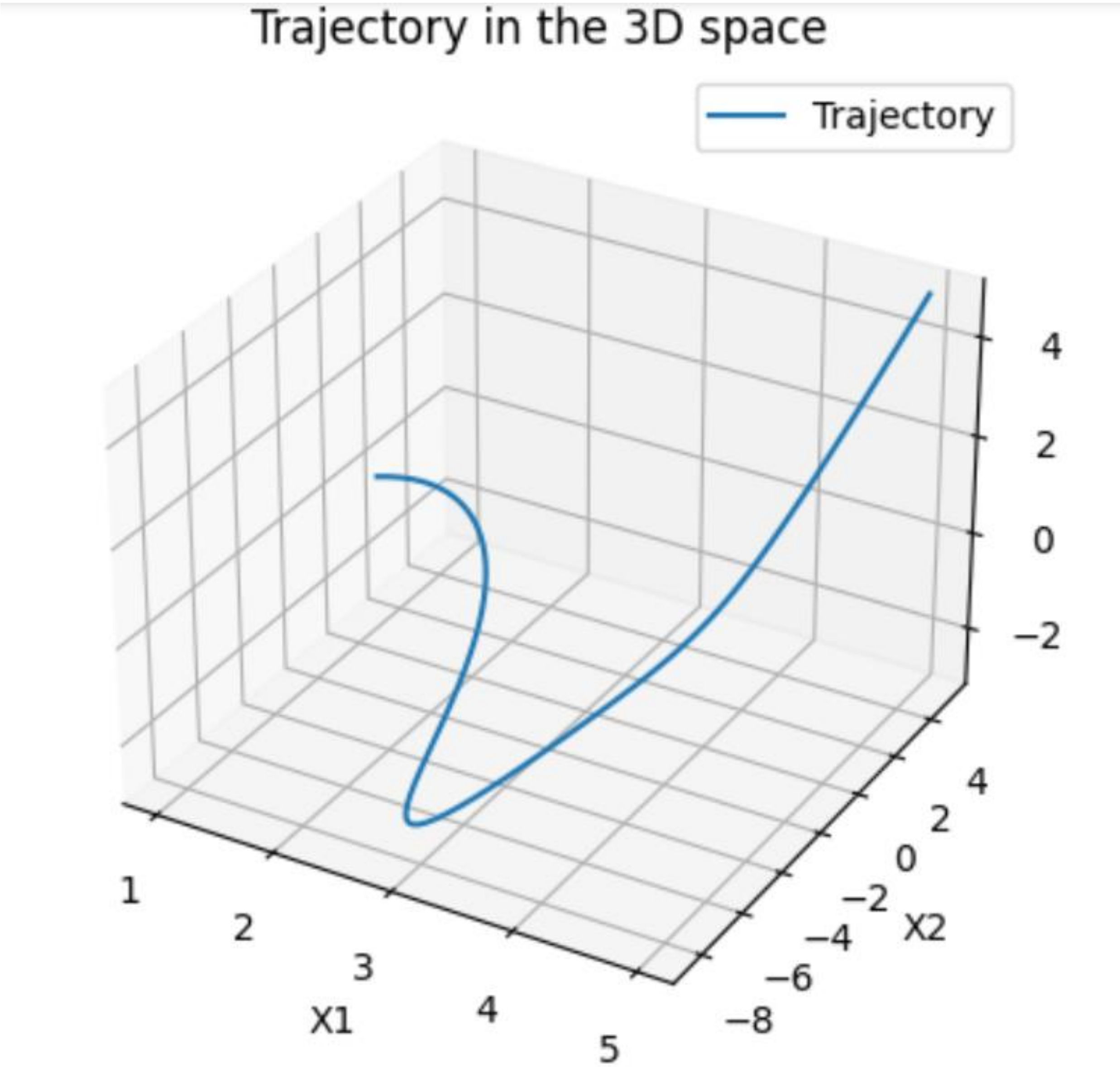
Answer to the Question no. – 1(di)

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:



Trajectory in the 3D space
Trajectory
4
2
0
−2
4
2
0
−2
−4
−6
−8
X2
1
2
3
4
5
X1

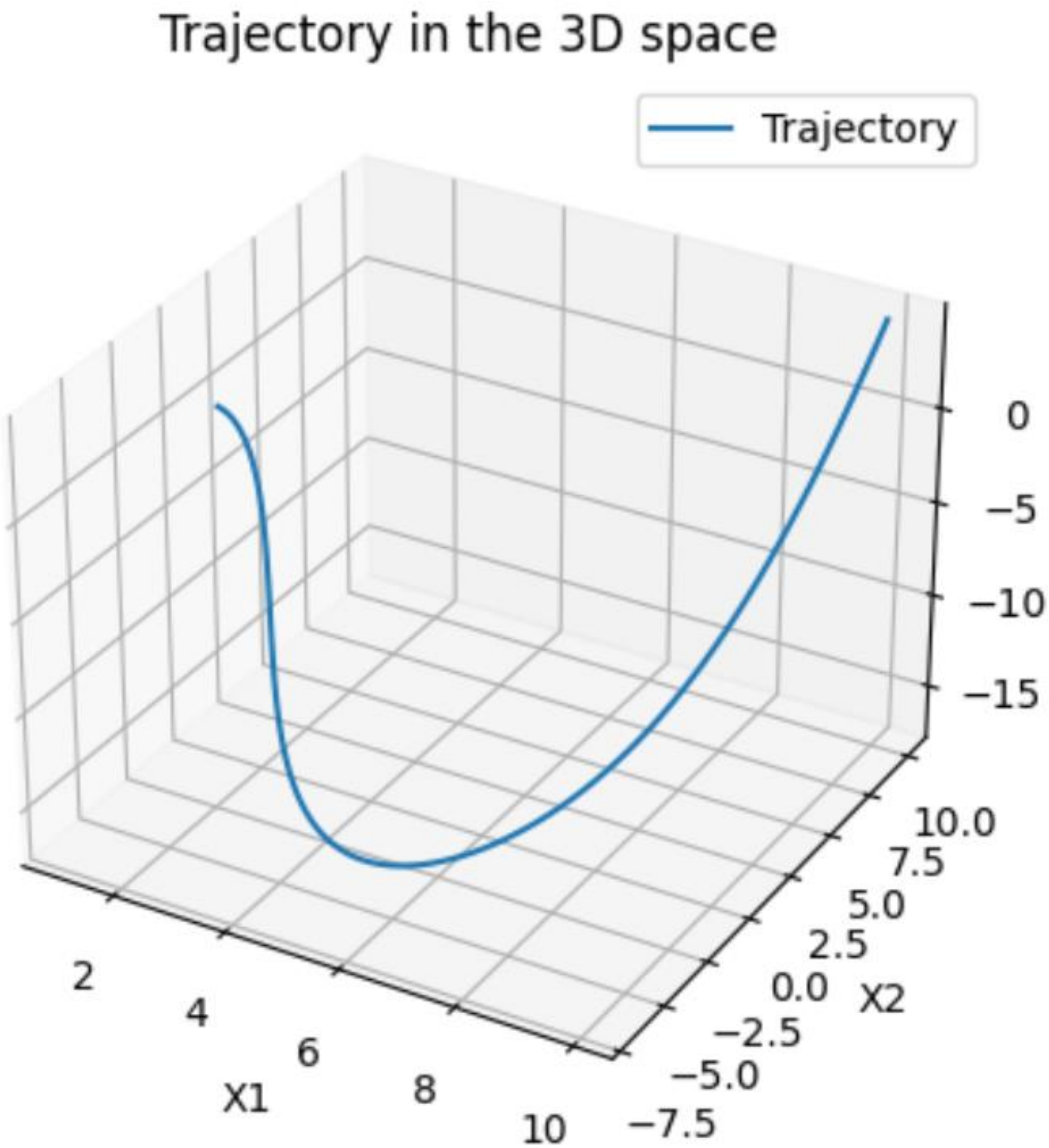
Answer to the Question no. – 1(dii)

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:


Trajectory in the 3D space
Trajectory
0
−5
−10
−15
10.0
7.5
5.0
2.5
0.0
X2
−2.5
−5.0
−7.5
2
4
6
8
10
X1

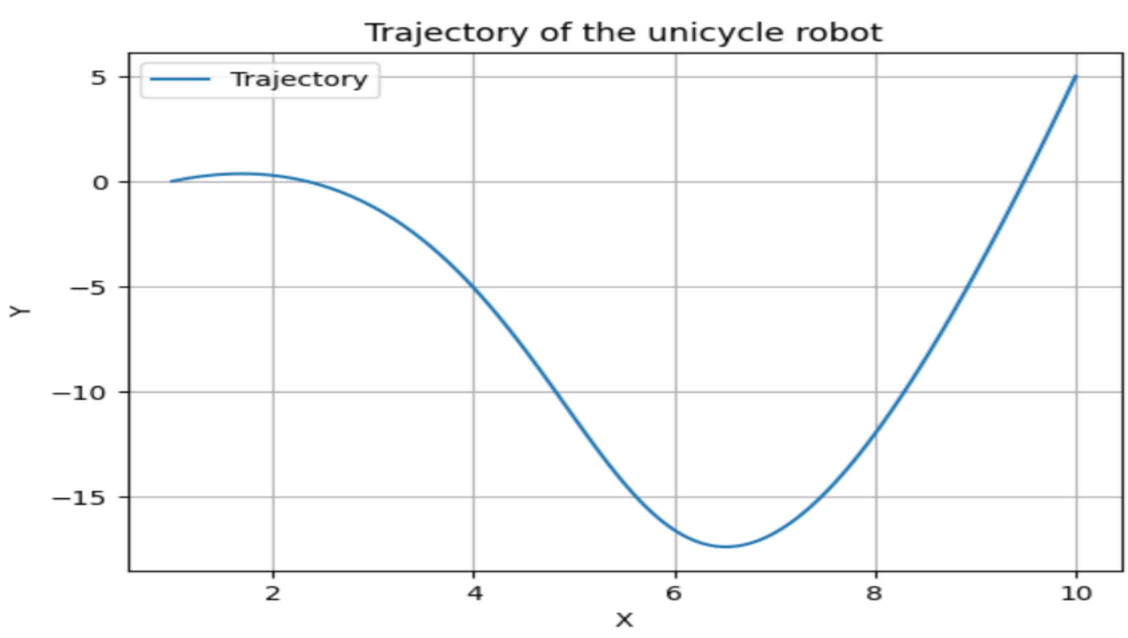
Answer to the Question no. – 2(b)

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)

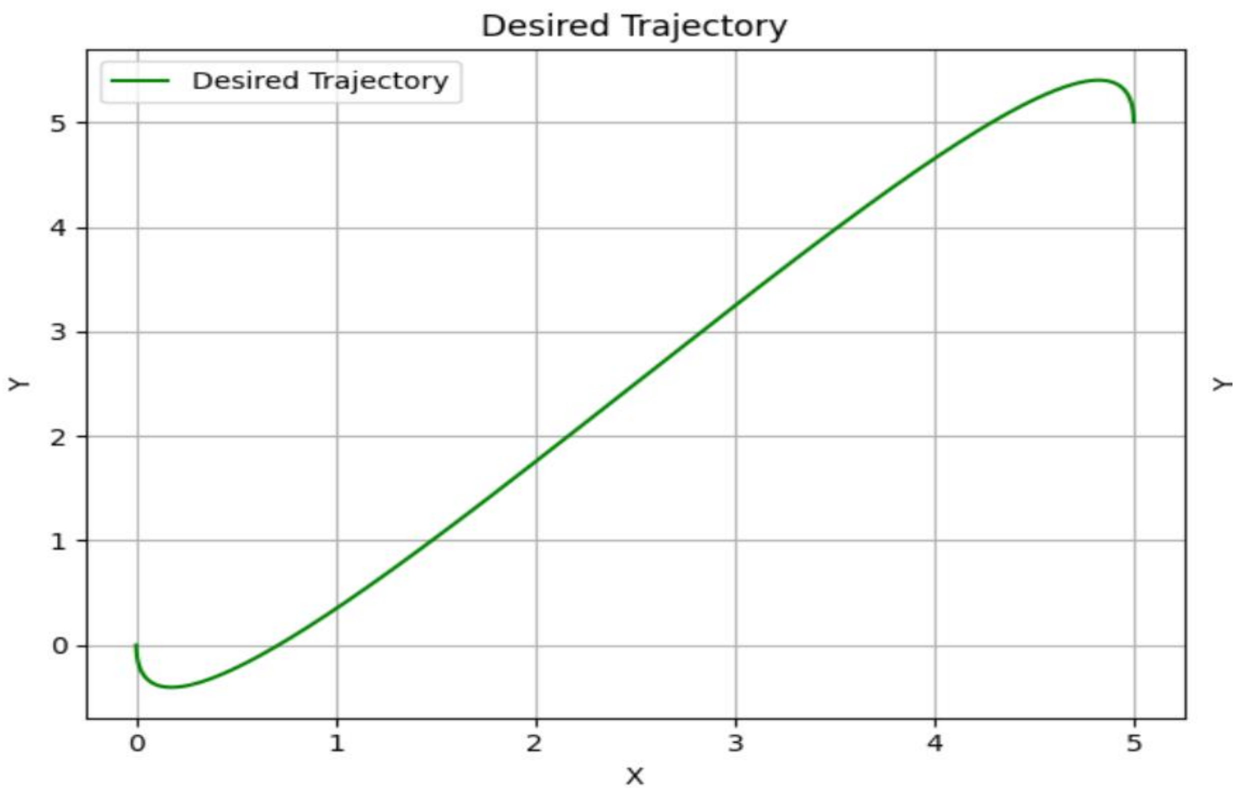
Google Colab Code: Source code file has also attached(<u>Name</u>: *HomeWork2_Question2(c).ipynb*

```
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]
```

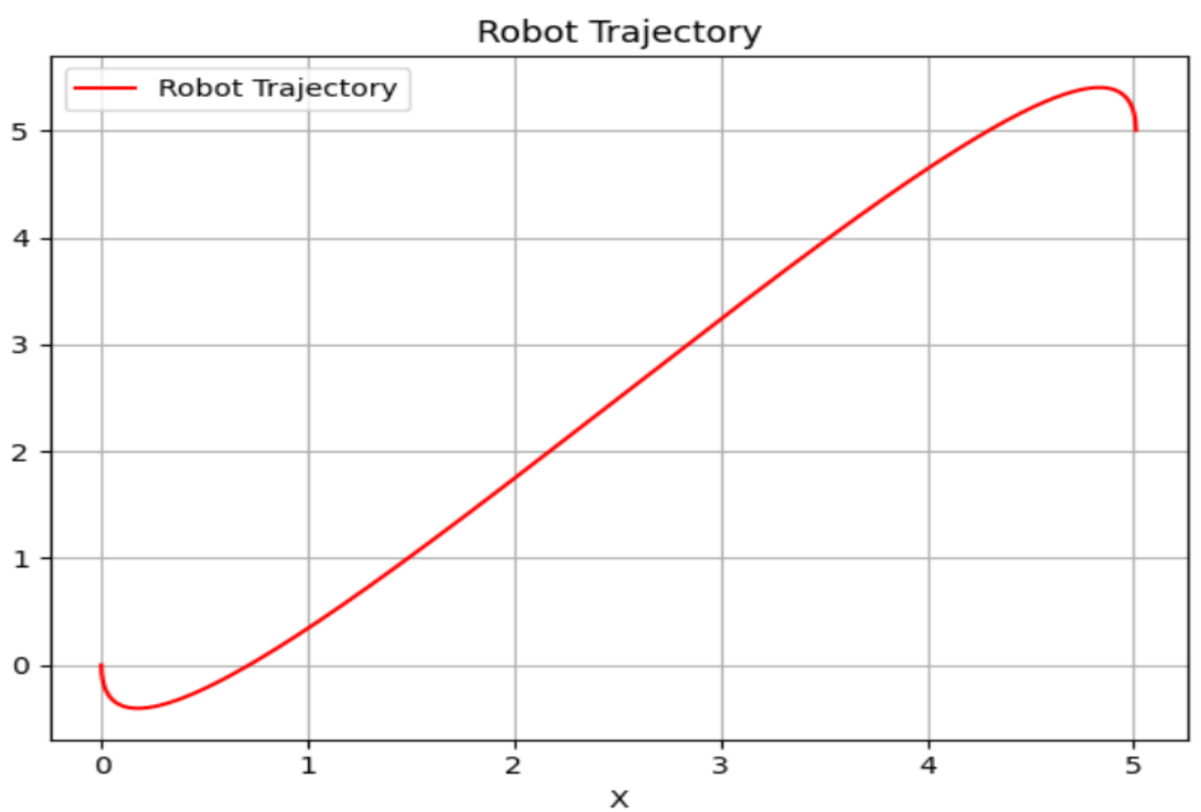
```
a21, a22, a23, a24 = x[4], x[5], x[6], x[7]

# 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()
```

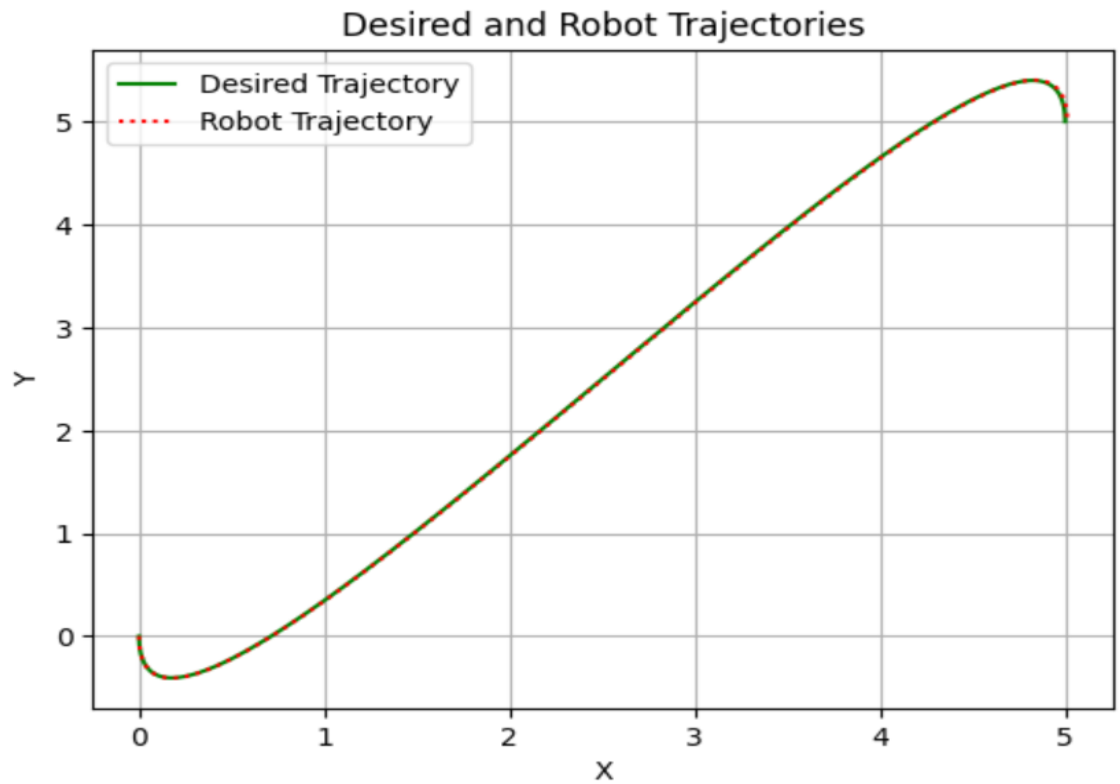
Output:

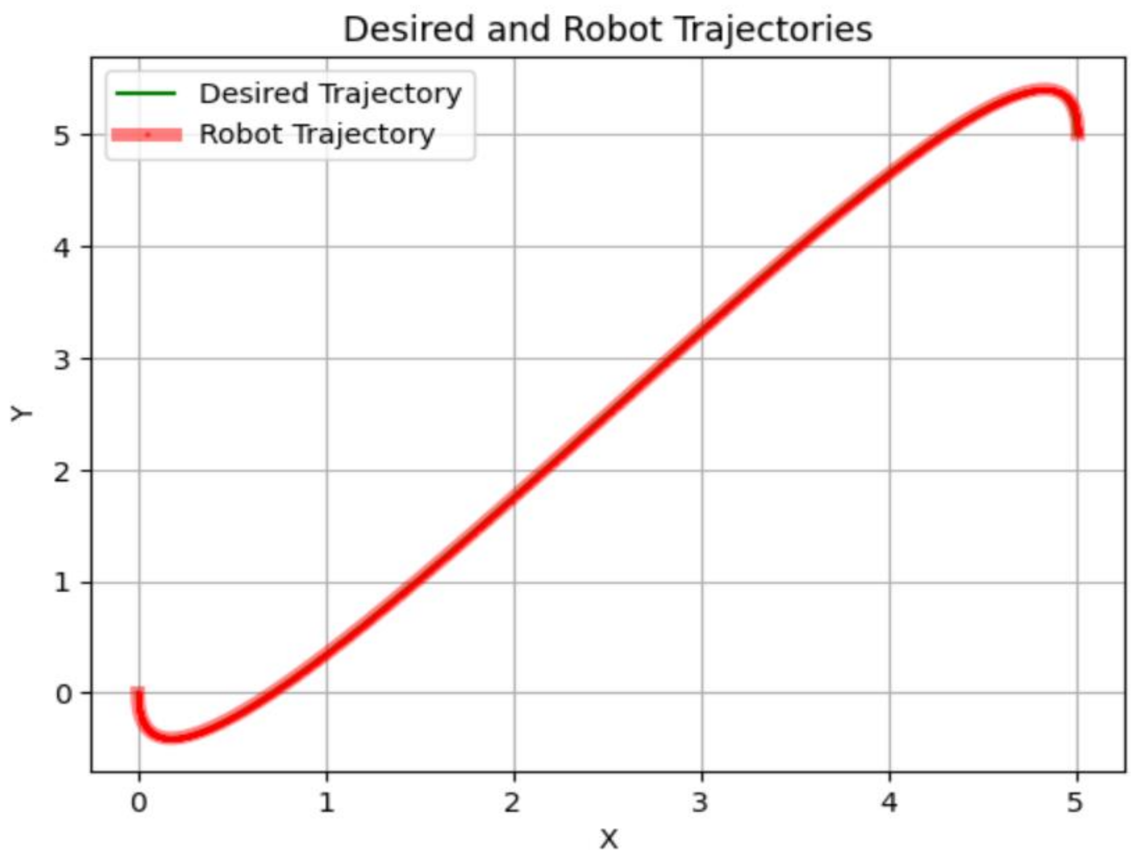


Desired Trajectory
Desired Trajectory
Y
X
0
1
2
3
4
5



Robot Trajectory
Robot Trajectory
X
0
1
2
3
4
5


Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5


Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5

Answer to the Question no. – 3

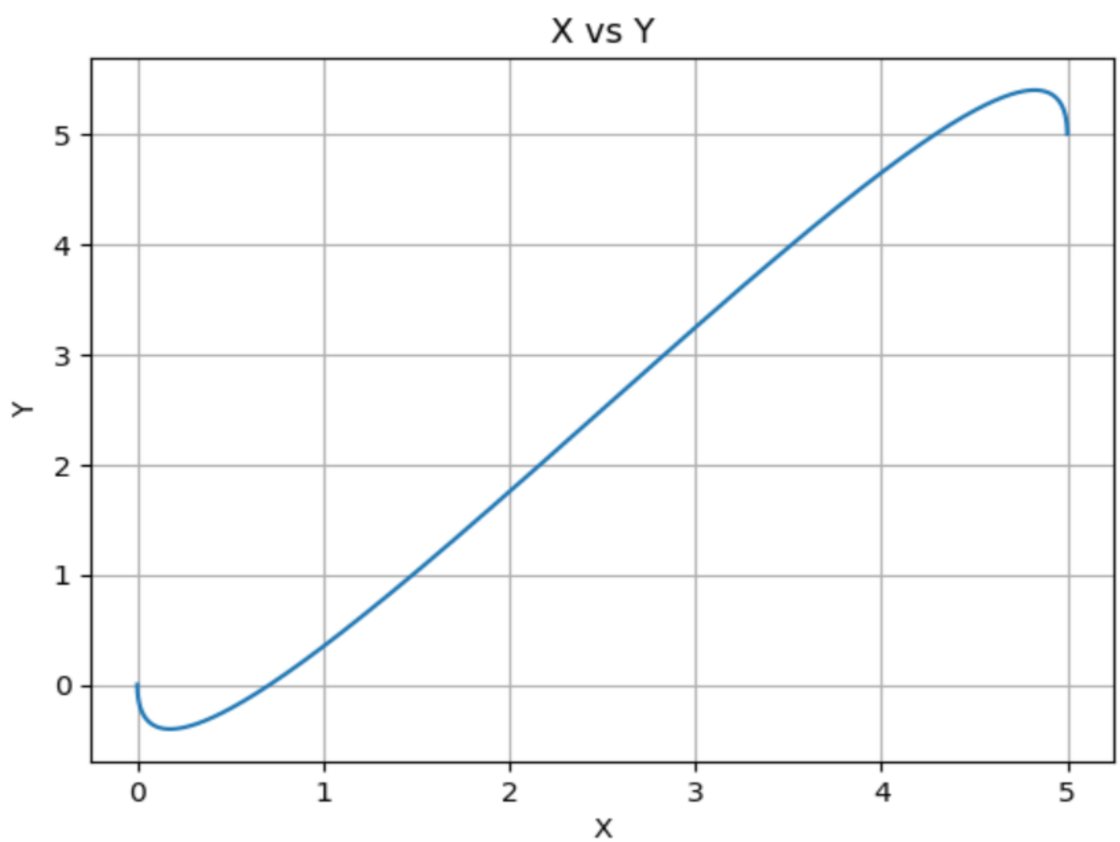
Google Colab Code: Source code file has also attached(Name: *HomeWork2_Question3.ipynb*

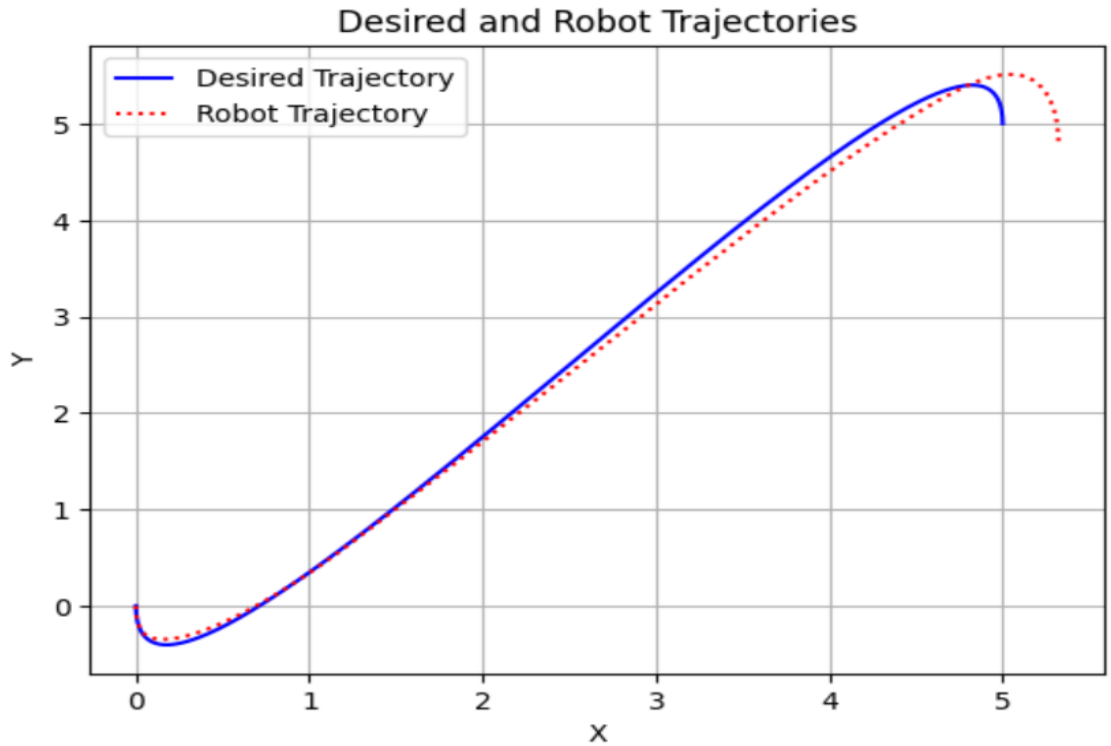
```
import numpy as np
import matplotlib.pyplot as plt

# 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
```

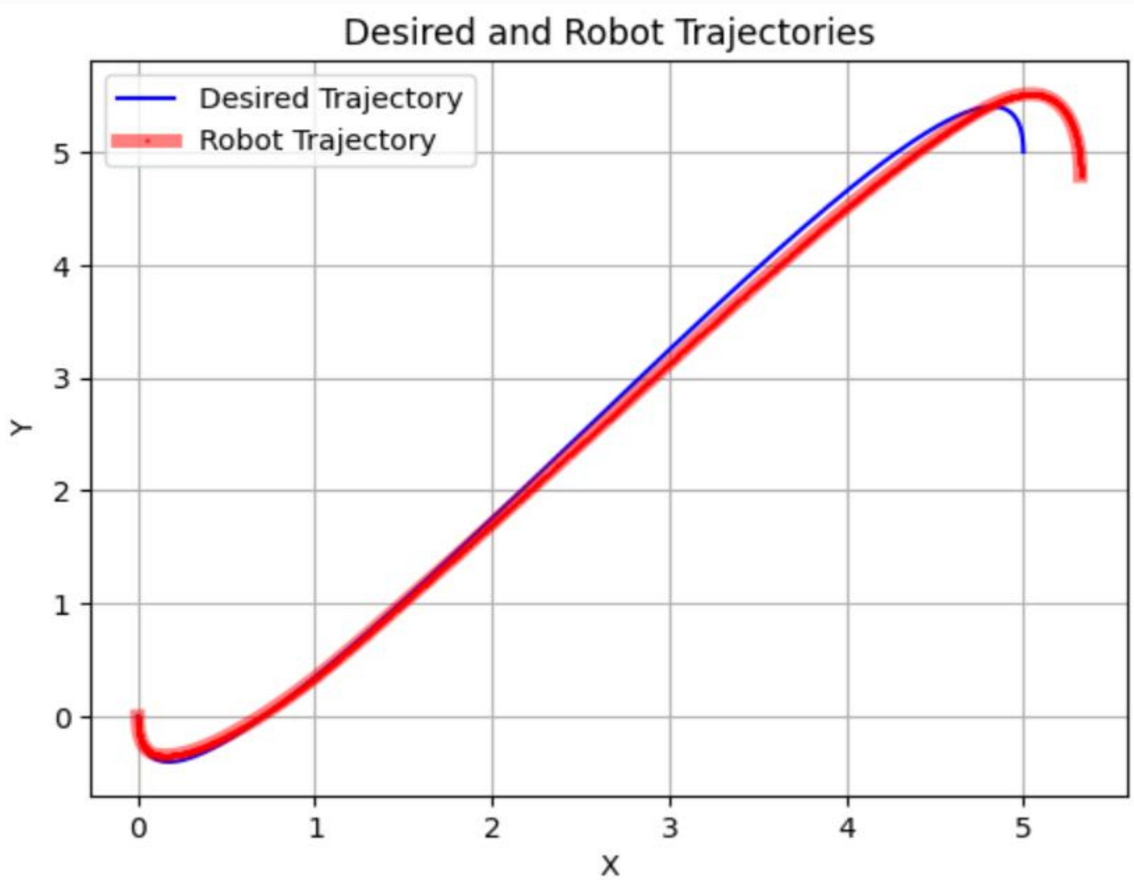
```
X = a11 + a12 * t + a13 * t**2 + a14 * t**3
Y = a21 + a22 * t + a23 * t**2 + a24 * t**3

# 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:




Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5


Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5