Práctica 2. Modelado y simulación

1. Objetivos

- Tener la capacidad de analizar las características de un robot y modelarlo.
- Conocer las características de la librería Robotics Toolbox para Matlab.
- Ser capaz de modelar-simular robots manipuladores y móviles con la Robotics Toolbox
- Conocer las posibilidades de simulación gráfica de otras herramientas.

2. Introducción: modelado y simulación de sistemas robóticos

Actualmente, existen diversas herramientas de modelado y/o simulación para los sistemas mecánicos de múltiples cuerpos, como lo es el robot. Como toda plataforma software, existen algunos de ellos comerciales o de uso libre (licencia GNU/GPL). Entre ellas, hay que distinguir las que permiten realizar un modelo matemático y una simulación virtual del mecanismo, y las que permiten a partir de un modelo CAD del mecanismo, modelar y simular la dinámica de los sistemas multi-cuerpo. Estos últimos programas, incorporan amplias librerías de articulaciones y fuerzas que permiten a los usuarios construir un modelo de un robot en un tiempo relativamente corto. Sin embargo, las altas prestaciones de este tipo de paquetes hacen que su coste económico sea alto para una finalidad educativa. Entre ellos, cabe citar a los siguientes paquetes software:

- **MSC ADAMS**. Este software posee un potente paquete para la simulación dinámica multi-cuerpo y control no lineal. Se emplea en muchas áreas de la industria, no sólo en robótica. Además, permite la co-simulación con Matlab-Simulink para poder desarrollar un controlador que cierre el lazo con la simulación dinámica del software (www.mscsoftware.com/es/product/adams).

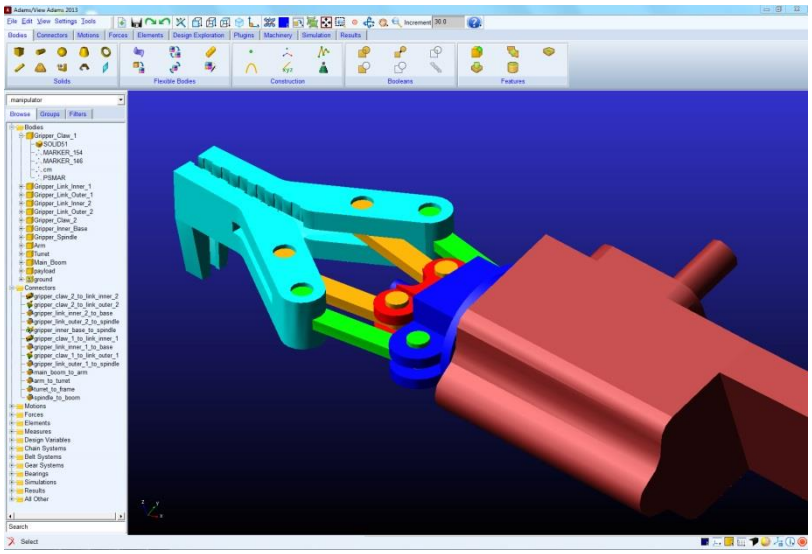


Figura 1. Imagen software MSC ADAMS

- **LMS Siemens Virtual Lab**. LMS Virtual Lab consiste en un software que cuenta con un ambiente integral enfocado para el desarrollo de la ingeniería concurrente aplicada en el diseño y análisis de la cinemática, la dinámica, la durabilidad, etc. Dicho software permanece abierto a todo tipo de vínculos mediante archivos CAD, CAM y análisis CAE, permitiendo la exportación e importación de estos archivos. (https://www.plm.automation.siemens.com/en_us/products/lms/virtual-lab/).

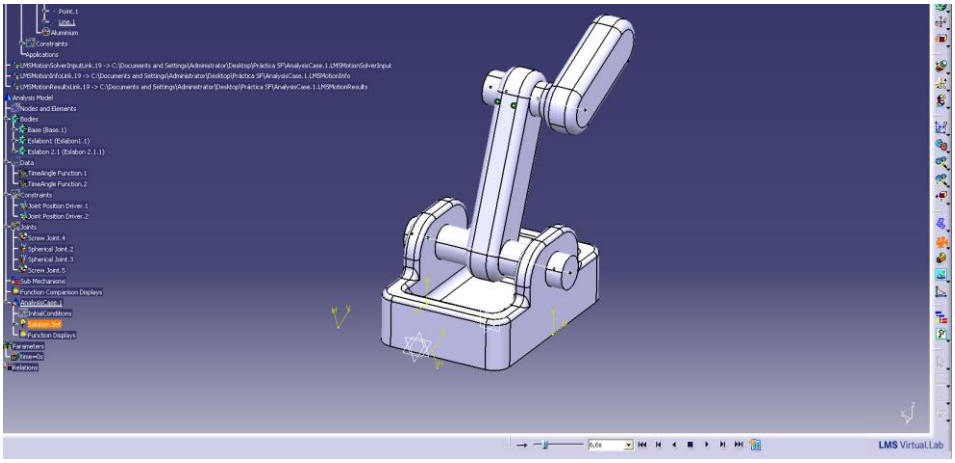


Figura 2. Imagen software LMS Virtual Lab

- **Autodesk Inventor**. Este software permite el análisis de sistemas multi-cuerpo pero siempre realizando un simplificado del sistema a un sistema lineal. Es posible calcular pares, esfuerzos, fuerzas, etc. en diversos sistemas mecánicos. Además, posee herramientas para la generación de engranajes y diferentes piezas mecánicas complejas. También permite exportar el modelo a un archivo *SimMechanics* que puede emplearse en el *Simulink* de Matlab para probar diferentes estrategias de control (http://www.autodesk.es/).

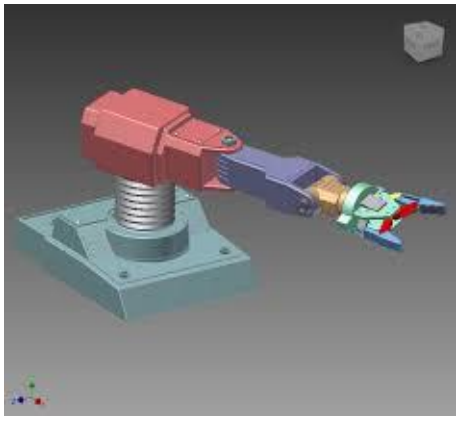


Figura 3. Imagen software Autodesk Inventor

Asimismo, como se ha comentado anteriormente, se pueden encontrar herramientas o paquetes gratuitos de software especializado en el modelado y simulación de robots. La mayoría de estos programas incorporan una interfaz gráfica de simulación avanzada, con resultados bastante vistosos. Sin embargo, el comportamiento dinámico de los mecanismos no suele estar implementado en estos simuladores, o suele estarlo implementado desde un punto de vista lineal o simplificado. Estos paquetes, están normalmente destinados a la educación y a la realización de prácticas por parte de los alumnos. Incluso algunos de ellos,

tienen una parte de uso libre limitada y otro paquete comercial completo. A continuación, se muestra una lista de algunos simuladores y paquetes que pueden accederse a través de Internet.

- **Easy Robot**. (www.easy-rob.de).

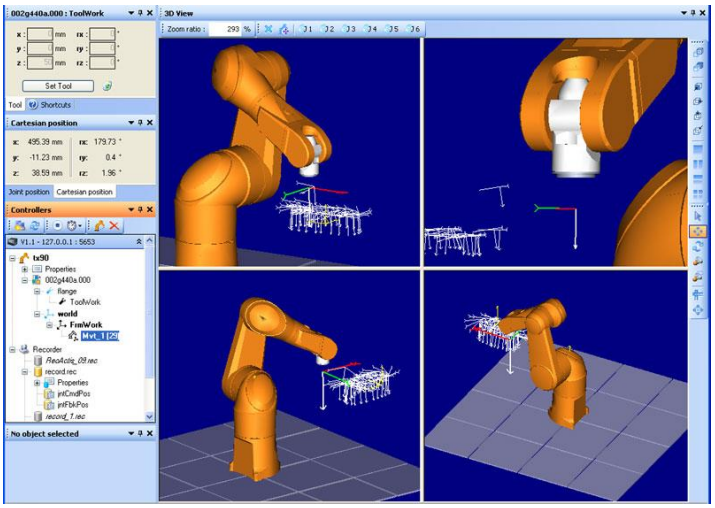


Figura 4. Imagen software Easy Robot.

- **VREP** (www.coppeliarobotics.com).

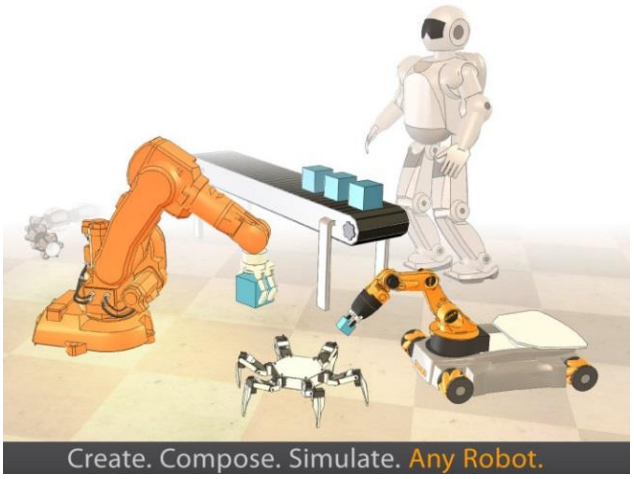


Figura 5. Imagen software V-REP

- **RobotStudio ABB** (http://new.abb.com/products/robotstudio).

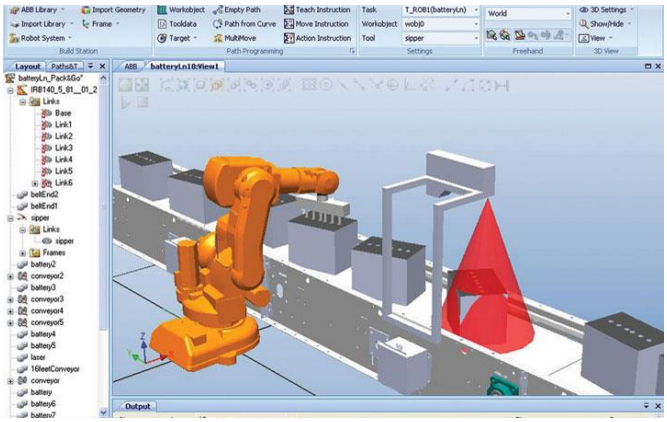


Figura 6. Imagen software RobotStudio ABB

- **ARTE** (https://arvc.umh.es/arte/index_en.html).

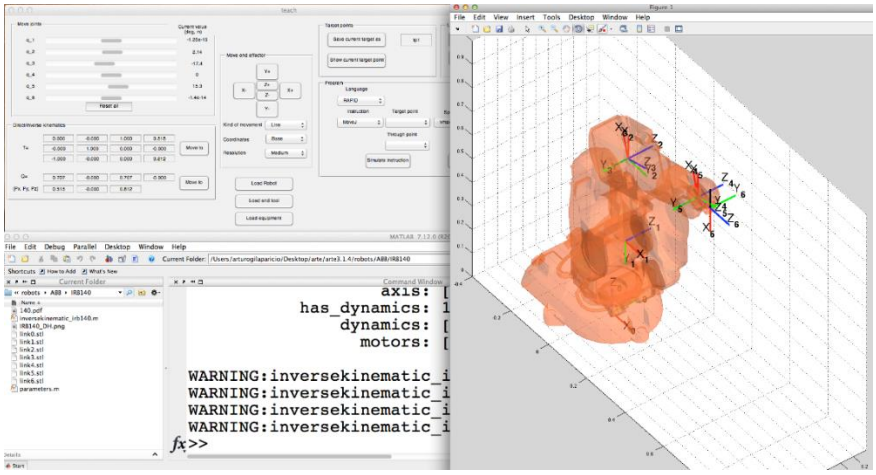


Figura 7. Imagen software ARTE (A Robotics Toolbox for Education)

- **RoboDK** (https://robodk.com/)

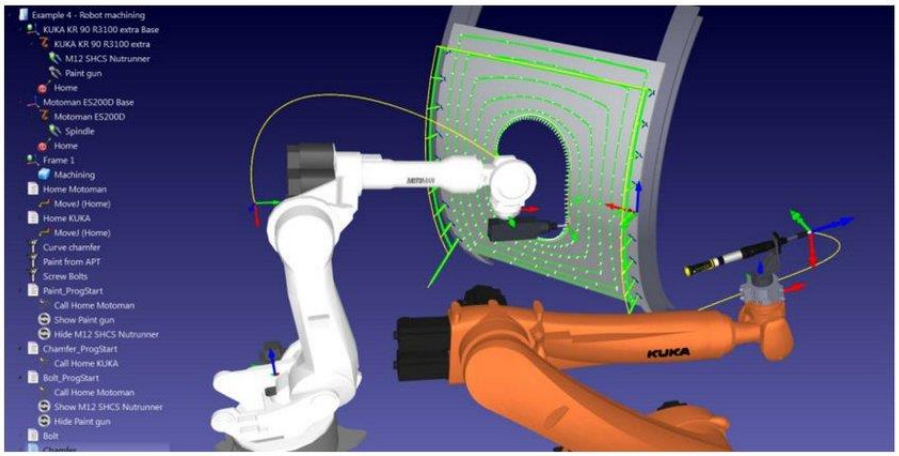


Figura 8. Imagen software RoboDK

- **Gazebo** (http://gazebosim.org/)

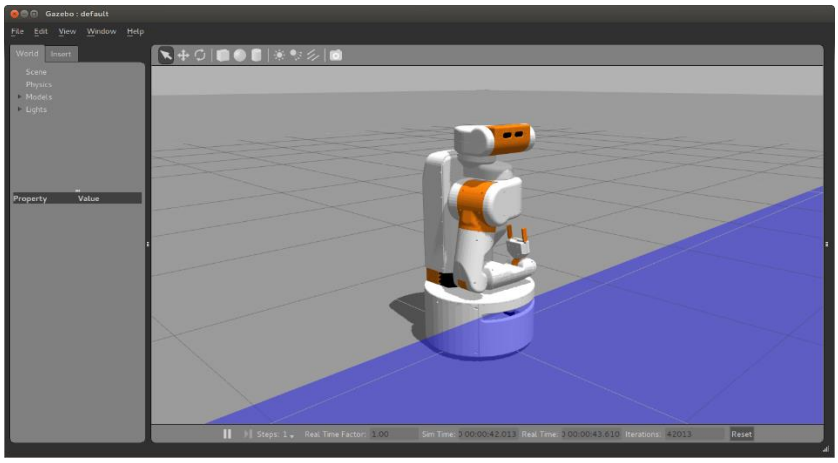


Figura 9. Imagen software Gazebo

Finalmente, existen herramientas diseñadas para el análisis de sistemas robotizados que se presentan con el código fuente accesible al usuario o al estudiante. De esta manera, se permite añadir o modificar el código para poder programar o diseñar cualquier sistema robotizado. Desde el punto de vista educativo, esta filosofía es la que más permite a los estudiantes enfocar la simulación y modelado de un robot desde un punto de constructivo. Así, se le da al estudiante la posibilidad de saber realmente cómo funcionan las cosas y los algoritmos matemáticos. Tal es el caso de la herramienta que se va a emplear en esta práctica: la *Robotics Toolbox* (RT) de Matlab (http://www.petercorke.com/RTB/), desarrollada por Peter I. Corke.

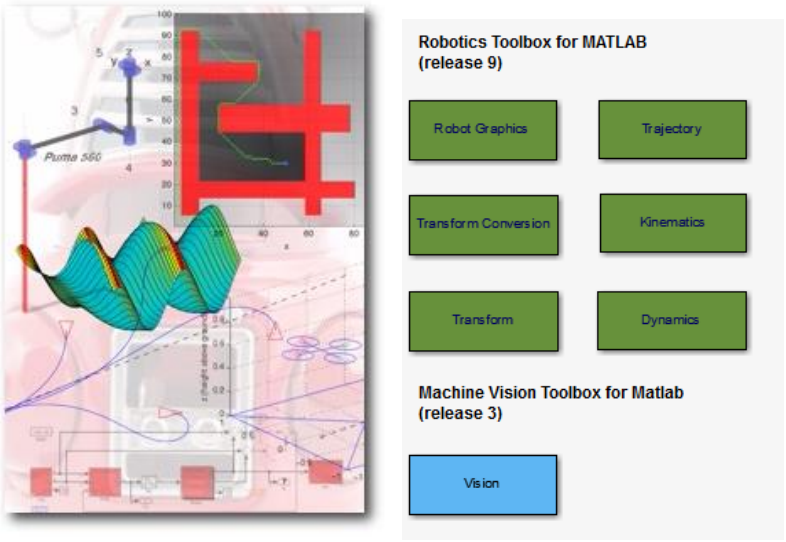


Figura 10. Imagen librería Robotics Toolbox

3. Introducción a la Robotics Toolbox

De forma general, la librería permite el estudio y diseño de brazos robóticos: desde la cinemática, la dinámica y generación de trayectorias. Además, también proporciona herramientas para trabajar con distintos tipos de datos básicos en el posicionamiento de robots, como vectores, transformaciones homogéneas o cuaternios. También posee herramientas adicionales para el modelado y simulación de otro tipo de robots como los móviles u autónomos, que también se tratará en esta práctica, junto con el modelado y simulación de robots seriales o antropomórficos (manipulador mecánico de cadena cinemática abierta).

La librería de Peter I. Corke se ha adjuntado como documentación de la práctica. De la página web, también se puede descargar un manual en PDF con diversos ejemplos (http://www.petercorke.com/RTB/r9/robot.pdf). Una vez descargada la librería y descomprimida, tan sólo hay que incluirla dentro del PATH de Matlab (véase Figura 11) con la opción *Add with subfolders*.

Una vez instalada la librería, se puede verificar el funcionamiento mediante el comando en la consola de Matlab *rtbdemo* donde podrías generar y modelar diferentes tipos de robot mediante una interfaz (véase Figura 11) o también con el comando *roblocks* donde aparece una interfaz con una serie de bloques *simulink* con las diferentes funcionalidades de la librería.

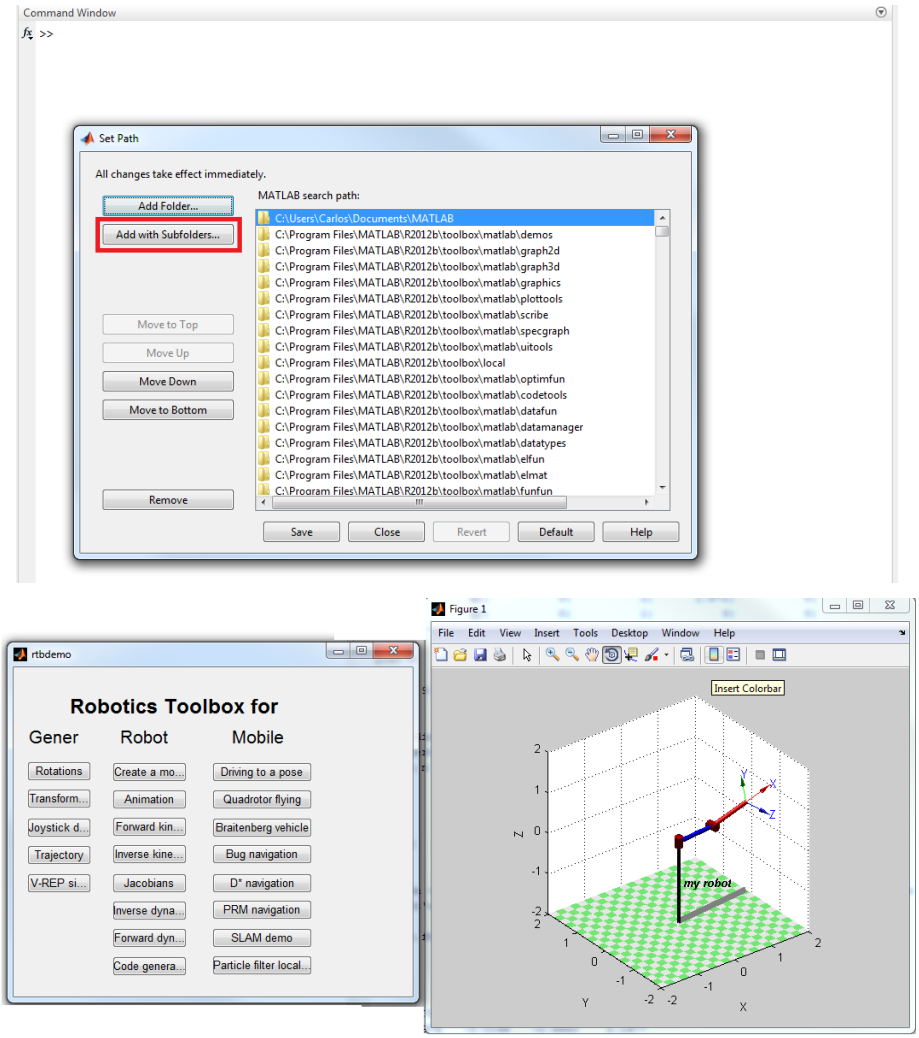


Figura 11. Instalación y ejecución de la RT

4. Herramientas matemáticas.

La RT posee una extensa funcionalidad para trabajar con distintos tipos de datos básicos en el posicionamiento de robots, como vectores, matrices de rotación, transformaciones homogéneas o cuaternios. En este apartado, se va a describir brevemente la funcionalidad para las matrices de rotación (MR) y cuaternios, y en cuanto a las funciones para las matrices de transformación homogénea (TH), se describirán en el apartado 5.2.

Respecto a las matrices de rotación, existen funcionalidades para la descripción de la rotación en los tres ejes principales X, Y, Z, además de distintas funciones para cambiar su representación, como por ejemplo a ángulos de Euler, cuaternios, o TH. A continuación, se muestra expresión matemática de las distintas rotaciones sobre los ejes principales y su función con la RT de Matlab.

MR sobre eje X, ángulo α *Función RT Matlab (α en rad)*

$$\mathbf{Rot}(x,\alpha)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\text{sen}\alpha\\ 0 & \text{sen}\alpha & \cos\alpha\end{bmatrix}$$

```
Rx=rotx(α)
```

MR sobre eje Y, ángulo β *Función RT Matlab (β en rad)*

$$\mathbf{Rot}(y,\beta)=\begin{bmatrix}\cos\beta & 0 & \text{sen}\beta\\ 0 & 1 & 0\\ -\text{sen}\beta & 0 & \cos\beta\end{bmatrix}$$

```
Ry=roty(β)
```

MR sobre eje Z, ángulo γ *Función RT Matlab (γ en rad)*

$$\mathbf{Rot}(z,\gamma)=\begin{bmatrix}\cos\gamma & -\text{sen}\gamma & 0\\ \text{sen}\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{bmatrix}$$

```
Rz=rotz(γ)
```

Se muestra a continuación, una tabla con la descripción de más funcionalidades de la RT para el uso de las MR y su transformación a otro tipo de representación.

Función	Descripción
`angvec2r`	forma angular/vectorial a matriz de rotación (MR)
`eul2r`	ángulos de Euler a MR
`rotx`	MR alrededor del eje X
`roty`	MR alrededor del eje Y
`rotz`	MR alrededor del eje Z
`rpy2r`	ángulos roll/pitch/yaw a MR
`r2t`	MR a TH

Un cuaternio (Q) está constituido por cuatro componentes {q0, q1, q2, q3}. Para la utilización de los cuaternios como metodología de representación de orientación, se realiza una asociación arbitraria del giro de un ángulo θ sobre un vector k al cuaternio definido por:

$$Q = \mathbf{Rot}(\boldsymbol{k},\theta) = \left(cos\frac{\theta}{2}, \boldsymbol{k}sen\frac{\theta}{2}\right)$$

La definición de un cuaternio en Matlab mediante la RT se realiza mediante la clase `Quaternion`, con la que se puede construir un objeto tipo cuaternio especificando el ángulo y el vector. Existen otras maneras de definir los cuaternios, como por ejemplo a partir de una MR o los cuatro componentes del elemento (véase la ayuda de la RT en Matlab).

```
>> Q=Quaternion(pi/2,[3 2 1])

Q =

0.70711 < 0.56695, 0.37796, 0.18898 >
```

A continuación, se muestra una tabla con la descripción la funcionalidad más importante de la RT para el uso de cuaternios (en la ayuda de la RT se encuentra descrita en detalle toda la funcionalidad).

Función	Descripción
`+`	adición
`-`	substracción
`*`	multiplicación
`/`	división entre cuaternios o con escalar
`inv`	invertir un cuaternio
`norm`	magnitud de un cuaternio
`unit`	cuaternio unitario
`plot`	plotear la orientación del cuaternio
`R`	transformar a MR
`T`	transformar a TH

5. Modelado de un robot antropomórfico: definición de articulaciones y parámetros cinemáticos.

5.1. Introducción a la cinemática de un robot serial

Los robots seriales presentan una arquitectura antropomórfica similar al brazo humano. Consisten en una serie de barras rígidas unidas entre sí a través de articulaciones de un grado de libertad del tipo rotacional o prismático. Dentro de la cinemática de un robot, se presenta la resolución de dos problemas:

- Problema cinemático directo: determinar la posición y orientación del extremo final de la cadena cinemática conocidos las coordenadas articulares (**q**) y las características geométricas del robot.
- Problema cinemático inverso: determinar las variables articulares conocida la posición y orientación del extremo de la cadena cinemática.

Para la resolución de estos problemas se utiliza la representación *Denavit-Hartenberg* y las matrices de transformación homogénea. A continuación, se explican brevemente estos dos conceptos y cómo se resuelve la cinemática de cualquier brazo robot.

5.2. El problema cinemático directo: matriz de transformación y Denavit-Hartenberg

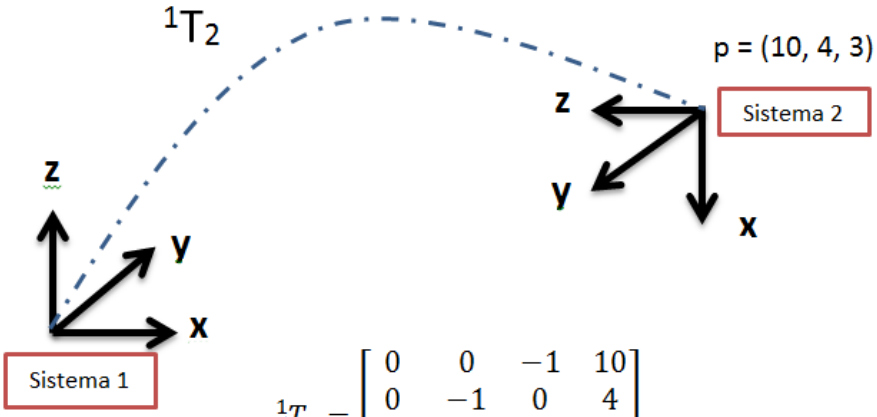
El problema cinemático directo se plantea encontrar una matriz de transformación que relaciona el sistema de coordenadas ligado al cuerpo en movimiento respecto a un sistema de coordenadas que se toma como referencia. Para lograr esta representación se usan las matrices de transformación homogénea 4x4 (TH), que incluye tanto traslación como rotación:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{p}_{3\times1} \\ \mathbf{f}_{1\times3} & \mathbf{w}_{1\times1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix}$$

En robótica la submatriz $\mathbf{f}_{1x3}$, que representa una transformación de perspectiva, es nula; y la submatriz $\mathbf{W}_{1x1}$, que representa un escalado global, es la unidad. Finalmente, la matriz de transformación homogénea queda de la forma siguiente:

$$\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde los vectores **n**, **o**, **a**, son vectores ortogonales unitarios que representan la orientación y **p** es un vector que describe la posición **x**, **y**, **z** del origen del sistema actual respecto del sistema de referencia. A continuación, se muestra un ejemplo para entender las propiedades de la matriz de TH.



$$^1T_2 = \begin{bmatrix} 0 & 0 & -1 & 10 \\ 0 & -1 & 0 & 4 \\ -1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Los valores de la submatriz de rotación de la matriz $^1T_2$ se han obtenido como si un observador localizado en el origen del Sistema 1, puede ver cómo están orientados los ejes **x**, **y**, **z** del Sistema 2. Con respecto a la posición, la submatriz de traslación da información de los que se ha desplazado en coordenadas cartesianas el origen del Sistema 2 respecto al Sistema 1.

Respecto a la funcionalidad de la RT para el uso de las matrices de TH, posee una extensa funcionalidad para la creación y realización de operaciones con las mismas. Al igual que se ha explicado con anterioridad con las MR, se describe la funcionalidad básica para la construcción de TH:

TH traslación *Función RT Matlab*

$$\mathbf{Tras}(p) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
T=transl(x, y, z)
```

TH rotación eje X, Y, Z *Función RT Matlab*

$$\mathbf{Rot}(x,\alpha)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\text{sen}\alpha & 0\\ 0 & \text{sen}\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

```
T=trotx(α)
```

$$\mathbf{Rot}(y,\beta)=\begin{bmatrix}\cos\beta & 0 & \text{sen}\beta & 0\\ 0 & 1 & 0 & 0\\ -\text{sen}\beta & 0 & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

```
T=troty(β)
```

$$\mathbf{Rot}(z,\gamma)=\begin{bmatrix}\cos\gamma & -\text{sen}\gamma & 0 & 0\\ \text{sen}\gamma & \cos\gamma & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

```
T=trotz(γ)
```

TH rotación eje X y traslación *Función RT Matlab*

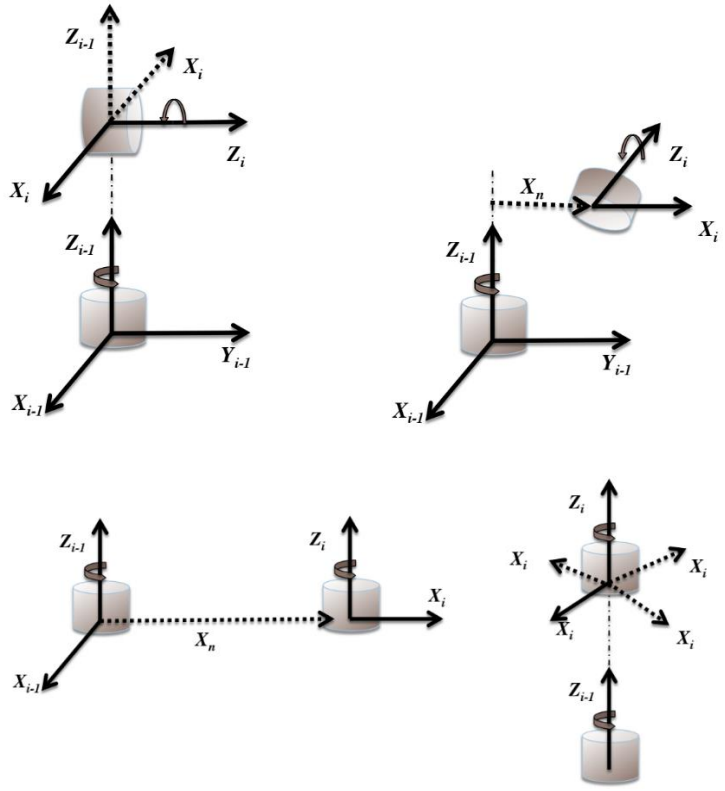
$$\mathbf{T}=\begin{bmatrix}1 & 0 & 0 & x\\ 0 & \cos\alpha & -\text{sen}\alpha & y\\ 0 & \text{sen}\alpha & \cos\alpha & z\\ 0 & 0 & 0 & 1\end{bmatrix}$$

```
T=trotx(α)*transl(x, y, z)
```

Función	Descripción
`angvec2tr`	forma angular/vectorial a TH
`eul2tr`	ángulos de Euler a TH
`rpy2tr`	ángulos roll/pitch/yaw a TH
`tr2angvec`	TH a forma angular/vectorial
`tr2eul`	TH a ángulos de Euler
`t2r`	TH a MR
`tr2rpy`	TH a ángulos roll/pitch/yaw
`trotx`	TH para rotación alrededor del eje X
`troty`	TH para rotación alrededor del eje Y
`trotz`	TH para rotación alrededor del eje Z
`transl`	TH de traslación
`trplot`	plotear el sistema de coordenadas de una TH

La representación Denavit-Hartenberg (DH) se aplica a robots de cadena cinemática abierta y consiste en una serie de reglas para colocar los sistemas de referencia de cada eslabón del robot. Para aplicar la metodología DH, es muy importante tener en cuenta los siguientes puntos:

- Se parte de una configuración cualquiera del robot, si bien es aconsejable colocarlo en una posición sencilla de analizar. Normalmente se coloca en la posición en la que todas sus variables articulares son cero **q**=[0, 0, ...,0].
- Se numeran los **eslabones**, asignando el 0 para la base y *n-1* para el último eslabón, siendo *n* el número de grados de libertad del robot.
- Identificar y numerar cada una de las **articulaciones** del robot, comenzando con 1 con la primera articulación y *n* para la última.
- Se selecciona un punto en la base para el sistema $\mathbf{S_0}$ = ($\mathbf{x_0}$, $\mathbf{y_0}$, $\mathbf{z_0}$). El sistema de coordenadas ortonormal dextrógiro de la base ($\mathbf{x_0}$, $\mathbf{y_0}$, $\mathbf{z_0}$) se establece con el eje $\mathbf{z_0}$ localizado a lo largo del eje del movimiento de la articulación **1**. El sistema ortonormal dextrógiro debe cumplir que $\mathbf{z_0} = \mathbf{x_0} \times \mathbf{y_0}$.
- El sistema de referencia $\mathbf{S_i}$ de cada eslabón se coloca al final del mismo, en el extremo de la articulación a la que está conectado el siguiente eslabón.
- El ángulo o desplazamiento de cada eslabón siembre se mide tomando como base el sistema de referencia del eslabón anterior.
- Al colocar el sistema de referencia del **eslabón-i**, se deben seguir las siguientes **reglas:**
    - Se colocan los ejes $\mathbf{z_i}$ alineados con el eje de movimiento de la articulación **i+1** (y marcando su dirección de movimiento positiva). Es decir, $\mathbf{z_0}$ se colocará describiendo el movimiento de la articulación 1, $\mathbf{z_1}$ con la articulación 2,...y así sucesivamente.
    - El eje $\mathbf{x_i}$ debe colocarse con orientación normal al plano formado por los ejes $\mathbf{z_{i-1}}$ y $\mathbf{z_i}$. Esto significa que el eje $\mathbf{x_i}$ debe ser paralelo al vector normal común, calculado como $\mathbf{x_n} = \mathbf{z_i} \times \mathbf{z_{i-1}}$, y que además corte a $\mathbf{z_i}$. A continuación, se muestran una serie de ejemplos de colocación de los ejes $\mathbf{x_i}$.



Además de los puntos anteriores, la metodología DH establece las siguientes condiciones para los demás parámetros geométricos (véase Figura 12), planteando el cambio entre el sistema de referencia **i-1** y el sistema **i** utilizando 4 transformaciones sucesivas:

- **θ_i**: ángulo de la articulación desde el eje **x_{i-1}** hasta el eje **x_i**, medido respecto al eje **z_{i-1}**, usando la regla de la mano derecha.
- **d_i:** distancia medida desde el origen del sistema **i-1**, a lo largo del eje **z_{i-1}** hasta la intersección del eje **z_{i-1}** con el eje **x_i**.
- **a_i:** distancia de separación entre los orígenes de los sistemas de referencia **i-1** e **i,** medida a lo largo del eje **x_i** hasta la intersección con el eje **z_{i-1}** (o la distancia más corta entre los ejes **z_{i-1}** y **z_i** cuando estos no se interceptan).
- **α_i**: ángulo que separa los ejes **z_i** y **z_{i-1}**, medido respecto del eje **x_i**.

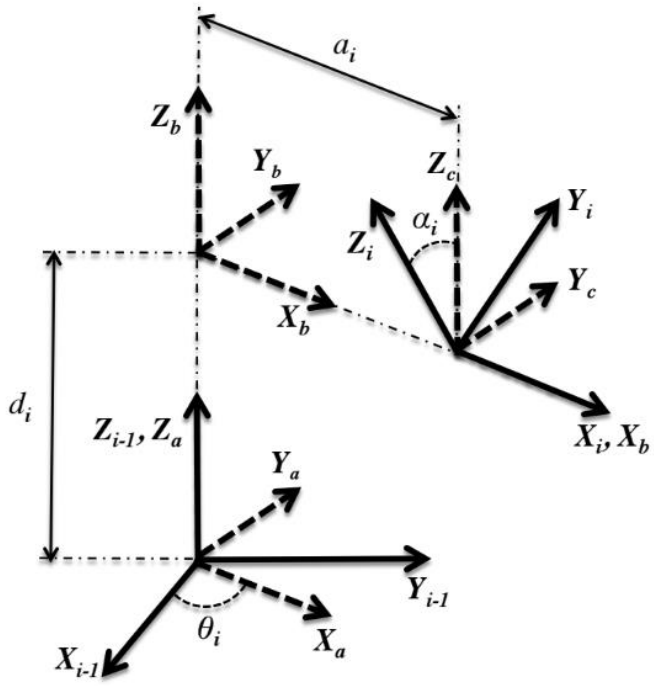


Figura 12. Parámetros de cambio entre sistemas de coordenadas

De esta manera, la metodología DH permite definir el cambio entre sistemas de referencia (i-1 , i) con 4 parámetros (θ, d, a, α), de los que uno de ellos lo constituye la variable articular o prismática que conecta ambos eslabones. A continuación, se muestra la matriz de transformación $^{i-1}\mathbf{A}_i$

$$ {}^{i-1}A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

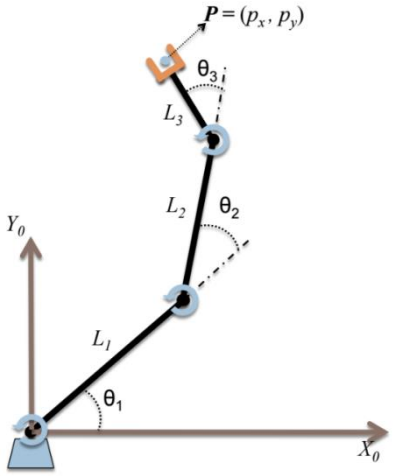
$$ {}^{i-1}A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Con todos los sistemas de referencias establecidos, se realiza una tabla de parámetros DH para ir de un sistema de referencia $\mathbf{S}_{i-1}$ a $\mathbf{S}_i$. Con esta tabla bien parametrizada, se resuelve el problema cinemático directo calculando la matriz de transformación T:

$$T = {}^{0}A_1 {}^{1}A_2 \cdots {}^{n-1}A_n$$

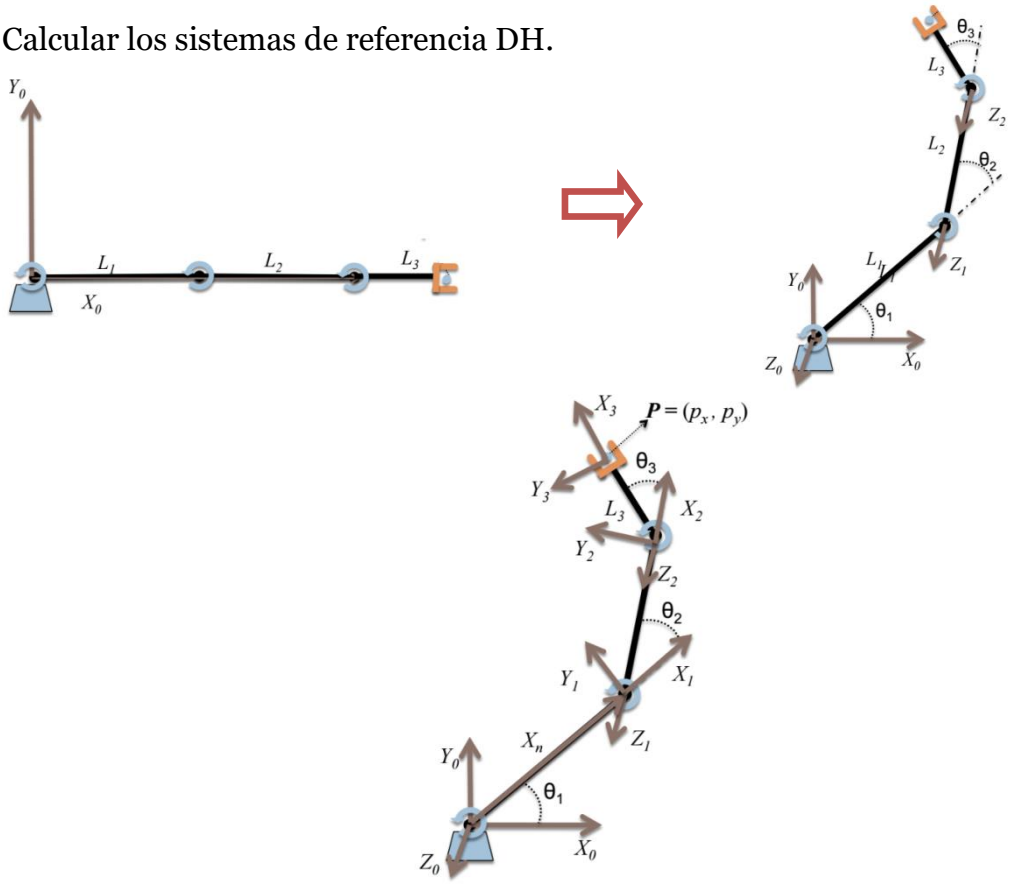
Ejemplo

Tabla de parámetros DH de un robot planar de 3 GDL (Grados De Libertad).



Los pasos para la resolución de la cinemática directa son: 1) calcular los sistemas de referencia DH; 2) Calcular los parámetros DH y realizar la tabla; 3) Calcular la matriz de transformación T.

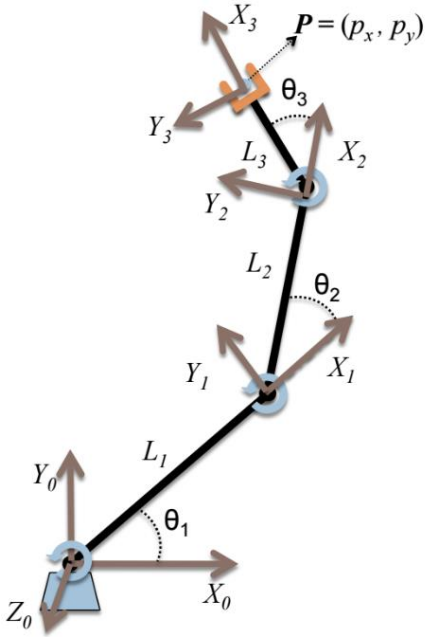
1) Calcular los sistemas de referencia DH.



2) Calcular los parámetros DH y realizar la tabla.

Transformación	θ	d	a	α
0 → 1 $^0A_1$	θ_1	0	L_1	0
1 → 2 $^1A_2$	θ_2	0	L_2	0
2 → 3 $^2A_3$	θ_3	0	L_3	0

3) Calcular la matriz de transformación $\mathbf{T} = {}^0\mathbf{A}_3$.



$$ {}^0A_1 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & l_1 \cdot \cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & l_1 \cdot \sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^1A_2 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & l_2 \cdot \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & l_2 \cdot \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

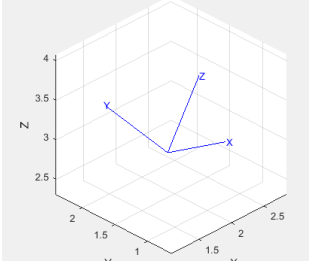
$$ {}^2A_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & l_3 \cdot \cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & l_3 \cdot \sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Ejercicio práctico 1: Mediante las funciones de las herramientas matemáticas, obtener la matriz de transformación y graficar el resultado que representa las siguientes transformaciones sobre un sistema OXYZ fijo de referencia: traslación de (-3,10,10); giro de -90º sobre el eje O'U del sistema trasladado y giro de 90º sobre el eje O'V' del sistema girado.

Ejemplo

A continuación, se muestra el resultado de una serie de transformaciones de rotación-traslación.

```
>> tr = trotx(.2)*troty(.3)*transl(1,2,3)

tr =

    0.9553         0    0.2955    1.8419
    0.0587    0.9801   -0.1898    1.4495
   -0.2896    0.1987    0.9363    2.9166
         0         0         0    1.0000
```



`trplot(tr)`

5.3. Implementación de la cinemática con la RT de robots seriales.

5.3.1. Creación del objeto *Link* y *Serial Link*

Para construir un robot se deben definir las distintas articulaciones que lo componen. Para definir una articulación en la Toolbox de robótica se utiliza la clase *Link*. Información sobre esta clase en la web http://www.petercorke.com/RTB/r9/html/Link.html.

- `>> L = Link([theta,d,a,alpha,sigma,mdh,offset,qlim])` donde:
  - `theta, d, a, alpha` son los parámetros del Denavit-Hartenberg.
  - `sigma` es el tipo de articulación: `sigma=0` (por defecto) es articulación rotacional, `sigma=1` prismática.
  - `mdh` parámetro que indica el tipo de parámetros DH: `mdh=0 (por defecto)` si es el sistema clásico de parámetros DH, ó `mdh=1` si es el modificado.
  - `offset` parámetro que indica un giro adicional sobre la variable articular.
  - `qlim` vector [min max] que indica los límites de la articulación.

Posteriormente, para construir un robot se deben definir las distintas articulaciones que lo componen. A partir de los distintos *Link* que conforman el robot, se crea un objeto *SerialLink*, que es la base del objeto robot.

- `>> robot = SerialLink (Links,'name','Nombre del robot')`

A continuación, se muestra la implementación mediante la RT del ejemplo del robot planar de 3 GDL (Grados De Libertad)con L1=L2=L3=1.

```
>> L1 = Link ([0 0 1 0]);
>> L2 = Link ([0 0 1 0]);
>> L3 = Link ([0 0 1 0]);
>> L=[L1 L2 L3];
>> robot = SerialLink(L,'name','Planar3GDL');
```

Mediante los siguientes comandos, se puede obtener información referente a la cadena cinemática o robot modelado:

- `>> robot`

```
robot =

Planar3GDL (3 axis, RRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          1|          0|          0|
|  2|         q2|          0|          1|          0|          0|
|  3|         q3|          0|          1|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =    0  base = 1  0  0  0   tool =  1  0  0  0
          0         0  1  0  0           0  1  0  0
       9.81         0  0  1  0           0  0  1  0
                    0  0  0  1           0  0  0  1
```

El comando `robot` muestra los parámetros Denavit-Hartenberg así como más información como la posición respecto al mundo de la base, y posición y orientación de la herramienta respecto al sistema de referencia del extremo. Además, también hace referencia al sistema de cinemática por defecto implementado (`stdDH`, standar DH) y al método de cálculo dinámico por defecto implementado (`fastRNE`, Newton-Euler).

- `>> robot.n`
- `>> robot.links`

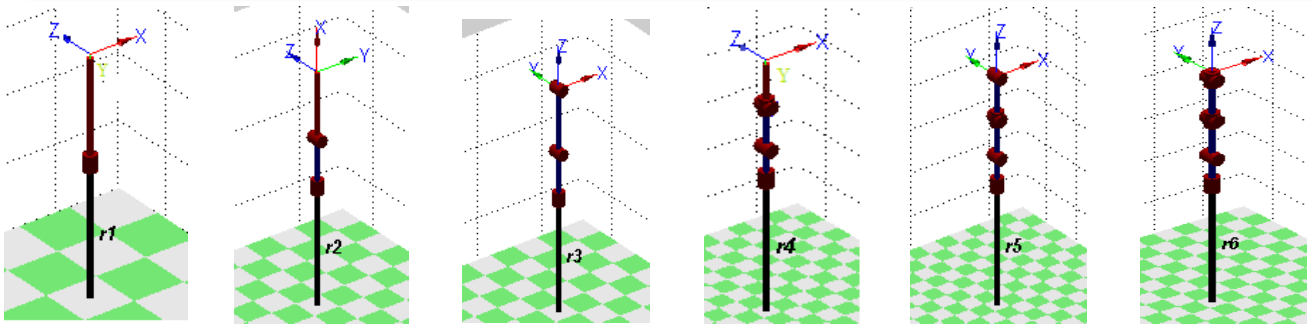
```
>> robot.n

ans =

     3

>> robot.links

ans =
 theta=q1, d=          0, a=          1, alpha=          0, offset=          0 (R,stdDH)
 theta=q2, d=          0, a=          1, alpha=          0, offset=          0 (R,stdDH)
 theta=q3, d=          0, a=          1, alpha=          0, offset=          0 (R,stdDH)
```

Por otro lado, `robot.n` muestra el número de GDL de nuestro robot, y la instrucción `robot.links` muestra los parámetros DH de cada articulación.

Ejercicio práctico 2: modelado del robot PA10 de 6GDL a partir de la siguiente tabla de sus parámetros DH estándar y los límites articulares. Para introducir los límites articulares y el offset de la articulación, mira en la web siguiente o teclea el comando "*help SerialLink*" http://www.petercorke.com/RTB/r9/html/Link.html.

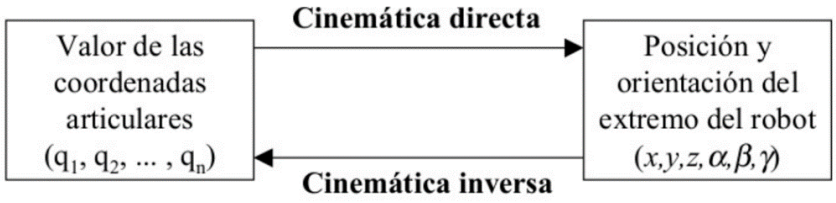
Transformación	Θ	d	a	α	Límite q(º)	Offset
0 → 1 $^0A_1$	q_1	0.317	0	-pi/2	[-177,177]	0
1 → 2 $^1A_2$	q_2	0	0.45	0	[-64,124]	-pi/2
2 → 3 $^2A_3$	q_3	0	0	pi/2	[-107,158]	pi/2
3 → 4 $^3A_4$	q_4	0.48	0	-pi/2	[-255,255]	0
4 → 5 $^4A_5$	q_5	0	0	pi/2	[-165,165]	0
5 → 6 $^5A_6$	q_6	0.07	0	0	[-255,255]	0

Nota: para poder visualizar el funcionamiento los parámetros DH, podéis ir creando objetos robot con 1, 2, 3,...hasta 6 eslabones e ir visualizando el cambio en los sistemas de referencia.



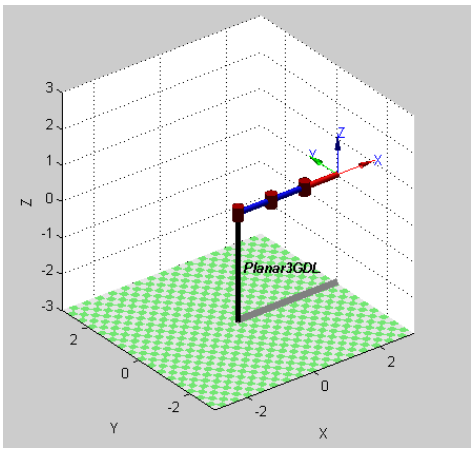
5.3.2. Cinemática directa e inversa

Tal y como se ha comentado, la **cinemática directa** proporciona la posición y orientación del extremo del robot en función de las coordenadas articulares de la cadena cinemática. Con respecto a la **cinemática inversa**, permite obtener las coordenadas articulares correspondientes a una determinada posición y orientación del extremo del robot.



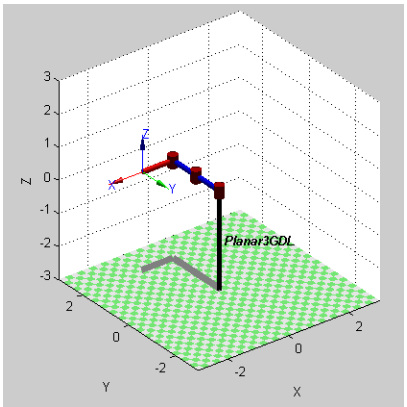
Mediante la RT es posible resolver la cinemática directa mediante la función de la clase objeto robot *fkine*. A continuación, se muestra un ejemplo de la cinemática directa del robot planar de 3 GDL (véase apartado anterior). En este ejemplo, en primer lugar se define un vector de coordenadas articulares **q**, se calcula la matriz de transformación **T** mediante la cinemática directa *fkine* y posteriormente se dibuja el robot mediante la función *plot* (**q**).

```
>> q = [0 0 0];
>> T = robot.fkine(q)

T =

     1     0     0     3
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> robot.plot(q)
```



Como se puede observar en la representación del robot, la matriz de transformación **T** muestra cómo el extremo se encuentra posicionado respecto al sistema de referencia de la base del robot. En este caso, no existe rotación en los ejes y sólo hay un desplazamiento en el eje X de 3 unidades (la suma de la longitud de los 3 eslabones del robot L1, L2 y L3). A continuación, se muestra otro ejemplo del robot planar con un valor de **q** = [pi/2 0 pi/2].

```
>> q = [pi/2 0 pi/2];
>> T = robot.fkine(q)

T =

   -1.0000   -0.0000         0   -1.0000
    0.0000   -1.0000         0    2.0000
         0         0    1.0000         0
         0         0         0    1.0000

>> robot.plot(q);
```



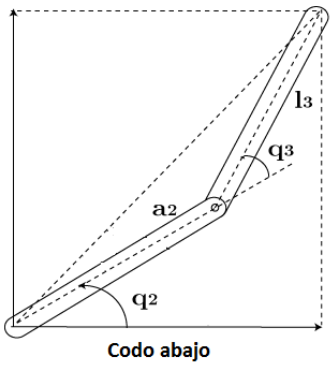
En este ejemplo, se puede observar cómo existe una rotación de los ejes del extremo respecto a la base del robot, y un desplazamiento negativo en X de -1 y uno positivo de 2 en el eje Y.

Ejercicio práctico 3: definir las siguientes posiciones articulares para el PA10 (las posiciones se indican en grados, pero en Matlab hay que introducirlas en radianes), calcular la cinemática directa (matriz **T**) para cada uno de ellos y realizar un *plot* en esa posición.

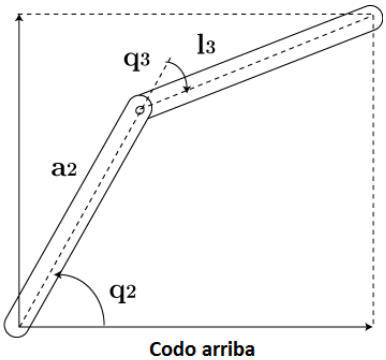
Posición de home: q_h = [0, 0, 0, 0, 0, 0].
Posición de escape: q_e = [0, 30, 90, 0, 60, 0].
Posición de seguridad: q_s = [0, 45, 90, 0, -45, 0].
Posición q_1 = [0, 45, 45, 0, 90, 0].
Posición q2 = [20, 90, 45, -22.5, 60, 0].

Con respecto a la **cinemática inversa**, este problema consiste en hallar los valores de las coordenadas articulares del robot **q** = [q_1,...,q_n] conocida la posición y orientación del extremo del robot. Contrariamente a la cinemática directa, la resolución de la cinemática inversa puede dar lugar a diferentes soluciones. Para la resolución de este problema, se pueden encontrar diversos métodos, aunque frecuentemente se resuelve por medio de métodos geométricos o métodos numéricos basados en el Jacobiano (véase punto 5). La mayor parte de los robots de 6 GDL suelen tener cadenas cinemáticas no muy complejas, que facilitan la resolución de la cinemática inversa, ya que los tres primeros ejes son para posicionar el robot en un punto del espacio cartesiano (estructura planar) y los tres últimos ejes suelen emplearse para la orientación de la herramienta. Esto permite una resolución del problema desacoplada de la posición de la muñeca del robot y de la orientación de la herramienta (principio de Pieper's).

Como se ha comentado con anterioridad, los robots de 6 GDL, poseen una estructura de robot planar en los 3 primeros ejes. La resolución de la cinemática inversa de esta estructura puede dar a diferentes soluciones de valores articulares **q** para obtener la misma posición (x,y,z). Estas dos soluciones suelen denominarse configuraciones de **codo arriba** y **codo abajo**, que representan los casos en que la articulación 3 está situada por encima o por debajo de la articulación 2, para lograr la misma posición del punto cartesiano.

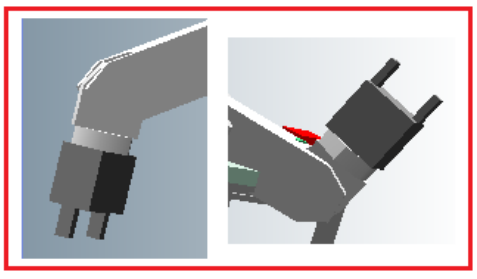


Codo abajo



Codo arriba

Asimismo, la muñeca también puede tomar dos configuraciones distintas, **muñeca arriba** y **muñeca abajo** para obtener la misma orientación del efector final.

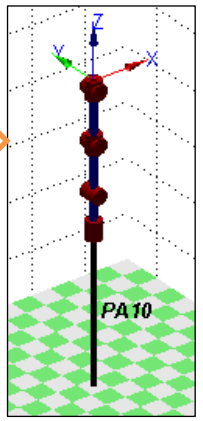


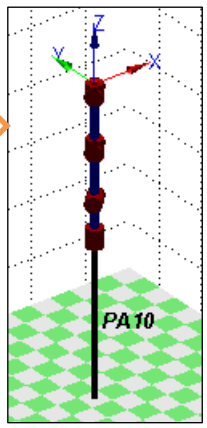
Para la resolución de la cinemática inversa se va a hacer uso de diferentes funciones de la RT. Las funciones para esta resolución permiten obtener las coordenadas articulares a partir de una determinada posición y orientación del extremo del robot. La librería de robótica permite utilizar diversas funciones de resolución numérica de la cinemática inversa:

- `>> robot.ikine6s(T);`
- `>> robot.ikunc(T);`

A continuación, se muestra un ejemplo de resolución de la cinemática inversa empleando el robot PA10. Para la resolución y verificación de este problema, primero se calcula la matriz **T** para una determinada posición articular **q**, y posteriormente, para esa **T** obtenida se calculan las **q** ($q_{inversa}$) con el método de la cinemática inversa *ikine6s*. Como se ha comentado, la resolución de la cinemática inversa puede dar lugar a diferentes soluciones, tal y como se muestra en el ejemplo, que el valor de las variables articulares son diferentes ($q_h \neq q_{inversa}$) aunque el valor de la matriz de transformación final es la misma ($T = T_{inversa}$).

```
>> qh

qh =

     0     0     0     0     0     0

>> T=robotPA10.fkine(qh)

T =

    1.0000         0         0    0.0000
         0    1.0000         0   -0.0000
         0         0    1.0000    1.3170
         0         0         0    1.0000

>> qinversa = robotPA10.ikine6s(T)

qinversa =

    2.3562         0   -0.0000   -3.1416   -0.0000    0.7854

>> robotPA10.plot(qinversa)
>> Tinversa=robotPA10.fkine(qinversa)

Tinversa =

    1.0000         0   -0.0000    0.0000
         0    1.0000   -0.0000   -0.0000
    0.0000    0.0000    1.0000    1.3170
         0         0         0    1.0000
```





Ejercicio práctico 4: realizar la resolución de la cinemática inversa para el resto de posiciones del PA10 (q_e, q_s, q_1, q_2) siguiendo el mismo procedimiento que en el ejemplo mostrado utilizando las funciones *ikine6s* e *ikunc*. Para más información de los métodos, se puede acceder mediante el comando *"help ikine6s"* y *"help ikunc"* en Matlab.

Para la resolución de la cinemática del robot a modelar, se recomienda el uso de la herramienta `robot.teach()`. Esta herramienta permite posicionar el robot en un punto del entorno de trabajo mediante una interfaz y visualizar los valores articulares y posición/orientación del extremo del robot. A continuación, se muestra dicha interfaz:

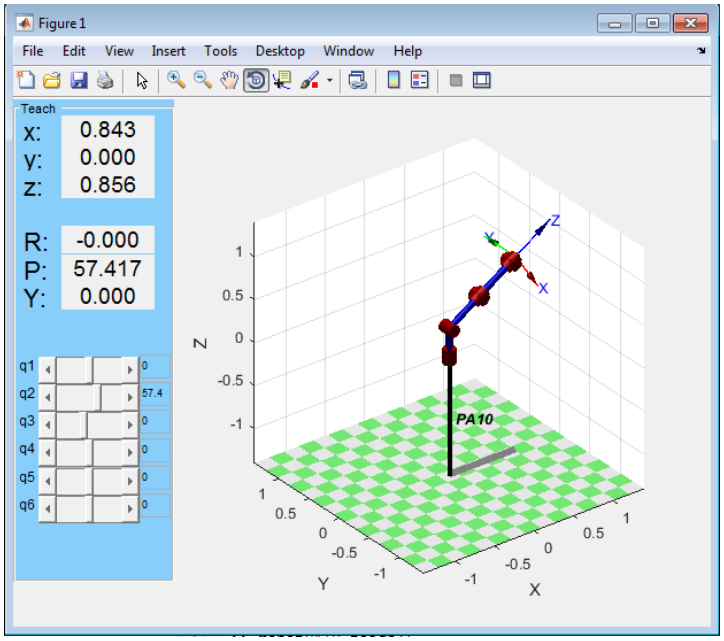


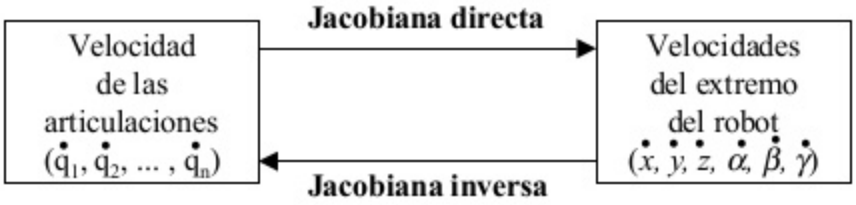
Figura 13. Interfaz de robot.teach() de la RT

6. Matriz Jacobiana

Se ha estudiado previamente que el modelado cinemático de un robot relaciona las variables articulares con la posición-orientación de su extremo. En esta relación no se tienen en cuenta las fuerzas o pares que actúan sobre el sistema (eso se estudia en el modelo dinámico). Además de esta relación entre las coordenadas articulares y las del extremo, la relación entre sus respectivas velocidades está relacionado mediante la **matriz Jacobiana**. Mediante esta matriz, se puede establecer qué velocidad debe fijarse para cada articulación de modo que se consiga en el extremo una trayectoria temporal concreta (velocidades en el extremo). Como se puede ver en la imagen siguiente, existen dos matrices Jacobianas, el Jacobiano directo y el inverso.

La matriz Jacobiana o Jacobiano (**J**) es importante para el análisis y control del movimiento de un robot ya que, además de relacionar las velocidades de las articulaciones con las del extremo, permite dar información sobre el área de trabajo y las singularidades del robot. El valor numérico de la matriz **J** dependerá de los valores instantáneos de las coordenadas articulares, por lo que el valor de J será diferente en cada uno de los puntos del espacio articular. Las singularidades del robot se obtienen cuando el determinante de la matriz Jacobiana es nulo. En las inmediaciones de los puntos singulares, se pierde alguno de los grados de libertad del robot, siendo imposible que el extremo del robot se mueva en una determinada dirección cartesiana (por ejemplo, en una alineación de ejes). Se distinguen dos tipos de singularidades:

- Singularidades en los límites del espacio de trabajo.
- Singularidades en el interior del espacio de trabajo.



$$V_e = \boldsymbol{J} * \dot{q}$$

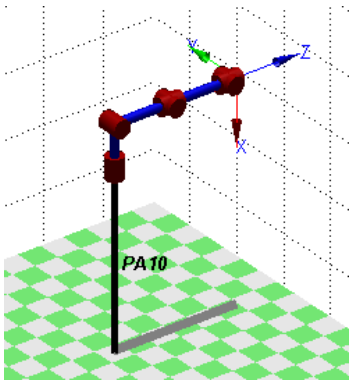
$det[\boldsymbol{J}] = 0 \rightarrow$ Singularidades del robot

Para calcular el Jacobiano con la RT de Matlab, se emplean las siguientes funciones:

- Función *jacobo (robot, q)*: relaciona las velocidades articulares con los vectores de velocidad lineal y angular ($\boldsymbol{v}$, $\boldsymbol{w}$) con que se mueve el extremo del robot expresado en el sistema de referencia de la base.
- Función *jacobn (robot, q)*: relaciona las velocidades articulares con las velocidades de la localización del extremo del robot, siendo ésta la posición y orientación expresada en base a sus coordenadas cartesianas y a sus ángulos de Euler ($\dot{x}, \dot{y}, \dot{z}, \dot{\emptyset}, \dot{\theta}, \dot{\gamma}$)

Para comprobar el funcionamiento de las funciones del Jacobiano, se pasa un vector de coordenadas **q** para el que el extremo se encuentra en el límite de su espacio de trabajo o existen alineaciones de ejes (se pierden grados de libertad del robot). A continuación, se muestra un ejemplo para el robot PA10, donde se ha pasado una configuración articular donde el robot se encuentra al límite de su espacio de trabajo y totalmente en horizontal con una posición desfavorable en cuanto a alineación de ejes. Evidentemente, al calcular el determinante de la matriz Jacobiana, da un resultado nulo.

```
>> q = [0 pi/2 0 0 0 0];
>> J = jacob0(robotPA10,q)

J =

   -0.0000    0.0000    0.0000         0    0.0000         0
    1.0000    0.0000    0.0000         0    0.0000         0
    0.0000   -1.0000   -0.5500         0   -0.0700         0
         0         0         0    1.0000         0    1.0000
    0.0000    1.0000    1.0000    0.0000    1.0000    0.0000
    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000

>> det(J)

ans =

     0
```



Ejercicio práctico 5: evalúa al robot PA10 y al robot planar en otras posiciones al límite de su espacio de trabajo o donde existan alineaciones de ejes (puedes emplear la función *rand* para probar diferentes posiciones). Para el robot planar, sólo ten en cuenta las dos primeras filas y la última de la matriz Jacobiana, ya que el resultado no es una matriz cuadrada, y sólo es necesario evaluar el espacio cartesiano plano y uno de los vectores de orientación del robot en el plano (el robot planar sólo puede posicionarse y orientarse en el plano).

El valor del determinante de la matriz Jacobiana también se emplea para saber si la posición evaluada se encuentra cerca o lejos de una singularidad. Para los robots no redundantes (como un robot de 6 GDL) el determinante del Jacobiano también se le conoce como manipulabilidad. Este valor se puede calcular con la función `robot.maniplty(q)`.

7. Dinámica de un robot antropomórfico

La dinámica del robot relaciona el movimiento del robot y las fuerzas implicadas en el mismo. El modelo dinámico establece relaciones matemáticas entre las coordenadas articulares (o las coordenadas del extremo del robot), sus derivadas (velocidad y aceleración), las fuerzas y pares aplicados en las articulaciones (o en el extremo) y los parámetros del robot (masas de los eslabones, inercias, etc). El modelo dinámico del robot se puede representar mediante la siguiente ecuación cerrada:

$$Q = M(q) \cdot \ddot{q} + C(q, \dot{q}) \cdot \dot{q} + F(\dot{q}) + G(q) + J(q)^T f$$

Donde **Q** son los pares articulares efectivos en cada uno de los eslabones, **M** es la matriz de inercia, **C** representa la matriz columna de fuerzas de Coriolis y centrípeta, **F** es la columna referida al rozamiento viscoso de las articulaciones, **G** identifica a la matriz de gravedad y finalmente el término $\boldsymbol{J}(q)^T \boldsymbol{f}$ es el término referido a posibles fuerzas externas.

Para el cálculo de las fuerzas generadas en un robot, existen diferentes algoritmos, distinguiendo a los algoritmos de formulación de ecuaciones diferenciales, como es Lagrange-Euler, y los que están basados en algoritmos recursivos, como Newton-Euler y Walker-Orin. La formulación Lagrange-Euler presenta un modelo simple, resultando una serie de ecuaciones diferenciales no lineales de 2º orden acopladas útiles para el estudio de estrategias de control en el espacio de estados de las variables articulares del robot, pero que se presentan ineficaces para aplicaciones en tiempo real dado el elevado tiempo de computación que requieren las operaciones con matrices de transformación homogénea.

Los modelos dinámicos que se estudian en esta práctica están basados en el algoritmo recursivo Newton-Euler y Walker-Orin, que permiten obtener y lograr tiempos de cálculo rápidos en la evaluación de los pares y fuerzas articulares para controlar el manipulador. Este tipo de algoritmo es muy diferente a la formulación de Lagrange-Euler, donde se obtiene ecuaciones no lineales para el análisis de control, con un fuerte acoplamiento en las variables del sistema (fuerza, par, velocidad y aceleración).

7.1. Introducción de los parámetros dinámicos

Antes de resolver el modelo dinámico del robot, es necesario introducir los parámetros dinámicos de cada *Link* que compone el objeto robot. A continuación, se muestra una lista de los parámetros dinámicos:

- **m**: masa del eslabón.
- **r**: centro de gravedad del eslabón con referencia al sistema de referencia del eslabón (vector 3x1).
- **I**: matriz de inercia del eslabón con respecto a centro de gravedad (matriz simétrica de tamaño 3x3).
- $\mathbf{B_m}$: rozamiento viscoso de la articulación (referido al motor).
- $\mathbf{T_c}$: rozamiento de Coulomb de la articulación.
- **G**: parámetro de reducción del accionamiento (si existe un moto-reductor).

- **J_m**: inercia del motor (referida al motor).

Los parámetros más importantes son **m**, **r** e **I**. Los parámetros dinámicos se introducen en cada uno de los *Links* que forma el objeto robot *SerialLink*.

Properties (read/write)

theta	kinematic: joint angle
d	kinematic: link offset
a	kinematic: link length
alpha	kinematic: link twist
sigma	kinematic: 0 if revolute, 1 if prismatic
mdh	kinematic: 0 if standard D&H, else 1
offset	kinematic: joint variable offset
qlim	kinematic: joint variable limits [min max]
m	dynamic: link mass
r	dynamic: link COG wrt link coordinate frame 3x1
I	dynamic: link inertia matrix, symmetric 3x3, about link COG.
B	dynamic: link viscous friction (motor referred)
Tc	dynamic: link Coulomb friction
G	actuator: gear ratio
Jm	actuator: motor inertia (motor referred)

A continuación, se muestra una función en código Matlab que introduce los parámetros dinámicos de un robot. En concreto, los datos son referidos al robot PA10.

```
function robot = DynamicParams(robot)

 % Vectores desde el SR DH al CDG del eslabón
 rdata = zeros (6,3);
 rdata(1,:) = [0,0.1170,0];
 rdata(2,:) = [-0.3210,-0.0300,0];
 rdata(3,:) = [0,0, 0.0485];
 rdata(4,:) = [0,0.1122,0];
 rdata(5,:) = [0,0,-0.0420];
 rdata(6,:) = [0,0,-0.0480];

 IData = LoadInertiaData; % Intoducimos en IData los valores de Inercia

 mass = [9.29, 12.43, 4.86, 3.08, 2.07, 1.05];   % masas de los eslabones
 Ixx = IData(:,1)';   % Valores de las matrices de inercia
 Iyy = IData(:,2)';
 Izz = IData(:,3)';
 Ixy = IData(:,4)';
 Iyz = IData(:,5)';
 Ixz = IData(:,6)';
 Jm = zeros(1,6);         % Inercia del motor
 G = 50 * ones(1,6);      % Coeficiente de reducción del moto-reductor
 B = zeros(1,6);          % Fricción viscosa
 Tc_plus = zeros(1,6);    % Fricción de Coulomb
 Tc_minus = zeros(1,6);   % Fricción de Coulomb


 for i=1:6
     robot.links(i).m = mass(i);
     robot.links(i).r = rdata(i,:);
     robot.links(i).I = [Ixx(1,i) Ixy(1,i) Ixz(1,i);
                         Ixy(1,i) Iyy(1,i) Iyz(1,i);
                           Ixz(1,i) Iyz(1,i) Izz(1,i)];
     robot.links(i).Jm = Jm(i);
     robot.links(i).G = G(i);
     robot.links(i).B = B(i);
     robot.links(i).Tc = [Tc_plus(i); Tc_minus(i)];
 end
```

Al ejecutar este código, los datos del robot PA10 deben quedar como se muestra en la siguiente figura. Se pueden comprobar accediendo a los parámetros de la clase Link del objeto robot (por ejemplo, para visualizar el valor de la matriz de inercia del segundo eslabón se emplea el comando `robot.links(2).I`).

Link 1
m = 9.290000
r = 0.000000 0.117000 0.000000
I = | 0.106822 -0.000000 -0.000000 |
| -0.000000 0.052800 -0.000000 |
| -0.000000 -0.000000 0.130758 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

Link 2
m = 12.430000
r = -0.321000 -0.030000 0.000000
I = | 0.378280 -0.020125 0.000295 |
| -0.020125 0.064096 0.000876 |
| 0.000295 0.000876 0.375820 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

Link 3
m = 4.860000
r = 0.000000 0.000000 0.048500
I = | 0.059857 0.000000 0.000000 |
| 0.000000 0.057734 -0.001817 |
| 0.000000 -0.001817 0.008374 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

Link 4
m = 3.080000
r = 0.000000 0.112200 0.000000
I = | 0.028269 0.000000 0.000000 |
| 0.000000 0.010902 -0.000000 |
| 0.000000 -0.000000 0.019938 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

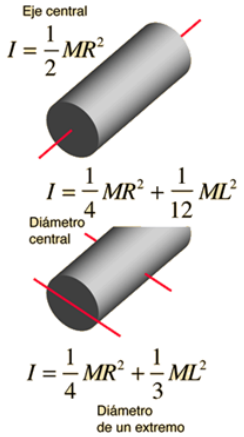
Link 5
m = 2.070000
r = 0.000000 0.000000 -0.042000
I = | 0.007413 0.000000 -0.000016 |
| 0.000000 0.006985 0.000000 |
| -0.000016 0.000000 0.003019 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

Link 6
m = 1.050000
r = 0.000000 0.000000 -0.048000
I = | 0.001350 0.000000 0.000000 |
| 0.000000 0.001350 0.000000 |
| 0.000000 0.000000 0.000583 |
Jm = 0.000000
Bm = 0.000000
Tc = 0.000000(+) 0.000000(-)
G = 50.000000

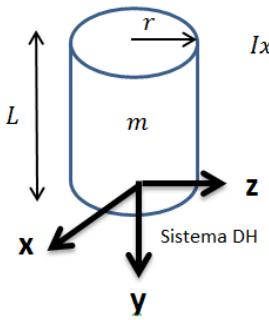
Si se desea modelar la dinámica de un robot comercial, donde normalmente sólo existe información de masa de los eslabones, se puede realizar una aproximación de los datos dinámicos de la siguiente manera.

- **r**: calcular el vector desde el sistema DH al centro geométrico del eslabón (se considera el centro geométrico como el centro de gravedad – CDG).
- **I**: considerar el eslabón como si fuera un cilindro, con las dimensiones del eslabón. Calcular el momento de inercia del cilindro con respecto al sistema DH. A continuación, se muestra un ejemplo.

Información sobre momentos de inercia del cilindro



Cálculo del momento de inercia del eslabón con respecto al sistema DH



$$Ixx = Izz = \frac{1}{4} * m * r^2 + \frac{1}{3} * m * L^2$$

$$Iyy = \frac{1}{2} * m * r^2$$

- **B_m**: se puede considerar un rozamiento de 0.05 en cada una de las articulaciones.
- **T_c**: se puede considerar un rozamiento de Coulomb de 0.01 en cada una de las articulaciones.
- **G y J_m**: son parámetros del motor y del mecanismo de reducción. Si se disponen de los datos, se introducen. En caso contrario, se introducen valores nulos.

7.2. Dinámica inversa

La dinámica inversa expresa las fuerzas y pares que intervienen en función de la evaluación temporal de las coordenadas articulares. Es decir, consiste en obtener la fuerza que debe existir en cada articulación prismática y el par, en cada articulación rotacional, para el movimiento del extremo del robot se produzca con las velocidades y aceleraciones deseadas. Mediante el algoritmo Newton-Euler se puede obtener un conjunto de ecuaciones recursivas, que permiten en primer lugar, calcular velocidad y aceleración lineal/angular de las articulaciones en base a cada sistema de referencia articular, y en segundo lugar, el cálculo de los pares y fuerzas necesarios para cada articulación desde el extremo hasta el sistema referencia base. Para el cálculo de la dinámica inversa (fuerzas/pares en articulaciones) con la RT, se emplea la siguiente función:

- `>> tau = robot.rne(q0, v0, a0).` Esta función permite obtener los pares articulares a partir de las coordenadas articulares del robot, la velocidad articular y la aceleración articular.

Para calcular los pares que se están ejerciendo sobre una posición en concreto, hay que emplear esta función con velocidad y aceleración nulas. Por ejemplo, para las posiciones q_h y q_e para el robot PA10 los pares resultantes en las articulaciones son los siguientes:

```
---------------- POSICION HOME ------------------------
njoints 6
Fast RNE: (c) Peter Corke 2002-2011

tau =

         0    3.6581         0         0         0

---------------- POSICION ESCAPE ------------------------
njoints 6

tau =

    0.0000  -53.4590  -24.3498    0.0000    0.0000
```

Ejercicio práctico 6: calcula los pares articulares del resto de posiciones del robot PA10 (q_s, q_1 y q_2) utilizando el comando `robot.rne(q0, v0, a0)`.

Nota: por defecto, la RT calcula la dinámica inversa de un robot empleando un método rápido llamado *fastRNE*. Si el comando `robot.rne` no funcionara correctamente (da un error que cierra Matlab), será necesario cambiarlo mediante la opción `robot.fast` al método *slowRNE*. Para ello se pone el valor de `robot.fast` a 0 (`robot.fast=0`).

7.3. Cálculo de las matrices del modelo dinámico y carga externa

La RT de Matlab, también proporciona una serie de funciones para calcular el par debido a cada una de las matrices del modelo dinámico. A continuación, se describen diversas funciones de Matlab para el tratamiento de dichas matrices y características asociadas:

- Par debido a la gravedad **G** (término $G(q)$). La componente gravitacional de la dinámica del robot se calcula utilizando la función `gravload` del *SerialLink*.
  - `>> G = robot.gravload(q0)`

  Por defecto, al construir el *SerialLink* se toma la gravedad de la Tierra. Sin embargo, se puede estudiar el comportamiento dinámico del robot cambiando el valor del término de gravedad mediante la propiedad `robot.gravity`. A continuación, se muestra un ejemplo de cómo la componente gravitacional al cambiar la gravedad de la Tierra por la gravedad de la Luna (que es de 1/6 la gravedad terrestre).

```
>> robotPA10.gravity=[0 0 9.81/6]

robotPA10 =

PA10 (6 axis, RRRRRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|      0.317|          0|     -1.571|          0|
|  2|         q2|          0|       0.45|          0|     -1.571|
|  3|         q3|          0|          0|      1.571|      1.571|
|  4|         q4|       0.48|          0|     -1.571|          0|
|  5|         q5|          0|          0|      1.571|          0|
|  6|         q6|       0.07|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =     0  base = 1  0  0  0   tool =  1  0  0  0
           0         0  1  0  0           0  1  0  0
       1.635         0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1
```

- Par debido a la inercia **M** (término $M(q)\cdot\ddot{q}$). La componente inercial de la dinámica del robot se calcula utilizando la función `itorque` del *SerialLink*.
  - `>> M = robot.itorque(q0,qdd)`

  Esta función calcula el par debido a la inercia, que depende de las coordenadas articulares y de la aceleración.
- Matriz de Coriolis **C** (término $C(q,\dot{q})$). Esta matriz de dimensión nxn (siendo n el número de GDL) representa el efecto, en cuanto a fuerza o par, generado sobre un eslabón como consecuencia del movimiento relativo entre los otros eslabones. La matriz de Coriolis de la dinámica del robot se calcula utilizando la función coriolis del *SerialLink*.
  - `>> C = robot.coriolis(q0, qd)`

  Esta matriz depende de las coordenadas articulares como las anteriores, pero también de la velocidad articular. Para calcular el par debido a Coriolis, se debe multiplicar esta matriz **C** por la velocidad articular.
- Carga en el extremo del robot. Para añadir una carga en el extremo del robot para simular que el robot ha agarrado un objeto, se puede utilizar la función `payload` de la clase *SerialLink*.
  - `>> robot.payload(2.5, [0, 0, 0.1]).` Esta orden añade una masa de 2.5 Kg al extremo del robot separada 10 cm (0.1 m) de éste en el eje Z. Una masa no se puede colocar en el mismo extremo, siempre tiene que tener un desplazamiento.

Ejercicio práctico 7: Calcula los resultados dinámicos (par articular, par de gravedad, par de coriolis, par de inercia) para distintas posiciones con el valor de la gravedad en la Luna (g=1,62 m/s²). Justifica los resultados.

Ejercicio práctico 8: ¿Cómo afecta añadir una carga de este tipo a la componente gravitacional e inercial? ¿Y si la separamos también 0.3 m en el eje X? ¿Añadir una carga afectará sólo a la componente gravitacional? Justifica las respuestas haciendo uso del robot PA10.

8. Planificación de trayectorias de robots antropomórficos

Una vez obtenidos los modelos cinemáticos y dinámicos del robot, se puede abordar el problema del control. El control del movimiento del robot implica controlar dicho robot de manera que siga un camino pre-planificado. El objetivo del control del robot es establecer las trayectorias para conseguir los objetivos fijados, a la vez que exige cumplir una serie de restricciones físicas impuestas por los actuadores y de precisión.

El problema del control del robot se suele dividir en dos bloques:

- Control cinemático o planificación de trayectorias. Consiste en describir el movimiento deseado del manipulador como una secuencia de puntos en el espacio (con posición y orientación). El control cinemático interpola el camino deseado mediante una serie clase de funciones polinomiales y genera una secuencia de puntos a lo largo del tiempo.
- Control dinámico o control de movimiento. Trata de conseguir que el robot siga realmente las trayectorias marcadas por el control cinemático teniendo en cuenta las limitaciones de los actuadores y el modelo dinámico del robot.

Esta práctica aborda el control cinemático del robot, realizando trayectorias articulares y cartesianas con las siguientes funciones de la RT.

- En el espacio articular:
  - `>> q = mtraj(@tpoly, q1, q2, t)`. Esta orden realiza una trayectoria articular entre $\mathbf{q_1}$ y $\mathbf{q_2}$ mediante una interpolación polinomial y devuelve el vector de valores articulares **q**. El valor de **t** se proporciona en número de muestras en la interpolación. Por defecto, el orden del polinomio de interpolación es de quinto orden, y equivale a la orden siguiente: `jtraj(q1, q2, t)`.
  - `>> q = mtraj(@lspb, q1, q2, t)`. Esta orden realiza una trayectoria articular entre **q1** y **q2** mediante una interpolación trapezoidal.
  - `>> [q, qd, qdd] = jtraj(q1, q2, t)`. Las funciones de interpolación articulares también devuelven el valor de las velocidades y aceleraciones articulares. Con el comando anterior, se obtiene el vector de velocidad articular **qd** y el vector de aceleración articular **qdd**.

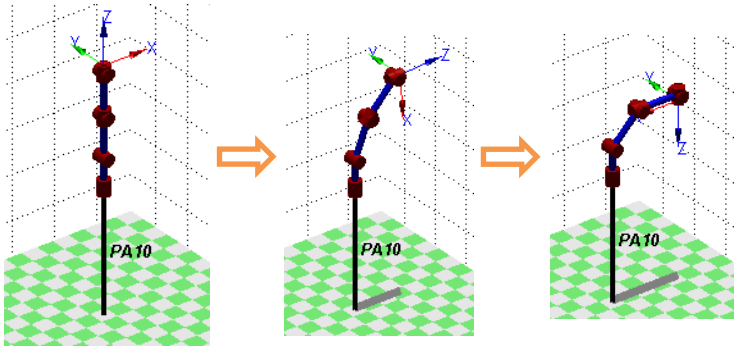
- En el espacio cartesiano:
  - `>> Ts = ctraj(T1, T2, length(t))`. Esta orden calcula una trayectoria cartesiana entre las poses T1 y T2 mediante un perfil trapezoidal. El resultado es un vector que contiene todas las poses en los puntos intermedios (Ts). Para visualizar el movimiento del robot en el espacio articular, se tienen que calcular los valores de **q** mediante la cinemática inversa (*ikine6s* y/o *ikunc*).

Por otro lado, la función *plot* de la clase *SerialLink* permite obtener una animación del robot a partir del vector de valores articulares.

- `>> robot.plot(q).`

Si **q** es una configuración articular se representa el robot en esa posición. Si **q** representa un conjunto de configuraciones articulares en el tiempo, se representa en la figura el movimiento del robot según ese conjunto de coordenadas articulares. A continuación, se muestra un ejemplo del cálculo de una trayectoria articular del robot PA10 entre $\mathbf{q}_h$ y $\mathbf{q}_1$.

```
>> [q, qd, qdd] = jtraj(qh, q1, 50);
>> robotPA10.plot(q)
```



Ejercicio práctico 9: Realiza 3 trayectorias articulares con el robot PA10 entre diferentes puntos probando el perfil trapezoidal y polinomial. Para visualizar los valores de velocidad y aceleración puedes emplear el comando `plot(qd)`. Realiza 3 trayectorias cartesianas con el robot PA10 cambiando los valores de la posición cartesiana del robot. Para todas las trayectorias, representa gráficamente los valores de las posiciones en los tres ejes del espacio cartesiano X Y Z a lo largo de la trayectoria y los valores de su jacobiano (determinante matriz **J**).

9. Cinemática de robots móviles

Dentro de esta sección, se va a tratar las funcionalidades de la RT de Matlab para el modelado cinemático de robots móviles. Previamente, se va a realizar una breve descripción de la parte del modelo, y posteriormente cómo emplearlo mediante la RT.

El modelo cinemático básico normalmente usado para un robot móvil o vehículo de cuatro ruedas es el modelo de ruedas direccionales o tipo *bicycle* mostrado en la Figura 14. Este tipo de robots autónomos posee unas ruedas traseras fijas al cuerpo y las ruedas delanteras son las direccionales, que giran alrededor de su eje vertical para dirigir el vehículo.

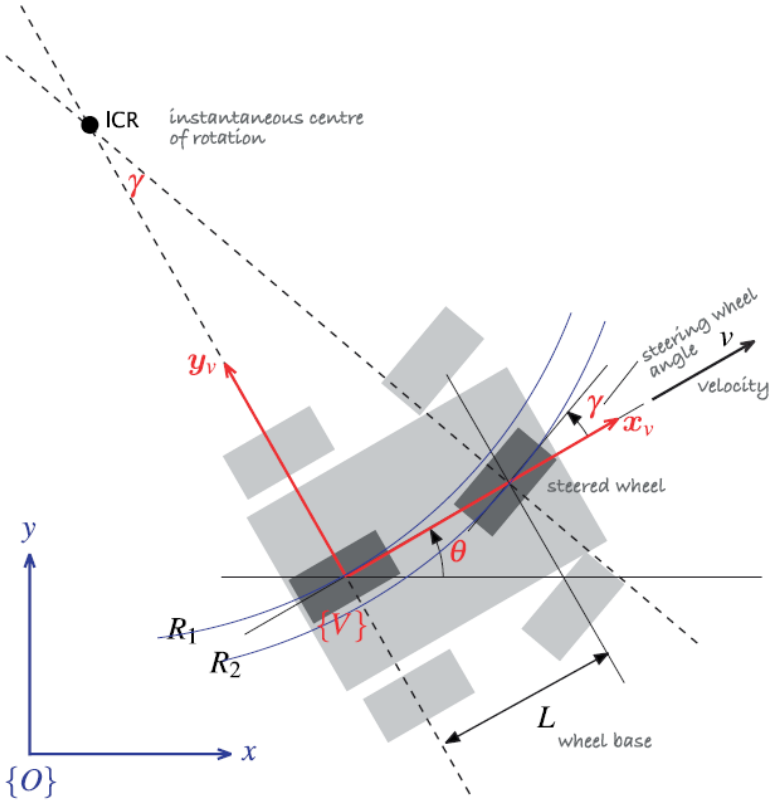


Figura 14. Modelado cinemático de un robot móvil con ruedas direccionales

La configuración del robot está representada por las coordenadas generalizadas (x, y, ϑ). La velocidad del vehículo v actúa en la dirección del eje X (por lo tanto, es nula en el eje Y), ya que las ruedas no pueden deslizarse hacia los lados. Las líneas discontinuas muestran la dirección a lo largo de la que las ruedas no pueden moverse e intersecan en un punto conocido como el Centro Instantáneo de Rotación (ICR). El punto de referencia del vehículo sigue así una trayectoria circular y su velocidad angular es:

$$\dot{\vartheta} = \frac{v}{R_1}$$

Por geometría, se puede deducir que el radio de giro es $R_1 = L/\tan\gamma$, donde L es la longitud del vehículo o la distancia entre ejes. Por lo tanto, el modelo cinemático de este robot móvil se puede describir mediante las siguientes ecuaciones:

$$\dot{x} = v\cos\vartheta$$
$$\dot{y} = v\,sen\vartheta$$
$$\dot{\vartheta} = \frac{v}{L}\tan\gamma$$

donde el parámetro $\dot{\vartheta}$ se le denomina velocidad de giro o de guiñada. Además, puede deducirse que la velocidad angular de las ruedas izquierda y derecha del vehículo (que siguen curvas de diferente radio) giran a diferentes velocidades. De este modelo cinemático, se deducen otras características importantes de un vehículo con ruedas direccionales. Por ejemplo, cuando la velocidad es nula ($v = 0$), entonces $\dot{\vartheta} = 0$, es decir, no es posible cambiar la orientación del vehículo cuando no se está moviendo. Si el ángulo de dirección es $\pi/2$, la rueda delantera está ortogonal a la rueda trasera, por lo que el vehículo no puede avanzar y el modelo cinemático no está definido para ese valor ($\tan \pi/2 = \infty$).

El modelo cinemático de un vehículo direccional tipo *bicycle* se encuentra implementado en la RT (Figura 15). Se encuentra definido dentro del bloque Bicycle, y al que se puede acceder mediante el comando *roblocks*, dentro de la parte de *Dynamics*. El modelo también incluye un límite de velocidad máxima, y un limitador en el ángulo de dirección para tener un valor finito del parámetro γ.

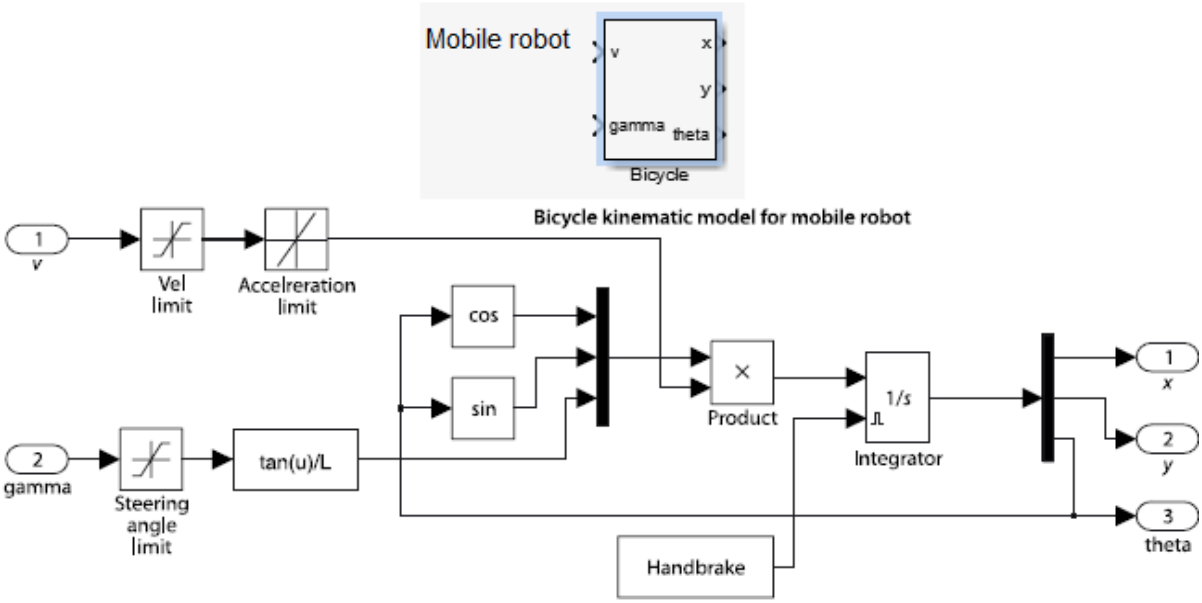


Figura 15. Modelado cinemático *Bicycle* defindio en la RT de Matlab

Dentro de la RT, se encuentra un ejemplo de una maniobra de cambio de dirección que emplea el modelo cinemático mostrado anteriormente. El ejemplo se puede acceder mediante el comando *sl_lanechange*, que ejecuta un modelo *simulink* en el que se pueden variar los valores del modelo y de las entradas del sistema. La entrada de dirección (*Steering angle*) se trata de un vector que genera un pulso positivo y luego negativo en el ángulo de dirección.

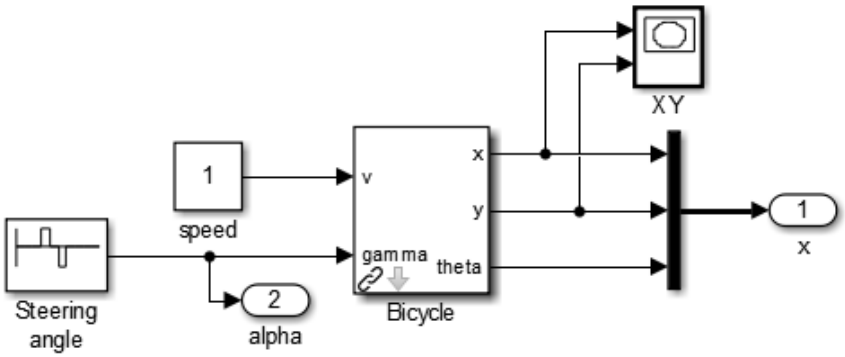


Figura 16. Ejemplo del cambio de dirección de un vehículo tipo *Bicycle*

Ejercicio práctico 10: Inserta el comando *sl_lanechange* en la línea de comandos de Matlab para abrir el archivo *Simulink*. Ejecuta dicho archivo y visualiza la entrada de dirección (*Steering angle*), así como el valor del ángulo ϑ (*theta*). Cambia los valores máximos/mínimos de la dicha entrada y visualiza los cambios en el visor XY. ¿Qué es lo que representa esta gráfica XY? Cambia los parámetros del bloque Bicycle y visualiza los cambios en la posición del vehículo.

Ejercicio práctico 11: Sobre el archivo *Simulink* introduce otras entradas en la dirección del vehículo y visualiza los cambios en la trayectoria. ¿Qué tipo de entrada y qué valor se debe introducir al vehículo para que la trayectoria XY sea una circunferencia en un tiempo de 10 seg?