MATLAB 中的机械臂算法——运动学

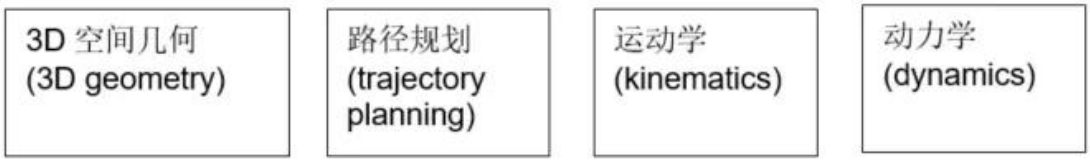
原创：周末MATLAB2019-04-03

作 者 简 介

周末，MathWorks 中国行业市场部经理，专注于基于模型的系统和软件设计。负责 MATLAB/Simulink 在机器人、汽车电子、电力电子等行业的推广和应用，曾就职于 IBM 和上海贝尔。

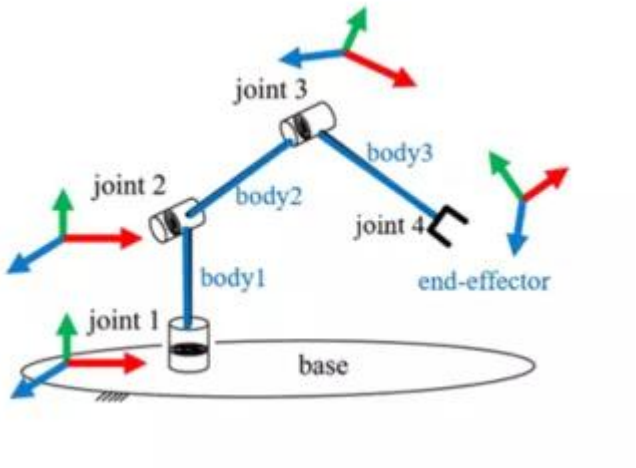
机械臂算法

MATLAB 在 2016 年就推出了 Robotics System Toolbox(RST)，其中有很多关于机械臂方面的算法。而且随着客户需求的增加，也在加入一些新的功能。为了试图让读者了解更多 RST 在机械臂方面的支持，让我们来看一下机械臂方面的算法概貌。



这些名词听起来都比较深奥，但是在机械臂的世界里，这些都非常有用。

让我们看一个简单的例子。下图是一个简单的机械臂示意：机械臂的 end-effector（末端机构）受到 4 个旋转关节和 3 个连杆的共同作用，可以到达不同的作业地点，也可以处于不同的旋转角度。



为了分析 end-effector 的具体位置和角度，我们看到：

它相对底座开始，做了 4 次旋转（rotation）和 3 次转置（translation）。那这 4 此旋转和 3 此转置的总和，我们可以用一个矩阵来表示：

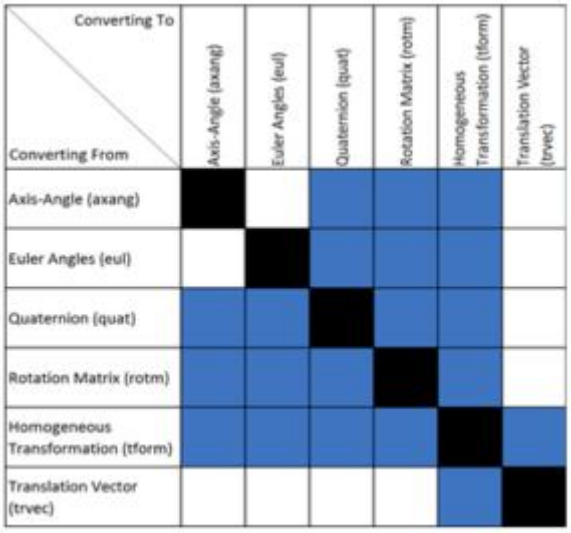
$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation　　translation

这个矩阵也叫 Homogeneous Transformation（齐次变换）。

有时候，对于旋转会有不同的表达方式，例如欧拉角（Euler Angles）、四元素（Quaternion）、旋转矩阵（Rotation Matrix）等等；表达转置，也可使用转置向量（Translation Vector）。有了 RST 这些都可以轻松通过不同的函数进行互换。

下图为具体的函数列表：



例如：将欧拉角转为 Homogeneous Transformation：

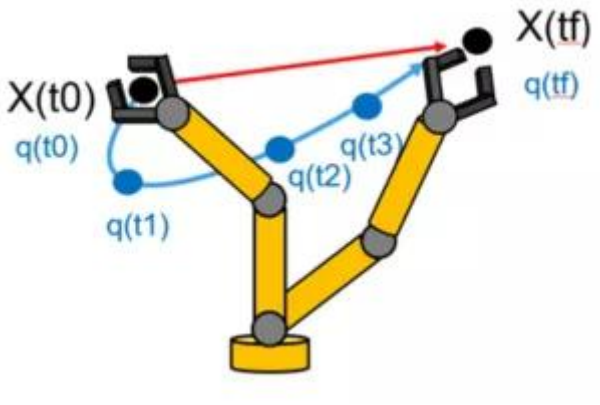
```
>> eul = [0 pi/2 0];
tformZYX = eul2tform(eul)

tformZYX =

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

由于机械臂的连杆长度是已知的，只要确定了各个关节转动的角度，我们就可以确定 end-effector 的最终位置和方向。这个我们称之为 forward kinematics（正向运动学）。反过来，如果我们知道了 end-effector 的最终位置和方向，我们也可以推导各个关节的角度，这个我们称之为 inverse kinematics（反向运动学）。

机械臂关注的主要是反向运动学。

如果 end-effector，需要走一段比较长的路程(path)，从甲点运行到乙点。我们为了使得机械臂的 end-effector 的路径平滑，需要规划一系列的路径点（waypoints），这个我们叫做路径规划（trajectory planning）或者叫运动插补

（interpolation）。例如下图，蓝色的曲线叫 path，而各个时间经过的路径点叫 trajectory。如何设计经过这些路径点的 trajectory，比较显而易见的指标是 “平滑” 。那什么是“平滑” ，它可能意味着 “速度连续” 、“加速度连续” 、 “没有顿挫” 等等。这些指标，都会转化成数学算法。 RST 也会有相应的算法支持，作者将另外写文章描述。



机械臂的关节位置我们一般用电机来驱动。电机通过产生力矩来转动机械装置，驱动机械臂。不同场合或者时机，需要的力矩不尽相同。

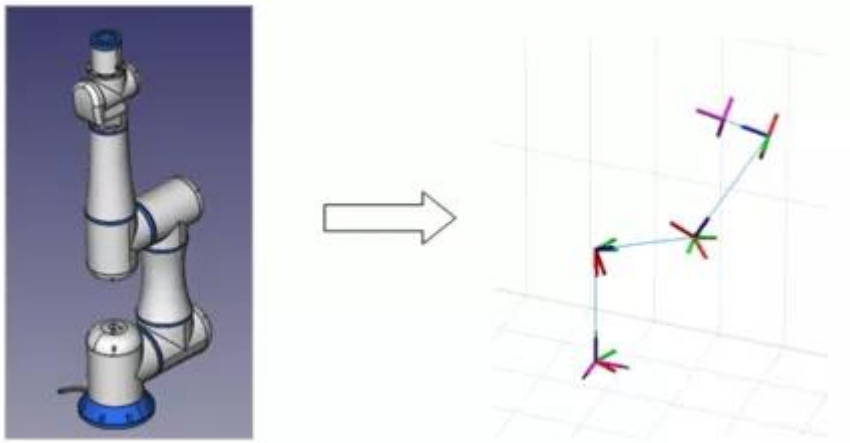
例如，机械臂水平放置的时候需要关节电机产生力矩来抵消地球引力；当机械臂需要迅速移动的时候，需要的力矩比缓慢移动的要大，当机械臂弯曲或者平展时候，重心发生变化，由于惯量（$I = mr^2$）的不同，需要的关节力矩也不相同；另外，在很多场合，机械臂需要和人交互（collaborative robots），在碰到人体的时候，需要做出安全的保护动作，并对力矩进行调整。

这些需要考虑力矩的因素，我们称之为动力学（dynamics）。和运动学类似，动力学分为正向动力学(forward dynamics)和反向运动学(inverse dynamics)。 RST 里支持两种都有相应的 MATLAB 函数和 Simulink block。作者也会另外写文章详细介绍 RST 关于动力学的部分。

运动学

1. Rigid Body Tree （刚体树）

我们说研究运动学（主要是反向运动学），就是研究 end-effector 的位置改变会带动各个关节的角度如何改变。RST 用 Rigid Body Tree 这样一个对象，在这个对象上可以使运动学设计易用且可视化。下图展示了机械臂的刚体树样例，可以在 MATLAB 界面中展示各个 body 的详细参数。



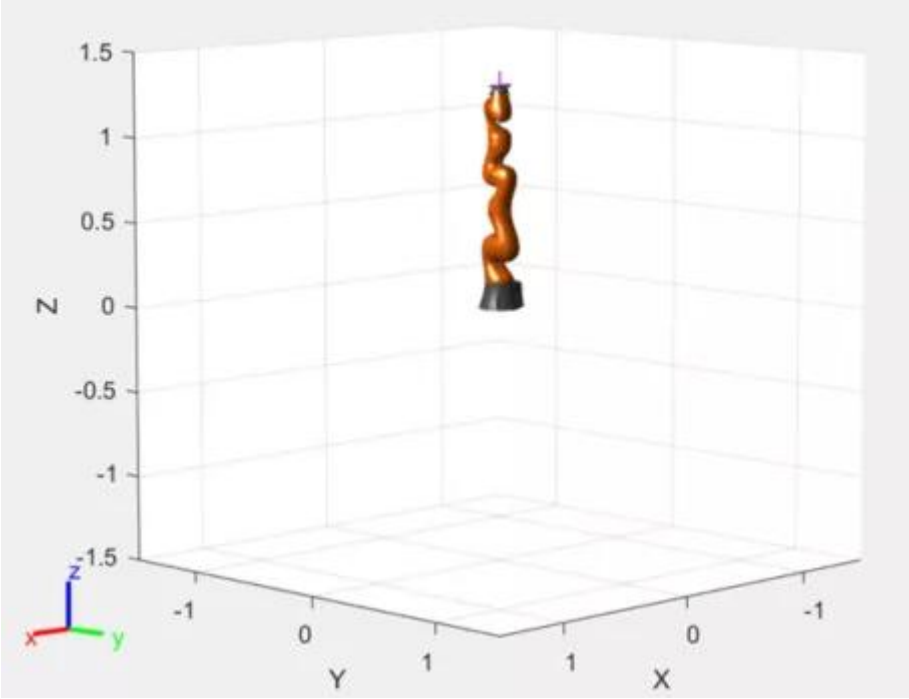
```
>> showdetails(lbr)
--------------------
Robot: (9 bodies)

 Idx     Body Name           Joint Name          Joint Type     Parent Name(Idx)   Children Name(s)
 ---     ---------           ----------          ----------     ----------------   ----------------
   1       link_1              joint_a1           revolute         base_link(0)     link_2(2)
   2       link_2              joint_a2           revolute            link_1(1)     link_3(3)
   3       link_3              joint_a3           revolute            link_2(2)     link_4(4)
   4       link_4              joint_a4           revolute            link_3(3)     link_5(5)
   5       link_5              joint_a5           revolute            link_4(4)     link_6(6)
   6       link_6              joint_a6           revolute            link_5(5)     link_7(7)
   7       link_7              joint_a7           revolute            link_6(6)     tool0(8)
   8        tool0        joint_a7_tool0              fixed            link_7(7)
   9         base       base_link_base              fixed         base_link(0)
--------------------
```

一般来说，Rigid Body Tree 都是直接从机械臂的 CAD 文件或者 URDF（Unified Robot Description Format）文件导入。不过，也支持每个 body 的逐步添加。

我们随便敲几行 MATLAB 命令：

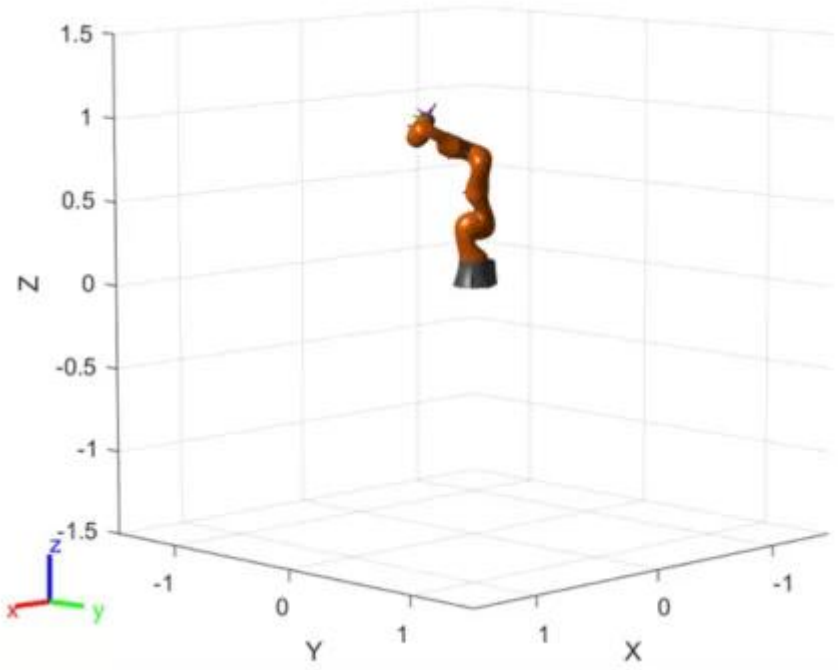
```
robot = importrobot('iiwa14.urdf');
```

```
show(robot);
```



让我们来改变一下机器人的各个关节角度（configuration），比如让 MATLAB 自动给一个随机角度配置，再看一下结果。显然各个角度发生了变化。

```
q=randomConfiguration(robot);
show(robot,q);
```



我们看看这个机械臂最末端的 end-effector 是什么？

```
showdetails(robot)
```

Idx	Body Name	Joint Name	Joint Type	Parent Name(Idx)	Children Name(s)
1	iiwa_link_0	world_iiwa_joint	fixed	world(0)	iiwa_link_1(2)
2	iiwa_link_1	iiwa_joint_1	revolute	iiwa_link_0(1)	iiwa_link_2(3)
3	iiwa_link_2	iiwa_joint_2	revolute	iiwa_link_1(2)	iiwa_link_3(4)
4	iiwa_link_3	iiwa_joint_3	revolute	iiwa_link_2(3)	iiwa_link_4(5)
5	iiwa_link_4	iiwa_joint_4	revolute	iiwa_link_3(4)	iiwa_link_5(6)
6	iiwa_link_5	iiwa_joint_5	revolute	iiwa_link_4(5)	iiwa_link_6(7)
7	iiwa_link_6	iiwa_joint_6	revolute	iiwa_link_5(6)	iiwa_link_7(8)
8	iiwa_link_7	iiwa_joint_7	revolute	iiwa_link_6(7)	iiwa_link_ee(9)
9	iiwa_link_ee	iiwa_joint_ee	fixed	iiwa_link_7(8)	
10	iiwa_link_ee_kuka	iiwa_joint_ee_kuka	fixed	iiwa_link_7(8)	

我们再看看 end-effector 相对机器人底座(base)的 Homogeneous Transformation(相对位置和角度)。

```
>> transform = getTransform(robot,q,'iiwa_link_ee_kuka')
```

```
transform =

   -0.5562    0.5275   -0.6422    0.2603
   -0.6685   -0.7430   -0.0313   -0.1106
   -0.4937    0.4119    0.7659    1.0296
         0         0         0    1.0000
```

Rotation

Translation

2. 反向运动学算法

反向运动学算法求解分两种：一种是分析解法（Analytic solutions）；一种是数值解法（Numerical solutions）。

MATLAB 用的是数值解法，可以理解为迭代寻优，或者近似解。

MATLAB 里面的反向运动学求解器(solver)有两个：

1. Inverse Kinematics
2.

3.

Generalized Inverse Kinematics

4.

两者的区别是，后者比前者多了很多限制(constraints) 。例如 end-effector 的方向限制、机械臂各个关节的角度限制、位置限制等等。

我们先看一下比较简单的 Inverse Kinematics：

这是一个 6 轴机器人，end-effector 是 L6。

我们想要的最终结果就是下图：

tform 是 L6 相对 base 的位置和方向（合称 pose）。

下面的 MATLAB 代码是计算出最终的各个关节的角度（configSoln），由于是用了迭代的数值解法，weights 为权重，initialguess 为给出一个初始估计。

```
ik = robotics.InverseKinematics('RigidBodyTree',puma1);
weights = [0.25 0.25 0.25 1 1 1];
initialguess = puma1.homeConfiguration;

[configSoln,solnInfo] = ik('L6',tform,weights,initialguess);
```

我们再看一下比较复杂的 Generalized Inverse Kinematics:

下面的代码，做了这么几件事情：

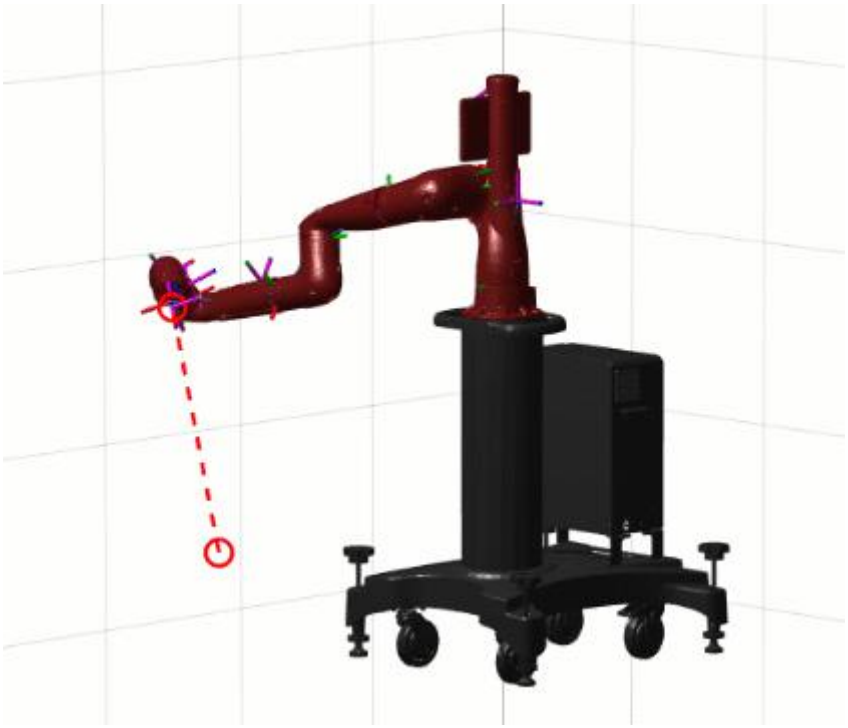
1.

导入了一个 7 自由度的 rethink 机械臂 — sawyer

2.
3. 设定反向运动学的求解限制 — 例如机械臂的 end-effector 永远指向地面的一个物体
4.
5. 对反向运动学进行求解
6.

```
sawyer = importrobot('sawyer.urdf', 'MeshPath', ...
    fullfile(fileparts(which('sawyer.urdf')),'..','meshes','sawyer_pv'));
gik = robotics.GeneralizedInverseKinematics('RigidBodyTree',sawyer, ...
    'ConstraintInputs',{'position','aiming'});

% Target Position constraint
targetPos = [0.5, 0.5, 0];
handPosTgt =
robotics.PositionTarget('right_hand','TargetPosition',targetPos);

% Target Aiming constraint
targetPoint = [1, 0, -0.5];
handAimTgt =
robotics.AimingConstraint('right_hand','TargetPoint',targetPoint);

% Solve Generalized IK
[gikSoln,solnInfo] = gik(sawyer.homeConfiguration,handPosTgt,handAimTgt)
show(sawyer,gikSoln);
```

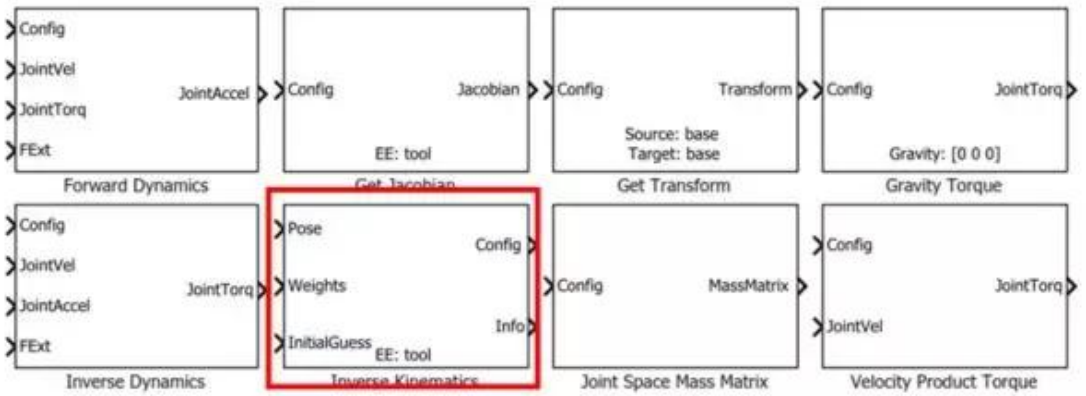
如果我们加一段 end-effector 位置变化后，调用这段代码的动画效果，你会发现 end-effector 的指向没有变化 – 带限制的反向动力学求解成功了：



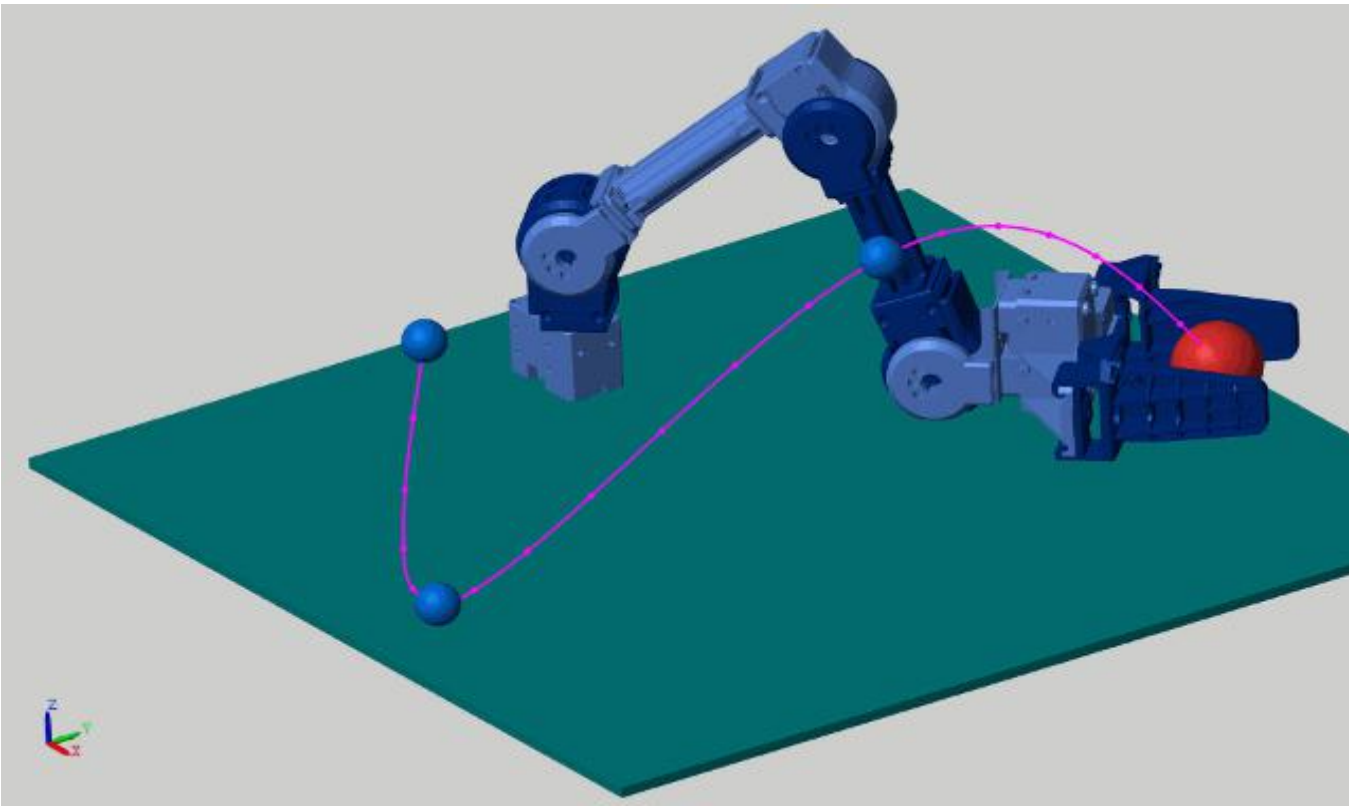
3. Simulink 示例

在安装 RST 之后，Simulink 的 library 里就会出现几个和机械臂（manipulator）相关的 block：

其中 Inverse Kinematics 就是反向运动学 block，其他的一些模块顾名思义和动力学有关，在下一篇文章我会重点介绍。

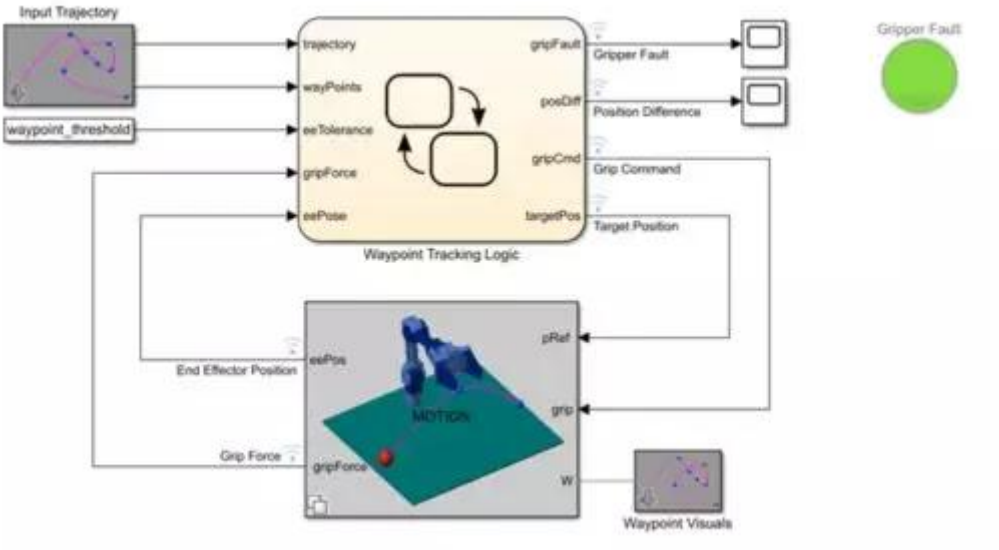


在 MATLAB Central File Exchange 上搜索 “Designing Robot Manipulator Algorithms”，这是一个基于 Simulink 和 Stateflow 的例子。我们先看一下运行结果：

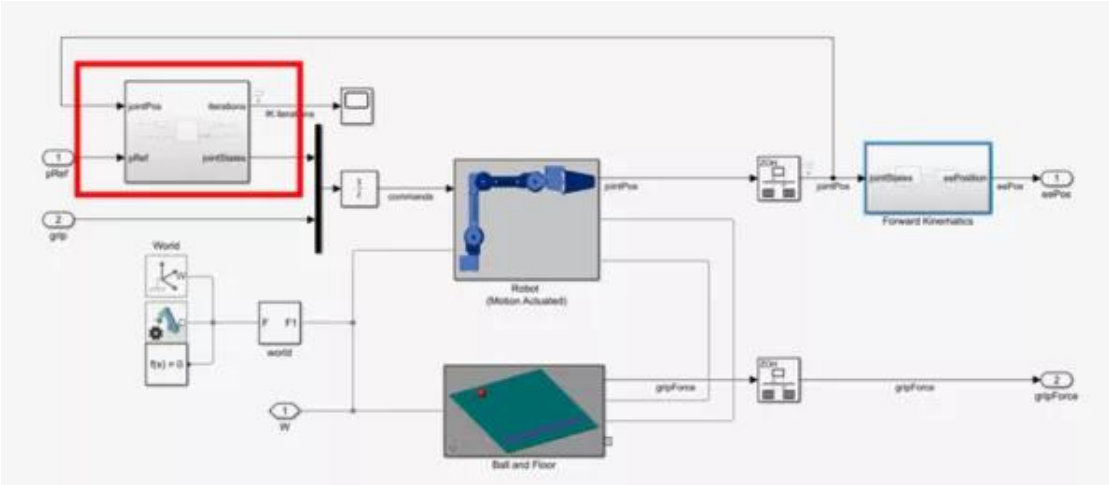


这个例子展现了机械臂的 end-effector 抓了红色物体，沿着规划好的紫色 trajectory，进行运动。

下图的 stateflow 状态机是一个 trajectory tracking 的算法，它的作用是确保 end-effctor 沿着预设的 trajectory 运行。



状态机下面的是运动控制部分和环境和物理模型。运动控制很简单 – 直接计算反向运动学，将算好的关节角度交给物理模型去展现。物理模型构建也很简单—— 用 SimScape 中的 SimMultibody 直接导入机械臂的 URDF 文件即可。



这里可以看到物理模型并没有包含伺服电机，而是“透明传输” — 反向运动学的结果直接发给了机械模型去展现。实际上真实的运动控制器会将位置、速度、力矩指令通过伺服总线（例如 EtherCAT）发给每个关节的电机去执行，电机通过减速器去带动机械结构。例如，一个 6 轴机械臂会有 6 个伺服电机，运动控制器会将运动过程解析为 6 个电机可以理解的位置、速度、力矩指令。