MATLAB Robotics Toolbox
仿真实验手册

1. 理论知识

理论知识请参考：

- 机器人学导论--分析、控制及应用（第 2 版）美）Saeed B.Niku 著 孙富春等译
- 课堂 PPT

2. Matlab Robotics Toolbox 安装

2.1 MATLAB 的安装

- 建议使用 R2017 以上的版本，MATLAB 安装软件获取及安装方法，请网上查找。

2.2 MATLAB 中 Robotics Toolbox（机器人工具箱）的安装

1、 下载

在官网 http://petercorke.com/wordpress/toolboxes/robotics-toolbox 下载最新的安装包，我这里下载的是 10.3.2 版本的。

matlab robotics toolbox 目前提供了两种安装方式。有两种安装包，分别是.mltbx 格式和.zip 格式。

其中.mltbx 格式安装：可以直接从 matlab 文件浏览器中双击就可以打开（或者像打开 m 文件一样打开）。如下图，点击安装就可以自动安装了。

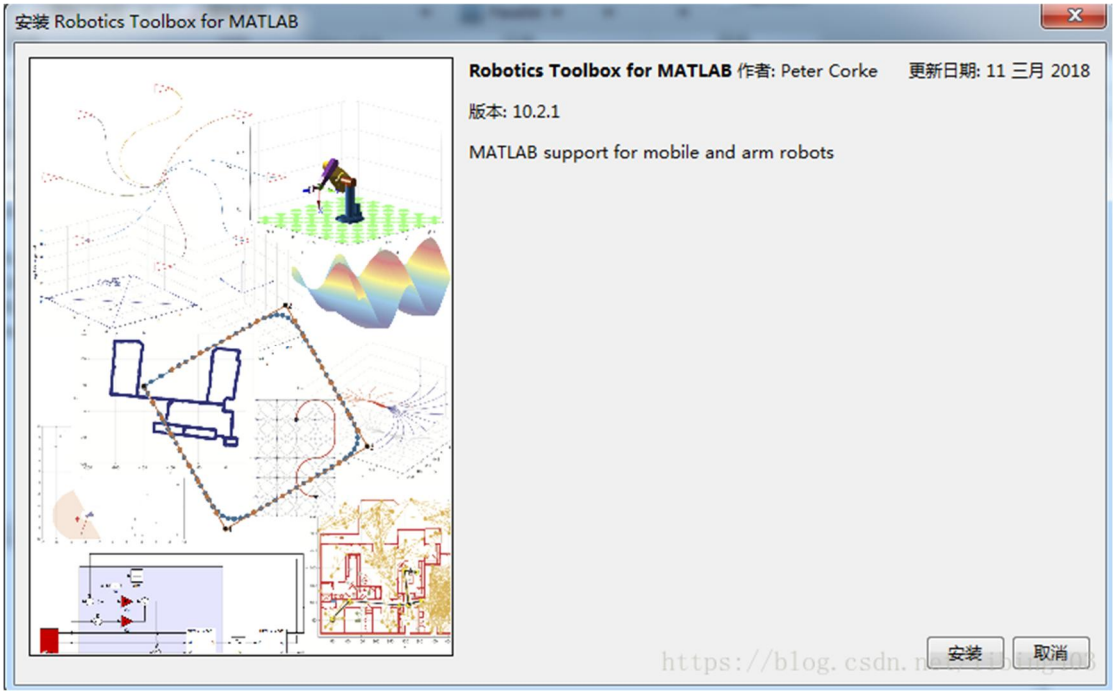


图 2.1　MATLAB Robotics Toolbox mltbx 格式

.zip 格式安装过程稍微复杂些，下面就是这种方式的安装过程。

2、　zip 格式安装

（1）解压

将解压出来的文件夹 rvctools 放在 MATLAB 安装目录下的 toolbox 文件夹下。不放在这里，放在其他地方也是可以的，只不过这里是 matlab 放工具箱的地方，所以建议放这里

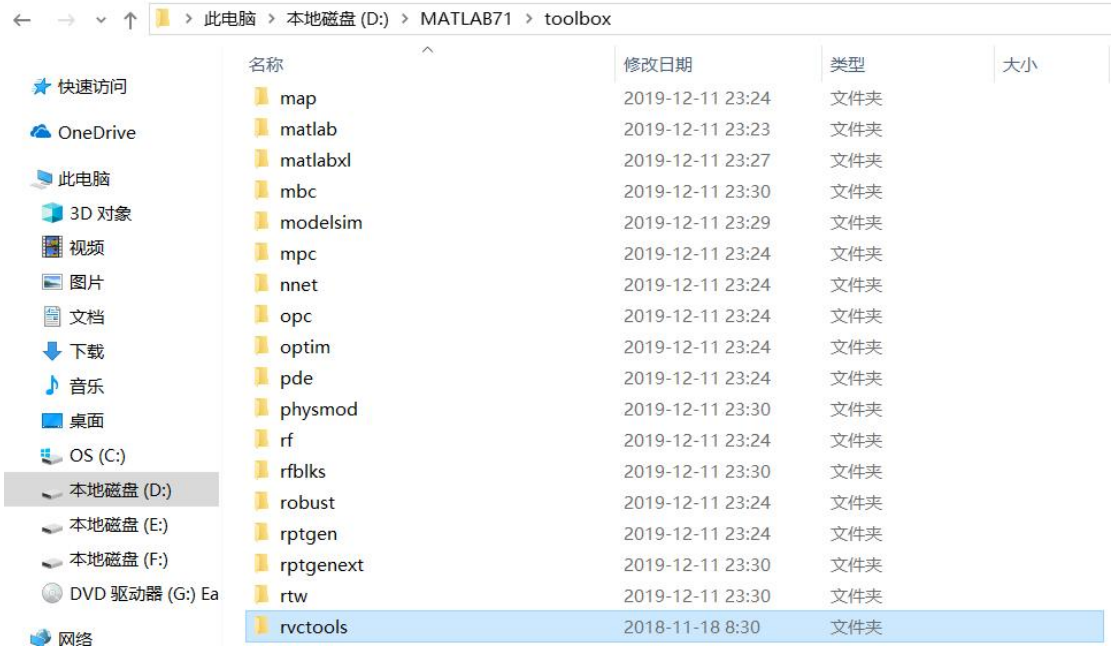


图 2.2　MATLAB Robotics Toolbox zip 格式安装 1

（2）打开 MATLAB，主页-设置路径

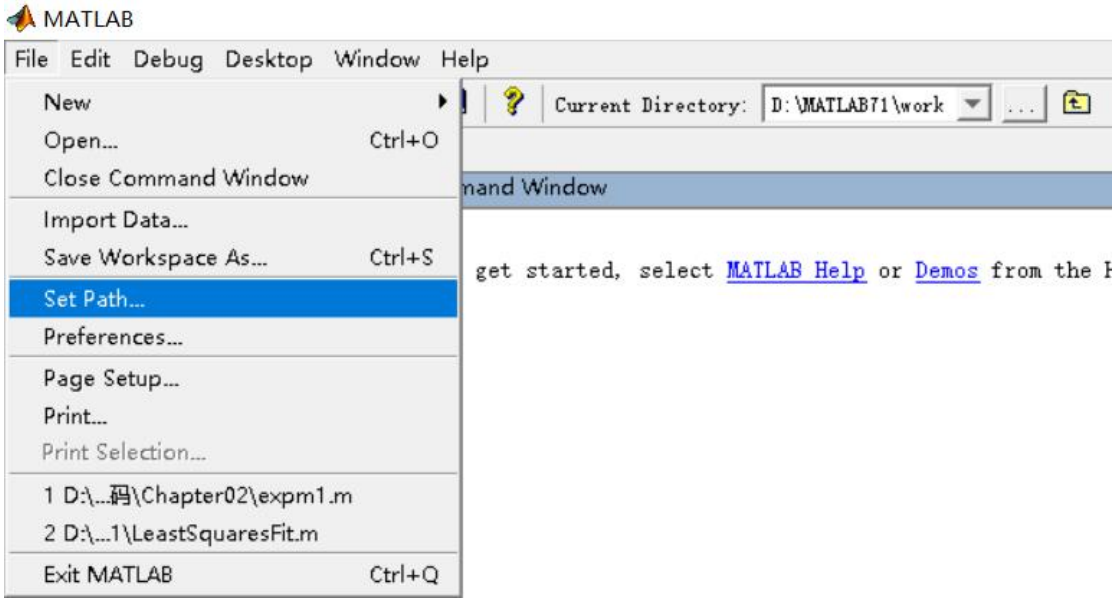


图 2.3　MATLAB Robotics Toolbox zip 格式安装 2

出现图 2.4 界面。

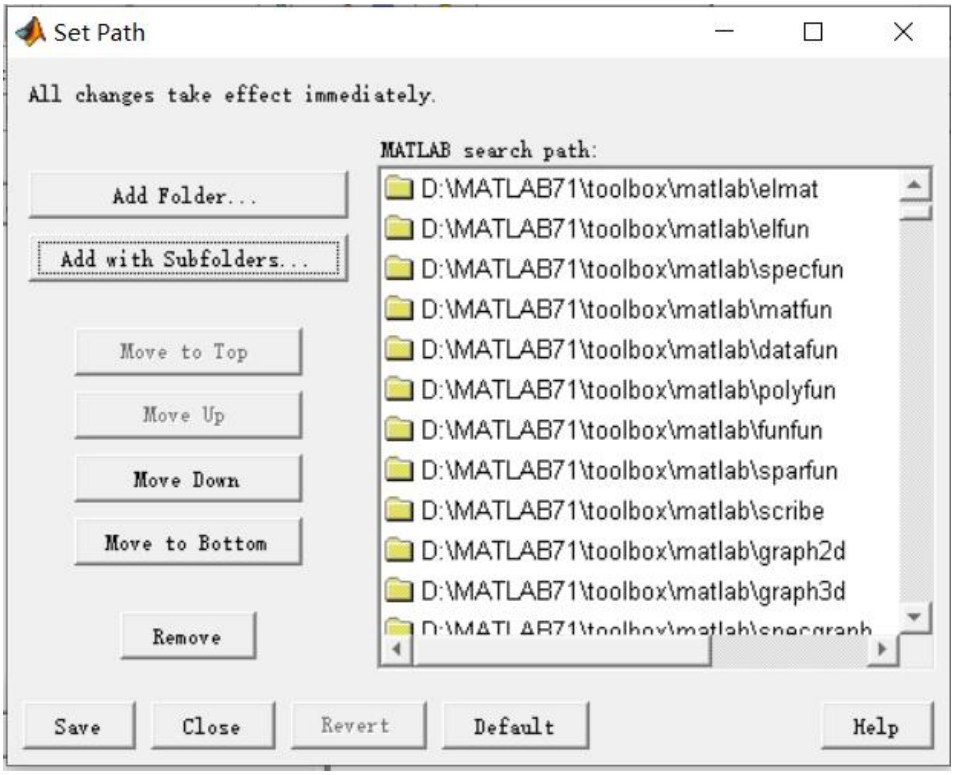


图 2.4　MATLAB Robotics Toolbox zip 格式安装 3

点击“Add with Subfolder”添加并包含子文件夹。出现图 2.5 所示界面。

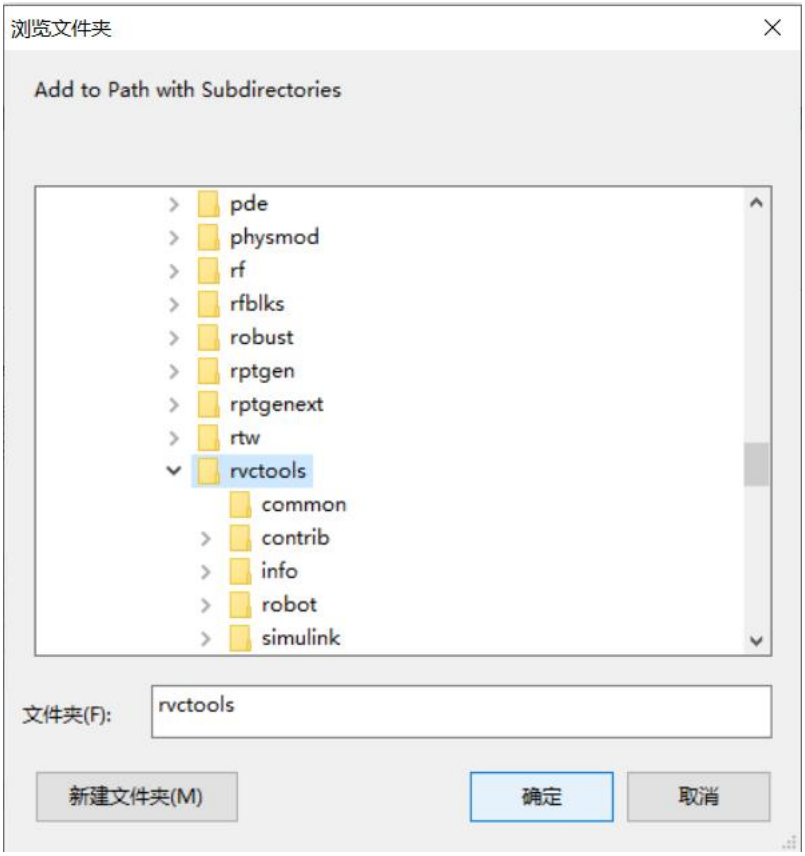


图 2.5　MATLAB Robotics Toolbox zip 格式安装 4

选中放入 toolbox 中的 rvctools，确定添加。然后出现图 2.6 所示界面：

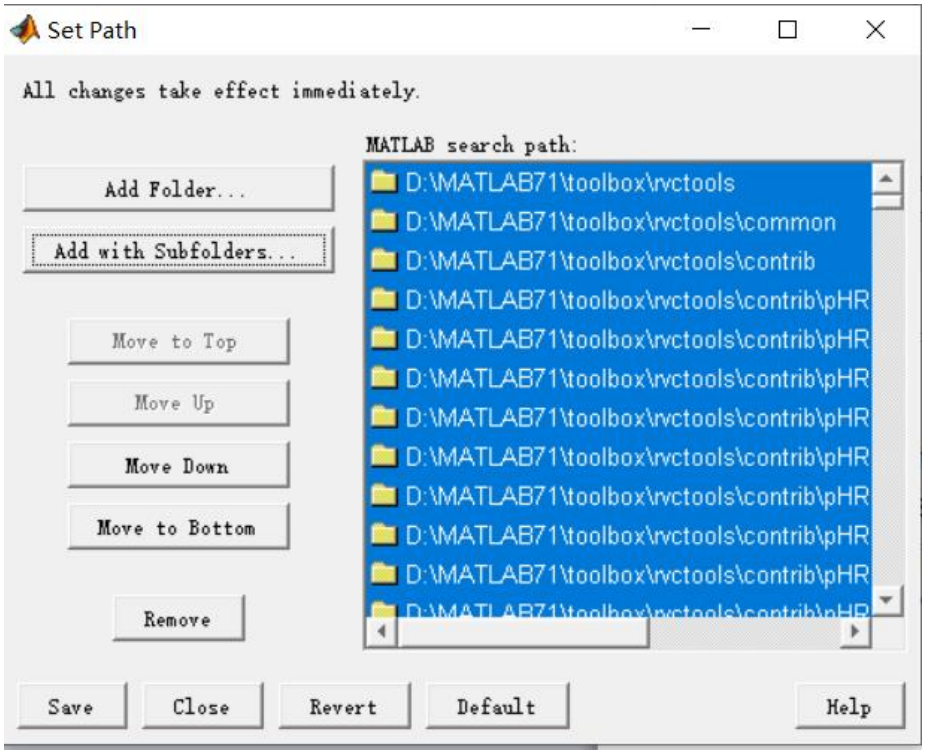


图 2.6　MATLAB Robotics Toolbox zip 格式安装 5

保存（save）并关闭（close）。

（3）安装

然后路径定位到刚才的文件夹（如图 2.7 所示），命令行输入 startup_rvc，回车，安装完毕。

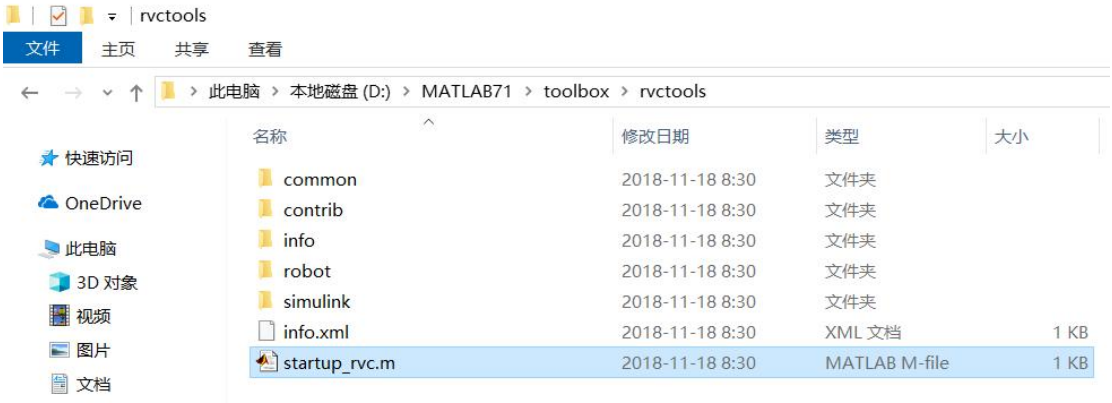


图 2.7　MATLAB Robotics Toolbox zip 格式安装 6

（4）检验

在 matlab 中运行 ver 命令，看看工具箱列表中有没有一个叫 robotics toolbox 的工具箱，有的话，代表安装成功。如图 2.8 所示。

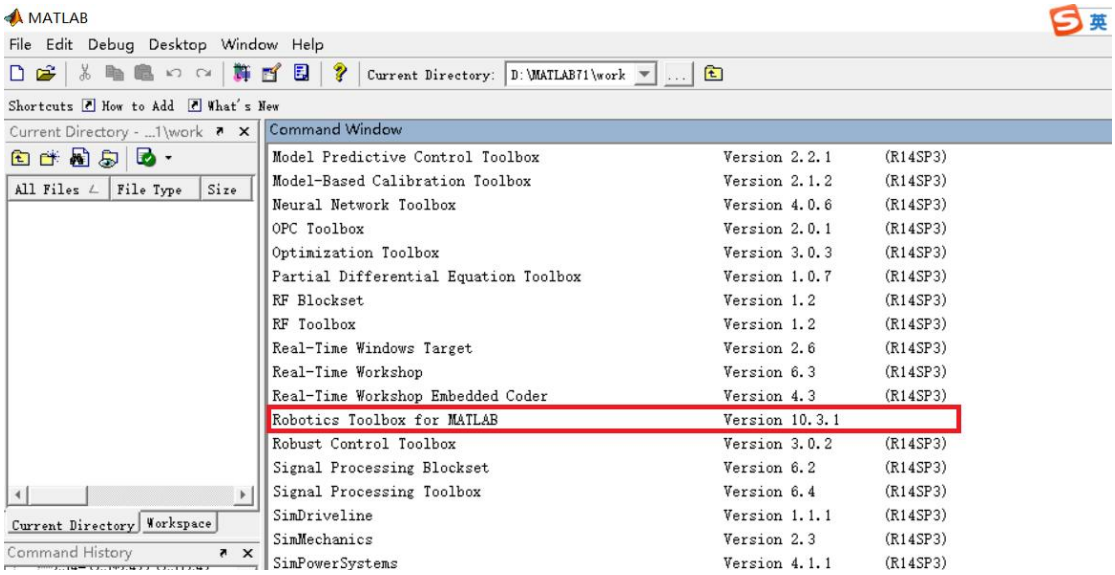


图 2.8　MATLAB Robotics Toolbox 安装成功验证

（5）示例运行

打开例程的方法为，在 MATLAB 命令输入界面，输入如下命令：

>>rtbdemo

将出现如下图 2.9 所示界面，点击这些按钮，可以看到相关示例。

图 2.9　MATLAB Robotics Toolbox 示例程序

2.3 本章实训内容

（1）成功安装 MATLAB2017 以上版本及 Robotics Toolbox 10.3.2 工具箱。

（2）熟悉 MATLAB 的使用方法。。

3. MATLAB Robotics Toolbox 简单应用--坐标系变换

3.1 旋转变换

在机器人工具箱中，可以用 rotx(θ)、roty(θ)、rotz(θ)计算旋转 θ 的旋转矩阵，默认单位为弧度。可以用 trplot()和 tranimate()实现坐标旋转可视化。

1、　获取旋转矩阵

R=**rotx**(θ)是表示围绕 X 轴旋转角度为θ得到的旋转矩阵，返回一个 3x3 的矩阵。

R=**roty**(θ)是表示围绕 Y 轴旋转角度为θ得到的旋转矩阵，返回一个 3x3 的矩阵。

R=**rotz**(θ)是表示围绕 Z 轴旋转角度为θ得到的旋转矩阵，返回一个 3x3 的矩阵。

T=**trotx**(θ) 表示绕 X 轴旋转θ得到的齐次变换矩阵（4x4);

T=**troty**(θ)表示绕 Y 轴旋转θ得到的齐次变换矩阵（4x4);

T=**trotz**(θ) 表示绕 Z 轴旋转θ得到的齐次变换矩阵（4x4);

【例 3.1】

```
>> R=rotx(30*pi/180)
   R =
    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660
```

2、 绘制坐标系

函数: trplot()

格式：有两种，分别是:

1) trplot(R)绘制由旋转矩阵 R 得到的坐标系，该坐标系由正交旋转矩阵围绕原点旋转得到，R 为 3x3 的矩阵。

2) trplot(T)绘制由齐次矩阵 T 表示的三维坐标系，T 为 4x4 的矩阵。

另外函数 trplot()还包含了以下主要可选参数

参数	意义	参数	意义
‘noaxes’	在绘图上不显示坐标轴	‘axis’, A	将图形显示的轴尺寸设置为A，其中 A=[xmin xmax ymin ymax zmin zmax]
‘color’,C	设置轴的颜色，C代表MATLAB图形内置的颜色类型	‘frame’,F	将绘制出来的坐标系命名为F，并且X,Y,Z轴的下标含有F
‘text_opts’,opt	调整显示文本的字体等属性；例如{‘FontSize’,10,’FontWeight’,’bold’}	‘view’,V	把绘图视图参数设置为V=[az,el]角度，或者对于坐标系的原点查看‘atuo’
‘length’,s	坐标轴的长度（默认值1）	‘arrow’	设置坐标轴的末端为箭头，而不是线段
‘with’,w	箭头宽度（默认值1）	’thick‘,t	线条粗细，默认0.5
’3d‘	在三维空间中使用浮雕图形绘制	’anaglyph‘,A	将“3d”的浮雕颜色指定为左右两个字符（默认为rc）：选自红，绿，蓝，青，品红
’dispar‘,D	3d显示差异（默认0.1）	’text‘	启用在框架上显示X,Y,Z标签
’labels‘,L	使用字符串L的第1个，第2个，第3个字符标记X,Y,Z轴	’rbg‘	以红色，绿色，蓝色分别显示X,Y,Z轴

3、 动画展示

函数为：tranimate()

格式：有三种，分别是：

1) tranimate(x1,x2,options)展示坐标系从姿态 x1 变换到姿态 x2 的动画效果。其中姿态 x1 和 x2 有三种表示方法：一个 4x4 的齐次矩阵，或一个 3x3 的旋转矩阵，或一个四元素。
2) tranimate(x,options)展示坐标系由上一个姿态变换到当前姿态 x 的动画效果。其中姿态 x 有三种表示方法：一个 4x4 的齐次矩阵，或一个 3x3 的旋转矩阵，或一个四元素。
3) tranimate(xseq,options)展示了移动一段轨迹的动画效果。xseq 可以是一组 4x4xN 的齐次矩阵，或一组 3x3xN 的旋转矩阵，或是一组四元素向量（Nx1）.

options 可以是以下可选参数：

参数	意义	参数	意义
‘fps’ ,fps	每秒显示的帧数（默认为10）	’nstep’ ,n	沿路径的步数（默认50）
‘axis’ ,A	设置三个轴的边界，A=[xmin xmax ymin ymax zmin zmax]	‘movie’ ,M	将帧保存为文件夹M中的文件，M为文件的路径

【例 3.2】

```
R=rotx(0);

R2=rotx(90);

tranimate(R,R2);
```

结果如下图所示（静态图无法展示动态过程，MATLAB 执行时可以看到动态过程）：

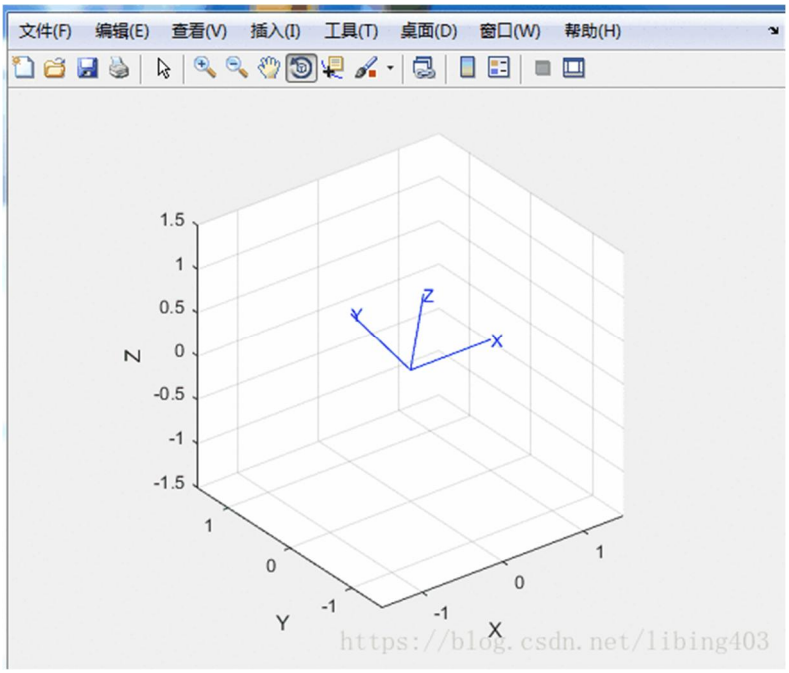


图 3.1　动画展示示例

3.2 平移变换

1、　创建平移变换矩阵

函数: transl ()

格式: 有两种，分别是:

1）T=transl(x,y,z)

表示能够获取一个分别沿着 x,y,z 轴平移一段距离得到的 4x4 齐次变换矩阵。

2）T=transl(p)

表示由经过矩阵（或向量）p=[x,y,z]的平移得到的齐次变换矩阵。如果 p 为（Mx3）的矩阵，则 T 为一组齐次变换矩阵（4x4xM），其中 T（：，：，i）对应于 p 的第 i 行。

【例 3.3】

```
>> T=transl(1,2,3)
T =
     1     0     0     1
     0     1     0     2
     0     0     1     3
     0     0     0     1
>>p=[2 3 4];
>>T=transl(p)
T =
     1     0     0     2
     0     1     0     3
     0     0     1     4
     0     0     0     1
```

2、 使用 transl()获取一个矩阵中的平移分量。

格式： 2 种

1) [x ,y, z]=transl(T)：x,y,z 是齐次变换矩阵 T 中的三个分量，是一个 1xM 的向量。
2) p=transl(T)：p 是齐次变换矩阵 T 中平移部分，是一个 3xM 的矩阵。

【例 3.4】

```
>>T=transl(3,4,5)
   T =
        1     0     0     3
        0     1     0     4
        0     0     1     5
        0     0     0     1
>> p=transl(T)
   p =
        3
        4
        5
>> [x,y,z]=transl(T)
```

```
x =
     3
y =
     4
z =
     5
```

3.3 矩阵分解

（1）R=t2r(T)

用于获取齐次变换矩阵 T 中正交旋转矩阵分量。如果 T 是一个 4x4 的旋转矩阵，则 R 是一个 3x3 的矩阵。

【例 3.5】

```
>> T=trotx(pi/6)*transl(3,4,5)
  T =
       1.0000            0            0       3.0000
            0       0.8660      -0.5000       0.9641
            0       0.5000       0.8660       6.3301
            0            0            0       1.0000
>> R=t2r(T)
  R =
       1.0000            0            0
            0       0.8660      -0.5000
            0       0.5000       0.8660
```

（2）T=r2t(R)

可将旋转矩阵转换为齐次矩阵，获取一个正交旋转矩阵 R 等价的具有零平移分量的齐次变换矩阵。如果 R 是一个 3x3 的矩阵，则 T 是一个 4x4 的矩阵；如果 R 是一个 2x2 的矩阵，则 T 是一个 3x3 的矩阵。

【例 3.6】

```
>> R=rotz(pi/3)
R =
       0.5000      -0.8660            0
       0.8660       0.5000            0
            0            0       1.0000
>> T=r2t(R)
T =
       0.5000      -0.8660            0            0
```

0.8660	0.5000	0	0
0	0	1.0000	0
0	0	0	1.0000

【例 3.7】

坐标系{A}与坐标系{B}重合，首先{B}相对于{A}的 y_A 轴旋转 60 度，再沿{A}的 x_A 轴移动 4 个单位，最后沿着 A 的 z_A 轴移动 3 个单位。点 p1 在坐标系 B 中描述为 ${}^{B}p1=[2,4,3]^{T}$，求它在坐标系 A 中的描述。

可以用 toolbox 仿真如下

```
clc
clear
close('all')

%坐标系正变换，坐标点逆变换
T1=troty(60);
T2=transl(4,0,3);
T=T2*T1;
p1=[2;4;3;1];
Ap1=T*p1;

%坐标系逆变换，坐标点正变换
Tr=inv(T);
Bp1=Tr*p1;

%%绘图
T0=transl(0,0,0);
trplot(T0,'frame','A','color','b');
axis([-5 5 -5 5 -5 5]);
hold on,
tranimate(T0,T,'frame','B','color','r');
```

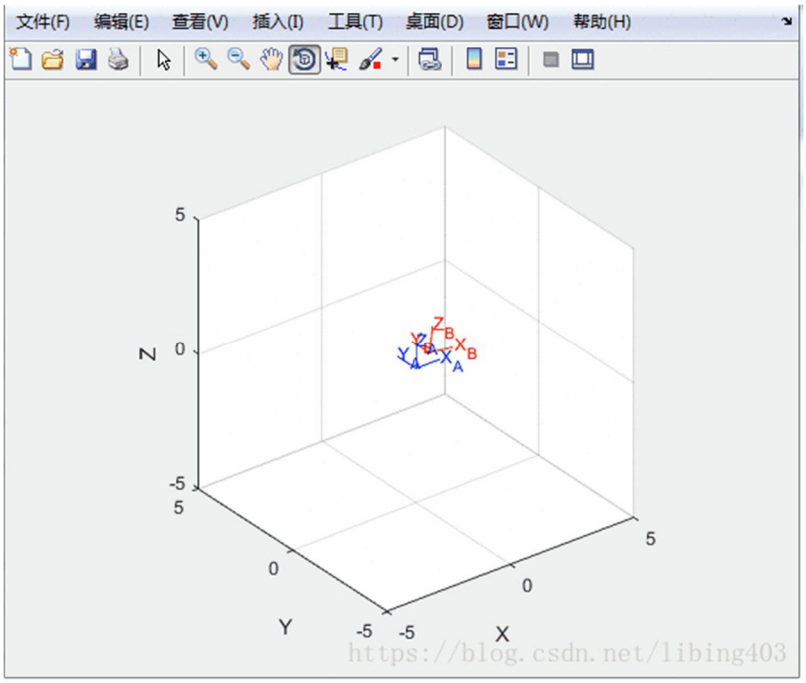


图 3.2　例 3.7 图

3.4 本章实训内容

（1）练习前面的 7 个例题，熟悉相关命令

（2）编写 MATLAB 程序，对教材中例 2.6（P34），例 2.7（P36），例 2,9（P38），例 2.11（P40）的内容进行实验验证，要求在程序运行结果图中用不同颜色展示出变换前的坐标系和变换后的坐标系，以及相应的点位。

（3）写出实验报告，包含：实验内容，所编写的程序，以及程序运行结果的 MATLAB 截图，必要时对图中内容做个简单解释。

4. MATLAB Robotics Toolbox 机器人建模

4.1 D-H 机器人建模方法分析

1、 标准 D-H 建模

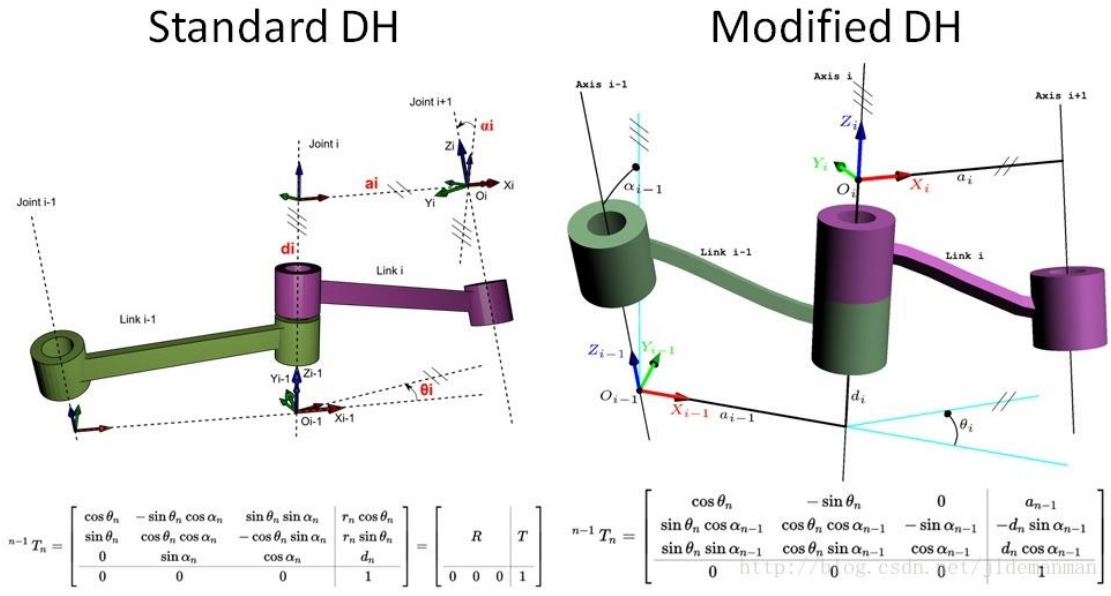
三个两两相互垂直的 XYZ 轴构成欧几里得空间，存在六个自由度：沿 XYZ 平移的三个自由度，绕 XYZ 旋转的三个自由度。在欧几里得空间中任意线性变换都可以通过这六个自由度完成。

Denavit-Hartenberg 提出的 D-H 参数模型能满足机器人学中的最小线性表示约定，用 4 个参数就能描述坐标变换：绕 X 轴平移距离 a；绕 X 轴旋转角度 alpha；绕 Z 轴平移距离 d；绕 Z 轴旋转角度 theta。具体建模步骤见教材 2.12 节，这里不再重复。

2、 标准 D-H 模型和改进 D-H 模型的区别

D-H 建模除了标准版本还有一个改进版本，后者不一定比前者先进多少，只不过二者各自有各自的应用场合。二者的区别如下表及图 4.1 所示：

STD_DH	MOD_DH
1.连杆选用的**固连坐标系**不同	
以连杆的后一个**关节坐标系**为其**固连坐标系**	以连杆的前一个**关节坐标系**为其**固连坐标系**
2.连杆坐标系的X轴方向确定方式不同	
以当前Z轴和“前一个”坐标系的Z轴叉乘确定X轴（以加入叉乘顺序）	以“后一个”坐标的Z轴与当前Z轴叉乘确定X轴（以加入叉乘顺序）
3.连杆坐标系之间的变换规则不一样	
相邻关节坐标系之间的参数变化顺序为：**θ、d、a、α**	相邻关节坐标系之间的参数变化顺序为：**α、a、θ、d**
$^{n-1}T_n = \mathrm{Trans}_{z_{n-1}}(d_n)\cdot\mathrm{Rot}_{z_{n-1}}(\theta_n)\cdot\mathrm{Trans}_{x_n}(r_n)\cdot\mathrm{Rot}_{x_n}(\alpha_n)$	$^{n-1}T_n = \mathrm{Rot}_{x_{n-1}}(\alpha_{n-1})\cdot\mathrm{Trans}_{x_{n-1}}(a_{n-1})\cdot\mathrm{Rot}_{z_n}(\theta_n)\cdot\mathrm{Trans}_{z_n}(d_n)$



$$^{n-1}T_n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

$$^{n-1}T_n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\ \sin\theta_n\cos\alpha_{n-1} & \cos\theta_n\cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n\sin\alpha_{n-1} \\ \sin\theta_n\sin\alpha_{n-1} & \cos\theta_n\sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n\cos\alpha_{n-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

图 4.1　标准 D-H 和改进 D-H 建模方法区别

3、 MATLAB 仿真对比

（1）标准 DH

下面为采用标准 D-H 建模的程序段：

```
%标准 D-H
%Link(D-H, option):
%DH = [THETA    D    A    ALPHA    SIGMA]
L1 = Link([0    0.2    1    pi/4            0],'standard');
L2 = Link([0    0.2    1    pi/4            0],'standard');
L3 = Link([0    0      0.5    pi/4          0],'standard');
robot = SerialLink([L1 L2 L3]); %建立连杆机器人
robot.plot([0 0 0]) %显示并赋三个关节变量 theta 值都为 0//此处才可以初始 theta
```

所建机器人如下图所示：

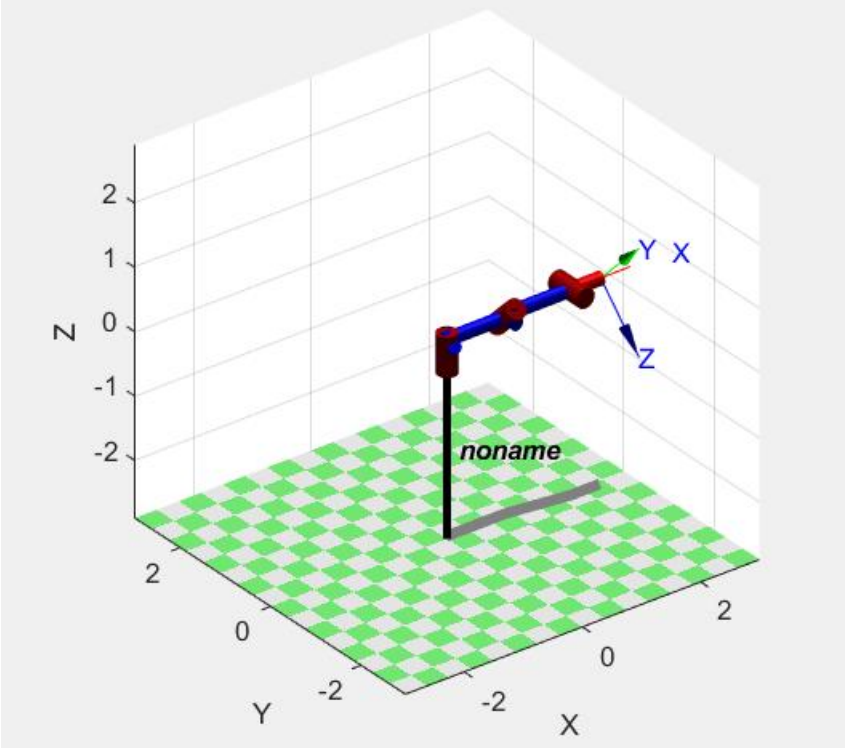


图 4.2　标准 D-H 建模图 1

坐标系分析如下图所示：

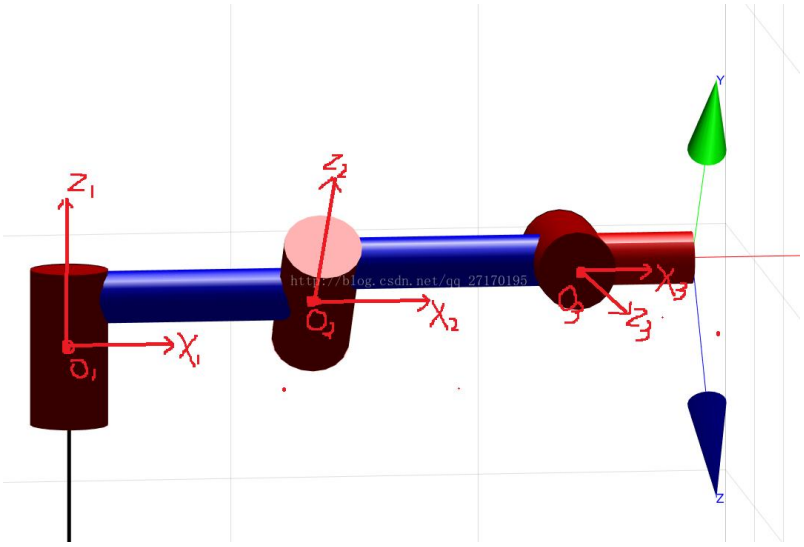


图 4.3　标准 D-H 建模图 2

标准 DH 四个参数含义：

- theta：绕 Zi 轴，从 Xi 旋转到 Xi+1 的角度
- D：沿 Zi 轴，从 Xi 移动到 Xi+1 的距离
- A：沿 Xi 轴，从 Zi 移动到 Zi+1 的距离
- alpha：绕 Xi+1 轴，从 Zi 旋转到 Zi+1 的角度

(2) 改进 DH

下面为采用改进 D-H 建模的程序段：

```
%改进 DH
%Link(DH,option)
%DH = [THETAi    Di    Ai-1    ALPHAi-1    SIGMA]
L1 = Link([0        0.2     1       pi/4            0],'modified');
L2 = Link([0        0.2     1       pi/4            0],'modified');
L3 = Link([0        0       0.5     pi/4            0],'modified');
robot = SerialLink([L1 L2 L3]); %建立连杆机器人
robot.plot([0 0 0]) %显示并赋三个关节变量 theta 的初始值都为 0
```

所建机器人如下图所示：

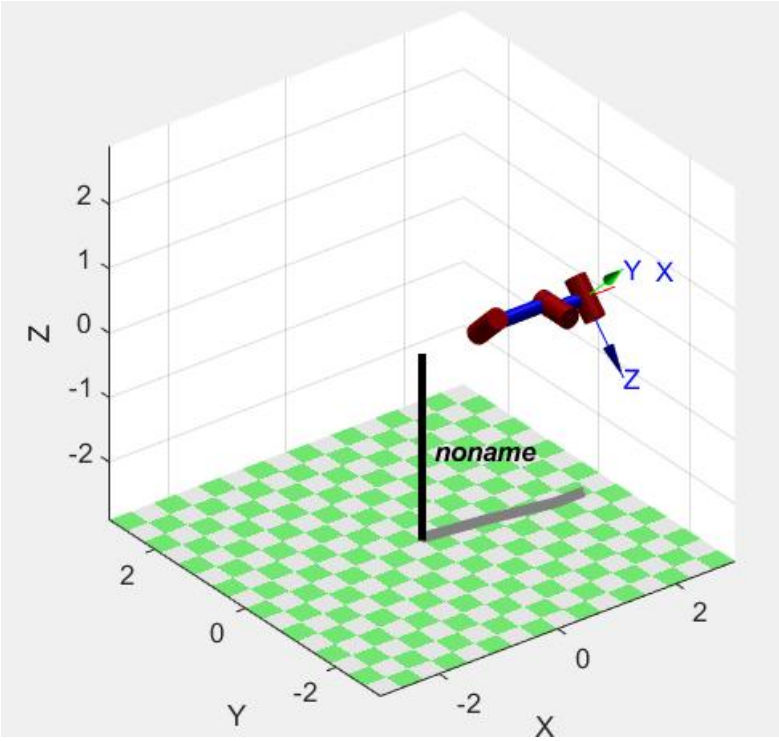


图 4.4　改进 D-H 建模图 1

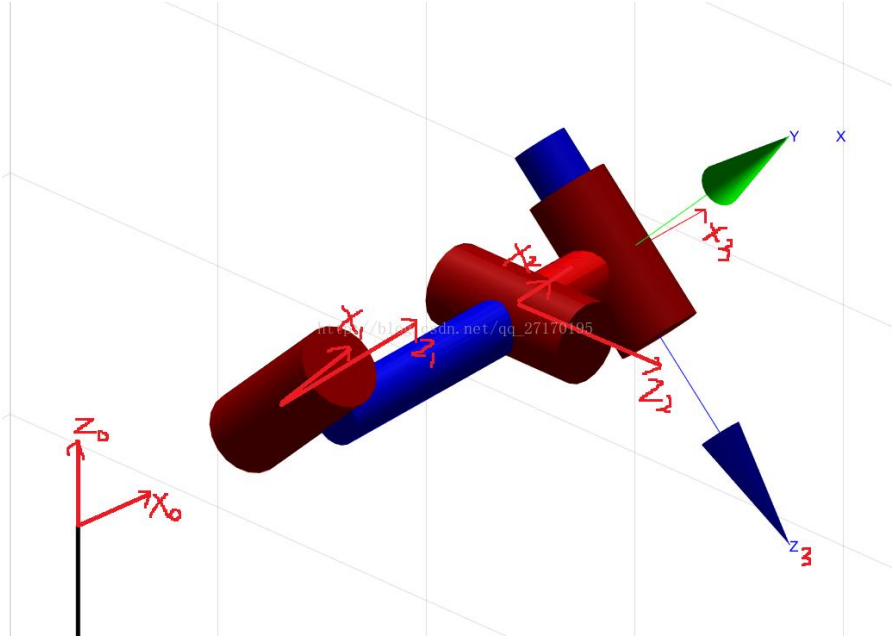


图 4.5　改进 D-H 建模图 2

改进 DH 连杆时一定要注意 DH[theta d a alpha]中前两个参数下表为 i，即当前关节的 DH 值，后两个参数下表为 i-1，及前一个关节的 DH 值参数值！

此时对比代码中的 DH 参数和图中画的坐标系，得到 DH 定义为：

- theta：绕 Zi 轴，从 Xi-1 旋转到 Xi 的角度
- D：沿 Zi 轴，从 Xi-1 移动到 Xi 的距离

- A：沿 Xi 轴，从 Zi 移动到 Zi+1 的距离
- alpha：绕 Xi 轴，从 Zi 旋转到 Zi+1 的角度

对比两段程序，以及图 4.2 和图 4.4 建模的结果，可见相同的参数，因为建模方法的不同所建机器人不同。

实际上,标准型和改进型中 A 和 alpha 定义相同,都是相对于下一关节而言，不同的是 theta 和 D 在标准型中都是相对于下一关节,在改进型中是相对于上一关节。

定义 D-H 表格时:

标准型的列标题为：THETAi　Di　Ai　ALPHAi

改进型的列标题为：THETAi　Di　Ai-1　ALPHAi-1

所以一定要注意 MATLAB 机器人工具箱中连杆定义中 DH 的对应。默认状态是标准 D-H 模式。

4.2 连杆 Link 类

1、 连杆 Link 的建立方法

创建一个连杆，需要用到 link（）函数。该函数可以根据输入的参数：关节角，连杆偏距，连杆长度，连杆角等产生一个 link 对象。

Link 对象包含了连杆的各种属性：运动学参数、惯性张量、电机、传递矩阵等。

Link 函数的用法如下：

L1=Link([0 0.138 0 -pi/2])。括号内输入的是参数，返回的 L1 是对象。

2、 Link 的类函数：

- A ：关节传动矩阵
- type：关节类型
- friction ： 摩擦力
- nofriction ： 摩擦为 0
- dyn ： 显示动力学参数
- islimit：检测关节变量是否超出范围

- isrevolute ： 检测关节是否为转动关节
- isprismatic ： 检测关节是否为移动关节
- display ： 显示 D-H 矩阵
- char ： 转化为字符串

3、 Link 的类属性（读/写）：

- theta：关节角度，D-H 参数
- d：连杆偏移量，D-H 参数
- a：连杆长度，D-H 参数
- alpha：连杆扭角，D-H 参数
- sigma：默认 0，旋转关节；1，移动关节
- mdh：默认 0，标准 D-H；1，改进 D-H
- offset:关节变量偏移量
- qlim:关节变量范围[min max]
- m：质量
- r：质心
- I：惯性张量
- B：粘性摩擦
- Tc：静摩擦
- G：减速比
- Jm：转子惯量

【例 4.1】定义连杆

```
>> L1=Link([0 0.138 0 -pi/2])

  L1 =

    Revolute(std): theta=q, d=0.138, a=0, alpha=-1.5708, offset=0

>>L1.type()

  ans =

       'R'
```

```
>> L1.theta

ans =

     0

>>L1.d

ans =

   0.1380

>> L1.a

ans =

     0

>>L1.alpha

ans =

  -1.5708

>> L1.offset

ans =

     0
```

4.3 机械臂 Seriallink 类

1、 Seriallink 的类函数：

类函数比较多，包括显示机器人、动力学、逆动力学、雅可比等，用的最多的是 SerialLink 和 plot。

R=SerialLink(links, options);

R.plot（theta）;

2、 Seriallink 的类属性（读/写）：

- links ： 连杆向量
- gravity ： 重力加速度
- base ： 基座标系
- tool：与基座标系的变换矩阵
- qlim ： 关节极限位置
- offset ： 关节偏移量
- name ： 机器人的名字
- manuf ： 制造者的名字
- comment： 注释

3、 Seriallink 的类属性（读）：

- n ： 关节数
- config： 关节配置，如‘RRRRRR’
- mdh ： D-H 矩阵类型
- theta ： D-H 参数
- d ： D-H 参数
- a ： D-H 参数
- alpha： D-H 参数

【例 4.2】定义 3 轴机械臂

首先用 Link 建立连杆，再用 SerialLink 建立机器人，最后用 plot 显示机器人。

```
clear;
clc;
L1 =   Link([ pi/4, 0, 0, 0, 0], 'modified');
L2 =   Link([ pi/2, 0.2, 0.1, 0 ,0], 'modified');
L3 =   Link([ 0, 0.1, 0.2, 0 ,0], 'modified');
robot=SerialLink([L1,L2,L3]); %用定义好的关节建立机器人
robot.display();   %显示建立的机器人的 DH 参数
theta=[0 0 0];     %6 个关节的角度变量值都设为 0，可以更改
robot.plot(theta);       %显示机器人的图像
```

在 Matlab 的命令行窗口中显示 display 的结果，即各个关节的参数，

```
robot =

noname:: 3 axis, RRR, modDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|          0|          0|
|  2|         q2|        0.2|        0.1|          0|          0|
|  3|         q3|        0.1|        0.2|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

plot 的图形为：

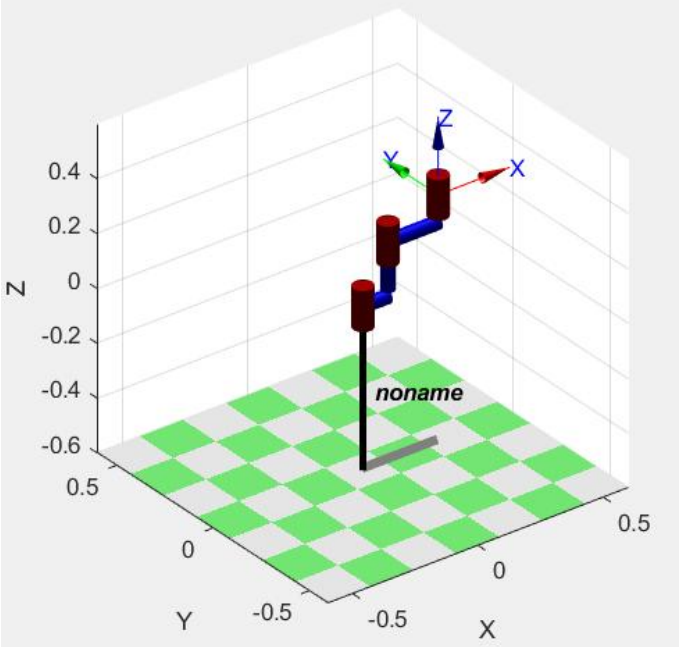


图 4.6　例 4.2 图

【例 4.3】定义 6 轴机械臂

定义连杆时，还可以用字符串指定参数的名称，随后跟该参数的数值。如下：

```
L1 = Link('d', 0, 'a', 0, 'alpha', pi/2);

L2 = Link('d', 0, 'a', 0.5, 'alpha', 0,'offset',pi/2);

L3 = Link('d', 0, 'a', 0, 'alpha', pi/2,'offset',pi/4);

L4 = Link('d', 1, 'a', 0, 'alpha', -pi/2);

L5 = Link('d', 0, 'a', 0, 'alpha', pi/2);

L6 = Link('d', 1, 'a', 0, 'alpha', 0);
```

```
robot=SerialLink([L1,L2,L3,L4,L5,L6]); %用定义好的关节建立机器人
robot.display();   %显示建立的机器人的 DH 参数
theta=[0 0 0 0 0 0];   %6 个关节的角度变量值都设为 0，可以更改
robot.plot(theta);      %显示机器人的图像
```

display 的结果：

```
robot =

noname:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|        0.5|          0|     1.5708|
|  3|         q3|          0|          0|     1.5708|   0.785398|
|  4|         q4|          1|          0|    -1.5708|          0|
|  5|         q5|          0|          0|     1.5708|          0|
|  6|         q6|          1|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

plot 的图形为：

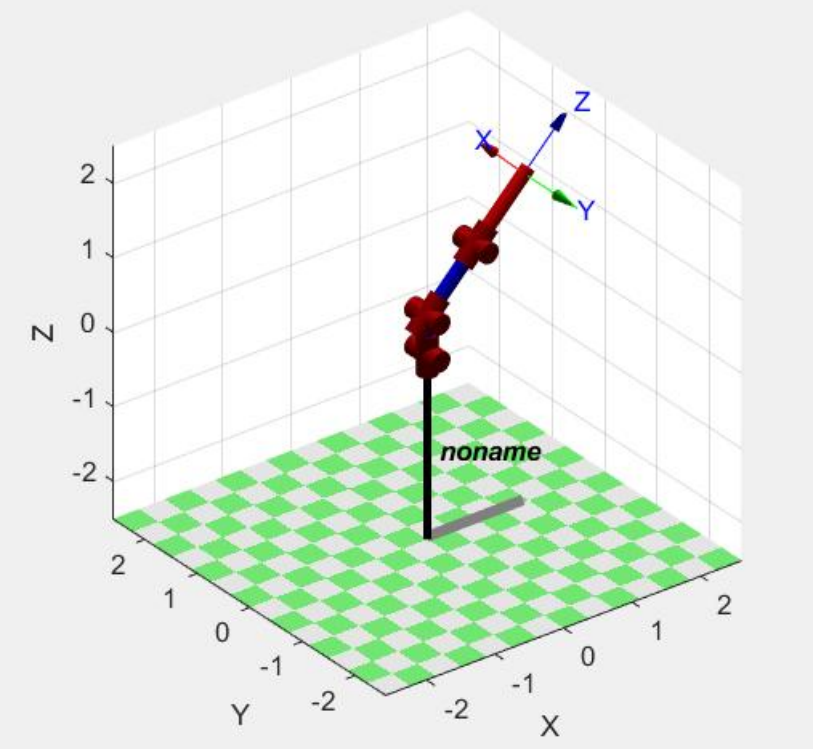


图 4.7　例 4.3 图

除了用户自己构建机器人连杆，然后构建机器人外，Robotics Toolbox 也自带了一些常见的机器人对象，如最常见的 puma560，standford 等。通过下面的语句即可调用工具箱已构建好的 puma560 机器人，并显示在三维空间中：

```
>> mdl_puma560
>> p560=SerialLink(p560)
p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

```
>> plot(p560,qz)；
```

将输出 p560 的图如下：

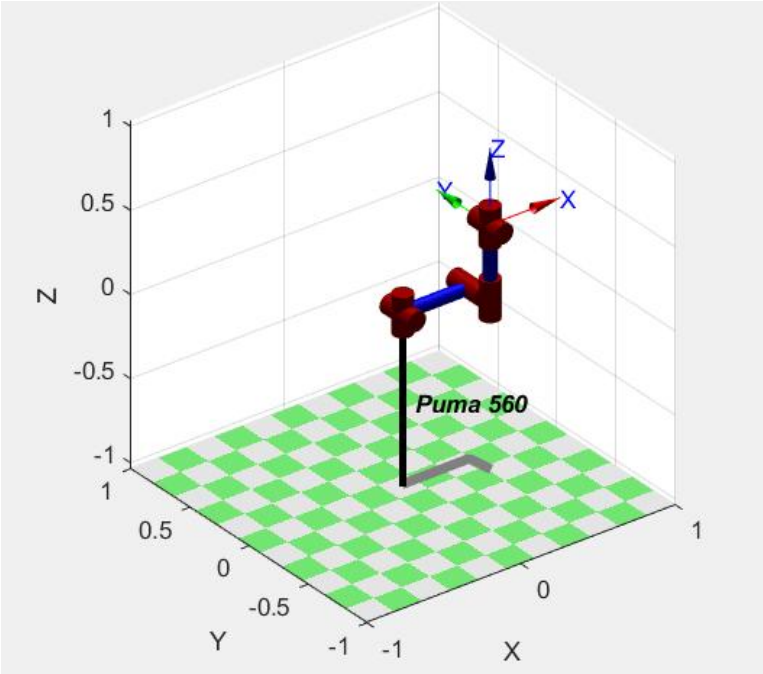


图 4.8　PUMA560 qz 状态图

在 mdl_puma560 工作空间里还自动创建了一些关节坐标向量，代表了一些典型的机器人位形。

qz：(0,0,0,0,0,0)　零角度

qr：(0，pi/2,-pi/2,0,0,0)　就绪状态，机械臂伸直切垂直

qs：(0,0，-pi/2，0,0,0)　伸展状态，机械臂伸直且水平

qn：(0，pi/4,-pi,0,pi/4,0)　标准状态，机械臂处于一个灵巧工作状态。

前面我们展示了 puma560 零角度状态的图，下面，我们展示一下它的其他状态：

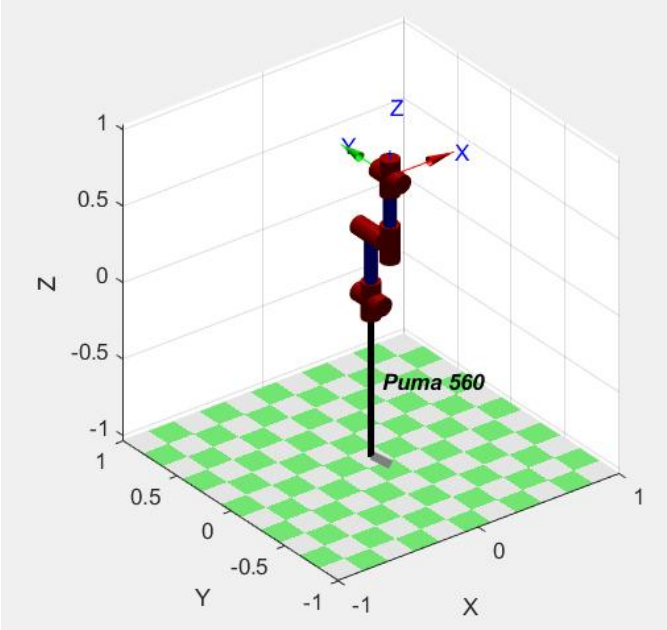


图 4.9　PUMA560 qr 状态图

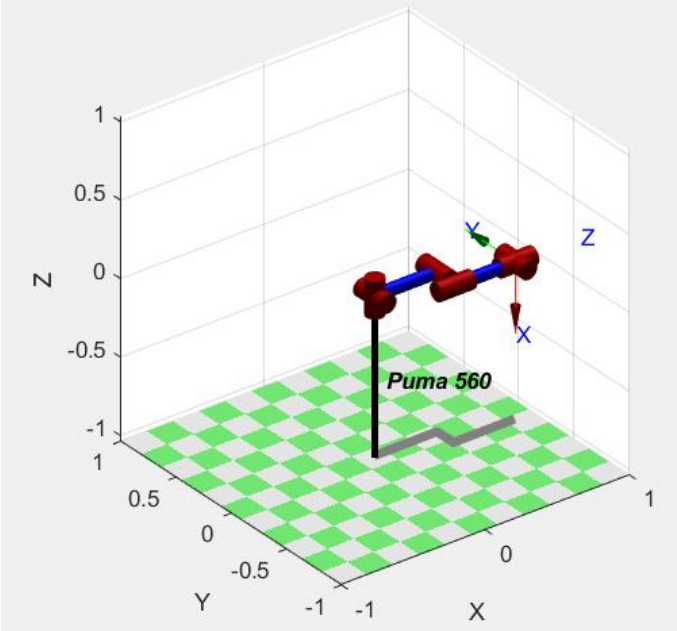


图 4.10　PUMA560 qs 状态图

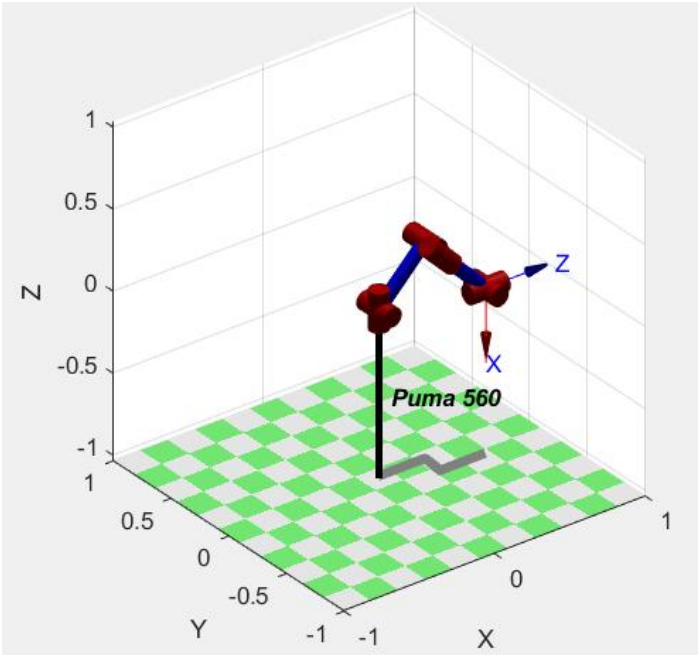


图 4.11　PUMA560 qn 状态图

4、　机器人示教

（1）首先建立机器人

```
clear;
clc;
%建立机器人模型
%      theta      d         a          alpha      offset
L1=Link([0       0.4       0.025      pi/2         0 ]); %定义连杆的 D-H 参数
L2=Link([pi/2    0         0.56       0            0 ]);
L3=Link([0       0         0.035      pi/2         0 ]);
L4=Link([0     0.515       0          pi/2         0 ]);
L5=Link([pi      0         0          pi/2         0 ]);
L6=Link([0       0.08      0          0            0 ]);
robot=SerialLink([L1 L2 L3 L4 L5 L6],'name','man'); %连接连杆，机器人取名 man
robot.plot([0,pi/2,0,0,pi,0]);%输出机器人模型，后面的六个角为输出时的 theta 姿态
```

所建立的机器人如下图所示：

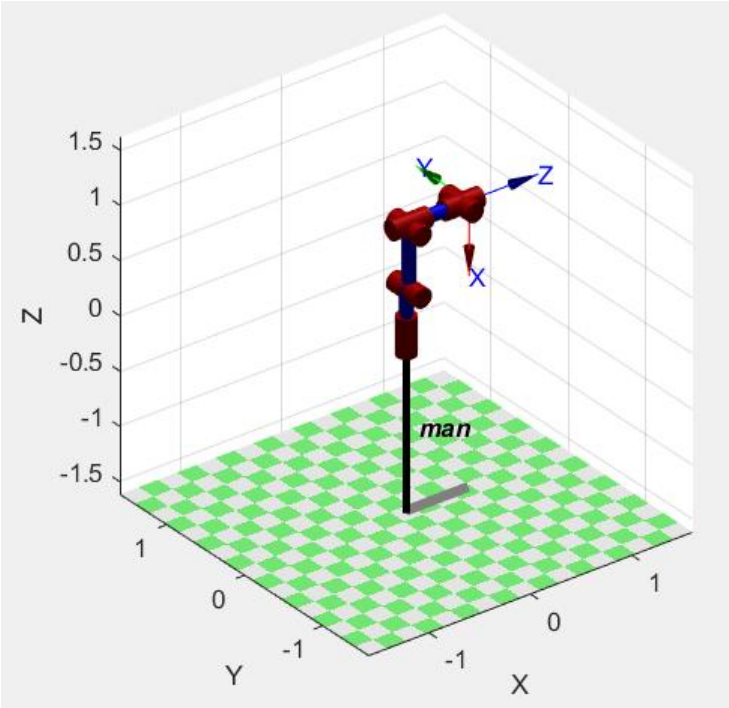


图 4.12　示教机器人图 1

display 函数可以输出模型的一些参数：

```
>> robot.display();

robot =

man:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|        0.4|      0.025|     1.5708|          0|
|  2|         q2|          0|       0.56|          0|          0|
|  3|         q3|          0|      0.035|     1.5708|          0|
|  4|         q4|      0.515|          0|     1.5708|          0|
|  5|         q5|          0|          0|     1.5708|          0|
|  6|         q6|       0.08|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

利用 teach 指令，则可调整各个关节角度，以达到示教的目的。

执行：>>teach(robot);

命令后图 4.12 变成了如图 4.13 所示的示教界面：

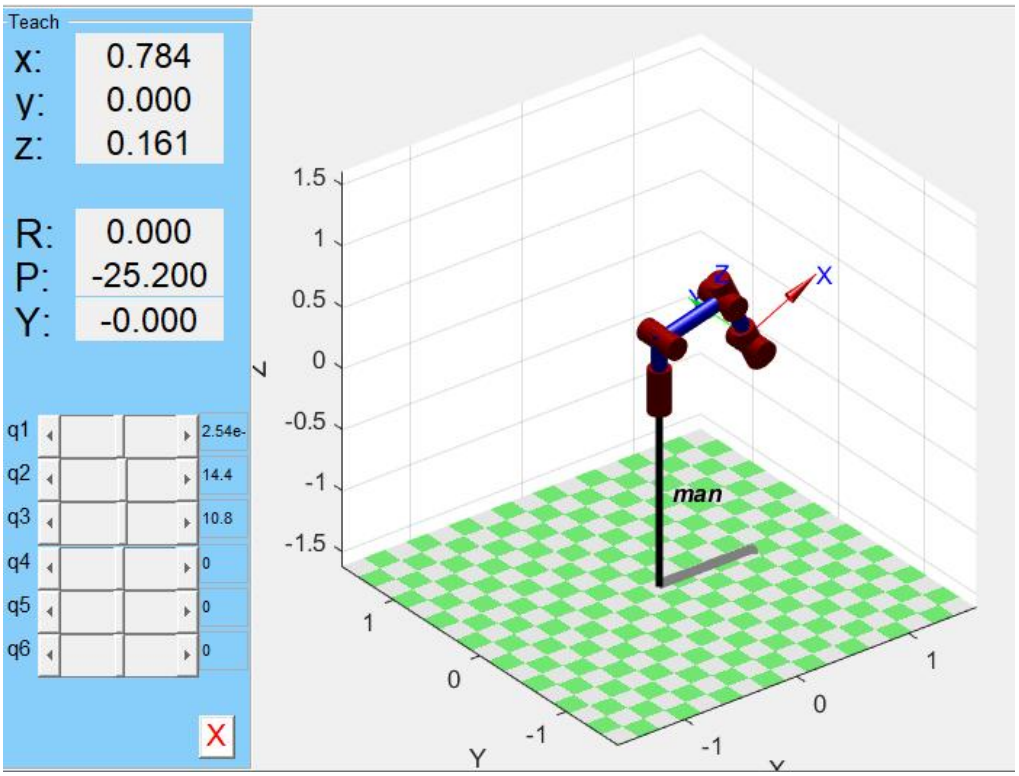


图 4.13　示教机器人图 2

点击图 4.13 中的 q1,q2,q3,q4,q5,q6 左右两边的箭头可以对相应的关节进行调整，使其绕自己的关节轴进行旋转。

4.3 本章实验内容

（1）练习所有例题，熟悉相关命令

（2）编写 MATLAB 程序，对教材中习题 2.33（P81）机器人建模，并用图展示出建模结果。

（3）写出实验报告，包含：实验内容，所编写的程序，以及程序运行结果的 MATLAB 截图，必要时对图中内容做个简单解释。

5. 基于 MATLAB Robotics Toolbox 机器人正逆运动学分析

机器人运动学主要是研究关节变量空间和机器人末端执行器位置和姿态之间的关系。常见的机器人运动学问题可以归纳为两大类：（1）运动学正问题（又

称为直接问题）：对一给定的机器人，一直杆件几何参数和关节角矢量，求机器人末端执行器相对参考坐标系的位置和姿态；（2）运动学逆问题（又称为解臂形问题）：已知机器人杆件的几何参数，给定机器人末端执行器相对于参考坐标系的期望位置和姿态（位姿），机器人能否使其末端执行器达到这个预期的位姿？

手工进行机器人运动学的求解，非常繁琐甚至无法得到最终的数值结果，这对于实际机器人的设计非常不利。因此，在仿真实验教学中，我们希望能通过计算机编程的形式来进行机器人运动学的求解，把设计人员从繁琐的数值计算中解脱出来。

5.1 基于 Robotics Toolbox 的正逆运动学分析

matlab 机器人工具箱 robotic toolbox 做运动学分析非常方便，SerialLink 类中有现成的函数：

SerialLink.fkine(theta)，可以直接对已经建立的机器人模型做正运动学分析

SerialLink.ikine（T），可以根据位姿求逆运动学参数。

下面以常用的 puma560 型机器人为例，演示如何运用 Robotics Toolbox 进行正运动学与逆运动学的求解。

如前面所述，我们先定义 puma560 机器人，注意系统同时还定义了 puma560 机器人两个特殊的配置：所有关节变量为 0 的 qz 状态，以及表示“READY”状态的 qr 状态。如我们要求解所有关节变为 0 时的末端机械手的状态，则相应的正运动学可由下述语句求解：

```
>> mdl_puma560;
>> p560=SerialLink(p560);
>>fkine(p560,qz)
ans =
    1    0    0    0.4521
    0    1    0     -0.15
    0    0    1    0.4318
    0    0    0         1
```

得到的即为末端机械手位姿所对应的齐次变换矩阵。

逆运动学问题则是通过一个给定的齐次变换矩阵，求解对应的关节变量。例如，假定机械手需要运动到[0, -pi/4,-pi/4,0,pi/8,0]姿态，则此时末端机械手位姿所对应的齐次变换矩阵为：

```
>>q=[0 -pi/4 -pi/4 0 pi/8 0];
>>T=fkine(p560,q)
T =
    0.3827         0    0.9239    0.7371
         0         1         0   -0.1501
   -0.9239         0    0.3827   -0.3256
         0         0         0    1
```

现在假定已知上述的齐次变换矩阵 T，则可以通过 ikine 函数，求解对应的关节转角：

```
>>qi=ikine(p560,T)
qi =
    0.0000   -0.7854   -0.7854    0.0000    0.3927   -0.0000
```

对比一下前面的 q

```
q =
         0   -0.7854   -0.7854         0    0.3927         0
```

发现两组数据基本一样，也就是说逆运动学分析的结果与原始的关节转角数值相同。值得指出的是，这样的逆运动学求解在手工计算中几乎是无法完成的。

5.2 本章实验内容

（1）练习本章例题，熟悉相关命令

（2）更改两组 q 的位置，重新利用该方法进行正逆运动学分析。

（3）写出实验报告，包含：实验内容，所编写的程序，以及程序运行结果的 MATLAB 截图，必要时对图中内容做个简单解释。

6. 基于 MATLAB Robotics Toolbox 机器人轨迹规划

轨迹规划是根据作业任务要求事先规定机器人的操作顺序和动作过程，轨迹规划分为关节空间和笛卡尔空间轨迹规划。

- 确定末端操作器的初始位置和目标位置
- 根据逆运动学求出各关节的初始角度和目标角度
- 估计规划，求出各关节的角度变化曲线
- 进行运动控制，使机器人按照轨迹规划结果运动

6.1 关节空间轨迹规划

思路：给定起始点 p1，目标点 p2, 利用运动学反解的得到 q1，q2，在关节空间内利用五次多项式做插补，得到[q, qd, qdd]，再通过运动学正解，得到末端执行器的位置，线速度与角速度的值

代码如下：

```
%TrajectoryGeneration_p2p_lnksps.m
clear;
clc;
%    :          theta          d            a            alpha        offset
ML1 = Link([ 0,          0.4967,        0,            0,             0], 'modified');
ML2 = Link([ -pi/2,    -0.18804,      0.2,          3*pi/2,        0], 'modified');
ML3 = Link([ 0,          0.17248,     0.79876,      0 ,            0], 'modified');
ML4 = Link([ 0,          0.98557,     0.25126,    3*pi/2,        0], 'modified');
ML5 = Link([ 0,            0,           0,            pi/2 ,         0], 'modified');
ML6 = Link([ 0,            0,           0,            pi/2 ,         0], 'modified');
robot = SerialLink([ML1 ML2 ML3 ML4 ML5 ML6],'name','Fanuc M20ia');


%给定末端执行器的初始位置
```

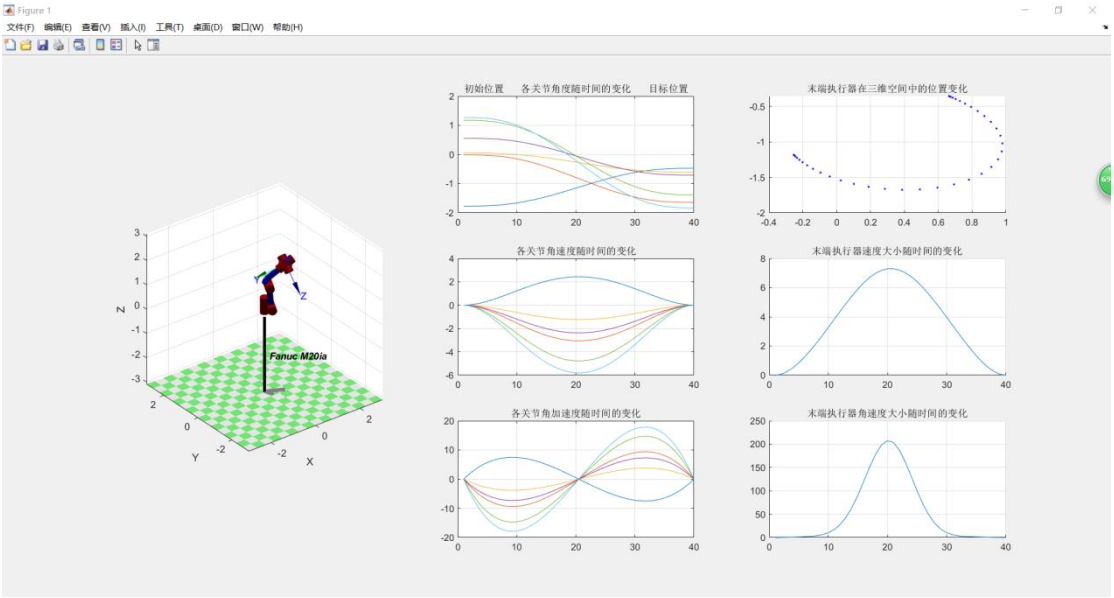
```
p1 =[
    0.617222144     0.465154659     -0.634561241     -0.254420286
    -0.727874557    0.031367208     -0.684992502     -1.182407321
    -0.298723039    0.884673523      0.357934776     -0.488241553
         0               0               0               1
];

%给定末端执行器的终止位置
p2 = [
    -0.504697849    -0.863267623     -0.007006569     0.664185871
    -0.599843651    0.356504321      -0.716304589    -0.35718173
    0.620860432     -0.357314539     -0.697752567     2.106929688
         0               0               0               1
];

%利用运动学反解 ikine 求解各关节转角
% Inverse kinematics by optimization without joint limits
% q = R.ikine(T) are the joint coordinates    corresponding to the robot
    end-effector
% pose T which is an SE3 object or homogenenous transform matrix (4 4), and
  N is the number of robot joints.

init_ang = robot.ikine(p1);%使用运动学迭代反解的算法计算得到初始的
                           关节角度
targ_ang = robot.ikine(p2);%使用运动学迭代反解的算法计算得到目标关
                           节角度

%利用五次多项式计算关节速度和加速度
% Compute a joint space trajectory
```

```
% [q,qd,qdd] = jtraj(q0, qf, m) is a joint space trajectory q (m N) where the
    joint coordinates vary from q0 (1 N) to qf (1 N). A quintic (5th order)
    polynomial is used
% with default zero boundary conditions for velocity and acceleration. Time is
    assumed to vary from 0 to 1 in m steps. Joint velocity and acceleration can
    be optionally returned as qd (m N) and qdd (m N) respectively. The
    trajectory q, qd and qdd are m*N matrices, with one row per time step, and
    one column per joint.

step=40;
[q,qd,qdd] = jtraj(init_ang, targ_ang, step);

%  显示机器人姿态随时间的变化
subplot(3,3,[1,4,7]);
robot.plot(q);

%显示机器人关节运动状态
subplot(3,3,2);
i=1:6;
   plot(q(:,i));
title('初始位置  各关节角度随时间的变化  目标位置');
grid on;
subplot(3,3,5);
i=1:6;
   plot(qd(:,i));
title('各关节角速度随时间的变化');
grid on;
subplot(3,3,8);
i=1:6;
   plot(qdd(:,i));
```

```
 title('各关节角加速度随时间的变化');
 grid on;

 %显示末端执行器的位置
 subplot(3,3,3);
 hold on
 grid on
 title('末端执行器在三维空间中的位置变化');
 for i=1:step
     position = robot.fkine(q(i,:));
 plot3(position.t(1),position.t(2),position.t(3),'b.','MarkerSize',5);
end

 %显示末端执行器的线速度与角速度
 % Jacobian in world coordinates
 % j0 = R.jacob0(q, options) is the Jacobian matrix (6 N) for the robot in pose q
    (1 *N), and N is the number of robot joints. The manipulator Jacobian matrix
    maps joint
 % velocity to end-effector spatial velocity V = j0*QD expressed in the
      world-coordinate frame.
 subplot(3,3,6);
 hold on
 grid on
 title('末端执行器速度大小随时间的变化');
 vel = zeros(step,6);
 vel_velocity = zeros(step,1);
 vel_angular_velocity = zeros(step,1);
for i=1:step
 vel(i,:) = robot.jacob0(q(i,:))*qd(i,:)';
 vel_velocity(i) = sqrt(vel(i,1)^2+vel(i,2)^2+vel(i,3)^2);
```

```
    vel_angular_velocity(i) = sqrt(vel(i,4)^2+vel(i,5)^2+vel(i,3)^6);
end
 x = linspace(1,step,step);
 plot(x,vel_velocity);
 subplot(3,3,9);
 hold on
 grid on
 title('末端执行器角速度大小随时间的变化');
 x = linspace(1,step,step);
 plot(x,vel_angular_velocity);

 %EOF
```

效果如下



从图中解得的曲线证明：在关节空间内进行五次多项式插补得到的机械臂关节运动学曲线较为平滑，而且末端执行器的速度和角速度随时间的变化都比较均匀而平滑。

6.2 笛卡尔空间轨迹规划

思路：给定起始点 p1，目标点 p2, 利用梯形速度插补得到末端坐标系位姿矩阵 Tc 随时间的变化，再对每一个 Tc 通过运动学反解，得到各个关节的角度，并估算各个关节的角速度与角加速度的值，以及末端执行器的速度与加速度的大小。

代码如下

```
%TrajectoryGeneration_p2p_Cartesian.m
clear;
clc;
%                    theta d      a      alpha offset
ML1 = Link([ 0, 0.4967, 0, 0, 0], 'modified');
ML2 = Link([ -pi/2, -0.18804, 0.2, 3*pi/2, 0], 'modified');
ML3 = Link([ 0, 0.17248, 0.79876, 0 , 0], 'modified');
ML4 = Link([ 0, 0.98557, 0.25126, 3*pi/2, 0], 'modified');
ML5 = Link([ 0, 0, 0, pi/2 , 0], 'modified');
ML6 = Link([ 0, 0, 0, pi/2 , 0], 'modified');
robot = SerialLink([ML1 ML2 ML3 ML4 ML5 ML6],'name','Fanuc M20ia');

%给定末端执行器的初始位置
p1 =[
        0.617222144      0.465154659      -0.634561241      -0.254420286
        -0.727874557      0.031367208      -0.684992502     -1.182407321
        -0.298723039      0.884673523       0.357934776     -0.488241553
             0                  0                  0                 1
];

%给定末端执行器的终止位置
p2 = [
        -0.504697849      -0.863267623      -0.007006569       0.664185871
        -0.599843651      0.356504321     -0.716304589       -0.35718173
        0.620860432       -0.357314539     -0.697752567       2.106929688
```

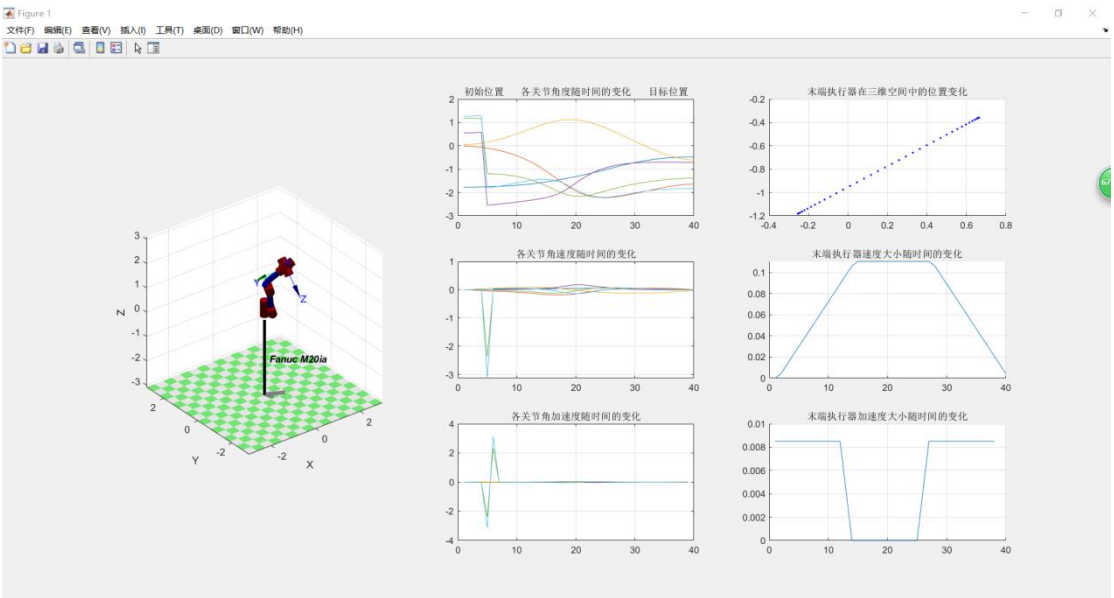
```
                0                0                0                1
];

%在笛卡尔空间坐标系内，根据梯形速度生成轨迹
% Cartesian trajectory between two poses
 % tc = ctraj(T0, T1, n) is a Cartesian trajectory (4 4 n) from pose T0 to T1
   with n points that follow a trapezoidal velocity profile along the path. The
   Cartesian trajectory is a homogeneous transform sequence and the last
   subscript being the point index, that is, T(:,:,i) is the i^th point along the
   path.

step = 40;
Tc = ctraj(p1,p2,step);

% Inverse kinematics by optimization without joint limits
 % q = R.ikine(T) are the joint coordinates (1 N) corresponding to the robot
   end-effector pose T which is an SE3 object or homogenenous transform
   matrix (4 4), and N is the number of robot joints.

%  显示机器人姿态随时间的变化
subplot(3,3,[1,4,7]);
q = zeros(step,6);
for i = 1:step
    q(i,:) = robot.ikine(Tc(:,:,i));
end
robot.plot(q);

qd = zeros(step-1,6);
for i = 2:step
    qd(i,1) = q(i,1)-q(i-1,1);
```

```
    qd(i,2) = q(i,2)-q(i-1,2);
    qd(i,3) = q(i,3)-q(i-1,3);
    qd(i,4) = q(i,4)-q(i-1,4);
    qd(i,5) = q(i,5)-q(i-1,5);
    qd(i,6) = q(i,6)-q(i-1,6);
end

qdd = zeros(step-2,6);
for i = 2:step-1
    qdd(i,1) = qd(i,1)-qd(i-1,1);
    qdd(i,2) = qd(i,2)-qd(i-1,2);
    qdd(i,3) = qd(i,3)-qd(i-1,3);
    qdd(i,4) = qd(i,4)-qd(i-1,4);
    qdd(i,5) = qd(i,5)-qd(i-1,5);
    qdd(i,6) = qd(i,6)-qd(i-1,6);
end

 %显示机器人关节运动状态
 subplot(3,3,2);
 i=1:6;
    plot(q(:,i));
 title('初始位置                各关节角度随时间的变化                目标位置');
 grid on;
 subplot(3,3,5);
    i=1:6;
 plot(qd(:,i));
 title('各关节角速度随时间的变化');
 grid on;
 subplot(3,3,8);
 i=1:6;
```

```
    plot(qdd(:,i));
title('各关节角加速度随时间的变化');
grid on;


%显示末端执行器的位置
subplot(3,3,3);
hold on
grid on
title('末端执行器在三维空间中的位置变化');
for i=1:step
    plot3(Tc(1,4,i),Tc(2,4,i),Tc(3,4,i),'b.','MarkerSize',5);
end

%显示末端执行器的速度
subplot(3,3,6);
hold on
grid on
title('末端执行器速度大小随时间的变化');
vel_velocity = zeros(step,1);
for i=2:step
     vel_velocity(i) =
          sqrt((Tc(1,4,i)-Tc(1,4,i-1))^2+(Tc(2,4,i)-Tc(2,4,i-1))^2+(Tc(3,4,i)-Tc(3,
          4,i-1))^2);
end
x = linspace(1,step,step);
plot(x,vel_velocity);

%显示末端执行器的加速度
subplot(3,3,9);
```

```
hold on
grid on
title('末端执行器加速度大小随时间的变化');
vel_acceleration= zeros(step-2,1);
for i=3:step
    vel_acceleration(i-2) =
        sqrt((Tc(1,4,i)-Tc(1,4,i-1)-(Tc(1,4,i-1)-Tc(1,4,i-2)) )^2+( Tc(2,4,i)-Tc
        (2,4,i-1)-(Tc(2,4,i-1)-Tc(2,4,i-2)) )^2+( Tc(3,4,i)-Tc(3,4,i-1)-(Tc(3,4,i
        -1)-Tc(3,4,i-2)))^2);
end
x = linspace(1,step-2,step-2);
plot(x,vel_acceleration);

%EOF
```

效果如下：



从图中解得的曲线证明：在笛卡尔坐标系内做速度的梯形插补，其末端执行器的轨迹是一条直线，速度是一个梯形，即为匀加速，匀速，匀减速过程。加速度在两个值之间突变，机械臂存在振动和柔性冲击。另外，关节空间内的曲线也

出现尖锐的突变点，说明在这段范围内机械臂的运动经过奇点附近，即使末端执行器的速度比较小，但是关节的角速度和角加速度却急剧变化。

6.3 本章实验内容

（1）练习所有例题，熟悉相关命令

（2）编写 MATLAB 程序，对教材中例题 5.7（图 5.18，P152）机器人建模，并用图展示出建模结果，完成对本例题的仿真计算。

（3）写出实验报告，包含：实验内容，所编写的程序，以及程序运行结果的 MATLAB 截图，必要时对图中内容做个简单解释。