Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey Campus

Puebla

Fundamentación de robótica (Gpo 101)

ACTIVIDAD

Actividad 1 (Velocidades Lineales y angulares) Robot Planar 3GDL

Elián Cantalapiedra Sabugal | A01738462

El programa original modela la cinemática directa y el Jacobiano de un robot planar de 2 grados de libertad usando matrices de transformación homogénea.
El objetivo de la conversión fue extender el modelo a 3 grados de libertad (RRR) manteniendo la estructura del código y su funcionamiento en base al ciclo

```
%Limpieza de pantalla
clear all
close all
clc
```

Se limpian variables, figuras y consola para evitar interferencias con ejecuciones anteriores.

```
%Declaración de variables simbólicas
syms th1(t) th2(t) th3(t) t l1 l2 l3

%Configuración del robot, 0 para junta rotacional, 1 para junta prismática
RP=[0 0 0];

%Creamos el vector de coordenadas articulares
Q= [th1, th2, th3];
disp('Coordenadas generalizadas');
pretty (Q);

%Creamos el vector de velocidades generalizadas
Qp= diff(Q, t);
disp('Velocidades generalizadas');
pretty (Qp);
```

Ahora el sistema tiene tres articulaciones rotacionales, pasando de 2 gdl a 3 gdl
Se añadió un tercer elemento 0, indicando que el robot ahora es RRR).
El sistema ahora tiene 3 variables independientes, el Jacobiano será 3×3, las velocidades articulares son tres

```
%Número de grado de libertad del robot
GDL= size(RP,2);
GDL_str= num2str(GDL);
```

```
%% Junta 1
P(:,:,1)= [l1*cos(th1); l1*sin(th1);0];
R(:,:,1)= [cos(th1) -sin(th1)  0;
           sin(th1)  cos(th1)  0;
           0         0         1];
```

```
%% Junta 2
P(:,:,2)= [l2*cos(th2); l2*sin(th2);0];
R(:,:,2)= [cos(th2) -sin(th2)  0;
           sin(th2)  cos(th2)  0;
           0         0         1];
```

```
%% Junta 3
P(:,:,3)= [l3*cos(th3); l3*sin(th3);0];
R(:,:,3)= [cos(th3) -sin(th3)  0;
           sin(th3)  cos(th3)  0;
           0         0         1];
```

El código es escalable con el bucle principal porque el número de gdl se calcula automáticamente a partir del vector RP, en este caso 3.

Se añade una junta más, con la posición del tercer eslabón y su matriz de rotación alrededor del eje Z

```
            0           0           1];

%Vector de ceros
Vector_Zeros= zeros(1, 3);

%Inicialización
A(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
T(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
PO(:,:,GDL)= P(:,:,GDL);
RO(:,:,GDL)= R(:,:,GDL);
RO_inv(:,:,GDL)= R(:,:,GDL);

for i = 1:GDL
    i_str= num2str(i);

    %Locales
    disp(strcat('Matriz de Transformación local A', i_str));
    A(:,:,i)=simplify([R(:,:,i) P(:,:,i); Vector_Zeros 1]);
    pretty (A(:,:,i));

    %Globales
    try
       T(:,:,i)= T(:,:,i-1)*A(:,:,i);
    catch
       T(:,:,i)= A(:,:,i);
    end
```

Se realiza la inicialización y construcción de las matrices de transformación homogénea, tanto locales como globales, tomando en cuenta los GDL, en el bucle

```
    disp(strcat('Matriz de Transformación global T', i_str));
    T(:,:,i)= simplify(T(:,:,i));
    pretty(T(:,:,i))

    RO(:,:,i)= T(1:3,1:3,i);
    RO_inv(:,:,i)= transpose(RO(:,:,i));
    PO(:,:,i)= T(1:3,4,i);
end

%Calculamos la matriz de transformación del marco de referencia inercial
%visto desde el actuador final
% disp(strcat('Matriz de Transformación T', GDL_str,'_O calculada de forma manual'));
% RF_O=RO_inv(:,:,GDL);
% PF_O=-RF_O*PO(:,:,GDL);
% TF_O= simplify([RF_O PF_O; Vector_Zeros 1]);
% pretty(TF_O);

%disp(strcat('Matriz de Transformación T', GDL_str,'_O calculada de forma auntomática'));
%pretty(simplify(inv(T(:,:,GDL))));


%Calculamos el jacobiano lineal de forma diferencial
disp('Jacobiano lineal obtenido de forma diferencial');
```

Se muestra en pantalla la matriz de transformación global, se aplica simplify() para reducir la expresión simbólica, se imprime en formato matemático legible mediante pretty()

```
%Derivadas parciales de x respecto a th1, th2 y th3
Jv11= functionalDerivative(PO(1,1,GDL), th1);
Jv12= functionalDerivative(PO(1,1,GDL), th2);
Jv13= functionalDerivative(PO(1,1,GDL), th3);

%Derivadas parciales de y respecto a th1, th2 y th3
Jv21= functionalDerivative(PO(2,1,GDL), th1);
Jv22= functionalDerivative(PO(2,1,GDL), th2);
Jv23= functionalDerivative(PO(2,1,GDL), th3);

%Derivadas parciales de z respecto a th1, th2 y th3
Jv31= functionalDerivative(PO(3,1,GDL), th1);
Jv32= functionalDerivative(PO(3,1,GDL), th2);
Jv33= functionalDerivative(PO(3,1,GDL), th3);

%Creamos la matríz del Jacobiano lineal
jv_d=simplify([Jv11 Jv12 Jv13;
               Jv21 Jv22 Jv23;
               Jv31 Jv32 Jv33]);
pretty(jv_d);
```

En esta sección se obtiene el Jacobiano lineal del manipulador mediante derivadas parciales de la posición del efector final respecto a cada coordenada articular.

Se realizan las derivadas parciales del componente x, y, z.
Ahora el jacobiano pasó de dimensión 3×2 a 3×3.

```
%Calculamos el jacobiano lineal de forma analítica
Jv_a(:,GDL)=PO(:,:,GDL);
Jw_a(:,GDL)=PO(:,:,GDL);

for k= 1:GDL
    if RP(k)==0 %Casos: articulación rotacional

        %Para las juntas de revolución
        try
            Jv_a(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL)-PO(:,:,k-1));
            Jw_a(:,k)= RO(:,3,k-1);
        catch
            Jv_a(:,k)= cross([0;0;1], PO(:,:,GDL));
            Jw_a(:,k)=[0;0;1];
        end

    elseif RP(k)==1 %Casos: articulación prismática

        try
            Jv_a(:,k)= RO(:,3,k-1);
        catch
            Jv_a(:,k)=[0;0;1];
        end

        Jw_a(:,k)=[0;0;0];
    end
end

Jv_a= simplify (Jv_a);
Jw_a= simplify (Jw_a);
```

```
disp('Jacobiano lineal obtenido de forma analítica');
pretty (Jv_a);

disp('Jacobiano ángular obtenido de forma analítica');
pretty (Jw_a);


disp('Velocidad lineal obtenida mediante el Jacobiano lineal');
V=simplify (Jv_a*Qp');
pretty(V);

disp('Velocidad angular obtenida mediante el Jacobiano angular');
W=simplify (Jw_a*Qp');
pretty(W);
```

Muestra en pantalla los resultados del Jacobiano y calcula las velocidades del efector final usando las velocidades articulares.