Actividad 5.

Fundamentación de robótica gpo101

Hecho por:

A01736196 | Abraham Ortiz Castro

A01736001 | Alan Iván Flores Juárez

A01735823 | Ulises Hernández Hernández

A01736171 | Jesús Alejandro Gómez Bautista

ITESM puebla

Profesor:

Alfredo García Suárez.

07 de marzo del 2024.

Primer ejercicio.

Modelo.

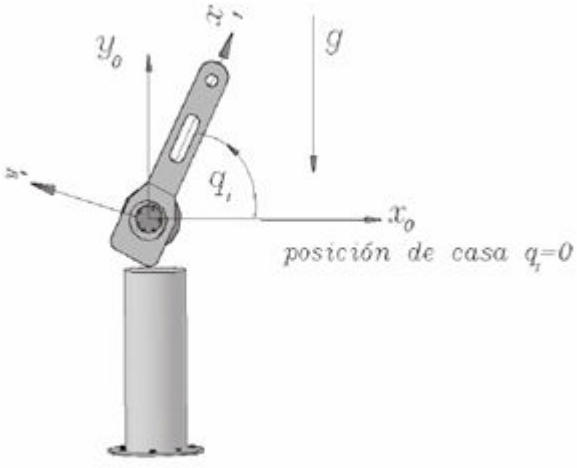
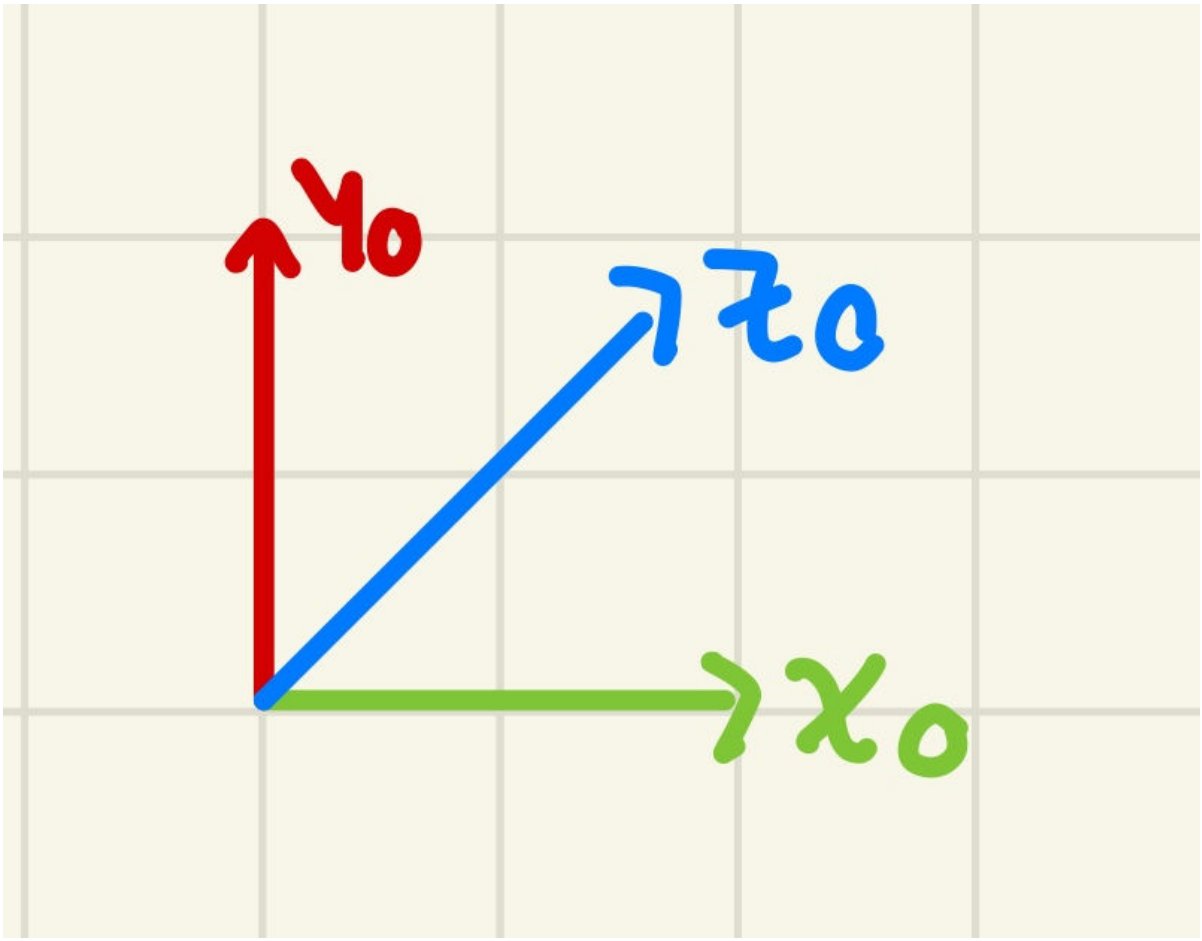


Figura 4.10 Péndulo robot.

Marco de referencia para determinar las alturas de la energía potencial.



Debido a que todo el código ya se ha explicado anteriormente, solamente queda explicar la selección de las alturas en la energía potencial, en este caso solamente se tiene una junta, por lo tanto solamente hay que seleccionar una altura, en el marco de referencia se puede observar que respecto al mundo la altura estará en y, por lo tanto se le pone dos a ese lugar.

En este caso la suma de la energía potencial es la misma energía de la junta porque solamente es uno. Al final lo único que se hace es sumar la energía total potencial y cinética para poder obtener el modelo de energía y para obtener el langrangiano se debe de restar la energia potencial a la energía cinética.

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

tic
%Declaración de variables simbólicas
syms th1(t)  t  %Angulos de cada articulación
syms m1 Ixx1 Iyy1 Izz1  %Masas y matrices de Inercia
syms l1 lc1   %l=longitud de eslabones y lc=distancia al centro de masa de cada
eslabón
syms g
```

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

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

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

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

%Articulación 1
%Posición 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];


%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


%Calculamos el jacobiano lineal de forma diferencial
%disp('Jacobiano lineal obtenido de forma diferencial');
%Derivadas parciales de x respecto a th1 y th2
Jv11= functionalDerivative(PO(1,1,GDL), th1);
%Derivadas parciales de y respecto a th1 y th2
Jv21= functionalDerivative(PO(2,1,GDL), th1);
%Derivadas parciales de z respecto a th1 y th2
Jv31= functionalDerivative(PO(3,1,GDL), th1);

%Creamos la matríz del Jacobiano lineal
jv_d=simplify([Jv11;
               Jv21 ;
               Jv31 ]);
%pretty(jv_d);

%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
       %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));%Matriz de rotación de 0 con
respecto a 0 es la Matriz Identidad, la posición previa tambien será 0
            Jw_a(:,k)=[0,0,1];%Si no hay matriz de rotación previa se obtiene la
Matriz identidad
         end
     else
```

```
%         %Para las juntas prismáticas
        try
            Jv_a(:,k)= RO(:,3,k-1);
        catch
            Jv_a(:,k)=[0,0,1];
        end
            Jw_a(:,k)=[0,0,0];
    end
 end

%Obtenemos SubMatrices de Jacobianos
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);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac= [Jv_a;
      Jw_a];
Jacobiano= simplify(Jac)
```

```
Jacobiano =
```

$$\begin{pmatrix} -l_1 \sin(\mathrm{th}_1(t)) \\ l_1 \cos(\mathrm{th}_1(t)) \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

```
%pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares
% 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);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Energía Cinética

%Distancia del origen del eslabón a su centro de masa
%Vectores de posición respecto al centro de masa
 P01=subs(P(:,:,1)/2, l1, lc1); %La función subs sustituye l1 por lc1 en
                                %la expresión P(:,:,1)/2
```

```
%Creamos matrices de inercia para cada eslabón

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


%Función de energía cinética

%Extraemos las velocidades lineales en cada eje
V=V(t);
Vx= V(1,1);
Vy= V(2,1);
Vz= V(3,1);

%Extraemos la velocidad angular en cada ángulo de Euler
W=W(t);
W_pitch= W(1,1);
W_roll= W(2,1);
W_yaw= W(3,1);

%Calculamos las velocidades para cada eslabón

%Eslabón 1

%Ya lo calculamos previamente al multiplicar la matriz jacobiana por Qp

%Calculamos la energía cinética para cada uno de los eslabones

%Eslabón 1
V1_Total= V+cross(W,P01); %Se suma la velocidad lineal producida por la
                          % velocidad angular producida en el punto P01
K1= (1/2*m1*(V1_Total))'*(1/2*m1*(V1_Total)) + (1/2*W)'*(I1*W);
disp('Energía Cinética en el Eslabón 1');
```

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

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

```
     |  d        | 2    |  d        | 2     ______                2          2            2
Izz1 | -- th1(t) |      | -- th1(t) |   cos(th1(t) - th1(t)) |m1|  (l1 |lc1|  + 2 lc1 |l1| ) (2 l1 + lc1)
     | dt        |      | dt        |
-------------------- + ------------------------------------------------------------------------------
         2                                          16 l1 lc1
```

```
K_Total= simplify (K1);

toc
```

```
Elapsed time is 2.543802 seconds.
```

```
%Obtenemos las alturas respecto a la gravedad
 h1= P01(2); %Tomo la altura paralela al eje y

 U1=m1*g*h1;

 %Calculamos la energía potencial total
 U_Total= U1;

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

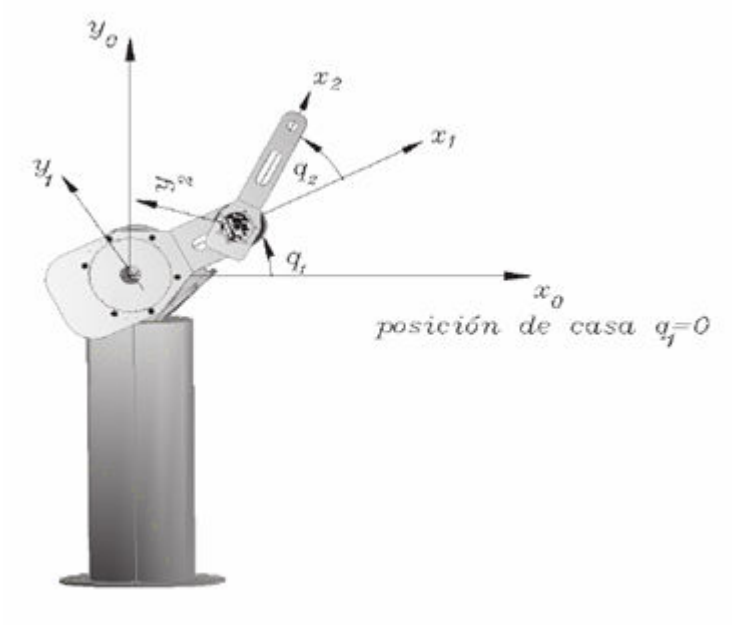
```
       |  d         | 2                                 |  d         | 2      ______                 2          2             2
Izz1   | -- th1(t)  |                                   | -- th1(t)  |   cos(th1(t) - th1(t)) |m1|   (l1 |lc1|   + 2 lc1 |l1| )
       | dt         |     g lc1 m1 sin(th1(t))          | dt         |
-------------------- - -------------------- + ---------------------------------------------------------------------------
          2                      2                                              16 l1 lc1
```

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

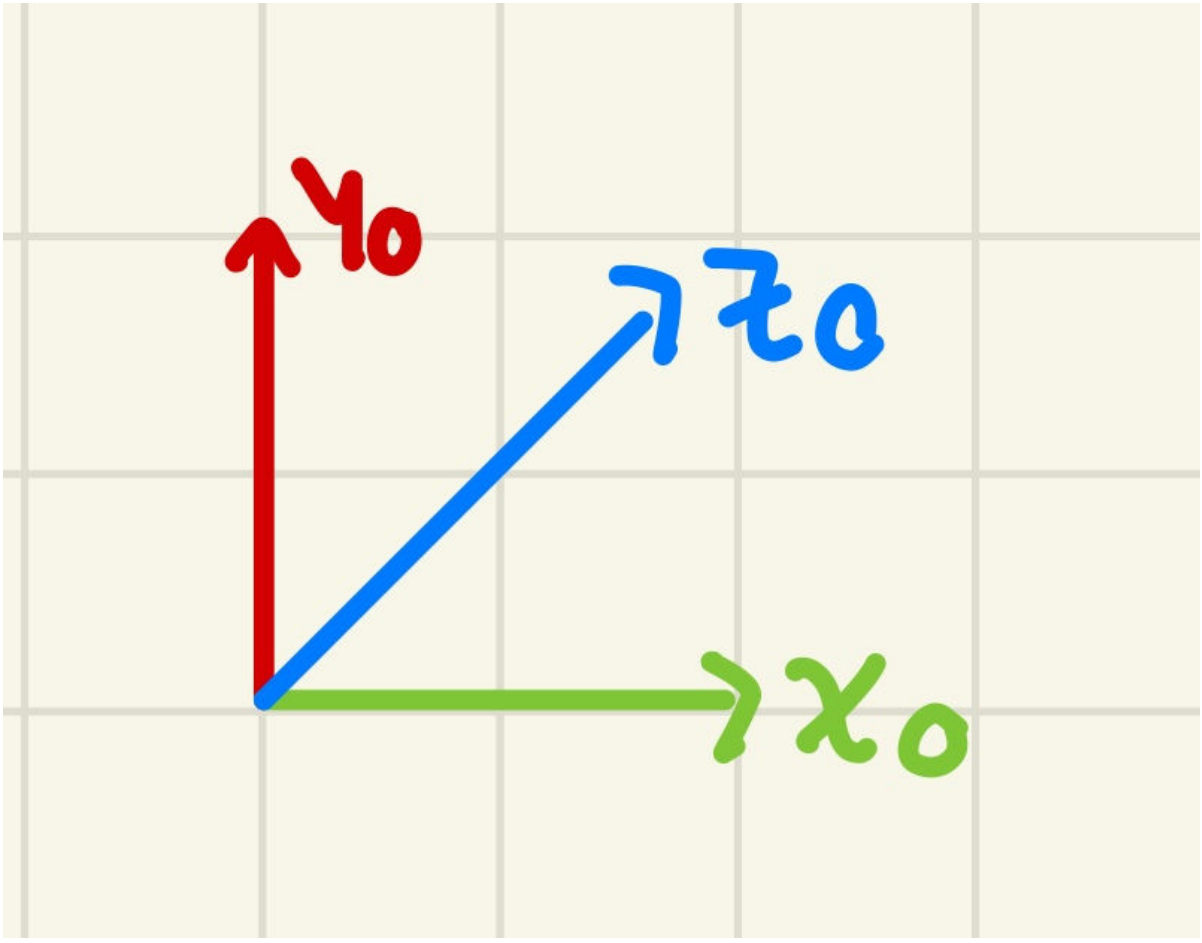
```
       |  d         | 2                                 |  d         | 2      ______                 2          2             2
Izz1   | -- th1(t)  |                                   | -- th1(t)  |   cos(th1(t) - th1(t)) |m1|   (l1 |lc1|   + 2 lc1 |l1| )
       | dt         |     g lc1 m1 sin(th1(t))          | dt         |
-------------------- + -------------------- + ---------------------------------------------------------------------------
          2                      2                                              16 l1 lc1
```

Segundo ejercicio.

***Modelo*.**



Marco de referencia para determinar las alturas de la energía potencial.



Debido a que todo el código ya se ha explicado anteriormente, solamente queda explicar la selección de las alturas en la energía potencial, es muy similar al primer modelo, debido a que el sistema de referencia

es el mismo, la diferencia es que son dos juntas, pero el sistema no cambia, por lo tanto ambas alturas se encontraran respecto a y, esto es un dos dentro del código. (segundo eje en orden de x , y, z)

En este caso la suma de la energía potencial se debe de sumar el de la primera junta y el de la segunda, dando así la total.

Al final lo único que se hace es sumar la energía total potencial y cinética para poder obtener el modelo de energía y para obtener el langrangiano se debe de restar la energia potencial a la energía cinética.

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

tic
%Declaración de variables simbólicas
syms th1(t) th2(t) t  %Angulos de cada articulación
syms th1p(t) th2p(t)    %Velocidades de cada articulación
syms th1pp(t) th2pp(t)     %Aceleraciones de cada articulación
syms m1 m2 Ixx1 Iyy1 Izz1 Ixx2 Iyy2 Izz2 %Masas y matrices de Inercia
syms l1 l2 lc1 lc2  %l=longitud de eslabones y lc=distancia al centro de masa de
cada eslabón
syms pi g a cero

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

 %Creamos el vector de velocidades articulares
  Qp= [th1p; th2p];
 %disp('Velocidades generalizadas');
 %pretty (Qp);
 %Creamos el vector de aceleraciones articulares
  Qpp= [th1pp; th2pp];
 %disp('Aceleraciones generalizadas');
 %pretty (Qpp);

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

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

%Articulación 1
%Posición 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) 0  -sin(th1);
```

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

%Articulación 2
%Posición de la articulación 2 respecto a 1
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);


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



%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
       %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));%Matriz de rotación de 0 con
respecto a 0 es la Matriz Identidad, la posición previa tambien será 0
            Jw_a(:,k)=[0,0,1];%Si no hay matriz de rotación previa se obtiene la
Matriz identidad
         end
     else
%         %Para las juntas prismáticas
        try
            Jv_a(:,k)= RO(:,3,k-1);
        catch
            Jv_a(:,k)=[0,0,1];
        end
            Jw_a(:,k)=[0,0,0];
     end
 end

%Obtenemos SubMatrices de Jacobianos
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);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac= [Jv_a;
      Jw_a];
Jacobiano= simplify(Jac);
% pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares
% 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);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%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), l1, lc1);%La función subs sustituye l1 por lc1 en
 P12=subs(P(:,:,2), l2, lc2); %la expresión P(:,:,1)/2

%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];

%Función de energía cinética

%Extraemos las velocidades lineales en cada eje
V= V(t);
Vx= V(1,1);
Vy= V(2,1);
Vz= V(3,1);

%Extraemos las velocidades angular en cada ángulo de Euler
W=W(t);
W_pitch= W(1,1);
W_roll= W(2,1);
W_yaw= W(3,1);

%Calculamos las velocidades para cada eslabón%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Eslabón 1
%Calculamos el jacobiano lineal y angular de forma analítica
Jv_a1(:,GDL-1)=PO(:,:,GDL-1);
Jw_a1(:,GDL-1)=PO(:,:,GDL-1);

for k= 1:GDL-1
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a1(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-1)-PO(:,:,k-1));
            Jw_a1(:,k)= RO(:,3,k-1);
        catch
            Jv_a1(:,k)= cross([0,0,1], PO(:,:,GDL-1));%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
 %disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 1');
Qp=Qp(t);
V1=simplify (Jv_a1*Qp(1));
 %pretty(V1);
 % disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 1');
W1=simplify (Jw_a1*Qp(1));
 % pretty(W1);



%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');
K1= simplify (K1);
%pretty (K1);


%Eslabón 2
V2_Total= V+cross(W,P12);
K2= (1/2*m2*(V2_Total))'*((V2_Total)) + (1/2*W)'*(I2*W);
%disp('Energía Cinética en el Eslabón 3');
K2= simplify (K2);
```

```
%pretty (K3);

K_Total= simplify (K1+K2);
pretty (K_Total);
```

```
                        __
                        m2 (lc2 cos(th2(t)) th1p(t) + cos(th1(t)) th1p(t) (l1 + #6) - l2 sin(th1(t)) sin(th2(t)) th2p(t
Izz1 #1   Izz2 #1
------- + ------- + ---------------------------------------------------------------------------------------------------
   2         2                                                                             2

                                                                                                    /
     __                                                                                             | #1 #3 |lc2
     m2 (lc2 sin(th2(t)) th1p(t) + sin(th1(t)) th1p(t) (l1 + #6) + l2 cos(th1(t)) sin(th2(t)) th2p(t)) | ----------
                                                                                                    \  lc2 th1p(t
   + ------------------------------------------------------------------------------------------------------------
                                                                          2

                                     __                                        2       _______  _______      2
     Ixx2 #2 sin(th1(t)) #4   #2 m2 (#6 + lc2 cos(th1(t) - th2(t))) (lc2 #9 |l2|  + l2 cos(th1(t) - th2(t)) |lc2| )
   + ---------------------- + -----------------------------------------------------------------------------------
                2                                                     2 l2 lc2

where

                   2
   #1 == |th1p(t)|

                   2
   #2 == |th2p(t)|

                ______
   #3 == sin(th2(t))

                ______
   #4 == sin(th1(t))

                ______
   #5 == cos(th1(t))

   #6 == l2 cos(th2(t))

   #7 == l1 l2 th1p(t)

                 2              2
   #8 == l2 |l1|  + l1 #9 |l2|

                ______
   #9 == cos(th2(t))
```

```
%Obtenemos las alturas respecto a la gravedad
 h1= P01(2); %Tomo la altura paralela al eje z
 h2= P12(2); %Tomo la altura paralela al eje y

 U1=m1*g*h1;
 U2=m2*g*h2;

 %Calculamos la energía potencial total
 U_Total= U1 + U2;
```

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

```
                    __
                    m2 (lc2 cos(th2(t)) th1p(t) + cos(th1(t)) th1p(t) (l1 + #6) - l2 sin(th1(t)) sin(th2(t)) th2p(t
Izz1 #1   Izz2 #1
------- + ------- + -----------------------------------------------------------------------------------------------
   2         2                                                                        2

                                                                                                            /
     __                                                                                                     | #1 #3 |lc2
     m2 (lc2 sin(th2(t)) th1p(t) + sin(th1(t)) th1p(t) (l1 + #6) + l2 cos(th1(t)) sin(th2(t)) th2p(t)) | ----------
                                                                                                            \  lc2 th1p(
   + --------------------------------------------------------------------------------------------------------------
                                                                         2

                                                                                   __
     Ixx2 #2 sin(th1(t)) #4                                                 #2 m2 (#6 + lc2 cos(th1(t) - th2(t))) (
   + ---------------------- - g lc1 m1 sin(th1(t)) - g lc2 m2 sin(th2(t)) + --------------------------------------
                2                                                                                                  2

              ______                        2           2
     #1 cos(th1(t) - th1(t)) m1 (l1 + lc1) (lc1 |l1|  + l1 |lc1| )
   + -------------------------------------------------------------
                               2 l1 lc1

where

                  2
   #1 == |th1p(t)|

                  2
   #2 == |th2p(t)|

                 ______
   #3 == sin(th2(t))

                 ______
   #4 == sin(th1(t))

                 ______
   #5 == cos(th1(t))

   #6 == l2 cos(th2(t))

   #7 == l1 l2 th1p(t)

                2             2
   #8 == l2 |l1|  + l1 #9 |l2|

                 ______
   #9 == cos(th2(t))
```

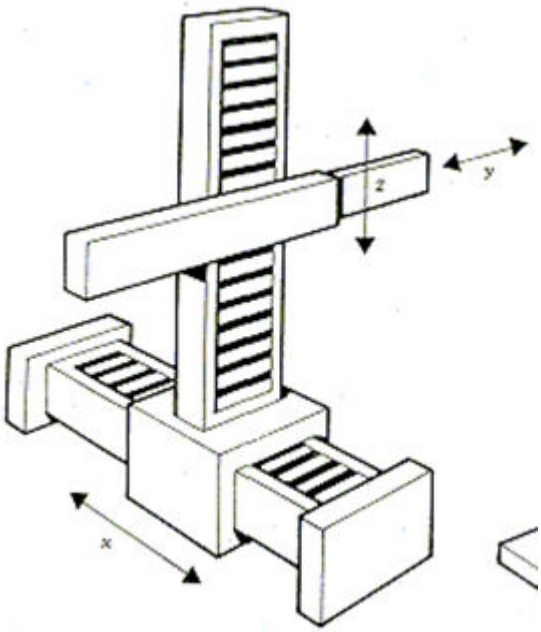
```
%Modelo de Energía
H= simplify (K_Total+U_Total);
 pretty (H)
```

```
                    __
```

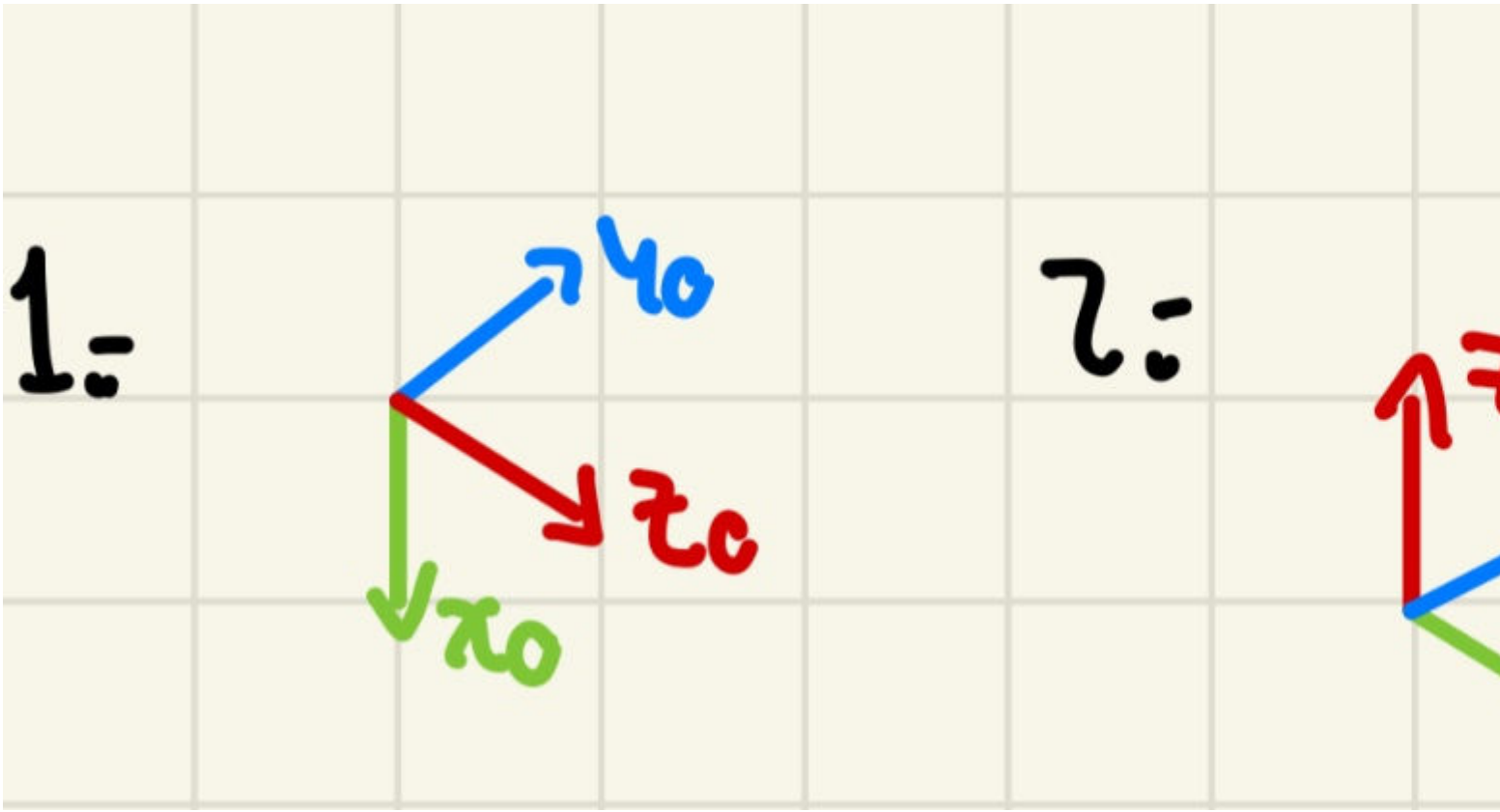
```
                     m2 (lc2 cos(th2(t)) th1p(t) + cos(th1(t)) th1p(t) (l1 + #6) - l2 sin(th1(t)) sin(th2(t)) th2p(t
Izz1 #1   Izz2 #1
------- + ------- + -----------------------------------------------------------------------------------------
   2         2                                                                        2

                                                                                                              /
     __                                                                                                       | #1 #3 |lc2
     m2 (lc2 sin(th2(t)) th1p(t) + sin(th1(t)) th1p(t) (l1 + #6) + l2 cos(th1(t)) sin(th2(t)) th2p(t)) | -----------
                                                                                                              \  lc2 th1p(t
   + ---------------------------------------------------------------------------------------------------------------
                                                                        2

                                                                                     __
     Ixx2 #2 sin(th1(t)) #4                                                      #2 m2 (#6 + lc2 cos(th1(t) - th2(t))) (l
   + ---------------------- + g lc1 m1 sin(th1(t)) + g lc2 m2 sin(th2(t)) + -------------------------------------
               2                                                                                                 2

            _______          __                          2           2
     #1 cos(th1(t) - th1(t)) m1 (l1 + lc1) (lc1 |l1|  + l1 |lc1| )
   + -------------------------------------------------------------
                               2 l1 lc1

where

                    2
   #1 == |th1p(t)|

                    2
   #2 == |th2p(t)|

                ______
   #3 == sin(th2(t))

                ______
   #4 == sin(th1(t))

                ______
   #5 == cos(th1(t))

   #6 == l2 cos(th2(t))

   #7 == l1 l2 th1p(t)

                  2              2
   #8 == l2 |l1|  + l1 #9 |l2|

                ______
   #9 == cos(th2(t))
```

Tercer ejercicio.

Modelo.



Marco de referencia para determinar las alturas de la energía potencial.



Debido a que todo el código ya se ha explicado anteriormente, solamente queda explicar la selección de las alturas en la energía potencial, para este modelo se tienen tres sistemas, en el caso de la primera junta se puede observar en el sistema de referencias que la altura respecto al mundo será en x, por lo tanto el eje seleccionado será 1. En el caso de la segunda junta, la altura se encuentra en el eje z (como se observa en el sistema de referencia 2), ya que se encuentra en z se le dará el valor de 3. Y para la tercera junta, según

el sistema de referencia se puede observar que la altura respecto al mundo es de y, por lo tanto se le dará el valor de 2 a la toma de altura.

En este caso para obtener la energía potencial total solamente queda sumar las tres energías potenciales de cada junta individualmente.

Al final lo único que se hace es sumar la energía total potencial y cinética para poder obtener el modelo de energía y para obtener el langrangiano se debe de restar la energia potencial a la energía cinética.

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

tic
%Declaración de variables simbólicas
syms m1 m2 m3 Ixx1 Iyy1 Izz1 Ixx2 Iyy2 Izz2 Ixx3 Iyy3 Izz3 %Masas y matrices de
Inercia
syms l1(t) l2(t) l3(t) t lc1 lc2 lc3  %l=longitud de eslabones y lc=distancia al
centro de masa de cada eslabón
syms l1p(t) l2p(t) l3p(t)
syms l1pp(t) l2pp(t) l3pp(t)
syms pi g a cero

 %Creamos el vector de coordenadas articulares
  Q= [l1; l2; l3];
 %disp('Coordenadas generalizadas');
 %pretty (Q);

 %Creamos el vector de velocidades articulares
  Qp= [l1p; l2p; l3p];
 %disp('Velocidades generalizadas');
 %pretty (Qp);
 %Creamos el vector de aceleraciones articulares
  Qpp= [l1pp; l2pp; l3pp];
 %disp('Aceleraciones generalizadas');
 %pretty (Qpp);

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

%Número de grado de libertad del robot
GDL= size(RP,2);
GDL_str= num2str(GDL);
%Articulación 1
%Posición de la articulación 1 respecto a 0
P(:,:,1)= [0 ; 0; l1];
%Matriz de rotación de la junta 1 respecto a 0.... -90º alrededor del eje "y1"
R(:,:,1)= [0  0 -1 ;
           0  1  0;
```

```
          1  0  0];


%Articulación 2
%Posición de la articulación 2 respecto a 1
P(:,:,2)= [0; 0 ; l2];
%Matriz de rotación de la junta 2 respecto a 1  -90º alrededor del eje "x2"
R(:,:,2)= [1  0 0;
           0  0 1;
           0 -1 0];

%Articulación 3
%Posición de la articulación 3 respecto a 2
P(:,:,3)= [0; 0 ;l3];
%Matriz de rotación de la junta 3 respecto a 2
R(:,:,3)= [1  0 0;
           0  1 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);


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



%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
       %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));%Matriz de rotación de 0 con
respecto a 0 es la Matriz Identidad, la posición previa tambien será 0
            Jw_a(:,k)=[0,0,1];%Si no hay matriz de rotación previa se obtiene la
Matriz identidad
         end
     else
%         %Para las juntas prismáticas
        try
            Jv_a(:,k)= RO(:,3,k-1);
        catch
            Jv_a(:,k)=[0,0,1];
        end
            Jw_a(:,k)=[0,0,0];
     end
 end

%Obtenemos SubMatrices de Jacobianos
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);

%Matriz de Jacobiano Completa
%disp('Matriz de Jacobiano');
Jac= [Jv_a;
      Jw_a];
Jacobiano= simplify(Jac);
% pretty(Jacobiano);

%Obtenemos vectores de Velocidades Lineales y Angulares
```

```
% 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);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%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), l1, lc1);%La función subs sustituye l1 por lc1 en
 P12=subs(P(:,:,2), l2, lc2); %la expresión P(:,:,1)/2
 P23=subs(P(:,:,3), l3, lc3);

%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];

%Función de energía cinética

%Extraemos las velocidades lineales en cada eje
V=V(t);
Vx= V(1,1);
Vy= V(2,1);
Vz= V(3,1);

%Extraemos las velocidades angular en cada ángulo de Euler
W=W(t);
W_pitch= W(1,1);
W_roll= W(2,1);
W_yaw= W(3,1);

%Calculamos las velocidades para cada eslabón%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Eslabón 1

%Calculamos el jacobiano lineal y angular de forma analítica
Jv_a1(:,GDL-2)=PO(:,:,GDL-2);
```

```
Jw_a1(:,GDL-2)=PO(:,:,GDL-2);

for k= 1:GDL-2
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a1(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-2)-PO(:,:,k-1));
            Jw_a1(:,k)= RO(:,3,k-1);
        catch
            Jv_a1(:,k)= cross([0,0,1], PO(:,:,GDL-2));%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
 %disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 1');
Qp=Qp(t);
V1=simplify (Jv_a1*Qp(1));
 %pretty(V1);
 % disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 1');
W1=simplify (Jw_a1*Qp(1));
 % pretty(W1);
```

```
%Eslabón 2

%Calculamos el jacobiano lineal de forma analítica
Jv_a2(:,GDL-1)=PO(:,:,GDL-1);
Jw_a2(:,GDL-1)=PO(:,:,GDL-1);

for k= 1:GDL-1
    if RP(k)==0
       %Para las juntas de revolución
        try
            Jv_a2(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL-1)-PO(:,:,k-1));
            Jw_a2(:,k)= RO(:,3,k-1);
        catch
            Jv_a2(:,k)= cross([0,0,1], PO(:,:,GDL-1));%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
%disp('Velocidad lineal obtenida mediante el Jacobiano lineal del Eslabón 2');
V2=simplify (Jv_a2*Qp(1:2));
% pretty(V2);
%disp('Velocidad angular obtenida mediante el Jacobiano angular del Eslabón 2');
W2=simplify (Jw_a2*Qp(1:2));
```

```
% pretty(W2);

%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');
K1= simplify (K1);
%pretty (K1);


%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');
K2= simplify (K2);
%pretty (K2);


%Eslabón 3
V3_Total= V+cross(W,P23);
K3= (1/2*m3*(V3_Total))'*((V3_Total)) + (1/2*W)'*(I3*W);
%disp('Energía Cinética en el Eslabón 3');
K3= simplify (K3);
%pretty (K3);

K_Total= simplify (K1+K2+K3);
pretty(K_Total)
```

```
__          2          2          2           2 __   __          2          2
m3 (|l1p(t)|  + |l2p(t)|  + |l3p(t)| )   |l1p(t)|  m1   m2 (|l1p(t)|  + |l2p(t)| )
------------------------------------ + ------------ + --------------------------
                 2                          2                    2
```

```
%Analisis de la energia potencial
%Obtenemos las alturas respecto a la gravedad
 h1= P01(1); %Tomo la altura paralela al eje x
 h2= P12(3); %Tomo la altura paralela al eje z
 h3= P23(2); %Tomo la altura paralela al eje y

 U1=m1*g*h1;
 U2=m2*g*h2;
 U3=m3*g*h3;

 %Calculamos la energía potencial total
 U_Total= U1 + U2 +U3;

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

```
__           2           2           2           2 __   __           2           2
m3 (|l1p(t)|  + |l2p(t)|  + |l3p(t)| )   |l1p(t)|  m1   m2 (|l1p(t)|  + |l2p(t)| )
------------------------------------ + ------------ + -------------------------- - g lc2 m2
                 2                          2                    2
```

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

```
__           2           2           2           2 __   __           2           2
m3 (|l1p(t)|  + |l2p(t)|  + |l3p(t)| )   |l1p(t)|  m1   m2 (|l1p(t)|  + |l2p(t)| )
------------------------------------ + ------------ + -------------------------- + g lc2 m2
                 2                          2                    2
```