# 添加杆件

添加杆件，这里pe的意思为position and euler angle，函数的参数指定了位姿以及惯性向量

**auto& p1 = model->addPartByPe(link1\_position\_and\_euler321, "321", iv);**

**auto& p2 = model->addPartByPe(link2\_position\_and\_euler321, "321",iv);**

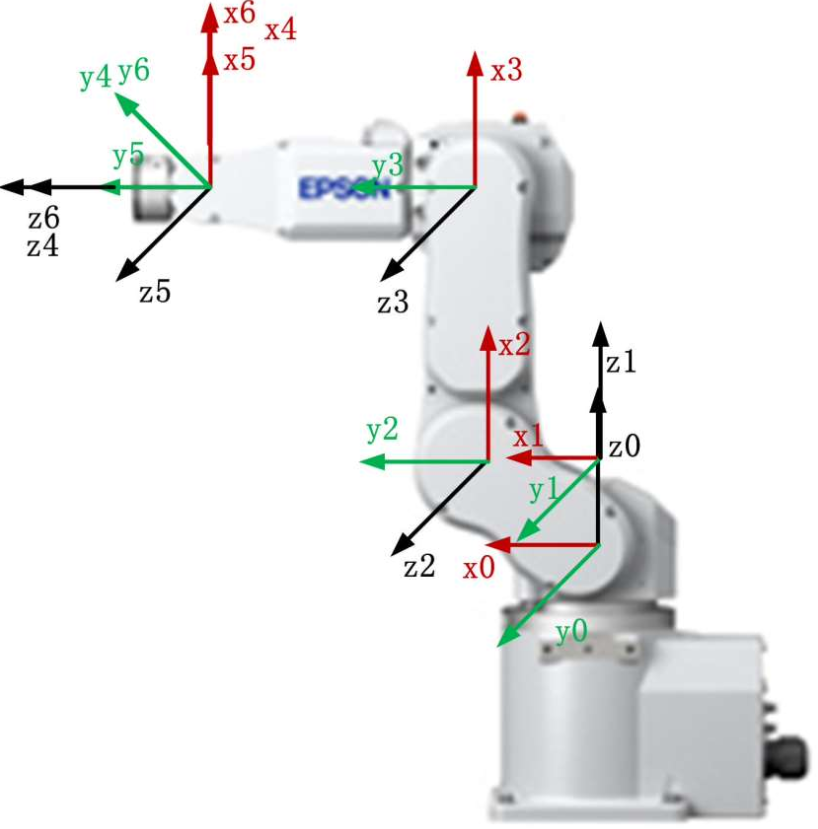
**auto& p3 = model->addPartByPe(link3\_position\_and\_euler321, "321", iv);**

**auto& p4 = model->addPartByPe(link4\_position\_and\_euler321, "321",iv);**

**auto& p5 = model->addPartByPe(link5\_position\_and\_euler321, "321", iv);**

**auto& p6 = model->addPartByPe(link6\_position\_and\_euler321, "321", iv);**

其中linki\_position\_and\_euler321是下图每个杆件的坐标系（可通过DH方法建立坐标系）在基座标系下的位姿；iv是该杆件在其自身局部杆件坐标系下的惯性向量，iv的定义为：[m, m\*x, m\*y, m\*z, Ixx, Iyy, Izz, Ixy, Ixz, Iyz]，其中x,y,z, Ixx, Iyy, Izz, Ixy, Ixz, Iyz为在杆件局部坐标系质心位置和惯性张量



# 添加关节

**直线运动关节**

**auto &joint3 = m.addPrismaticJoint(p3, p2, joint3\_position, joint3\_axis);**

其中p3、p2为上一步骤增加的杆件，joint3\_position是关节的坐标系在基座标系下的位置向量，joint3\_axis是关节轴线在基座标系下的单位向量。

**旋转运动关节**

**auto &joint4 = m.addRevoluteJoint(p4, p3, joint4\_position, joint4\_axis);**

其中p4、p3为上一步骤增加的杆件，joint3\_position是关节的坐标系在基座标系下的位置向量，joint3\_axis是关节轴线在基座标系下的单位向量。

# 添加驱动

**参考程序**

# 添加末端

**方法一：**

**auto &end\_effector = m.addGeneralMotionByPe(link4, m.ground(), end\_effector\_position\_and\_euler321, "321");**

第一个参数表明末端位于link4上，第二个参数表明末端的位姿是相对于地面的，后两个参数定义了末端的起始位姿（在基座标系下）

**方法二：**

**auto& makI = p6.markerPool().add<Marker>("tool0", ee\_j\_pm);**

//在p6杆件（最后一根杆件）增加一个坐标系tool0，ee\_j\_pm是该坐标系相对于//p6杆自身坐标系的旋转矩阵

**auto& makJ = model->ground().markerPool().add<Marker>("base", ee\_j\_pm);**

//在ground（地面基座标系）增加一个坐标系base，ee\_j\_pm是该坐标系相对于

//基坐标系的旋转矩阵

**auto& ee = model->generalMotionPool().add<model::GeneralMotion>("ee", &makI, &makJ, false);**

# 添加求解器

**求解器的添加必须按以下顺序，依次为逆运动学求解器、正运动学求解器、逆动力学求解器和正动力学求解器。如果后续有其他求解器需要加入，须在这些之后增加。**

**auto& inverse\_kinematic = model->solverPool().add<codeit::model::UrInverseKinematicSolver>();**

**auto& forward\_kinematic = model->solverPool().add<codeit::model::ForwardKinematicSolver>();**

**auto& inverse\_dynamic = model->solverPool().add<codeit::model::InverseDynamicSolver>();**

**auto& forward\_dynamic = model->solverPool().add<codeit::model::ForwardDynamicSolver>();**

# 调用求解器

**逆运动学求解器**

参考程序

**逆动力学求解器**

参考程序