clc  
clear  
close all  
  
addpath('..\functions')  
  
% Define plant dynamics  
Ix = 0.3; % moment of inertia (kg\*m^2)  
Iy = 0.4; % moment of inertia (kg\*m^2)  
Iz = 0.5; % moment of inertia (kg\*m^2)  
Attitude\_f = @(x) [(x(4) + x(5)\*(sin(x(1))\*tan(x(2))) + x(6)\*(cos(x(1))\*tan(x(2))));  
 (x(5)\*cos(x(1)) - x(6)\*sin(x(1)));  
 (x(5)\*sin(x(1))/cos(x(2)) + x(6)\*cos(x(1))/cos(x(2)));  
 ((Iy - Iz) / Ix \* x(5) \* x(6));  
 ((Iz - Ix) / Iy \* x(4) \* x(6));  
 ((Ix - Iy) / Iz \* x(4) \* x(5))];  
Attitude\_g = [0 0 0; 0 0 0; 0 0 0; 1/Ix 0 0; 0 1/Iy 0; 0 0 1/Iz];  
  
% Define training Parameters  
N\_states = 6;  
N\_patterns = 100;  
max\_training\_loop = 40000;  
threshold = 1e-5;  
dt = 0.001;  
Attitude\_Q = dt\*diag([1e6,1e6,1e6,1e5,1e5,1e5]);  
Attitude\_R = dt\*diag([0.5e4,0.5e4,0.5e4]);  
  
% Define domains of training  
PHI\_max = 1; PHI\_min = -1;  
THE\_max = 1; THE\_min = -1;  
PSI\_max = 1; PSI\_min = -1;  
  
p\_max = 1; p\_min = -1;  
q\_max = 1; q\_min = -1;  
r\_max = 1; r\_min = -1;  
  
% Partial x\_k+1 / partial x\_k  
A = @(x)...  
 [  
 dt\*(x(5)\*cos(x(1))\*tan(x(2)) - x(6)\*sin(x(1))\*tan(x(2))) + 1, dt\*(x(6)\*cos(x(1))\*(tan(x(2))^2 + 1) + x(5)\*sin(x(1))\*(tan(x(2))^2 + 1)), 0, dt, dt\*sin(x(1))\*tan(x(2)), dt\*cos(x(1))\*tan(x(2));  
 -dt\*(x(6)\*cos(x(1)) + x(5)\*sin(x(1))), 1, 0, 0, dt\*cos(x(1)), -dt\*sin(x(1));  
 dt\*((x(5)\*cos(x(1)))/cos(x(2)) - (x(6)\*sin(x(1)))/cos(x(2))), dt\*((x(6)\*cos(x(1))\*sin(x(2)))/cos(x(2))^2 + (x(5)\*sin(x(1))\*sin(x(2)))/cos(x(2))^2), 1, 0, (dt\*sin(x(1)))/cos(x(2)), (dt\*cos(x(1)))/cos(x(2));  
 0, 0, 0, 1, (dt\*x(6)\*(Iy - Iz))/Ix, (dt\*x(5)\*(Iy - Iz))/Ix;  
 0, 0, 0, -(dt\*x(6)\*(Ix - Iz))/Iy, 1, -(dt\*x(4)\*(Ix - Iz))/Iy;  
 0, 0, 0, (dt\*x(5)\*(Ix - Iy))/Iz, (dt\*x(4)\*(Ix - Iy))/Iz, 1;  
 ]; % row representation  
  
% Define simulation parameters  
t\_f = 100;  
  
% Euler integration  
Attitude\_F = @(x) x + dt \* Attitude\_f(x);  
Attitude\_G = @(x) Attitude\_g \* dt;  
  
% Additonal variables  
N\_neurons = length(Basis\_Func\_84(ones(N\_states,1)));  
N = t\_f/dt;  
Attitude\_W = rand(N\_neurons, N\_states);  
  
tic  
% Nonvectorzied SNAC training loop  
for i = 1:max\_training\_loop  
 basis\_func = zeros(N\_neurons, N\_patterns);  
 lambda\_k\_plus\_1\_target = zeros(N\_states, N\_patterns);  
 % Generating target costate for all number of patterns  
 for j = 1 : N\_patterns  
 X1 = PHI\_min + (PHI\_max - PHI\_min) \* rand;%(1, N\_patterns);  
 X2 = THE\_min + (THE\_max - THE\_min) \* rand;%(1, N\_patterns);  
 X3 = PSI\_min + (PSI\_max - PSI\_min) \* rand;%(1, N\_patterns);  
 X4 = p\_min + (p\_max - p\_min) \* rand;%(1, N\_patterns);  
 X5 = q\_min + (q\_max - q\_min) \* rand;%(1, N\_patterns);  
 X6 = r\_min + (r\_max - r\_min) \* rand;%(1, N\_patterns);  
  
 % Random states within defined domain of trainig  
 x\_k = [X1; X2; X3; X4; X5; X6];  
  
 % Running states through nerual network  
 basis\_func(:,j) = Basis\_Func\_84(x\_k);  
 lambda\_k\_plus\_1 = Attitude\_W' \* Basis\_Func\_84(x\_k);  
  
 % Optimal control equation  
 u\_k = -Attitude\_R^-1 \* Attitude\_G(x\_k).' \* lambda\_k\_plus\_1;  
  
 % Discretized dynamics  
 x\_k\_plus\_1 = Attitude\_F(x\_k) + Attitude\_G(x\_k) \* u\_k;  
  
 % States through nerual network  
 lambda\_k\_plus\_2 = Attitude\_W' \* Basis\_Func\_84(x\_k\_plus\_1);  
  
 % Target costate equation  
 A\_k\_plus\_1 = A(x\_k\_plus\_1);  
 lambda\_k\_plus\_1\_target(:,j) = Attitude\_Q \* (x\_k\_plus\_1) ...  
 + A\_k\_plus\_1.' \* lambda\_k\_plus\_2;  
 end  
  
 % Least squares to update network weights  
 Attitude\_W = (basis\_func \* basis\_func')\(basis\_func \* lambda\_k\_plus\_1\_target');  
 if isnan(Attitude\_W)  
 fprintf('Divergence in trainig \n')  
 break  
 end  
  
 % Check for convergence  
 error(:, :) = Attitude\_W' \* basis\_func - lambda\_k\_plus\_1\_target;  
 if mae(error(:,:))< threshold  
 fprintf('Weights converged, mae = %f \n', mae(error(:,:)))  
 break  
 end  
end

[*Published with MATLAB® R2022a*](https://www.mathworks.com/products/matlab)