Instituto Tecnológico de Estudios Superiores de Monterrey

Campus Puebla

Fundamentación de Robótica (Gpo 101)

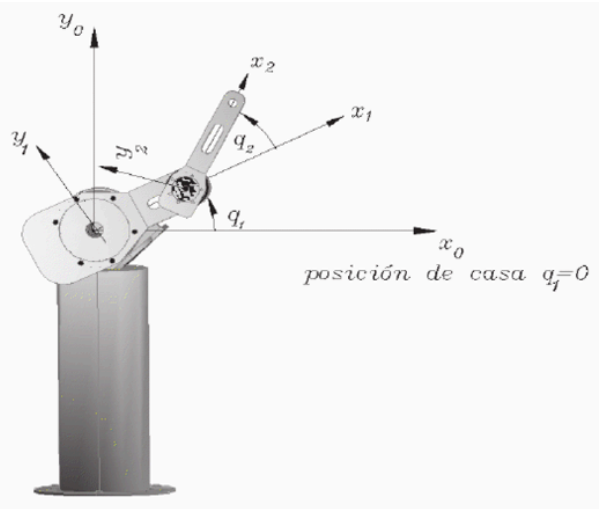
Actividad 1 (Velocidades Lineales y angulares)

Alumno
José Diego Tomé Guardado A01733345

Fecha de entrega
Viernes 23 de Febrero de 2024

Diseño de un robot planar de 2gdl

Tendremos el diseño de un robot planar con 2 grados de libertad GDL, este tipo de robot cuenta con dos articulaciones rotacionales que permiten moverse en un plano definido, cada grado de libertad que podemos observar corresponde a una de sus articulaciones que gira en torno a su propio eje.



Tiene un sistema de referencia fijo (Xo, Yo, Zo) se coloca en la base del robot, su sistema de coordenadas se considera como su punto de referencia para poder medir todas las posiciones que puede tener el robot. El eje Z es perpendicular al plano de la imagen.

Código de Matlab

Primeramente, vamos a tener los comandos para poder realizar la limpieza de la pantalla y de las variables también.

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

Tenemos la definición de las variables simbólicas donde `th1(t)` y `th2(t)` son la representación de las posiciones angulares de las articulaciones en la función del tiempo, `l1` y `l2` se definen como las longitudes de los eslabones. También vamos a tener en cuenta la configuración del robot con el vector de `RP` en que pondremos un 0 porque las juntas con las que cuenta nuestro robot son rotacionales.

```
%Declaración de variables simbólicas (no tienen valor específico)
syms th1(t) l1
syms th2(t) l2
RP=[0 0]; %configuracion del robot, 0 para junta rotacional y 1 para
prismatica
```

`Q` se define como nuestro vector de coordenadas articulares que contiene las posiciones angulares del robot que dependen del movimiento de este mismo. `th1(t) y th2(t)` son estas mismas variables de las qu3 nuestro robot depende porque son los ángulos de rotacion con los que cuenta el robot.

```
%Vector de coordenadas articulares
Q = [th1 th2];
disp('Coordenadas articulares');
pretty(Q);
```

```
Coordenadas articulares
(th1(t), th2(t))
```

Es importante tomar en cuenta las velocidades articulares para poder llegar al resultado esperado, `Qp` se va a calcular como la derivada temporal de `Q` con respecto al tiempo de *t* lo qué va a representar las velocidades angulares de las articulaciones. Se representan las velocidades angulares de las articulaciones como se muestra en la ejecución que van a depender de la derivada con respecto al tiempo de `th1(t)` y `th2(t)`

```
%vector de velocidad articulares
Qp = diff(Q,t); %se utiliza diff para derivadas cuya variable no depende de
otras
disp('Velocidades articulares');
pretty(Qp);
```

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

Definición de los grados de libertad

Podemos observar en el diagrama del robot que tiene dos juntas rotacionales las cuales significan que cuenta con dos grados de libertad en los que el robot puede mover sus articulaciones, de esta manera `GDL` se calcula como el número de columnas en nuestra matriz `RP` lo qué representa nuestro número de grados de libertad qué es de dos guardándolo en esta variable.

```
%numero de grado de libertad del robot
GDL = size(RP,2); %se coloca 2 para referirse a las columnas de la matriz
GDL_str = num2str(GDL); %convertir numero a string
```

Tendremos ahora que encontrar la matriz homogénea de transformación, haciendo un análisis podemos darnos cuenta de que solo contamos con dos articulaciones, por lo que obtenerla resulta algo sencillo. Pero una mejor idea es multiplicar todas las matrices homogéneas locales, las cuales toman como origen el extremo final de la anterior articulación y de esta forma podemos decir que tendremos dos matrices homogéneas locales.

Teniendo por consiguiente nuestros matrices de posición P dónde lo haremos de manera local por cada una de nuestras articulaciones.

```
%Articulación 1
%posicion de la junta 1 respecto a 0
P(:,:,1) = [l1*cos(th1);
           l1*sin(th1);
           0];  %vector de posicion indexado por pagina

%Articulación 2
%posicion de la junta 2 respecto a 1
P(:,:,2) = [l2*cos(th2);
           l2*sin(th2);
           0];  %vector de posicion indexado por pagina
```

También declaramos nuestras matrices de rotación `R` para cada articulación y por último tendremos un vector de ceros:

```
%matriz de rotacion de articulacion 1
R(:,:,1) = [cos(th1) -sin(th1) 0; %análisis de robot pendulo
           sin(th1) cos(th1)  0;
           0        0         1];
%matriz de rotacion de articulacion 2
R(:,:,2) = [cos(th2) -sin(th2) 0; %análisis de robot pendulo
           sin(th2) cos(th2)  0;
           0        0         1];
Vector_Zeros = zeros(1,3);
```

Inicializamos con una matriz de transformación homogenea local donde se le da la asignación con la letra `A` en la última posición `GDL` de esta matriz tridimensional `A`, qué tiene de concatenación a la matriz de rotación, la matriz de posición y una fila de ceros seguida de un 1 para poder indicar toda la matriz de transformación homogénea completa.

```
%inicializamos matriz de transformacion homogenea local
A(:,:,GDL) = simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1])
```

Aquí se inicializa la matriz de transformación homogénea global para la última articulación del robot, donde también tenemos el mismo significado de la local. Pero aquí `T` se le da a la asignación en la última posición `GDL` de esta matriz, que es también tridimensional, que también tiene la concatenación para formarla completamente.

```
%inicializamos matriz de transformacion homogenea global ()
T(:,:,GDL) = simplify([R(:,:,GDL) P(:,:,GDL); Vector_Zeros 1]);
```

Ahora se representarán los vectores de posición donde se copia la matriz de posición `P` y son vistos desde el marco de referencia inercial, también las matrices de rotación, pero copia la matriz de rotación `R` e igualmente vistas desde el marco de referencia inercial.

```
PO(:,:,GDL)= P(:,:,GDL); %vectores de posicion vistos desde el marco de
referencia inercial
RO(:,:,GDL)= R(:,:,GDL); %matrices de rotacion vistas desde el marco de
referencia inercial
```

Obtención de matrices

Primeramente, como podemos ver en este ciclo for se va a empezar iterando a través de cada grado de libertad de nuestro robot, desde el 1 hasta el total de números de los grados de libertad qué tiene, también convierte el número entero i en una cadena de caracteres. Imprime el mensaje para mostrar la matriz de transformación local y calcula la matriz de transformación homogénea local `A` compuesta de la matriz de rotación, posición y la fila de ceros, seguida de un uno para la traslación.

```
for i= 1: GDL
   i_str = num2str(i);
   %locales
   disp(strcat('Matriz de transformacion local A',i_str));
   A(:,:,i)=simplify([R(:,:,i) P(:,:,i); Vector_Zeros 1])
   pretty (A(:,:,i));
```

Ahora con el uso de try y catch vamos a poder manejar nuestro primer grado de libertad, en un caso general se calcula la matriz de transformación homogénea global `T` con la recursividad de esta misma. En el otro caso simplemente se copiará la matriz de transformación homogénea local a la matriz de transformación global `T`.

```
%globales
   try
       T(:,:,i) = T(:,:,i-1)*A(:,:,i);
   catch
       T(:,:,i) = A(:,:,i); %caso especifico cuando i= 1 nos marcaria error
en try
   end

   disp(strcat('Matriz de transformacion global T',i_str));
   T(:,:,i) = simplify(T(:,:,i));
   pretty(T(:,:,i));
```

Ejecución

- **Articulación 1:**

```
Matriz de transformacion local A1

A(:,:,1) =

[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]
```

Tenemos la primera articulación local que posteriormente vamos a observar qué la global también es la misma porque no existe una articulación anterior para que se englobe. También observamos que en nuestra matriz de rotación solo existe esta rotación en el eje z con función en el ángulo `th1(t)` porque no existen otros ángulos que estén presentes. En el vector de traslación tenemos un movimiento en x y y infiere ser por la rotación qué tiene la articulación y también están en función de `th1(t)` porque hablamos de la primera articulación y solo engloba ese ángulo.

```
Matriz de transformacion 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       /

/ cos(th1(t)), -sin(th1(t)), 0 \
|                              |
| sin(th1(t)),  cos(th1(t)), 0 |
|                              |
\      0,            0,      1 /

/ l1 cos(th1(t)) \
|                |
| l1 sin(th1(t)) |
|                |
\        0       /
```

- **Articulación 2:**

```
Matriz de transformacion local A2

A(:,:,1) =

[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 transformacion 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))
```

```
/ cos(th1(t) + th2(t)), -sin(th1(t) + th2(t)), 0 \
|                                                 |
| sin(th1(t) + th2(t)),  cos(th1(t) + th2(t)), 0 |
|                                                 |
\          0,                    0,           1 /

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

Ahora para la segunda articulación vamos a ver que ya no coincide la matriz de transformación local con la global y es correcto debido a que la global está considerando todas las variables del sistema del robot mientras qué la local solo con el ángulo de `th2(t)` qué corresponde únicamente para esa articulación. La matriz de rotación nos indica que el movimiento rotacional será en `z, pero` ya tomando en cuenta el ángulo rotacional qué teníamos en la articulación 1, por tanto, en el vector de traslación se observa qué el movimiento es en `x` y en `y`, recordando que se tienen las variables de la articulación anterior en cuenta para todo el análisis completo del sistema.

Por último se va a extraer la matriz de rotación `RO` y vector de traslación `PO` de la matriz de transformación homogénea global, las matrices y vectores hacen la representación de la rotación y también de la traslación de lo que es el extremo del robot con respecto al marco de referencia inercial.

```
    %obtenemos la matriz de rotacion RO y el vector de translacion PO de la
    %matriz de transformacion homogenea global T(:,:,GDL)
    RO(:,:,i) = T(1:3,1:3,i);
    PO(:,:,i) = T(1:3,4,i);
    pretty(RO(:,:,i));
    pretty(PO(:,:,i));
end
```

Obtención del Jacobiano de la matriz

- **Forma diferencial**

Para poder llegar al resultado de las velocidades lineales y angulares debemos tener en cuenta y calcular de forma correcta los jacobianos lineal y angular respectivamente.
Con el jacobiano lineal se van a calcular las derivadas parciales de las coordenadas de la posición qué tiene `PO` con respecto a las posiciones angulares qué tenemos de `th1` y `th2`. Lo qué funciona para qué estas derivadas parciales sirvan para construir nuestro jacobiano lineal, qué tiene la relación de las velocidades lineales del extremo de nuestro robot con las otras velocidades qué son angulares de las articulaciones.

```
%calculamos el jacobiano lineal de forma diferencial
disp('Jacobiano lineal obtenido de forma diferencial');
%derivadas parciales de x con respecto a th1 y th2
Jv11= functionalDerivative(PO(1,1,GDL), th1);
Jv12= functionalDerivative(PO(1,1,GDL), th2);
%derivadas de y con respecto a th1 y th2
Jv21= functionalDerivative(PO(2,1,GDL), th1);
Jv22= functionalDerivative(PO(2,1,GDL), th2);
%parciales de z con respecto a th1 y th2
Jv31= functionalDerivative(PO(3,1,GDL), th1);
Jv32= functionalDerivative(PO(3,1,GDL), th2);
jv_d = simplify([Jv11 Jv12; ...
                 Jv21 Jv22 ; ...
```

```
                Jv31 Jv32]);
pretty(jv_d)
```

Ejecución

```
Jacobiano lineal obtenido de forma diferencial
/ - 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             /
```

- **Forma análitica**

Ahora para la forma analítica, primeramente se van a inicializar nuestros jacobianos analíticos para las velocidades angulares y lineales con las posiciones del efector final del robot en la última columna de los grados de libertad y se empezará iterando a través de cada grado de libertad del robot.

Dentro del bucle va a verificar si la articulación es rotacional 0 o si es prismática 1:

Rotacional: Se calcula el jacobiano lineal utilizando el producto cruz entre el vector de eje de rotación con la resta de las posiciones del extremo final del robot y la articulación actual y para el jacobiano angular solo es copiar nuestro vector de eje de rotación.

Prismática: En este caso el jacobiano lineal solo va a depender del vector de eje de rotación `RO` lo qué significa que es el tercer columna de la matriz de rotación anterior, y para el jacobiano angular se establece en ceros porque en este caso se identifica como un vector de ceros.

Finalmente, los jacobianos se simplifican gracias a la función de `simplify` y `pretty` para que se vean de mejor forma y se imprimen en la consola.

```
%Calculamos el jacobiano lineal y angular de forma analitica
%inicializamod jacobianos analiticos (lineal y angular)
Jv_a(:,GDL)=PO(:,:,GDL);
Jw_a(:,GDL)=PO(:,:,GDL);
for k=1:GDL
   if((RP(k)==0)|(RP(k)==1))
       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
   else
```

```
        %articulaciones prismaticas
        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);
```

Ejecución

```
Jacobiano lineal obtenido de forma analítica
/ - 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           /

Jacobiano ángular obtenido de forma analítica
/ 0, 0 \
|      |
| 0, 0 |
|      |
\ 1, 1 /
```

Podemos indicar que tendremos el mismo jacobiano lineal que obtuvimos de forma diferencial por lo que estamos en un resultado correcto porque se debe de obtener el mismo jacobiano, ya que solo cambia la forma de obtenerlo, podemos afirmar que el movimiento solo será en el eje x y y pero un movimiento 0 en el eje z. El jacobiano angular solo indica un movimiento en el eje z, manteniendo x y y con cero, qué posteriormente podremos observar cómo esto nos va guiando cada vez más al resultado de las velocidades lineales y angulares

Resultados de Velocidades lineales y angulares

Gracias a todos los cálculos anteriores de los jacobianos, podremos obtener los resultados de las velocidades, para la velocidad lineal se obtiene del producto del jacobiano lineal `Jv_a` y las velocidades articulares `Qp`. Para la velocidad angular se obtiene del producto del jacobiano angular `Jw_a` y las velocidades articulares `Qp`. El `Qp'` va a indicar transposición de la matriz de velocidades porque se esperan las velocidades articulares como una columna.

```
disp('Velocidad lineal obtenida mediante el Jacobiano Lineal');
V=simplify (Jv_a*Qp');
pretty(V);
disp('Velocidad angular obtenido mediante el Jacobiano angular')
W=simplify (Jw_a*Qp');
pretty(W);
```

Ejecución

```
Velocidad lineal obtenida mediante el Jacobiano Lineal
/   _________                                  _________     \
|    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))
```

```
Velocidad angular obtenido mediante el Jacobiano angular
/            0            \
|                          |
|            0             |
|                          |
| __________   __________  |
|  d            d          |
| -- th1(t) + -- th2(t)    |
\ dt           dt          /
```

Observamos finalmente nuestros resultados, teniendo el vector de velocidades lineales que nos dice que sí existe el movimiento en `x` y `y` pero el movimiento en `z` se encuentra en cero, no teniendo velocidad lineal en z. Caso contrario de la velocidad angular que se observa que tenemos cero en los ejes de `x` y `y` pero si encontramos con que tenemos velocidad angular en el eje `z` que será la suma de las derivadas con respecto a *t* de las posiciones angulares de las articulaciones en la función del tiempo `th1(t)` y `th2(t)`, indicando rotación solo en `z`.

Se representan correctamente las velocidades tanto lineales como angulares en función de las velocidades articulares, son completamente importantes para poder comprender cómo se mueve el extremo del robot con repuestos a los cambios de posición de sus articulaciones.