MATLAB + ROBOTICA

Una introducción a al uso de MatLab y su aplicación en la robótica

Por Javier Alejandro Jorge

Introducción

Este documento creado para la asignatura robótica y animatrónica a cargo del ing. Hugo Pailos esta dirigido a estudiantes de ingeniería de los cuales solo se requieren conocimientos básicos de álgebra lineal y robótica.

Aquí se explicaran comandos básicos de MatLab, conceptos importantes de robótica, el uso del toolbox de robótica para MatLab y uso del puerto serie.

Acompañan a este documento una colección de archivos ".m" que son citados a lo largo del texto y que muestran implementaciones de lo explicado.

Durante el desarrollo del texto encontrara información sobre como utilizar MatLab para las funciones básicas y como se implementan los conceptos de robótica en esta maravillosa herramienta para ingeniería. Se comienza con una explicación del uso de matrices y funciones de MatLab, a continuación se introducen conceptos de robótica y se muestran aplicaciones de lo aprendido anteriormente. El mayor salto se da en la sección del toolbox donde se podrá aprovechar el potencial de MatLab mediante un amplio conjunto de instrucciones fáciles de utilizar.

En todo momento se acompaña al texto con código que puede ser ejecutado por el lector, este es destacado con una fuente diferente como se muestra a continuación.

```
>> Esto será código ejecutable en MatLab
```

Además, se incluye una práctica experimental con un robot real, comandado a travez del puerto serie y por último, en el apéndice podrá encontrar las principales funciones de MatLab para comunicaciones serie RS232 [1] y algunas de las funciones mas importantes del toolbox.

Se recomienda que ante cualquier duda sobre los comandos en MatLab se consulte a su sistema de ayuda.

[1] Se recomienda al lector interesado que investigue posibles dificultades para realizar lecturas, la escritura en el puerto no presenta mayores problemas

Índice

1. Introducción a MatLab

En esta sección se propone un breve introducción a MatLab introduciendo también conceptos básicos de robótica sin hacer uso del toolbox

1.1 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 algunos ejemplos de operaciones con MatLab

1.1.1. Asignación de valores a vectores y matrices:

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

Para las columnas se utiliza el espacio en blanco como separador y el punto y coma para las filas.

1.1.2. Productos matriciales y escalares:

```
>> y=A*x
>> z=x'*y
>> w=x.*y
```

El primer producto representa la operación de multiplicar el vector x con la matriz A. El segundo es el producto escalar de x por y, donde x' representa el vector x traspuesto. El tercer producto tiene como resultado un vector w en el que cada componente se obtiene multiplicando elemento a elemento las componentes correspondientes de x e y, es decir $w_i=x_i*y_i$

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 al valor final. En este caso 1;1.5;2
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.

Pueden extraerse submatrices o elementos de una matriz indicándose :
matriz(fila,columna)

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

extrae el elemento de matriz (3,3)

```
>> 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)
```

extrae la primera columna.

1.1.3. Funciones y gráficas

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

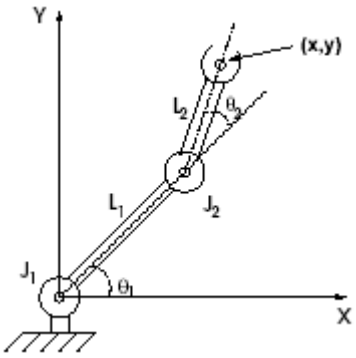
```
>> t=0:0.01:2*pi;
>> y=sin(t);
>> plot(t,y)
```

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 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 ultimo 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.

1.6. Problema Cinemático Inverso

Normalmente las trayectorias se expresan en coordenadas cartesianas que deben ser traducidas a variables de articulación. Esto es lo que se conoce como problema cinemático inverso.
En este caso el resultado de la operación es completamente dependiente de la configuración del robot, lo que dificulta la utilización de métodos generalizados
Existen dos formas de resolver el problema:
Mediante métodos iterativos
Mediante una expresión matemática

La primer aproximación, años atrás era prácticamente inconcebible debido a la gran cantidad de procesamiento requerida por dichas aplicaciones, hoy en día no es una alternativa despreciable.
La segunda aproximación es la mas elegante pero la mas complicada pero presenta ciertas ventajas entre ellas esta el tiempo de procesamiento casi nulo que en aplicaciones de tiempo real es muy preciado, la otra ventaja es que en algunos casos se pueden obtener varias soluciones para un mismo problema permitiendo este último método seleccionar la mas conveniente.

Un método interesante que se debe tener en cuenta es el método del desacoplo cinemático, que permite descomponer el problema y resolverlo para los primeros grados de libertad del robot (los dedicados al posicionamiento) y luego por separado resolver el problema para los grados de libertad dedicados a la orientación.

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

2.1 Instalación:

Un toolbox en MatLab no es mas 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
  7. Siéntase feliz hizo las cosas bien!

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

2.2 Matrices de Transformación homogéneas

Estas matrices son utilizadas para describir la localización de sólidos en un espacio ndimencional, 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.

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

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 rotación se utiliza "rot*e*" donde *e* es el eje de rotación
TR = ROTY(theta)

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

Para crear una combinación de ellos simplemente se multiplican las matrices

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

2.3 Creando el primer robot

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

2.3.1 Algoritmo de DENAVIT-HARTENBERG

1. Numerar los eslabones asignando 1 al primer eslabón móvil de la cadena
2. Numerar las articulaciones asignando 1 a la correspondiente al primer grado de libertad desde la base
3. Marcar el eje de cada articulación, en las rotativas es el eje de giro y en las prismáticas es el eje de desplazamiento
4. Situar el eje zi sobre el eje de la articulación hiel
5. Situar el s0 en cualquier lugar del eje z0 con los ejes x0 e y0 formando un sistema dextrógiro
6. Situar el sistema si solidario al eslabón i en la intersección del eje zi con la normal del zi-1, en el caso de ser paralelos, se lo situara en la articulación i+1
7. Situar xi en la normal comun a zi y zi-1
8. Situar yi de manera que forme un sistema dextrogiro con xi y zi
9. Situar el sistema sn en el extremo del robot de modo que zn coincida con la dirección de zn-1
10. Obtener θ
11. Obtener di
12. Obtener ai
13. Obtener αi

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 rotatorias) 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

2.3.2Puesta en práctica[2]

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)

[2] Vea el ejemplo adjunto en el archivo robot1.m

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

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

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

2.4 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.

2.4.1 Problema cinemático directo[3]

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 de la extremidad 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);
```

2.4.1 Problema cinemático inverso [4]

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)

---

[3] vea el archivo adjunto cinematicadirecta.m
[4] vea el archivo adjunto cinematicainversa.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 2x3 que actúa 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 6 grados de libertad que pueden ser restringidos mediante un “0” en dicha matriz.

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

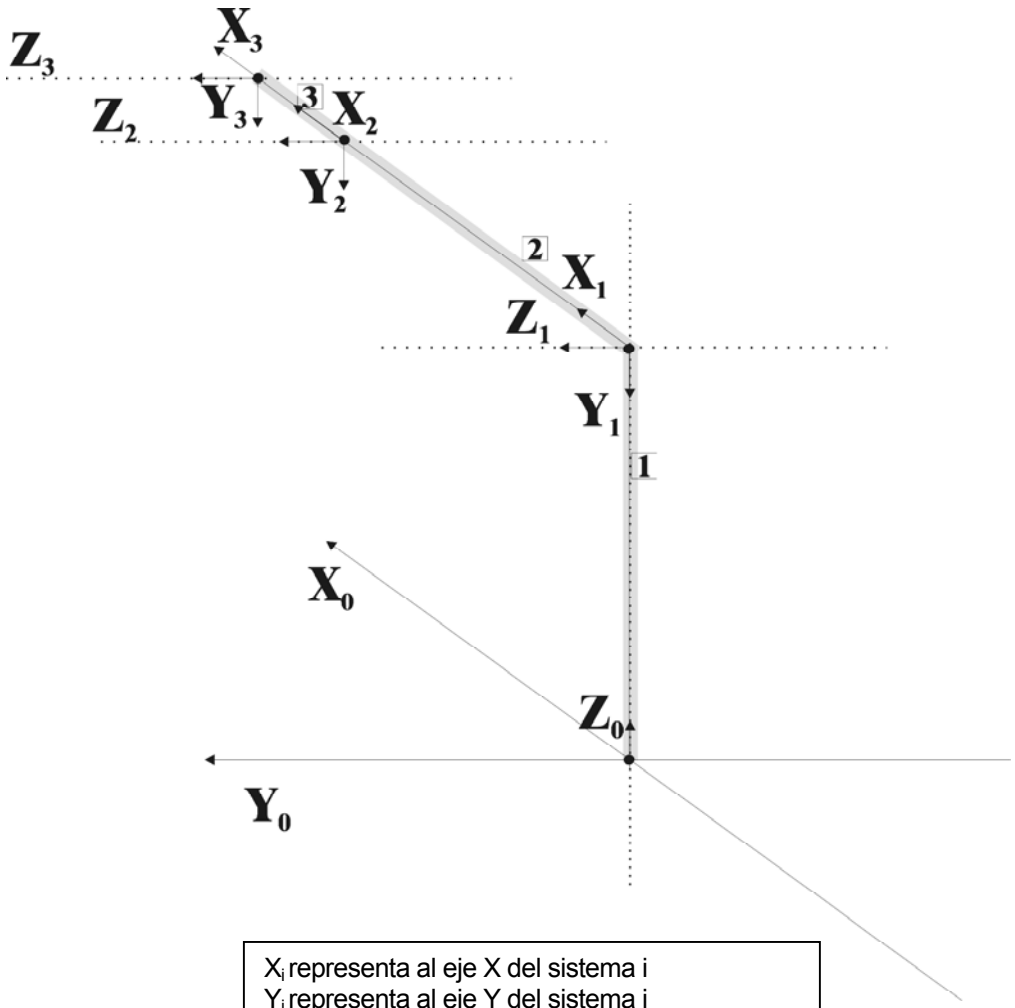
3 En el mundo real el robot Natalí

Natalí es simplemente un brazo robótico con tres segmentos, comandado por el puerto serial mediante la función mover[5] que toma como argumento un vector de desplazamiento de las articulaciones o una matriz de transformación homogénea que indique la posición final del robot.

3.1 Modelado del robot

Mediante el algoritmo de DH se procedió a modelar al robot obteniéndose el siguiente cuadro de resultados

A	a_i	θ	d_i
-90	0	0	10
0	15	0	0
0	3	0	0

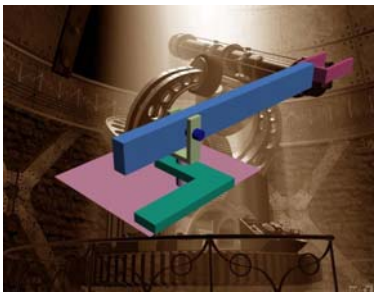


X_i representa al eje X del sistema i
Y_i representa al eje Y del sistema i
Z_i representa al eje Z del sistema i

..... representan ejes de giro
representa una sección física del robot

[5] refiérase al archivo mover.m

3.2 moviendo al robot [6]



una vez que tenemos el modelado se crean los links y el objeto robot el MatLab. A continuación se creo la función mover que nos entregara una grafica de la posición del robot y enviara las instrucciones al PIC del robot para que este las ejecute.
La interfaz es sencilla, se abre la comunicación con ":" luego se le pasa el número de motor que se moverá y finalmente un número comprendido entre 0 y 64H que es proporcional al ángulo de giro del motor.
La función mover puede admitir dos tipos de parámetros: un vector de movimientos individuales o una matriz de transformación homogénea que representara la posición y orientación del extremo del robot.
Entre los archivos adjuntos podrá encontrar ejemplos de trayectorias y movimientos del robot.

[6] vea el archivo mover.m, secuencia.m (contiene una serie de rutinas básicas) y noname.m (crea el modelo)

APENDICE

A1 Funciones para puerto serie de MatLab 7

Clear (serial)Remove serial port object from MATLAB workspace
Delete (serial)Remove serial port object from memory
Fclose (serial)Disconnect serial port object from the device
Fgetl (serial)Read from device and discard the terminator
Fgets (serial)Read from device and include the terminator
Fopen (serial)Connect serial port object to the device
Fprintf (serial)Write text to the device
Fread (serial)Read binary data from the device
Fscanf (serial)Read data from device and format as text
Fwrite (serial)Write binary data to the device
Get (serial)Return serial port object properties
Instrcallbackdisplay event information when an event occurs
Instrfindreturn serial port objects from memory to the MATLAB workspace
Isvaliddetermine if serial port objects are valid
Length (serial)Length of serial port object array
Load (serial)Load serial port objects and variables into MATLAB workspace
Readasyncread data asynchronously from the device
Recordrecord data and event information to a file
Save (serial)Save serial port objects and variables to MAT-file
Serialcreate a serial port object
Serialbreaksend break to device connected to the serial port
Set (serial)Configure or display serial port object properties
Size (serial)Size of serial port object array
Stopasyncstop asynchronous read and write operations

A2 Funciones Principales del TOOLBOX

ctraj

Purpose Compute a Cartesian trajectory between two points
Synopsis `TC = ctraj(T0, T1, m)`
`TC = ctraj(T0, T1, r)`
Description `ctraj` returns a Cartesian trajectory (straight line motion) `TC` from the point represented by homogeneous transform `T0` to `T1`. The number of points along the path is `m` or the length of
the given vector `r`. For the second case `r` is a vector of distances along the path (in the range
0 to 1) for each point. The first case has the points equally spaced, but different spacing may be specified to achieve acceptable acceleration profile. `TC` is a 4 4 *m* matrix.
Examples To create a Cartesian path with smooth acceleration we can use the `jtraj` function to create
the path vector `r` with continuous derivitives.

```
>> T0 = transl([0 0 0]); T1 = transl([-1 2 1]);
```

drivebot

Purpose Drive a graphical robot
Synopsis `drivebot(robot)`

Description Pops up a window with one slider for each joint. Operation of the sliders will drive the
graphical robot on the screen. Very useful for gaining an understanding of joint limits and robot workspace.
The joint coordinate state is kept with the graphical robot and can be obtained using the `plot` function. The initial value of joint coordinates is taken from the graphical robot.

fkine

Purpose Forward robot kinematics for serial link manipulator
Synopsis `T = fkine(robot, q)`
Description `fkine` computes forward kinematics for the joint coordinate `q` giving a homogeneous transform for
the location of the end-effector. `robot` is a robot object which contains a kinematic model in either standard or modified Denavit-Hartenberg notation. Note that the robot object can specify an arbitrary homogeneous transform for the base of the robot.
If `q` is a vector it is interpreted as the generalized joint coordinates, and `fkine` returns a homogeneous
transformation for the final link of the manipulator. If `q` is a matrix each row is interpreted as a joint state vector, and `T` is a 4 4 *m* matrix where `m` is the number of rows in `q`.
Cautionary Note that the dimensional units for the last column of the `T` matrix will be the same as the dimensional
units used in the `robot` object. The units can be whatever you choose (metres, inches, cubits or furlongs) but the choice will affect the numerical value of the elements in the last column of `T`. The Toolbox definitions `puma560` and `stanford` all use SI units with dimensions in metres.

ikine

Purpose Inverse manipulator kinematics
Synopsis `q = ikine(robot, T)`
`q = ikine(robot, T, q0)`
`q = ikine(robot, T, q0, M)`
Description `ikine` returns the joint coordinates for the manipulator described by the object `robot` whose endeffector
homogeneous transform is given by `T`. Note that the robot's base can be arbitrarily specified within the robot object.
If `T` is a homogeneous transform then a row vector of joint coordinates is returned. If `T` is a homogeneous
transform trajectory of size 4 4 *m* then `q` will be an *n m* matrix where each row is a vector of joint coordinates corresponding to the last subscript of `T`.
The initial estimate of `q` for each time step is taken as the solution from the previous time step. The estimate for the first step in `q0` if this is given else 0. Note that the inverse kinematic solution is generally not unique, and depends on the initial value `q0` (which defaults to 0).
For the case of a manipulator with fewer than 6 DOF it is not possible for the end-effector to satisfy the end-effector pose specified by an arbitrary homogeneous transform. This typically leads to nonconvergence
in `ikine`. A solution is to specify a 6-element weighting vector, `M`, whose elements are 0 for those Cartesian DOF that are unconstrained and 1 otherwise. The elements correspond to translation along the X-, Y- and Z-axes and rotation about the X-, Y- and Z-axes. For example, a 5-axis manipulator may be incapable of independantly controlling rotation about the end-effector's Z-axis. In this case `M = [1 1 1 1 1 0]` would enable a solution in which the end-effector adopted the pose `T` *except* for the end-effector rotation. The number of non-zero elements should equal the number of robot DOF.
Cautionary

jtraj

Purpose Compute a joint space trajectory between two joint coordinate poses
Synopsis `[q qd qdd] = jtraj(q0, q1, n)`
`[q qd qdd] = jtraj(q0, q1, n, qd0, qd1)`
`[q qd qdd] = jtraj(q0, q1, t)`

```
[q qd qdd] = jtraj(q0, q1, t, qd0, qd1)
```
Description `jtraj` returns a joint space trajectory `q` from joint coordinates `q0` to `q1`. The number of points is `n`
or the length of the given time vector `t`. A 7th order polynomial is used with default zero boundary conditions for velocity and acceleration.
Non-zero boundary velocities can be optionally specified as `qd0` and `qd1`.
The trajectory is a matrix, with one row per time step, and one column per joint. The function can optionally return a velocity and acceleration trajectories as `qd` and `qdd` respectively.

link

Purpose Link object
Synopsis `L = link`
```
L = link([alpha, a, theta, d])
L = link([alpha, a, theta, d, sigma])
L = link(dyn row)
A = link(q)
show(L)
```
Description The `link` function constructs a **link** object. The object contains kinematic and dynamic parameters
as well as actuator and transmission parameters. The first form returns a default object, while the second and third forms initialize the kinematic model based on Denavit and Hartenberg parameters.

By default the standard Denavit and Hartenberg conventions are assumed but a flag (`mdh`) can be set if modified Denavit and Hartenberg conventions are required. The dynamic model can be initialized using the fourth form of the constructor where `dyn row` is a 1 20 matrix which is one row of the legacy `dyn` matrix.
The second last form given above is not a constructor but a link method that returns the link transformation
matrix for the given joint coordinate. The argument is given to the link object using parenthesis.

The single argument is taken as the link variable *q* and substituted for θ or *D* for a revolute or prismatic link respectively.

The Denavit and Hartenberg parameters describe the spatial relationship between this link and the previous one. The meaning of the fields for each model are summarized in the following table.

Since Matlab does not support the concept of public class variables methods have been written to allow link object parameters to be referenced (r) or assigned (a) as given by the following table

Method	Operations	Returns
link.alpha	r+a	link twist angle
link.A	r+a	link length
link.theta	r+a	link rotation angle
link.D	r+a	link offset distance
link.sigma	r+a	joint type; 0 for revolute, non-zero for prismatic
link.RP	r	joint type; 'R' or 'P'
link.mdh	r+a	DH convention: 0 if standard, 1 if modified
link.I	r	3 3 symmetric inertia matrix
link.I	a	assigned from a 3 3 matrix or a 6-element vector interpretted as I_{xx} I_{yy} I_{zz} I_{xy} I_{yz} I_{xz}
link.m	r+a	link mass
link.r	r+a	3 1 link COG vector
link.G	r+a	gear ratio
link.Jm	r+a	motor inertia
link.B	r+a	viscous friction
link.Tc	r	Coulomb friction, 1 2 vector where τ τ
link.Tc	a	Coulomb friction; for symmetric friction this is a scalar, for asymmetric friction it is a 2-element vector for positive and negative velocity
link.dh	r+a	row of legacy DH matrix
link.dyn	r+a	row of legacy DYN matrix
link.qlim	r+a	joint coordinate limits, 2-vector

link.offset r+a joint coordinate offset (see discussion for robot object).
The default is for standard Denavit-Hartenberg conventions, zero friction, mass and inertias.
The display method gives a one-line summary of the link's kinematic parameters. The show method displays as many link parameters as have been initialized for that link.

rne

Purpose Compute inverse dynamics via recursive Newton-Euler formulation

Synopsis
```
tau = rne(robot, q, qd, qdd)
tau = rne(robot, [q qd qdd])
tau = rne(robot, q, qd, qdd, grav)
tau = rne(robot, [q qd qdd], grav)
tau = rne(robot, q, qd, qdd, grav, fext)
tau = rne(robot, [q qd qdd], grav, fext)
```

Description rne computes the equations of motion in an efficient manner, giving joint torque as a function of joint
position, velocity and acceleration.
If q, qd and qdd are row vectors then tau is a row vector of joint torques. If q, qd and qdd are matrices then tau is a matrix in which each row is the joint torque for the corresponding rows of q, qd and qdd.
Gravity direction is defined by the robot object but may be overridden by providing a gravity acceleration vector grav = [gx gy gz].
An external force/moment acting on the end of the manipulator may also be specified by a 6-element vector fext = [Fx Fy Fz Mx My Mz] in the end-effector coordinate frame.
The torque computed may contain contributions due to armature inertia and joint friction if these are specified in the parameter matrix dyn.
The MEX file version of this function is around 300 times faster than the M file.

robot

Purpose Robot object

Synopsis
```
r = robot
r = robot(rr)
r = robot(link ...)
r = robot(DH ...)
r = robot(DYN ...)
```

Description robot is the constructor for a robot object. The first form creates a default robot, and the second
form returns a new robot object with the same value as its argument. The third form creates a robot from a cell array of link objects which define the robot's kinematics and optionally dynamics. The fourth and fifth forms create a robot object from legacy DH and DYN format matrices.
The last three forms all accept optional trailing string arguments which are taken in order as being robot name, manufacturer and comment.
Since Matlab does not support the concept of public class variables methods have been written to allow robot object parameters to be referenced (r) or assigned (a) as given by the following table
method Operation Returns
robot.n r number of joints
robot.link r+a cell array of link objects
robot.name r+a robot name string
robot.manuf r+a robot manufacturer string
robot.comment r+a general comment string
robot.gravity r 3-element vector defining gravity direction
robot.mdh r DH convention: 0 if standard, 1 if modified.
Determined from the link objects.
robot.base r+a homogeneous transform defining base of robot
robot.tool r+a homogeneous transform defining tool of robot
robot.dh r legacy DH matrix
robot.dyn r legacy DYN matrix
robot.q r+a joint coordinates
robot.qlim r+a joint coordinate limits, *n* 2 matrix
robot.islimit r joint limit vector, for each joint set to -1, 0 or
1 depending if below low limit, OK, or greater
than upper limit

robot.offset r+a joint coordinate offsets
robot.plotopt r+a options for plot()
robot.lineopt r+a line style for robot graphical links
robot.shadowopt r+a line style for robot shadow links
Some of these operations at the robot level are actually wrappers around similarly named link object functions: offset, qlim, islimit.

The offset vector is added to the user specified joint angles before any kinematic or dynamic function is invoked (it is actually implemented within the link object). Similarly it is subtracted after an operation such as inverse kinematics. The need for a joint offset vector arises because of the constraints of the Denavit-Hartenberg (or modified Denavit-Hartenberg) notation. The pose of the robot with zero joint angles is frequently some rather unusual (or even unachievable) pose. The joint coordinate offset provides a means to make an arbitrary pose correspond to the zero joint angle case.
Default values for robot parameters are:
robot.name 'noname'
robot.manuf ''
robot.comment ''
robot.gravity 009 81 m s2
robot.offset ones(n,1)
robot.base eye(4,4)
robot.tool eye(4,4)
robot.lineopt 'Color', 'black', 'Linewidth', 4
robot.shadowopt 'Color', 'black', 'Linewidth', 1
The multiplication operator, *, is overloaded and the product of two robot objects is a robot which is the series connection of the multiplicands. Tool transforms of all but the last robot are ignored, base transform of all but the first robot are ignored.
The plot() function is also overloaded and is used to provide a robot animation.

robot/plot

Purpose Graphical robot animation
Synopsis plot(robot, q)
plot(robot, q, arguments...)
Description plot is overloaded for **robot** objects and displays a graphical representation of the robot given the
kinematic information in robot. The robot is represented by a simple stick figure polyline where line segments join the origins of the link coordinate frames. If q is a matrix representing a joint-space trajectory then an animation of the robot motion is shown.
OPTIONS
Options are specified by a variable length argument list comprising strings and numeric values. The allowed values are:
workspace w set the 3D plot bounds or workspace to the matrix [xmin xmax ymin ymax zmin zmax]
perspective show a perspective view
ortho show an orthogonal view
base, nobase control display of base, a line from the floor upto joint 0
wrist, nowrist control display of wrist axes
name, noname control display of robot name near joint 0
shadow, noshadow control display of a 'shadow' on the floor
joints, nojoints control display of joints, these are cylinders for revolute joints and boxes for prismatic joints
xyz wrist axis labels are X, Y, Z
noa wrist axis labels are N, O, A
mag scale annotation scale factor
erase, noerase control erasure of robot after each change
loop, noloop control whether animation is repeated endlessly
The options come from 3 sources and are processed in the order:
1. Cell array of options returned by the function PLOTBOTOPT if found on the user's current path.
2. Cell array of options returned by the .plotopt method of the robot object. These are set by the .plotopt method.

3. List of arguments in the command line.

rtdemo

Purpose Robot Toolbox demonstration
Synopsis `rtdemo`
Description This script provides demonstrations for most functions within the Robotics Toolbox.