Antonio Silva Martínez A01173663

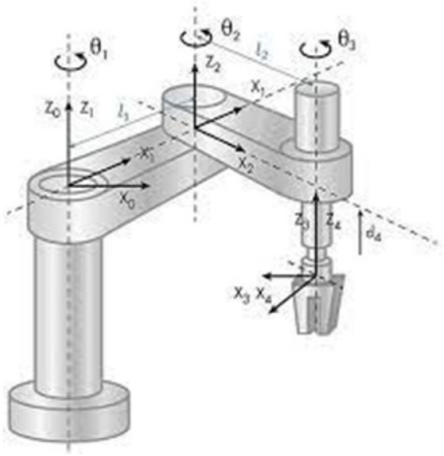
03/03/2023

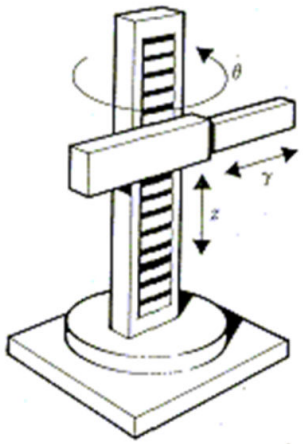
Actividad 3 Evaluación

Los robots son dispositivos programables y automatizados que pueden realizar tareas específicas de manera autónoma o bajo la supervisión humana. En particular, los robots de 3GDL tienen tres articulaciones que les permiten moverse en tres direcciones diferentes: rotación en torno a un eje, inclinación hacia arriba y hacia abajo, y movimiento hacia adelante y hacia atrás.

Para controlar el movimiento de un robot de 3GDL, se utilizan matrices de transformación homogéneas. Estas matrices son herramientas matemáticas que permiten describir la posición y orientación de un objeto en el espacio tridimensional. Cada matriz representa una transformación que lleva el robot desde una posición inicial hasta una posición final en el espacio. Al multiplicar varias matrices juntas, se puede describir el movimiento completo del robot en términos de las tres articulaciones.

En esta actividad se realizó el análisis de 2 robots, uno cilindrico y otro un robot :





El primer robot es uno con articulaciones cilindricas las cuales el marco de referencia es z debido a que el eje de rotación que se utilizó fue este mismo, en las artuculaciones ya que el marco de referencia utilizado fue dentro del plano "z x y", en donde las rotaciones se realizaban sobre el eje z y el desplazamiento de los eslabones se proyectaba dentro de "x" y "y". Una de las cosas de las que tuvimos que tomar en cuenta es la altura a la que se encuentran nuestras articulaciones, ya que esta no se encuentra al ras del suelo, si no que esta se encuentra sobre un soporte el cual se encuentra sobre el eje z y por ende es necesario el tomarlo en cuenta dentro de nuestro sistema,

El robot al encontrarce sobre el mismo marco de referencia podíamos utilizar la matriz de rotación del eje z, siendo que las unicas diferencias que podemos encontrar es en el tamaño de los eslabones, siendo que al

no conocer las medidas de destos, unicamente se deja expresado y se toman en cuenta dentro de nuestro sistema, por otra parte la ultima articulación unicamente rota sobre su propio eje, por eso esta no cuenta como tal con un eslabon el cual cuente con un desplazamiento.

Explicación del código:

La primera de parte dentro de los códigos nos sirve para poder declarar variables, declarar que tipo de articulación es, el numero de grados de libertad con los que contará, crear los vectores de coordenadas y de velocidades generalizadas.

```
%Limpieza de pantalla
clear all
close all
clc
%Declaración de variables simbólicas
syms th1(t) th2(t) th3(t) t l1 l2 a1
%Configuración del robot, 0 para junta rotacional, 1 para junta prismática
RP=[0 0 0];
%Creamos el vector de coordenadas articulares
Q= [th1, th2 th3];
%disp('Coordenadas generalizadas');
%pretty (Q);
%Creamos el vector de velocidades generalizadas
Qp= diff(Q, t);
%disp('Velocidades generalizadas');
%pretty (Qp);
%Número de grado de libertad del robot
GDL= size(RP,2);
GDL_str= num2str(GDL);
```

La segunda parte de nuestro código es donde nosotros declaramos las articulaciones y cómo será el comportamiento de estas, es en esta parte en donde tenemos que tomar en cuenta la altura inicial para poder tomarlo en cuenta dentro de nuestro sistema

```
%Articulación 1
%Posición de la articulación 1 respecto a 0
P(:,:,1)= [l1*cos(th1); l1*sin(th1);a1];
%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];
%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];

%Articulación 2
```

```
%Posición de la articulación 2 respecto a 1
P(:,:,3)= [cos(th3); sin(th3);0];
%Matriz de rotación de la junta 1 respecto a 0
R(:,:,3)= [cos(th3) -sin(th3)  0;
           sin(th3)  cos(th3)  0;
           0         0         1];
```

En la tercera parte del código nos encontramos que se inicializan el vector de ceros, las matrices globales y locales, para posteriormente el rellenarlas con las matrices obtenidas anteriormente.

```
%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);
```

Dentro de este ciclo for, nosotros utilizamos las matrices obtenidas y ahora si las guardamos para poder realizar los jacobianos posteriormente.

```
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
```

Para poder calcular los jacobianos es necesario el realizar unas deruvadas con respecto a los angulos de los que se rotaron los eslabones y posteriormente los guardamos en una matriz para poder obtener un jacobiano lineal

```
%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);
Jv12= functionalDerivative(PO(1,1,GDL), th2);
Jv13= functionalDerivative(PO(1,1,GDL), th3);
%Derivadas parciales de y respecto a th1 y th2
Jv21= functionalDerivative(PO(2,1,GDL), th1);
Jv22= functionalDerivative(PO(2,1,GDL), th2);
Jv23= functionalDerivative(PO(2,1,GDL), th3);
%Derivadas parciales de z respecto a th1 y th2
Jv31= functionalDerivative(PO(3,1,GDL), th1);
Jv32= functionalDerivative(PO(3,1,GDL), th2);
Jv33= functionalDerivative(PO(3,1,GDL), th3);
%Creamos la matríz del Jacobiano lineal
jv_d=simplify([Jv11 Jv12 Jv13;
               Jv21 Jv22 Jv23;
               Jv31 Jv32 Jv33]);
```

En esta ultima parte del código obtenemos el jacobiano lineal y angular por medio del producto cruz

```
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 respect
            Jw_a(:,k)=[0,0,1];%Si no hay matriz de rotación previa se obtiene la Matriz
         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
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);
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);
```

El segundo robot es un robot cilindrico el cual cuenta con una articulación rotacional y dos articulaciones prismaticas, por ende tendremos que realizar una correción en los planos en los que planteamos nuestros ejes, debido a que tenemos que ver que estos correspondan en el dezplasamiento.

En este robot el desplazamiento se tomó dentro del eje z y los otros ejes se tomaban de referencia para poder realizar las rotaciones de los ejes, siendo que la primera articulación al ser una articulación rotacional sin eslabon esta si contaría con un desplazamiento dentro de los ejes x y y pero este no contaria con una longitud para poder tomarse en cuenta. Mientras que la articulación dos y tres estas cuentan un desplasamiento dentro de un solo eje, es por esto que se toma que el desplazamiento se hace es dentro del eje de las z y tendríamos que realizar una rotación para poder encontrar la matriz de rotación que corresponde a dicha articulación.

Explicación del código

El código de este robot funciona de manera similar al código anterior, unicamente cambia en algunos aspectos como lo son los siguientes

Dentro de la declaración de las variables unicamente se tendría que declarar una th y dos longitudes (o 3 si se toman en cuenta la pequeña superficie de la primera articulación), además de la declaración del tipo de articulación

```
%Declaración de variables simbólicas
syms th1(t) l1(t) l2(t) l3(t) t

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

%Creamos el vector de coordenadas articulares
Q= [th1, l2, l3];
%disp('Coordenadas generalizadas');
%pretty (Q);
```

Las articulaciones se colocarian de la siguiente forma

```
%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....
R(:,:,1)= [cos(th1) -sin(th1)  0;
           sin(th1)  cos(th1)  0;
           0         0         1];
```

```
%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
R(:,:,2)= [1 0 0;
           0 0 1;
           0 -1 0];


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

El ultimo cambio que podemos encontrar es en cómo se declaran los jacobianos

```
%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);
Jv12= functionalDerivative(PO(1,1,GDL), l2);
Jv13= functionalDerivative(PO(1,1,GDL), l3);
%Derivadas parciales de y respecto a th1 y th2
Jv21= functionalDerivative(PO(2,1,GDL), th1);
Jv22= functionalDerivative(PO(2,1,GDL), l2);
Jv23= functionalDerivative(PO(2,1,GDL), l3);
%Derivadas parciales de z respecto a th1 y th2
Jv31= functionalDerivative(PO(3,1,GDL), th1);
Jv32= functionalDerivative(PO(3,1,GDL), l2);
Jv33= functionalDerivative(PO(3,1,GDL), l3);
```