

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

%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];
```

Al tratarse de un robot con 3 grados de libertad, y los 3 girando al rededor del eje z, se tratan de juntas rotacionales, por lo que se asignan tres 0 dentro del vector, uno por cada grado de libertad.

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

Coordenadas generalizadas

```
pretty (Q);
```

(th1(t), th2(t), th3(t))

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

Velocidades generalizadas

```
pretty(Qp);
```

$$\begin{pmatrix} \frac{d}{dt} \text{th1}(t) \\ \frac{d}{dt} \text{th2}(t) \\ \frac{d}{dt} \text{th3}(t) \end{pmatrix}$$

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

```
GDL =
3
```

```
%Vector de traslación de la articulación 1 respecto a 0
P(:,:,1)= [l1*cos(th1);
            l1*sin(th1);
            0];
%Matriz de rotación de la junta 1 respecto a 0
R(:,:,1)= [cos(th1) -sin(th1) 0;
```

```

sin(th1)  cos(th1) 0;
0          0          1];

```

```

%Vector de traslación de la articulación 2 respecto a 0
P(:,:,2)= [12*cos(th2);
            12*sin(th2);
            0];
%Matriz de rotación de la junta 2 respecto a 0
R(:,:,2)= [cos(th2) -sin(th2) 0;
            sin(th2)  cos(th2) 0;
            0          0          1];

```

```

%Vector de traslación de la articulación 3 respecto a 0
P(:,:,3)= [13*cos(th3);
            13*sin(th3);
            0];
%Matriz de rotación de la junta 3 respecto a 0
R(:,:,3)= [cos(th3) -sin(th3) 0;
            sin(th3)  cos(th3) 0;
            0          0          1];

```

El código anterior es el encargado de hacer cada matriz de rotación y cada vector de traslación para los respectivos grados de libertad. Estos son dos de los 4 elementos de la matriz homogénea, que se hará 1 para cada grado de libertad e igualmente resultará una global al multiplicarse.

```

%Creamos un vector de ceros
Vector_Zeros = zeros(1,3);

```

Igualmente se crea un vector de tres 0, pues es parte de los elemnts de la matriz homogénea.

```

%Inicializamos las matrices de transformación Homogénea locales
A(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
%Inicializamos las matrices de transformación Homogénea globales
T(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
%Inicializamos las posiciones vistas desde el marco de referencia inercial
PO(:,:,GDL)= P(:,:,GDL);
%Inicializamos las matrices de rotación vistas desde el marco de referencia inercial
RO(:,:,GDL)= R(:,:,GDL);
%Inicializamos las INVERSAS de las matrices de rotación vistas desde el marco de referencia inercial
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
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);
%pretty(RO(:,:,i));
%pretty(RO_inv(:,:,i));
%pretty(PO(:,:,i));
end

```

```

Matriz de Transformación local A1
/ cos(th1(t)), -sin(th1(t)), 0, 11 cos(th1(t)) \
| sin(th1(t)),  cos(th1(t)), 0, 11 sin(th1(t)) |
| 0,          0,          1,          0           |
\ 0,          0,          0,          1           /
Matriz de Transformación global T1
/ cos(th1(t)), -sin(th1(t)), 0, 11 cos(th1(t)) \
| sin(th1(t)),  cos(th1(t)), 0, 11 sin(th1(t)) |
| 0,          0,          1,          0           |
\ 0,          0,          0,          1           /
Matriz de Transformación local A2
/ cos(th2(t)), -sin(th2(t)), 0, 12 cos(th2(t)) \
| sin(th2(t)),  cos(th2(t)), 0, 12 sin(th2(t)) |
| 0,          0,          1,          0           |
\ 0,          0,          0,          1           /
Matriz de Transformación global T2
/ #2, -#1, 0, 11 cos(th1(t)) + 12 #2 \
| #1,  #2, 0, 11 sin(th1(t)) + 12 #1 |
| 0,  0,  1,          0           |
\ 0,  0,  0,          1           /

```

where

```

#1 == sin(th1(t) + th2(t))
#2 == cos(th1(t) + th2(t))
Matriz de Transformación local A3
/ cos(th3(t)), -sin(th3(t)), 0, 13 cos(th3(t)) \
| sin(th3(t)),  cos(th3(t)), 0, 13 sin(th3(t)) |
|   0,          0,          1,          0           |
\   0,          0,          0,          1           /
Matriz de Transformación global T3
/ #2, -#1, 0, 11 cos(th1(t)) + 13 #2 + 12 cos(th1(t) + th2(t)) \
| #1,  #2, 0, 11 sin(th1(t)) + 13 #1 + 12 sin(th1(t) + th2(t)) |
|   0,  0,  1,          0           |
\   0,  0,  0,          1           /

```

where

```

#1 == sin(th1(t) + th2(t) + th3(t))
#2 == cos(th1(t) + th2(t) + th3(t))

```

El código anterior se encarga de las matrices de transformación locales y globales. Las locales se refieren al movimiento entre dos "secciones" consecutivas del robot, pero de un sol grado de libertad. Por otro lado, las matrices locales se refieren al movimiento de alguna "sección" del robot respecto a la base, es por ello que la matriz global 3 es la de mayor tamaño, pues ya es la multiplicación de las anteriores y ya tiene en cuenta los 3 grados de libertad.

```

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

```

Jacobiano lineal obtenido de forma diferencial

```

%Diferivadas parciales de x respecto a th1, th2 y th3
Jv11= functionalDerivative(P0(1,1,SDL), th1);
Jv12= functionalDerivative(P0(1,1,SDL), th2);
Jv13= functionalDerivative(P0(1,1,SDL), th3);

```

$$Jv13(t) = -l_3 \sin(\theta_1(t) + \theta_2(t) + \theta_3(t))$$

```

%Diferivadas parciales de y respecto a th1, th2 y th3
Jv21= functionalDerivative(P0(2,1,SDL), th1);
Jv22= functionalDerivative(P0(2,1,SDL), th2);
Jv23= functionalDerivative(P0(2,1,SDL), th3);
%Diferivadas parciales de z respecto a th1, th2 y th3
Jv31= functionalDerivative(P0(3,1,SDL), th1);
Jv32= functionalDerivative(P0(3,1,SDL), th2);
Jv33= functionalDerivative(P0(3,1,SDL), th3);

```

Igualmente se calcula el jacobiano lineal de forma diferencial, esto quiere decir que se toman derivadas parciales de la posición de un punto respecto a cada coordenada articular. Este análisis se usa para relacionar velocidades articulares con lineales o angulares y en MATLAB se utiliza la función **functionalDerivative**.

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

/ - l1 sin(th1(t)) - #1 - l2 sin(th1(t) + th2(t)), - #1 - l2 sin(th1(t) + th2(t)), -#1 \
|   l1 cos(th1(t)) + #2 + l2 cos(th1(t) + th2(t)),  #2 + l2 cos(th1(t) + th2(t)),  #2 |
|   \                               0,                                0,                                0   /
where

#1 == l3 sin(th1(t) + th2(t) + th3(t))
#2 == l3 cos(th1(t) + th2(t) + th3(t))
```

```
%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(:,:,k-1), PO(:,:,GDL)-PO(:,:,k-1));
            Jw_a(:,k)= RO(:,:,k-1);
        catch
            Jv_a(:,k)= cross([0,0,1], PO(:,:,GDL));
            Jw_a(:,k)=[0,0,1];
        end
    end
```

El código anterior es el encargado de calcular el jacobiano lineal analítico, el cual se basa en la geometría y en la fórmula: $Jv_i = Z_{i-1} \cdot (O_n - O_{i-1})$ la cual corresponde a una junta rotacional, que fue la que se simuló en este caso. Esta junta rotacional a su vez produce una velocidad angular, que se relaciona con la velocidad lineal:

$$\nu = \omega \cdot r$$

```
%Para las juntas prismáticas
elseif RP(k)==1 %Casos: articulación prismática
%
try
    Jv_a(:,k)= RO(:,:,k-1);
catch
```

```

        Jv_a(:,k)=[0,0,1];
    end
    Jw_a(:,k)=[0,0,0];
end
end

```

Igualmente se calcula el jacobiano lineal de forma analítica para las juntas prismáticas, que igualmente se basa en la geometría y en la fórmula: $Jv_i = Z_{i-1}$

```

Jv_a= simplify (Jv_a);
Jw_a= simplify (Jw_a);
disp('Jacobiano lineal obtenido de forma analítica');

```

Jacobiano lineal obtenido de forma analítica

```
pretty (Jv_a);
```

```

/ - l1 sin(th1(t)) - #1 - 12 sin(th1(t) + th2(t)), - #1 - 12 sin(th1(t) + th2(t)), -#1 \
|   l1 cos(th1(t)) + #2 + 12 cos(th1(t) + th2(t)),   #2 + 12 cos(th1(t) + th2(t)),   #2 |
|   \                               0,                                0,                                0   /

```

where

```

#1 == 13 sin(th1(t) + th2(t) + th3(t))
#2 == 13 cos(th1(t) + th2(t) + th3(t))

```

```
disp('Jacobiano ángular obtenido de forma analítica');
```

Jacobiano ángular obtenido de forma analítica

```
pretty (Jw_a);
```

```

/ 0, 0, 0 \
| 0, 0, 0 |
| 1, 1, 1 /

```

```
disp('Velocidad lineal obtenida mediante el Jacobiano lineal');
```

Velocidad lineal obtenida mediante el Jacobiano lineal

```

V=simplify (Jv_a*Qp');
pretty(V);

```

```

/ - #4 (13 sin(#1) + 12 sin(#2)) - #5 (l1 sin(th1(t)) + 13 sin(#1) + 12 sin(#2)) - 13 #3 sin(#1) \
|   #4 (13 cos(#1) + 12 cos(#2)) + #5 (l1 cos(th1(t)) + 13 cos(#1) + 12 cos(#2)) + 13 #3 cos(#1) |
|   \                               0

```

where

```

#1 == th1(t) + th2(t) + th3(t)

#2 == th1(t) + th2(t)

_____
d
#3 == -- th3(t)
dt

_____
d
#4 == -- th2(t)
dt

_____
d
#5 == -- th1(t)
dt

disp('Velocidad angular obtenida mediante el Jacobiano angular');

```

Velocidad angular obtenida mediante el Jacobiano angular

```

W=simplify (Jw_a*Qp');
pretty(W);

```

$$\begin{pmatrix} \theta & 0 & 0 \\ 0 & \theta & 0 \\ \frac{d}{dt} \text{th1}(t) + \frac{d}{dt} \text{th2}(t) + \frac{d}{dt} \text{th3}(t) & \frac{d}{dt} \text{th1}(t) & \frac{d}{dt} \text{th2}(t) \end{pmatrix}$$

Finalmente, obtenemos una matriz de velocidad lineal y otra de velocidad angular. La primera tiene valores en x y y , no tiene en z pues como se mencionó al inicio, los giros que se hacen son en el eje z , por lo que no existe un movimiento lineal en ese eje. Como se puede observar, el resultado obtenido para x y y es la suma de los giros de cada una de las articulaciones o grados de libertad, pues el primer grado de libertad afecta el siguiente y así sucesivamente. Finalmente, en el eje z tenemos 0, pues el giro se produce en ese eje y no hay velocidad lineal en ese movimiento.

Por el otro lado, en la matriz de velocidad angular se puede observar que para los ejes x y y , el valor es igual a 0, pues no se producen movimientos rotatorios, al contrario del eje z , donde sí hay y es por eso que obtenemos la derivada de la orientación del robot, es decir, su velocidad angular.