clc  
clear  
close all  
  
addpath('..\functions')  
  
% Define plant dynamics  
grav = 9.81;  
Position\_f = @(x) [x(4); x(5); x(6); 0; 0; grav];  
Position\_g = @(x) [0 0 0; 0 0 0; 0 0 0; -1 0 0; 0 -1 0; 0 0 -1];  
  
% Define training Parameters  
N\_states = 6;  
N\_patterns = 100;  
max\_training\_loop = 40000;  
threshold = 1e-5;  
dt = 0.001;  
Position\_Q = dt\*diag([1e6,1e6,1e5,1e5,1e5,1e5]);  
Position\_R = dt\*diag([0.5e4,0.5e4,0.5e4]);  
  
% Define domains of training  
X\_max = 100; X\_min = -100;  
Y\_max = 100; Y\_min = -100;  
Z\_max = 100; Z\_min = -100;  
  
u\_max = 100; u\_min = -100;  
v\_max = 100; v\_min = -100;  
w\_max = 100; w\_min = -100;  
  
% Partial x\_k+1 / partial x\_k  
A = @(x)...  
 [  
 1, 0, 0, dt, 0, 0;  
 0, 1, 0, 0, dt, 0;  
 0, 0, 1, 0, 0, dt;  
 0, 0, 0, 1, 0, 0;  
 0, 0, 0, 0, 1, 0;  
 0, 0, 0, 0, 0, 1;  
 ]; % row representation  
  
% Euler integration  
Position\_F = @(x) x + dt \* Position\_f(x);  
Position\_G = @(x) Position\_g(x) \* dt;  
  
% Additonal variables  
N = t\_f/dt;  
N\_neurons = length(Basis\_Func\_84(ones(N\_states,1)));  
Position\_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 = X\_min + (X\_max - X\_min) \* rand;%(1, N\_patterns);  
 X2 = Y\_min + (Y\_max - Y\_min) \* rand;%(1, N\_patterns);  
 X3 = Z\_min + (Z\_max - Z\_min) \* rand;%(1, N\_patterns);  
 X4 = u\_min + (u\_max - u\_min) \* rand;%(1, N\_patterns);  
 X5 = v\_min + (v\_max - v\_min) \* rand;%(1, N\_patterns);  
 X6 = w\_min + (w\_max - w\_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 = Position\_W' \* Basis\_Func\_84(x\_k);  
  
 % Optimal control equation  
 u\_k = -Position\_R^-1 \* Position\_G(x\_k).' \* lambda\_k\_plus\_1;  
  
 % Discretized dynamics  
 x\_k\_plus\_1 = Position\_F(x\_k) + Position\_G(x\_k) \* u\_k;  
  
 % States through nerual network  
 lambda\_k\_plus\_2 = Position\_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) = Position\_Q \* (x\_k\_plus\_1) ...  
 + A\_k\_plus\_1.' \* lambda\_k\_plus\_2;  
 end  
  
 % Least squares to update network weights  
 Position\_W = (basis\_func \* basis\_func')\(basis\_func \* lambda\_k\_plus\_1\_target');  
 if isnan(Position\_W)  
 fprintf('Divergence in trainig \n')  
 break  
 end  
  
 % Check for convergence  
 error(:, :) = Position\_W' \* basis\_func - lambda\_k\_plus\_1\_target;  
 if mae(error(:,:))< threshold  
 break  
 end  
end

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