% Initial

function q\_dot = computeJacobian(q, v\_des)

coder.extrinsic('geometricJacobian');

coder.extrinsic('loadrobot');

q\_dot = zeros(6,1);

% Read robot arm model

robot = loadrobot('universalUR5', 'DataFormat', 'column');

% Compute Jacobian tool0=end\_effector

J = geometricJacobian(robot, q, 'tool0');

% Compute the Jacobian pseudoinverse

J\_pinv = pinv(J);

% Calculating joint velocities q\_dot = J\_pinv \* v\_des

q\_dot = J\_pinv \* v\_des;

end

function q\_dot = computeJacobian(q)

coder.extrinsic('geometricJacobian');

coder.extrinsic('getTransform');

coder.extrinsic('loadrobot');

% Initialize output

q\_dot = zeros(6,1);

% Load the UR5 robot model

robot = loadrobot('universalUR5', 'DataFormat', 'column');

% Compute the geometric Jacobian at the end-effector (tool0)

J = geometricJacobian(robot, q, 'tool0');

J\_pinv = pinv(J); % Compute the pseudoinverse of the Jacobian

% Define the desired end-effector position in world coordinates (in meters)

p\_des = [0.5; 0.2; 0.3]; % Target position [X; Y; Z]

T = getTransform(robot, q, 'tool0');

T = double(T); % change mxArray to double

p\_current = zeros(3,1); % initialize the size

p\_current = T(1:3, 4); % take the values

% Compute the position error vector

e = p\_des - p\_current;

% Define the proportional gain for generating desired linear velocity

Kp = 1;

% Generate desired end-effector velocity based on position error

v\_des\_linear = Kp \* e; % Only consider linear velocity

v\_des = [v\_des\_linear; zeros(3,1)]; % Combine linear and zero angular velocities

% If the error is small enough, stop the motion

if norm(e) < 0.01

v\_des = zeros(6,1);

end

% Compute joint velocities using the RRMC equation

q\_dot = J\_pinv \* v\_des;

end

有點問題

function [q\_dot, v\_des, p\_current] = computeJacobian(q)

% Declare extrinsic functions

coder.extrinsic('geometricJacobian');

coder.extrinsic('getTransform');

coder.extrinsic('loadrobot');

% Initialize outputs

q\_dot = zeros(6,1); % Joint velocity output

v\_des = zeros(6,1); % Desired end-effector velocity

p\_current = zeros(3,1); % Current end-effector position

% Load the UR5 robot model

robot = loadrobot('universalUR5', 'DataFormat', 'column');

% Compute the geometric Jacobian at the end-effector (tool0)

J = geometricJacobian(robot, q, 'tool0');

J\_pinv = pinv(J); % Compute the pseudoinverse of the Jacobian

% Define the desired end-effector position in world coordinates (in meters)

p\_des = [0.5; 0.2; 0.3]; % Target position [X; Y; Z]

% Compute the current end-effector pose

T = getTransform(robot, q, 'tool0');

T = double(T); % Convert mxArray to double

p\_current = T(1:3, 4); % Extract position

% Compute the position error vector

e = p\_des - p\_current;

% Proportional gain for velocity generation

Kp = 0.3;

% Generate desired end-effector velocity based on position error

v\_des\_linear = Kp \* e; % Only linear velocity

v\_des = [v\_des\_linear; zeros(3,1)]; % Append 0 angular velocity

% Stop motion if error is sufficiently small

if norm(e) < 0.01

v\_des = zeros(6,1);

end

if any(isnan(q\_dot)) || any(isinf(q\_dot))

q\_dot = zeros(6,1);

end

% Compute joint velocities using the RRMC formula

q\_dot = J\_pinv \* v\_des;

disp('p\_current = '); disp(p\_current);

disp('e = '); disp(e);

disp('v\_des = '); disp(v\_des);

end