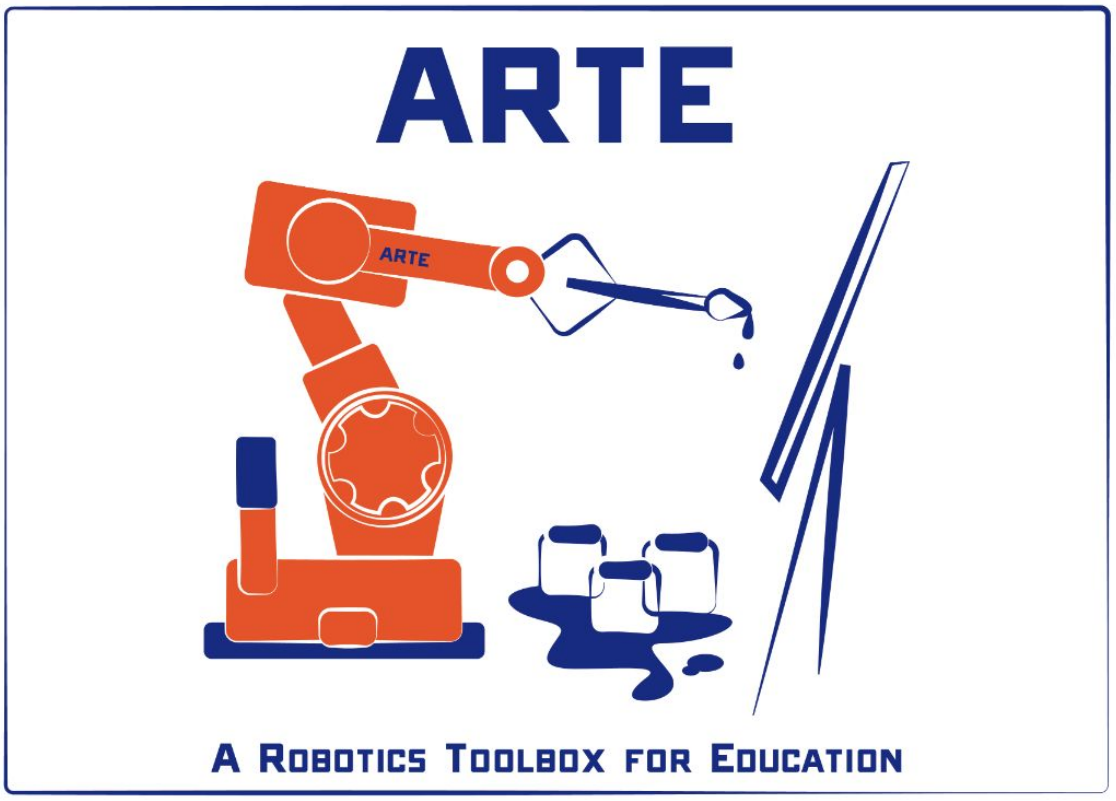
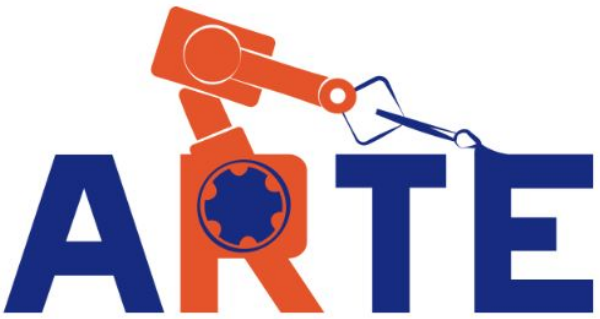


SESIÓN PRÁCTICA 2: CINEMÁTICA INVERSA

Arturo Gil Aparicio

arturo.gil@umh.es



ÍNDICE

OBJETIVOS

Objetivos generales: Después de esta sesión práctica, el estudiante debería ser capaz de:

- Resolver las ecuaciones de la cinemática inversa de cualquier manipulador de tipo serie hasta 6GDL y con muñeca esférica.
- Ser capaz de escribir estas ecuaciones en el entorno Matlab para un robot de su elección.
- Poder generar trayectorias articulares que hagan que el extremo siga un movimiento rectilíneo en el espacio cartesiano.
- Analizar puntos singulares en base a la matriz Jacobiana.

1 Primeros pasos

Nótese que la Cinemática plantea el estudio del movimiento de los cuerpos. Cuando hemos planteado la cinemática directa de los robots de tipo serie, hemos hablado, hasta ahora, de posición y orientación. Por tanto, siendo detallistas, deberíamos haber hablado de la "cinemática directa de la posición y orientación del robot de tipo serie". Hablaremos, también, en otras sesiones prácticas de la cinemática directa en velocidad, la cual plantea el estudio de la velocidad lineal y angular del robot como función de las velocidades articulares. Con las mismas ideas en mente, abordaremos en esta sesión el estudio de la cinemática inversa del robot. La solución inicial planteará el cálculo de las posiciones articulares como función de la posición y orientación del extremo del robot. Igualmente, hablaremos también en futuras lecciones de la cinemática inversa en velocidad (cuando nos interese calcular las velocidades articulares que permiten al robot alcanzar una determinada velocidad lineal/angular.

Comenzaremos con un análisis de un robot SCARA representado en la Figura 1. A continuación, se especifica la tabla de parámetros DH de este robot:

	Theta	**D**	**A**	**Alfa**
1	q_1	l_1=0.5m	l_2=0.5m	0
2	q_2	0	l_3=0.3m	π
3	0	q_3	0	0
4	q_4	0	0	0

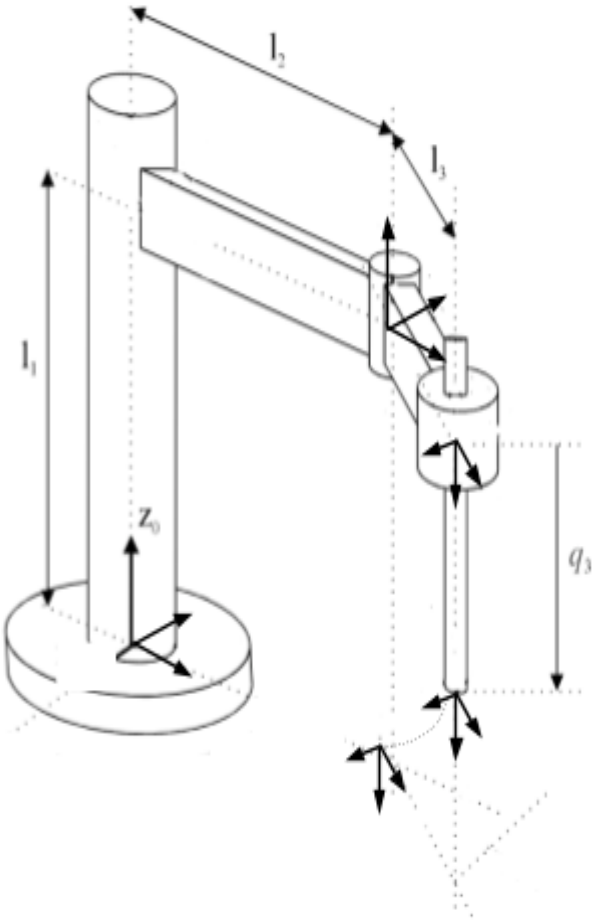


Figura 1

Para comenzar, vamos a calcular la posición y orientación del extremo efector como función de las coordenadas articulares. Para ello, teclee los siguientes comandos en Matlab:

```
>> init_lib
>> robot=load_robot('example','scara')
>> T=directkinematic(robot, [0 0 0.1 0])

T =

    1.0000         0         0    0.8000
         0   -1.0000   -0.0000   -0.0000
         0    0.0000   -1.0000    0.4000
         0         0         0    1.0000
>> drawrobot3d(robot, [0 0 0.1 pi/4])
```

Este robot tiene una representación alámbrica que se observa en la figura 2.

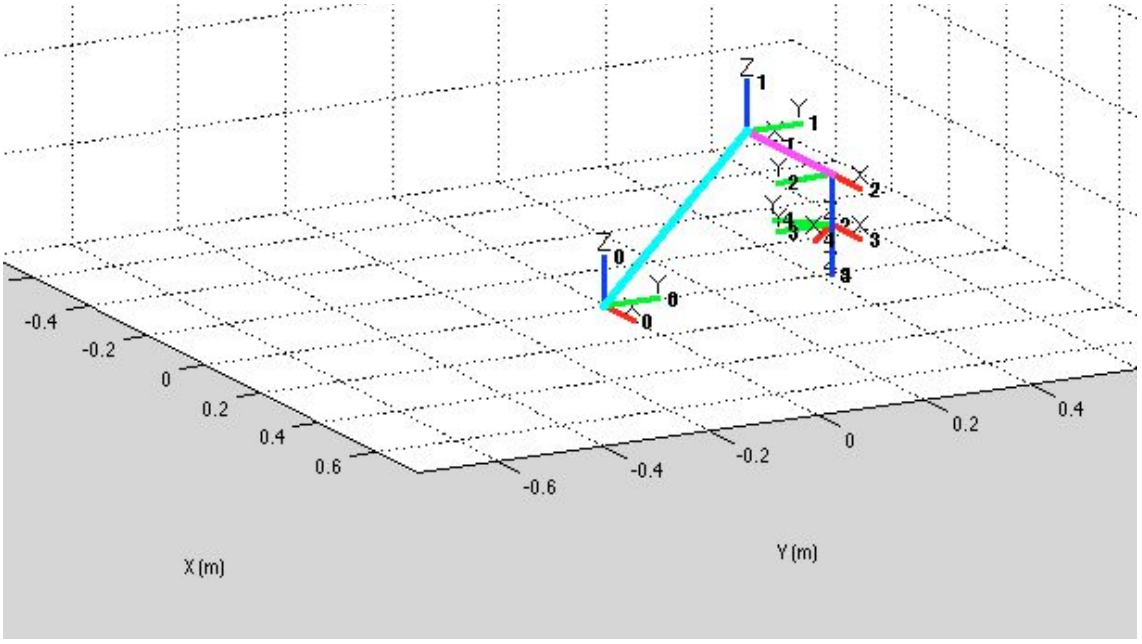


Figura 2

2 El problema cinemático inverso

En la mayoría de aplicaciones robóticas, la posición (y orientación) de:

- una pieza que hay que recoger.
- un punto de soldadura que hay que realizar.
- una caja que hay que paletizar,

son conocidas. Así pues, muchas veces estaremos interesados en calcular los ángulos de las articulaciones que permiten que el extremo del robot se encuentren en una posición y orientación determinadas. Dependiendo del robot, existirá más de una solución posible para conseguir esto:

```
>> pwd

ans =
/Users/arturogilaparicio/Desktop/arte

>> init_lib
>> T=directkinematic(robot, [0 0 0 0])
>> qinv = inversekinematic(robot, T)
Computing inverse kinematics for the Scara example 4GDL arm robot
qinv =

     0     0
     0     0
     0     0
     0     0
>> T=directkinematic(robot, [pi/2 pi/4 0.1 pi/8])
>> qinv = inversekinematic(robot, T)
Computing inverse kinematics for the Scara example 4GDL arm robot
qinv =
```

```
    2.1498    1.5708
   -0.7854    0.7854
    0.1000    0.1000
   -0.5991    0.3927
```

Nótese que en el código anterior, después de inicializar la librería, calculamos la cinemática directa para la posición inicial del robot `q=[0 0 0 0]=[q1 q2 d3 q4]`. Nótese que la tercera coordenada articular es prismática y está dada en metros. La llamada a

```
qinv = inversekinematic(robot, T)
```

requiere dos parámetros. El primero almacena los parámetros del robot (variable robot), mientras que la segunda especifica, mediante una matriz homogénea, la posición y orientación del extremo. La función `inversekinematic(robot, T)` es común a todos los robots de la librería para facilitar su uso, sin embargo, dicha función llama internamente a otra función llamada `inversekinematic_scara` que es particular al robot SCARA y que se encuentra en el directorio `arte/robots/example/scara`. El nombre de la función de cinemática inversa se especifica en el fichero parameters.m, en concreto en la línea:

```
robot.inversekinematic_fn = 'inversekinematic_scara(robot, T)'
```

Se indica, a continuación, un extracto del fichero parameters.m.

```
function robot = parameters()

%Path where everything is stored for this robot
robot.path = 'robots/example/scara';

%Kinematic parameters
robot.DH.theta= '[q(1) q(2) 0 q(4)]';
robot.DH.d='[0.5  0      q(3)  0]';
robot.DH.a='[0.5    0.3  0    0]';
robot.DH.alpha= '[ 0 pi  0 0]';

%Jacobian matrix. Variation of (X, Y, Z) as a function of (w1, w2, w3)
robot.J='[-a(2)*sin(q(1)+q(2))-a(1)*sin(q(1)) -a(2)*sin(q(1)+q(2)) 0;
a(2)*cos(q(1)+q(2))+a(1)*cos(q(1)) a(2)*cos(q(1)+q(2)) 0; 0 0 -1];';

robot.name='Scara example 4GDL arm';

robot.inversekinematic_fn = 'inversekinematic_scara(robot, T)';
```

Es importante tener en cuenta que la función de cinemática inversa devuelve dos posibles soluciones para `qinv`. Únicamente una de las dos coincide con el valor inicial de q que se especifica en la función de cinemática directa. Sin embargo, llamar a la cinemática directa del robot con cualquiera de las dos soluciones de `qinv` da, como resultado, la misma matriz T de posición y orientación del extremo (puesto que ambas son soluciones a la cinemática

inversa). Podemos comprobar estas ideas con el código siguiente:

```
>> T=directkinematic(robot, [pi/2 pi/4 0.1 pi/8])

T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827   -0.0000    0.7121
   -0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000
>>qinv = inversekinematic(robot, T)

>>T=directkinematic(robot, qinv(:,1))

T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827   -0.0000    0.7121
   -0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000

>> T=directkinematic(robot, qinv(:,2))

T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827    0.0000    0.7121
    0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000
```

Se pueden representar ambas soluciones y llamamos a la función `drawrobot3d` con el resultado de la cinemática inversa qinv:

```
>> drawrobot3d(robot, qinv(:,1))
>> drawrobot3d(robot, qinv(:,2))
```

3 Solución de la cinemática inversa para un robot de 6GDL

Se presenta, en la figura 3, un robot ABB IRB 140. Seguidamente, se indica su tabla de parámetros DH.

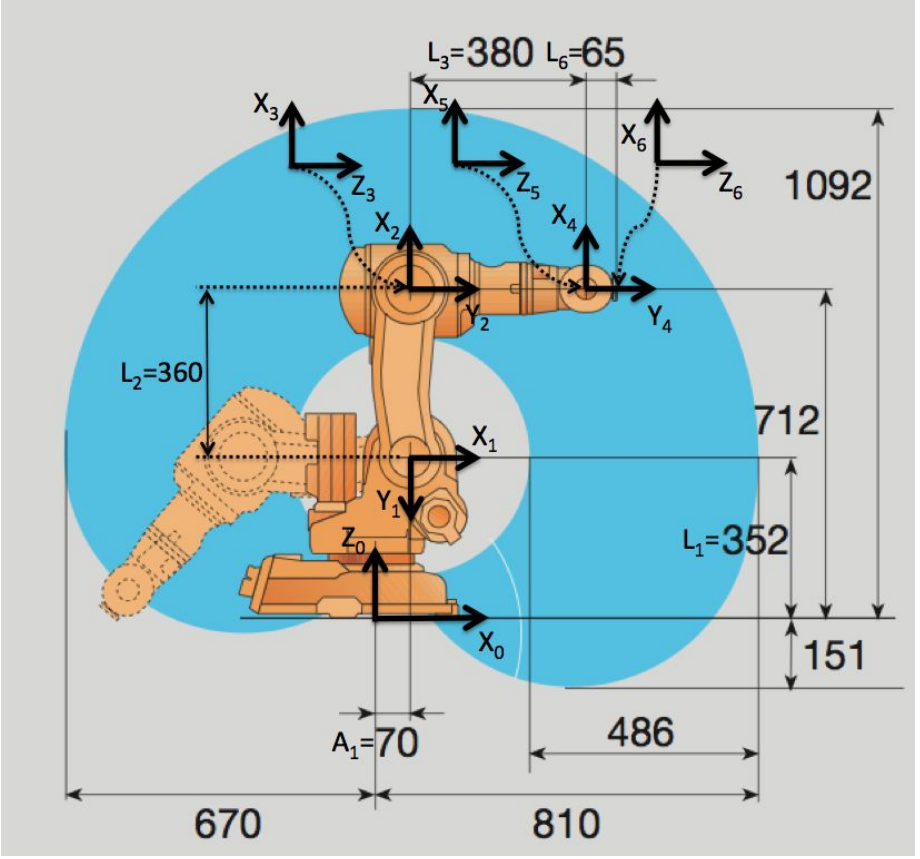


Figura 3

θ	d	a	α
q_1	0.352	0.070	$-\pi/2$
$q_2 - \pi/2$	0	0.360	0
q_3	0	0	$-\pi/2$
q_4	0.380	0	$\pi/2$
q_5	0	0	$-\pi/2$
$q_6+\pi$	0.065	0	0

Estos parámetros están definidos en el fichero `parameters.m` que se encuentra en `arte/robots/ABB/IRB140`

```
robot.DH.theta= '[q(1) q(2)-pi/2 q(3) q(4) q(5) q(6)+pi]';
robot.DH.d='[0.352 0 0 0.380 0 0.065]';
robot.DH.a='[0.070 0.360 0 0 0 0]';
robot.DH.alpha= '[-pi/2 0 -pi/2 pi/2 -pi/2 0]';
robot.J=[];
```

```
robot.inversekinematic_fn = 'inversekinematic_irb140(robot, T)';
```

Nótese que no existe una solución cerrada y universal para la cinemática inversa de todos los brazos manipuladores. Así pues, es necesario calcular las ecuaciones específicas de este robot llamando internamente a la función `inversekinematic_irb140(robot, T)` que se encuentra en el directorio `arte/robots/ABB/IRB140`.

Ejercicio 1:

Analiza el problema cinemático para el robot IRB140. Para este robot, en general, existen 8 soluciones diferentes de los valores articulares que permiten llevar el extremo del robot a una determinada posición/orientación. Para conseguir esto deberías. **Escribe un script que realice de forma automática todas las funciones que se indican a continuación**.

a) Obtener una matriz T con la cinemática directa y q = [pi/4 pi/8...] a su elección.

b) Obtener las soluciones qinv de la cinemática inversa.

c) Comprobar que todas las anteriores soluciones permiten llegar a la misma matriz T.

d) ¿Qué más deberíamos comprobar para que una solución q_i sea válida, para i=1...8?

Ejercicio 2:

a) Utiliza el script `test_kinematics_demo`. Para ello, deberás cargar un robot de 6GDL de la librería y lanzar el script. Observa el resultado por pantalla.

b) A continuación, edita el script y lee el código. Deberás ser capaz de entender el código y relacionarlo con los conceptos de cinemática directa e inversa.

c) Pruébalo con varios robots.

4 Seguir una línea recta en el espacio

Seguir una línea en el espacio de trabajo del robot es una tarea habitual para un robot de tipo serie. Esta afirmación se justifica pensando que los humanos (o, al menos, los ingenieros) tienden a resolver las tareas de manipulación de un robot uniendo los diferentes target points con líneas rectas. Este tipo de

movimiento no es, de ninguna manera, óptimo en términos de consumo energético, sin embargo, existen multitud de aplicaciones que requieren alta precisión en el seguimiento de rectas por parte del robot. Son aplicaciones típicas:

- **Soldadura de líneas:** Soldadura de, por ejemplo, dos chapas de acero sobre la línea que las une, utilizando soldadura TIG.

- **Soldadura por puntos:** O bien soldadura por puntos extraídos de la misma recta. Este tipo de soldadura es típica en la industria de manufactura de vehículos automóviles.

Ejercicio 5:

Implementa una función como la siguiente:

function q=followline_myrobot(robot, p1, p2, R, options)

La función debería ser capaz de hacer que el robot siga una línea recta en el espacio cartesiano. La función debe interpolar un conjunto de puntos pertenecientes a una recta que conecta los puntos p1 y p2. La orientación debe ser constante y definida por la matriz de rotación R. Tenga Ud. en cuenta que, para cada uno de esos puntos definidos deberá encontrar una solución de la cinemática inversa. El parámetro options puede usarse para seleccionar una de las posibles soluciones iniciales de la cinemática inversa. Finalmente, deberá:
a) Animar la trayectoria usando la función drawrobot3d.
b) Representar las trayectorias articulares a lo largo del movimiento.
c) Plotear las velocidades articulares asumiendo que el movimiento entre los puntos p1 and p2 se hace a velocidad constante (cartesiana) y termina en 1 s.
d) Enumera los posibles errores que pueden aparecer durante la consecución de la tarea como, por ejemplo:
- Puntos fuera del espacio de trabajo del robot.
- Puntos singulares donde det(J)=0. En estos puntos singulares pueden aparecer velocidades infinitas para intentar conseguir velocidades finitas del extremo.
- Valores articulares fuera de rango.

5 Cinemática inversa en velocidad

En estos momentos habrás resuelto correctamente (espero) el Ejercicio 5. En la simulación, habrás visto moverse al robot siguiendo una línea recta en el espacio. Sin embargo, esta simulación contiene un error de concepto importante: hemos definido todas las posiciones q que debe seguir el robot. A continuación, el robot se mueve de una a otra de estas posiciones articulares q…. pero, ¿con qué velocidad articular se mueve el robot? Para que el armario de control del robot pueda, efectivamente, hacer que el robot se mueva sobre

la recta, deberá controlar todas las velocidades articulares y las posiciones articulares (usando algún algoritmo de control de posición y velocidad).

Así pues, comenzamos recordando la cinemática directa en velocidad. La matriz Jacobiana del manipulador me permitía escribir:

$$\begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix} = J * \dot{q}$$

Recordemos que J se podía calcular para cada posición articular del robot usando la librería ARTE:

>> J = manipulator_jacobian(robot, q)

De esta manera, dado que conocemos las velocidades articulares, podemos calcular la velocidad lineal/angular del extremo del robot (en coordenadas de la base). Pero… atención, que ocurre si deseamos, por ejemplo, que el robot siga una trayectoria recta en el entorno. Obviamente, cuando el robot sigue una línea en el espacio, su velocidad v deberá ser tangente a esta trayectoria y, por tanto, será conocida. Supuesta conocida v y w, calcularemos la velocidad haciendo:

$$\dot{q} = J^{-1} \cdot \begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix}$$

Esta se confiere como la ecuación básica de la cinemática inversa en velocidad para un robot manipulador. Un sinfín de preguntas aparecen cuando planteamos esta ecuación.

Ejercicio 6:

Escribe, a continuación, estas preguntas.

Para que no se nos olvide ninguno de estos conceptos importantes, escribiremos, a continuación estas preguntas.

- ¿Qué dimensiones tiene qd, J y vw?
- ¿Es posible calcular siempre esta ecuación?

$$\dot{q} = J^{-1} \cdot \begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix}$$

- Existe siempre la Jacobiana inversa.
- ¿La matriz J es siempre cuadrada?
- ¿Qué ocurre cuando tenemos un robot de 3 GDL?
- ¿Qué ocurre cuando tenemos un robot de 7 GDL?

Ejercicio 7:

7.1 Cargue el robot IRB140. Halle J para la posición articular q = pi/8*[1 1 1 1 1 1] rad halle la velocidad angular si qd = [1 1 1 1 1 1] rad/s

7.2 Cargue el robot IRB140. Halle J para la posición articular q = [0 0 0 0 0 0] rad halle la velocidad angular si qd = [1 1 1 1 1 1] rad/s

7.3 Cargue el robot IRB140. Halle qd para la posición articular q = pi/8*[1 1 1 1 1 1] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

7.4 Cargue el robot IRB140. Halle qd para la posición articular q = pi/8*[0 0 0 0 0 0] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

Anote las conclusiones principales que extrae de todo lo anterior.

Del ejercicio anterior deberías haber extraído que:

- J es de dimensiones 6xn.
- Para que exista la inversa de J, debe ocurrir:
  - Que J sea cuadrada (n=6).
  - Que el rango de J sea 6.
- Cuando el rango de J es menor de 6, hablamos de un punto singular. En estos puntos articulares, el robot pierde uno o más grados de libertad y no es capaz de imprimir cualquier velocidad lineal/angular al robot.

Ejercicio 8:

7.1 Cargue el robot RETHINK/SAWYER. Halle J para la posición articular q = pi/8*[1 1 1 1 1 1 1] rad halle la velocidad angular si qd = [1 1 1 1 1 1 1] rad/s

7.2 Cargue el robot RETHINK/SAWYER. Halle J para la posición articular q = [0 0 0 0 0 0 0] rad halle la velocidad angular si qd = [1 1 1 1 1 1 1] rad/s

7.3 Cargue el robot RETHINK/SAWYER. Halle qd para la posición articular q = pi/8*[1 1 1 1 1 1 1] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

7.4 Cargue el robot RETHINK/SAWYER. Halle qd para la posición articular q = pi/8*[0 0 0 0 0 0 0] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

Anote las conclusiones principales que extrae de todo lo anterior.

En estos momentos debes haber experimentado alguna dificultad al resolver el ejercicio 8… ya que, ¡el robot Sawyer tiene 7 GDL! y, por tanto:

- J es de dimensiones 6x7.
- No existe la inversa de J.

Lógicamente, el hecho de tener un robot con más GDL ofrece una ventaja (y no un inconveniente). Se puede demostrar que es posible resolver la cinemática inversa en velocidad haciendo:

$$\dot{q} = J^{\dagger} \cdot \begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix}$$

Donde se utiliza la pseudo-inversa Moore-Penrose que se implementa en Matlab con la función `pinv(J)`.

Ejercicio 8:

8.1 Cargue el robot RETHINK/SAWYER. Halle qd para la posición articular q = pi/8*[1 1 1 1 1 1 1] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

8.2 Cargue el robot RETHINK/SAWYER. Halle qd para la posición articular q = pi/8*[0 0 0 0 0 0 0] rad de manera que el extremo se mueva con vw = [1 1 0 0 0 1] m/s, rad/s.

Anote las conclusiones principales que extrae de todo lo anterior.

5 Ejercicios finales

5.1 La cuadratura del círculo

Escriba una función que consiga que el robot siga una trayectoria semicircular. Nótese que una trayectoria semicircular se puede especificar de varias formas:

- Especificando el punto central del círculo, el punto inicial de la trayectoria y el punto final de la trayectoria.
- Especificando el punto central del círculo, el punto inicial de la trayectoria y los grados barridos sobre el círculo.
- Especificando tres puntos en el espacio 3D. En este caso, el punto p1 es la posición actual del robot y los puntos p2 y p3 definen la trayectoria semicircular.

El estudiante deberá escribir una función que genere la trayectoria semicircular que pase (en este orden) por los puntos p1, p2 y p3. A continuación, deberá crear otra función diferente que permita seguir la trayectoria anterior. Para ello deberá utilizar una variación de la función del ejercicio 5 para hacer que el robot siga esa trayectoria. Puede simular el proceso de seguimiento de trayectoria llamando repetidamente a la función `drawrobot3d` para diferentes valores de q.

Como ayuda, se proporciona el siguiente código, basado en la función `circlefit3d`. El código, a continuación, genera tres puntos 3D aleatorios y calcula el centro y radio de la única esfera (trayectoria circular) que incluye a esos 3 puntos (siempre y cuando los tres puntos sean diferentes entre sí).

```
p1 = rand(1,3);
p2 = rand(1,3);
p3 = rand(1,3);
[center, radius] = circlefit3d(p1,p2,p3);
```

El lector/a debería, en este momento, verificar la ecuación de la esfera con los resultados obtenidos. El código siguiente, devuelve, además, dos vectores v1 y v2 normales entre sí y pertenecientes al plano que contiene los 3 puntos.

```
p1=rand(1,3);
p2=rand(1,3);
p3=rand(1,3);
[center,radius,v1,v2] = circlefit3d(p1,p2,p3);
plot3(p1(:,1),p1(:,2),p1(:,3),'bo');
hold on;
plot3(p2(:,1),p2(:,2),p2(:,3),'bo');
plot3(p3(:,1),p3(:,2),p3(:,3),'bo');

for a=0:0.05:2*pi
    x = center(:,1)+sin(a)*radius.*v1(:,1)+cos(a)*radius.*v2(:,1);
    y = center(:,2)+sin(a)*radius.*v1(:,2)+cos(a)*radius.*v2(:,2);
```

```
    z = center(:,3)+sin(a)*radius.*v1(:,3)+cos(a)*radius.*v2(:,3);
    plot3(x,y,z,'r.');
end
axis equal;grid on;rotate3d on;
```

Para resolver el ejercicio se debería buscar un vector v que uniera el centro del círculo con el punto inicial (medio y final) de la trayectoria. Para ayudar al estudiante, se proporciona, a continuación, un código que representa todos los vectores en el espacio 3D. La figura siguiente presenta una vista 3D de los puntos p1, p2 y p3 (rojo, verde y azul). Los vectores vp1, vp2 y vp3. Y los vectores normales v1 y v2. La trayectoria deseada está marcada con puntos rojos.

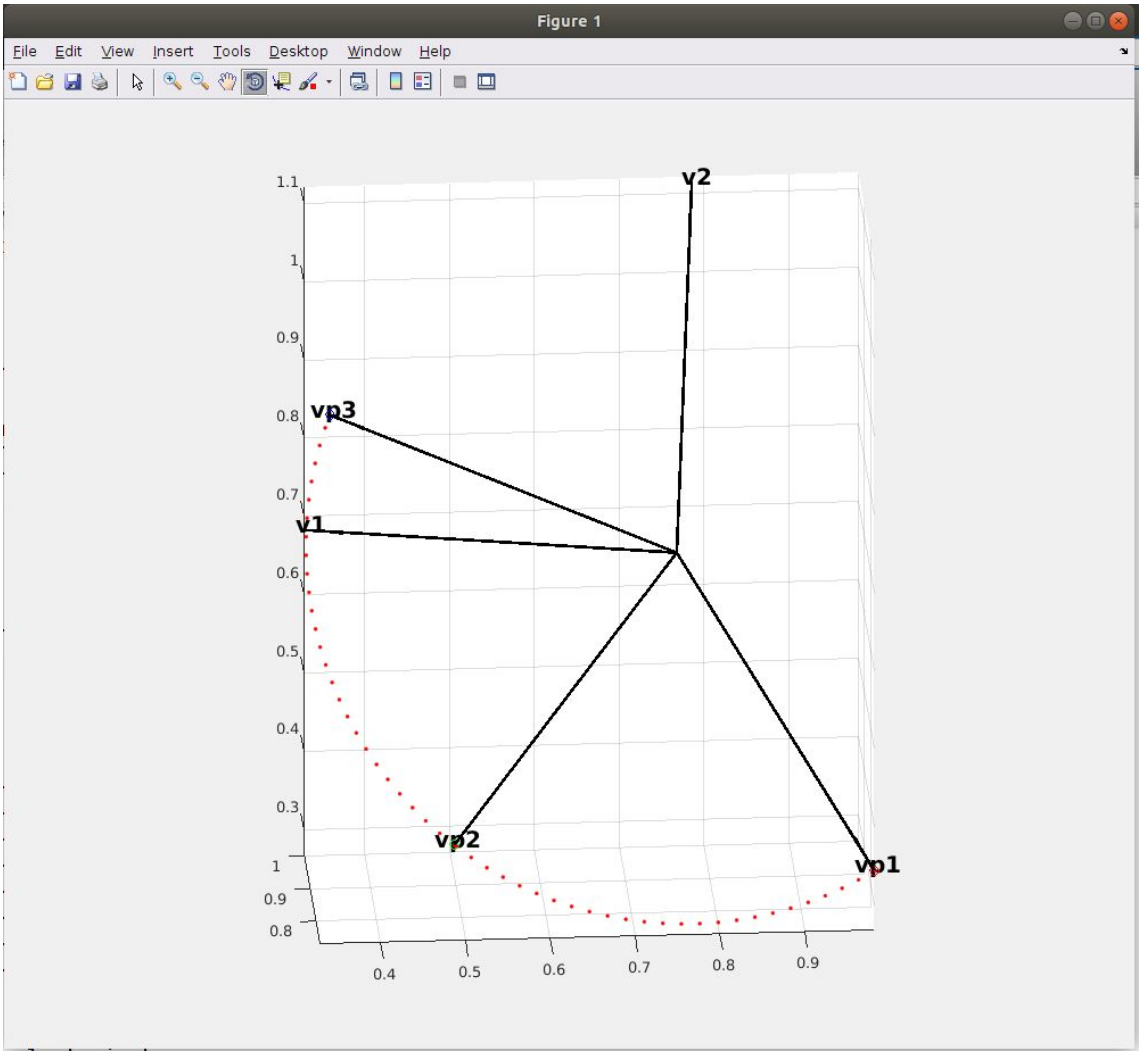
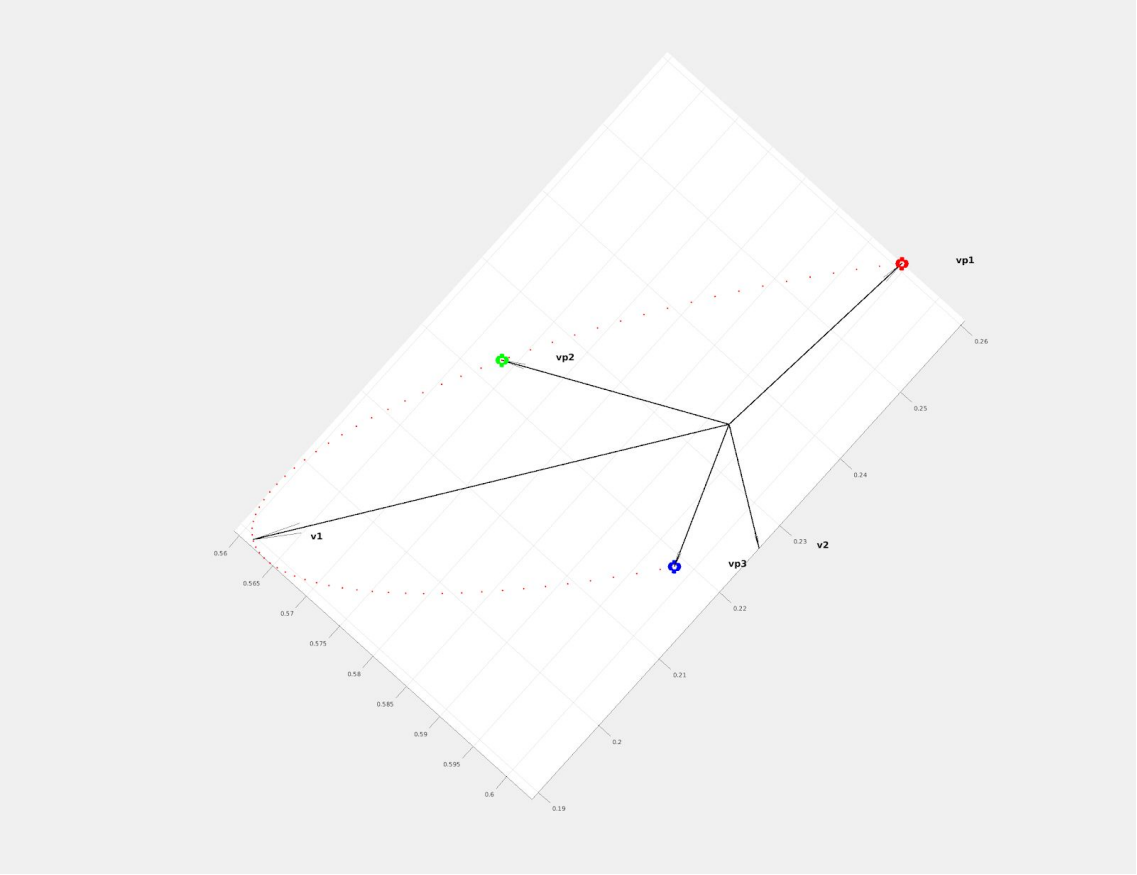


Figure 1
File Edit View Insert Tools Desktop Window Help
v2
vp3
v1
vp2
vp1
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.4 0.5 0.6 0.7 0.8 0.9



Se proporciona el código siguiente como ayuda:

```
function circle_trajectory()
close all
figure
% points belonging to trajectory p1-->p2-->p3
[p1, p2, p3] = rand3points();
[center,radius,v1,v2] = circlefit3d(p1',p2',p3');

plot3(p1(1),p1(2),p1(3),'ro');
hold on;
plot3(p2(1),p2(2),p2(3),'go');
plot3(p3(1),p3(2),p3(3),'bo');

%build up unit vectors
vp1 = p1(:)-center(:);
vp1 = vp1/norm(vp1);
vp2 = p2(:)-center(:);
vp2 = vp2/norm(vp2);
vp3 = p3(:)-center(:);
vp3 = vp3/norm(vp3);
% draw vectors vp1, vp2 and vp3
draw_my_vector(radius*vp1', center', 'vp1')
draw_my_vector(radius*vp2', center', 'vp2')
```

```
draw_my_vector(radius*vp3', center', 'vp3')

% draw ortonormal vectors v1 and v2 on the trajectory plane
draw_my_vector(radius*v1', center', 'v1')
draw_my_vector(radius*v2', center', 'v2')
```

- **En este punto, el alumno debería ser capaz de representar vp1, vp2 y vp3 en el sistema de coordenadas v1 y v2.**
- **Seguidamente, el estudiante deberá calcular puntos que comiencen en vp1 y roten este vector hacia vp2 y vp3 (en este orden).**
- **finalmente, el robot deberá seguir estos puntos, para lo que deberá encontrarse una solución de la cinemática inversa para cada punto de la trayectoria.**

function draw_my_vector(V, p0, text_label)

```
p1 = p0(:) + V(:);
vect_arrow(p0, p1, 'k', 2) %standard WIDTH for arrows is 3
text(p1(1)+0.005, p1(2)+0.005, p1(3)+0.005, text_label, 'FontWeight', 'bold',
'HorizontalAlignment', 'Center', 'FontSize', 15);
```

function [p1, p2, p3]=rand3points()

```
 p1=rand(1,3);
center=rand(1,3);
vp1 = p1-center;
radius = norm(vp1);
vp1 = vp1/radius;
vp1 = vp1(:);
theta1 = pi/4*rand;
theta2 = pi/4*rand;
R1 = Rot(theta1, 'Z');
R2 = Rot(theta2, 'X');
p1 = p1(:);
p2=center(:)+radius*R1*vp1(:);
p3=center(:)+radius*R1*R2*vp1(:);
```