4.a)Differential flatness means that you can express all state variables as functions of a set of flat outputs and their derivatives. In this case, we want to show that (x1, x3) can serve as flat outputs.

Given the system dynamics:

x1_dot = u1
x2_dot = u2
x3_dot = x2 * u1

We need to find expressions for x1 and x3 in terms of flat outputs and their derivatives.

For x1:
Integrate x1_dot with respect to time:

$x1 = \int(u1)\, dt$

For x3:
Integrate x3_dot with respect to time:

$x3 = \int(x2 * u1)\, dt$

Now, let's express u1 and u2 in terms of x1 and x3. To do this, we can differentiate the expressions for x1 and x3 with respect to time and solve for u1 and u2:

For u1:

u1 = x1_dot

For u2:

u2 = x2_dot

So, we have successfully expressed u1 and u2 in terms of x1, x2, and x3, which means that the system is differentially flat for the flat output z = (x1, x3).

b) We want to express x1, x2, and x3 in terms of the basis functions y1, y2, y3, and y4. We can use the following matrices:

Let's define the following matrices:

A = [0, 1, 0; 0, 0, 1; 0, 0, 0]

B = [0; 0; 1]

x = [x1; x2; x3]

u = [u1]

y = [y1; y2; y3; y4]

Now, the system dynamics can be written as:
x_dot = Ax + Bu

y = Cx

Where C is the matrix that relates the state vector x to the output vector y.

To find C, we can compute C as follows:

C = [y1_dot, y2_dot, y3_dot, y4_dot] = [0, 1, 2 * t, 3 * t^2]

c)To plot the trajectory for the specified initial and final conditions, you can use numerical integration methods like Euler's method or a numerical ODE solver. Here's a general outline of the steps:

Define the initial conditions: x1(0), x2(0), x3(0), x4(0), and the final time Tf.
Set up a time vector from 0 to Tf with a desired time step.
Initialize an array to store the state variables (x1, x2, x3) at each time step.
Use the state space equations and Euler's method to simulate the system from t = 0 to t = Tf.
Plot the trajectories of x1, x2, and x3 as functions of time.

Here's a Python code snippet to get you started. This code assumes you have already defined the matrices A, B, C, and the initial and final conditions:

Python code:

```
import numpy as np
```

```
import matplotlib.pyplot as plt


# Define initial conditions and parameters

x1_0 = 1.0

x2_0 = 1.0

x3_0 = 0.0

x_dot_0 = 1.0

Tf = 10.0  # Final time

Dt = 0.01  # Time step

timesteps = int(Tf / Dt)


# Initialize arrays to store state and time

x1_traj = np.zeros(timesteps)

x2_traj = np.zeros(timesteps)

x3_traj = np.zeros(timesteps)

time = np.zeros(timesteps)


# Main loop using Euler's method

x1 = x1_0

x2 = x2_0

x3 = x3_0
```

```
x_dot = x_dot_0


for i in range(timesteps):
    # Store current state and time
    x1_traj[i] = x1
    x2_traj[i] = x2
    x3_traj[i] = x3
    time[i] = i * Dt


    # Euler's method for state updates
    x1_dot = x_dot
    x2_dot = x3
    x3_dot = x2 * x_dot


    x1 += x1_dot * Dt
    x2 += x2_dot * Dt
    x3 += x3_dot * Dt
    x_dot = x1_dot  # Update x_dot for the next iteration


# Plot results
plt.figure(figsize=(12, 6))
```

```
plt.subplot(3, 1, 1)
plt.plot(time, x1_traj, label='x1 vs Time')
plt.xlabel('Time')
plt.ylabel('x1')
plt.title('x1 vs Time')

plt.subplot(3, 1, 2)
plt.plot(time, x2_traj, label='x2 vs Time')
plt.xlabel('Time')
plt.ylabel('x2')
plt.title('x2 vs Time')

plt.subplot(3, 1, 3)
plt.plot(time, x3_traj, label='x3 vs Time')
plt.xlabel('Time')
plt.ylabel('x3')
plt.title('x3 vs Time')

plt.tight_layout()
plt.show()
```

5.To solve this problem, we'll follow the steps you provided:

a) Show that the system is differentially flat with flat output z = (x, y):

Differential flatness means that you can express all state variables as functions of a set of flat outputs and their derivatives. In this case, we want to show that (x, y) can serve as flat outputs.

Given the system dynamics:

$x\_dot(t) = V(t) * \cos(\theta(t))$
$y\_dot(t) = V(t) * \sin(\theta(t))$
$V\_dot(t) = a(t)$
$\theta\_dot(t) = \omega(t)$

We need to find expressions for x and y in terms of flat outputs and their derivatives.

For x:

Integrate x_dot(t) with respect to time:

$x(t) = \int[V(t) * \cos(\theta(t))]\, dt$

For y:

Integrate y_dot(t) with respect to time:

$y(t) = \int[V(t) * \sin(\theta(t))]\, dt$

Now, let's express a(t) and ω(t) in terms of x and y. To do this, we can differentiate the expressions for x and y with respect to time and solve for a(t) and ω(t):

For a(t):

$a(t) = V\_dot(t) = d^2x/dt^2$

For ω(t):

$\omega(t) = \theta\_dot(t)$

So, we have successfully expressed a(t) and ω(t) in terms of x, y, and their derivatives, which means that the system is differentially flat for the flat output z = (x, y).

b) Generate a differentially flat trajectory using the four basis functions:

To generate a differentially flat trajectory, you can use the four basis functions (y1 = 1, y2 = t, y3 = t^2, y4 = t^3). You need to specify desired values for x and y as functions of these basis functions.

For example, you can define:

x_desired(t) = a0 * y1 + a1 * y2 + a2 * y3 + a3 * y4

y_desired(t) = b0 * y1 + b1 * y2 + b2 * y3 + b3 * y4

Here, a0, a1, a2, a3, b0, b1, b2, and b3 are coefficients that you can choose to shape your desired trajectory.

c) Calculate the controls a(t) and ω(t) to navigate the robot along the generated trajectory:

To calculate the controls a(t) and ω(t) to navigate the robot along the generated trajectory, you can use the following equations derived from the given dynamics:

> Calculate x_double_dot(t) and y_double_dot(t) using the differentials of x and y with respect to time.

Calculate a(t) by taking the matrix inverse using the equations:

> a(t) = (M * [x_double_dot(t), y_double_dot(t)])^(-1) * [V(t) * cos(θ(t)), V(t) * sin(θ(t))]
> Calculate ω(t) as ω(t) = θ_dot(t).
> You can use Euler's method or another numerical integration method to propagate the robot's state forward in time using the calculated controls a(t) and ω(t).
> Plot the resulting trajectory for different time step values (Dt = 0.1 and 0.01) to observe how the robot navigates the generated trajectory.

Here's a high-level Python code outline to help you get started:

```
# Define desired coefficients a0, a1, a2, a3, b0, b1, b2, b3
```

```
# Generate x_desired(t) and y_desired(t) using the basis functions


# Initialize time parameters and arrays to store state and controls

# Loop over time steps

    # Calculate x_double_dot(t) and y_double_dot(t)

    # Calculate a(t) and ω(t) using the equations

    # Update the state of the robot using Euler's method

    # Store the state and controls in arrays


# Plot the resulting trajectory and controls for different time step values
```

6.To simulate the open-loop controller's performance in the presence of noise or disturbances, we can use Python. In this simulation, we'll integrate the unicycle equations with Euler's method while injecting noise into the system as described. Here's a Python code example for this simulation:

```
import numpy as np

import matplotlib.pyplot as plt


# Define initial conditions

x0 = 0.0

y0 = 0.0

V0 = 0.5

theta0 = -np.pi / 2  # Initial angle in radians
```

```
tf = 15.0  # Final time

Dt = 0.01  # Time step

timesteps = int(tf / Dt)


# Initialize arrays to store state variables

x = np.zeros(timesteps)

y = np.zeros(timesteps)

V = np.zeros(timesteps)

theta = np.zeros(timesteps)


# Initialize the state variables with the initial conditions

x[0] = x0

y[0] = y0

V[0] = V0

theta[0] = theta0


# Define noise parameters

noise_std_dev_V = 0.01  # Standard deviation of noise for V

noise_std_dev_theta = 0.001  # Standard deviation of noise for theta


# Main simulation loop
```

```
for i in range(1, timesteps):

    # Calculate control inputs a(t) and omega(t) without noise (use the open-loop
controller)

    # Replace the following lines with your control input calculations from problem 5(c)

    a_t =  # Calculate a(t) without noise

    omega_t =  # Calculate omega(t) without noise



    # Inject noise into the control inputs

    noise_V = noise_std_dev_V * np.random.randn()

    noise_theta = noise_std_dev_theta * np.random.randn()



    a_with_noise = a_t + noise_V

    omega_with_noise = omega_t + noise_theta



    # Integrate the state equations with Euler's method

    x_dot = V[i - 1] * np.cos(theta[i - 1])

    y_dot = V[i - 1] * np.sin(theta[i - 1])

    V_dot = a_with_noise

    theta_dot = omega_with_noise



    x[i] = x[i - 1] + x_dot * Dt

    y[i] = y[i - 1] + y_dot * Dt
```

```
    V[i] = V[i - 1] + V_dot * Dt
    theta[i] = theta[i - 1] + theta_dot * Dt

# Plot the resulting trajectory
plt.figure(figsize=(10, 6))
plt.plot(x, y, label='Robot Trajectory with Noise')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Robot Trajectory with Noise')
plt.legend()
plt.grid(True)
plt.show()
```

In this code, we inject noise into the control inputs a(t) and omega(t) as described in your problem statement and then simulate the robot's trajectory using Euler's method. The noise causes the robot to deviate from the desired trajectory, demonstrating the effect of noise on open-loop control performance.You can adjust the values of noise_std_dev_V and noise_std_dev_theta to control the level of noise in the system. Run the code with different noise levels to observe how noise affects the robot's trajectory.