4º GIERM

Control y Programación de Robots

trabajo de control de robots manupuladores

Álvaro Calvo matos

# 1. Análisis Cinemático y Dinámico.

El objetivo de esta parte de la asignatura es el de profundizar un poco más en lo que ya se empezó a ver el año pasado en la asignatura de Fundamentos de Robótica donde se simuló el comportamiento de un brazo robótico para su posterior control.

Esta vez se intentarán simular problemas que tendríamos en un robot real tales como medidas imperfectas o ruidosas o desconocimiento inicial de algunos de los parámetros dinámicos del robot, que tendrán que ser estimados para desarrollar un modelo dinámico a partir del cual poder diseñar controles.

### 1.1. Cinemática directa e inversa.

La obtención de las cinemáticas directa e inversa del robot es materia del curso pasado, así que no se profundizará en la explicación de su obtención.

A partir del esquema del robot, mediante el algoritmo de Denavit-Hartenberg, se obtienen los parámetros homónimos con los que se pueden generar las matrices de transformación homogéneas que trasladan las coordenadas del extremo del robot al sistema de referencia de la base.

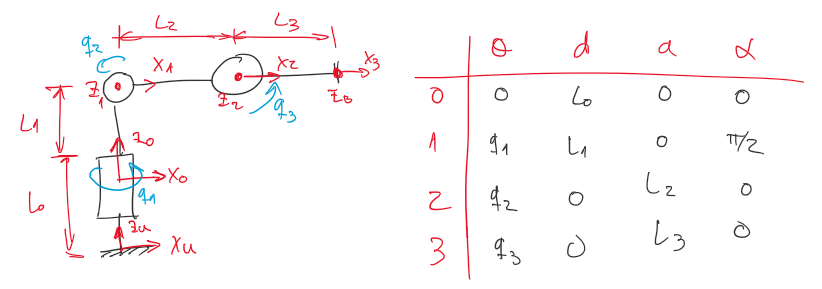


Figura 1. Parámetros de Denavit-Hartenberg.

Con la ayuda del Robotics-Toolbox, se generan las MTH, obteniendo la cinemática directa del robot, implícita en el producto de las 4 matrices de transformación homogéneas.

syms L0 L1 L2 L3 q1 q2 q3 real;

PI = sym('pi'); %para que el numero pi sea exacto

AU0 = MDH(0, L0, 0, 0);

A01 = MDH(q1, L1, 0, PI/2);

A12 = MDH(q2, 0, L2, 0);

A23 = MDH(q3, 0, L3, 0);

T = simplify(AU0\*A01\*A12\*A23)

El modelo cinemático directo resultante es:

x = cos(q1)\*(L3\*cos(q2 + q3) + L2\*cos(q2));

y = sin(q1)\*(L3\*cos(q2 + q3) + L2\*cos(q2));

z = L0 + L1 + L3\*sin(q2 + q3) + L2\*sin(q2);

Para resolver el problema de la cinemática inversa podemos usar también alguno de los procedimientos vistos el curso pasado. Aquí se ha optado por su resolución mediante métodos geométricos.

Se obtiene:

q1 = atan2(y,x);

C3 = (x^2+y^2+(z-L0-L1)^2-L2^2-L3^2)/(2\*L2\*L3);

q3 = atan2(sqrt(1-C3^2),C3);

if q3 < 0

q3 = atan2(-1\*sqrt(1-C3^2),C3);

end

q2 = atan2((z-L0-L1),sqrt(x^2+y^2))-atan2((L3\*sin(q3)),(L2+L3\*cos(q3)));

q = [q1 q2 q3]';

La función que cumple la sentencia condicional que se observa en el código es la de decidir que con que configuración alcanzará el robot el punto en cartesianas que le indiquemos.

Sin estas tres líneas de código podemos tener complicaciones en bloques posteriores que requieran de la cinemática inversa tales como el GTCL.

### 1.2. Ecuaciones dinámicas del robot.

La obtención del modelo dinámico del robot se lleva a cabo de la misma forma que en la asignatura de Fundamentos de Robótica del curso pasado.

Se ha partido del algoritmo de Newton-Euler recursivo que se usó el pasado curso, adaptándolo a la morfología del robot actual y modificándolo para su funcionamiento con variables simbólicas.

Los resultados de aplicar el mencionado algoritmo son las fuerzas y/o pares en cada articulación.

Este año los transformaremos en las intensidades eléctricas que circulan en cada motor multiplicando por la constante de par de los motores y por las reductoras, pasando a trabajar con intensidades eléctricas como señal de control.

El código del que se habla se encuentra en el fichero “GAMMA\_REDUCIDA.m”.

Llegados a este punto, dado que los resultados obtenidos dependen de parámetros dinámicos desconocidos del robot, se reescriben las ecuaciones para facilitar la posterior estimación de estos parámetros.

# 3. Control cinemático.

El objetivo del control cinemático es el de establecer la trayectoria que debe seguir cada articulación para que el extremo del robot describa el movimiento deseado en coordenadas cartesianas, concretando para ello tanto posiciones, como velocidades y aceleraciones articulares en cada instante de tiempo.

De entre las distintas especificaciones de movimiento podríamos mencionar el movimiento punto a punto, en el que el trayecto intermedio nos sería indiferente, movimiento lineal, movimiento circular…, especificaciones de tiempo, velocidad, etc.

Es importante tener en cuenta las limitaciones físicas del robot a la hora de establecer referencias de velocidad o bien saber si la trayectoria pedida es realizable con la configuración del brazo de la que se dispone.

En el bloque de control cinemático se ha introducido un sencillo código que comprueba la accesibilidad de la trayectoria. La configuración del brazo genera un espacio de trabajo comprendido entre dos esferas de radios y dentro de los cuales cualquier trayectoria es válida. Dado que el generador de trayectorias que se programará solo genera trayectorias lineales para el extremo del robot, este problema se reduce a comprobar tres condiciones, que, si se cumplen, aseguran que el recorrido es alcanzable físicamente por el brazo robótico con el que se está trabajando.

* El punto inicial debe estar fuera de la esfera de menor tamaño, pero dentro de la mayor.
* El punto final debe estar fuera de la esfera de menor tamaño, pero dentro de la mayor.
* El punto más cercano de la trayectoria al centro de las esferas debe estar dentro del espacio de trabajo. Esto se calcula mediante álgebra vectorial, a través de proyecciones.

if (L2-L3)^2 <= (XYZinicio(1)^2+XYZinicio(2)^2+(XYZinicio(3)-L0-L1)^2) && (XYZinicio(1)^2+XYZinicio(2)^2+(XYZinicio(3)-L0-L1)^2) <= (L2+L3)^2

if (L2 - L3)^2 <= (XYZfin(1)^2 + XYZfin(2)^2 + (XYZfin(3) - L0 - L1)^2) && (XYZfin(1)^2 + XYZfin(2)^2 + (XYZfin(3) - L0 - L1)^2) <= (L2 + L3)^2

% si los dos extremos de la recta están en el espacio de trabajo

% comprobamos si también lo está la recta que los une

C = [0; 0; L0 + L1]; % Centro de las esferas

u = (XYZfin - XYZinicio)'/norm(XYZfin - XYZinicio); % Vector unitario en direccion de la trayectoria

PiC = C - XYZinicio; % Vector de XYZinicio a C

Proy = ((u\*PiC)/(u\*u'))\*u; % Proyeccion del vector PiC en direccion de u

distancia = norm(PiC - Proy'); % Menor distancia de C a la trayectoria

if distancia >= (L2 - L3)

% La recta está dentro del espacio de tarea.

disp('La trayectoria fijada es VALIDA')

else

disp('La trayectoria fijada NO es valida. Atraviesa zonas no alcanzables')

end

else

% Error: recta no valida

disp('La trayectoria fijada NO es valida. El punto final no es alcanzable')

end

else

% Error: recta no válida

disp('La trayectoria fijada NO es valida. El punto inicial no es alcanzable')

end

### Generador de trayectorias cartesianas punto a punto

Lo primero que se pide es desarrollar un generador de trayectorias cuyas únicas especificaciones sean los puntos inicial y final, siendo válida cualquier tipo de trayectoria que pueda realizar el extremo del brazo para alcanzar el punto final desde la posición inicial.

Este apartado se presenta como paso previo o simplificación del objetivo real de esta parte del proyecto: desarrollar un generador de trayectoria cartesiano lineal para el robot de tres grados de libertad (GTCLR3GDL).

A pesar de ello, hemos desarrollado desde un inicio el generador de trayectorias cartesiano lineal, ya que no considerábamos que supusiera mucha dificultad. Esto no significa sin embargo que no se haya obtenido un generador de trayectorias punto a punto, ya que basta con especificar un número de puntos intermedios igual a cero, bien desde fuera, o bien fijando la condición desde el interior del propio código, para obtener este generador de trayectorias a partir de el de mayor complejidad.

En la siguiente figura se demuestra que se posee la capacidad de generar trayectorias punto a punto.



Figura 2. Trayectoria punto a punto generada.

Las referencias de posiciones, velocidades y aceleraciones articulares en el tiempo para recorrer esta trayectoria:



### Generador de trayectorias cartesianas lineal.

El funcionamiento del generador de trayectorias cartesiano lineal al que se ha llegado tras el desarrollo puede dividirse en X partes bien diferenciadas:

* Estudio de la trazabilidad de la trayectoria especificada: Se realiza tal y como se ha descrito en la introducción de este apartado, comprobando que la trayectoria cumpla las tres condiciones.
* Generación y muestreo de la ecuación que describe la trayectoria en coordenadas cartesianas respecto al tiempo a partir de los puntos inicial y final. Primero se obtiene la ecuación de la recta en cartesianas, teniendo en cuenta el tiempo de inicio y la duración del movimiento; y luego se muestrea la ecuación obtenida, extrayendo el número de puntos intermedios en cartesianas especificados N.
* Conversión de los N puntos intermedios y los dos extremos a coordenadas articulares por medio de la Cinemática Inversa. Se toman también los tiempos de inicio de cada tramo para uso futuro.

Estos dos últimos puntos han pasado por un proceso de optimización, fusionándose y resultando en un menor numero de operaciones cuyo fragmento de código es el siguiente.

q = zeros(3, n + 2);

t\_tramos = zeros(1, n + 2);

pendiente = (XYZfin - XYZinicio)/duracion;

for punto = 0:(n+1)

aux = (pendiente\*T\*punto + XYZinicio);

q(:, punto+1) = CinematicaInversa(aux);

% Calculamos los tiempos a los que comienza cada tramo

t\_tramos(punto+1) = inicio + T \* punto;

end

* Aplicación del método heurístico para obtener las velocidades de paso por cada punto. Se deciden las referencias en velocidad para cada uno de los N puntos intermedios observando el signo de la velocidad en los tramos anterior y posterior a cada uno. De esta forma, si los signos de las velocidades en los tramos coinciden, se establece la velocidad de paso por dicho punto como la media de las dos. Si por el contrario las velocidades tienen signos contrarios, se fija la velocidad de paso a cero. Las velocidades de los puntos inicial y final también se fijan a cero.

qd = zeros(3, n + 2);

qd(:, 1) = [0 0 0]';

qd(:, n + 2) = [0 0 0]';

for i = 2:(n + 1)

if (sign(q(:,i) - q(:,i-1)) ~= sign(q(:,i+1) - q(:,i)))

qd(:,i) = [0 0 0]';

else

qd(:,i) = [((q(:,i+1)-q(:,i))/T+(q(:,i)-q(:,i-1))/T)/2];

end

end

* Interpolación de cada par de puntos con un polinomio de orden tres. Se calculan y almacenan los coeficientes que definen los polinomios de cada tramo para que no se necesite realizar más de una sola vez estos cálculos.

a = zeros(3, n + 1);

b = zeros(3, n + 1);

c = zeros(3, n + 1);

d = zeros(3, n + 1);

for i = 1:(n+1)

a(:, i) = q(:,i);

b(:, i) = qd(:,i);

c(:, i) = 3/T^2\*(q(:,i+1)-q(:,i))-1/T\*(qd(:,i+1)+2\*qd(:,i));

d(:, i) = -2/T^3\*(q(:,i+1)-q(:,i))+1/T^2\*(qd(:,i+1)+qd(:,i));

end

* Identificación del tramo desde el que se ha llamado a la función. Es necesario conocer el tramo actual para poder evaluar el polinomio correspondiente y devolver las referencias adecuadas. Este, junto con los posteriores cálculos para evaluar el tiempo actual en el tramo correspondiente, serán los únicos cálculos a realizar en sucesiones llamadas al programa para obtener la siguiente referencia en posición, velocidad y aceleración articulares.

% Calculos a realizar una vez por llamada a la función:

% Comprobamos en que tramo ha sido llamada la funcion

if t <= inicio

% t < tiempo de inicio (reposo inicial) --> ref = q\_inicio

trayectoria = [q(:, 1); [0 0 0]'; [0 0 0]'];

elseif t >= inicio + duracion

% t > tiempo de inicio + duracion (reposo final) --> ref = q\_fin

trayectoria = [q(:, n + 2); [0 0 0]'; [0 0 0]'];

else

% Hay que detectar en que tramo nos encontramos

for i = 1:(n+1)

if t\_tramos(i) <= t && t < t\_tramos(i+1)

tramo = i;

break

end

end

% Evaluamos la ecuacion para el tramo y tiempo actual

% q = a + b\*(t - ti) + c\*(t - ti)^2 + d\*(t - ti)^3

% qd = b + c\*(2\*t - 2\*ti) + 3\*d\*(t - ti)^2

% qdd = 2\*c + 3\*d\*(2\*t - 2\*ti)

q\_t = a(:,tramo)+b(:,tramo)\*(t-t\_tramos(tramo))+c(:,tramo)\*

(t-t\_tramos(tramo))^2+d(:,tramo)\*(t-t\_tramos(tramo))^3;

qd\_t = b(:,tramo)+c(:,tramo)\*(2\*t-2\*t\_tramos(tramo))+3\*d(:,tramo)\*

(t-t\_tramos(tramo))^2;

qdd\_t = 2\*c(:,tramo)+3\*d(:,tramo)\*(2\*t-2\*t\_tramos(tramo));

trayectoria = [q\_t; qd\_t; qdd\_t];

end

Hay diversas formas de mejorar este algoritmo para hacerlo más eficiente, algunos de estos métodos son el uso de memoria persistente, mencionado implícitamente en la descripción del algoritmo, o la inclusión de la cinemática inversa en el código para así prescindir de llamadas a funciones externas que ralenticen la ejecución.

Cabe recalcar que la implementación de este mismo algoritmo en algún lenguaje de programación compilado, y por tanto ejecutable a nivel de procesador, será en principio mucho más rápido que otro implementado en un script de Matlab.

El código completo del generador de trayectorias se encuentra en el archivo adjunto de nombre y extensión “GTCL\_R3GDL.m”.

Las gráficas que se obtienen como resultado de ejecutar el presente código con unas especificaciones de:

|  |  |
| --- | --- |
| Tiempo de simulación | 2 s |
| Punto inicial em cartesianas (XYZinicial) | CinematicaDirecta([0 0 0]) |
| Punto final en cartesianas (XYZfinal) | CinematicaDirecta([1 1 1]) |
| Número de puntos intermedios (N) | 5 |
| Tiempo de inicio del movimiento (t\_ini) | 0.5 s |
| Tiempo de finalización del movimiento (t\_fin) | 1 s |

Son:





Aumentando el número de puntos de la trayectoria se pueden lograr mejores resultados en cuanto a posición, pero por el contrario se podrían necesitar unas referencias de aceleración o velocidad muy exigentes para las que necesitaríamos un modelo y control muy buenos.

Durante la prueba de clase se presenció un comportamiento anómalo del generador de trayectorias. Este comportamiento parecía no deberse al propio generador de trayectorias, ya que solo parecía ocurrir con determinadas trayectorias. El problema fue atribuido pues a esa trayectoria en particular.

Tras examinarlo más detenidamente se llegó a la conclusión de que era un error presente en la cinemática inversa, y es que, a pesar de ser correcta la solución obtenida para el problema, no se tomó ninguna decisión respecto a que configuración tomar. Esto provocaba un cambio de configuración durante la trayectoria que, al ser muestreada y reconstruida, no se veía claramente como una discontinuidad en las trayectorias articulares.





Esta es una de las complicaciones a las que se hace referencia en el primer punto de la memoria. La solución en cuanto a código podemos observarla ahí.

La idea tras la solución es la de fijar por defecto una de las dos configuraciones posibles, ya que se especifica muy concretamente cuales deben ser las entradas y salidas del bloque del GTCL. En caso de disponer de flexibilidad, se habría dado la opción a decidir la configuración del brazo. Para ello simplemente se habría cambiado la condición del bucle “if” presente en la cinemática inversa por el valor (0 ó 1) de un parámetro externo.