|  |  |  |
| --- | --- | --- |
| gama_logo_color1 | **Cairo University Faculty of Engineering Aerospace Department** | þÿ |

**AER 651 (Optimal Control)**

LQR-Feedback Linearization Quad-Rotor Control

Submitted to Prof. Gamal Bayoumy

|  |  |  |
| --- | --- | --- |
| **Name:** | ***Mohamed Ahmed Hassan Ahmed*** | **Code: 202210656**  **Date: 2023*/* 5 */ 10*** |
| **Report No:** | **4** |

LQR-Feedback Linearization Quad-Rotor Control

# Abstract

The report aims to investigate the potentials of Optimal control techniques like Linear quadratic regulators through designing LQR for a quad rotor to control the attitude and the altitude which are the inner loop of controlling a quad rotor. The goal is to investigate the applicability of such a controller and the performance. Based on my previous work of autopilot and rigid body solver I could successfully have the simulation done and along with the knowledge acquired from optimal control course beside investigating a novel work (2022) of a master degree student which was a partial fulfillment of the requirement for the degree of master of science in electrical engineering (attached with the report), I came up with the following results which demonstrate the good performance the applicability of such a control system. My future plans to complete this project by design the outer control loop which control the position .

# Definition of state variables

# Inner Control Loop ( Altitude + ) State Variables

# System Equations

Where,

By rearranging state space into and applying small angle approximation to neglect the terms, we shall end up with the following representation:

# Feedback Linearization (Input Transformation)

Now we shall define new set of inputs in order to linearize the system

1. For Altitude
2. For angle
3. For angle
4. For angle

So that the system equation shall be expressed in new linear state space as following:

# Linear Quadratic Regulator

For the simple linear state space, we could design LQR so that we can achieve state feedback control based on the error between desired input and current state of the system.

Then we shall use the relation between the control inputs to calculate the original system control inputs which are the thrust and moments required.

I have chosen the Q and R matrices to be as following:

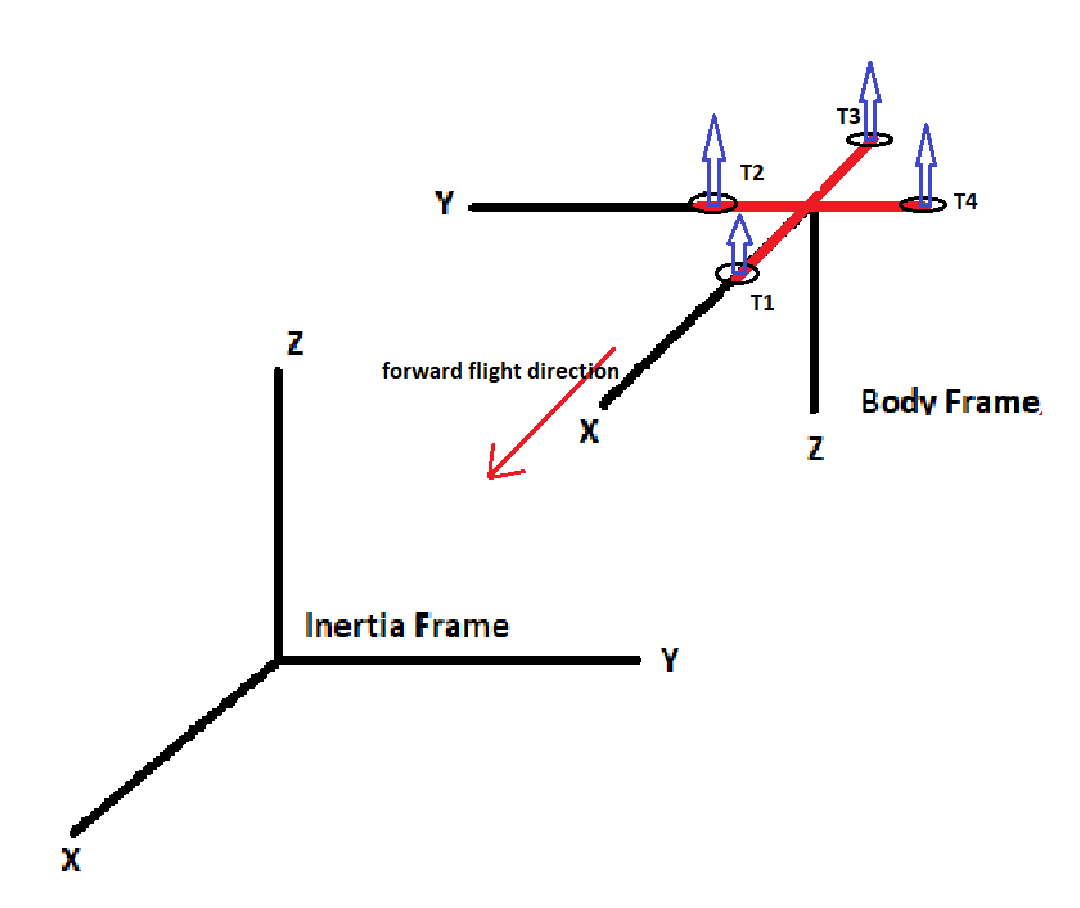
Then, the gain matrix value of K is obtained by solving the Riccati’s algebraic equation performed with MATLAB using LQR function:

# Quad-Rotor Simulation Parameters Specifications

All parameters in the SI unit’s

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Mass |  |
|  |  |
| Arm length |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Axes Convention and Motors Placement



# Results

## Hovering State @ 5 m altitude and zero initial conditions

|  |  |
| --- | --- |
|  |  |
|  |  |

## Climbing from 5m to 10m altitude and zero other initial condition

|  |  |
| --- | --- |
|  | |
|  |  |
|  |  |

## Climbing from 5m to 10m with non-zero initial conditions

|  |  |
| --- | --- |
|  | |
|  |  |
|  |  |

# Code

## Main.m

|  |
| --- |
| clc, clear , close all  *%% Inputs & Configurations*  g = 9.81;  Mass = 1.605; *% Mass Kg*  aLen = 0.24; *% arm length meter*  Ixy=0; Iyz=0; Ixz=0; *% inertia Kg m^2*  Ixx = 0.0017; Iyy = 0.0166; Izz = 0.0218;  Inertia = [*...*  Ixx, -Ixy, -Ixz;*...*  -Ixy, Iyy, -Iyz;*...*  -Ixz, -Iyz, Izz];  K\_th = 2.7778e-08;  K\_tor = 3.6559e-07;  K\_matrix = [*...*  K\_th K\_th K\_th K\_th;  K\_th\*aLen 0 -K\_th\*aLen 0;  0 -K\_th\*aLen 0 K\_th\*aLen;  K\_tor -K\_tor K\_tor -K\_tor  ];  K\_matrix\_inv = inv(K\_matrix);  *% [u; v; w; p; q; r; phi; theta; epsi; xe0; ye0; ze0]*  initialState = [0; 0; 0; 5\*pi/180; 5\*pi/180; 0; 10\*pi/180; 0\*pi/180; 10\*pi/180; 0; 0; -5]; *%% example of free fall0*  gravity\_Iframe = [0;0;Mass\*g];  *%% simulation options*  final\_time = 10;  Ts = 0.01;  timeSteps = final\_time/Ts;  time\_V = 0:Ts:final\_time;  result = NaN(12, timeSteps);  dForces = NaN(timeSteps, 3);  dMoments = NaN(timeSteps, 3);  result(:, 1) = initialState;  *%% Desired values - ref signal*  z\_d = -10;  omega\_d = [0;0;0]; *% phi theta epsi*  *%% inital rigid body solver*  rigidBodySolver = RigidBodySolver(Mass, Inertia, Ts, g);  dForces(1, :) = [0 0 -Mass\*g];  dMoments(1, :) = [0 0 0];  *%% LQR Controller Gains*  run('lqr\_linearized\_feedback\_system');  *%% Simulation*  **for** i =1:timeSteps    *% Rigid Body Solver*  result(:, i+1) = rigidBodySolver.nextStep(result(:, i), dForces(i, :)', dMoments(i, :)');    *% Controller 1*  u\_0 = controller\_lqr(Inertia, Mass, g, K\_matrix\_inv, rigidBodySolver, omega\_d, z\_d, result(:, i+1), LQR\_K);    *% Motor Rotional Speed to Force and Moments*  thrust\_moment = K\_matrix\*u\_0.^2;  dForces(i+1, :) = [0 0 -thrust\_moment(1)];  dMoments(i+1, :) = thrust\_moment(2:4)';  **end**  *%% Plotting*  u = result(1,:); p = result(4,:)\*180/pi;  v = result(2,:); q = result(5,:)\*180/pi;  w = result(3,:); r = result(6,:)\*180/pi;  phi = result(7,:)\*180/pi; x = result(10,:);  theta = result(8,:)\*180/pi; y = result(11,:);  psi = result(9,:)\*180/pi; z = result(12,:);  Vtotal = (u.^2 + v.^2 + w.^2).^(0.5);  beta\_deg = asind(v./Vtotal); alpha\_deg = atand(w./u);  plot3(x,-y,-z); title('Position');  text(x(1),-y(1), -z(1),'\leftarrow Start Position', 'Color', 'red', 'FontSize', 10);  text(x(**end**),-y(**end**), -z(**end**),'\leftarrow Final Position')  zlim([-15 15]);xlim([-15 15]); ylim([-15 15]);  grid on;  xlabel('x'); ylabel('y'); zlabel('z');  figure;  subplot(3, 1, 1);  plot(time\_V, phi); title('Angles (degs)');  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('$\phi$', 'Interpreter', 'latex');  hold on;  yline(omega\_d(1)\*180/pi, 'Color', 'red');  grid on;  subplot(3, 1, 2);  plot(time\_V, theta);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('$\theta$', 'Interpreter', 'latex');  hold on;  yline(omega\_d(2)\*180/pi, 'Color', 'red');  grid on;  subplot(3, 1, 3);  plot(time\_V, psi);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('$\psi$', 'Interpreter', 'latex');  hold on;  yline(omega\_d(3)\*180/pi, 'Color', 'red');  grid on;  figure;  subplot(3, 1, 1);  plot(time\_V, p); title('Angler Velocities');  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('p');  grid on;  subplot(3, 1, 2);  plot(time\_V, q);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('q');  grid on;  subplot(3, 1, 3);  plot(time\_V, r);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('r');  grid on;  figure;  subplot(3, 1, 1);  plot(time\_V, x); title('Position');  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('x');  grid on;  subplot(3, 1, 2);  plot(time\_V, -y);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('y');  grid on;  subplot(3, 1, 3);  plot(time\_V, -z);  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('z');  grid on;  *%% Plotting Input Forces & Moments*  figure;  subplot(1, 2, 1);  plot(time\_V, dForces(:, 3));title('Force In Body Frame');  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('Z-Force');  legend({'$F\_{z}$'},'Interpreter', 'latex', 'FontSize', 14);  subplot(1, 2, 2);  plot(time\_V, dMoments(:, 1)); hold on;  plot(time\_V, dMoments(:, 2)); hold on;  plot(time\_V, dMoments(:, 3)); hold on;  xlabel('time (sec)', 'Interpreter', 'latex');  ylabel('Moments'); title('Moments');  legend({'$\tau\_{\phi}$', '$\tau\_{\theta}$', '$\tau\_{\psi}$'},'Interpreter', 'latex', 'FontSize', 14);  **function** u = controller\_lqr(Inertia, Mass, g, K\_matrix\_inv, rigidBodySolver, omega\_d, z\_d, state, LQR\_K)  *% LQR Gains*  K\_z = LQR\_K(1, 1); K\_z\_dot = LQR\_K(1, 2);  K\_phi = LQR\_K(2, 3); K\_phi\_dot = LQR\_K(2, 4);  K\_theta = LQR\_K(3, 5); K\_theta\_dot = LQR\_K(3, 6);  K\_epsi = LQR\_K(4, 7); K\_epsi\_dot = LQR\_K(4, 8);  *% Inertia Values*  Ixx = Inertia(1, 1); Iyy = Inertia(2, 2); Izz = Inertia(3, 3);  a1 = (Iyy - Izz) / Ixx;  a2 = (Izz - Ixx) / Iyy;  a3 = (Ixx - Iyy) / Izz;  b1 = 1/Ixx; b2 = 1/Iyy; b3 = 1/Izz;    *% Calculate Angler Rates & velocity in Inertial Frame*  phi\_theta\_epsi\_dot = rigidBodySolver.currentWr2omegaDot \* state(4:6);  x\_y\_z\_dot = rigidBodySolver.inertial2Body^-1\*state(1:3);    *% Error*  angles\_err = state(7:9) - omega\_d;  wr\_err = phi\_theta\_epsi\_dot;  z\_err = z\_d - state(12);  z\_dot\_err = -x\_y\_z\_dot(3);    *% Control Input To Feedbak Linearized System*  v1 = -K\_z\*tanh(z\_err) - K\_z\_dot\*tanh(z\_dot\_err);  v2 = -K\_phi\*angles\_err(1) - K\_phi\_dot\*wr\_err(1);  v3 = -K\_theta\*angles\_err(2) - K\_theta\_dot\*wr\_err(2);  v4 = -K\_epsi\*angles\_err(3) - K\_epsi\_dot\*wr\_err(3);    *% Force & Moments as an Input to Original System*  v1\_sat = saturation([-4 4]);  u1 = Mass\*(v1\_sat.evaluate(v1) + g);  u2 = 1/b1\*(v2 - a1\*phi\_theta\_epsi\_dot(2)\*phi\_theta\_epsi\_dot(3));  u3 = 1/b2\*(v3 - a2\*phi\_theta\_epsi\_dot(1)\*phi\_theta\_epsi\_dot(3));  u4 = 1/b3\*(v4 - a3\*phi\_theta\_epsi\_dot(1)\*phi\_theta\_epsi\_dot(2));    *% Forces & Moments to Motor Rotional Speeds*  d0 = K\_matrix\_inv\*[u1 u2 u3 u4]';  u = sign(d0).\*sqrt(abs(d0));  **end** |

## RigidBodySolver.m

|  |
| --- |
| **classdef** RigidBodySolver < handle    **properties**  Mass, Inertia, invInertia, dt, g , currentWr2omegaDot, inertial2Body  **end**    **methods**  **function** obj = RigidBodySolver(Mass, Inertia, dt,g)  obj.Mass = Mass;  obj.Inertia = Inertia;  obj.invInertia = inv(Inertia);  obj.dt = dt;  obj.g = g;  **end**    **function** state = nextStep(RBS, currentState, Force, Moments)  K = zeros(12, 4);    K(:, 1) = RBS.dt\*DOF6(RBS, currentState ,Force, Moments);  K(:, 2) = RBS.dt\*DOF6(RBS, currentState+0.5\*K(:, 1) ,Force, Moments);  K(:, 3) = RBS.dt\*DOF6(RBS, currentState+0.5\*K(:, 2) ,Force, Moments);  K(:, 4) = RBS.dt\*DOF6(RBS, currentState+K(:, 3) ,Force, Moments);  state = currentState + (*...*  K(:, 1)+*...*  2\*K(:, 2)+*...*  2\*K(:, 3)+*...*  K(:, 4))/6;  **end**    **function** F = DOF6(RBS, currentState, forces, Moments)    *% (Sin, Cos, Tan) of (phi, theta, epsi)*  [S, C, T] = SCT(RBS, currentState(7:9));  s\_theta = S.theta;  c\_theta = C.theta;  t\_theta = T.theta;  s\_epsi = S.epsi;  c\_epsi = C.epsi;  s\_phi = S.phi;  c\_phi = C.phi;    RBS.inertial2Body = inertia2Body(RBS, S, C);    Forces = forces + RBS.inertial2Body\*[  0;  0;  RBS.Mass\*RBS.g  ];  *% (u, v, w) dot*  u\_v\_w\_dot = (1/RBS.Mass)\*Forces - cross(*...*  currentState(4:6, 1), currentState(1:3, 1)*...*  );  *% (p, q, r) dot*  p\_q\_r\_dot = RBS.invInertia \*(Moments - cross(*...*  currentState(4:6, 1), RBS.Inertia \* currentState(4:6, 1)*...*  ));  *% (phi, theta, epsi) dot*  RBS.currentWr2omegaDot = [  1, s\_phi\*t\_theta, c\_phi\*t\_theta;  0, c\_phi, -s\_phi;  0, s\_phi/c\_theta, c\_phi/c\_theta;  ];  phi\_theta\_epsi\_dot = RBS.currentWr2omegaDot \* currentState(4:6, 1);  *% (x, y, z) dot*  x\_y\_z\_dot = [  c\_theta\*c\_epsi, (s\_phi\*s\_theta\*c\_epsi - c\_phi\*s\_epsi), (c\_phi\*s\_theta\*c\_epsi + s\_phi\*s\_epsi);  c\_theta\*s\_epsi, (s\_phi\*s\_theta\*s\_epsi + c\_phi\*c\_epsi), (c\_phi\*s\_theta\*s\_epsi - s\_phi\*c\_epsi);  -s\_theta, s\_phi\*c\_theta, c\_phi\*c\_theta  ] \* currentState(1:3, 1);  F = [u\_v\_w\_dot; p\_q\_r\_dot; phi\_theta\_epsi\_dot; x\_y\_z\_dot];    **end**    **function** [S, C, T] = SCT(RBD, s)    sct = [*...*  sin(s(1)), sin(s(2)), sin(s(3));  cos(s(1)), cos(s(2)), cos(s(3));  tan(s(1)), tan(s(2)), tan(s(3));  ];  S = struct(*...*  'phi', sct(1,1),*...*  'theta', sct(1,2),*...*  'epsi', sct(1,3)*...*  );  C = struct(*...*  'phi', sct(2,1),*...*  'theta', sct(2,2),*...*  'epsi', sct(2,3)*...*  );  T = struct(*...*  'phi', sct(3,1),*...*  'theta', sct(3,2),*...*  'epsi', sct(3,3)*...*  );    **end**    **function** R\_i2b = inertia2Body(~, S, C)  R1 = [*...*  1 0 0;  0 C.phi S.phi;  0 -S.phi C.phi;  ];  R2 = [*...*  C.theta 0 -S.theta;  0 1 0;  S.theta 0 C.theta;  ];  R3 = [*...*  C.epsi S.epsi 0;  -S.epsi C.epsi 0;  0 0 1;  ];  R\_i2b = R1\*R2\*R3;    **end**    **end**  **end** |

## lqr\_linearized\_feedback\_system.m

|  |
| --- |
| *%% Inputs*  A = [  0 1 0 0 0 0 0 0;  0 0 0 0 0 0 0 0;  0 0 0 1 0 0 0 0;  0 0 0 0 0 0 0 0;  0 0 0 0 0 1 0 0;  0 0 0 0 0 0 0 0;  0 0 0 0 0 0 0 1;  0 0 0 0 0 0 0 0;  ];  B = [  0 0 0 0;  1 0 0 0;  0 0 0 0;  0 1 0 0;  0 0 0 0;  0 0 1 0;  0 0 0 0;  0 0 0 1;  ];  Q = diag([50 50 5 1 5 1 5 1]);  R = diag([0.002 0.002 0.002 0.002]);  *%% LQR*  LQR\_K = lqr(A, B, Q, R); |