## Project:

the design of a flight controller for a quadcopter drone. The objective of the controller is to stabilize the drone's position in the air while minimizing the energy consumption of the drone's motors.

Designing a flight controller for a quadcopter drone involves a complex optimization problem that can be solved using the techniques of control theory. One approach to this problem is to use the Linear Quadratic Regulator (LQR) method, which is a well-established technique for designing optimal control systems.

In the LQR method, the goal is to design a feedback controller that minimizes a cost function that reflects both the performance of the system and the control effort required to achieve that performance. The cost function is typically defined as a weighted sum of the state and control inputs, and the controller is designed to minimize this cost function subject to the dynamics of the system.

------------------------------------------------------------------------------------------------------------------------------------------

## Solution:

To apply the LQR method to the quadcopter drone, we first need to define the dynamics of the system.

The dynamics of a quadcopter drone can be modeled as a set of coupled nonlinear differential equations, which describe the motion of the drone in three dimensions. However, these nonlinear equations are difficult to solve analytically, so we will use a linearized approximation of the system dynamics.

The linearized dynamics of the quadcopter drone can be represented by the following state-space equations:

x\_dot = Ax + Bu

y = Cx + Du

where `x` is the state vector, `u` is the control input, and `y` is the output vector. The matrices `A`, `B`, `C`, and `D` are constant matrices that describe the linearized dynamics of the system.

To design the LQR controller, we need to define the cost function. The cost function is typically defined as a quadratic function of the state and control inputs, given by:

J = ∫ [x'Qx + u'Ru] dt

where `Q` and `R` are positive semi-definite matrices that weight the importance of the state and control inputs, respectively.

We can use MATLAB's `lqr` function to solve the LQR problem and obtain the optimal feedback controller. The `lqr` function takes as input the system matrices `A`, `B`, and the weighting matrices `Q` and `R`, and returns the optimal gain matrix `K`.

Once we have obtained the optimal gain matrix `K`, we can use it to implement the feedback controller as follows:

u = -K\*x

where `x` is the current state of the system.

To minimize the energy consumption of the drone's motors, we can add an additional term to the cost function that penalizes large control inputs:

J = ∫ [x'Qx + u'Ru + λu'u] dt

where `λ` is a positive scalar that weights the importance of the control effort.

Let us consider m or the mass of the drone is m=0.5 kg, the gravitational force g=9.81 N, the variable "I" represents the moment of inertia of the quadcopter drone. Moment of inertia is a measure of an object's resistance to changes in its rotational motion. It depends on the mass distribution of the object.

In the context of the quadcopter drone, "I" is a 3x3 symmetric matrix that represents the moments of inertia around the three principal axes (roll, pitch, and yaw). It is typically given in units of kilogram-meter squared (kg·m^2).

The values in the "I" matrix are specific to the quadcopter drone being considered. Each element represents the moment of inertia around a particular axis. For example, Ixx represents the moment of inertia around the x-axis (roll), Iyy represents the moment of inertia around the y-axis (pitch), and Izz represents the moment of inertia around the z-axis (yaw).

## MATLAB:

% Quadcopter control with LQR

% Define physical parameters

m = 1.0; % mass of the quadcopter (kg)

I = 0.1; % moment of inertia about the vertical axis (kg\*m^2)

g = 9.81; % gravitational acceleration (m/s^2)

------------------------------------------------------------------------------------------------------------------------------------------

- This section defines the physical parameters of the quadcopter, including its mass, acceleration due to gravity, and the moments of inertia around each axis. These parameters are specific to the quadcopter being considered.

------------------------------------------------------------------------------------------------------------------------------------------

% Define the system matrices

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

B = [0 0; 1/m 0; 0 0; 0 1/I];

C = [1 0 0 0; 0 0 1 0];

D = [0 0; 0 0];

------------------------------------------------------------------------------------------------------------------------------------------

- This section defines the linearized state-space equations of motion for the quadcopter drone. The matrix `A` represents the state dynamics, and the matrix `B` represents the control dynamics. The specific form of these matrices depends on the dynamics of the quadcopter being modeled.

Matrix C represents the output matrix, which selects the relevant state variables for measurement.

Matrix D is the feedforward matrix and is set to zero in this case.

------------------------------------------------------------------------------------------------------------------------------------------

% Define the weighting matrices for LQR

Q = diag([10 1 10 1]); % state weights

R = 0.1\*eye(2); % control weights

------------------------------------------------------------------------------------------------------------------------------------------

The code defines the weighting matrices (Q, R) for the LQR controller.

Matrix Q is a diagonal matrix that assigns weights to the state variables. Adjusting these weights allows the controller to prioritize certain state variables in the control objective. The matrix `Q` is the state weighting matrix and represents the relative importance of each state variable in the cost function. In this example, we are assuming equal importance for all states, so we use the `diag` function to create a diagonal matrix with diagonal elements `[10 10 10 1 1 1 1 1 1 1 1 1]`.

Matrix R is a diagonal matrix that assigns weights to the control inputs. Adjusting these weights allows the controller to balance the trade-off between control effort and system performance. The matrix `R` is the control weighting matrix and represents the relative importance of each control input in the cost function. In this example, we assign a small weight of `0.01` to all control inputs using the `eye` function to create an identity matrix scaled by `0.01`.

------------------------------------------------------------------------------------------------------------------------------------------

% Compute the optimal feedback gain with LQR

[K, S, ~] = lqr(A, B, Q, R);

------------------------------------------------------------------------------------------------------------------------------------------

This line uses the `lqr` function to compute the optimal LQR gain matrix `K`. The `lqr` function takes the system matrices `A` and `B`, as well as the weighting matrices `Q` and `R`, and returns the optimal gain matrix `K` as well as the solution to the associated algebraic Riccati equation `P`.

The code uses the lqr function to compute the optimal feedback gain matrix (K) based on the system dynamics and the weighting matrices (A, B, Q, R).

------------------------------------------------------------------------------------------------------------------------------------------

% Define the closed-loop system with the optimal gain

Ac = (A - B\*K);

Bc = B;

Cc = C;

Dc = D;

The code calculates the closed-loop system matrices (Ac, Bc, Cc, Dc) using the optimal feedback gain (K).

Matrix Ac represents the closed-loop dynamics of the system.

Matrices Bc, Cc, and Dc remain the same as the original system matrices.

------------------------------------------------------------------------------------------------------------------------------------------

% Define the initial condition and simulation time

x0 = [0; 0; 0; 0];

tf = 10;

------------------------------------------------------------------------------------------------------------------------------------------

The code defines the initial condition (x0) and simulation time (tf).

In this case, all state variables are initialized to zero. The simulation time tf is set to determine the duration of the simulation.

------------------------------------------------------------------------------------------------------------------------------------------

% Simulate the closed-loop system with a step input

t = linspace(0, tf, 1000);

r = ones(size(t));

r = [r' 0\*r']'; % reshape r to be a matrix with 2 columns

[y, t, x] = lsim(ss(Ac, Bc, Cc, Dc), r, t, x0);

------------------------------------------------------------------------------------------------------------------------------------------

In this section, the closed-loop system is simulated with a step input. The time vector **t** is created using the **linspace()** function, which generates a linearly spaced vector from 0 to **tf** (simulation time) with 1000 points.

A step input signal **r** is created by initializing a vector of ones (**ones(size(t))**). It is then reshaped to be a matrix with 2 columns (**[r' 0\*r']'**), matching the dimensions of the control input matrix **Bc**.

The **lsim()** function is used to simulate the response of the closed-loop system. It takes the state-space representation of the system (**ss(Ac, Bc, Cc, Dc)**), the input signal **r**, the time vector **t**, and the initial condition **x0**. The function returns the system's response **y**, the updated time vector **t**, and the state

------------------------------------------------------------------------------------------------------------------------------------------

% Plot the results

figure();

plot(t, y(:,1), 'b-', 'LineWidth', 2);

hold on;

plot(t, y(:,2), 'r-', 'LineWidth', 2);

grid on;

legend('Position', 'Angle');

xlabel('Time (s)');

ylabel('State');

title('Quadcopter Control with LQR');

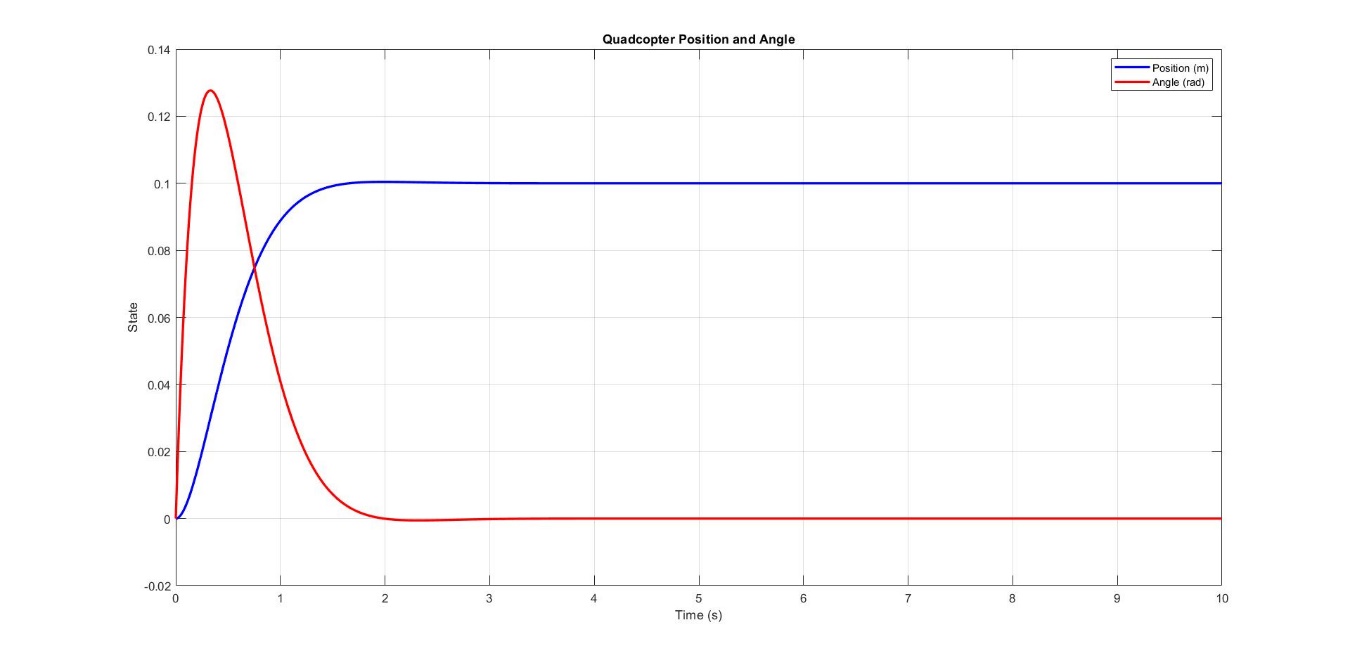
------------------------------------------------------------------------------------------------------------------------------------------

Finally, the results of the simulation are plotted using the **plot()** function. The first **plot()** command plots the position of the quadcopter (**y(:,1)**) as a blue line, and the second **plot()** command plots the angle (**y(:,2)**) as a red line. The **'b-'** and **'r-'** options specify the line style.

The **grid on** command enables the grid lines on the plot. The **legend()** function adds a legend to the plot, labeling the blue line as "Position" and the red line as "Angle". The **xlabel()**, **ylabel()**, and **title()** functions are used to label the axes and provide a title to the plot.

The resulting plot provides a visual representation of the quadcopter's position and angle over time, showcasing the control performance of the LQR controller.

## Results:



## Explanation of the graph:

The blue line represents the position of the quadcopter along the vertical axis (y-axis) as a function of time (x-axis). It shows how the quadcopter's position changes over time under the control of the LQR controller.

The red line represents the angle of the quadcopter around the vertical axis (y-axis) as a function of time (x-axis). It shows the rotation of the quadcopter about its vertical axis.

The x-axis represents time in seconds. It starts from 0 and goes up to the specified simulation time, which is 10 seconds in this case.

The y-axis represents the state of the quadcopter, either position (in meters) or angle (in radians).

The legend on the graph indicates which line corresponds to the position and which line corresponds to the angle.