**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

Khoa Điện – Điện tử

Bộ môn Kỹ thuật điều khiển và Tự động hoá



`

**BÁO CÁO MÔ PHỎNG ROBOT SCARA**

**Môn học: Kỹ thuật ROBOT**

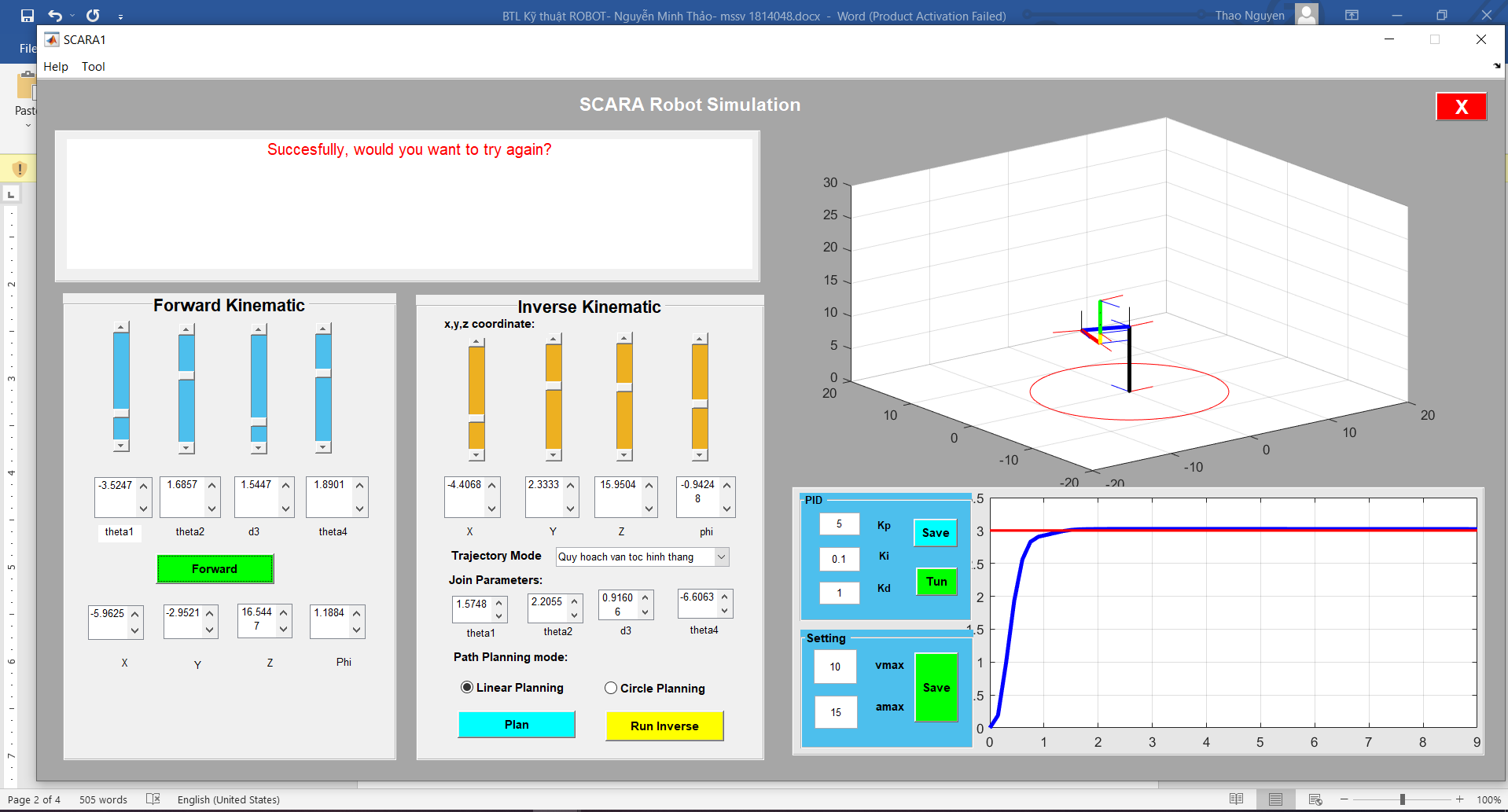
**Giảng viên hướng dẫn: Nguyễn Hoàng Giáp**

**Tên thành viên: MSSV:**

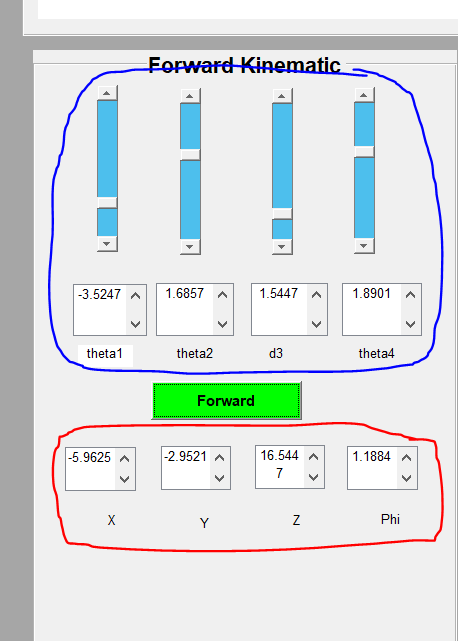
Nguyễn Minh Thảo 1814048

**Email**: thao.nguyen\_13032000@hcmut.edu.vn

1. **Giao diện của GUI:**



Hình 1: Giao diện đồ họa của GUI SCARA Robot Simulation

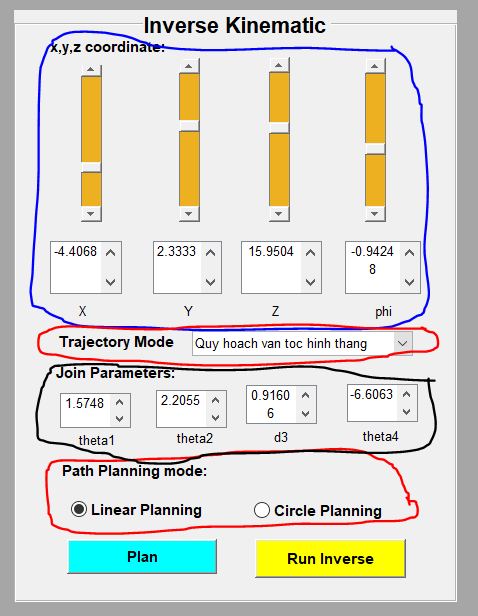


Hình 2: Khu vực thực hiện Forward Kinematic

Khu vực khoanh tròn màu xanh dương là để cài đặt giá trị các biến theta1, theta2, d3, theta4

Khu vực khoanh tròn màu đỏ là hiển thị tọa độ và góc phi của End Effecter sau khi thực hiện Forward Kinematic

Nhấn nút Forward màu xanh để chạy Forward Kinematic khi đã cài đặt xong các giá trị



Hình 3: Khu vực thực hiện Inverse Kinematic

Khoanh tròn màu xanh là khu vực cài đặt tọa độ mong muốn EF di chuyển tới

Khoanh trong màu đỏ là khu vực cài đặt chế độ hoạt động của robot:

* Quy hoặc vận tốc: Cửa sổ sổ xuống gồm

+ Quy hoạch vận tốc hình thang

+ Quy hoạch vận tốc hình S curve

* Quy hoặc quỹ đạo: Click chọn

+ Linear Planning: Quy hoạch quỹ đạo đường thẳng

+ Circle Planning: Quy hoạch quỹ đạo hình tròn

Sau khi cài đặt xong ấn Plan để thực hiện Quy hoạch

Ấn Run Inverse để thực hiện chạy Inverse Kinematic:

* Có thể chạy Run Inverse với giá trị tính toán hoặc giá trị sau khi thực hiện điều khiển PID

1. **Forward Kinematic:**
2. **Hàm nhập giá trị theta1 từ editbox của GUI:**

function editTheta1\_Callback(hObject, eventdata, handles)

global theta1;

theta1 = str2num(get(handles.editTheta1,'string'));

if (theta1 > 6.283185)

theta1 = 6.283185;

elseif (theta1 < -6.283185)

theta1 = -6.283185;

end

set(handles.sliderTheta1,'value',theta1);

* Giải thích:
* Giá trị các góc theta em giới hạn trong đoạn từ [-2\*pi, 2\*pi] để có thể quay được cả 2 chiều: chiều âm và chiều dương
* Vì vậy các giá trị nào > 6.283185 (> 2\*pi) hoặc < -6.283185 (<-2\*pi) thì sẽ tự động được cài đặt là giá trị giới hạn (2\*pi và -2\*pi)

1. **Các hàm nhập giá trị theta khác tương tự**
2. **Hàm nhập giá trị d3:**

function editD3\_Callback(hObject, eventdata, handles)

global d3;

d3 = str2num(get(handles.editD3,'string'));

if (d3 > 20)

d3 = 20;

elseif (d3 < 0)

d3 = 0;

end

set(handles.sliderD3,'value',d3);

* Giải thích:
* Em đặt giá trị của d3 giới hạn trong đoạn [0,20]

1. **Hàm thực hiện Forward Kinematic và hiển thị lên cửa sổ Gui**

function pushbuttonForward\_Callback(hObject, eventdata, handles)

global theta1;

global theta2;

global d3;

global theta4;

global tmp1;

global tmp2;

global tmp3;

global tmp4;

[T01,T02,T03,T04] = EF\_HomoTransform(theta1,theta2,d3,theta4);

EF = vedothi3d(theta1,theta2,d3,theta4);

tmp1 = theta1;

tmp2 = theta2;

tmp3 = d3;

tmp4 = theta4;

set(handles.editX,'string',num2str(EF(1)));

set(handles.editY,'string',num2str(EF(2)));

set(handles.editZ,'string',num2str(EF(3)));

phi = atan((EF(3)/sqrt(EF(1)^2 + EF(2)^2)));

set(handles.editPhi,'string',num2str(phi));

* Giải thích:
* Các biến **theta1,theta2,d3,theta4** là các biến **global** để các hàm khác trong Gui đều có thể sử dụng được
* Các biến **tmp1, tmp2, tmp3, tmp4** là giá trị ban đầu của **theta1, theta2, d3, theta4**
* Sau khi thực hiện Forward Kinematic thì các giá trị tọa độ mới sẽ được in lên edit box của Gui:

set(handles.editX,'string',num2str(EF(1)));

set(handles.editY,'string',num2str(EF(2)));

set(handles.editZ,'string',num2str(EF(3)));

phi = atan((EF(3)/sqrt(EF(1)^2 + EF(2)^2)));

set(handles.editPhi,'string',num2str(phi));

1. **Hàm [T01,T02,T03,T04] = EF\_HomoTransform(t1,t2,d3,t4):**

function [T01,T02,T03,T04] = EF\_HomoTransform(t1,t2,d3,t4)

a1 = 5;

a2 = 5;

d1 = 10;

d4 = 5;

T01 = [cos(t1) -sin(t1) 0 a1\*cos(t1); sin(t1) cos(t1) 0 a1\*sin(t1);0 0 1 d1;0 0 0 1];

T12 = [cos(t2) -sin(t2) 0 a2\*cos(t2); sin(t2) cos(t2) 0 a2\*sin(t2);0 0 1 0;0 0 0 1];

T23 = [1 0 0 0; 0 1 0 0; 0 0 1 d3; 0 0 0 1];

T34 = [cos(t4) sin(t4) 0 0; sin(t4) -cos(t4) 0 0; 0 0 -1 d4; 0 0 0 1];

T02 = T01\*T12;

T03 = T01\*T12\*T23;

T04 = T01\*T12\*T23\*T34;

* Giải thích:
* Hàm này tính ma trận Homophology Transform của robot SCARA dựa vào các biến **theta1, theta2, d3, theta4** đưa vào. Kết quả trả về lần lượt là các ma trận Homo Transform ứng với các Join tương ứng

1. Thực hiện vẽ realtime chuyển động của robot trong không gian 3D lên cửa sổ GUI

function [EF] = vedothi3d(theta1,theta2,d3,theta4)

1. **Inverse Kinematic**
2. **Hàm tính Inverse Kinematic**:

function [theta1,theta2,d3,theta4,mark] = Inverse\_kinematic(q\_x,q\_y,q\_z,dir,pre\_mark,q\_phi)

% - dir = [x1,x2]; x = 1:chiều dương, x = 0:chiều âm; x1 là chiều của theta1,x2 là chiều của theta2; dir = [3,3]: Điểm đầu tiên trong quỹ đạo => pre\_mark = [3,3]

% - pre\_mark = [x1,x2]; x1 = 1: theta1\_pre có dấu dương, x1 = 0: theta1\_pre mang dấu âm; x1 = 3: theta1 là điểm đầu tiên trong quỹ đạo. Tương tự với theta2

d3 = q\_z - 15;

c2 = (q\_x^2 + q\_y^2 - 2\*25)/(2\*25);

s2 = +sqrt(1-c2^2);

theta2 = atan2(s2,c2);

s1 = ((5 + 5\*c2)\*q\_y - 5\*s2\*q\_x)/(q\_x^2+q\_y^2);

c1 = ((5 + 5\*c2)\*q\_x + 5\*s2\*q\_y)/(q\_x^2+q\_y^2);

theta1 = atan2(s1,c1);

% - Xử lí chiều quay của theta1, tránh tình trạng atan2(theta1 > pi)<0 hoặc atan2(theta1<-pi) > 0 dẫn đến tình trạng theta1 tăng vọt đột ngột không điều khiển được PID

if ((dir(1) == 1)||(dir(1) == 3))&&(pre\_mark(1) == 1)&&(theta1 < -0.04)

theta1 = theta1 + 2\*pi;

elseif ((dir(1) == 0)||(dir(1) == 3))&&(pre\_mark(1) == 0)&&(theta1 > 0.04)

theta1 = theta1 - 2\*pi;

elseif (dir(1) == 3)&&(theta1 == pi)&&(pre\_mark(1) == 3)

theta1 = -pi;

end

if theta1 >= 0

mark(1) = 1;

else

mark(1) = 0;

end

if theta2 >= 0

mark(2) = 1;

else

mark(2) = 0;

end

theta4 = q\_phi - theta1 - theta2;

* Giải thích:
* Do em cho **theta1** có thể quay cả theo chiều âm và chiều dương nên theta1 có thể âm và dương, dẫn đến các trường hợp là **theta1 > pi** hoặc **theta1 < -pi.** Nhưng hàm **atan2** sẽ trả về các giá trị góc âm nếu tọa độ (sin(t), cos(t)) nằm ở nửa dưới đường tròn lượng giác và giá trị góc dương nếu (sin(t),cos(t)) nằm ở nửa trên đường tròn lượng giác:
* Nếu **theta1 đang dương** và **theta1 tiếp theo** lớn hơn pi **(theta1>pi)** thì hàm atan2 sẽ trả về **theta1 < 0.** Điều này làm cho **theta1** từ dương đột ngột chuyển sang âm khiến cho bộ điều khiển PID không thể điều khiển được
* Nếu **theta1 đang âm** và **theta1 tiếp theo** nhỏ hơn -pi thì điều ngược lại xảy ra
* Để giải quyết vấn đề trên em sử dụng 2 cặp biến **dir = [x1, x2]** để biểu thị hướng chuyển động của quỹ đạo theta1, theta2 và **pre\_mark = [x1, x2]** để biểu thị dấu của giá trị góc **theta1, theta2** liền trước đó

+ **xi = 3** nếu đây là điểm đầu tiên của quỹ đạo ( và cả điểm thứ 2 đối với biến **dir**)

+ **xi = 1:**

* + - Nếu quỹ đạo đang quay theo chiều dương ( ngược chiều kim đồng hồ) với biến **dir**
    - Nếu **theta** trước đó mang dấu dương với biến **pre\_mark**

**+ Ngược lại xi = 0**

* Với góc **theta2** để tránh bài toán trở nên phức tạp và nhiều trường hợp đặc biệt phát sinh nên em giới hạn **theta2** chỉ quay theo chiều dương và giới hạn trong đoạn [0, pi]:

s2 = +sqrt(1-c2^2);

theta2 = atan2(s2,c2);

* Để giải quyết trường hợp **theta1** quay theo **chiều dương** và từ **theta1 < pi** chuyển sang **theta1 > pi**, em sử dụng điều kiện if sau:

if ((dir(1) == 1)||(dir(1) == 3))&&(pre\_mark(1) == 1)&&(theta1 < -0.04)

theta1 = theta1 + 2\*pi;

**+** Trong đó **((dir(1) == 1)||(dir(1) == 3))** là 1 trong 2 trường hợp hoặc theta1 đang quay theo chiều dương và chuyển từ (**theta1< pi**) sang (**theta1 > pi**) hoặc từ điểm đầu tiên (**theta1< pi**) chuyển sang điểm thứ 2 (**theta1> pi**) kèm với điều kiện điểm trước đó **theta1** mang dấu dương **&&(pre\_mark(1) == 1)**. Điều kiện **&&(theta1 < -0.04)** vì em nhận thấy tại một vài điểm đầu tiên khi quy hoạch quỹ đạo đường tròn, **theta1** có xu hướng giảm xuống một vài giá trị âm rất nhỏ sau đó tăng lên giá trị dương (hoặc ngược lại) vì thế em sử dụng **theta1 < -0.04** thay vì **theta1 < 0.** Nếu tất cả điều kiện trên thỏa mãn thì **theta1** trả ra từ hàm **atan2** sẽ được cộng thêm **2\*pi** để chuyển sang giá trị dương (**theta1 > pi**) mà không làm thay đổi tọa độ của quỹ đạo.

* Trường hợp **theta1** quay theo chiều âm và chuyển từ (**theta1 > -pi**) sang (**theta1 < -pi**) được xử lí bằng điều kiện sau:

elseif ((dir(1) == 0)||(dir(1) == 3))&&(pre\_mark(1) == 0)&&(theta1 > 0.04)

theta1 = theta1 - 2\*pi;

+ Giải thích tương tự như trường hợp trên

* Trường hợp điểm đầu tiên của quỹ đạo là **theta1 = pi** sẽ được chuyển sang **theta1 = -pi** để tiện cho việc xử lí quỹ đạo hơn:

elseif (dir(1) == 3)&&(theta1 == pi)&&(pre\_mark(1) == 3)

theta1 = -pi;

* Sau đó sẽ trả về các giá trị **theta1, theta2, d3, theta4** và cặp biến **mark** biểu thị dấu của **theta1** và **theta2**

1. **Quy hoạch vận tốc:**

function [delta\_q] = velocity\_trajectory(q\_max,amax,v\_max)

* Các biến điều khiển:

global TrajectoryMode

* Chọn Mode quy hoạch vận tốc 1 là quy hoạch vận tốc hình thang, 2 là quy hoạch vận tốc hình S curve

global run\_enable;

* **Run\_enable = [x1, x2]: xi = 0** thì xảy ra lỗi và không cho phép chạy, **xi = 1** thì không xảy ra lỗi và được phép chạy. **x1 = 0** khi xảy ra workspace Singularities hoặc Kinematic Singularities ( Kinematic Singularities được phép chạy mô phỏng nếu được sự cho phép của người sử dụng). **x2 = 0** khi xảy ra vận tốc bị suy biến
* Giải thích hoạt động:
* Hàm này nhận vào các giá trị đặt **q\_max** ( là độ dài dịch chuyển của quỹ đạo), **amax, v\_max** và trả ra **ma trận delta\_q** là các vi phân độ dài đại số quãng đường dịch chuyển liên tiếp nhau ( có thể âm hoặc dương). Từ vi phân quãng đường dịch chuyển này có thể tính ra tọa độ của quỹ đạo trong không gian 3D.
* Hàm này sẽ trả về cảnh báo và không cho phép chạy nếu có lỗi xảy ra. Nếu không có lỗi xảy ra thì có thể thực hiện quy hoạch vận tốc hình thang hoặc hình S curve

1. **Quy hoạch quỹ đạo**
2. **Quy hoạch quỹ đạo đường thẳng:**

function [q\_x,q\_y,q\_z,q\_phi] = linear\_path\_planning(amax,v\_max)

* Giải thích hoạt động:
* Hàm này nhận vào giá trị đặt amax, v\_max = vmax\_set. Sau đó hàm kiểm tra điều kiện **run\_enable**, nếu cho phép chạy thì thực hiện quy hoạch quỹ đạo đường thẳng. Hàm truyền v**\_max, amax,q\_max** cho hàm velocity\_trajectory(q\_max,amax,v\_max)để thực hiện quy hoạch vận tốc, **q\_max** chính là độ dài của quỹ đạo
* Các biến **x,y,z,phi** là các biến **global** chứa tọa độ EF mà người dùng muốn di chuyển đến
* Các biến **tmp\_x,tmp\_y,tmp\_z,tmp\_phi** là tọa độ hiện tại của EF
* Hàm này tính vecto quỹ đạo dịch chuyển của EF, sau đó tính hệ số góc của vector này đến 3 trục tọa độ Ox, Oy, Oz. Vector quỹ đạo dịch chuyển này lần lượt được nhân với các giá trị vi phân quãng đường dịch chuyển sẽ ra vector vi phân quỹ đạo dịch chuyển, sau đó nhân với các hệ s
* ố góc sẽ ra quỹ đạo dịch chuyển trên 3 trục tọa độ
* Hàm trả về các ma trận **q\_x, q\_y, q\_z, q\_phi** là tọa độ quỹ đạo trên 3 trục tọa độ và quỹ đạo góc phi

1. **Quy hoạch quỹ đạo nửa hình tròn:**

function [q\_x,q\_y,q\_z,q\_phi] = circle\_path\_planning(amax,v\_max)

* Giải thích hoạt động:
* Hàm này nhận vào amax, v\_max người dùng cài đặt, sau đó thực hiện quy hoạch quỹ đạo hình tròn và trả về các ma trận tọa độ quỹ đạo đường tròn của EF trên 3 trục tọa độ và quỹ đạo góc phi.
* Hàm truyền amax, v\_max, q\_max (chính là nửa chu vi đường tròn) cho hàm velocity\_trajectory(q\_max,amax,v\_max)để thực hiện quy hoạch vận tốc.
* Từ vi phân độ dài quỹ đạo tính được vi phân góc t trong phương trình đường tròn, từ đó tính được tọa độ của quỹ đạo đường tròn
* Chi tiết em có chú thích trong code ạ.

1. **Differential Kinematic:**
2. **Hàm tính Differential Kinematic**

function [singular,warning,theta1\_dot,theta2\_dot,d3\_dot,theta4\_dot,index] = differential\_kinematic(q\_x,q\_y,q\_z,q\_phi)

* Giải thích:
* Hàm này nhận vào ma trận quỹ đạo q\_x,q\_y,q\_z và thực hiện tính toán vận tốc các Joins dựa vào Differential Kinematic và kiểm tra liệu điểm đó có bị Kinematic Singularities hay không, nếu có thì trả ra tọa độ các điểm Kinematic Singularities trong biến Singular, số lượng các điểm singular và chỉ số của các điểm đó trong các ma trận q\_x,q\_y,q\_z
* Chi tiết em có chú thích trong code ạ

1. **Hàm tính ma trận Jacobian**

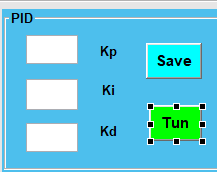
function [Jacobian] = Jacobian(T01,T02,T03,T04)

* Giải thích:
* Hàm này nhận và tham số là các ma trân Homophology transform của các Join, trả ra ma trận Jacobian 6x4

1. **Điều khiển PID vị trí của các Join:**
2. **Hàm cài đặt và tinh chỉnh các giá trị Kp, Ki, Kd**

function pushbuttonPID\_Tun\_Callback(hObject, eventdata, handles)

* Giải thích hoạt động:
* Hàm này sẽ được gọi khi người dùng ấn nút Tun trong khu vực PID của GUI:



* Hàm này sẽ thực hiện mô phỏng đáp ứng của động cơ DC (giả sử là động cơ của các khớp robot) với giá trị đặt là hàm nấc yd = 3 và hiển thị lên axes2 của GUI. Từ đó người dùng có thể tinh chỉnh giá trị **Kp, Ki, Kd** sau đó lưu lại bằng nút Save

1. **Hàm lưu các giá trị PID:**

function pushbuttonPID\_SaveVal\_Callback(hObject, eventdata, handles)

* Giải thích:
* Hàm này sẽ được gọi khi người dùng ấn nút Save của ô PID
* Hàm này thực hiện lưu các giá trị Kp, Ki, Kd đã tinh chỉnh, chạy đáp ứng PID của các khớp, EF và hiển thị lên đồ thị. Đồng thời hàm này còn chạy Differential Kinematic để tính toán vận tốc điều khiển PID của các khớp và hiển thị lên đồ thị
* Sau khi ấn nút Save, chương trình sẽ vẽ mô phỏng của robot SCARA với các giá trị đã được điều khiển PID
* Sau khi ấn nút Save, biến **PID\_Ready** sẽ bằng 1, chương trình sẵn sàng để chạy điều khiển PID
* File Simulink mô phỏng hệ thống điều khiển PID động cơ DC:

PID\_Tun.slx

1. **Hàm vẽ vận tốc các Join lên đồ thị**

plot\_Joins\_velocities(theta1\_dot,theta2\_dot,d3\_dot,theta4\_dot,time,n,color)