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

INICIALIZAMOS VARIABLES PARA AMBOS ROBOTS

```
%Declaración de variables simbólicas
syms th1(t) th2(t) t l1 l2 q1(t) q2(t) q3(t) l3

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

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

```
Coordenadas generalizadas
```

```
pretty (Q);
```

```
(th1(t), th2(t))
```

```


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

```
Velocidades generalizadas
```

```
pretty (Qp);
```

```
/  d           d         \
| -- th1(t), -- th2(t) |
\ dt          dt         /
```

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

%Junta 1
%Posición de la junta 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];



%Junta 2
%Posición de la junta 1 respecto a 0
P(:,:,2)= [l2*cos(th2); l2*sin(th2);0];
%Matriz de rotación de la junta 1 respecto a 0
```

```
R(:,:,2)= [cos(th2) -sin(th2)  0;
           sin(th2)  cos(th2)  0;
           0         0         1];

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

%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);
    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, l1 cos(th1(t)) \
|                                               |
| sin(th1(t)),  cos(th1(t)), 0, l1 sin(th1(t)) |
|                                               |
|      0,            0,      1,        0       |
|                                               |
\      0,            0,      0,        1       /
```

```
Matriz de Transformación global T1
/ cos(th1(t)), -sin(th1(t)), 0, l1 cos(th1(t)) \
|                                              |
| sin(th1(t)),  cos(th1(t)), 0, l1 sin(th1(t)) |
|                                              |
|      0,            0,      1,        0       |
|                                              |
\      0,            0,      0,        1       /
Matriz de Transformación local A2
/ cos(th2(t)), -sin(th2(t)), 0, l2 cos(th2(t)) \
|                                              |
| sin(th2(t)),  cos(th2(t)), 0, l2 sin(th2(t)) |
|                                              |
|      0,            0,      1,        0       |
|                                              |
\      0,            0,      0,        1       /
Matriz de Transformación global T2
/ #2, -#1, 0, l1 cos(th1(t)) + l2 #2 \
|                                    |
| #1,  #2, 0, l1 sin(th1(t)) + l2 #1 |
|                                    |
|  0,  0,  1,            0           |
|                                    |
\  0,  0,  0,            1           /

where

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

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

EN ESTE CASO, LAS LOCALES SE EXPRESAN DE LA MISMA MANERA QUE LAS TRANSFORMACIONES HOMOGENEAS PARA EL PÈNDULO-ROBOT

ADEMÁS, PODEMOS VER COMO LAS ROTACIONES SON EN Z YA QUE EL ROBOT SE MUEVE EN X Y Y, POR LO QUE LA TRANSFORMACIÒN GLOBAL, ES UNA MULTIPLICACIÒN SENCILLA Y CONSERVA UNA ESTRUCTURA SIMILAR A LAS LOCALES

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

```
Jacobiano lineal obtenido de forma diferencial
```

```
%Derivadas parciales de x respecto a th1 y th2
Jv11= functionalDerivative(PO(1,1,GDL), th1);
Jv12= functionalDerivative(PO(1,1,GDL), th2);
%Derivadas parciales de y respecto a th1 y th2
Jv21= functionalDerivative(PO(2,1,GDL), th1);
Jv22= functionalDerivative(PO(2,1,GDL), th2);
%Derivadas parciales de z respecto a th1 y th2
Jv31= functionalDerivative(PO(3,1,GDL), th1);
Jv32= functionalDerivative(PO(3,1,GDL), th2);

%Creamos la matríz del Jacobiano lineal
```

```
jv_d=simplify([Jv11 Jv12;
              Jv21 Jv22;
              Jv31 Jv32]);
pretty(jv_d);
```

```
/ - l1 sin(th1(t)) - l2 sin(th1(t) + th2(t)), -l2 sin(th1(t) + th2(t)) \
|                                                                        |
|  l1 cos(th1(t)) + l2 cos(th1(t) + th2(t)),   l2 cos(th1(t) + th2(t)) |
|                                                                        |
\                     0,                                0               /
```

```
%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

        %Para las juntas prismáticas
     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);
V=simplify (Jv_a*Qp');
W=simplify (Jw_a*Qp');
```

CODIGO DEL ROBOT ANTROPOMORFICO

```
% Configuración del robot antropomórfico, 0 para junta rotacional, 1 para
junta prismática
joint_types_A=[0 0 0]; % Robot antropomórfico

% Creamos el vector de coordenadas articulares para el robot antropomórfico
```

```
Q_A= [q1 q2 q3];
disp('Coordenadas articulares (Robot Antropomórfico)');
```

```
Coordenadas articulares (Robot Antropomórfico)
```

```
pretty (Q_A);
```

```
(q1(t), q2(t), q3(t))
```

```

% Creamos el vector de velocidades articulares para el robot antropomórfico
Qp_A= diff(Q_A, t); % Utilizo diff para derivadas cuya variable de
referencia no depende de otra: ejemplo el tiempo
disp('Velocidades articulares (Robot Antropomórfico)');
```

```
Velocidades articulares (Robot Antropomórfico)
```

```
pretty (Qp_A);
```

```
/  d         d         d       \
| -- q1(t), -- q2(t), -- q3(t) |
\ dt        dt        dt       /
```

```

% Número de grado de libertad del robot antropomórfico
GDL_A= size(joint_types_A,2); % ***Siempre se coloca 3, ya que indica la
dimensión de las columnas
GDL_str_A= num2str(GDL_A); % Convertimos el valor numérico a una cadena de
carácteres tipo string

% Articulación 1 para el robot antropomórfico
% Posición de la junta 1 respecto a 0
P_A(:,:,1)= [0;
             0;
             l1]; % *** Vector de posición indexado por página

% Articulación 2 para el robot antropomórfico
P_A(:,:,2)=[l2*cos(q2);
          l2*sin(q2);
                   0];

% Articulación 3 para el robot antropomórfico
P_A(:,:,3)=[l3*cos(q3);
          l3*sin(q3);
                   0];

% Matriz de rotación de la articulación 1 respecto a 0
R_A(:,:,1)= [cos(q1)  0  sin(q1);
           sin(q1)  0 -cos(q1);
           0         1        0];

R_A(:,:,2)= [cos(q2) -sin(q2) 0;
           sin(q2)  cos(q2) 0;
```

```
            0         0        1];

R_A(:,:,3)= [cos(q3) -sin(q3) 0;
            sin(q3)  cos(q3) 0;
            0         0        1];

% Inicializamos las matrices de transformación Homogénea locales
A_A(:,:,GDL_A)=simplify([R_A(:,:,GDL_A) P_A(:,:,GDL_A); Vector_Zeros 1]);

% Inicializamos las matrices de transformación Homogénea globales
T_A(:,:,GDL_A)=simplify([R_A(:,:,GDL_A) P_A(:,:,GDL_A); Vector_Zeros 1]);

% Inicializamos los vectores de posición vistos desde el marco de referencia
inercial
PO_A(:,:,GDL_A)= P_A(:,:,GDL_A);

% Inicializamos las matrices de rotación vistas desde el marco de referencia
inercial
RO_A(:,:,GDL_A)= R_A(:,:,GDL_A);

for i = 1:GDL_A
    i_str_A= num2str(i);

    % Locales
    A_A(:,:,i) = simplify([R_A(:,:,i) P_A(:,:,i); Vector_Zeros 1]);
    disp(strcat('Matriz de Transformación local A', i_str_A));
    pretty(A_A(:,:,i));  % Mostramos la matriz local A_A

    % Globales
    try
       T_A(:,:,i)= T_A(:,:,i-1)*A_A(:,:,i);
    catch
       T_A(:,:,i)= A_A(:,:,i);  % Caso específico cuando i=1 nos marcaría
error en try
    end
    disp(strcat('Matriz de Transformación global T', i_str_A));
    T_A(:,:,i)= simplify(T_A(:,:,i));
    pretty(T_A(:,:,i));

    % Obtenemos la matriz de rotación "RO" y el vector de translación PO de
la
    % matriz de transformación Homogénea global T_A(:,:,GDL_A)
    RO_A(:,:,i)= T_A(1:3,1:3,i);
    PO_A(:,:,i)= T_A(1:3,4,i);
    pretty(RO_A(:,:,i));
    pretty(PO_A(:,:,i));
end
```

```
Matriz de Transformación local A1
/ cos(q1(t)), 0,  sin(q1(t)),  0 \
|                                |
```

```
| sin(q1(t)), 0, -cos(q1(t)),  0 |
|                                |
|      0,     1,      0,      l1 |
|                                |
\      0,     0,      0,       1 /
Matriz de Transformación global T1
/ cos(q1(t)), 0,  sin(q1(t)),  0 \
|                                |
| sin(q1(t)), 0, -cos(q1(t)),  0 |
|                                |
|      0,     1,      0,      l1 |
|                                |
\      0,     0,      0,       1 /
/ cos(q1(t)), 0,  sin(q1(t)) \
|                            |
| sin(q1(t)), 0, -cos(q1(t)) |
|                            |
\      0,     1,      0      /
/  0 \
|    |
|  0 |
|    |
\ l1 /
Matriz de Transformación local A2
/ cos(q2(t)), -sin(q2(t)), 0, l2 cos(q2(t)) \
|                                           |
| sin(q2(t)),  cos(q2(t)), 0, l2 sin(q2(t)) |
|                                           |
|      0,           0,      1,       0       |
|                                           |
\      0,           0,      0,       1       /
Matriz de Transformación global T2
/ cos(q1(t)) cos(q2(t)), -cos(q1(t)) sin(q2(t)),  sin(q1(t)), l2 cos(q1(t)) cos(q2(t)) \
|                                                                                     |
| cos(q2(t)) sin(q1(t)), -sin(q1(t)) sin(q2(t)), -cos(q1(t)), l2 cos(q2(t)) sin(q1(t)) |
|                                                                                     |
|       sin(q2(t)),            cos(q2(t)),            0,         l1 + l2 sin(q2(t))    |
|                                                                                     |
\           0,                      0,                0,                  1            /
/ cos(q1(t)) cos(q2(t)), -cos(q1(t)) sin(q2(t)),  sin(q1(t)) \
|                                                           |
| cos(q2(t)) sin(q1(t)), -sin(q1(t)) sin(q2(t)), -cos(q1(t)) |
|                                                           |
\       sin(q2(t)),            cos(q2(t)),            0      /
/ l2 cos(q1(t)) cos(q2(t)) \
|                          |
| l2 cos(q2(t)) sin(q1(t)) |
|                          |
\    l1 + l2 sin(q2(t))    /
Matriz de Transformación local A3
/ cos(q3(t)), -sin(q3(t)), 0, l3 cos(q3(t)) \
|                                           |
| sin(q3(t)),  cos(q3(t)), 0, l3 sin(q3(t)) |
|                                           |
|      0,           0,      1,       0       |
|                                           |
\      0,           0,      0,       1       /
Matriz de Transformación global T3
/ cos(q1(t)) cos(#2), -cos(q1(t)) sin(#2),  sin(q1(t)),          cos(q1(t)) #1          \
|                                                                                       |
| sin(q1(t)) cos(#2), -sin(q1(t)) sin(#2), -cos(q1(t)),          sin(q1(t)) #1          |
|                                                                                       |
|       sin(#2),            cos(#2),            0,      l1 + l2 sin(q2(t)) + l3 sin(#2) |
|                                                                                       |
```

```
\          0,                  0,               0,                      1                /

where

   #1 == l2 cos(q2(t)) + l3 cos(#2)

   #2 == q2(t) + q3(t)
/ cos(q1(t)) #2, -cos(q1(t)) #1,  sin(q1(t)) \
|                                            |
| sin(q1(t)) #2, -sin(q1(t)) #1, -cos(q1(t)) |
|                                            |
\       #1,            #2,            0      /

where

   #1 == sin(q2(t) + q3(t))

   #2 == cos(q2(t) + q3(t))
/ cos(q1(t)) (l2 cos(q2(t)) + l3 cos(q2(t) + q3(t))) \
|                                                    |
| sin(q1(t)) (l2 cos(q2(t)) + l3 cos(q2(t) + q3(t))) |
|                                                    |
\     l1 + l2 sin(q2(t)) + l3 sin(q2(t) + q3(t))     /
```

EN ESTE CASA, PODEMOS NOTAR COMO NUEVAMENTE LAS LOCALES SON TRANSFORMACIONES HOMOGENEAS PARA EL PENDULO-ROBOT, SIN EMBARGO, LA H10, ES UNA ROTACIÓN EN Y CON UNA TRANSLACIÓN EN X A 90º, CAMBIANDO LA ESTRUCTURA DE LA TRANSFORMACIÓN GLOBAL, AGREGANDO MÁS COMPONENTES

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

```
Jacobiano lineal obtenido de forma diferencial (Robot Antropomórfico)
```

```
% Derivadas parciales de x respecto a q1
Jv_A11= functionalDerivative(PO_A(1,1,GDL_A), q1);
Jv_A12= functionalDerivative(PO_A(1,1,GDL_A), q2);
Jv_A13= functionalDerivative(PO_A(1,1,GDL_A), q3);
% Derivadas parciales de y respecto a q1
Jv_A21= functionalDerivative(PO_A(2,1,GDL_A), q1);
Jv_A22= functionalDerivative(PO_A(2,1,GDL_A), q2);
Jv_A23= functionalDerivative(PO_A(2,1,GDL_A), q3);
% Derivadas parciales de z respecto a q1, q2, q3
Jv_A31= functionalDerivative(PO_A(3,1,GDL_A), q1);
Jv_A32= functionalDerivative(PO_A(3,1,GDL_A), q2);
Jv_A33= functionalDerivative(PO_A(3,1,GDL_A), q3);
% Creamos la matriz del Jacobiano lineal
jv_d_A=simplify([Jv_A11 Jv_A12 Jv_A13;
               Jv_A21 Jv_A22 Jv_A23;
               Jv_A31 Jv_A32 Jv_A33]);
pretty(jv_d_A);
```

```
/ -sin(q1(t)) #1, -cos(q1(t)) #2,   -l3 cos(q1(t)) #3   \
|                                                       |
```

```
|  cos(q1(t)) #1, -sin(q1(t)) #2,   -l3 sin(q1(t)) #3   |
|                                                       |
\       0,             #1,       l3 cos(q2(t) + q3(t)) /

where

   #1 == l2 cos(q2(t)) + l3 cos(q2(t) + q3(t))

   #2 == l2 sin(q2(t)) + l3 #3

   #3 == sin(q2(t) + q3(t))
```

```
% Calculamos el jacobiano lineal de forma analítica
% Inicializamos jacobianos analíticos (lineal y angular)
Jv_a_A(:,GDL_A)=PO_A(:,:,GDL_A);
Jw_a_A(:,GDL_A)=PO_A(:,:,GDL_A);

for k= 1:GDL_A
    if ((joint_types_A(k)==0)|(joint_types_A(k)==1)) % Casos: articulación
rotacional y prismática

       % Para las articulaciones rotacionales
        try
            Jv_a_A(:,k)= cross(RO_A(:,3,k-1), PO_A(:,:,GDL_A)-
PO_A(:,:,k-1)); % *****
            Jw_a_A(:,k)= RO_A(:,3,k-1);
        catch
            Jv_a_A(:,k)= cross([0,0,1], PO_A(:,:,GDL_A)); % Matriz de
rotación de 0 con respecto a 0 es la Matriz Identidad, la posición previa
también será 0
            Jw_a_A(:,k)=[0,0,1]; % Si no hay matriz de rotación previa se
obtiene la Matriz identidad
         end
     else
        % Para las articulaciones prismáticas
        try
            Jv_a_A(:,k)= RO_A(:,3,k-1);
        catch
            Jv_a_A(:,k)=[0,0,1]; % Si no hay matriz de rotación previa se
obtiene la Matriz identidad
        end
            Jw_a_A(:,k)=[0,0,0];
     end
end

Jv_a_A= simplify (Jv_a_A);
Jw_a_A= simplify (Jw_a_A);
V_A=simplify (Jv_a_A*Qp_A');
W_A=simplify (Jw_a_A*Qp_A');
```

COMPARACIÓN DE AMBOS ROBOTS EN JACOBIANO Y ANTROPOMÓRFICO

```
% Mostrar jacobianos y velocidades para el robot convencional
disp('----------- Robot Planar -----------');
```

```
----------- Robot Planar -----------
```

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

```
Jacobiano lineal obtenido de forma analítica
```

```
pretty(Jv_a);
```

```
/ - l1 sin(th1(t)) - l2 sin(th1(t) + th2(t)), -l2 sin(th1(t) + th2(t)) \
|                                                                      |
|  l1 cos(th1(t)) + l2 cos(th1(t) + th2(t)),   l2 cos(th1(t) + th2(t)) |
|                                                                      |
\                     0,                                  0            /
```

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

```
Jacobiano angular obtenido de forma analítica
```

```
pretty(Jw_a);
```

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

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

```
Velocidad lineal obtenida mediante el Jacobiano lineal
```

```
pretty(V);
```

```
/   _________                                 _________    \
|    d                                         d           |
| - -- th1(t) (l1 sin(th1(t)) + l2 #1) - l2 -- th2(t) #1 |
|   dt                                        dt           |
|                                                          |
|  _________                                 _________     |
|   d                                         d            |
|  -- th1(t) (l1 cos(th1(t)) + l2 #2) + l2 -- th2(t) #2  |
|  dt                                        dt            |
|                                                          |
\                            0                             /

where

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

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

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

```
Velocidad angular obtenida mediante el Jacobiano angular
```

PODEMOS VER QUE LA VELOCIDAD ANGULAR SOLO ESTÁ EN Z

```
pretty(W);
```

```
/           0           \
|                       |
|           0           |
|                       |
| _________   _________ |
|  d           d        |
| -- th1(t) + -- th2(t) |
\ dt          dt        /
```

```
% Separación para mayor claridad
disp('-----------------------------------------');
```

```
-----------------------------------------
```

```
% Mostrar jacobianos y velocidades para el robot antropomórfico
disp('----------- Robot Antropomórfico -----------');
```

```
----------- Robot Antropomórfico -----------
```

```
disp('Jacobiano lineal obtenido de forma analítica (Robot Antropomórfico)');
```

```
Jacobiano lineal obtenido de forma analítica (Robot Antropomórfico)
```

```
pretty(Jv_a_A);
```

```
/ -sin(q1(t)) #1, -cos(q1(t)) #2,   -l3 cos(q1(t)) #3   \
|                                                       |
|  cos(q1(t)) #1, -sin(q1(t)) #2,   -l3 sin(q1(t)) #3   |
|                                                       |
\        0,             #1,       l3 cos(q2(t) + q3(t)) /

where

   #1 == l2 cos(q2(t)) + l3 cos(q2(t) + q3(t))

   #2 == l2 sin(q2(t)) + l3 #3

   #3 == sin(q2(t) + q3(t))
```

```
disp('Jacobiano angular obtenido de forma analítica (Robot Antropomórfico)');
```

```
Jacobiano angular obtenido de forma analítica (Robot Antropomórfico)
```

```
pretty(Jw_a_A);
```

```
/ 0,  sin(q1(t)),  sin(q1(t)) \
|                             |
| 0, -cos(q1(t)), -cos(q1(t)) |
|                             |
\ 1,      0,           0      /
```

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

```
Velocidad lineal obtenida mediante el Jacobiano lineal (Robot Antropomórfico)
```

```
pretty(V_A);
```

```
/   ________                                                       \
|    d                                                             |
| - -- q1(t) sin(q1(t)) #3 - #2 cos(q1(t)) #4 - l3 #1 cos(q1(t)) #6 |
|   dt                                                             |
|                                                                  |
|  ________                                                        |
|   d                                                              |
|  -- q1(t) cos(q1(t)) #3 - #2 sin(q1(t)) #4 - l3 #1 sin(q1(t)) #6  |
|  dt                                                              |
|                                                                  |
\                          #2 #3 + l3 #1 #5                        /

where

        ________
         d
   #1 == -- q3(t)
        dt

        ________
         d
   #2 == -- q2(t)
        dt

   #3 == l2 cos(q2(t)) + l3 #5

   #4 == l2 sin(q2(t)) + l3 #6

   #5 == cos(q2(t) + q3(t))

   #6 == sin(q2(t) + q3(t))
```

```
disp('Velocidad angular obtenida mediante el Jacobiano angular (Robot
Antropomórfico)');
```

```
Velocidad angular obtenida mediante el Jacobiano angular (Robot Antropomórfico)
```

EN ESTE, PODEMOS VER COMO LA VELOCIDAD ANGULAR ESTÁ EN LOS 3 EJES

```
pretty(W_A);
```

```
/             / ________   ________ \ \
|             |  d          d       | |
|  sin(q1(t)) | -- q2(t) + -- q3(t) | |
|             \ dt         dt       / |
|                                     |
|             / ________   ________ \ |
|             |  d          d       | |
| -cos(q1(t)) | -- q2(t) + -- q3(t) | |
|             \ dt         dt       / |
|                                     |
|               ________              |
|                d                    |
|               -- q1(t)              |
\               dt                    /
```