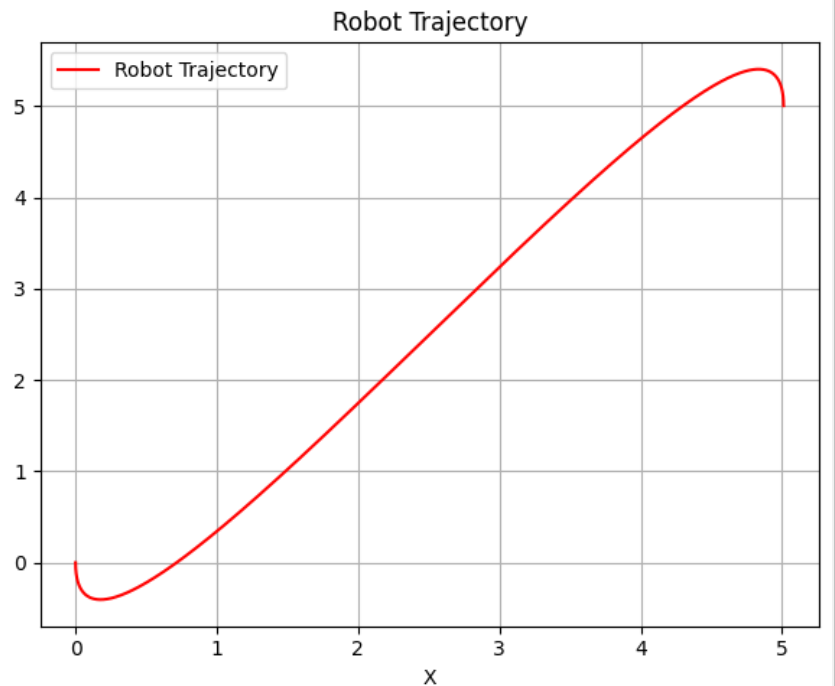
**Answer to the Question no. – 2(c)**

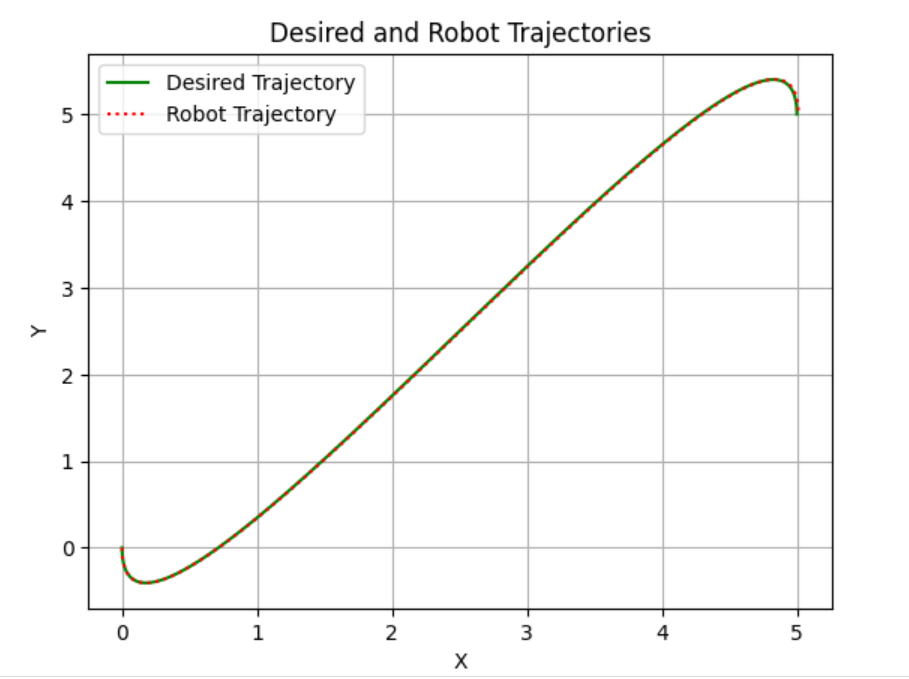
**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question2(c).ipynb*

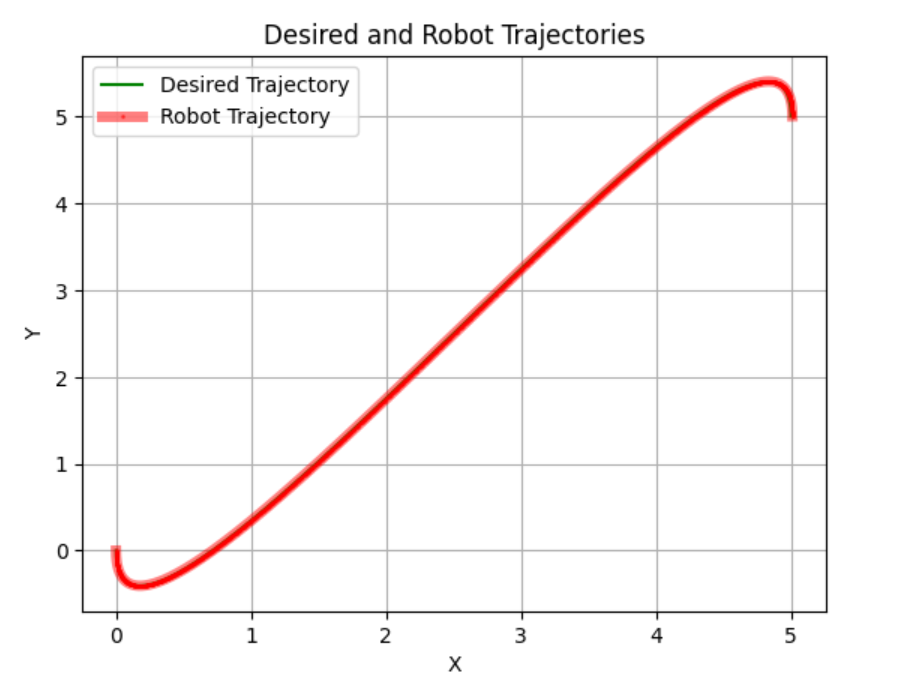
|  |  |
| --- | --- |
|  |  |
| import numpy as np  import matplotlib.pyplot as plt  # Time array  t = np.arange(0, 15, 0.01)  len(t)  # Final time T  T = 15  Tsq = np.power(T, 2)  Tcb = np.power(T, 3)  # Initialize matrix A  A = np.array([  [1, 0, 0, 0, 0, 0, 0, 0],  [0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0],  [1, T, Tsq, Tcb, 0, 0, 0, 0],  [0, 1, 2\*T, 3\*Tsq, 0, 0, 0, 0],  [0, 0, 0, 0, 1, T, Tsq, Tcb],  [0, 0, 0, 0, 0, 1, 2\*T, 3\*Tsq]  ])  # Initialize vector b with initial and final conditions for position and velocity  b = np.array([  [0], # Initial position X  [0], # Initial velocity X  [0], # Initial position Y  [-0.5], # Initial velocity Y  [5], # Final position X  [0], # Final velocity X  [5], # Final position Y  [-0.5] # Final velocity Y  ])  # Calculate the pseudo-inverse of matrix A  A\_inv = np.linalg.pinv(A)  # Calculate polynomial coefficients x = A\_inv \* b  x = np.matmul(A\_inv, b)  # Extract polynomial coefficients  a11, a12, a13, a14 = x[0], x[1], x[2], x[3]  a21, a22, a23, a24 = x[4], x[5], x[6], x[7]  # Calculate the desired trajectory for X and Y coordinates  X\_new = a11 + a12 \* t + a13 \* np.power(t, 2) + a14 \* np.power(t, 3)  Y\_new = a21 + a22 \* t + a23 \* np.power(t, 2) + a24 \* np.power(t, 3)  # Calculate the second derivatives  Xdd = np.gradient(np.gradient(X\_new, t), t)  Ydd = np.gradient(np.gradient(Y\_new, t), t)  # Calculate the angle theta  theta = np.arctan2(np.gradient(Y\_new, t), np.gradient(X\_new, t))  # Calculate the speed  V = np.sqrt(np.gradient(X\_new, t)\*\*2 + np.gradient(Y\_new, t)\*\*2)  # Calculate the acceleration and angular velocity  a = np.cos(theta) \* Xdd + np.sin(theta) \* Ydd  omega = (-np.sin(theta) \* Xdd + np.cos(theta) \* Ydd) / V  # Initialize final states  x\_final = X\_new[0]  y\_final = Y\_new[0]  theta\_final = theta[0]  V\_final = V[0]  # Initialize lists to hold robot's states  x\_states = [x\_final]  y\_states = [y\_final]  # Calculate robot trajectory  for i in range(1, len(t)):  dt = t[i] - t[i - 1] # Calculate time step    # Update final states  x\_final += V\_final \* np.cos(theta\_final) \* dt  y\_final += V\_final \* np.sin(theta\_final) \* dt  theta\_final += omega[i] \* dt  V\_final += a[i] \* dt    # Append updated states to the lists  x\_states.append(x\_final)  y\_states.append(y\_final)  # Visualize the desired trajectory and robot trajectory  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))  # Desired trajectory  ax1.plot(X\_new, Y\_new, label='Desired Trajectory', color='green')  ax1.set\_xlabel('X')  ax1.set\_ylabel('Y')  ax1.legend()  ax1.set\_title('Desired Trajectory')  ax1.grid(True)  # Robot trajectory  ax2.plot(x\_states, y\_states, label='Robot Trajectory', color='red')  ax2.set\_xlabel('X')  ax2.set\_ylabel('Y')  ax2.legend()  ax2.set\_title('Robot Trajectory')  ax2.grid(True)  # plots  plt.tight\_layout()  plt.show()  # Plot the desired and robot trajectories  plt.figure()  plt.plot(X\_new, Y\_new, label='Desired Trajectory', color='green')  plt.plot(x\_states, y\_states, label='Robot Trajectory', linestyle='dotted', color='red')  plt.xlabel('X')  plt.ylabel('Y')  plt.title('Desired and Robot Trajectories')  plt.legend()  plt.grid(True)  plt.show()  # Plot the desired and robot trajectories with additional properties  plt.figure()  plt.plot(X\_new, Y\_new, label='Desired Trajectory', color='green')  plt.plot(x\_states, y\_states, label='Robot Trajectory', linestyle='-',  linewidth=5, color='red', alpha=0.5, marker='o', markersize=1,  markeredgecolor='red')  plt.xlabel('X')  plt.ylabel('Y')  plt.legend()  plt.title('Desired and Robot Trajectories')  plt.grid(True)  plt.show() |  |

**Output:**

****

****

****

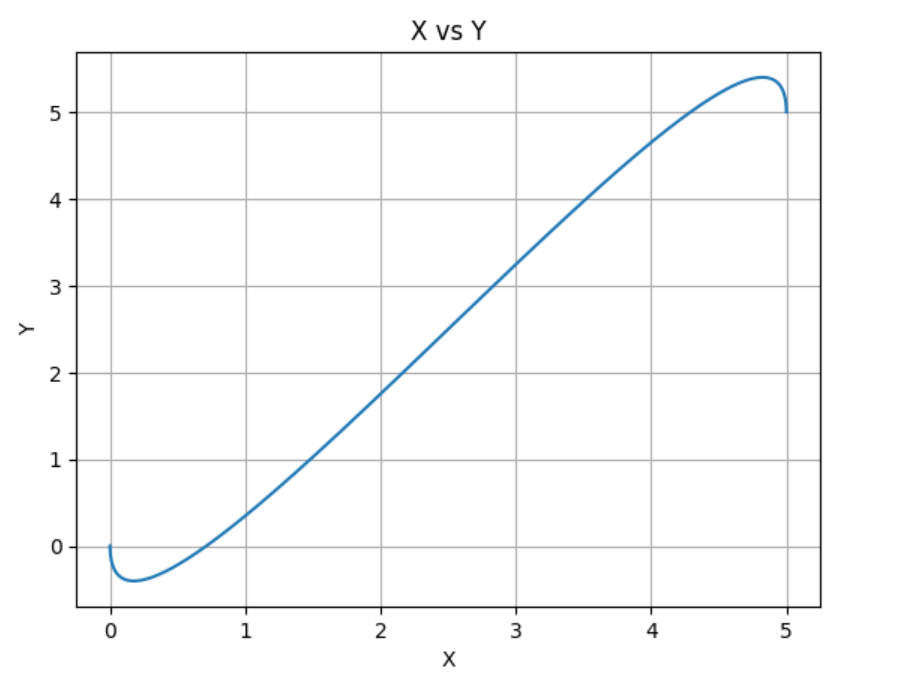
****

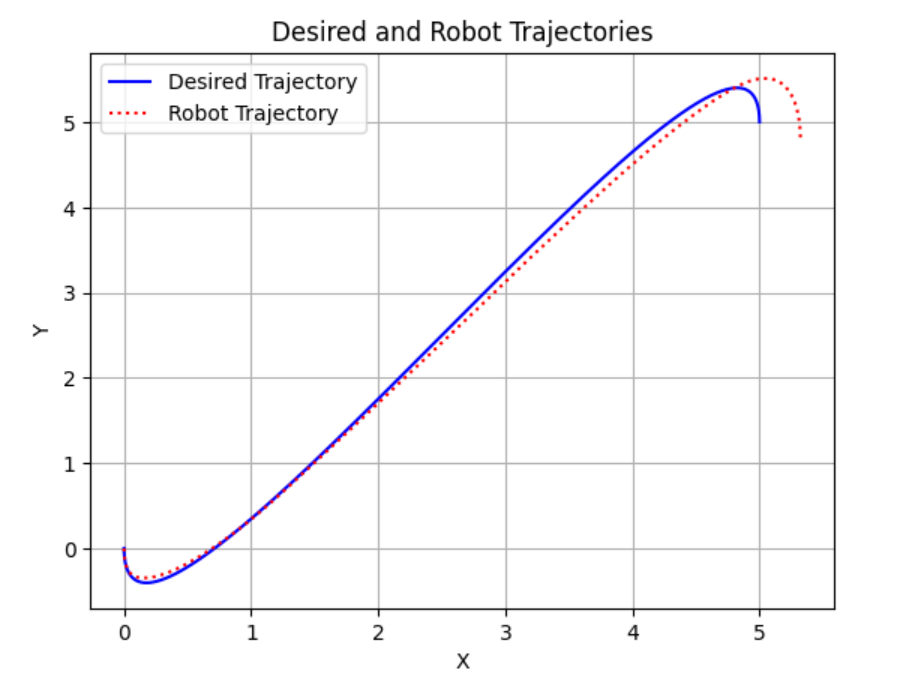
**Answer to the Question no. – 3**

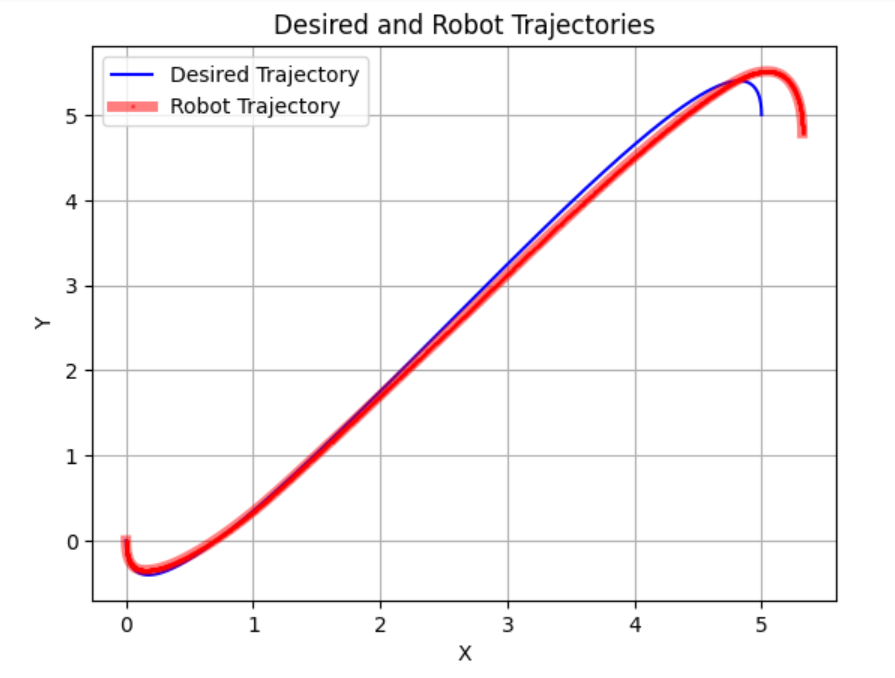
**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question3.ipynb*

|  |  |
| --- | --- |
| import numpy as np  import matplotlib.pyplot as plt  # Time variable  t = np.arange(0, 15, 0.01)  # Desired trajectory  T = 15  Tsq = T\*\*2  Tcb = T\*\*3  # Matrix A for calculating trajectory coefficients  A = np.array([  [1, 0, 0, 0, 0, 0, 0, 0],  [0, 1, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0],  [0, 0, 0, 0, 0, 1, 0, 0],  [1, T, Tsq, Tcb, 0, 0, 0, 0],  [0, 1, 2\*T, 3\*Tsq, 0, 0, 0, 0],  [0, 0, 0, 0, 1, T, Tsq, Tcb],  [0, 0, 0, 0, 0, 1, 2\*T, 3\*Tsq]  ])  # Vector b for calculating trajectory coefficients  b = np.array([  [0],  [0],  [0],  [-0.5],  [5],  [0],  [5],  [-0.5]  ])  # Calculate pseudo inverse of A  A\_inv = np.linalg.pinv(A)  # Calculate x  x = np.matmul(A\_inv, b)  # Extract coefficients from x  a11, a12, a13, a14 = x[:4]  a21, a22, a23, a24 = x[4:8]  # Calculate trajectories X and Y  X = a11 + a12 \* t + a13 \* t\*\*2 + a14 \* t\*\*3  Y = a21 + a22 \* t + a23 \* t\*\*2 + a24 \* t\*\*3  # Calculate gradients and other derived values  dX = np.gradient(X, t)  dY = np.gradient(Y, t)  theta = np.arctan2(dY, dX)  V = np.sqrt(dX\*\*2 + dY\*\*2)  a = np.cos(theta) \* np.gradient(dX, t) + np.sin(theta) \* np.gradient(dY, t)  omega = (-np.sin(theta) \* np.gradient(dX, t) + np.cos(theta) \* np.gradient(dY, t)) / V  # Noise levels for velocity and angle  noise\_std\_v = 0.01  noise\_std\_theta = 0.001  # Generate noise  noise\_v = np.random.normal(0, noise\_std\_v, len(t))  noise\_theta = np.random.normal(0, noise\_std\_theta, len(t))  # Initialize state variables  x\_final = X[0]  y\_final = Y[0]  theta\_final = theta[0]  V\_final = V[0]  # Lists to store robot trajectory states  x\_states = [x\_final]  y\_states = [y\_final]  # Calculate robot trajectory  for i in range(1, len(t)):  dt = t[i] - t[i - 1]  x\_final += V\_final \* np.cos(theta\_final) \* dt  y\_final += V\_final \* np.sin(theta\_final) \* dt  theta\_final += omega[i] \* dt + noise\_theta[i]  V\_final += a[i] \* dt + noise\_v[i]  x\_states.append(x\_final)  y\_states.append(y\_final)  # Plot X vs Y  plt.figure()  plt.plot(X, Y)  plt.title('X vs Y')  plt.xlabel('X')  plt.ylabel('Y')  plt.grid(True)  # Plot desired and robot trajectories  plt.figure()  plt.plot(X, Y, label='Desired Trajectory', color='blue')  plt.plot(x\_states, y\_states, label='Robot Trajectory', linestyle='dotted', color='red')  plt.xlabel('X')  plt.ylabel('Y')  plt.legend()  plt.title('Desired and Robot Trajectories')  plt.grid(True)  plt.show()  # Plot desired and robot trajectories with additional properties  plt.figure()  plt.plot(X, Y, label='Desired Trajectory', color='blue')  plt.plot(x\_states, y\_states, label='Robot Trajectory', linestyle='-', linewidth=5, color='red', alpha=0.5, marker='o', markersize=1, markeredgecolor='red')  plt.xlabel('X')  plt.ylabel('Y')  plt.legend()  plt.title('Desired and Robot Trajectories')  plt.grid(True)  plt.show() |  |

**Output:**

****

****

****