```
%Limpieza de pantalla
clear
close all
clc

tic
%Declaración de variables simbólicas
syms th1(t) th2(t) th3(t) th4(t) th5(t) th6(t)  t   %Angulos de cada
articulación
syms th1p(t) th2p(t) th3p(t) th4p(t) th5p(t) th6p(t)  %Velocidades de cada
articulación
syms th1pp(t) th2pp(t) th3pp(t) th4pp(t) th5pp(t) th6pp(t)%Aceleraciones de
cada articulación
syms m1 m2 m3 m4 m5 m6 Ixx1 Iyy1 Izz1 Ixx2 Iyy2 Izz2 Ixx3 Iyy3 Izz3 Ixx4
Iyy4 Izz4 Ixx5 Iyy5 Izz5 Ixx6 Iyy6 Izz6 %Masas y matrices de Inercia
syms d1 d2 l3 d4 d5 d6 dc1 dc2 lc3 dc4 dc5 dc6 %l=longitud de eslabones y
lc=distancia al centro de masa de cada eslabón
syms pi g a cero theta

 %Creamos el vector de coordenadas articulares
  Q= [th1; th2; th3; th4; th5; th6];
 %disp('Coordenadas generalizadas');
 %pretty (Q);

 %Creamos el vector de velocidades articulares
  Qp= [th1p; th2p; th3p; th5p; th5p; th6p];
 %disp('Velocidades generalizadas');
 %pretty (Qp);
 %Creamos el vector de aceleraciones articulares
  Qpp= [th1pp; th2pp; th3pp; th4pp; th5pp; th6pp];
 %disp('Aceleraciones generalizadas');
 %pretty (Qpp);

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

Se comienza limpiando la pantalla y las variables en MATLAB con los comandos `clear`, `close all` y `clc`. Luego se definen las variables simbólicas necesarias para el cálculo de la cinemática y dinámica del robot, como los ángulos de las articulaciones (`th1` a `th6`), las velocidades (`th1p` a `th6p`), las aceleraciones (`th1pp` a `th6pp`), y las propiedades físicas del robot como las masas (`m1` a `m6`) y las matrices de inercia (`Ixx1`, `Iyy1`, etc.). Se define también el vector de coordenadas articulares `Q`, que representa los ángulos de cada articulación. Asimismo, se definen los vectores de velocidades (`Qp`) y aceleraciones (`Qpp`) articulares. En esta sección, también se especifica la configuración del robot mediante el vector RP, que indica si las articulaciones son rotacionales (0) o prismáticas (1). Finalmente, se obtiene el número de grados de libertad del robot, que se define según el tamaño del vector RP.

```
%Número de grado de libertad del robot
```

```
GDL= size(RP,2);
GDL_str= num2str(GDL);

%Articulación 1
P(:,:,1)= [0;0;d1];
R(:,:,1)= rotacion_z(90)*rotacion_x(-90);

%Articulación 2
P(:,:,2)= [0;0;d2];
R(:,:,2)= rotacion_z(90)*rotacion_x(90);

%Articulación 3
P(:,:,3)= [0;0;l3];
R(:,:,3)= eye(3);

%Articulación 4
P(:,:,4)= [0;0;d4];
R(:,:,4)= rotacion_z(90)*rotacion_x(-90);

%Articulación 5
P(:,:,5)= [0;0;d5];
R(:,:,5)= rotacion_z(90)*rotacion_x(90);

%Articulación 6
P(:,:,6)= [0;0;d6];
R(:,:,6)= eye(3);

%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);


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);
    PO(:,:,i)= T(1:3,4,i);
    %pretty(RO(:,:,i));
    %pretty(PO(:,:,i));
end
```

En esta parte se definen las matrices de posición y rotación para cada articulación del robot. Las matrices de posición Pindican la ubicación de cada articulación en el espacio, considerando desplazamientos en los ejes X, Y y Z. Por ejemplo, la posición del primer eslabón está definida por `[0; 0; d1]`, donde `d1` es la distancia del primer eslabón respecto al origen. Las matrices de rotación R describen la orientación de cada eslabón en relación con la articulación anterior. Se usan combinaciones de rotaciones sobre los ejes Z y X para cada eslabón, lo que refleja cómo cambia la orientación del eslabón respecto a la articulación previa. Estas matrices son fundamentales para transformar las coordenadas locales de cada eslabón a coordenadas globales.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CALCULAMOS LAS VELOCIDADES PARA CADA
ESLABÓN%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 6  %%%%%%%%%%

%Calculamos el jacobiano lineal de forma analítica
Jv_a6(:,GDL)=PO(:,:,GDL);
Jw_a6(:,GDL)=PO(:,:,GDL);

for k= 1:GDL-5
    if RP(k)==0
        % Para las juntas de revolución
        try
            Jv_a6(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL)-PO(:,:,k-1));
            Jw_a6(:,k)= RO(:,3,k-1);
        catch
            Jv_a6(:,k)= cross([0,0,1], PO(:,:,GDL)); % Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa también
será 0
            Jw_a6(:,k)=[0,0,1]; % Si no hay matriz de rotación previa se
obtiene la Matriz identidad
        end
    else
        % Para las juntas prismáticas
        try
            Jv_a6(:,k)= RO(:,3,k-1);
```

```
        catch
            Jv_a6(:,k)=[0,0,1];
        end
        Jw_a6(:,k)=[0,0,0];
    end
end

% Obtenemos SubMatrices de Jacobianos
Jv_a6 = simplify(Jv_a6);
Jw_a6 = simplify(Jw_a6);

% Matriz de Jacobiano Completa
Jac6 = [Jv_a6;
        Jw_a6];
Jacobiano6 = simplify(Jac6);
Qp=Qp(t);

% Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 6
disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
6');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 6
```

```
V6 = simplify(Jv_a6*Qp);
pretty(V6)
```

```
/ - th6p(t) (d2 + d6) - th1p(t) (d4 + l3) \
|                                         |
|  th6p(t) (d4 + l3) - th1p(t) (d2 + d6)  |
|                                         |
\            th6p(t) (d1 + d5)            /
```

```
disp('Velocidad angular obtenida mediante el Jacobiano lineal del Eslabón
6');
```

```
Velocidad angular obtenida mediante el Jacobiano lineal del Eslabón 6
```

```
W6 = simplify(Jw_a6*Qp);
pretty(W6)
```

```
/      -th6p(t) (d2 + d6)     \
|                             |
|      th6p(t) (d4 + l3)      |
|                             |
\ th1p(t) + th6p(t) (d1 + d5) /
```

```
%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 5  %%%%%%%%%%

%Calculamos el jacobiano lineal de forma analítica
Jv_a5(:,GDL-1)=PO(:,:,GDL-1);
Jw_a5(:,GDL-1)=PO(:,:,GDL-1);
```

```
for k= 1:GDL-5
    if RP(k)==0
        % Para las juntas de revolución
        try
            Jv_a5(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-1)-PO(:,:,k-1));
            Jw_a5(:,k)= RO(:,3,k-1);
        catch
            Jv_a5(:,k)= cross([0,0,1], PO(:,:,GDL-1)); % Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa también
será 0
            Jw_a5(:,k)=[0,0,1]; % Si no hay matriz de rotación previa se
obtiene la Matriz identidad
        end
    else
        % Para las juntas prismáticas
        try
            Jv_a5(:,k)= RO(:,3,k-1);
        catch
            Jv_a5(:,k)=[0,0,1];
        end
        Jw_a5(:,k)=[0,0,0];
    end
end

% Obtenemos SubMatrices de Jacobianos
Jv_a5 = simplify(Jv_a5);
Jw_a5 = simplify(Jw_a5);

% Matriz de Jacobiano Completa
Jac5 = [Jv_a5;
        Jw_a5];
Jacobiano5 = simplify(Jac5);

% Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 5
disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
5');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 5
```

```
V5 = simplify(Jv_a5*Qp(1:5));
pretty(V5)
```

```
/ - th1p(t) (d4 + l3) - d2 th5p(t) \
|                                  |
|  th5p(t) (d4 + l3) - d2 th1p(t)  |
|                                  |
\         th5p(t) (d1 + d5)        /
```

```
disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón
5');
```

Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 5

```
W5 = simplify(Jw_a5*Qp(1:5));
pretty(W5)
```

```
/         -d2 th5p(t)          \
|                              |
|      th5p(t) (d4 + l3)       |
|                              |
\ th1p(t) + th5p(t) (d1 + d5) /
```

```
%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 4  %%%%%%%%%%

%Calculamos el jacobiano lineal de forma analítica
Jv_a4(:,GDL-2)=PO(:,:,GDL-2);
Jw_a4(:,GDL-2)=PO(:,:,GDL-2);

for k= 1:GDL-2
    if RP(k)==0
        % Para las juntas de revolución
        try
            Jv_a4(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-2)-PO(:,:,k-1));
            Jw_a4(:,k)= RO(:,3,k-1);
        catch
            Jv_a4(:,k)= cross([0,0,1], PO(:,:,GDL-2)); % Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa también
será 0
            Jw_a4(:,k)=[0,0,1]; % Si no hay matriz de rotación previa se
obtiene la Matriz identidad
        end
    else
        % Para las juntas prismáticas
        try
            Jv_a4(:,k)= RO(:,3,k-1);
        catch
            Jv_a4(:,k)=[0,0,1];
        end
        Jw_a4(:,k)=[0,0,0];
    end
end

% Obtenemos SubMatrices de Jacobianos
Jv_a4 = simplify(Jv_a4);
Jw_a4 = simplify(Jw_a4);

% Matriz de Jacobiano Completa
Jac4 = [Jv_a4;
        Jw_a4];
Jacobiano4 = simplify(Jac4);
```

```
% Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 4
disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
4');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 4
```

```
V4 = simplify(Jv_a4*Qp(1:4));
pretty(V4)
```

```
/  -th1p(t) (d4 + l3)  \
|                      |
| th3p(t) - d2 th1p(t) |
|                      |
\  -th2p(t) (d4 + l3)  /
```

```
disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón
4');
```

```
Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 4
```

```
W4 = simplify(Jw_a4*Qp(1:4));
pretty(W4)
```

```
/ -th2p(t) \
|          |
|  th5p(t) |
|          |
\  th1p(t) /
```

```
%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 3  %%%%%%%%%%

%Calculamos el jacobiano lineal de forma analítica
Jv_a3(:,GDL-3)=PO(:,:,GDL-3);
Jw_a3(:,GDL-3)=PO(:,:,GDL-3);

for k= 1:GDL-3
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a3(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-3)-PO(:,:,k-1));
            Jw_a3(:,k)= RO(:,3,k-1);
        catch
            Jv_a3(:,k)= cross([0,0,1], PO(:,:,GDL-3));%Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa tambien
será 0
            Jw_a3(:,k)=[0,0,1];%Si no hay matriz de rotación previa se
obtiene la Matriz identidad
         end
     else
%         %Para las juntas prismáticas
        try
            Jv_a3(:,k)= RO(:,3,k-1);
```

```
        catch
            Jv_a3(:,k)=[0,0,1];
        end
            Jw_a3(:,k)=[0,0,0];
    end
 end

%Obtenemos SubMatrices de Jacobianos
Jv_a3= simplify (Jv_a3);
Jw_a3= simplify (Jw_a3);
%disp('Jacobiano lineal obtenido de forma analítica');
%pretty (Jv_a);
%disp('Jacobiano ángular obtenido de forma analítica');
%pretty (Jw_a);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac3= [Jv_a3;
      Jw_a3];
Jacobiano3= simplify(Jac3);
% pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 2
 disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
3');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 3
```

```
V3=simplify(Jv_a3*Qp(1:3));
pretty(V3)
```

```
/      -l3 th1p(t)     \
|                       |
| th3p(t) - d2 th1p(t)  |
|                       |
\      -l3 th2p(t)     /
```

```
 disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón
3');
```

```
Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 3
```

```
W3=simplify(Jw_a3*Qp(1:3));
pretty(W3)
```

```
/ -th2p(t) \
|          |
|     0    |
|          |
\  th1p(t) /
```

```
%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 2  %%%%%%%%%%

%Calculamos el jacobiano lineal y angular de forma analítica
Jv_a2(:,GDL-4)=PO(:,:,GDL-4);
Jw_a2(:,GDL-4)=PO(:,:,GDL-4);

for k= 1:GDL-4
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a2(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-4)-PO(:,:,k-1));
            Jw_a2(:,k)= RO(:,3,k-1);
        catch
            Jv_a2(:,k)= cross([0,0,1], PO(:,:,GDL-4));%Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa tambien
será 0
            Jw_a2(:,k)=[0,0,1];%Si no hay matriz de rotación previa se
obtiene la Matriz identidad
         end
     else
%         %Para las juntas prismáticas
        try
            Jv_a2(:,k)= RO(:,3,k-1);
        catch
            Jv_a2(:,k)=[0,0,1];
        end
            Jw_a2(:,k)=[0,0,0];
     end
 end

%Obtenemos SubMatrices de Jacobianos
Jv_a2= simplify (Jv_a2);
Jw_a2= simplify (Jw_a2);
%disp('Jacobiano lineal obtenido de forma analítica');
%pretty (Jv_a);
%disp('Jacobiano ángular obtenido de forma analítica');
%pretty (Jw_a);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac2= [Jv_a2;
      Jw_a2];
Jacobiano2= simplify(Jac2);
% pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 2
disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
2');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 2
```

```
V2=simplify (Jv_a2*Qp(1:2));
pretty(V2)
```

```
/      0      \
|             |
| -d2 th1p(t) |
|             |
\      0      /
```

```
 disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón
2');
```

```
Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 2
```

```
W2=simplify (Jw_a2*Qp(1:2));
pretty(W2)
```

```
/ -th2p(t) \
|          |
|     0    |
|          |
\  th1p(t) /
```

```
%%%%%%%%%%  VELOCIDADES PARA ESLABÓN 1  %%%%%%%%%%%%%%%%%%%%%%%%%%

%Calculamos el jacobiano lineal y angular de forma analítica
Jv_a1(:,GDL-5)=PO(:,:,GDL-5);
Jw_a1(:,GDL-5)=PO(:,:,GDL-5);

for k= 1:GDL-5
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a1(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-5)-PO(:,:,k-1));
            Jw_a1(:,k)= RO(:,3,k-1);
        catch
            Jv_a1(:,k)= cross([0,0,1], PO(:,:,GDL-5));%Matriz de rotación
de 0 con respecto a 0 es la Matriz Identidad, la posición previa tambien
será 0
            Jw_a1(:,k)=[0,0,1];%Si no hay matriz de rotación previa se
obtiene la Matriz identidad
         end
     else
%         %Para las juntas prismáticas
        try
            Jv_a1(:,k)= RO(:,3,k-1);
        catch
            Jv_a1(:,k)=[0,0,1];
        end
            Jw_a1(:,k)=[0,0,0];
     end
```

```
 end

%Obtenemos SubMatrices de Jacobianos
Jv_a1= simplify (Jv_a1);
Jw_a1= simplify (Jw_a1);
%disp('Jacobiano lineal obtenido de forma analítica');
%pretty (Jv_a);
%disp('Jacobiano ángular obtenido de forma analítica');
%pretty (Jw_a);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac1= [Jv_a1;
      Jw_a1];
Jacobiano1= simplify(Jac1);
% pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares para el eslabón 1
 disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón
1');
```

```
Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 1
```

```
V1=simplify (Jv_a1*Qp(1:1));
pretty(V1)
```

```
/ 0 \
|   |
| 0 |
|   |
\ 0 /
```

```
 disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón
1');
```

```
Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 1
```

```
W1=simplify (Jw_a1*Qp(1:1));
pretty(W1)
```

```
/    0    \
|         |
|    0    |
|         |
\ th1p(t) /
```

El cálculo de las velocidades articulares se realiza mediante el Jacobiano, que relaciona las velocidades articulares con las velocidades lineales y angulares de los eslabones. Para cada articulación, dependiendo de si es rotacional o prismática, se calcula la velocidad lineal (Jv_a6) y la velocidad angular (Jw_a6). Para las juntas de revolución, se utiliza el producto cruzado entre el vector de rotación de la articulación anterior y la diferencia de las posiciones de los eslabones, mientras que para las juntas prismáticas, la velocidad angular es cero y la velocidad lineal es simplemente la rotación de la articulación. Los Jacobianos resultantes se

almacenan y simplifican para obtener las submatrices correspondientes de las velocidades articulares, que son clave para la simulación de las trayectorias y el análisis del movimiento del robot.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Energía Cinética
%%%%%%%%%%%%%%%%%%%%%%%%%%Omitimos la división de cada lc%%%%%%%%%%%%%%%
%Distancia del origen del eslabón a su centro de masa
%Vectores de posición respecto al centro de masa
 P01=subs(P(:,:,1), d1, dc1);%La función subs sustituye l1 por lc1 en
 P12=subs(P(:,:,2), d2, dc2); %la expresión P(:,:,1)/2
 P23=subs(P(:,:,3), l3, lc3);
 P34=subs(P(:,:,4), d4, dc4);
 P45=subs(P(:,:,5), d5, dc5);
 P56=subs(P(:,:,6), d6, dc6);

%Creamos matrices de inercia para cada eslabón

I1=[Ixx1 0 0;
    0 Iyy1 0;
    0 0 Izz1];

I2=[Ixx2 0 0;
    0 Iyy2 0;
    0 0 Izz2];

I3=[Ixx3 0 0;
    0 Iyy3 0;
    0 0 Izz3];

I4 = [Ixx4 0 0;
      0 Iyy4 0;
      0 0 Izz4];

I5 = [Ixx5 0 0;
      0 Iyy5 0;
      0 0 Izz5];

I6 = [Ixx6 0 0;
      0 Iyy6 0;
      0 0 Izz6];


%Función de energía cinética

%Calculamos la energía cinética para cada uno de los eslabones%%%%%%%%%%

%Eslabón 1
V1_Total= V1+cross(W1,P01);
```

```
K1= (1/2*m1*(V1_Total))'*((V1_Total)) + (1/2*W1)'*(I1*W1);
disp('Energía Cinética en el Eslabón 1');
```

```
Energía Cinética en el Eslabón 1
```

```
K1= simplify (K1);
pretty (K1);
```

```
             2
Izz1 |th1p(t)|
---------------
       2
```

```
%Eslabón 2
V2_Total= V2+cross(W2,P12);
K2= (1/2*m2*(V2_Total))'*((V2_Total)) + (1/2*W2)'*(I2*W2);
disp('Energía Cinética en el Eslabón 2');
```

```
Energía Cinética en el Eslabón 2
```

```
K2= simplify (K2);
pretty (K2);
```

```
                                                                  /          2     2            2      2 \
                                                                  | |th1p(t)|  |d2|    |th2p(t)|  |dc2|  |
             2                 2   __                             | ---------------- - ----------------- |
Ixx2 |th2p(t)|    Izz2 |th1p(t)|  m2 (d2 th1p(t) - dc2 th2p(t)) \    d2 th1p(t)         dc2 th2p(t)    /
--------------- + --------------- + ------------------------------------------------------------------------
       2                 2                                      2
```

```
%Eslabón 3
V3_Total= V3+cross(W3,P23);
K3= (1/2*m3*(V3_Total))'*((V3_Total)) + (1/2*W3)'*(I2*W3);
disp('Energía Cinética en el Eslabón 3');
```

```
Energía Cinética en el Eslabón 3
```

```
K3= simplify (K3);
pretty (K3);
```

```
                                                    /          2           2            2  \
                                                 __ | |th3p(t)|     #2 |d2|      #1 |lc3|   |
                          2 __          2 __     m3 | ---------- - ---------- + ----------- | (th3p(t) - d2
Ixx2 #1   Izz2 #2   #2 |l3|  m3   #1 |l3|  m3      \   th3p(t)    d2 th1p(t)   lc3 th2p(t) /
------- + ------- + ----------- + ----------- + ------------------------------------------------------------
   2         2           2             2                                    2
```

```
where

                2
   #1 == |th2p(t)|

                2
   #2 == |th1p(t)|
```

```
%Eslabón 4
V4_Total = V4 + cross(W4, P34);
K4 = (1/2 * m4 * (V4_Total))' * (V4_Total) + (1/2 * W4)' * (I4 * W4);
disp('Energía Cinética en el Eslabón 4');
```

```
Energía Cinética en el Eslabón 4
```

```
K4 = simplify(K4);
pretty(K4);
```

```
                                                                  /          2                  \      /
                                                                  |  #4 |dc4|         #3 #1     |    __ |
                          __                                      | ----------- - ------------- |   m4 |
Ixx4 #2   Iyy4 #4   Izz4 #3  m4 (th1p(t) (d4 + l3) - dc4 th5p(t)) \ dc4 th5p(t)   d4 l3 th1p(t) /      \
------- + ------- + ------- - ------------------------------------------------------------------- + ----
   2         2         2                                    2
```

where

```
                2          2
   #1 == l3 |d4|  + d4 |l3|

                2
   #2 == |th2p(t)|

                2
   #3 == |th1p(t)|

                2
   #4 == |th5p(t)|
```

```
%Eslabón 5
V5_Total = V5 + cross(W5, P45);
K5 = (1/2 * m5 * (V5_Total))' * (V5_Total) + (1/2 * W5)' * (I5 * W5);
disp('Energía Cinética en el Eslabón 5');
```

```
Energía Cinética en el Eslabón 5
```

```
K5 = simplify(K5);
pretty(K5);
```

```
                                                                  __
                                                                  m5 (th1p(t) (d4 + l3) + d2 th5p(t) -
                                  /     #4           #1 #3      \
Izz5 (th1p(t) + th5p(t) (d1 + d5)) | --------- + ---------------- | + ------------------------------------
                                  \ 2 th1p(t)   2 d1 d5 th5p(t) /
```

where

```
                2
   #1 == |th5p(t)|

                2          2
   #2 == l3 |d4|  + d4 |l3|

                2          2
   #3 == d5 |d1|  + d1 |d5|
```

```
                      2
   #4 == |th1p(t)|

   #5 == d2 dc5 th5p(t)
```

```
%Eslabón 6
V6_Total = V6 + cross(W6, P56);
K6 = (1/2 * m6 * (V6_Total))' * (V6_Total) + (1/2 * W6)' * (I6 * W6);
disp('Energía Cinética en el Eslabón 6');
```

```
Energía Cinética en el Eslabón 6
```

```
K6 = simplify(K6);
pretty(K6);
```

```
                                                                              __
                                                                              m6 (th6p(t) (d4 + l3) - th1p(t) (d2 -
                                           /      #5            #1 #4      \
Izz6 (th1p(t) + th6p(t) (d1 + d5)) | --------- + --------------- | + -----------------------------------
                                           \ 2 th1p(t)    2 d1 d5 th6p(t) /

where

                      2
   #1 == |th6p(t)|

                    2           2
   #2 == l3 |d4|  + d4 |l3|

                    2           2
   #3 == d6 |d2|  + d2 |d6|

                    2           2
   #4 == d5 |d1|  + d1 |d5|

                      2
   #5 == |th1p(t)|
```

```
K_Total= simplify (K1+K2+K3+K4+K5+K6);
disp('Energía Cinética Total');
```

```
Energía Cinética Total
```

```
pretty (K_Total);
```

```
                                                                                                        /
                                                                                                        |
                                                                                                 __     |
                                                                              2 __         2 __  m3 | #7 - #9 -
          Ixx4 #5    Iyy4 #1    Izz1 #13                Izz4 #13    #13 |l3|  m3    #5 |l3|  m3     \
Ixx2 #5 + -------- + -------- + --------- + Izz2 #13 + --------- + ------------ + ----------- + -------------
             2          2           2                       2             2             2

where

                      2
   #1 == |th5p(t)|
```

```
                2
#2 == |th6p(t)|

              2          2
#3 == d6 |d2|  + d2 |d6|

              2          2
#4 == d5 |d1|  + d1 |d5|

                2
#5 == |th2p(t)|

#6 == th1p(t) (d4 + l3)

                2
       |th3p(t)|
#7 == ----------
        th3p(t)

#8 == d2 dc5 th5p(t)

                2
       #13 |d2|
#9 == ----------
      d2 th1p(t)

          #13 #12
#10 == -------------
       d4 l3 th1p(t)

          #13
#11 == ---------
       2 th1p(t)

              2          2
#12 == l3 |d4|  + d4 |l3|

                2
#13 == |th1p(t)|
```

```
%Energia Potencial  p=mgh

%Obtenemos las alturas respecto a la gravedad
 h1= P01(3); %Tomo la altura paralela al eje z
 h2= P12(2); %Tomo la altura paralela al eje y
 h3= P23(3);
 h4= P34(3);
 h5= P45(2);
 h6= P56(3);

 U1=m1*g*h1
```

U1 = $\mathrm{dc}_1\, g\, m_1$

```
 U2=m2*g*h2
```

U2 = 0

```
U3=m3*g*h3
```

U3 = $g\, \mathrm{lc}_3\, m_3$

```
U4=m4*g*h4
```

U4 = $\mathrm{dc}_4\, g\, m_4$

```
U5=m5*g*h5
```

U5 = 0

```
U6=m6*g*h6
```

U6 = $\mathrm{dc}_6\, g\, m_6$

```
%Calculamos la energía potencial total
U_Total= U1 + U2 + U3 + U4 + U6 +U5;

%Obtenemos el Lagrangiano
Lagrangiano= simplify (K_Total-U_Total);
pretty (Lagrangiano);
```

```
                                                                                                                              /
                                                                                                                           __ |
                                                                                           2 __          2 __   m3 | #7 - #9 -
           Ixx4 #5    Iyy4 #1    Izz1 #13                Izz4 #13    #13 |l3|  m3    #5 |l3|  m3        \
Ixx2 #5 + ------- + -------- + --------- + Izz2 #13 + --------- + ------------ + ----------- + -------------
              2          2           2                      2             2              2

where

                     2
   #1 == |th5p(t)|

                     2
   #2 == |th6p(t)|

                   2          2
   #3 == d6 |d2|  + d2 |d6|

                   2          2
   #4 == d5 |d1|  + d1 |d5|

                     2
   #5 == |th2p(t)|

   #6 == th1p(t) (d4 + l3)

                     2
         |th3p(t)|
   #7 == ----------
           th3p(t)

   #8 == d2 dc5 th5p(t)

                     2
```

```
          #13 |d2|
   #9 == ----------
         d2 th1p(t)

             #13 #12
   #10 == -------------
          d4 l3 th1p(t)

             #13
   #11 == ---------
          2 th1p(t)

                 2           2
   #12 == l3 |d4|  + d4 |l3|

                   2
   #13 == |th1p(t)|
```

```
%Modelo de Energía
 H= simplify (K_Total+U_Total);
 pretty (H)
```

```
                                                                                               /
                                                                                            __ |
                                                                        2 __          2 __  m3 | #7 - #9 -
          Ixx4 #5   Iyy4 #1   Izz1 #13              Izz4 #13   #13 |l3|  m3   #5 |l3|  m3      \
Ixx2 #5 + ------- + ------- + -------- + Izz2 #13 + --------- + ------------ + ----------- + --------------
             2         2          2                     2            2             2

where

                 2
   #1 == |th5p(t)|

                 2
   #2 == |th6p(t)|

               2          2
   #3 == d6 |d2|  + d2 |d6|

               2          2
   #4 == d5 |d1|  + d1 |d5|

                 2
   #5 == |th2p(t)|

   #6 == th1p(t) (d4 + l3)

                 2
         |th3p(t)|
   #7 == ----------
           th3p(t)

   #8 == d2 dc5 th5p(t)

                 2
          #13 |d2|
   #9 == ----------
         d2 th1p(t)

             #13 #12
   #10 == -------------
```

```
       d4 l3 th1p(t)

          #13
#11 == ---------
       2 th1p(t)

                2          2
#12 == l3 |d4|  + d4 |l3|

                2
#13 == |th1p(t)|
```

Finalmente, en esta parte se calcula la energía cinética de cada eslabón del robot. La energía cinética de un eslabón se calcula utilizando la fórmula K=12mV2+12WTIWK=21mV2+21WTIW, donde mm es la masa del eslabón, VV es su velocidad lineal, WW es su velocidad angular, y II es la matriz de inercia. Para el primer eslabón, se calcula la velocidad total como la suma de la velocidad de traslación y la velocidad angular, y luego se obtiene la energía cinética. Este proceso se repite para cada eslabón del robot. La energía cinética es un parámetro importante en el análisis de la dinámica del robot, ya que permite calcular el trabajo realizado por los actuadores y analizar la eficiencia energética del sistema.

```
function Rx = rotacion_x(theta)
    % Matriz de rotación alrededor de X en grados
    Rx = [1,         0,           0;
          0, cosd(theta), -sind(theta);
          0, sind(theta),  cosd(theta)];
end

function Rz = rotacion_z(theta)
    % Matriz de rotación alrededor de Z en grados
    Rz = [cosd(theta), -sind(theta), 0;
          sind(theta),  cosd(theta), 0;
                 0,           0, 1];
end
```

Definicion de las funciones de las matrices de rotacion