INTEGRANTES:

A01735217 - Diego García Rueda

A01734225 - Jonathan Josafat Vázquez Suárez

A01734193 - Jhonatan Yael Martinez Vargas

DESARROLLO DE LA ACTIVIDAD:

Se usan comandos de limpieza de *workspace*

```
clear all
close all
clc
```

Se crean variables simbolicas que se usarán en todo el programa.

```
syms q1(t) q2(t) q3(t) a1 a2 a3 t
```

Se establece la configuración del robot, en este caso las 3 articulaciones son rotacionales.

```
RP=[0 0 0];
```

Se crea un vector de coordenadas particulares. Posteriormente este vector es derivado para obtener un vector de velocidades articulares.

```
Q = [q1, q2, q3];
Qp = diff(Q, t);
```

Se establece el número de los Grados De Libertad (GDL) tanto como valor numerico como dato de tipo *String*

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

ARTICULACIÓN 1

Se crea el vector de traslación de la junta 1 respecto a la 0. (Cinemática directa de un pendulo). Se crea también la matriz de rotación en el eje z. Esto es debido a que esta articulación rotará respecto al eje Z. Ambos se guardan en la página 1.

```
P(:,:,1)= [a1*cos(q1); a1*sin(q1); 0];
R(:,:,1)= [cos(q1) -sin(q1)  0; sin(q1)  cos(q1) 0; 0 0 1];
```

ARTICULACIÓN 2

Se crea el vector de traslación de la junta 2 respecto a la 1. (Cinemática directa de un pendulo). Se crea también la matriz de rotación en el eje z. Esto es debido a que esta articulación rotará respecto al eje Z. Ambos se guardan en la página 2.

```
P(:,:,2)= [a2*cos(q2); a2*sin(q2); 0];
R(:,:,2)= [cos(q2) -sin(q2)  0; sin(q2)  cos(q2) 0; 0 0 1];
```

ARTICULACIÓN 3

Se crea el vector de traslación de la junta 3 respecto a la 2. (Cinemática directa de un pendulo). Se crea también la matriz de rotación en el eje z. Esto es debido a que esta articulación rotará respecto al eje Z. Ambos se guardan en la página 3.

```
P(:,:,3)= [a3*cos(q3); a3*sin(q3); 0];
R(:,:,3)= [cos(q3) -sin(q3)  0; sin(q3)  cos(q3) 0; 0 0 1];
```

CREACIÓN DE MATRICES GLOBALES Y LOCALES

Se crea un vector de ceros, Este vector se usa para "completar" las matrices homogéneas locales y globales, este vector permite que esta matriz sea una matriz cuadrada.

```
Vector_Zeros= zeros(1, 3);
```

Se inicializan las matrices de transformacion locales y global usando los vectores de traslación y las matrices de rotación. Para realizar las matrices cuadradas se agrega el vector de ceros.

```
A(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
T(:,:,GDL)=simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
```

Se inicializan los valores de los vectores de posición y de rotación (vistos desde el marco de referencia inercial).

```
PO(:,:,GDL)= P(:,:,GDL);
RO(:,:,GDL)= R(:,:,GDL);
```

Se realiza un ciclo *for* en el que usando los Grados De LIbertad (GDL) como iterador. Usando este iterador se van creando las matrices locales de cada junta.

Posteriormente se vn creando las matrices de transformación globales. Aqui existen 2 casos en los que se pueden generar estas matrices:

- Caso #1: Solo hay 1 GDL por lo que la matriz global es igual a la matriz de transformación local de esa junta.
- Caso #2: Hay más de 1 GDL, en eset caso, la matriz global se genera al multiplicar la matriz global anterior (es decir, de la articulacion anterior) por la matriz local actual (de la articulación actual) generando asi la nueva matriz de tranformación global.

Obtenemos la matriz de rotación (RO) y el vector de traslación (PO) a partir de la Matriz de Transformación Homogénea Global.

```
for i = 1:GDL
    i_str= num2str(i);
    %Matrices Locales
    A(:,:,i)=simplify([R(:,:,i) P(:,:,i); Vector_Zeros 1]);

    %Matrices Globales
```

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

    T(:,:,i)= simplify(T(:,:,i));

    %Da la matriz de rotacion de la matriz global
    RO(:,:,i)= T(1:3,1:3,i);
    %Vector de traslación de la matriz global
    PO(:,:,i)= T(1:3,4,i);

end
```

CALCULO DE JACOBIANOS

Método de diferenciación:

Primero obtenemos el Jacobiano Lineal usando el método de diferenciación. Se realizan derivadas parciales respecto a x, y, z obteniendo asi 9 derivadas parciales. Para esto se usan los vectores de posición

```
Jv11= functionalDerivative(PO(1,1,GDL), q1);
Jv12= functionalDerivative(PO(1,1,GDL), q2);
Jv13= functionalDerivative(PO(1,1,GDL), q3);

Jv21= functionalDerivative(PO(2,1,GDL), q1);
Jv22= functionalDerivative(PO(2,1,GDL), q2);
Jv23= functionalDerivative(PO(2,1,GDL), q3);

Jv31= functionalDerivative(PO(3,1,GDL), q1);
Jv32= functionalDerivative(PO(3,1,GDL), q2);
Jv33= functionalDerivative(PO(3,1,GDL), q3);
```

Se crea la matriz del Jacobiano lineal.

```
jv_d=simplify([Jv11 Jv12 Jv13; Jv21 Jv22 Jv23; Jv31 Jv32 Jv33]);
```

Método Análitico:

Se inicializan los Jacobianos Análiticos (Lineal y Angular). Posteriomente, usando de nuevo un ciclo *for* usando como argumento la configuración del robot (RP), en este caso las 3 articulaciones son rotacionales. El ciclo *for* queda de la siguiente manera:

```
Jv_a(:,GDL)=PO(:,:,GDL);
Jw_a(:,GDL)=PO(:,:,GDL);

for k= 1:GDL
    if RP(k)==0
        try
            Jv_a(:,k)= cross(RO(:,3,k-1), PO(:,:,GDL)-PO(:,:,k-1));         %Formula Ja
```

```
            Jw_a(:,k)= RO(:,3,k-1);                                    %Formula Ja
        catch
            Jv_a(:,k)= cross([0,0,1], PO(:,:,GDL));
            Jw_a(:,k)=[0,0,1];
        end
    end
end
```

En la sección de *try* se encuentran las formulas del Jacobiano Lineal Analiticoo y del Jacobiano Angular Analitico respectivamente.

En la sección de *catch* se encuentran las formulas para los Jacobianos en caso de que solo haya 1 GDL, debido a que no tenemos articulaciones previas usando asi una matriz identidad.

Se simplifican los Jacobianos obtenidos:

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

Se mandan a imprimir las velocidades lineal y angular obtenidas usando sus Jacobianos correspondientes diferenciandolos.

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

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

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

```
/ - #4 (a3 sin(#1) + a2 sin(#2)) - #5 (a1 sin(q1(t)) + a3 sin(#1) + a2 sin(#2)) - a3 #3 sin(#1) \
|                                                                                               |
|  #4 (a3 cos(#1) + a2 cos(#2)) + #5 (a1 cos(q1(t)) + a3 cos(#1) + a2 cos(#2)) + a3 #3 cos(#1)  |
|                                                                                               |
\                                             0                                                 /

where

   #1 == q1(t) + q2(t) + q3(t)

   #2 == q1(t) + q2(t)

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

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

         ________
          d
   #5 == -- q1(t)
         dt
```

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

Velocidad angular obtenida mediante el Jacobiano angular

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

```
/                0                \
|                                 |
|                0                |
|                                 |
| ________   ________   ________ |
|  d          d          d       |
| -- q1(t) + -- q2(t) + -- q3(t) |
\ dt         dt         dt       /
```