Modelo en Matlab del robot industrial **ABB IRB2400**

Carlos Baraza Haro

13 de mayo de 2012

Índice

Parte I
Análisis del robot industrial IRB2400

1. Robot Industrial ABB IRB2400

En primer lugar, durante este trabajo de modelado de un robot industrial con Matlab, he utilizado como ejemplo de modelado un robot industrial de la compañia ABB. Concretamente, el modelo IRB 2400 que se puede ver en la figura 1.

Figura 1: ABB IRB2400

De la página oficial del producto podemos obtener las siguientes características:

- El IRB 2400, en sus distintas versiones y con su mejor precisión, proporciona unas prestaciones excelentes para manipulación de materiales, asistencia a máquinas y aplicaciones de procesos. El IRB 2400 ofrece mayores volúmenes de producción, menores tiempos de producción y en consecuencia de entrega más rápidas para los productos fabricados.

    - **Fiable:** Alta disponibilidad El IRB2400 es el robot industrial más popular. Su construcción robusta y el uso de pocos componentes contribuye a su alta fiabilidad y pocas necesidades de mantenimiento.
    - **Rápido:** Reducidos tiempos de ciclos Gracias al sistema de control del movimiento de ABB, el robot optimiza la aceleración y desaceleración, lo que implica un menor tiempo de ciclo.
    - **Preciso:** Calidad uniforme de las piezas producidas El mejor de su clase de en cuanto a la pecisión de la trayectoria y la repetibilidad de la posición. (RP= 0.06 mm)
    - **Potente:** Utilización Optima La combinación de una capacidad de carga de 7 - 20 kg y un alcance de 1810 mm.
    - **Robusto:** Para ambientes de trabajo exigentes Clasificado IP67, Steam Washable, Foundry Plus y Clean Room ISO class 100.
    - **Versátil:** Integración flexible Todos los modelos se ofrecen con la posibilidad de montaje invertido.

1.1. Análisis de la morfología.

El robot consta de 6 grados de libertad con 6 articulaciones rotativas. No dispone de ninguna articulación telescópica. Por ello, los actuadores de los que dispone el robot son servomotores en todas las articulaciones.

Es interesante destacar que el robot tiene una disposición de los eslabones que permite realizar un desacoplo cinemático de la muñeca del robot (3 últimas articulaciones) de las 3 primeras articulaciónes (Utilizadas para fijar la posición del extremo).

1.2. Volumen de trabajo del robot.

En la figura 2podemos ver el volumen de trabajo del robot, siguiendo las diferentes limitaciones que tiene cada una de las articulaciones.

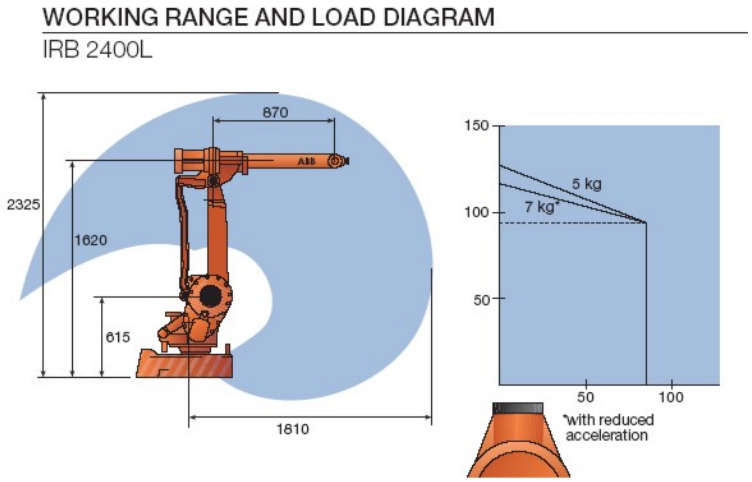

Figura 2: Volumen de trabajo

2. Modelo del IRB2400

En primer lugar, vamos a analizar cada una de las partes necesarias para realizar un modelo en Matlab del robot industrial de nuestro estudio.

2.1. Parámetros Denavit-Hartenberg.

Los parámetros Denavit-Hartenberg se obtienen siguiendo exaustivamente el método sistemático desarrollado por estos ingenieros. De esta forma obtenemos los siguientes parámetros para nuestro robot.

Articulación	θ	d	a	α
1	θ_1	615 mm	100 mm	$-\frac{\pi}{2}$
2	$\theta_2 - \frac{\pi}{2}$	0	855 mm	0
3	θ_3	0	150 mm	$-\frac{\pi}{2}$
4	θ_4	869.18 mm	0	$\frac{\pi}{2}$
5	θ_5	0	0	$-\frac{\pi}{2}$
6	θ_6	65 mm	0	0

En la figura 3 en la página siguientepodemos ver como fueron definidos los parámetros Denavit-Hartenberg de nuestro robot de estudio IRB2400.

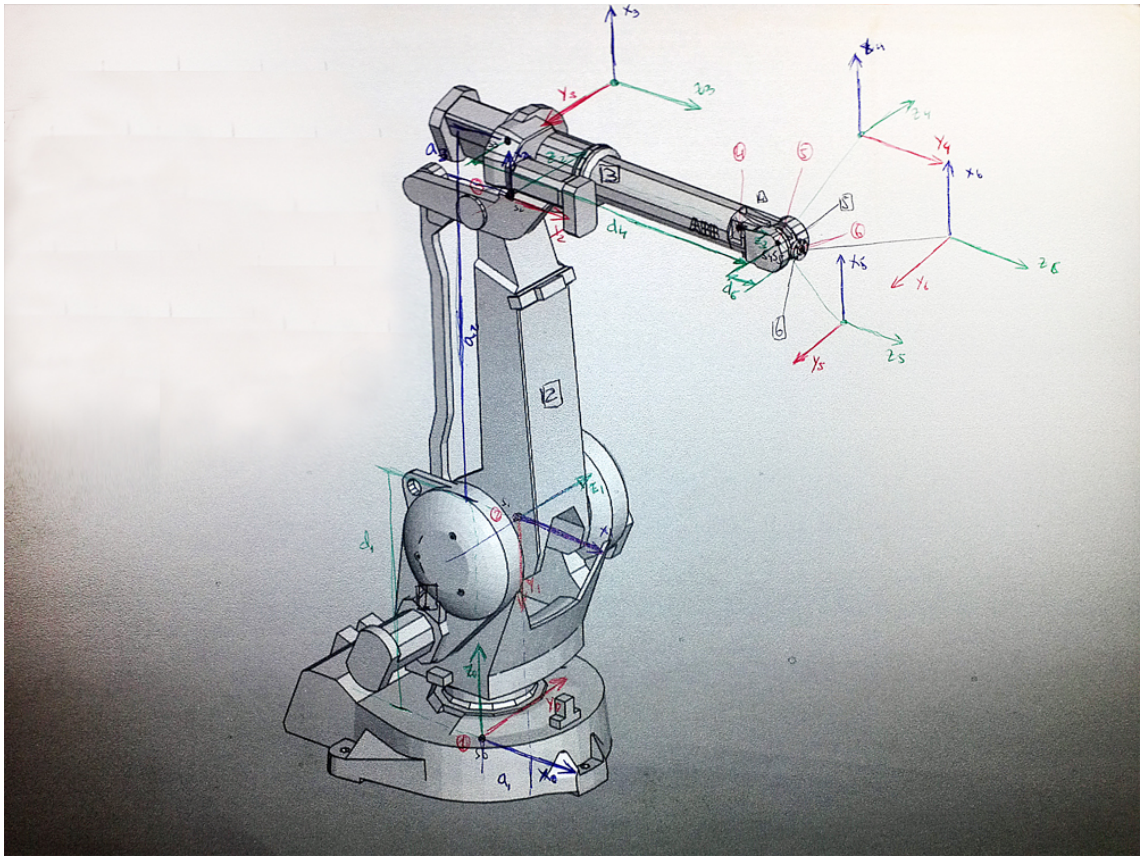

Figura 3: Parámetros Denavit-Hartenberg

2.2. Transformada cinemática directa.

La transformada cinemática directa consiste en obtener la posición del extremo del robot con respecto al origen con los valores deseados de cada variable articular.

Para ello calculamos la matriz ${}^0A_6$, que será el producto de todas las matrices homogéneas de paso de una articulación X-1 a la articulación X. Por lo tanto,

$$ {}^0A_6 = {}^0A_1 * {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 * {}^5A_6 $$

siendo cada una de esas matrices obtenibles muy facilmente a partir de su correspondiente variable articular $\theta_x / x \epsilon \{1,2,3,4,5,6\}$

El algoritmo para ser implementado en Matlab es el siguiente:

```
function T = TCD( q )
    Ap = transl(100,0,615);
    Ap(1:3,1:3) = rotx(-pi/2);
    Arotz = [rotz(q(1)) [0 0 0]'; 0 0 0 1];
    A{1} = Arotz*Ap;

    Ap = transl(855,0,0);
    Ap(1:3,1:3) = rotx(0);
    Arotz = [rotz(q(2)-pi/2) [0 0 0]'; 0 0 0 1];
    A{2} = Arotz*Ap;

    Ap = transl(150,0,0);
    Ap(1:3,1:3) = rotx(-pi/2);
    Arotz = [rotz(q(3)) [0 0 0]'; 0 0 0 1];
    A{3} = Arotz*Ap;
```

```
        Ap = transl(0,0,869.18); %posible error por la coma
        Ap(1:3,1:3) = rotx(pi/2);
        Arotz = [rotz(q(4)) [0 0 0]'; 0 0 0 1];
        A{4} = Arotz*Ap;

        Ap = transl(0,0,0);
        Ap(1:3,1:3) = rotx(-pi/2);
        Arotz = [rotz(q(5)) [0 0 0]'; 0 0 0 1];
        A{5} = Arotz*Ap;

        Ap = transl(0,0,65);
        Ap(1:3,1:3) = rotx(0);
        Arotz = [rotz(q(6)) [0 0 0]'; 0 0 0 1];
        A{6} = Arotz*Ap;

        T = A{1}*A{2}*A{3}*A{4}*A{5}*A{6};
    end
```

2.3. Transformada cinemática inversa.

La transformada cinemática inversa consiste en obtener las variables articulares a partir de una matriz homogénea que define el extremo del robot, tanto en orientación como en posición.

Este algoritmo es más sencillo de realizar gracias a la morfología de este tipo de robots industriales, que reduce las posibilidades que, en otros robots podrían ser infinitas, complicando la obtención de el resultado más óptimo.

Por lo tanto, el primer paso será "desacoplar" las tres primeras articulaciones de las tres últimas. Las tres primeras situarán la muñeca del robot en la posición correcta y, las tres últimas se encargarán de la orientación del extremo y la posición.

2.3.1. Posición de la muñeca

Como sabemos cual será el vector w del extremo del robot, la posición del extremo y que, la posición de la muñeca está a una distancia de 65mm en la dirección del vector w, podemos obtener la posición de la muñeca de la siguiente forma.

```
pm = T(1:3,4) - 65.*T(1:3,3);
```

2.3.2. Variables articulares $\theta_1, \theta_2, \theta_3$

Por métodos geométricos, podemos obtener expresiones que relacionen directamente la posición de la muñeca con las tres primeras variables articulares. De esta forma, obtenemos el siguiente algoritmo de Matlab para el cálculo de las coordenadas articulares 1, 2 y 3.

```
%Calculo de posicion muñeca
pm = T(1:3,4) - 65.*T(1:3,3);
pm2 = [norm(pm(1:2))-a1, pm(3)-d1];

%Primer articulacion
q1 = atan2(pm(2),pm(1));

%Segunda articulación
alpha = acos(-(a3^2+d4^2-a2^2-pm2(1)^2-pm2(2)^2)/(2*a2*sqrt(pm2↩
    (1)^2+pm2(2)^2))); %Angulo auxiliar alpha
q2 = pi/2 - alpha - atan2(pm2(2),pm2(1));

%Tercera Articulación
v3 = [a2*sin(q2), a2*cos(q2)];
pm2 = [pm2(1) pm2(2)];
u3 = pm2 - v3;
```

```
q3 = -( atan(d4/a3) - acos( sum(u3 .* v3)/ (norm(u3)*norm(v3)) ) );
```

2.3.3. Variables articulares $\theta_4, \theta_5, \theta_6$

A partir de las tres primeras coordenadas articulares y jugando un poco simbólica y numéricamente con los elementos de las matrices homogéneas, podemos obtener relaciones directas entre las tres primeras variables articulares y las tres últimas variables articulares. En Matlab, se implementa de la siguiente forma:

```
%% Articulaciones 4, 5 y 6
A01 = R.GetANum(1,q1);
A12 = R.GetANum(2,q2);
A23 = R.GetANum(3,q3);

%R36 = inv(R.A{3})*inv(R.A{2})*inv(R.A{1})*A0;
R36 = A23\( A12\( A01\T ) );

q4 = atan2(-R36(2,3),-R36(1,3));
q5 = acos(R36(3,3));
q6 = atan2(-R36(3,2),R36(3,1));

warning on all %Volver a activar los warnings

enLimitesArticulaciones = q1 > -pi && q1 < pi && q2 > -1.7453 && q2 < 1.7453 && q3 > -1.0908 && q3 < 1.0908 && q4 > -3.2289 && q4 < 3.2289 && q5 > -2.0944 && q5 < 2.0944 && q6 > -6.9813 && q6 < 6.9813;

if ( isreal(q1) && isreal(q2) && isreal(q3) && isreal(q4) && isreal(q5) && isreal(q6) && enLimitesArticulaciones)

    %% Devolver el valor de las articulaciones
    q = [q1,q2,q3,q4,q5,q6];
    R.err = '';
else
    q = R.q;
    R.err = 'Error␣01:␣Punto␣solicitado␣fuera␣del␣volumen␣de␣trabajo';
    display(R.err);
end
```

2.3.4. El algoritmo completo de la TCI

El algoritmo completo de Matlab para el cálculo de la TCI sería el siguiente:

```
function q = TCI( R, T )
    %Parametros DH
    d1=615; a1=100; a2=855; a3=150; d4=869.18; d6=65;

    %% Primeras tres articulaciones

    warning off all %Desactivar los warnings por las partes imaginarias

    %Calculo de posicion muñeca
    pm = T(1:3,4) - 65.*T(1:3,3);
    pm2 = [norm(pm(1:2))-a1, pm(3)-d1];
```

```
        %Primer articulacion
        q1 = atan2(pm(2),pm(1));

        %Segunda articulación
        alpha = acos(-(a3^2+d4^2-a2^2-pm2(1)^2-pm2(2)^2)/(2*a2*sqrt(↩
            pm2(1)^2+pm2(2)^2))); %Angulo auxiliar alpha
        q2 = pi/2 - alpha - atan2(pm2(2),pm2(1));

        %Tercera Articulación
        v3 = [a2*sin(q2), a2*cos(q2)];
        pm2 = [pm2(1) pm2(2)];
        u3 = pm2 - v3;
        q3 = -( atan(d4/a3) - acos( sum(u3 .* v3)/ (norm(u3)*norm(v3)↩
            ) ) );


        %% Articulaciones 4, 5 y 6
        A01 = R.GetANum(1,q1);
        A12 = R.GetANum(2,q2);
        A23 = R.GetANum(3,q3);

        %R36 = inv(R.A{3})*inv(R.A{2})*inv(R.A{1})*A0;
        R36 = A23\( A12\( A01\T ) );

        q4 = atan2(-R36(2,3),-R36(1,3));
        q5 = acos(R36(3,3));
        q6 = atan2(-R36(3,2),R36(3,1));

        warning on all %Volver a activar los warnings

        enLimitesArticulaciones = q1 > -pi && q1 < pi && q2 > -1.7453↩
             && q2 < 1.7453 && q3 > -1.0908 && q3 < 1.0908 && q4 > ↩
            -3.2289 && q4 < 3.2289 && q5 > -2.0944 && q5 < 2.0944 && ↩
            q6 > -6.9813 && q6 < 6.9813;

        if ( isreal(q1) && isreal(q2) && isreal(q3) && isreal(q4) && ↩
            isreal(q5) && isreal(q6) && enLimitesArticulaciones)

             %% Devolver el valor de las articulaciones
             q = [q1,q2,q3,q4,q5,q6];
             R.err = '';
        else
             q = R.q;
             R.err = 'Error␣01:␣Punto␣solicitado␣fuera␣del␣volumen␣de␣↩
                 trabajo';
             display(R.err);
        end
    end
```

2.4. Jacobiana directa

Puesto que en este trabajo no vamos a necesitar trabajar con jacobianas, solamente las he desarrollado simbólicamente, sin crear un método para trabajar con ella desde matlab.

La jacobiana de la posición de la muñeca se calcula de la siguiente forma:

$$Jpm = \begin{pmatrix} \frac{dx}{dq_1} & \frac{dx}{dq_2} & \frac{dx}{dq_3} \\ \frac{dy}{dq_1} & \frac{dy}{dq_2} & \frac{dy}{dq_3} \\ \frac{dz}{dq_1} & \frac{dz}{dq_2} & \frac{dz}{dq_3} \end{pmatrix}$$

$$Jpm = \begin{pmatrix} s_1(a_1 + s_2a_2 + c_2a_3s_3 + s_2c_3a_3) & c_1(c_2a_2 - s_2s_3a_3 + c_2c_3a_3) & c_1(c_2c_3a_3 - s_3s_2a_3) \\ c_1(a_1 + s_2a_2 + c_2a_3s_3 + c_3s_2a_3) & s_1(c_2a_2 - s_2s_3a_3 + a_3c_3c_2) & s_1(c_2c_3a_3 - s_3s_2a_3) \\ 0 & c_2s_3a_3 - s_2c_3a_3 - s_2a_2 & s_2c_3a_3 - c_2s_3a_3 \end{pmatrix}$$

Parte II
Modelo IRB2400 con Matlab

3. Clase IRB2400

Para el realizar un modelo virtual del sistema, he utilizado el sistema de clases de Matlab. He creado una clase cuyas instancias pueden tratarse de forma totalmente autónoma. De esta manera, creando un objeto de tipo IRB2400, obtendremos un "robot virtual" sobre el que se podrán comandar diferentes ordenes, visualizando en todo momento la representación en tres dimensiones del estado del robot en una figura de Matlab.

Además, para facilitar es uso a modo de ejemplo, he incluido un interfaz de usuario realizado con Matlab GUIDE que acerca mucho más el uso del robot al usuario con poco conocimiento del sistema de clases.

Resumidamente, la clase IRB2400 contará con las siguientes propiedades y métodos:

3.1. Propiedades

3.1.1. Links3d

Propiedad que contiene los Vertex, Faces, CData y Patch de cada uno de los eslabones. Es una propiedad imprescindible para la representación en 3D ya que contiene los datos del estado actual para cada uno de los eslabones. Se puede acceder a los datos de un eslabón x-1 de la siguiente forma R.Links3d{x}. Cada uno de los objetos tipo Link3d dentro de la propiedad estructura Links3d contiene las propiedades:

- Faces
- Vert
- CData
- Patch

Estas propiedades se cargan en el constructor de la clase directamente de los archivos .STL del robot IRB2400 simplificados por mi de forma manual para reducir el número de cálculos de la CPU al representar el robot en una nueva posición. Los STL originales están disponibles en la web de ABB.

Links3d no es una propiedad útil ya que es tratada de forma interna por el método plot3d().

3.1.2. q

Se trata de un vector que contiene todas las coordenadas articulares de la posición en la que se encuentra el robot en el estado actual.

3.1.3. A

Array de matrices homogéneas para las actuales coordenadas articulares:

- A{1} = A01
- A{2} = A12

- A{3} = A23
- A{4} = A34
- A{5} = A45
- A{6} = A56

3.1.4. A0

Array de matrices homogéneas para el estado inicial de reposo (Home).

3.1.5. A06

Propiedad que contiene la matriz homogénea del extremo del robot con respecto al origen del mismo.

3.1.6. axesRobot

Contiene la referencia a los axes en los que están dibujados los eslabones en 3D. Evita que se pierda la misma pues, en tal caso no se podría representar el robot en 3D.

3.1.7. figRobot

Contiene la referencia a la figura donde se encuentra el axes del robot. Ayuda a volver al interfaz después de interactuar con otra figura.

3.1.8. figTray

Contiene la referencia a la figura donde se están dibujando los axes. Se crea con el método habTray().

3.1.9. gExtremo

Array de objetos de tipo Line y Text que contienen los ejes del extremo y las lineas discontinuas que ayudan a conocer la posición del extremo en la representación 3D.

- gExtremo{1}: Eje X del extremo (instancia de objeto Line)
- gExtremo{2}: Eje Y del extremo (instancia de objeto Line)
- gExtremo{3}: Eje Z del extremo (instancia de objeto Line)
- gExtremo{4}: Posición X del extremo (instancia de objeto Line)
- gExtremo{5}: Posición Y del extremo (instancia de objeto Line)
- gExtremo{6}: Posición Z del extremo (instancia de objeto Line)
- gExtremo{7}: Texto posición X del extremo (instancia de objeto Text)
- gExtremo{8}: Texto posición Y del extremo (instancia de objeto Text)
- gExtremo{9}: Texto posición Z del extremo (instancia de objeto Text)

3.1.10. err

Vector de tipo string que contiene el último error que ha ocurrido en el robot.

- "Error 01: Punto solicitado fuera del volumen de trabajo"

3.1.11. trayHistG

Array de objetos de tipo Line que contienen todas las trayectorias seguidas por el robot. Si limpiamos esta variable con el método clearTrayHist(R), borraríamos todo el historial de trayectorias descritas por el robot.

3.1.12. qlim

Esta propiedad contiene los límites fijados a cada una de las articulaciones.

- qlim(1) : Mínimo de la articulación q1
- qlim(2) : Máximo de la articulación q1
- qlim(3) : Mínimo de la articulación q2
- ...
- qlim(12) : Máximo de la articulación q6

3.2. Métodos

3.2.1. TCI(R , T)

Transformadas cinemáticas inversas

- R: Instancia de IRB2400
- T: Matriz homogénea

3.2.2. plot3d(R)

Representa el robot en los axes asociados a la instancia del robot en la propiedad "axesrobot". Esta función tarda un tiempo nada despreciable dependiendo del ordenador donde sea ejecutada. Es el motivo de que el interfaz no funcione correctamente en ciertos ordenadores, dependiendo directamente de su potencia de cálculo. Se solucionaría con un buen método para el cálculo de los vértices de los modelos de las diferentes piezas del robot.

- R: Instancia de IRB2400

3.2.3. Arefresh(R)

Actualiza la propiedad A dependiendo del valor de la propiedad q

- R: Instancia de IRB2400

3.2.4. clearTrayHist(R)

Limpia el historial de trayectorias.

- R: Instancia de IRB2400

3.2.5. ARefreshNum(obj , Num)

Actualiza la Matriz Num del Array de matrices A con los valores en la propiedad q.

3.2.6. goToPoint(robot , T , t)

Mueve el robot desde el estado actual hasta el estado final marcado por la matriz homogénea del segundo parámetro, en un tiempo de t segundos. Las articulaciones se mueven individualmente siguiendo un spline cúbico, lo que quiere decir que no describen ninguna trayectoria en cartesianas, simplemente van al punto final.

- robot : Instancia del IRB2400
- T : Homogénea del estado final
- t : tiempo en el que llegar. Si este tiempo es menor que cualquiera de los tiempos que tarda cada articulación en moverse hasta el punto a la máxima velocidad, el tiempo t será sustituido por el de la articulación crítica.

3.2.7. goline(robot , p)

Mueve el extremo del robot hasta el punto p conservando la actual orientación y, describiendo una trayectoria en forma de linea recta en las coordenadas cartesianas.

- robot : Instancia del IRB2400
- p: Punto final en coordenadas cartesianas.

3.2.8. IRB2400(*axes)

Este es el constructor de la clase. Puede llamarse directamente sin argumentos para crear una figura y un axes nuevo, si no hay ninguno seleccionado.

- axes: Axes donde será dibujado el robot.

3.2.9. A = GetANum(obj , Num , q)

Devuelve la matriz homogénea A_Num-1_Num.

3.2.10. habTray(obj , *figure)

Habilita la visualización trayectorias articulares cuando se hace un movimiento. En futuras versiones se visualizarán también las cartesianas. Modifica la propiedad protegida R.trayOn para darle un valor de 1, indicando que están preparadas las representaciones de trayectorias.

- El parámetro figure es opcional, si no se da ningún handles de figure, se creará uno nuevo.

3.2.11. deshabTray(obj)

Deshabilita la visualización de las trayectorias articulares.

3.3. Métodos estáticos

3.3.1. [ts,qs] = splineCubico(t,q)

A partir de un vector con los puntos de paso y otro vector con los tiempos de paso, devuelve un vector de tiempos intermedios y un vector de puntos intermedios.

3.3.2. TCD(q)

Transformada cinemática directa

- q: [q1,q2,q3,q4,q5,q6], Vector con las coordenadas articulares

4. Instalación

4.1. Primera opción

Ejecutar Matlab con permisos de superusuario y ejecutar la funcion “instalar”, en la raiz del proyecto.

4.2. Segunda opción

Para instalar la clase que he desarrollado y las funciones necesarias para la ejecución del programa hay que incluir la carpeta “IRB2400” en el Path de Matlab. Para hacerlo hay que ejecutar Matlab con permisos de superusuario ya que el archivo Path se encuentra en un área restringida a usuarios sin permisos de superusuario.

En File/Set Path... podemos añadir la carpeta IRB2400, donde la hayamos descomprimido. Sin embargo, es recomendable guardar la carpeta IRB2400 en MatlabInstallDir/R2011b/Toolbox/

4.3. Comprobación

Comprobamos que se ha instalado correctamente el toolbox ejecutando la siguiente función en el WorkSpace de Matlab: **R = IRB2400()**. Si aparece algún error quiere decir que no está instalado correctamente.

5. Ejemplos de uso de la clase IRB2400.

5.1. Con el IDE.

La primera opción para testear la clase IRB2400 es el uso del interfaz denominado "ide.m".

Para probarlo, después de instalar correctamente el toolbox, vamos a la carpeta CBHIRB2400 y ejecutamos la función llamada **ide**. Tras esto aparecerá una ventana con el robot dibujado en 3D y una serie de botones.

5.1.1. Control con teclado numérico:

Pinchamos en un área gris sin ningún control. A partir de este momento, con el teclado numérico podremos controlar el extremo del robot. Existen dos modos de control con teclado numérico:

- Modo Posición. Se activa pulsando el número 7.
  - Botones 1 y 3, desplazan el extremo en X.
  - Botones 2 y 5, desplazan el extremo en Y.
  - Botones 4 y 6, desplazan el extremo en Z.
  - Botones + y -, aumentan o decrementan el desplazamiento por cada movimiento. (300 mm defecto)
- Modo Orientación. Se activa pulsando el número 7.
  - Botones 1 y 3, giran el sistema del extremo sobre el eje u (X) del extremo.
  - Botones 2 y 5, giran el sistema del extremo sobre el eje v (Y) del extremo.
  - Botones 4 y 6, giran el sistema del extremo sobre el eje w (Z) del extremo.
  - Botones + y -, aumentan o decrementan el ángulo girado en cada movimiento. ($90^{\underline{o}}$ por defecto)

Si el punto que se solicita no puede ser alcanzado porque esta fuera del volumen de trabajo, aparecerá un mensaje de error en rojo en el IDE, además de ser almacenado tal error en la propiedad de la instancia del robot **err.**

5.1.2. Ir a puntos definidos en linea recta.

Para ir a un punto del volumen de trabajo en linea recta manteniendo la orientación en todo momento, colocamos el punto de destino en los textbox preparados para ello y pinchamos en el botón GoLine.

Si se atraviesa algún punto que no pertenece al volumen de trabajo, el robot dará un error y se quedará quieto. Por ello es importante asegurarse de que el robot puede realizar la trayectoria. El ejemplo que viene por defecto escrito en el IDE funciona correctamente:

1. Pinchamos en Go Home.
2. Escribimos un punto cómodo para el robot como por ejemplo el p = (1000, 0, 500) o p = (500, 500, 500).

5.1.3. Recortar ventana

Otro ejemplo de varios puntos encadenados en linea recta es el de recortar una ventana con el robot, atravesando todos los puntos correspondientes a el perímetro del cristal.

1. Pinchamos en Go Home para asegurar que el robot se encuentre en una posición "cómoda".
2. Pinchamos en el botón GO, con el siguiente código dentro del textarea:

```
1000 0 500
1000 1000 500
1000 1000 1500
1000 -1000 1500
1000 -1000 500
1000 0 500
```

Esto no son nada más que lo puntos intermedios que atravesará el robot. Si los modificamos por ejemplo a lo siguiente obtendremos otro perfil diferente de ventana recortada.

```
1000 0 500
1000 238 899
1000 700 899
1000 476 1268
1000 700 1697
1000 238 1697
1000 0 2096
1000 -238 1697
1000 -700 1697
1000 -476 1268
1000 -700 899
1000 -238 899
1000 0 500
```

5.2. Desde el WorkSpace de Matlab.

La clase IRB2400 fue diseñada para poder utilizarse sin ningún tipo de interfaz, simplemente utilizando el WorkSpace y su linea de comandos. Utilizarlo de esta forma no es la más cómoda pero, para determinados requisitos es la mejor opción. A continuación se puede ver el funcionamiento a groso modo con un ejemplo.

5.2.1. Crear una instancia de IRB2400.

Para crear un "robot" lo haremos de la siguiente forma:

```
R = IRB2400();
```

Si ya tenemos un axes creado con el handler en la variable llamada AXES1 y queremos asignarle el robot a ese Axes, lo haremos de la siguiente forma:

```
R = IRB2400(AXES1);
```

La instancia de la clase IRB2400 se llama R y, será el objeto con el que interactuaremos todo el tiempo. Con la sintaxis R.XXXX, accedemos a las diferentes propiedades o los diferentes métodos.

La figura 4 en la página siguientemuestra el resultado de crear una instancia de IRB2400. Pulsando en la flechita negra de la esquina superior derecha, podemos trabajar más cómodamente, viendo en todo momento el robot desde el WorkSpace.

5.2.2. Obtener la TCD.

Para obtener la TCD de unas coordenadas articulares dadas almacenadas en la variable q, lo haremos de la siguiente forma:

```
q = [1 0 0 0 0 0]; %q = [q1 q2 q3 q4 q5 q6];
T = R.TCD(q)
```

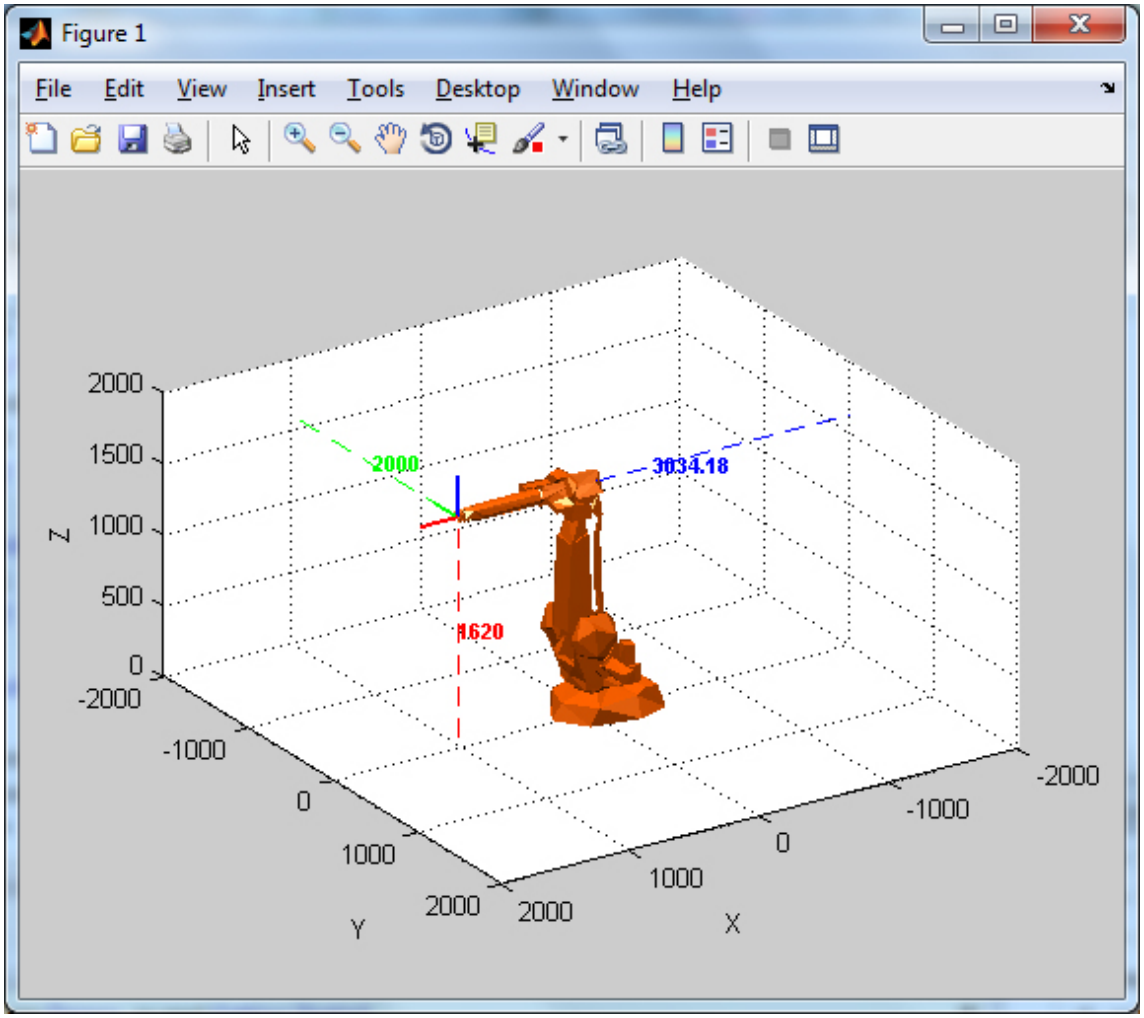

Figura 4: Figura con Axes de la instancia de IRB2400

5.2.3. Obtener la TCI.

Para obtener la TCI de una matriz homogénea dada, podemos utilizar la forma del siguiente ejemplo:

```
T = R.TCD([1 1 1 1 1 1]);
q = R.TCI(T)
```

Si la TCI de la matriz homogénea dada no existe, se imprime un error en el command line y, se almacena el error en la propiedad R.err

5.2.4. Ir a punto en tiempo t

Si quisiéramos desplazar el robot a la posición dada por una matriz homogénea o unas coordenadas articulares, el procedimiento sería el siguiente:

```
T = R.TCD([pi/2 0 0 0 0 0]);
tiempo = 2; %Segundos
R.goToPoint(T,tiempo);
```

Para ver las trayectorias que se están describiendo, ejecutaremos el siguiente código:

```
R.habTray(); %Habilitar visualización de trayectorias

T = R.TCD([pi/2 0 0 0 0 0]);
tiempo = 2; %Segundos
R.goToPoint(T,tiempo);
```

```
R.deshabTray(); %deshabilitamos la visualización si no queremos ver más trayectorias
```

5.2.5. Ir a punto en linea recta.

Si quisiéramos desplazar el robot a la posición dada por un vector p = [x y z], conservando la orientación, el procedimiento sería el del siguiente ejemplo.

```
R.goToPoint(R.TCD([0 0 0 0 0 0]),1); %volver posicion reposo
T = R.TCD([pi/2 0 0 0 0 0]);
p = T(1:3,4); %Vector columna p con la posicion de la homogenea T
R.goline(p);
```

Si quisiéramos limpiar las trayectorias rectilíneas que aparecen en rojo ejecutamos el siguiente método:

```
R.clearTrayHist();
```

5.2.6. Ver el estado actual del robot.

Si quisiéramos conocer en que posición se encuentra el robot en este momento, utilizaríamos los siguientes propiedades.

```
R.A06 %Se trata de la Matriz homogenea origen-extremo actual
R.q % Variables articulares del estado actual.
```

5.2.7. Representar forzosamente el robot en una posición.

Si quisiéramos representar el robot en una posición de forma forzosa, sin atravesar puntos intermedios ni nada para, por ejemplo, inicializarlo, lo haríamos de la siguiente forma:

```
R.q = [1 0 0 0 0 0]; %Modifica el estado actual del robot
R.Arefresh(); %Refresca las variables internas en funcion de la propiedad q
R.plot3d(); %Representa el estado forzado actual
```

Bibliografía

- **FUNDAMENTOS DE ROBOTICA**. 2 ED. Aut: BARRIENTOS. ANTONIO
- Corke's documentation.
- Matlab Documentation

Futuras versiones

El trabajo lleva muchísimas horas de esfuerzo pero, a pesar de todo, resultan insuficientes por lo que me gustaría enunciar las previsiones que tenía para esta versión pero que, debido a las fechas (Demasiado próximas a los exámenes), no he podido completar. Este verano probablemente mejore el software hasta llegar a las expectativas iniciales que tenía ya que quiero publicarlo y enviárselo a ABB.

Futuras características:

- Trayectorias articulares y cartesianas en cada desplazamiento
- Trayectorias articulares con spline quintico
- Trayectorias con splines en cartesianas.
- Añadir el tiempo como propiedad real del robot.

- Comentar todo el código de forma homogenea, no partes en ingles y otras en castellano.
- Mejorar el sistema de representación 3D que es muy lento por el uso de matrices homogeneas y ningun uso de GPU.
- Posibilidad de crear varios Robots en los mismos Axes (Bastante sencillo por la disposición de la clase IRB2400).