5 机器人运动学实验（逆解）

5.1 逆解方法简介

机器人逆运动学问题的求解就是根据已知末端执行器的位姿(位置和姿态)求解相应的关节变量。机器人运动学的难点一般在于如何快速求取运动学逆解。

求机械臂运动学逆解有很多种方法，传统算法包括**几何法**、**解析法** (代数法)、**数值法**，**智能控制算法**包括遗传算法、神经网络等智能方法。

解析法 (代数法)和几何法属于封闭解法，计算速度快，一般都可以找到可能的逆解，但该类方法对机器人结构的限制较大，对于一个 6R 机器人，仅当其几何结构满足 Pieper 准则（机器人的三个相邻关节轴交于一点或三轴线平行）时，采用解析法才可以求得其封闭解。

采用**数值解法**则不受机器人结构的限制，数值解法主要包括消元法、延拓法和迭代计算法三大类。消元法由机器人的非线性运动方程出发，构造相关的多项式方程，通过计算该多项式方程求得其逆解；延拓法通过跟踪解的路径，从一个已知的初始位姿（已知起始点的逆解，也就是初始关节角）出发计算目标位姿的逆解；迭代法则从一个给定的初始值出发，迭代地计算问题的精确解。

基于**智能控制算法**的求解思想是将机器人运动方程转化为一个控制问题来求解，主要包括遗传算法、神经网络算法及各种智能计算方法的组合。

目前也有些研究，比如类人机器人全身关节运动链，采用深度学习算法来达到较优的运动学控制。

几何作图法只适合简单的运动链，智能控制算法形式多样，所以本章的实验以 AUBO-i5 为例，只讲解满足 Pieper 准则的解析法、以及适用于 6 轴以上的数值法中的迭代算法。

5.2 满足 Pieper 准则的封闭解法

对于满足 pieper 准则的六轴工业机器人而言，给定末端姿态，求得的逆解最多为八组。一般教科书上以 PUMA560 机器人为例讲解的解析法求解过程如下：

(1)解 θ_1：

$$(T_0^1)^{-1}T_0^6=(T_0^1)^{-1}T_0^1T_1^2T_2^3T_3^4T_4^5T_5^6=T_1^6 \tag{5-1}$$

$$\begin{bmatrix} C_1 & 0 & -S_1 & aC_1 \\ S_1 & 0 & C_1 & aS_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}=T_1^6 \tag{5-2}$$

联立式 5-2 两边矩阵元素(3,4)(即三行四列元素，下同),有

$$C_1 p_y - S_1 p_x = 0 \tag{5-3}$$

使用三角代换，其中：

$p_x = R\cos(\phi)$，$p_y = R\sin(\phi)$，$\phi = A\tan 2(p_y, p_x)$，$R = \sqrt{p_x^2 + p_y^2}$

解得

$$\theta_1 = A\tan 2(\mathrm{p}_y, \mathrm{p}_x) - A\tan 2(0, \pm 1) \tag{5-4}$$

(2)解 θ_3：

分别联立式 5-2 两边矩阵元素(1,4)，(2,4)，(3,4)得方程组并解得：

$$\theta_3 = A\tan 2(K, \pm\sqrt{a_3^2 + d_4^2 - K^2}) - A\tan 2(a_3, d_4) \tag{5-5}$$

其中

$$K = \frac{p_x^2 + p_y^2 + p_z^2 - 2a_1(p_x C_1 + p_y S_1) - d_4^2 - a_3^2 - a_2^2 + a_1^2}{2a_2} \tag{5-6}$$

同理，由

$$(T_0^3)^{-1} T_0^6 = (T_2^3)^{-1}(T_1^2)^{-1}(T_0^1)^{-1} T_0^1 T_1^2 T_2^3 T_3^4 T_4^5 T_5^6 = T_3^6 \tag{5-7}$$

联立式 5-7 两边矩阵元素(1,4),(3,4),可得

$$\theta_{23} = \theta_2 + \theta_3 = A\tan 2(M, \mathrm{N}) \tag{5-8}$$

其中：

$$M = -(a_2 C_3 + a_3) p_z + (a_2 S_3 + d_4)(\mathrm{C_1 p_x + S_1 p_y - a_1})$$

$$N = (\mathrm{a_2 S_3 + d_4})\,\mathrm{p_z} + (\mathrm{a_2 C_3 + a_3})(\mathrm{C_1 p_x + S_1 p_y - a_1})$$

(3)解 θ_2：

根据 $\theta_2 = \theta_{23} - \theta_3$，由于 θ_1 与 θ_3 都有两个解，因此 θ_2 的值有四种情况。

(4)解 θ_4：

分别联立式 5-7 两边矩阵元素(1,3),(2,3),考虑到 θ_4 的解需要考虑奇异解问题，分两种情况讨论：

①当 $S_5 \neq 0$ 时，$\theta_4 = A\tan 2[(\mathrm{S_1 r_{13} - C_1 r_{23}}), -(\mathrm{C_1 C_{23} r_{13} + } S_1 C_{23} r_{23} - S_{23} r_{33})]$

②当 $S_5 = 0$ 时，关节 4 和关节 6 重合，机器人处于丢失了一个自由度的状态，此时，机器人处于奇异形位，仅能获取 θ_4 和 θ_6 的和或差，是否为奇异形位可根

据式中 $A\tan 2$ 的两个参数是否都趋近于零作为判断依据，若二者都趋近于零可以判断为奇异形位，反之则不是。处于奇异形位时一般采用前一时刻的 θ_4 作为当前 θ_4 的解。

(5)解 θ_5：

由式

$$(T_0^4)^{-1}T_0^6=(T_3^4)^{-1}(T_2^3)^{-1}(T_1^2)^{-1}(T_0^1)^{-1}T_0^1T_1^2T_2^3T_3^4T_4^5T_5^6=T_4^6 \tag{5-9}$$

分别联立式 5-9 两边矩阵元素(1,3),(2,3)解得：

$$\theta_5=A\tan 2(\mathrm{P},\mathrm{Q}) \tag{5-10}$$

其中

$$P=-(\mathrm{C_4C_1C_{23}-S_4S_1})\mathrm{r_{13}}-(\mathrm{C_4S_1C_{23}+S_4C_1})\mathrm{r_{23}}+\mathrm{C_4S_{23}r_{33}}$$
$$Q=C_1S_{23}r_{13}+S_1S_{23}r_{23}+C_{23}r_{33}$$

(6)解 θ_6：

由式

$$(T_0^5)^{-1}T_0^6=(T_4^5)^{-1}(T_3^4)^{-1}(T_2^3)^{-1}(T_1^2)^{-1}(T_0^1)^{-1}T_0^1T_1^2T_2^3T_3^4T_4^5T_5^6=T_5^6 \tag{5-11}$$

分别联立式 5-11 两边矩阵元素(1,1)，(2,1)解得：

$$\theta_6=A\tan 2(\mathrm{S_6},\mathrm{C_6}) \tag{5-12}$$

其中

$$C_6=[\mathrm{C_5(C_1C_4C_{23}-S_1S_4)+C_1S_{23}S_5}]\mathrm{r_{11}}+[\mathrm{C_5(C_4S_1C_{23}+S_4C_1)+S_1S_5S_{23}}]\mathrm{r_{21}}$$
$$+[\mathrm{S_5C_{23}-C_4C_5S_{23}}]\mathrm{r_{31}}$$
$$S_6=[-\mathrm{C_1S_4C_{23}-S_1C_4}]\mathrm{r_{11}}+[\mathrm{C_1C_4-S_1S_4C_{23}}]\mathrm{r_{21}}+S_{23}S_4r_{31}$$

至此，在已知机器人末端位姿的情况下，我们可以求出其封闭逆解，可能存在最多 8 组解，实际运行过程中需要根据机器人参数、碰撞及加工要求等选取一组合适的解。

这个逆解算法对于 AUBO-i5 是适用的，很多工业机器人基本都是用的这个算法步骤求解，参考程序网络上到处都有，但我不打算用这个解法来编程序，原因在于他求解的顺序，没有太多考虑腕部关节的姿态，解出来的 8 组解有可能一组都不能用。我打算参考这个地方的一个逆解算法来编写程序：

（ https://smartech.gatech.edu/bitstream/handle/1853/50782/ur_kin_tech_report_1.pdf）

他是用解析解参乎了球面几何法来解的，过程中都是矩阵运算来讲解，没有拆开矩阵、合并同类项，正好我们也打算用 Eigen3 来写算法，省的写一堆三角函数。

在正解那一章我们学过，如果知道六个关节角，那么：

$$ {}_{6}^{0}T(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = {}_{1}^{0}T(\theta_1)\,{}_{2}^{1}T(\theta_2)\,{}_{3}^{2}T(\theta_3)\,{}_{4}^{3}T(\theta_4)\,{}_{5}^{4}T(\theta_5)\,{}_{6}^{5}T(\theta_6) $$

这些 T 矩阵上一章我已讲过，转换矩阵使用 SDH 的转换矩阵一般形式，可以用我编写的 TransMatrixSDH 函数得到。

1） 先求 θ 1

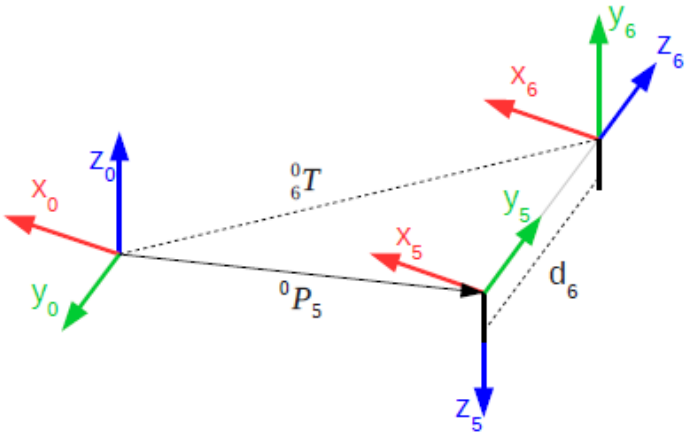


图 5.1 找到第 5 轴手腕（wrist）相对基坐标的位置

如图 5.1 所示，针对 base 坐标系来说（一般我们把 base 定义在全局坐标系重合），第 5 个坐标系的相对{base}来说，这个变换我们可以写成 $^{0}P_{5}$ ，这个 $^{0}P_{5}$ 可以由坐标系{6}沿着 Z6 转换过来。首先我们是已知末端 TCP 位姿的，所以 ${}_{6}^{0}T$ 以及 d_6 已知，那么我们可以先将{5}轴的位置确定下来，那么这个解一定既满足位置要求也满足姿态要求。

$$ {}^{0}P_5 = {}^{0}P_6 - d_6 \cdot {}^{0}\hat{Z}_6 \Leftrightarrow $$

$$ {}^{0}P_5 = {}_{6}^{0}T \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \qquad (5\text{-}13) $$

要求 θ 1，我们从 Z0 往下看，如图 5.2 所示。

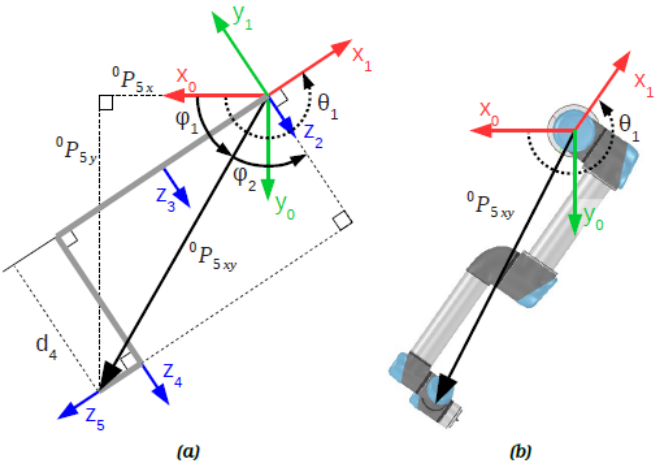


图 5.2 从{0}往{5}往下看它们的几何关系

$$v_{0\to1} = v_{0\to5} - v_{1\to5} \Leftrightarrow$$
$$v_{0\to1} = v_{0\to5} + v_{5\to1} \Leftrightarrow$$
$$\theta_1 = \phi_1 + \left(\phi_2 + \frac{\pi}{2}\right)$$

5-14

θ 1 实际上是{0}->{1}的转换角度，等于{0}->{5}和{1}->{5}之间的差，见公式 5-14。从图 5.2 中我们可知：

$$\phi_1 = \text{atan2}({}^0P_{5y}, {}^0P_{5x})$$

5-15

那么：

$$\cos(\phi_2) = \frac{d_4}{|{}^0P_{5xy}|} \Rightarrow$$
$$\phi_2 = \pm\text{acos}\left(\frac{d_4}{|{}^0P_{5xy}|}\right) \Leftrightarrow$$
$$\phi_2 = \pm\text{acos}\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right)$$

由此，我们可以解出 θ 1:

$$\theta_1 = \phi_1 + \phi_2 + \frac{\pi}{2} \Leftrightarrow$$
$$\theta_1 = \text{atan2}\left({}^0P_{5y}, {}^0P_{5x}\right) \pm \text{acos}\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) + \frac{\pi}{2}$$

5-16

θ 1 有两个解，一个是左手型，一个是右手型。

2）求θ5

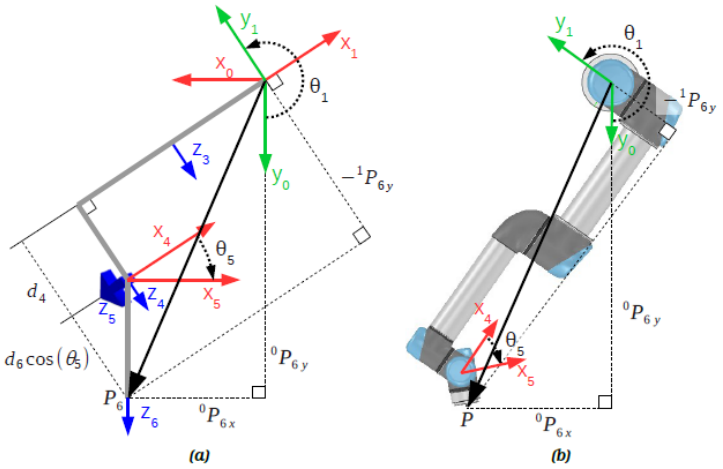


图 5.3 从{0}看到{6}坐标系

从图 5.3 的几何关系中我们可以得到 ${}^1P_{6y}$：

$$-{}^1P_{6y} = d_4 + d_6\cos\theta_5 \tag{5-17}$$

${}^1P_{6y}$ 可以按如下公式求出：

$$\begin{aligned}
{}^0P_6 &= {}^0_1R\cdot{}^1P_6 \Leftrightarrow \\
{}^1P_6 &= {}^0_1R^{\top}\cdot{}^0P_6 \Leftrightarrow \\
\begin{bmatrix}{}^1P_{6x}\\ {}^1P_{6y}\\ {}^1P_{6z}\end{bmatrix} &= \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1) & 0\\ \sin(\theta_1) & \cos(\theta_1) & 0\\ 0 & 0 & 1\end{bmatrix}^{\top}\begin{bmatrix}{}^0P_{6x}\\ {}^0P_{6y}\\ {}^0P_{6z}\end{bmatrix} \Leftrightarrow \\
\begin{bmatrix}{}^1P_{6x}\\ {}^1P_{6y}\\ {}^1P_{6z}\end{bmatrix} &= \begin{bmatrix}\cos(\theta_1) & \sin(\theta_1) & 0\\ -\sin(\theta_1) & \cos(\theta_1) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}{}^0P_{6x}\\ {}^0P_{6y}\\ {}^0P_{6z}\end{bmatrix} \Rightarrow \\
{}^1P_{6y} &= {}^0P_{6x}\cdot(-\sin\theta_1) + {}^0P_{6y}\cdot\cos\theta_1
\end{aligned} \tag{5-18}$$

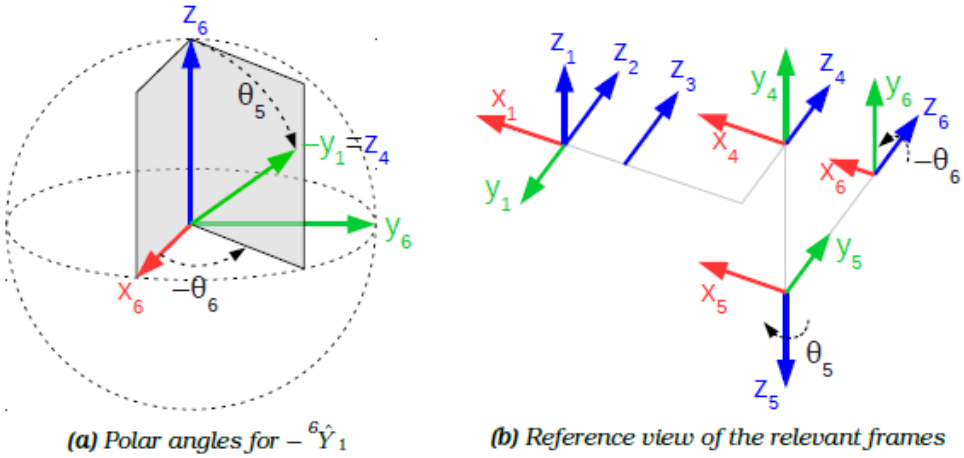
连立式（5-17）（5-18），可以得到θ5：

$$\begin{aligned}
-d_4 - d_6\cos\theta_5 &= {}^0P_{6x}(-\sin\theta_1) + {}^0P_{6y}\cos\theta_1 \Leftrightarrow \\
\cos\theta_5 &= \frac{{}^0P_{6x}\sin\theta_1 - {}^0P_{6y}\cos\theta_1 - d_4}{d_6} \Leftrightarrow \\
\theta_5 &= \pm\mathrm{acos}\left(\frac{{}^0P_{6x}\sin\theta_1 - {}^0P_{6y}\cos\theta_1 - d_4}{d_6}\right)
\end{aligned} \tag{5-19}$$

θ5 有两个解，对应的是{5}坐标系，也就是 wrist 轴是在“上”还是在“下”。无论上下，末端的姿态都将会由 θ6 来纠正。需要注意的是 acos 函数里面计算出来的弧度不能大于 1，也就是：

$$|{}^{1}P_{6y} - d_4| \leq |d_6|.$$

3）求 θ6



(a) Polar angles for $-{}^{6}\hat{Y}_1$ *(b) Reference view of the relevant frames*

图 5.4 轴-${}^{6}\hat{Y}_1$ 在球面坐标系中以方位角-θ6 和极值角 θ5 表示。 为简单起见，图中用 ${}^{6}\hat{Y}_1$ 表示为 y1。

从球面坐标将-${}^{6}\hat{Y}_1$ 转换到笛卡尔坐标系：

$$-{}^{6}\hat{Y}_1 = \begin{bmatrix} \sin\theta_5\cos(-\theta_6) \\ \sin\theta_5\sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix} \Leftrightarrow$$

$${}^{6}\hat{Y}_1 = \begin{bmatrix} -\sin\theta_5\cos\theta_6 \\ \sin\theta_5\sin\theta_6 \\ -\cos\theta_5 \end{bmatrix} \quad 5\text{-}20$$

在（5-20）中，我们单独考虑 θ6， θ6 可由${}^{6}_{1}T$表达，我们需要一个 θ6 从${}^{6}_{0}T$而来的表达式，球坐标 ${}^{6}\hat{Y}_1$ 是在{0}坐标系在 x/y 平面旋转 θ1，那么：

$${}^{6}\hat{Y}_1 = {}^{6}\hat{X}_0 \cdot (-\sin\theta_1) + {}^{6}\hat{Y}_0 \cdot \cos\theta_1 \Leftrightarrow$$

$${}^{6}\hat{Y}_1 = \begin{bmatrix} -{}^{6}\hat{X}_{0x} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0x} \cdot \cos\theta_1 \\ -{}^{6}\hat{X}_{0y} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0y} \cdot \cos\theta_1 \\ -{}^{6}\hat{X}_{0z} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0z} \cdot \cos\theta_1 \end{bmatrix} \quad 5\text{-}21$$

从（5-20）（5-21）的 ${}^{6}\hat{Y}_1$ 前两项有如下等式：

$$\left.\begin{aligned} -\sin\theta_5\cos\theta_6 &= -{}^6\hat{X}_{0x}\cdot\sin\theta_1+{}^6\hat{Y}_{0x}\cdot\cos\theta_1 \\ \sin\theta_5\sin\theta_6 &= -{}^6\hat{X}_{0y}\cdot\sin\theta_1+{}^6\hat{Y}_{0y}\cdot\cos\theta_1 \end{aligned}\right\}\Leftrightarrow$$

$$\left\{\begin{aligned} \cos\theta_6 &= \frac{{}^6\hat{X}_{0x}\cdot\sin\theta_1-{}^6\hat{Y}_{0x}\cdot\cos\theta_1}{\sin\theta_5} \\ \sin\theta_6 &= \frac{-{}^6\hat{X}_{0y}\cdot\sin\theta_1+{}^6\hat{Y}_{0y}\cdot\cos\theta_1}{\sin\theta_5} \end{aligned}\right\}\Rightarrow$$

$$\theta_6=\operatorname{atan2}\left(\frac{-{}^6\hat{X}_{0y}\cdot\sin\theta_1+{}^6\hat{Y}_{0y}\cdot\cos\theta_1}{\sin\theta_5},\frac{{}^6\hat{X}_{0x}\cdot\sin\theta_1-{}^6\hat{Y}_{0x}\cdot\cos\theta_1}{\sin\theta_5}\right) \quad 5\text{-}22$$

解出 θ 6，可以看出 sin θ 5 不能为零，如果为零，就没解，在这种情况下，关节轴 2,3,4 和 6 对齐（如图 5.4(b)所示）， 这表示自由度“太多”了。

4）求 θ 3

检查还剩下的三个关节{2，3，4}，我们发觉这三个关节的旋转轴是平行的，我们可以把它们当作是一个 3R 的平面机械臂。如图 5.5 我们建立坐标系。

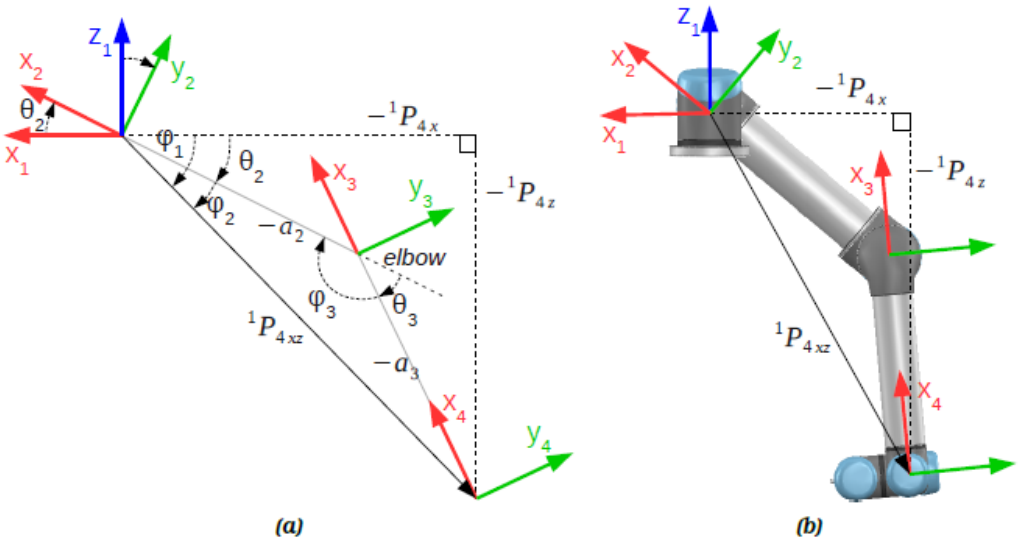


图 5.5 {2，3，4} 轴坐标关系

我们现在只看 ${}^1_4\mathrm{T}$（坐标系{4}相对于坐标系{1}），因为此时 ${}^0_1\mathrm{T}$，${}^4_5\mathrm{T}$和 ${}^5_6\mathrm{T}$是我们已经求出来了。${}^1_4\mathrm{T}$表示的变换在图 5.5a 坐标系{1} 的 x，z 平面中示出。从图中可以清楚地看到长度 $|{}^1P_{4xz}|$ 的值仅由 θ 3 确定，或类似地由 Φ 3 确定。 角度 Φ 3 可以通过余弦定律找到：

$$\cos\phi_3=\frac{(-a_2)^2+(-a_3)^2-|{}^1P_{4xz}|^2}{2(-a_2)(-a_3)}=\frac{{a_2}^2+{a_3}^2-|{}^1P_{4xz}|^2}{2a_2a_3} \quad 5\text{-}23$$

cos Φ 3 和 cos θ 3 的关系如下：

$$\cos\theta_3=\cos(\pi-\phi_3)=-\cos(\phi_3) \quad 5\text{-}24$$

联立方程（5-23），（5-24），即可求出 θ 3:

$$\cos\theta_3 = -\frac{{a_2}^2 + {a_3}^2 - |{}^1P_{4xz}|^2}{2a_2a_3} \Leftrightarrow$$
$$\theta_3 = \pm\text{acos}\left(\frac{|{}^1P_{4xz}|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}\right) \quad 5\text{-}25$$

5）求θ2

θ2=φ1-φ2，从图 5.5a 可以知道：

$$\phi_1 = \text{atan2}(-{}^1P_{4z}, -{}^1P_{4x})$$
$$\frac{\sin\phi_2}{-a_3} = \frac{\sin\phi_3}{|{}^1P_{4xz}|} \Leftrightarrow$$
$$\phi_2 = \text{asin}\left(\frac{-a_3\sin\phi_3}{|{}^1P_{4xz}|}\right)$$

由于 sinφ3=sin(180-θ3)=sinθ3，所以我们可以用θ3 代替φ3，这样θ2 完全求出：

$$\theta_2 = \phi_1 - \phi_2 = \text{atan2}(-{}^1P_{4z}, -{}^1P_{4x}) - \text{asin}\left(\frac{-a_3\sin\theta_3}{|{}^1P_{4xz}|}\right) \quad 5\text{-}26$$

6）求θ4

剩下最后一个θ4，求解非常简单了，为从 X3 到 X4 按 Z4 的旋转角度，可以从剩下的最后一个转换矩阵${}^3_4\text{T}$得出，直接取${}^3\hat{X}_4$的第一列。

$$\theta_4 = \text{atan2}({}^3\hat{X}_{4y}, {}^3\hat{X}_{4x}) \quad 5\text{-}27$$

至此，我们最多可以求出 8 组解：$2_{\theta_1} \times 2_{\theta_5} \times 1_{\theta_6} \times 2_{\theta_3} \times 1_{\theta_2} \times 1_{\theta_4}$

7）编程实现

代码如下：

```
//Makes sure minimal angle is 0
double regularAngle(double theta){
    if (theta < ZERO_THRESH && theta > -ZERO_THRESH){
        theta = 0;
    }
    if(std::isnan(theta))theta = 0;
```

```
    return theta;
}
//SDH general trans matrix
Eigen::Matrix<double,4,4> TransMatrixSDH(double a, double alpha, double d, double theta) {
    Eigen::Matrix<double,4,4> T;
    T.row(0)<<cos(theta),-sin(theta)*cos(alpha), sin(theta)*sin(alpha),a*cos(theta);
    T.row(1)<<sin(theta),cos(theta)*cos(alpha),-cos(theta)*sin(alpha),a*sin(theta);
    T.row(2)<<0,sin(alpha),cos(alpha),d;
    T.row(3)<<0,0,0,1;
    return T;
}

//Inverse Kinematics- Mixed method
int InverseKinematics(Eigen::Matrix<double,4,4> TcpPOS, double **q, double q6Des){
    int numSols = 0;
    double q1[2];
    double q5[2][2];
    double q6[2][2];
    double q3[2][2][2];
    double q2[2][2][2];
    double q4[2][2][2];

    //=========================================
    //            Solving for shoulder_joint q1
    //=========================================

    //the location of the fifth coordinate frame wrt the base:
    Eigen::Vector4d P0_5 = TcpPOS * Eigen::Vector4d(0, 0, -d6, 1) - Eigen::Vector4d(0, 0, 0, 1);
    double psi = atan2(P0_5[1], P0_5[0]);
    std::cout<<"psi:"<<psi<<std::endl;
    double phi_pos;
```

```
    if((fabs(d4 / (sqrt(pow(P0_5[0],2) + pow(P0_5[1],2))))>=1)&&(fabs(d4 / (sqrt(pow(P0_5[0],2) + pow(P0_5[1],2))))<(1+ZERO_THRESH))){phi_pos=0;}
    else
    phi_pos = acos(d4 / (sqrt(pow(P0_5[0],2) + pow(P0_5[1],2))));

    std::cout<<"d4/sqrt value:"<<d4/(sqrt(pow(P0_5[0],2) + pow(P0_5[1],2)))<<std::endl;
    std::cout<<"phi_pos:"<<phi_pos<<std::endl;

    q1[0] = regularAngle(psi + phi_pos + M_PI/2);
    q1[1] = regularAngle(psi - phi_pos + M_PI/2);
        std::cout<<"q1[0]:"<<psi + phi_pos + M_PI/2<<" q1[1]:"<<psi - phi_pos + M_PI/2<<std::endl;

    //=========================================
    //          Solving for wrist2_joint q5
    //=========================================
    //the location of the sixth coordinate frame wrt the base:
    //Eigen::Vector4d P0_6 = TcpPOS * Eigen::Vector4d(0, 0, 0, 1) - Eigen::Vector4d(0, 0, 0, 1);
    //For each solution to q1
    for (int i = 0; i < 2; i++)
    {
        //find transformation from frame 6 to frame 1
        Eigen::Matrix<double,4,4> T1_0 = (TransMatrixSDH(a1, alpha1, d1, q1[i])).inverse();
        Eigen::Matrix<double,4,4> T1_6 = T1_0* TcpPOS;

        //the z location of the 6th frame wrt the 1st frame
        //double P1_6z = P0_6[0]*sin(q1[i]) - P0_6[1]*cos(q1[i]);
            double P1_6z = T1_6(2,3);
        double temp_trig_ratio = (P1_6z - d4) / d6;
        std::cout<<"temp_trig_ratio:"<<temp_trig_ratio<<std::endl;
        //invalid if the argument to acos is not in [-1, 1]:
```

```
    if((fabs(temp_trig_ratio)>=1)&&(fabs(temp_trig_ratio)<(1+ZERO_THRESH))){tem
p_trig_ratio=1.0;}

        if (fabs(temp_trig_ratio) > 1.0) continue;

        q5[i][0] = regularAngle(acos(temp_trig_ratio));
        q5[i][1] = regularAngle(-acos(temp_trig_ratio));
            std::cout<<"q5["<<i<<"][0]:"<<q5[i][0]<<"
q5["<<i<<"][1]:"<<q5[i][1]<<std::endl;
        //=======================================
        //          Solving for q6
        //=======================================


        Eigen::Matrix<double,4,4> T0_1 = TransMatrixSDH(a1, alpha1, d1, q1[i]);
        Eigen::Matrix<double,4,4> T6_1 = TcpPOS.inverse()*T0_1;

        //For each solution to q5
        for(int j = 0; j < 2; j++)
        {
            double sin_q5 = sin(q5[i][j]);
            double z6_1_y = T6_1(1,2);
            double z6_1_x = T6_1(0,2);

            //invalid if sin(q5)=0 or z_x and z_y both =0, in this case q6 is free
            if ( (sin_q5 < ZERO_THRESH && sin_q5 > -ZERO_THRESH) ||
                (   (z6_1_x < ZERO_THRESH && z6_1_x > -ZERO_THRESH) &&
                (z6_1_y < ZERO_THRESH && z6_1_y > -ZERO_THRESH)   )   ){
                q6[i][j] = 0; //choose arbitrary q6
            } else {
            q6[i][j] = regularAngle(atan2( -z6_1_y / sin_q5, z6_1_x / sin_q5 ));
            }
                std::cout<<"q6[i][j]"<<q6[i][j]<<std::endl;
            /*=======================================
            /          Solving for q2-q4
```

```
            /=========================================*/
            //find location of frame 3 wrt frame 1
            Eigen::Matrix<double,4,4> T5_4 = (TransMatrixSDH(a5, alpha5, d5, q5[i][j])).inverse();
            Eigen::Matrix<double,4,4> T6_5 = (TransMatrixSDH(a6, alpha6, d6, q6[i][j])).inverse();
            Eigen::Matrix<double,4,4> T1_4 = T1_0 * TcpPOS * T6_5 * T5_4;
            Eigen::Vector4d P1_3 = T1_4 * Eigen::Vector4d(0, -d4, 0, 1) - Eigen::Vector4d(0,0,0,1);

            //solve for q3 first
            temp_trig_ratio = (pow(P1_3[0],2) + pow(P1_3[1],2) - pow(a2, 2) - pow(a3, 2)) / (2 * a2 * a3);


    if((fabs(temp_trig_ratio)>=1)&&(fabs(temp_trig_ratio)<(1+ZERO_THRESH))){temp_trig_ratio=1.0;}

            //invalid if the argument to acos is not in [-1, 1]:
            if (fabs(temp_trig_ratio) > 1) continue;

            double theta3 = acos(temp_trig_ratio);
            q3[i][j][0] = regularAngle(theta3);
            q3[i][j][1] = regularAngle(-theta3);
            std::cout<<"q3[i][j]"<<q3[i][j][0]<<" "<<q3[i][j][1]<<std::endl;
            //For each solution to q3
            for(int k = 0; k < 2; k++)
            {
                //solve for q2
                q2[i][j][k] = regularAngle(-atan2(P1_3(1), -P1_3(0)) + asin(a3 * sin(q3[i][j][k]) / sqrt(pow(P1_3[0],2) + pow(P1_3[1],2))));
                //find transformation from frame 3 to 4
                Eigen::Matrix<double,4,4> T1_2 = TransMatrixSDH(a2, alpha2, d2, q2[i][j][k]);
                Eigen::Matrix<double,4,4> T2_3 = TransMatrixSDH(a3, alpha3, d3,
```

```
q3[i][j][k]);
                    Eigen::Matrix<double,4,4> T3_4 = (T1_2 * T2_3).inverse() * T1_4;
                    //extract q4 from it
                    q4[i][j][k] = regularAngle(atan2(T3_4(1,0), T3_4(0,0)));
                    std::cout<<"q4[i][j][k]"<<q4[i][j][k]<<std::endl;
                    //write joint angles to solution buffer
                    q[numSols][0] = q1[i];
                    q[numSols][1] = q2[i][j][k];
                    q[numSols][2] = q3[i][j][k];
                    q[numSols][3] = q4[i][j][k];
                    q[numSols][4] = q5[i][j];
                    q[numSols][5] = q6[i][j];
                    numSols++;
                }
            }

        }


    return numSols;
}
```

运动学逆解编写完了，我们在主程序里写个测试例程：

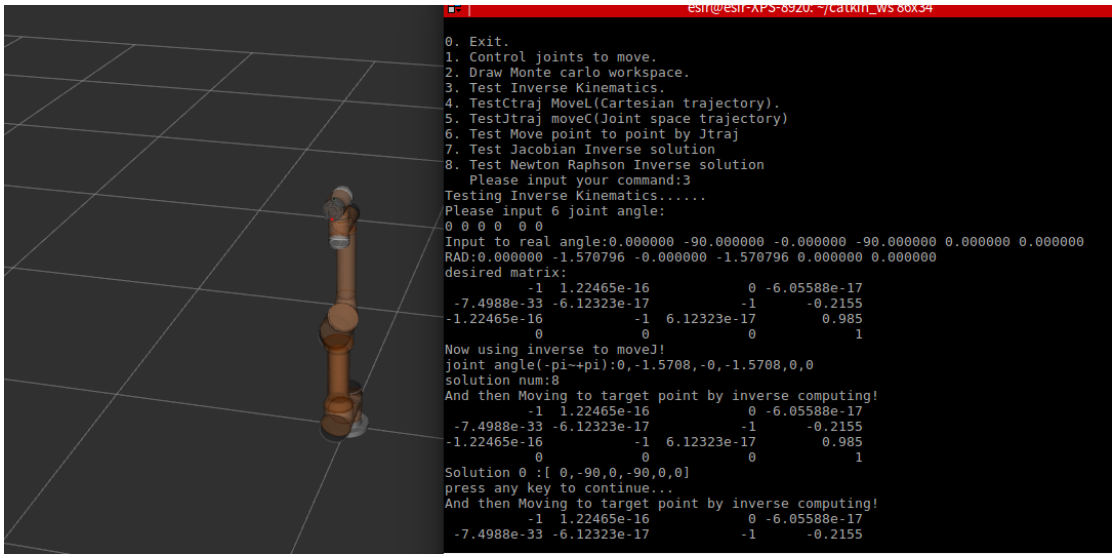
```
        printf("Testing Inverse Kinematics.....\n");
        printf("Please input 6 joint angle:\n");
        scanf("%lf%lf%lf%lf%lf%lf",&j0,&j1,&j2,&j3,&j4,&j5);
        q0 = j0*M_PI/180.0;
        q1 = j1*M_PI/180.0;
        q2 = j2*M_PI/180.0;
        q3 = j3*M_PI/180.0;
        q4 = j4*M_PI/180.0;
        q5 = j5*M_PI/180.0;
        printf("%lf %lf %lf %lf %lf %lf\n", q0,q1,q2,q3,q4,q5);

        //calculate forward kinematics
        TcpCenter=ForwardKinematics(q0,q1-M_PI/2.0,-q2,q3-M_PI/2.0,q4,q5);
```

```
pointX=TcpCenter(0,3);
pointY=TcpCenter(1,3);
pointZ=TcpCenter(2,3);
printf("%lf %lf %lf\n",pointX,pointY,pointZ);
points.header.stamp = ros::Time::now();
points.action = visualization_msgs::Marker::DELETE;//delete not work
points.points.clear();//work to clear

marker_pub.publish(points);

//draw tcp marker point
points.action = visualization_msgs::Marker::ADD;
points.lifetime=ros::Duration();
points.scale.x = 0.01f;
points.scale.y = 0.01f;
points.scale.z = 0.01f;
points.color.r = 1.0f;
points.color.g = 0.0f;
points.color.b = 0.0f;
points.color.a = 1.0;
DisplayPoint(pointX,pointY,pointZ);

printf("Now using inverse to moveJ!\n");

q0 = q0;
q1 = q1-M_PI/2.0;
q2 = -q2;
q3 = q3-M_PI/2.0;
q4 = q4;
q5 = q5;

std::cout << "joint angle(-pi~+pi):"<<q0<<","<<q1<<","<<q2<<","<<q3<<","<<q4<<","<<q5<<std::endl;
//calculate Inverse solution
```

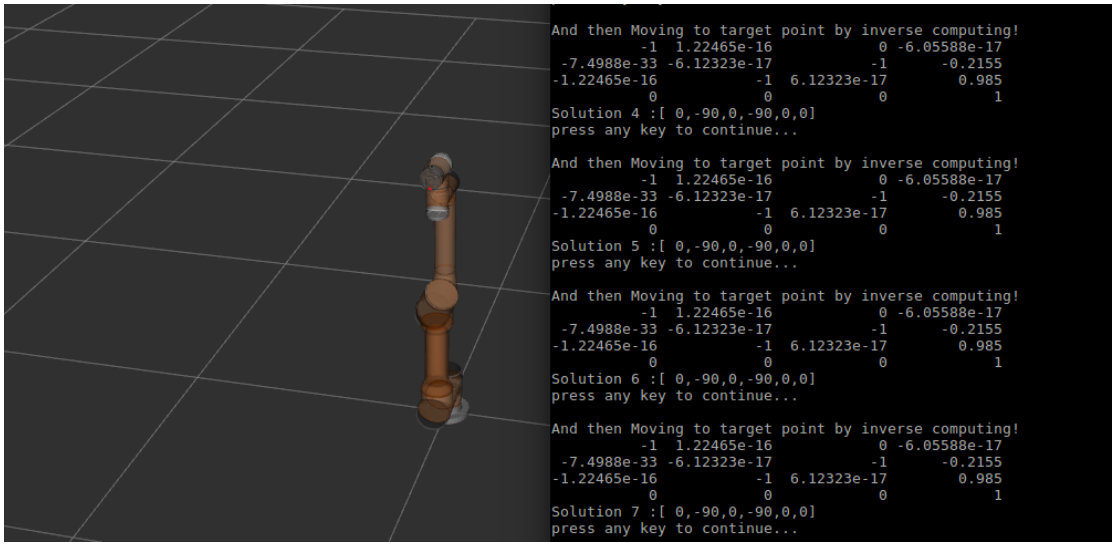
```
            //make solution buffer
            for(i = 0; i < 8; i++)
            {
                q_sol[i] = new double[6];
            }
            num_sol = InverseKinematics(ForwardKinematics(q0, q1, q2, q3, q4, q5),
q_sol, q6_des);
            std::cout << "solution num:"<<num_sol<<std::endl;
            if(num_sol == 0) std::cout << "Sorry,compute no solution! "<<std::endl;
            for(i=0;i<num_sol;i++){
                //printf("Moving to zero position first!\n");
                moveJ(0,0,0,0,0,0);
                printf("And then Moving to target point by inverse computing!\n");


    moveJ(q_sol[i][0],q_sol[i][1]+M_PI/2.0,-q_sol[i][2],q_sol[i][3]+M_PI/2.0,q_sol[i][
4],q_sol[i][5]);

                std::cout << "press any key to continue..."<<std::endl;
                getchar();

                std::cout << "Solution "<<i<<" :[ "<< RadtoAngle(q_sol[i][0]) <<","<<
RadtoAngle(q_sol[i][1]) <<","<< RadtoAngle(q_sol[i][2])
                  <<","<<  RadtoAngle(q_sol[i][3]  )  <<","<<  RadtoAngle(q_sol[i][4])
<<","<< RadtoAngle(q_sol[i][5]) << "]" << std::endl;
            }
```

该测试例程为了验证逆解的正确性，在输入 6 个关节角后，用正解得到末端姿态矩阵，然后再用逆解解算如果要到达这个坐标点，我们的逆解解出来的 6 个角度会是什么样的关系，通过测试，我们可以看出，这个逆解非常准确，位置、姿态都分毫不差，求解也非常快速。

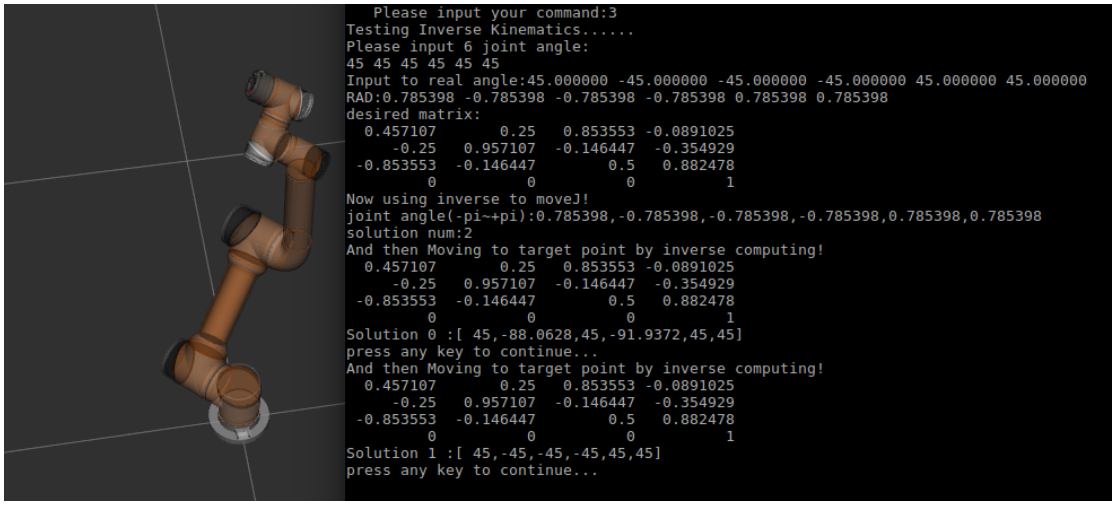
这个代码有部分是从网络借鉴过来的，我做了很多改动，经过大量测试，证明是比较好的，从图 5.6(a)(b)(c)中可以看到，最多 8 组解，每组解都和输入的正解姿态、位置分毫不差，逆解不但正确，精度、求解速度也非常好，但测试并不严格。



(a)



(b)



(c)

图 5.6 解析法封闭解测试，最多 8 个解姿态矩阵完全一致

5.3 牛顿迭代法(jacobian 矩阵迭代)

我们在讲义里提到过机械臂的微分运动学，其中最基本的矩阵就是 jacobian 矩阵，对于我们要实现的机械臂控制末端控制而言（对应笛卡尔空间坐标系），其基础就是对关节的转动角（位移）、速度、加速度进行控制。我们介绍过关节空间中的角速度、加速度和笛卡尔空间的速度、加速度是一个线性关系，其中我们详细介绍过位置、角度的雅可比矩阵，这个矩阵的求法在 peter coker 的 robotics toolbox 里面也有，这里我给出我的 C++实现代码如下：

```
Eigen::Matrix<double,6,6> jacob0(Eigen::Matrix<double,1,6> q){
    Eigen::MatrixXd Jv(3, 6);
    Eigen::MatrixXd Jw(3, 6);

    Eigen::Vector4d o_0(0,0,0,1);
    Eigen::Vector4d o_6 = ForwardKinematics(q[0],q[1],q[2],q[3],q[4],q[5]) * o_0;
    //The origin of frame i:
    Eigen::Vector4d o_i;

    Eigen::Vector3d e3(0,0,1);
    //The z-axis of frame i:
    Eigen::Vector3d z_i;

    //The transformation from frame 0 to frame i:
    Eigen::Matrix4d T = Eigen::Matrix4d::Identity();

    double alpha[6],a[6],d[6];
    alpha[0]=alpha0;alpha[1]=alpha1;alpha[2]=alpha2;alpha[3]=alpha3;alpha[4]=alpha4;alpha[5]=alpha5;alpha[5]=alpha6;
    a[0]=a0;a[1]=a1;a[2]=a2;a[3]=a3;a[4]=a4;a[5]=a5;a[6]=a6;
    d[0]=d1;d[1]=d2;d[2]=d3;d[3]=d4;d[4]=d5;d[5]=d6;

    for(int i = 0; i < 6; i++){
        o_i = T * o_0;
        z_i = T.block<3,3>(0,0) * e3;

        //build Jv one column at a time:
        Jv.block<3,1>(0,i) = z_i.cross((o_6 - o_i).head<3>());
```

```
        //build Jw one column at a time:
        Jw.block<3,1>(0,i) = z_i;

        T *= TransMatrixSDH(a[i+1], alpha[i+1], d[i], q[i]);

    }

    //Put Jv and Jw together
    Eigen::MatrixXd J(6,6);
    J.block<3,6>(0,0) = Jv;
    J.block<3,6>(3,0) = Jw;

    return J;
}
```

这个 jacob0 函数基本就是我从 peter coker 的 matlab 代码翻译过来，所以如果觉得注视写的不够多，看不清楚，请参考 robotics toolbox 里面的说明。

Peter coker 对于大于等于 6 关节的机械臂的逆解提供了一个 ikine.m 文件，采用的 Jacobian 迭代法，也就是我们常说的牛顿迭代法。我这里直接把他的代码简化翻译成 c++代码。

```
% If the manipulator has fewer than 6 DOF then this method of solution
% will fail, since the solution space has more dimensions than can
% be spanned by the manipulator joint coordinates.  In such a case
% it is necessary to provide a mask matrix, M, which specifies the
% Cartesian DOF (in the wrist coordinate frame) that will be ignored
% in reaching a solution.  The mask matrix has six elements that
% correspond to translation in X, Y and Z, and rotation about X, Y and
% Z respectively.  The value should be 0 (for ignore) or 1.  The number
% of non-zero elements should equal the number of manipulator DOF.
%
% Solution is computed iteratively using the pseudo-inverse of the
% manipulator Jacobian.
%
% Such a solution is completely general, though much less efficient
% than specific inverse kinematic solutions derived symbolically.
%
% This approach allows a solution to obtained at a singularity, but
% the joint angles within the null space are arbitrarily assigned.
%
% For instance with a typical 5 DOF manipulator one would ignore
% rotation about the wrist axis, that is, M = [1 1 1 1 1 0].
%
%
% See also: FKINE, TR2DIFF, JACOB0, IKINE560.

% Copyright (C) 1993-2008, by Peter I. Corke
%
% This file is part of The Robotics Toolbox for Matlab (RTB).
%
% RTB is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
```

图 5.7 peter coker 的 ikine.m 代码说明截图

```
Eigen::Matrix<double,1,6> tr2diff (Eigen::Matrix<double,4,4> T1, Eigen::Matrix<double,4,4> T2){
    Eigen::Matrix<double,1,6> d;
    d = Eigen::MatrixXd::Identity(1,6);
    Eigen::Vector3d  tmp,tmp1,tmp2;

    tmp1 << T1(0,0),T1(1,0),T1(2,0);
    tmp2 << T2(0,0),T2(1,0),T2(2,0);
    tmp=tmp1.cross(tmp2);

    tmp1 << T1(0,1),T1(1,1),T1(2,1);
    tmp2 << T2(0,1),T2(1,1),T2(2,1);
    tmp+=tmp1.cross(tmp2);

    tmp1 << T1(0,2),T1(1,2),T1(2,2);
    tmp2 << T2(0,2),T2(1,2),T2(2,2);
    tmp += tmp1.cross(tmp2);

    tmp = 0.5*tmp;

    d << T2(0,3)-T1(0,3), T2(1,3)-T1(1,3),T2(2,3)-T1(2,3), tmp(0,0),tmp(1,0),tmp(2,0);
    //std::cout<<"d:"<<d<<std::endl;
    return d;
}
```

迭代测试，最大迭代次数 50000 次，迭代终止误差设置 1e-16。

```
Eigen::Matrix<double,1,6> jacob_Iteration(Eigen::Matrix<double,4,4> Tobj, Eigen::Matrix<double,1,6> q){
    int ilimit = 50000;
    double stol = 1e-16;
    //single xform case
    double nm = 1.0;
    int count = 0;
    Eigen::Matrix<double,1,6> e;
    Eigen::Matrix<double,6,1>  m;
```

```
    Eigen::Matrix<double,1,6> dq;
    dq = Eigen::MatrixXd::Zero(1,6);
    m(0,0)=1;m(1,0)=1;m(2,0)=1;m(3,0)=1;m(4,0)=1;m(5,0)=1;

    Eigen::MatrixXd T(6,6) , pInv(6,6);
    Eigen::Matrix<double,1,6> qt;

    while(nm>stol){
        e = tr2diff(ForwardKinematics(q[0],q[1],q[2],q[3],q[4],q[5]),Tobj);
        T = jacob0(q);
        pInv = T.completeOrthogonalDecomposition().pseudoInverse();
        dq = pInv*e.transpose();
        q = q+dq;
        nm=dq.norm();
        count = count + 1;
        if(count>ilimit){
            std::cout<<"Error, solution wouldn't converge"<<std::endl;
            break;
        }

    }
    std::cout<<"count:"<<count<<std::endl;
    qt = q;
    return qt;
}
```

很简单的我们来写一个测试例程，来看看迭代效果如何。

测试代码如下：

```
void test_jacob_Iteration(){
    double pointX,pointY,pointZ, angle;
    Eigen::MatrixXd TcpTarget;
    Eigen::Matrix<double,1,6> qq;

    Eigen::Matrix<double,1,6> theta;
    double q0,q1,q2,q3,q4,q5;
    printf("Please input 6 joint angle:\n");
```

```
scanf("%lf%lf%lf%lf%lf%lf",&q0,&q1,&q2,&q3,&q4,&q5);
q0 = q0*M_PI/180.0;
q1 = q1*M_PI/180.0;
q2 = q2*M_PI/180.0;
q3 = q3*M_PI/180.0;
q4 = q4*M_PI/180.0;
q5 = q5*M_PI/180.0;

printf("%lf %lf %lf %lf %lf %lf\n", q0,q1,q2,q3,q4,q5);

TcpTarget=ForwardKinematics(q0,q1-M_PI/2.0,-q2,q3-M_PI/2.0,q4,q5);

//draw tcp marker point
points.action = visualization_msgs::Marker::ADD;
points.lifetime=ros::Duration();
points.scale.x = 0.01f;
points.scale.y = 0.01f;
points.scale.z = 0.01f;
points.color.r = 1.0f;
points.color.g = 0.0f;
points.color.b = 0.0f;
points.color.a = 1.0;
pointX = TcpTarget(0,3);
pointY = TcpTarget(1,3);
pointZ = TcpTarget(2,3);

DisplayPoint(pointX,pointY,pointZ);

std::cout<<"desired Tcp:\n"<<TcpTarget<<std::endl;

theta(0,0) = 0.0; theta(0,1)=0.0; theta(0,2)=0.0;theta(0,3)=0.0;
theta(0,4)=0.0; theta(0,5)=0.0;

printf("Begin jacob_Iteration!\n");
qq = jacob_Iteration(TcpTarget,theta);
```

```
        std::cout<<"get desired rad:"<<qq<<std::endl;
        std::cout<<"forward
matrix:\n"<<Exp_twist(T0,qq(0,0))*Exp_twist(T1,qq(0,1))*Exp_twist(T2,qq(0,2))*Exp
_twist(T3,qq(0,3))*Exp_twist(T4,qq(0,4))*Exp_twist(T5,qq(0,5))*gz<<std::endl;
        std::cout<<"last                                                    get
matrix:\n"<<ForwardKinematics(limit2pi(qq(0,0)),limit2pi(qq(0,1)),
limit2pi(qq(0,2)),limit2pi(qq(0,3)),limit2pi(qq(0,4)),limit2pi(qq(0,5)))<<std::endl;
        printf("And then Moving to target point by inverse computing!\n");
        moveJ(limit2pi(qq(0,0)),limit2pi(qq(0,1))+M_PI/2,
-limit2pi(qq(0,2)),limit2pi(qq(0,3))+M_PI/2 ,limit2pi(qq(0,4)),limit2pi(qq(0,5)));
    }
```

测试代码还是先输入 6 个关节角，我们先用前面已经验证过的正解函数，解除末端姿态矩阵，把这个矩阵作为期望矩阵带入迭代函数，初始矩阵为零位。我们来看看测试情况。

第一次我们输入的 6 个关节角为{0,90,0,90,0,0}，这个实际只是我们方便绘图、计算 DH 参数的时候定义的机械臂的零位，和我们在 rviz 以及实际机械臂的零位设置有差异，所以它对应的实际就是{0，0，0，0，0，0}初始位。如图 5.8 所示，可以看到，这个迭代计算根本就不需要迭代，即可得到逆解，因为它离初始迭代矩阵的差为零。

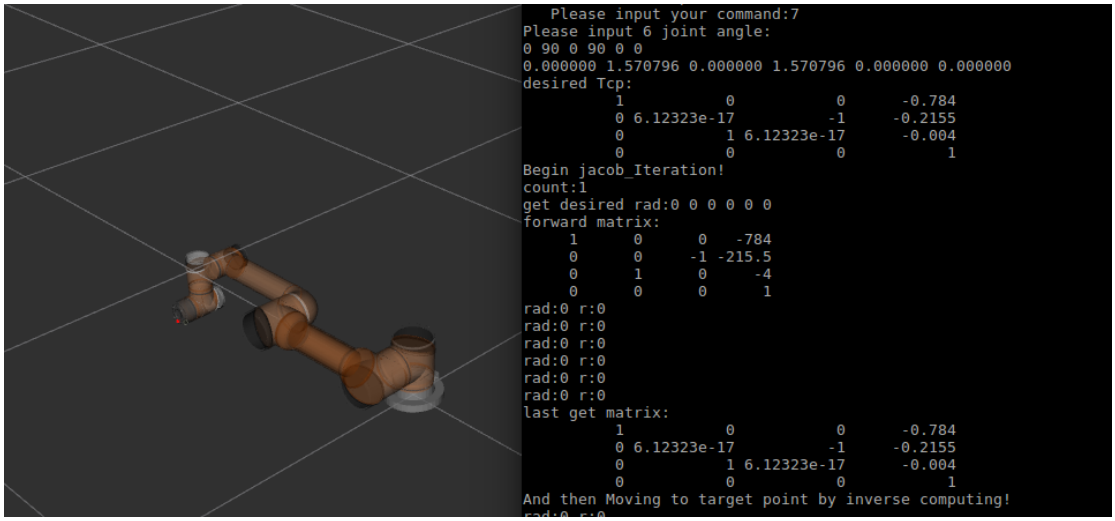


图 5.8 迭代测试一

接下来我们再测试一下{45，45 ，45 ，45 ，45 ，45}这 6 个关节角计算出来的目标姿态矩阵，看是否可以得到精确的解。测试结果如图 5.9 所示。

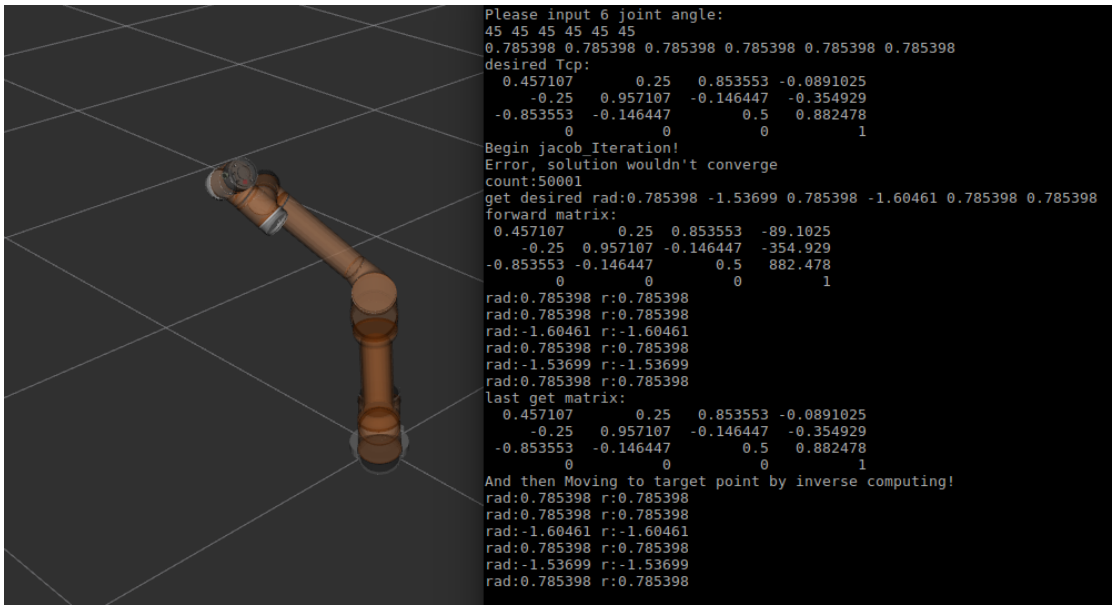


图 5.9 {45，45 ，45 ，45 ，45 ，45}的迭代结果

从图 5.9 中可以看到，程序迭代超出了最大次数，但迭代结果早已经得到了较为精确的位置，double 数据类型精确到小数点后面 6 位，姿态矩阵也完全一致。

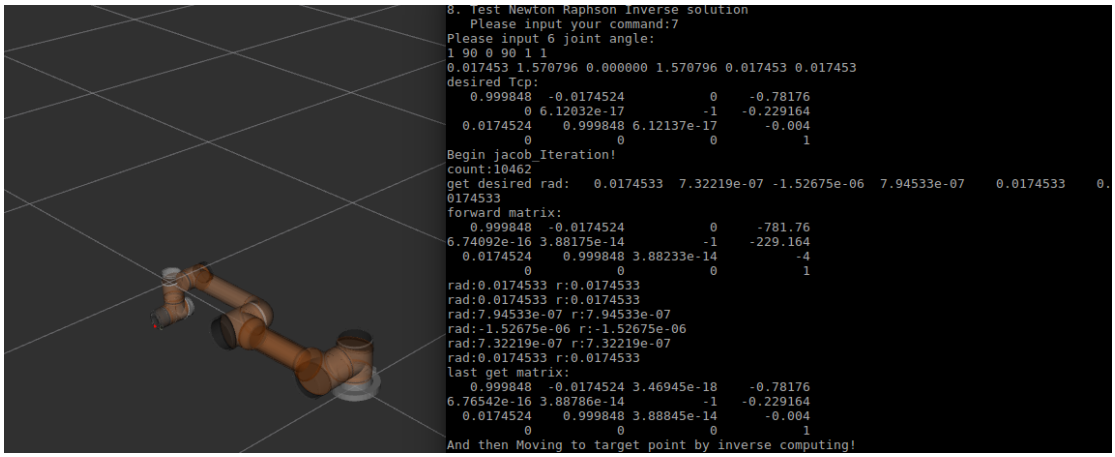


图 5.10 {1,90,0,90,1,1}的迭代结果

从图 5.10 中可以看到迭代角离我们定义的初始迭代位姿比较近，迭代误差设置 1e-16 情况下，迭代了 10461 次，得到很高的精度，位置、姿态都非常精确，只是迭代时间稍长一点而已，由于我们课程的基本实验，所以这里我不讨论迭代次数以及迭代时间的耗费，大家后面可以根据这个基本算法自行去采用梯度下降等优化算法去改善这个算法。

5.4 Newton-Raphson 迭代法

上面我们介绍的是最基本的 Jacobian 迭代法，这里我们再做一个结合我们前面讲的旋量法的经典迭代算法，那就是 Newton-Raphson 迭代算法。

Newton-Raphson 迭代算法（参见 Kevin M. Lynch and Frank C. Park. Modern Robotics: Mechanics, Planning, and Control May 3, 2017. http://modernrobotics.org），

Newton-Raphson 迭代法，我在这本网络预印刷的教材里看到描述还算比较全，其在关节空间的迭代算法步骤如下：

(1) 赋迭代初始角

$$\theta^{(k)} \leftarrow \theta^{(0)} \in \mathbb{R}^n$$

我们程序里用的 theta={0,0,0,0,0,0}

(2) 评估当前角度

$$\psi(\theta^{(k)}) = \left(\log\left(e^{\hat{\xi}_1\theta_1^{(k)}} e^{\hat{\xi}_2\theta_2^{(k)}} \cdots e^{\hat{\xi}_n\theta_n^{(k)}} g_{st}(0) g_d^{-1}\right)\right)^{\vee}$$

```
Vee(logm(Exp_twist(T1,theta(1,1))..*Exp_twist(T6,theta(6,1))*gz*(pinv(gd))));
```

(3) 计算 Jacobian 矩阵

$$J(\theta) = \left[\xi_1 \quad \mathrm{Ad}_{e^{\hat{\xi}_1\theta_1}}\xi_2 \quad \cdots \quad \mathrm{Ad}_{e^{\hat{\xi}_1\theta_1}\cdots e^{\hat{\xi}_{n-1}\theta_{n-1}}}\xi_n\right]$$

这个 Jacobian 矩阵我们打算用前面介绍的旋量法介绍运动旋量 twist 来求。

(4) 计算 Jacobian 矩阵伪逆

$$J^{\#}(\theta^{(k)}) = \left(J^T(\theta^{(k)})J(\theta^{(k)})\right)^{-1} J^T(\theta^{(k)}) \in \mathbb{R}^{n\times 6}$$

(5) 按运动旋量 twist 更新旋转角，并求迭代误差 error 的值用于评估迭代是否终止。

$$\theta^{(k+1)} = \theta^{(k)} - J^{\#}(\theta^{(k)})\psi(\theta^{(k)})$$

```
error = abs(norm(Psi(T1,T2,T3,T4,T5,T6,theta,gz,gd)));
```

(6) 重复以上步骤，直至终止迭代（迭代收敛）。

按照这个算法步骤，我们编写 Newton-Raphson 迭代算法函数：

```
Eigen::Matrix<double,1,6>  Newton_Raphson(Eigen::Matrix<double,6,6>  T,
Eigen::Matrix<double,1,6> theta, Eigen::Matrix4d gz, Eigen::Matrix4d gd){
    int ilimit = 50000;
    int count = 0;
    double err = 10.0;
    double stol = 1e-16;
    Eigen::MatrixXd J(6,6);
```

```
    Eigen::MatrixXd J35(6,6);
    Eigen::Matrix<double,1,6> delta_theta;
    double e;

    std::cout<<"initial theta:\n"<<theta<<std::endl;

    while(err>stol){
        J = jacbobian(T,theta);
        Eigen::CompleteOrthogonalDecomposition<Eigen::MatrixXd> cqr(J);
        J35 = cqr.pseudoInverse();//using Eigen3 newest head library
        delta_theta = J35*Psi(T,theta,gz,gd);
        err = fabs(norm(Psi(T,theta,gz,gd)));
        theta = theta-delta_theta;

        count = count + 1;

        if(count>ilimit){
            std::cout<<"Error, solution wouldn't converge"<<std::endl;
            break;
        }
    }

    std::cout<<"Iteration num:\n";
    std::cout<<count<<std::endl;
    return theta;
}
```

采用运动旋量 twist 构成的 Jacobian 矩阵函数：

```
Eigen::Matrix<double,6,6>  jacbobian(Eigen::Matrix<double,6,6>  T,
Eigen::Matrix<double,1,6> theta){
    Eigen::MatrixXd R(6,6);
    Eigen::Matrix<double,1,6> T0,T1,T2,T3,T4,T5;
    double theta0,theta1,theta2,theta3,theta4,theta5;
    //Eigen::Matrix<double,1,6> temp;

    theta0 = theta(0);
```

```
    theta1 = theta(1);
    theta2 = theta(2);
    theta3 = theta(3);
    theta4 = theta(4);
    theta5 = theta(5);


    T0.block<1,6>(0,0)=T.block<1,6>(0,0);
    T1.block<1,6>(0,0)=T.block<1,6>(1,0);
    T2.block<1,6>(0,0)=T.block<1,6>(2,0);
    T3.block<1,6>(0,0)=T.block<1,6>(3,0);
    T4.block<1,6>(0,0)=T.block<1,6>(4,0);
    T5.block<1,6>(0,0)=T.block<1,6>(5,0);


    R.block<6,1>(0,0) = T0.transpose(); //column 0
    R.block<6,1>(0,1)  =  (adj(Exp_twist(T0,theta0))*T1.transpose()).transpose();
//column 1
    R.block<6,1>(0,2)                                                    =
(adj(Exp_twist(T0,theta0))*adj(Exp_twist(T1,theta1))*T2.transpose()).transpose();
//column 2
    R.block<6,1>(0,3)                                                    =
(adj(Exp_twist(T0,theta0))*adj(Exp_twist(T1,theta1))*adj(Exp_twist(T2,theta2))*
T3.transpose()).transpose(); //column 3
    R.block<6,1>(0,4)                                                    =
(adj(Exp_twist(T0,theta0))*adj(Exp_twist(T1,theta1))*adj(Exp_twist(T2,theta2))*
adj(Exp_twist(T3,theta3))*T4.transpose()).transpose(); //column 4
    R.block<6,1>(0,5)                                                    =
(adj(Exp_twist(T0,theta0))*adj(Exp_twist(T1,theta1))*adj(Exp_twist(T2,theta2))*
adj(Exp_twist(T3,theta3))*adj(Exp_twist(T4,theta4))*T5.transpose()).transpose();
//column 5

    return R;
}
```

adj 函数：

```
//input x SE(3)
Eigen::Matrix<double,6,6> adj(Eigen::Matrix4d X){
    Eigen::Matrix<double,6,6> R;
    double p[3];
    p[0]= X(0,3);
    p[1]= X(1,3);
    p[2]= X(2,3);

    R.block<3,3>(0,0) = X.block<3,3>(0,0);
    R.block<3,3>(0,3) = w_hat(p)*X.block<3,3>(0,0);
    R.block<3,3>(3,0) = Eigen::Matrix3d::Zero();
    R.block<3,3>(3,3) = X.block<3,3>(0,0);

    return R;
}
```

w_hat 函数:

```
Eigen::Matrix<double,3,3> w_hat( double *p){
    Eigen::Matrix3d what;
    what = Eigen::Matrix3d::Zero();
    what(0,1) = - p[2];
    what(0,2) = p[1];
    what(1,2) = -p[0];
    what(1,0) = p[2];
    what(2,0) = -p[1];
    what(2,1) = p[0];

    return what;
}
```

Psi 函数：

```
Eigen::Matrix<double,6,1>  Psi(Eigen::Matrix<double,6,6>  T,
Eigen::Matrix<double,1,6> theta, Eigen::Matrix4d gz, Eigen::Matrix4d gd){
    Eigen::MatrixXd R(6,1);
    Eigen::Matrix<double,1,6> T0,T1,T2,T3,T4,T5;
    double theta0,theta1,theta2,theta3,theta4,theta5;
    Eigen::MatrixXd temp(4,4);
```

```
    Eigen::MatrixXd gd_pinv(4,4), logm(4,4), fk_gd(4,4), fk_gz(4,4);
    fk_gd = gd;
    fk_gz = gz;

    theta0 = theta(0);
    theta1 = theta(1);
    theta2 = theta(2);
    theta3 = theta(3);
    theta4 = theta(4);
    theta5 = theta(5);

    T0.block<1,6>(0,0)=T.block<1,6>(0,0);
    T1.block<1,6>(0,0)=T.block<1,6>(1,0);
    T2.block<1,6>(0,0)=T.block<1,6>(2,0);
    T3.block<1,6>(0,0)=T.block<1,6>(3,0);
    T4.block<1,6>(0,0)=T.block<1,6>(4,0);
    T5.block<1,6>(0,0)=T.block<1,6>(5,0);


    temp = Exp_twist(T0,theta0)*Exp_twist(T1,theta1)*Exp_twist(T2,theta2)*Exp_twist(T3,theta3)*Exp_twist(T4,theta4)*Exp_twist(T5,theta5);

    //gd_pinv = pseudoInverse(fk_gd);
    Eigen::CompleteOrthogonalDecomposition<Eigen::MatrixXd> cqr(fk_gd);
    gd_pinv = cqr.pseudoInverse();
    //std::cout<<"here\n"<<std::endl;
    temp=temp*fk_gz*gd_pinv;
    //std::cout<<"temp:\n"<<temp<<std::endl;
    logm = temp.log();    //#include <unsupported/Eigen/MatrixFunctions>
    //std::cout<<"logm:\n"<<logm<<std::endl;
    R = Vee(logm);

    return R;
}
```

Vee 函数：

```
Eigen::Matrix<double,6,1> Vee(Eigen::Matrix4d r){
    Eigen::Matrix<double,6,1> R;
    R.block<3,1>(0,0) = r.block<3,1>(0,3);
    //R.block<1,1>(3,0) = r.block<1,1>(2,1);
    //R.block<1,1>(4,0) = r.block<1,1>(0,2);
    //R.block<1,1>(5,0) = r.block<1,1>(1,0);
    R(3,0) = r(2,1);
    R(4,0) = r(0,2);
    R(5,0) = r(1,0);
    return R;
}
```

Norm 函数，总感觉 matlab 的 norm 函数和 Eigen 的 norm 函数有些区别，没有深究，这里简单借用一下，但还是留一个可改的函数体在这。

```
double norm(Eigen::Matrix<double,6,1> T){
    return T.norm();
}
```

涉及调用的运动旋量有关其他函数前面一章我已经介绍过，这里不重复贴了。

老规矩，写完算法，咱们得先测试，测试函数代码如下：

```
void test_Newton_Raphson(){
    double pointX,pointY,pointZ, angle;
    Eigen::MatrixXd TcpTarget;
    Eigen::Matrix<double,1,6> qq;

    Eigen::Matrix<double,1,6> T0, T1, T2, T3, T4, T5;
    T0 << 0, 0, 0, 0, 0, 1;
    T1 << 98.5, 0, 0, 0, -1, 0;
    T2 << 98.5, 0, 408,0, -1, 0;
    T3 << 98.5, 0, 784,0, -1, 0;
    T4 << 121.5, -784, 0, 0, 0, -1;
    T5 << -4, 0,784, 0, -1, 0;

    Eigen::Matrix4d gz;
    gz(0,0)=1; gz(0,1)=0; gz(0,2)=0; gz(0,3)= -784;
```

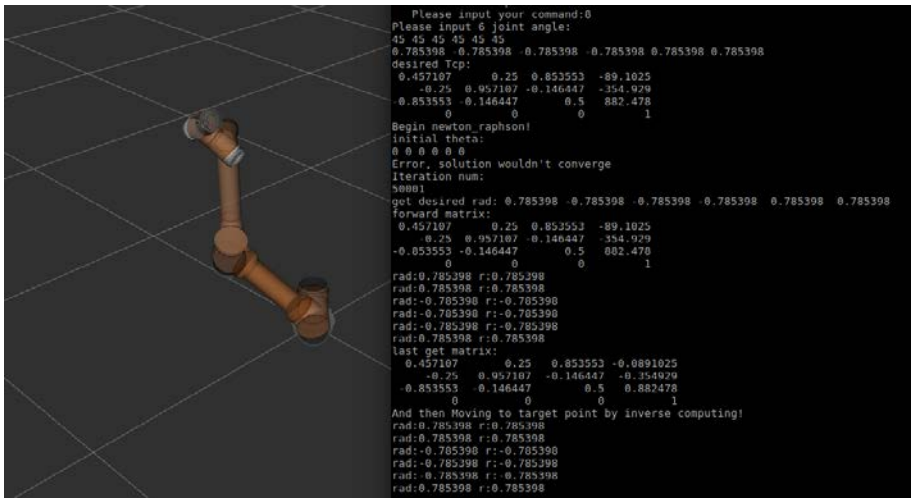
```
gz(1,0)=0; gz(1,1)=0; gz(1,2)=-1; gz(1,3)= -215.5;
gz(2,0)=0; gz(2,1)=1; gz(2,2)=0; gz(2,3)= -4;
gz(3,0)=0; gz(3,1)=0; gz(3,2)=0; gz(3,3)=1;
Eigen::Matrix<double,1,6> theta;
Eigen::Matrix<double,6,6> T;
double q0,q1,q2,q3,q4,q5;

printf("Please input 6 joint angle:\n");
scanf("%lf%lf%lf%lf%lf%lf",&q0,&q1,&q2,&q3,&q4,&q5);
q0 = q0*M_PI/180.0;
q1 = q1*M_PI/180.0;
q2 = q2*M_PI/180.0;
q3 = q3*M_PI/180.0;
q4 = q4*M_PI/180.0;
q5 = q5*M_PI/180.0;

printf("%lf %lf %lf %lf %lf %lf\n",
q0,q1-M_PI/2.0,-q2,q3-M_PI/2.0,q4,q5);

TcpTarget=ForwardKinematics(q0,q1-M_PI/2.0,-q2,q3-M_PI/2.0,q4,q5);

//draw tcp marker point
points.action = visualization_msgs::Marker::ADD;
points.lifetime=ros::Duration();
points.scale.x = 0.01f;
points.scale.y = 0.01f;
points.scale.z = 0.01f;
points.color.r = 1.0f;
points.color.g = 0.0f;
points.color.b = 0.0f;
points.color.a = 1.0;
pointX = TcpTarget(0,3);
pointY = TcpTarget(1,3);
pointZ = TcpTarget(2,3);
DisplayPoint(pointX,pointY,pointZ);
```

```
        TcpTarget(0,3) = TcpTarget(0,3) *1000;
        TcpTarget(1,3) = TcpTarget(1,3) *1000;
        TcpTarget(2,3) = TcpTarget(2,3) *1000;
        std::cout<<"desired Tcp:\n"<<TcpTarget<<std::endl;

        T.block<1,6>(0,0) = T0;
        T.block<1,6>(1,0) = T1;
        T.block<1,6>(2,0) = T2;
        T.block<1,6>(3,0) = T3;
        T.block<1,6>(4,0) = T4;
        T.block<1,6>(5,0) = T5;

        theta(0,0) = 0.0; theta(0,1)=0.0; theta(0,2)=0.0;theta(0,3)=0.0;
theta(0,4)=0.0; theta(0,5)=0.0;

        printf("Begin newton_raphson!\n");

        qq = Newton_Raphson(T,theta,gz,TcpTarget);
        std::cout<<"get desired rad:"<<qq<<std::endl;
        std::cout<<"forward
matrix:\n"<<Exp_twist(T0,qq(0,0))*Exp_twist(T1,qq(0,1))*Exp_twist(T2,qq(0,2))*
Exp_twist(T3,qq(0,3))*Exp_twist(T4,qq(0,4))*Exp_twist(T5,qq(0,5))*gz<<std::end
l;
        std::cout<<"last                                                    get
matrix:\n"<<ForwardKinematics(limit2pi(qq(0,0)),limit2pi(qq(0,1)),
limit2pi(qq(0,2)),limit2pi(qq(0,3)),limit2pi(qq(0,4)),limit2pi(qq(0,5)))<<std::endl;

        printf("And then Moving to target point by inverse computing!\n");
        moveJ(limit2pi(qq(0,0)),limit2pi(qq(0,1))+M_PI/2,
-limit2pi(qq(0,2)),limit2pi(qq(0,3))+M_PI/2 ,limit2pi(qq(0,4)),limit2pi(qq(0,5)));

    }
```

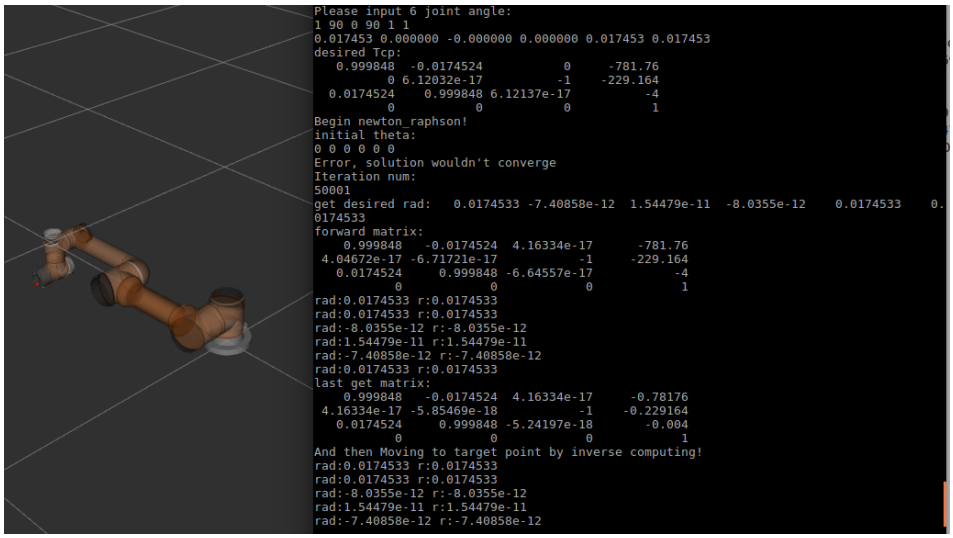
在这个测试案例中，我们还是先输入 6 个测试关节角，转成我们定义的

关节空间弧度值，然后用正解求出它的姿态矩阵，再用这个要达到的目标矩阵和零位姿态矩阵去比较，进行 Newton-Raphson 迭代。

(a)



(b)



(c)

图 5.11 (a) : {0 90 0 90 0 0}　(b) :{45 45 45 45 45 45}　(c): {1 90 0 90 1 1}迭代结果

从图 5.11 的三个迭代角度中可以看出，Newton-Raphson 迭代的精度也非常

高，但迭代次数基本都超出了程序设置的上限，Newton-Raphson 迭代法迭代的姿态更精确，在初始角都迭代了 5 次，这个是由于正解算法设置的精度误差导致，这个应该不是问题，最后一个迭代角度，比前一个算法迭代次数多出很多，也是由于这个精度造成，实际情况下我们可以降低 Newton-Raphson 迭代收敛误差要求，这样也许可以节约一点时间，但我们只是在讲基本原理，没有去做细致的实验验证，同学们有兴趣的可以去测试，并研究一下它们各自的优化算法。

有关机械臂的逆解实验我们介绍了四种算法（puma560 的通用逆解没贴代码，这个网上一大堆），请大家自己动手写代码，认真仔细考察掌握基本原理，只有牢固掌握了基本原理，才能深入去做算法优化研究工作，大部分教材这块基本没讲清，导致很多研究生写一堆的水货论文，实际都是教科书内容，甚至还拿去发表论文，确实太那个啥了。

下一章实验我们进行机械臂的轨迹规划算法介绍，请大家预习一下相关讲义和教材内容。

5.5 本章版权及声明

本章内容由闵华松(mhuasong@wust.edu.cn)编写。