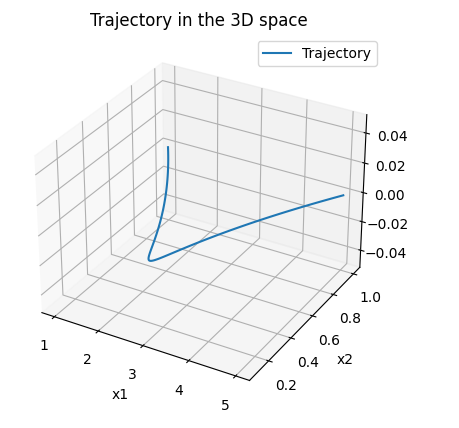
**Answer to the Question no. – 1(di)**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | import numpy as np  import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D  # Define the time range  t = np.arange(0, 10.1, 0.1)  # Define the basis functions  psi1 = np.ones\_like(t)  psi2 = t  psi3 = t\*\*2  psi4 = t\*\*3  # Initial conditions  x1\_initial = 1  x2\_initial = 1  x3\_initial = 0  x1\_dot\_initial = 1  # Final conditions  x1\_final = 5  x2\_final = 5  x3\_final = 5  x1\_dot\_final = 1  # Create the matrix A using the basis functions at ti=0 and tf=10  A = np.array([  [psi1[0], psi2[0], psi3[0], psi4[0]],  [psi1[-1], psi2[-1], psi3[-1], psi4[-1]],  [0, 1, 2 \* psi2[0], 3 \* psi3[0]],  [0, 1, 2 \* psi2[-1], 3 \* psi3[-1]],  ])  # Define the vector b for initial and final conditions  b = np.array([  x1\_initial,  x1\_final,  x1\_dot\_initial,  x1\_dot\_final,  ])  # Solve for the coefficients alpha  alpha = np.linalg.solve(A, b)  # Evaluate the trajectory and derivatives  x1 = alpha[0] \* psi1 + alpha[1] \* psi2 + alpha[2] \* psi3 + alpha[3] \* psi4  x1\_dot = alpha[1] + 2 \* alpha[2] \* t + 3 \* alpha[3] \* t\*\*2  # Plot the trajectory  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  ax.plot(x1, x1\_dot, label='Trajectory')  # Add labels and title  ax.set\_xlabel('x1')  ax.set\_ylabel('x2')  ax.set\_title('Trajectory in the 3D space')  # Add legend and show the plot  ax.legend()  plt.show() |
|  |  |

**Output:**

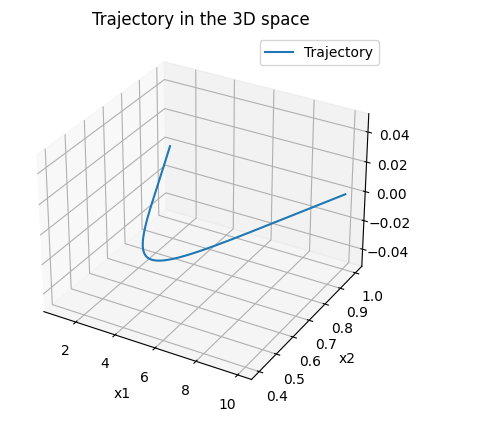
****

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

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | import numpy as np  import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D  # Define the time range  t = np.arange(0, 15.1, 0.1)  # Define the basis functions  psi1 = np.ones\_like(t)  psi2 = t  psi3 = t\*\*2  psi4 = t\*\*3  # Initial conditions  x1\_initial = 1  x2\_initial = 1  x3\_initial = 0  x1\_dot\_initial = 1  # Final conditions  x1\_final = 10  x2\_final = 10  x3\_final = 5  x1\_dot\_final = 1  # Create the matrix A using the basis functions at ti=0 and tf=15  A = np.array([  [psi1[0], psi2[0], psi3[0], psi4[0]],  [psi1[-1], psi2[-1], psi3[-1], psi4[-1]],  [0, 1, 2 \* psi2[0], 3 \* psi3[0]],  [0, 1, 2 \* psi2[-1], 3 \* psi3[-1]],  ])  # Define the vector b for initial and final conditions  b = np.array([  x1\_initial,  x1\_final,  x1\_dot\_initial,  x1\_dot\_final,  ])  # Solve for the coefficients alpha  alpha = np.linalg.solve(A, b)  # Evaluate the trajectory and derivatives  x1 = alpha[0] \* psi1 + alpha[1] \* psi2 + alpha[2] \* psi3 + alpha[3] \* psi4  x1\_dot = alpha[1] + 2 \* alpha[2] \* t + 3 \* alpha[3] \* t\*\*2  # Plot the trajectory  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  ax.plot(x1, x1\_dot, label='Trajectory')  # Add labels and title  ax.set\_xlabel('x1')  ax.set\_ylabel('x2')  ax.set\_title('Trajectory in the 3D space')  # Add legend and show the plot  ax.legend()  plt.show() |

**Output:**

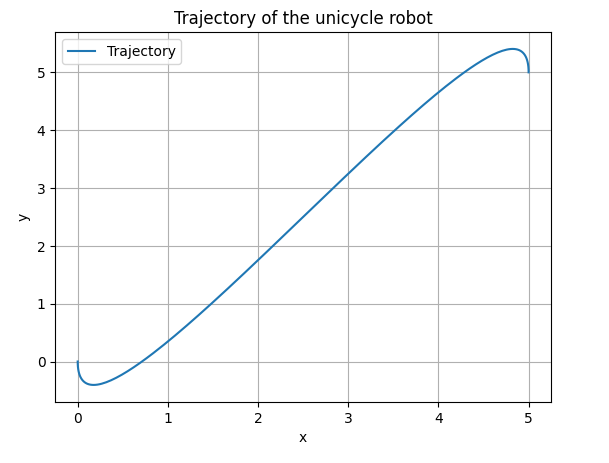
****

**Answer to the Question no. – 2(b)**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | **import** numpy **as** np  **import** matplotlib.pyplot **as** plt  **from** mpl\_toolkits.mplot3d **import** Axes3D  # Define the time range  t = np.arange(0, 15.1, 0.1)  # Define the basis functions  psi1 = np.ones\_like(t)  psi2 = t  psi3 = t\*\*2  psi4 = t\*\*3  # Initial conditions  x\_initial = 0  y\_initial = 0  v\_initial = 0.5  theta\_initial = -np.pi/2  # Final conditions  x\_final = 5  y\_final = 5  v\_final = 0.5  theta\_final = -np.pi/2  # Create the matrix A using the basis functions at ti=0 and tf=15  A = np.array([  [psi1[0], psi2[0], psi3[0], psi4[0]],  [psi1[-1], psi2[-1], psi3[-1], psi4[-1]],  [0, 1, 2 \* psi2[0], 3 \* psi3[0]],  [0, 1, 2 \* psi2[-1], 3 \* psi3[-1]],  ])  # Define the vector b for initial and final conditions  b\_x = np.array([x\_initial, x\_final, v\_initial \* np.cos(theta\_initial), v\_final \* np.cos(theta\_final)])  b\_y = np.array([y\_initial, y\_final, v\_initial \* np.sin(theta\_initial), v\_final \* np.sin(theta\_final)])  # Solve for the coefficients alpha  alpha\_x = np.linalg.solve(A, b\_x)  alpha\_y = np.linalg.solve(A, b\_y)  # Evaluate the trajectory  x = alpha\_x[0] \* psi1 + alpha\_x[1] \* psi2 + alpha\_x[2] \* psi3 + alpha\_x[3] \* psi4  y = alpha\_y[0] \* psi1 + alpha\_y[1] \* psi2 + alpha\_y[2] \* psi3 + alpha\_y[3] \* psi4  # Plot the trajectory  plt.figure()  plt.plot(x, y, label='Trajectory')  plt.xlabel('x')  plt.ylabel('y')  plt.title('Trajectory of the unicycle robot')  plt.legend()  plt.grid(True)  plt.show() |

**Output:**

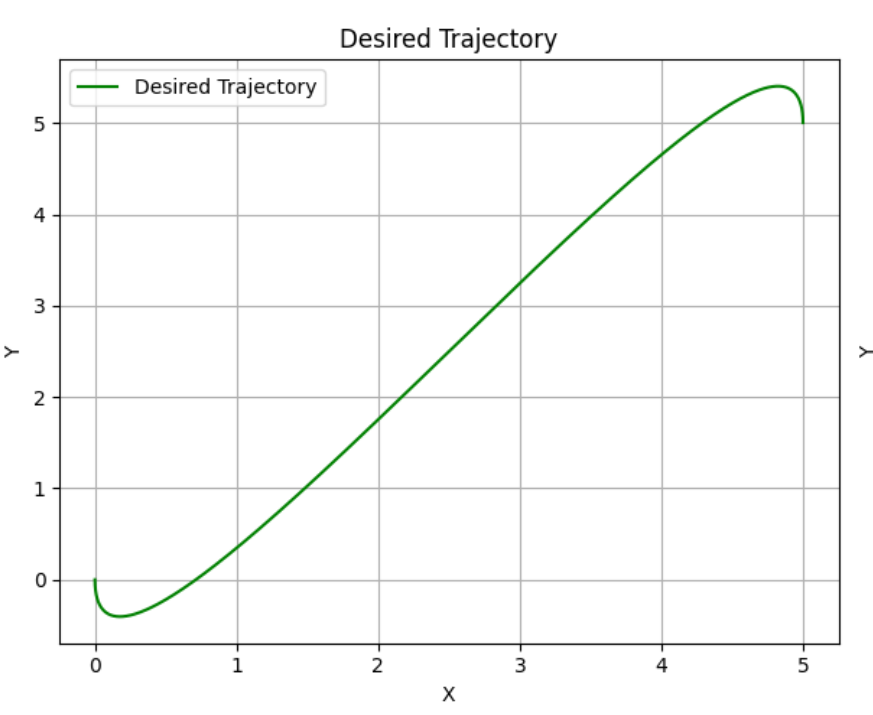
****

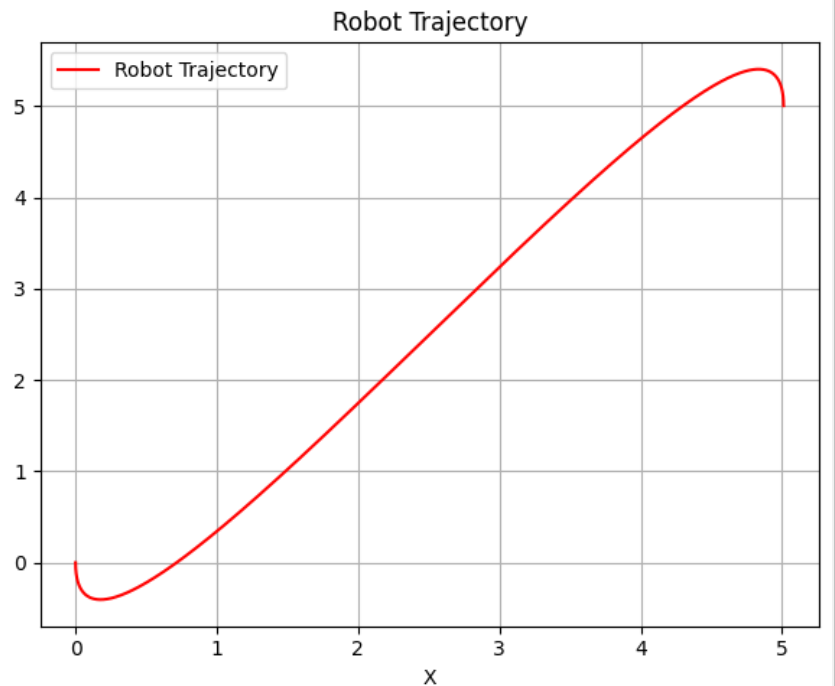
**Answer to the Question no. – 2(c)**

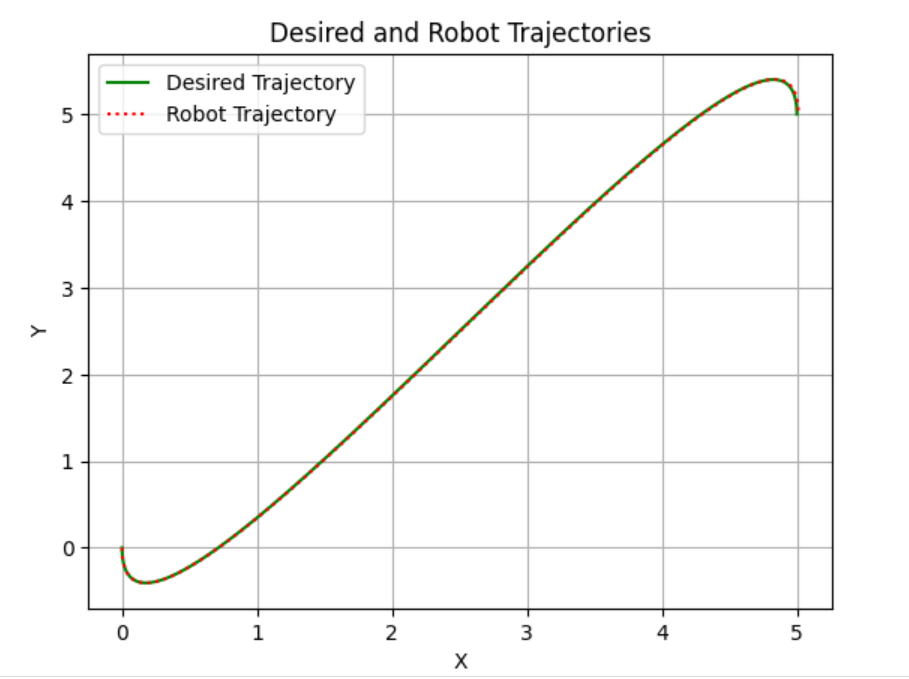
**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question2(c).ipynb*

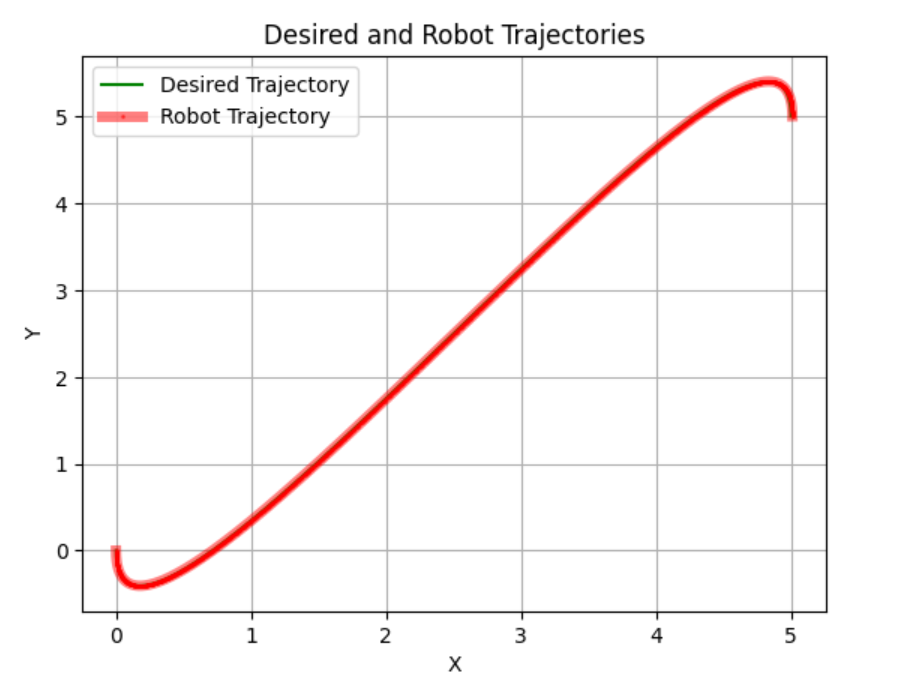
|  |  |
| --- | --- |
|  |  |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134 | 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)  # 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, Y, 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()  # Visualize the desired trajectory and robot trajectory separately  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))  # Plot 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)  # Plot 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)  # Show the plots  plt.tight\_layout()  plt.show() |

**Output:**

****

****

****

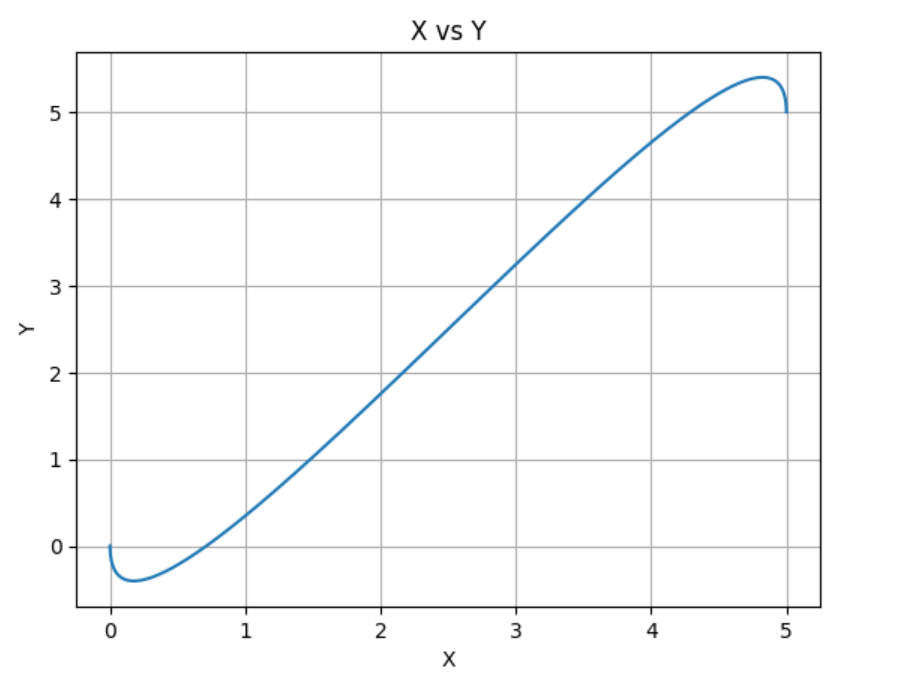
****

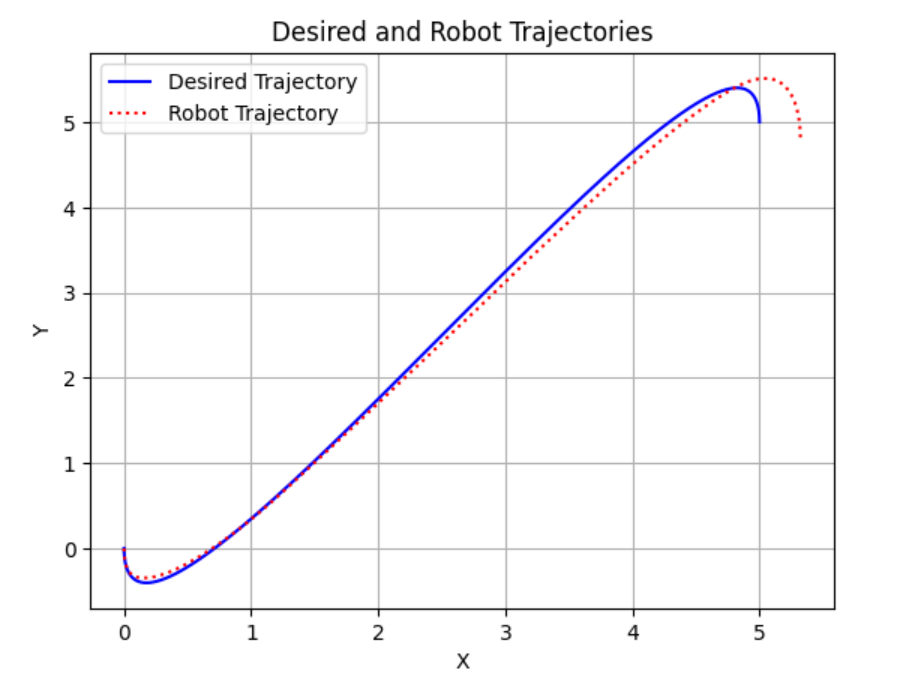
**Answer to the Question no. – 3**

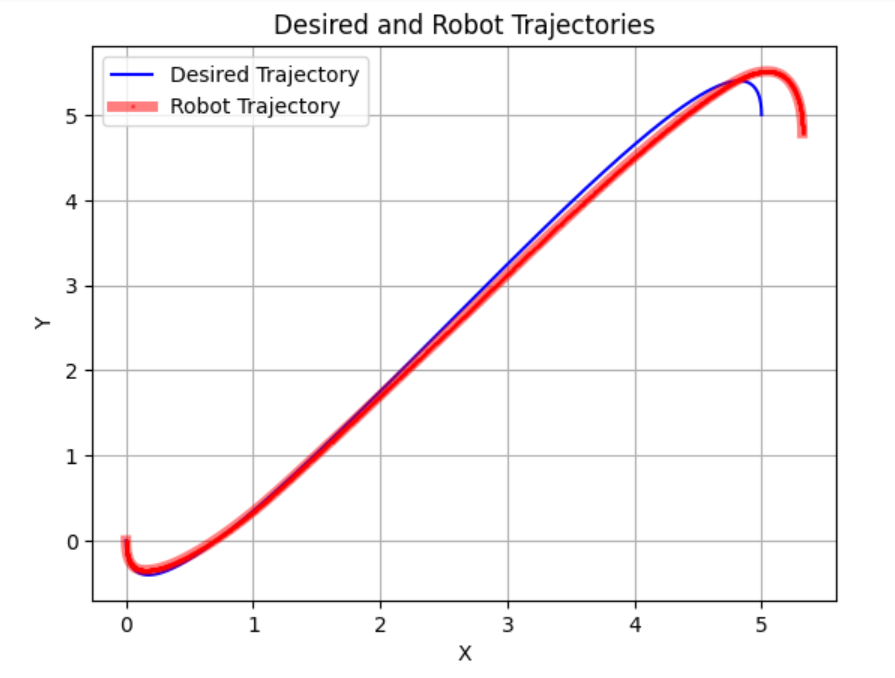
**Google Colab Code:** Source code file has also attached(Name: *HomeWork2\_Question3.ipynb*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114 | **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  # Plot X vs Y  plt.figure()  plt.plot(X, Y)  plt.title('X vs Y')  plt.xlabel('X')  plt.ylabel('Y')  plt.grid(True)  # 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 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:**

****

****

****