MatLab y Robótica

Introducción a Matlab y la ToolBox de Robótica

Adaptación del guión realizado por Javier Alejandro Jorge

01/02/2012
Universidad Politécnica de Madrid
Miguel Hernando

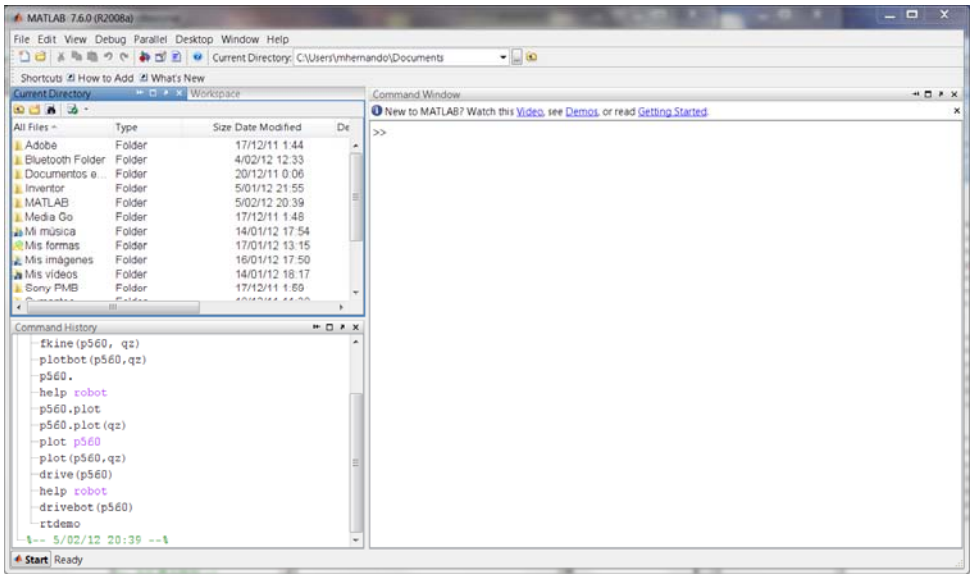
INTRODUCCIÓN

Una de las herramientas que actualmente se utilizan más en la ingeniería es MATLAB. Este programa, tiene entre otras características particulares su posibilidad de ampliación por medio de las denominadas ToolBox (Cajas de Herramientas). MATLAB es el nombre abreviado de "MATrix LABoratory" y es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares, tanto reales como complejos. Una de las capacidades más atractivas es la de realizar una amplia variedad de gráficos en dos y tres dimensiones.

Aunque en sus orígenes estaba centrado en el cálculo matemático y en concreto el matricial, poco a poco se ha ido extendiendo hasta abarcar prácticamente todos los ámbitos de la ingeniería. Algunas de estas extensiones pertenecen al propio programa, mientras otras -como es la ToolBox de robótica- son desarrollos de terceros que continúan su desarrollo y adaptación a las versiones sucesivas de Matlab, pero que requieren de una instalación aparte..

Para ciertas operaciones es muy rápido, cuando puede ejecutar sus funciones en código nativo con los tamaños más adecuados para aprovechar sus capacidades de vectorización. En otras aplicaciones resulta bastante más lento que el código equivalente desarrollado en C/C++ básicamente por su característica de lenguaje interpretado. Sin embargo, siempre es una magnífica herramienta de alto nivel para desarrollar aplicaciones técnicas, fácil de utilizar y que aumenta la productividad de los programadores respecto a otros entornos de desarrollo.

Al abrir Matlab aparecerá una ventana con distintos componentes, como la mostrada en la figura:



En el que destacan la ventana de pestañas en la que se muestra el directorio de trabajo actual, y el Workspace, el histórico de comandos, y la ventana de comandos de Matlab. Esta configuración es la que aparece por defecto, sin embargo a medida que se trabaja con el programa, podemos incluir o quitar algunos de estos componentes. En cualquier momento, si deseásemos recuperar esta configuración de subventanas inicial, se puede ejecutar el comando Desktop/Desktop LayOut/Default del menú, y nos reaparecerá esta configuración.

Básicamente nos centraremos en los comandos que podemos ejecutar en este programa, dado que sería excesivo pretender hacer en esta breve introducción una explicación de todas las capacidades básicas del mismo.

La ventana de comandos, refleja el hecho de encontrarse a la espera de un comando con una entradilla de línea dada por >>. A lo largo del guión cuando se indiquen instrucciones a Matlab, se pondrá con ese indicador, además de indicarse con una fuente distinta

1. MATLAB BÁSICO

MatLab es un lenguaje de programación interpretado en el que las variables son matrices y, por tanto, las operaciones básicas aritméticas y lógicas son operaciones matriciales. Esto hace que MatLab sea una herramienta muy adecuada para calculo matricial y, en concreto, para simulación de robots.

Puede obtener ayuda para cada función mediante el comando

```
>> help nombre_de_funcion
```

A continuación se muestran algunas de las operaciones básicas con MatLab:

VECTORES Y MATRICES

```
>> A=[1 2 3;4 3 2;3 2 1]
>> x=[2; 1; 3]
>> y=[1; 4; 6]
```

El primer aspecto a destacar es que en Matlab no es necesario decir de antemano -declarar- el tipo de una variable. Según se necesita se crea, y de hecho aparecerá reflejado en la ventana del WorkSpace. La primera línea cera una matriz llamada "A", con las filas {1,2,3}{4,3,2} y {3 2 1}. Se observa que para diferenciar los elementos de la fila se usan espacios, aunque tambien se admiten comas (","). Cualquier conjunto se delimita por los corchetes, y las filas quedan delimitadas por un punto y coma (";").

El resultado matemático de la operación anterior es:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 2 \\ 3 & 2 & 1 \end{pmatrix}, x = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, y = \begin{pmatrix} 1 \\ 4 \\ 6 \end{pmatrix}$$

Las operaciones que podemos realizar con matrices y vectores son diversas. Las más comunes son:

- Producto de una matriz por un vector

  ```
  >> y=A*x
  ```

- Trasposición de un vector o de una matriz

  ```
  >> z=x'
  >> B=B'
  ```

- Producto escalar de vectores

  ```
  >> z=x'*y
  ```

- Producto elemento a elemento (Hadamar)

  ```
  >> z=x.*y
  ```

- Determinante de una matriz

```
>> det(A)
```

- Inversa de una matriz

```
>> inv(A)
```

EL OPERADOR :

Pueden generarse secuencias de números para generar vectores o acceder a los elementos de las matrices sin necesidad de hacerlo individualmente(esto se mostrara a continuación)

```
>> D=1:.5:2
```

```
Variable = valor_inicial:incremento:valor_final
```

Genera una secuencia de números que comienza en uno incrementándose en 0,5 hasta llegar al máximo múltiplo del incremento menor o igual al valor final. En este caso [1 1.5 2]

En general el operador ":" indica una secuencia de números. Si sólo se pone una vez, entonces se supone que el incremento es unitario. O sea: D=1:5, significa la secuencia 1;2;3;4;5. Dependiendo del ámbito algunos de estos parámetros pueden ser omitidos, por ejemplo si solo se colocan dos números separados por ":" se los toma como el valor inicial y el final quedando como incremento por defecto la unidad.

Este operador es especialmente útil para obtener partes de una matriz. En general se extraen submatrices o elementos de una matriz mediante los paréntesis :

```
matriz(fila,columna)
```

Así por ejemplo:

```
>> A(3,3)
```

extrae el elemento de matriz (3,3). Sin embargo, es posible indicar rango de filas o de columnas gracias al operador ":"

```
>> A(2:3,1:2)
```

Extrae una submatriz que encierran las filas desde 2 hasta 3 y las columnas desde 1 hasta dos. Estos son los elementos A(2,1); A(2,2); A(3,1); A(3,2)

```
>> A(:,1)
```

extraerá la primera columna. Viene a significar "de todas las filas, el elemento 1"

FUNCIONES Y GRÁFICAS

Las funciones también se manejan vectorialmente. Por ejemplo, para generar y dibujar la función sin(t), podemos ejecutar:

```
>> t=0:0.01:2*pi;
>> y=sin(t);
>> plot(t,y)
```

La primera instrucción genera un vector con los valores de la variable independiente t desde 0 a 2π con incrementos de 0,01. El segundo comando genera el vector correspondiente con los valores de la función, y el tercero dibuja la grafica de y en función de t. Los ";" evitan que el resultado de cada instrucción sea mostrado por pantalla (solo útil en comandos que produzcan resultados numéricos).

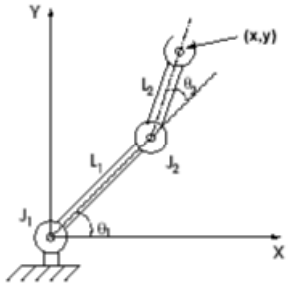
1.2. PROBLEMA CINEMÁTICO DIRECTO

El problema cinemático directo consiste en determinar la posición y orientación del final del extremo del robot a partir de las articulaciones y parámetros geométricos del robot.

Como ya se ha indicado, MatLab es una herramienta apropiada para el análisis y simulación

de problemas en robótica. Por ejemplo, para la trayectoria que recorre el EF de un manipulador RR (con enlaces de longitud L1 = L2 = 1) cuando las dos variables rotacionales θ 1 y θ 2 varían uniformemente de 0 a π/2, habría que ejecutar los siguientes comandos:

```
>> L1=1
>> L2=1
>> th1=0:(pi/2)/20:pi/2
>> th2=0:(pi/2)/20:pi/2
>> px=L1*cos(th1)+L2*cos(th1+th2)
>> py=L1*sin(th1)+L2*sin(th1+th2)
>> plot(px,py)
```



Las dos primeras instrucciones asignan a las longitudes L1 y L2 de los enlaces. Las dos siguientes generan dos vectores, th1 y th2, que contienen todos los valores de los ángulos θ1 y θ 2 entre 0 y π/2. Posteriormente, se ha hecho uso de las ecuaciones cinemáticas del manipulador para obtener las posiciones cartesianas (x;y) consecutivas. Las variables px y py son dos vectores que contienen los valores de las coordenadas X e Y, respectivamente.

Las instrucciones anteriores pueden ser incluidas en un archivo de texto con extensión .m, por ejemplo, practica1.m, y ejecutadas en la línea de comando de MatLab con el comando practica1

1.3. USO DE FUNCIONES

Cuando se va a hacer uso repetido de un conjunto de instrucciones, es conveniente escribir una función que las contenga. Así, podemos escribir una función que resuelva de forma genérica el problema cinemática directo de un robot de la siguiente manera:

```
function valor_a_retornar=nombre_de_funcion(parametros,…)
function p=pcd(L1,L2,th1,th2)
px=L1*cos(th1)+L2*cos(th1+th2);
py=L1*sin(th1)+L2*sin(th1+th2);
p=[px; py];
```

donde p es la variable que se retorna, y pcd es el nombre de la función seguido de los argumentos. La variable p contiene en la primera fila la secuencia de coordenadas

X, y en la segunda las Y. La funcion debe escribirse en un fichero de texto separado cuyo nombre debe ser el de la funcion seguido de la extensión .m (pcd.m).

Haciendo uso de la funcion anterior, la trayectoria anterior se trazara mediante los comandos:

```
>> p=pcd(L1,L2,th1,th2);
>> plot(p(1,:),p(2,:))
```

1.4. GRAFICA DE LA CONFIGURACIÓN ESPACIAL DEL ROBOT

En los ejemplos anteriores se ha dibujado exclusivamente la trayectoria descrita por el EF, pero no el robot. Dada una configuración espacial (por ejemplo, θ 1 = 300 θ 2 = 600), puede obtenerse un grafico del robot mediante:

```
>> th1=30*pi/180;
>> th2=60*pi/180;
>> p=pcd(L1,L2,th1,th2);
>> robotgraph(L1,th1,p);
```

donde robotgraph será la función que dibuja el robot en pantalla. Para desarrollar esta funcion, debe tenerse en cuenta que el robot se compone de dos segmentos lineales: el primero entre los puntos (0;0) y (cos θ1; sin θ1), y el segundo entre este último punto y la posición del EF p = (px; py).

Para trazar el robot, bastara con realizar un plot en el que las variables independiente y dependiente son dos vectores conteniendo las coordenadas X e Y, respectivamente, de los tres puntos mencionados. Una posible implementación de la funcion es la siguiente:

```
function robotgraph (L1,th1,p)
x=[0 L1*cos(th1) p(1)];
y=[0 L1*sin(th1) p(2)];
plot(x,y)
axis([-2 2 -2 2]);
```

El comando axis asegura que en sucesivos plots se conservan los rangos X e Y de la grafica.

2. MATLAB ROBOTICS TOOLBOX

En esta sección se continúan exponiendo conceptos y problemas de robótica y se incluye una explicación de como utilizar el toolbox para solucionarlos

INSTALACIÓN:

Un toolbox en MatLab no es más que una colección de funciones agrupadas en carpetas, por lo que para “instalar” el toolbox solo es necesario incorporar sus funciones al espacio de nombres de MatLab.

Existen dos formas de insertar nuevas funciones en MatLab:

- La menos elegante: que consiste simplemente en copiar todos los .m del directorio ‘robot' en su propio path de trabajo de MatLab.

- La mas elegante y profesional: que por su puesto es un poco mas “complicada” pues tiene la insoportable suma de 7 extremadamente tediosos pasos:

  1. copie el directorio \robot del archivo .zip al del directorio \MatLab\toolbox

  2. Seleccione File -> Preferences -> General.

  3. Click Update Toolbox Path Cache y click OK.

  4. Seleccione File -> Set Path -> Add Fólder

  5. seleccione la carpeta \MatLab\toolbox\robot click OK.

  6. click Save

Toda la documentación del toolbox se encuentra en robot.pdf dentro de la carpeta del toolbox

2.1 MATRICES DE TRANSFORMACIÓN HOMOGÉNEAS

Estas matrices son utilizadas para describir la localización de sólidos en un espacio n-dimensional, expresando las relaciones entre corderas cartesianas. La gran innovación que introdujeron estas matrices es que permiten representar en una única matriz la posición y la orientación del cuerpo.

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ f_{1x3} & R_{1x1} \end{bmatrix}$$

R- matriz de rotación

p- matriz de traslación

f- matiz de perspectiva

w- matriz de escaldo

Es importante notar que el producto de estas matrices en general no es conmutativo

Para crear una traslación pura se utiliza la función “transl”

TR = TRANSL(X, Y, Z)

```
>> transl(0.5, 0.0, 0.0)
```

Para crear una matriz de rotación se utiliza “rot**e**” donde **e** es el eje de rotación, y el ángulo como siempre en mathlab, se indica en radianes, pudiéndose utilizar la constante pi.

R = ROTY(theta)

```
>> roty(pi/2)
```

Para crear una matriz de transformación homogénea de rotación se antepone la t:

TR=TROTY(theta)

```
>> troty(theta)
```

Para crear una combinación de ellos simplemente se multiplican las matrices

```
>> t = transl(0.5, 0.0, 0.0) * roty(pi/2)
```

2.2 CREANDO EL PRIMER ROBOT

Para comenzar debemos tener en cuenta el algoritmo de DENAVIT-HARTENBERG pues sus parámetros serán de utilidad posteriormente.

ALGORITMO DE DENAVIT-HARTENBERG

Los parámetros de DENAVIT-HARTENBERG dependen únicamente de las características geométricas de cada eslabón y de la articulación que lo une con el anterior

Θi: (movimiento angular en rotación) es el ángulo que forman los ejes xi-1 y xi medido en un plano perpendicular al eje zi-1 utilizando la regla de la mano derecha (es variable en articulaciones giratorias)

di: (movimiento lineal en prismáticas)es la distancia a lo largo del eje zi-1, desde el origen del sistema de coordenadas i-1, hasta la intersección con el eje xi. (variable en articulaciones prismáticas)

ai :en el caso de articulaciones giratorias, es la distancia a lo lardo del eje xi que va desde la intersección con el eje zi-1 hasta el origen del sistema i pero en el caso de articulaciones prismáticas, es la distancia mas corta entre los ejes zi y zi-1

αi : es el Angulo de separación del eje zi-1 y el eje zi, medio en un plano perpendicular al eje xi utilizando la regla de la mano derecha

Es muy importante medir los ángulos utilizando la regla de la mano derecha

PUESTA EN PRÁCTICA

Basándose en los parámetros de la matriz crearemos los eslabones del robot mediante la función LINK. Esta función toma como parámetros los elementos de Denavit-Hartenberg de la siguiente manera:

```
LINK([alpha A theta D sigma])
```

El quinto argumento sigma es una bandera utilizada para indicar si la articulación es de revolución (sigma = 0) o si es prismática (sigma = 1)

```
>> L1=link([0 1 0 0 0])
>> L2=link([0 1 0 0 0])
```

Luego utilizando la clase ROBOT unimos todos los eslabones para dar lugar a nuestro primer robot.

```
>> r=robot({L1 L2})
```

2.3 CINEMÁTICA

La cinemática es el estudio del movimiento sin consideración alguna hacia las fuerzas que lo causan. Dentro de la cinemática uno estudia la posición, velocidad y aceleración, y todos los derivados de una orden más alta de las variables de la posición. La cinemática de manipulantes implica el estudio del geométrico y mide el tiempo de las características basadas del movimiento, y en detalle cómo los varios acoplamientos se mueven con respecto a uno otro y con tiempo.

PROBLEMA CINEMÁTICO DIRECTO

Como ya se mencionó, el problema cinemático directo consiste en determinar la posición y orientación final del extremo del robot a partir de las articulaciones y parámetros geométricos del robot.

Para ello se utiliza la función “fkine” que toma como parámetros un objeto robot y un vector de movimiento en el que se especifica el valor del parámetro variable de cada articulación y devuelve la matriz homogénea que describe la posición y orientación del extremo final.

```
TR = FKINE(ROBOT, Q)
```

Q puede no ser un vector, en ese caso tendrá que ser una matriz de nxm donde n es el número de ejes del manipulador y m es la cantidad de pasos de una secuencia. En este caso la función retornará una matriz tridimensional

```
>> puma560
>> fkine(p560, qz)
>> plotbot(p560, qz);
```

PROBLEMA CINEMÁTICO INVERSO

En el caso de la cinemática inversa el problema es justamente el opuesto, tratar de determinar los valores que deben asumir los parámetros variables de cada articulación para lograr una posición determinada por una matriz de transformación homogénea .

En el toolbox ya existe una función con dicho propósito, se llama “ikine” y toma como argumentos un objeto de tipo robot y una matriz de transformación que indique la posición final deseada.

```
Q = IKINE(ROBOT, T, Q, M)
```

Los demás parámetros son opcionales, Q es el punto inicial de la función de cálculo (recuérdese que las funciones de estimación son iterativas y en este caso requiere de una aproximación inicial) este valor por defecto es nulo. M es una matriz de mascara para el caso de manipuladores con mas de 6 grados de libertad. Esta mascara restringe los movimientos posibles para restringir las posibles soluciones; cada elemento de la matriz representa uno de los grados de libertad que pueden ser restringidos mediante un “0” en dicha matriz.

```
>> puma560
>> T=fkine(p560, qz)
>> ikine(p560, T)
```

ANEXO: FUNCIONES DE MATLAB ROBOT TOOLBOX 8

Homogeneous Transforms

`angvec2tr` angle/vector form to homogeneous transform
`eul2tr` Euler angle to homogeneous transform
`oa2tr` orientation and approach vector to homogeneous transform
`rpy2tr` Roll/pitch/yaw angles to homogeneous transform
`tr2angvec` homogeneous transform or rotation matrix to angle/vector form
`tr2eul` homogeneous transform or rotation matrix to Euler angles
`t2r` homogeneous transform to rotation submatrix
`tr2rpy` homogeneous transform or rotation matrix to roll/pitch/yaw angles
`trotx` homogeneous transform for rotation about X-axis

troty homogeneous transform for rotation about Y-axis
trotz homogeneous transform for rotation about Z-axis
transl set or extract the translational component of a homogeneous transform
trnorm normalize a homogeneous transform
trplot plot a homogeneous transformas a coordinate frame
Note that functions of the form tr2X will also accept a rotation matrixas the argument.

Rotation matrices

angvecr angle/vector form to rotation matrix
eul2r Euler angle to rotation matrix
oa2r orientation and approach vector to homogeneous transform
rotx rotation matrix for rotation about X-axis
roty rotation matrix for rotation about Y-axis
rotz rotation matrix for rotation about Z-axis
rpy2r Roll/pitch/yaw angles to rotation matrix
r2t rotation matrix to homogeneous transform

Trajectory Generation

ctraj Cartesian trajectory
jtraj joint space trajectory
trinterp interpolate homogeneous transforms

Quaternions

+ elementwise addition
elementwise
addition
/ divide quaternion by quaternion or scalar
* multiply quaternion by a quaternion or vector
inv invert a quaternion
norm norm of a quaternion
plot display a quaternion as a 3D rotation
q2tr quaternion to homogeneous transform
quaternion construct a quaternion
qinterp interpolate quaternions
unit unitize a quaternion

General serial link manipulators

link construct a robot link object
nofriction remove friction from a robot object
perturb randomly modify some dynamic parameters
robot construct a robot object
showlink show link/robot data in detail

Manipulator Models

Fanuc10L Fanuc 10L arm data (DH, kine)
MotomanHP6 Motoman HP6 arm data (DH, kine)
puma560 Puma 560 data (DH, kine, dyn)
puma560akb Puma 560 data (MDH, kine, dyn)
S4ABB2p8 ABB S4 2.8 arm data (DH, kine)
stanford Stanford arm data (DH, kine, dyn)
twolink simple 2-link example (DH, kine)

Kinematics

diff2tr differential motion vector to transform
fkine compute forward kinematics
ftrans transform force/moment
ikine compute inverse kinematics
ikine560 compute inverse kinematics for Puma 560 like arm
jacob0 compute Jacobian in base coordinate frame
jacobn compute Jacobian in end-effector coordinate frame
tr2diff homogeneous transform to differential motion vector
tr2jac homogeneous transform to Jacobian

Graphics

drivebot drive a graphical robot

`plot` plot/animate robot

Dynamics

`accel` compute forward dynamics
`cinertia` compute Cartesian manipulator inertia matrix
`coriolis` compute centripetal/coriolis torque
`fdyn` forward dynamics (motion given forces)
`friction` joint friction
`gravload` compute gravity loading
`inertia` compute manipulator inertia matrix
`itorque` compute inertia torque
`rne` inverse dynamics (forces given motion)

Other

`ishomog` test if argument is 4×4
`isrot` test if argument is 3×3
`isvec` test if argument is a 3-vector
`maniplty` compute manipulability
`rtdemo` toolbox demonstration
`unit` unitize a vector