

C++运动学轨迹规划与数据采集

用户手册

哈尔滨工业大学（深圳）

实验与创新实践教育中心



用户手册

C++运动学文档

设备代码：SI7400、SI7500、SI7600

版本 V1.0

日期 2020/1/2

版本历史

版本历史记录了每次文档版本更新的说明，最新版本的文档包含以前所有文档版本的更新内容。

文档版本	发布日期	修订内容
V1.0	2020/1/9	文档编写完成

目录

版本历史	I
目录	II
第 1 章 轨迹规划概述	1
1.1 总体概述	1
1.2 总框架	1
第 2 章 运动学算法介绍	3
2.1 SCARA 运动学算法	3
2.1.1 正运动学	3
2.1.2 逆运动	5
2.2 Delta 运动学算法	6
2.2.1 运动学正解	6
2.2.2 反解——几何方法	8
2.3 HL 六轴运动学算法	10
2.3.1 机械手的位置运动学正解	10
2.3.2 位置运动学反解	12
2.3.3 奇异位型分析	16
第 3 章 软件说明	19
3.1 功能概述	19
3.2 程序模块概述	19
3.2.1 点位规划程序讲解	21
3.2.2 视觉的取放料流程	26
3.2.3 码垛运动流程	28
3.2.4 避障流程	29
第 4 章 运动学算法调用接口	31
4.1 概述	31
4.2 轨迹规划调用流程	32

第 5 章 力矩数据计算.....	43
5.1 总体概述.....	43
5.1.1 操作流程框图.....	43
5.1.2 公式介绍.....	43
5.1.3 指令介绍.....	44
5.2 连接机器人	45
5.2.1 打开 Putty 软件.....	45
5.2.2 连接设置.....	45
5.3 采集数据设置.....	46
5.4 采集数据	47
5.5 数据导出	49
5.6 数据分析	49
5.6.1 采集数据获取.....	50
5.6.2 通道值获取	50
5.6.3 公式计算.....	51
第 6 章 注意事项.....	52
6.1 问题及解决办法	52
附录 A	56
ping-pong buffer 功能.....	56
附录 B	62
Ping-pong buffer 设计文档	62
文件实例.....	64
附录 C	65

第1章 轨迹规划概述

1.1 总体概述

轨迹规划，往往称为机器人轨迹规划，属于低层规划，基本上不涉及人工智能问题，而是在机械手运动学和动力学的基础上，讨论在关节空间和笛卡儿空间中机器人运动的轨迹规划和轨迹生成方法。所谓轨迹，是指机械手在运动过程中的位移、速度和加速度。而轨迹规划是根据作业任务的要求，计算出预期的运动轨迹。

对抓放作业的机器人(如用于上、下料)，需要描述它的起始状态和目标状态，即工具坐标系的起始值和目标值。在此，用“点”这个词来表示工具坐标系的位置和姿态(简称位姿)，例如起始点和目标点等。对于另外一些作业，如弧焊和曲面加工等，不仅要规定机械手的起始点和终止点，而且要指明两点之间的若干中间点(称路径点)，必须沿特定的路径运动(路径约束)。这类运动称为连续路径运动或轮廓运动，而前者称为点到点运动。

第一种方法要求用户对于选定的转变节点(插值点)上的位姿、速度和加速度给出一组显式约束(例如连续性和光滑程度等)，轨迹规划器从某一类函数(例如 n 次多项式)中选取参数化轨迹，对节点进行插值，并满足约束条件。

第二种方法要求用户给出运动路径的解析式；如为直角坐标空间中的直线路径，轨迹规划器在关节空间或直角坐标空间中确定一条轨迹来逼近预定的路径。

轨迹规划既可在关节空间也可在直角空间中进行，但是所规划的轨迹函数都必须连续和平滑，使得操作臂的运动平稳。在关节空间进行规划时，是将关节变量表示成时间的函数，并规划它的一阶和二阶时间导数；在直角空间进行规划是指将手部位姿、速度和加速度表示为时间的函数。而相应的关节位移、速度和加速度由手部的信息导出。通常通过运动学反解得出关节位移，用逆雅可比求出关节速度，用逆雅可比及其导数求解关节加速度。

1.2 总框架

本节讲解视觉代码中的程序总框架。

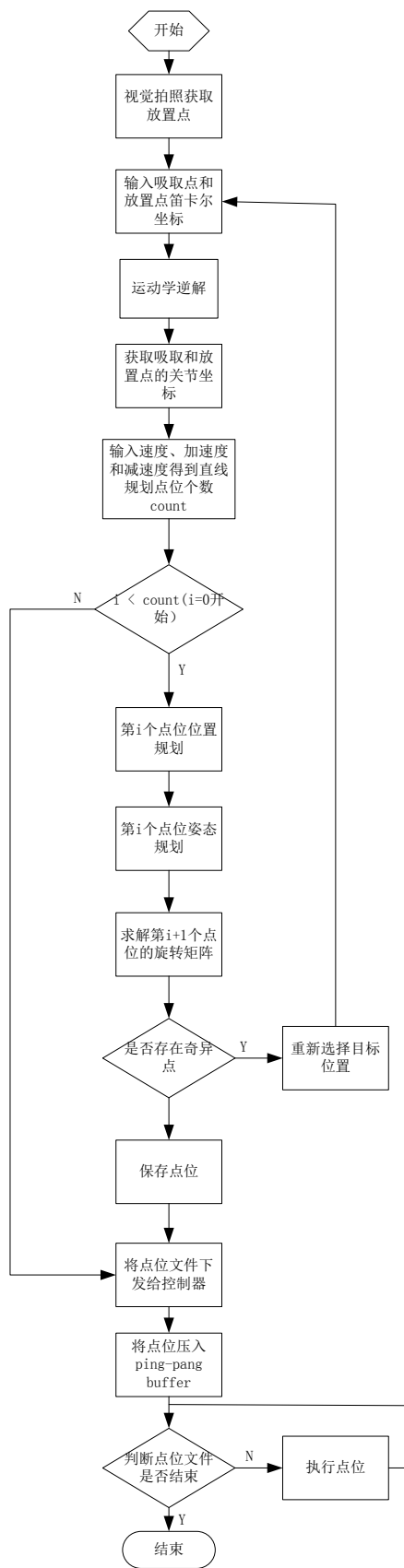


图 1-1 主函数流程框图

第2章 运动学算法介绍

2.1 SCARA 运动学算法

2.1.1 正运动学

关节变量， $\theta_1, \theta_2, d_3, \theta_4$ ，杆长 l_1, l_2 。

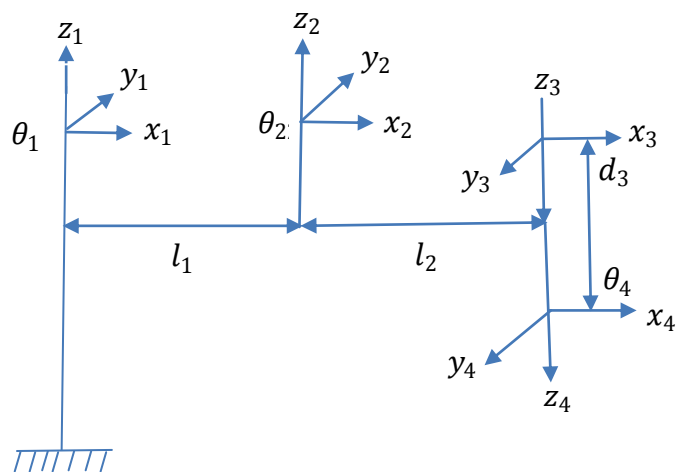


图 2-1 SCARA 结构图

SCARA 机器人 D-H 参数

	关节 转角 θ	连杆 偏移 d	连杆 距离 a	连杆扭 角 α
	θ_1	0	l_1	0
	θ_2	0	l_2	π
	0	d_3	0	0
	θ_4	0	0	0

$${}^0_4T = \begin{bmatrix} C_{12-4} & S_{12-4} & 0 & C_{12}l_2 + C_1l_1 \\ S_{12-4} & C_{12-4} & 0 & S_{12}l_2 + S_1l_1 \\ 0 & 0 & -1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中：

C_{12-4} --- 表示 $\cos(\theta_1 + \theta_2 - \theta_4)$

S_{12-4} --- 表示 $\sin(\theta_1 + \theta_2 - \theta_4)$

C_{12} ----表示 $\cos(\theta_1 + \theta_2)$

S_{12} ----表示 $\sin(\theta_1 + \theta_2)$

法兰末端的坐标 P 为：

$$P = \begin{bmatrix} l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) \\ -d_3 \end{bmatrix}$$

如果 $\theta_2 < 0$

Yaw = 0

Pitch = 180

Roll = $(\theta_4 - \theta_1 - \theta_2 - 180) - 360$

如果 $\theta_2 > 0$

Yaw = 0

Pitch = 180

Roll = $\theta_4 - \theta_1 - \theta_2 - 180$

2.1.2 逆运动

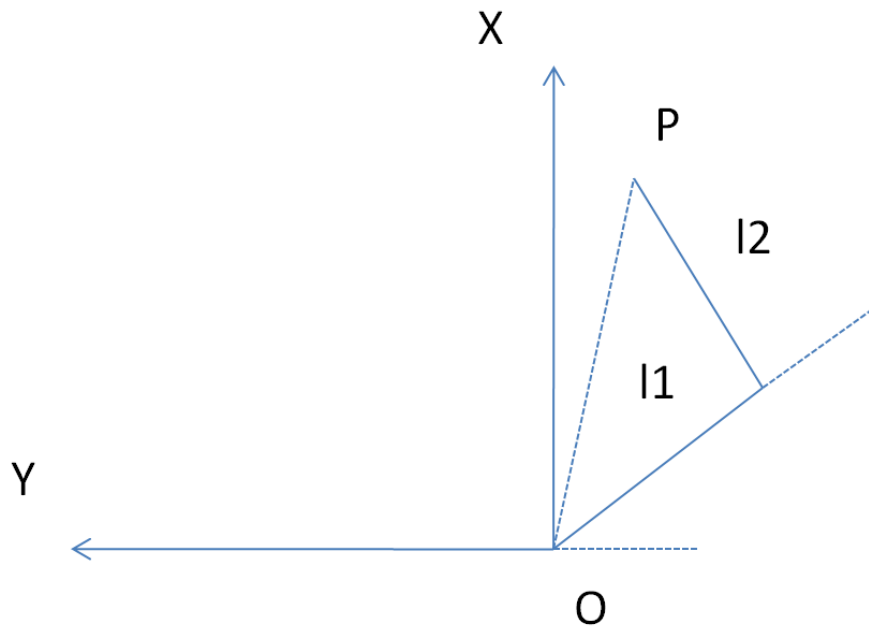


图 2-2 SCARA 简图

以右手系为例

末端位置 $P = [x \ y \ z]^T$ 和旋转矩阵 R （或末端旋转角度 Yaw）

如果是右手姿态：

$$\theta_1 = \beta - \alpha$$

$$\theta_2 = \arccos\left(\frac{l_1^2 + l_2^2 - x^2 - y^2}{2l_1l_2}\right)$$

$$d_3 = -z$$

$$\theta_4 = 180 + \text{Roll} + \theta_1 + \theta_2$$

如果是左手姿态：

$$\theta_1 = \beta + \alpha$$

$$\theta_2 = -1 * \arccos\left(\frac{l_1^2 + l_2^2 - x^2 - y^2}{2l_1l_2}\right)$$

$$d_3 = -z$$

$$\theta_4 = (180 + \text{Roll} + \theta_1 + \theta_2) + 360$$

2.2 Delta 运动学算法

2.2.1 运动学正解

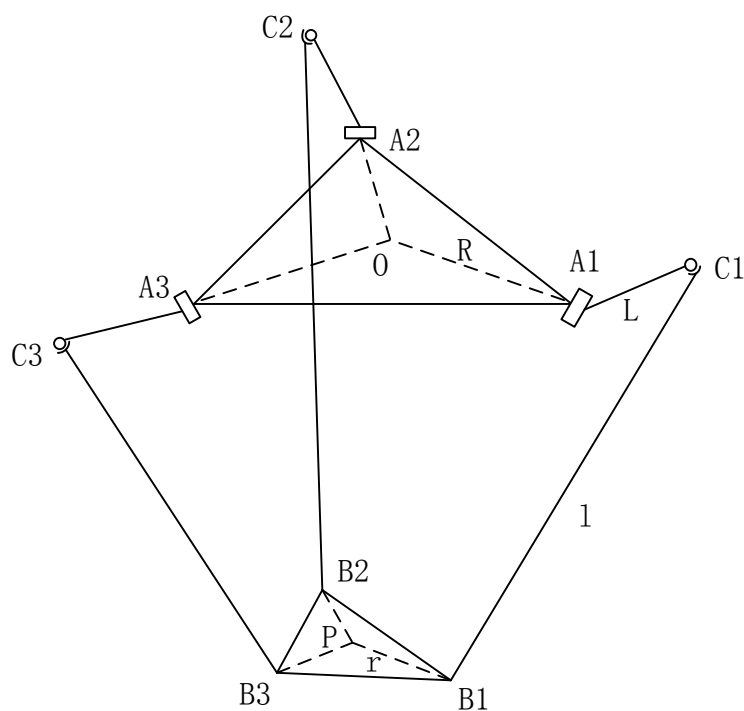


图 2-3 Delta 结构图

Delta 机器人的世界（固定）坐标系原点建立在静平台的 O 点，X 轴和 OA_1 重合，Z 轴和静平台所在平面垂直，Y 轴方向根据右手定则。 OA_i 与 X 轴的夹角为 $\alpha_i = (i - 1) \times \frac{2}{3}\pi, i = 1, 2, 3$ 。机器人动坐标系原点建立在动平台 P 点，X' 轴和 PB_1 重合，Z' 轴与动平台所在平面垂直。

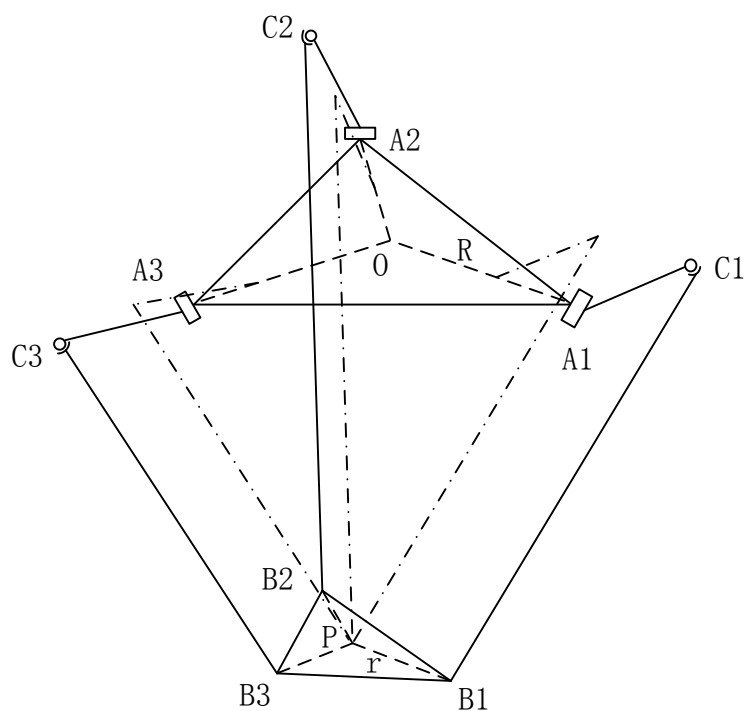


图 2-4 Delta 结构图

求解正解时，将每个支链沿 PB_i 移动 r 距离，使得 C_i 到达 C'_i ，并且 B_i 点和 P 点重合，由于三个主动臂的输入角度已知，可以求得 C'_i 点坐标。又三个从动臂等长，根据 C'_1, C'_2, C'_3, P 形成的三菱锥可以求得 P 点坐标。过 P 点做 PE 垂直于 C'_1, C'_2, C'_3 所在平面。接着，过 E 点做 EF 垂直于 $C'_1C'_2$ 。由于 $PC'_1 = PC'_2 = PC'_3$ ，所以 $EC'_1 = EC'_2 = EC'_3$ ，即 E 为 $\Delta C'_1C'_2C'_3$ 的外接圆圆心， EC'_i 为外接圆半径。 F 为 $C'_1C'_2$ 的中点。

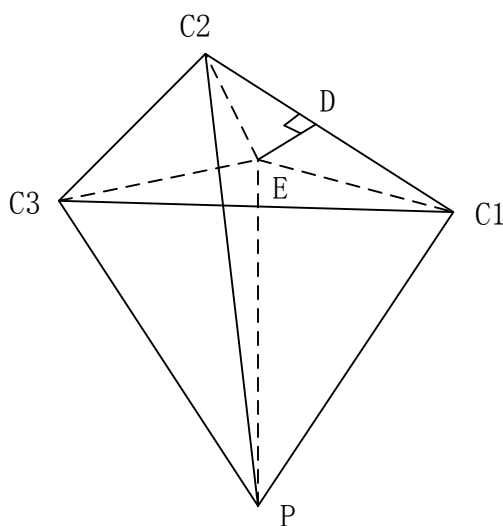


图 2-5 Delta 简图

首先，求得 C'_i 在世界坐标系中的位置

$$C'_i = \begin{bmatrix} (R - r + L \cos \theta_i) \cos(\alpha_i) \\ (R - r + L \cos \theta_i) \sin(\alpha_i) \\ -L \sin \theta_i \end{bmatrix}, i = 1, 2, 3$$

其次，根据海伦公式，可以求出外接圆半径

$$|EC'_i| = \frac{|C'_1 C'_2| \cdot |C'_2 C'_3| \cdot |C'_3 C'_1|}{4\sqrt{p(p - |C'_1 C'_2|) \cdot (p - |C'_2 C'_3|) \cdot (p - |C'_3 C'_1|)}}$$

其中 $p = (|C'_1 C'_2| + |C'_2 C'_3| + |C'_3 C'_1|)/2$ 。

接着，求出 F 点坐标

$$F = \frac{C'_1 + C'_2}{2}$$

FE 的方向向量和长度分别为：

$$n_{FE} = \frac{C'_2 C'_3 \times C'_3 C'_1 \times C'_1 C'_2}{|C'_2 C'_3 \times C'_3 C'_1 \times C'_1 C'_2|}$$

$$|FE| = \sqrt{|EC'_i|^2 - \frac{|C'_1 C'_2|^2}{4}}$$

可以得到 E 点坐标

$$E = |FE| \cdot n_{FE} + F$$

EP 的方向向量和长度分别为：

$$n_{EP} = \frac{-C'_1 C'_2 \times C'_2 C'_3}{|C'_1 C'_2 \times C'_2 C'_3|}$$

$$|EP| = \sqrt{l^2 - |EC'_i|^2}$$

可以得到 P 点坐标为：

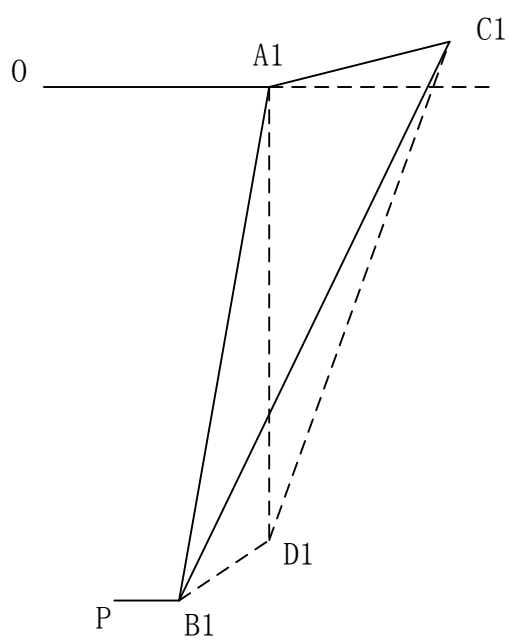
$$P = |EP| \cdot n_{EP} + E$$

这样就得到了 delta 机器人的正解。

2.2.2 反解——几何方法

以第一条支链为例，当 $P = [x \ y \ z]^T$ 已知时，可以计算得到， B_1 点坐标，过 B_1 做 D_1 点垂直于 $OA_1 C_1$ 平面，可以求得 D_1 点坐标。这样可以得到 $A_1 D_1$ 和 $C_1 D_1$ 的长度，从而可以求得 $\angle D_1 A_1 C_1$ 以及 $\angle D_1 A_1 O$ 的角度值，以及这样，可

以求出 $\theta_1 = \pi - \angle D_1 A_1 C_1 - \angle D_1 A_1 O$



2.3 HL 六轴运动学算法

2.3.1 机械手的位置运动学正解

为了后续方便描述其结构，对其关节做以下定义：1 号称为腰关节，2 号称为肩关节，3 号称为肘关节，后 3 个称为腕关节。通常机械手前三个关节运动所确定的运动空间比较大，通常用来确定机器人末端执行器的空间位置，而后三个腕关节则主要是用来调整机器人末端执行器的姿态。用 D-H 法获得其连杆坐标系，如下图所示。

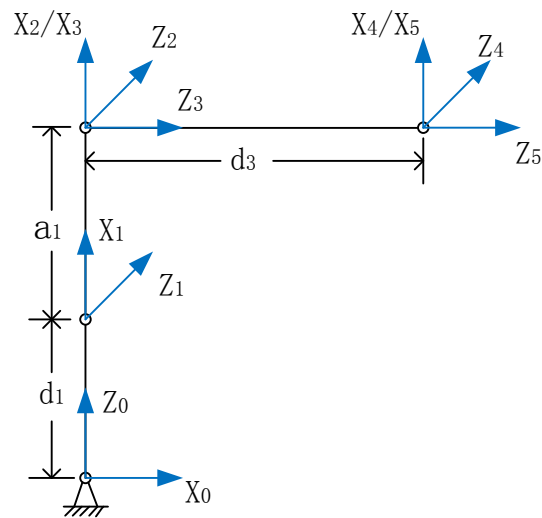


图 2-6 HL 结构图

HL6-900 机器人本体 D-H 参数

杆长 序号	$a_{i-1}(mm)$	$\alpha_{i-1}(^{\circ})$	$d_i(mm)$	$\theta_i(^{\circ})$
1	0	0	491	0
2	0	-90	0	-90
3	450	0	0	0
4	0	-90	450	0

5	0	90	0	0
6	0	-90	84	0

表中， a_{i-1} 代表连杆的长度， $\alpha_{i-1}(^\circ)$ 代表连杆的扭角， d_i 代表连杆的偏置， θ_i 代表连杆 i 和 $i-1$ 间夹角。其中前面三个参数是固定的，后面的 θ_i 角则是变动的。

末端连杆的位姿矩阵为 $T_6^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5$ ，其中

$$T_i^{i-1} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{若最终的位姿矩阵用 } T_6^0 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ 表示，则}$$

最终简化的运动学正解方程：（化简之后的公式见代码）

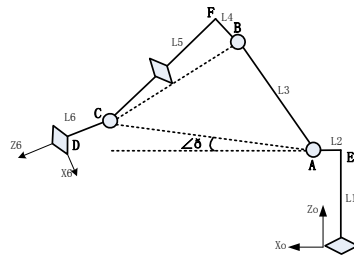
$$\begin{aligned} n_x &= f_1(\theta, \alpha) \\ n_y &= f_2(\theta, \alpha) \\ n_z &= f_3(\theta, \alpha) \\ o_x &= f_4(\theta, \alpha) \\ o_y &= f_5(\theta, \alpha) \\ o_z &= f_6(\theta, \alpha) \\ a_x &= f_7(\theta, \alpha) \\ a_y &= f_8(\theta, \alpha) \\ a_z &= f_9(\theta, \alpha) \\ p_x &= f_{10}(\theta, \alpha) \\ p_y &= f_{11}(\theta, \alpha) \\ p_z &= f_{12}(\theta, \alpha) \end{aligned}$$

2.3.2 位置运动学反解

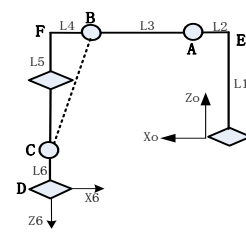
运动学反解指根据末端杆位姿反求关节位置，其结果通常存在多解情况。对六自由度串联机械手而言，通过解耦方式求取的运动学反解通常会产生八组解，而在实际的机械手控制过程中，只会使用一组解，从这个角度看，多解问题会给运动控制带来一些不利影响。实际控制过程中我们会根据机器人本身结构，以及机器人当前的关节状态来实时选择最优的一组解，也就是保证关节角度的连续性，保证机械手本体不产生干涉。当然，其反解的多解情况也使得机械手在笛卡尔空间具有更好的可规划性以及反向运动的灵活性。当前，国际上一些机器人公司对于多解问题也有自己的一套处理措施。例如，日本 FAUNC 公司在六轴机器人控制器中加入了数据配置，用 3 个字符数据组合来表示机器人当前的 8 种状态；EPSON 也在六轴机器人控制器加入了关节模式配置，用来描述机器人当前处于何种状态，从而合理的选择最合适的一组运动学反解。

对于上面图中所示结构类型的机械手本体，其运动学反解可以通过几何解析和坐标变换的方式来求解。

假设已知其末端的位姿矩阵为 $T = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$



a. 工作状态示意图



b. 初始状态示意图

如上图 a 所示，通过姿态矩阵 T 可以求得 C 点坐标为

$C(p_x - a_x l_6, p_y - a_y l_6, p_z - a_z l_6)$ ，则关节 1 角度为：

$$\theta_1 = \arctan 2(p_y - a_y l_6, p_x - a_x l_6), \quad \theta_1 = \arctan 2(-(p_y - a_y l_6), -(p_x - a_x l_6))。$$

从而A点坐标为 $A(l_2 c_1, l_2 s_1, l_1)$ 。其中， $c_1 = \cos \theta_1, s_1 = \sin \theta_1$ ，其余以此类推。

在 $\triangle ABC$ 中 $\theta_3' = \angle ABC = a \cos(\frac{l_3^2 + l_4^2 + l_5^2 - AC^2}{2 * \sqrt{l_4^2 + l_5^2} * l_3})$ ，AC可以通过A点和C点坐标求得。

如上图b所示，在初始状态时 $\theta_3^0 = \angle ABC = a \cos(\frac{l_3^2 + l_4^2 + l_5^2 - AC^2}{2 * \sqrt{l_4^2 + l_5^2} * l_3})$ ，此时

A点处坐标为 $A(l_2, 0, l_1)$ 。故关节3的角度为： $\theta_3 = \theta_3' - \theta_3^0$ ， $\theta_3 = 2\pi i - (\theta_3' + \theta_3^0)$ 。

如上图a所示， $\overrightarrow{AC} = (p_x - a_x l_6 - l_2 c_1, p_y - a_y l_6 - l_2 s_1, p_z - a_z l_6 - l_1)$ 。

当 $\theta_1 = \arctan 2(p_y - a_y l_6, p_x - a_x l_6)$ ，且

$$\sqrt{(p_y - a_y l_6)^2 + (p_x - a_x l_6)^2} \geq \sqrt{(l_2 s_1)^2 + (l_2 c_1)^2} \text{ 时}$$

当 $\theta_2' = \angle \alpha = \arctan 2(AC_z, \sqrt{AC_x^2 + AC_y^2})$ ，且

$$\sqrt{(p_y - a_y l_6)^2 + (p_x - a_x l_6)^2} \geq \sqrt{(l_2 s_1)^2 + (l_2 c_1)^2} \text{ 时}$$

$\theta_2' = \angle \alpha = \frac{\pi}{2} + \arctan 2(AC_z, \sqrt{AC_x^2 + AC_y^2})$ ，式中 AC_x, AC_y, AC_z 为向量

\overrightarrow{AC} 的X, Y轴分量， $\theta_2^0 = \angle BAC = a \cos(\frac{l_3^2 + AC^2 - (l_4^2 + l_5^2)}{2 * AC * l_3})$ 。当 $\theta_3 = \theta_3' - \theta_3^0$ 时，

关节2的角度为： $\theta_2 = \theta_2' + \theta_2^0$ 。当 $\theta_3 = 2\pi i - (\theta_3' + \theta_3^0)$ 时，关节2的角度为：

$$\theta_2 = \theta_2' - \theta_2^0。$$

当 $\theta_1 = \arctan 2(-(p_y - a_y l_6), -(p_x - a_x l_6))$ 时，

$$\theta_2' = \angle \partial = \pi - a \tan 2(AC_z, \sqrt{AC_x^2 + AC_y^2})。$$

因此，前三个关节共产生了4组解：

左上肘方向:

$$\begin{cases} \theta_1 = \arctan 2(p_y - a_y l_6, p_x - a_x l_6) \\ \theta_2 = \theta_2' + \theta_2^0 \\ \theta_3 = \theta_3' - \theta_3^0 \end{cases}$$

左手下肘方向:

$$\begin{cases} \theta_1 = \arctan 2(p_y - a_y l_6, p_x - a_x l_6) \\ \theta_2 = \theta_2' - \theta_2^0 \\ \theta_3 = 2\pi - (\theta_3' + \theta_3^0) \end{cases}$$

右上肘方向:

$$\begin{cases} \theta_1 = \arctan 2(-p_y + a_y l_6, -p_x + a_x l_6) \\ \theta_2 = \theta_2' + \theta_2^0 \\ \theta_3 = \theta_3' - \theta_3^0 \end{cases}$$

右手下肘方向:

$$\begin{cases} \theta_1 = \arctan 2(-p_y + a_y l_6, -p_x + a_x l_6) \\ \theta_2 = \theta_2' - \theta_2^0 \\ \theta_3 = 2\pi - (\theta_3' + \theta_3^0) \end{cases}$$

这四组解分别对应了机械手工作时的 4 中结构状态, 因此, 在实际的控制过程中, 可以根据当前的机械手的结构状态, 选择一种最优解, 从而来保证在控制过程中角度时间的连续性。

由于机械手后三个关节是一组 Euler 的组合关节, 通常是用来产生机械手的姿态, 其逆解过程也非常简单。因此, 机械手的最终姿态可以通过以下矩阵来描述:

$$R_6^0 = R_1^0 * R_2^1 * R_3^2 * R' * Euler(\theta_4, \theta_5, \theta_6) = R * Euler(\theta_4, \theta_5, \theta_6)$$

$$\text{式中 } R' = \text{rot}_x\left(\frac{\pi}{2}\right), \quad R = R_1^0 * R_2^1 * R_3^2 * R', \text{ 因此 } Euler(\theta_4, \theta_5, \theta_6) = R^{-1} * R_6^0。$$

$$\text{而 } Euler(\theta_4, \theta_5, \theta_6) = R_z(\theta_4) * R_y(\theta_5) * R_z(\theta_6) = \begin{bmatrix} c_4 s_5 c_6 - s_4 s_6 & -c_4 c_5 s_6 - s_4 c_6 & c_4 s_5 \\ s_4 c_5 c_6 + c_4 s_6 & -s_4 c_5 s_6 + c_4 c_6 & s_4 s_5 \\ -s_5 c_6 & s_5 s_6 & c_5 \end{bmatrix}$$

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \text{ 则 } \begin{cases} \theta_4 = A \tan 2(r_{23}, r_{13}) \\ \theta_5 = A \tan 2(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \text{ (非翻转腕方向)} \\ \theta_6 = A \tan 2(r_{32}, -r_{31}) \end{cases}$$

另一解为：

$$\begin{cases} \theta_4 = A \tan 2(-r_{23}, -r_{13}) \\ \theta_5 = A \tan 2(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \text{ (翻转腕方向)} \\ \theta_6 = A \tan 2(-r_{32}, r_{31}) \end{cases}$$

当 $\theta_5 = 0, \pi$ 时，此处为腕关节奇异点，关节 4 和关节 6 的角度和为一定值。关于机械手奇异在下一节会有介绍，在此不做分析。

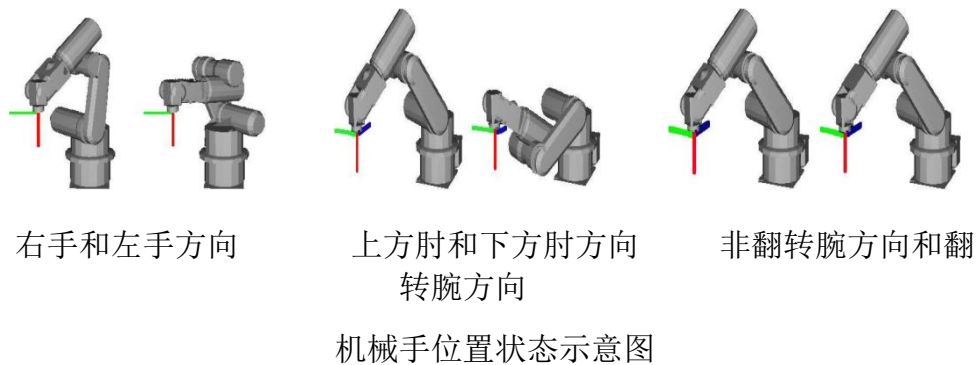
综上所述，可得到以上八组运动学反解。对于控制过程中反解结果的选取，除了根据机器人物理的关节角度限制以外，还可以根据机器人当前关节角度状态来选取（如下图所示）。以上八组解可以根据下图所示中八种机器人关节状态来选择一种合适的运动学反解。

关节状态定义：

手方向： $\overrightarrow{OC}_{xy} = (p_x - a_x l_6, p_y - a_y l_6)$ 和 $\overrightarrow{OA}_{xy} = (l_2 c_1, l_2 s_1)$ 方向相同表示右手方向，方向相反表示左手方向。

肘方向： 当 $A \tan(l_5, l_4) < \theta_3 \leq \pi + A \tan(l_5, l_4)$ 时，为上方肘方向，当 $\pi + A \tan(l_5, l_4) < \theta_3 < 2\pi + A \tan(l_5, l_4)$ 时，为下方肘方向。

腕方向： 当 $0 < \theta_5 \leq \pi$ 时，为非翻转手腕方向，当 $-\pi < \theta_5 < 0$ 时，为翻转手腕方向。

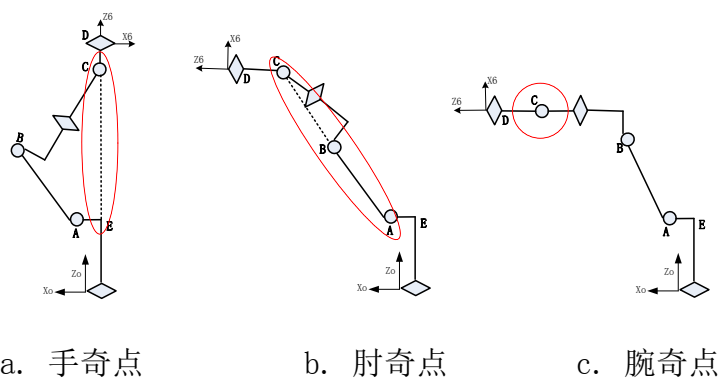


2.3.3 奇异位型分析

对于上述类型的六轴机械手而言，通常有 3 种奇异位型，以每一个奇异点作为机械手关节状态临界点可以将机械手分为 2 种关节状态属性，3 种奇异位型也就产生了前面所提到的 8 种机械手关节状态属性。机械手奇异位型临界状态如下图所示：

➤ 奇异位型分析

奇点是机械臂从一个方向切换到另一个方向的分界点，在该分界点处机械手通常会丧失一个自由度。



机械手奇异状态示意图

如上图 a 所示，当手腕中心位于关节 1 的轴线上时，无论关节 1 如何旋转 C 点的位置都不会改变，机械手因此而丧失了一个自由度，此位置点为机械手的手奇点，也就是右手状态和左手状态的分界点，根据该奇异点可以将机械手关节状态分为右手方向和左手方向。

如上图 b, 当 A 、 B 、 C 三点共线时, 也就关节 2、3、5 轴线共面时, 机械手速度运动学的雅克比矩阵处于非满秩状态, 此时关节 2 和关节 3 产生的速度效果相同, 机械手丧失一个自由度, 称此位置点为肘奇点, 也称为机械手速度奇点。以该奇点作为分界点, 可以将机械手关节状态分为上方肘方向和下方肘方向。

如上图 c 所示, 当关节 4 和关节 6 的轴线共线时, 它们旋转时产生的姿态效果是相同的, 相当于机械手丧失了一个关节自由度, 此位置点称为机械手的腕奇点, 也称为机械手的姿态奇点。以该奇点作为分界点, 可以将机械手关节状态分为非翻转腕方向和翻转腕方向。

以上 3 种关节属性可以组合成 8 种关节状态属性, 分别是: 右手上肘非翻转腕、右手上肘翻转腕、右手下肘非翻转腕、右手下肘翻转腕、左手上肘非翻转腕、左手上肘翻转腕、左手下肘非翻转腕、左手下肘翻转腕。这 8 中关节状态属性分别对应了前面的 8 组机械手运动学反解。

➤ 奇异点处理

关节状态属性的引入能够更直观的让控制者了解奇异点的位置状态, 提高运动学反解速度, 更可以用于预判机械手运动过程中是否过奇异点。

机械手奇异点问题在位置运动学中是可以避免的, 如上图 a 所示为手奇异, 它的奇异点出现在笛卡尔空间 $x=0, y=0$ 的空间直线上。上图 b 所示为肘奇异, 它的奇异点出现在关节空间 $\theta_3 = A \tan(l_3, l_4)$ 的外部边界空间曲面上。图 c 所示为腕奇异, 它的奇异点出现在机械手的腕部关节 $\theta_5 = 0$ 的情况下。其中, 手奇异和腕奇异称为机械手的内部奇异, 而肘奇异为机械手的外部边界奇异, 通常只要机械手不在其可运动区间的边界运动就可以避免。

对于手奇异而言, 在笛卡尔坐标空间规划下当机械手运动经过 $x=0, y=0$ 的空间直线时, 它会进入到另一个机械手关节状态属性, 此时保持了运动关节角度的连续, 只需要选择另一组反解即可。而对于规划点如果刚好和该奇点重合, 可以通过跳过该规划点, 而进入到下一个规划点, 避开该奇异点。该方法不仅能够保证反解的求取, 也保证了关节空间内各关

节角度轨迹的连续性。

同理，腕部奇异也是同样的方式，当规划点和奇异点重合时而进行反解运算时，此时反解结果中关节 5 的角度 $\theta_5 = 0$ ，而关节 θ_4, θ_6 由于无解，不做运算而保持当前的关节角度输出，也就是关节 4 和关节 6 在该规划点下不做运动。

综上所述，和速度运动学奇异点不同，位置运动学奇异点只会在某一个确定的点处产生奇异而导致运动学反解无解，因此，它的奇异位型完全可以通过直角坐标空间内的规划目标点进行避免。

第3章 软件说明

3.1 功能概述

C++运动规划界面目前提供了 4 种运动模式，分别是：

- ① 轨迹规划点位运动
- ② 结合视觉的取放料运动
- ③ 码垛运动
- ④ 避障碍运动

其中②，③，④的连续轨迹运动是①功能的复合运用。①功能的轨迹规划点位运动是点到点的运动。

举例子：现在要执行功能④避障，避障的路径是由 4 个点组成，分别是 A,B,C,D。那么运动的流程为 $A \rightarrow B$ ， $B \rightarrow C$ ， $C \rightarrow D$ ，可以拆分 3 组①功能点到点的运动。②和③的综合功能也是将连续的运动拆分为了几组①功能点到点的运动实现了连续的轨迹运动。

3.2 程序模块概述

上位机的源码由下表种的模块组成

模块名	意义
DhfLib	单例模式模板
HPSocket	Socket 网络通讯库
MotionPlan	轨迹规划算法
CmdControl	启动视觉软件
FtpControl	Ftp 通讯将 Data 文件放入、机器人的本体
IOControlDlg	IO 控制界面底层函数
Robot	机器人的宏指令接口

RobotApplicationMFC	上位机程序的入口
RobotApplicationMFCDlg	上位机主界面底层函数
RobotTeach	上位机示教界面底层函数
SystemParam	系统参数保存模块，用来保存点位等参数
VisionCalibrationDlg	视觉标定界面底层函数
VisionConveyorSelectDlg	传送带选择界面底层函数
VisionModeSelectDlg	视觉模式选择界面底层函数
RobotApplicationMFC	资源文件，包含了界面的 DATA 文件

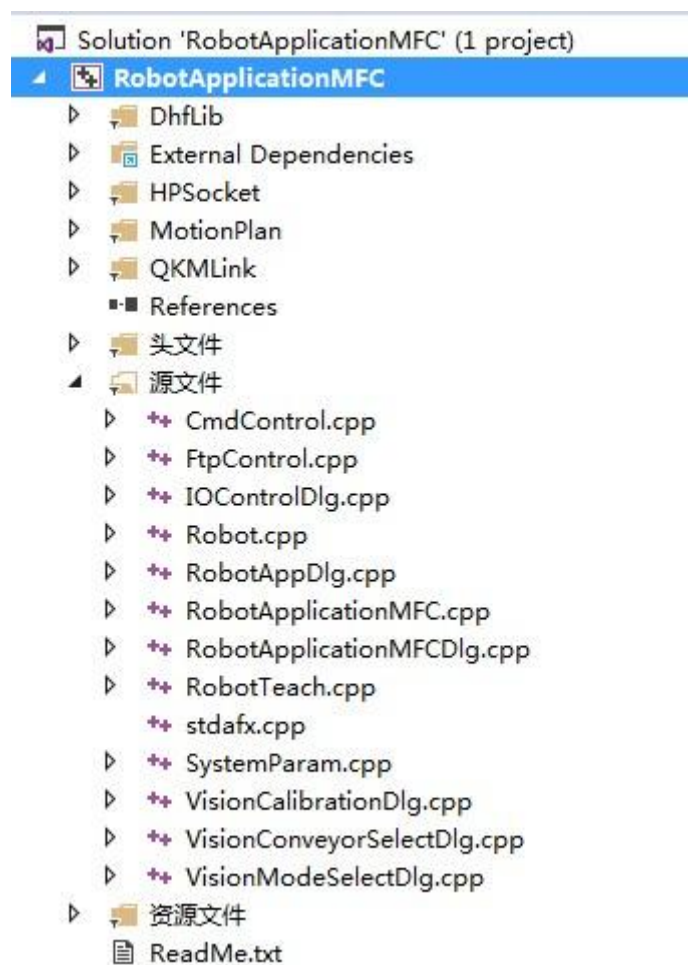


图 3-1 程序模块组成

RobotApplicationMFC 资源模块包含了窗口界面设计，一共有一下几个界面

IDD_DIALOG_CALIBRATION	视觉标定界面
IDD_DIALOG_CONVEYOR	选择传送带颜色界面
IDD_DIALOG_IO_CONTROL	IO 控制界面
IDD_DIALOG_ROBOT_APP	机器人应用界面
IDD_DIALOG_TEACH	示教界面
IDD_DIALOG_VISIONMODEL	视觉模式选择界面
IDD_ROBOTAPPLICATIONMFC_DIALOG	主界面

图 3-2 界面组成

3.2.1 点位规划程序讲解

1. 使用流程

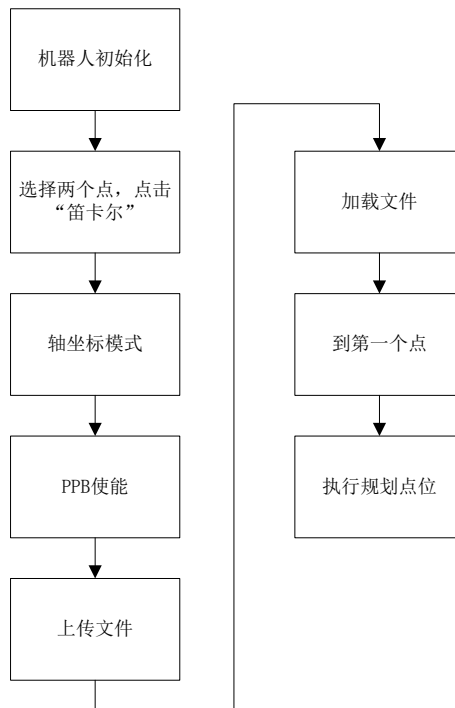


图 3-3 轨迹规划流程

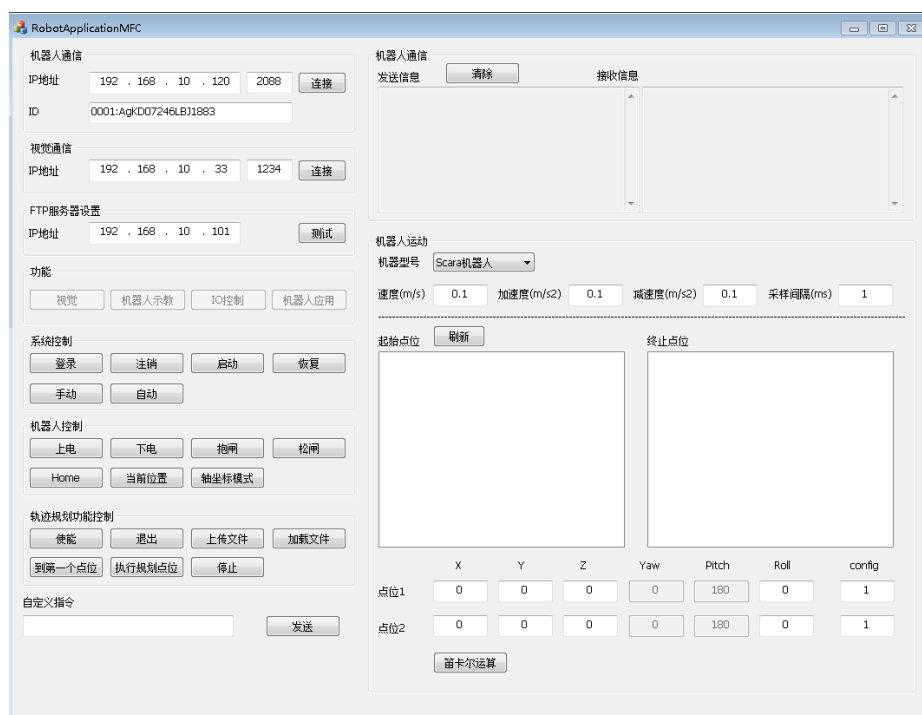


图 3-4 轨迹规划主界面图

如图所示的 Button 按钮，在点击时会触发相应的事件，并执行对应的操作。

比如点击“登陆”按钮，底层会触发 OnBnClickedButtonSystemLogin 函数，调用里面的 Login（）函数发送“System.Login 0”的宏指令到机器人端。同理其它按钮也是这样的效果。

```
//登陆
void CRobotApplicationMFCDlg::OnBnClickedButtonSystemLogin()
{
    // TODO: 在此添加控件通知处理程序代码
    Login();
}
```

图 3-5 点击 Button 触发此函数

```

bool Robot::Login()
{
    if (IsConnected)
    {
        //复位接收标志
        RecvLock.lock();
        RecvStatus = 0;
        Message = L"";
        RecvLock.unlock();

        CString sendData;
        sendData.Format(_T("System.Login 0"));

        //发送数据
        bool result = Send(sendData);
        if (!result)
        {
            return false;
        }

        //等待数据返回
        for (size_t i = 0; i < ROBOT_RECV_TIME_OUT; i++)
        {
            RecvLock.lock();

```

图 3-6 Robot 模块内部的宏指令函数 Login

2. 流程介绍

1) 机器人初始化

机器人初始化是指在机器人执行轨迹规划运动前的准备。常见的操作：

登陆（加载权限。宏指令“System.Login 0”）

↓

上电（机器人上使能。宏指令“Robot.PowerEnable 1,1”）

↓

回零（更新编码器值，一般断点重启后执行一次即可。宏指令“Robot.Home”）

2) 选择两个点，点击笛卡尔运算

界面点击“刷新”后会显示目前已记录的点位，选择起始点和终点后，点击“笛卡尔运算”按钮，便会生产“/Data.txt”文档，里面包含了规划的点位。

```
-11.0458 43.2569 10.024 154.71
-11.0458 43.2569 10.024 154.71
-11.0458 43.2569 10.024 154.71
-11.0458 43.2568 10.024 154.71
-11.0458 43.2568 10.024 154.71
-11.0458 43.2568 10.024 154.71
-11.0458 43.2567 10.024 154.71
-11.0458 43.2567 10.024 154.71
-11.0458 43.2566 10.024 154.71
-11.0458 43.2566 10.024 154.71
-11.0458 43.2565 10.024 154.71
-11.0458 43.2565 10.024 154.71
-11.0458 43.2564 10.024 154.71
-11.0458 43.2563 10.024 154.71
```

图 3-7 轨迹规划生产的关节坐标

3) 使能

使能是 PPB 宏指令的一种，使能是指是开启此模式

Ping-pong buffer (PPB) 功能是，使用者 bypass Pallas 内部默认的 motion planner 和 trajectory generator，以实时网络通信周期（e.g 1ms）把指令讯息直接给到 servo，然后执行 robot 动作。或者，用户给出一些关键点位，在底层进行样条曲线拟合，轨迹规划，最后将实时指令讯息给到伺服。

4) 上传文件

将点位文件通过 FTP 拷入 192.168.10.101/data。

5) 加载文件

调用宏指令 PPB.ReadFile 1,/data/XXX.txt 的方式，将规划的点位压入底层的 Buffer 中

6) 运动到第一个点

运动到起始位置，为执行后面的运动做准备

7) 执行轨迹规划点位

PPB.Run 1，执行 Data 文件内的点位

下表介绍了 PPB 模式的相关宏指令，更详细的内容翻看附录《Ping-pong buffer 设计说明》

指令 ID	关键字	描述
03850	PPB	ping-pong buffer
103851	PPB.Enable	进入和退出 ping-pong buffer 模式
103852	PPB. ReadFile	读取用户点位文件
103853	PPB. Prepare	机器人通过 Joint 运动到用户点位文件的第一个点
103854	PPB.Run	机器人开始执行 ping-pong buffer 中的点位
103855	PPB.Stop	机器人中断运动动作，执行减速停止
103856	PPB.Insert	宏指令方式向 ping-pong buffer 中插入点位
103857	PPB.FreeRoom	查询 ping-pong buffer 的剩余空间
103857	PPB.ClearBuffer	清理 ping-pong buffer 的剩余空间

3.2.2 视觉的取放料流程

1. 视觉使用界面

全局控制

示教点

视觉结果

视觉抓取

安全位

抓取位

Z轴安全距离 (>0, 为抓取时,在物料上方抓取就绪的)

抓取角度偏移((±180,为抓取物料时,Roll的偏移)

放置位

流程

到安全位置->视觉识别->到抓取位置上方(Z轴安全距离)->到抓取位置(视觉->吸气->回到抓取位置上方->到放置位置->放气->回到安全

图 3-8 视觉流程界面

界面介绍:

- ① 刷新: 更新点位信息
- ② 获取视觉结果: 触发相机拍照, 获取视觉点位
- ③ 安全位: 选择示教的点位-安全位, 机器人每次运行会先回到此点
- ④ 抓取位: 抓取的 X, Y, Roll 依靠视觉获取, 但是 Z 轴的高度需要示教出来, 所以抓取位置主要提供抓取高度
- ⑤ Z 轴安全距离: 距离取料点的相对高度, 为了安全起见, 机器人会先去抓取点的正上方, 然后才会去抓取点抓料
- ⑥ 抓取角度偏移: 抓取的角度偏移一般为第一次示教点位时, 视觉结果的角度的相反数, 用于纠偏, 因为夹具的安装会引起角度的偏差。
- ⑦ 放置位: 取完物体后, 放置的地方
- ⑧ 运行: 点击此按钮, 执行视觉抓取的一系列流程

2. 点击运行执行流程

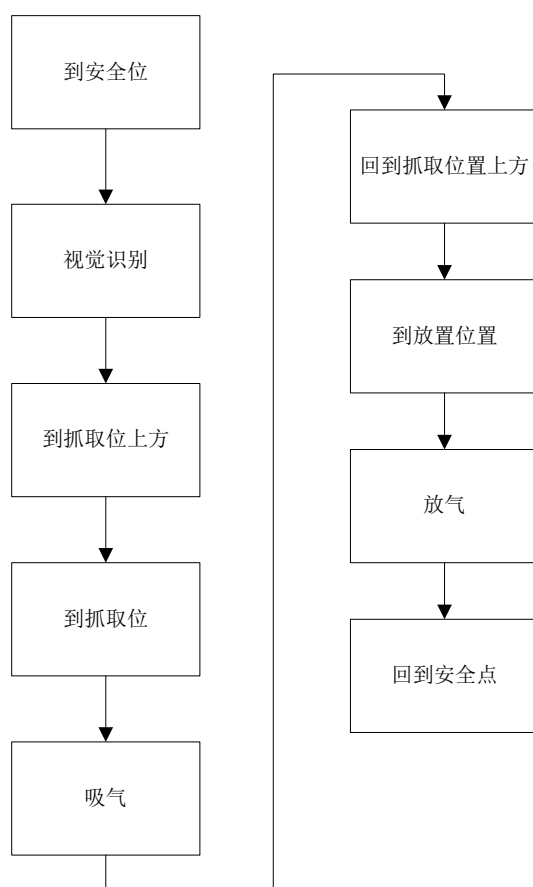


图 3-9 运行执行流程

点击“运行”时，触发 OnBnClickedEditApp1Run 函数，并执行函数体内的程序。

```
//运行视觉综合测试
void RobotAppDlg::OnBnClickedEditApp1Run()
{
    // TODO: 在此添加控件通知处理程序代码
    double zSafe = 0;
    double rolloffet = 0;
    int safePointIndex = 0;
    int pickPointIndex = 0;
    int placePointIndex = 0;

    //获取配置参数
    CString temp;
    App1ZSafeEdit.GetWindowTextW(temp);
    zSafe = _ttof(temp);
    App1RollOffsetEdit.GetWindowTextW(temp);
    rolloffet = _ttof(temp);
    safePointIndex = App1SafeCombo.GetCurSel();
    pickPointIndex = App1PickCombo.GetCurSel();
    placePointIndex = App1PlaceCombo.GetCurSel();

    //参数校验
    if ((RobotCtrl == nullptr) || (!RobotCtrl->IsConnected))
    {
```

图 3-10 运行按钮触发的函数体

3.2.3 码垛运动流程

1. 使用界面介绍

码垛功能

码盘

码盘

码盘

码垛结 行 列

叠料位

物料厚 mm

安全距 mm

物料数

示例: 3行5

Y ↑
↑
↑
↑
↑
X →

码盘A					
码盘B				码盘C	

图 3-11 码垛界面

界面介绍:

码盘: 需要选择 3 个码盘位置, 通过 3 点法计算出其它码盘的点位, 3 个点位位置分别为 A, B, C

码盘结构: 4 行 2 列 (与码盘选择的顺序有关)

叠料位: 叠料位是装垛时, 产品的汇聚点, 也是拆垛的拆分点

物理厚度: 决定了每次装垛抬升的距离和每次拆垛下降的距离

物料数: 决定了执行装垛或拆垛的次数, 注意物料数最多为 7, 因为要减去叠料位。

装垛: 将 Tray 盘的其它位置物料, 汇聚到叠料位

拆垛: 将叠料位的物料拆分到其它 Tray 盘位

```
//装垛
void RobotAppDlg::OnBnClickedEditApp3Run2()
{
    // TODO: 在此添加控件通知处理程序代码

    try
    {
        CString temp;
        App3RowEdit.GetWindowTextW(temp);
        size_t row = _ttoi(temp); //行
        App3ColEdit.GetWindowTextW(temp);
        size_t col = _ttoi(temp); //列
        App3ThickEdit.GetWindowTextW(temp);
        double thick = _ttof(temp); //厚度
        App3SafeEdit.GetWindowTextW(temp);
        double safe = _ttof(temp); //安全距离
        App3ItemCountEdit.GetWindowTextW(temp);
        int itemCount = _ttoi(temp); //物料数量

        if (itemCount == 0)
    }
```

图 3-12 装垛调用函数

```
//拆垛
void RobotAppDlg::OnBnClickedEditApp3Run1()
{
    // TODO: 在此添加控件通知处理程序代码

    try
    {
        CString temp;
        App3RowEdit.GetWindowTextW(temp);
        size_t row = _ttoi(temp); //行
        App3ColEdit.GetWindowTextW(temp);
        size_t col = _ttoi(temp); //列
        App3ThickEdit.GetWindowTextW(temp);
        double thick = _ttof(temp); //厚度
        App3SafeEdit.GetWindowTextW(temp);
        double safe = _ttof(temp); //安全距离
        App3ItemCountEdit.GetWindowTextW(temp);
        int itemCount = _ttoi(temp); //物料数量

        if ((row <= 1) || (col <= 1))
    }
```

图 3-13 拆垛调用函数

3.2.4 避障流程

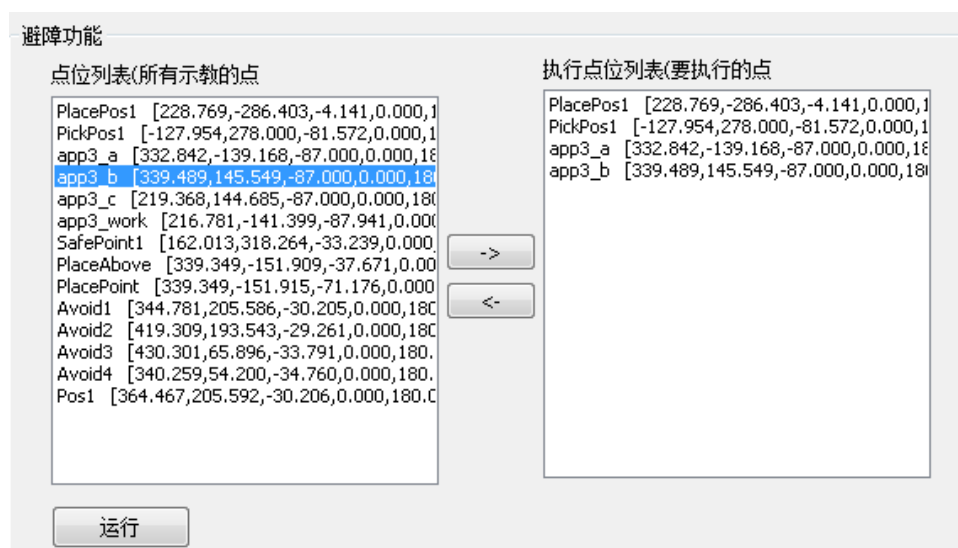


图 3-14 避障界面

- ① “→” 向右的箭头，可以将左边选中的点移动到右边执行列表中
- ② “←” 向左的箭头，可以将右边执行列表中的点位去除
- ③ 运行按钮，会按照右边执行列表中的点位依次执行下去，完成连续的轨迹运动。如图所示，右边的执行栏内有四个点，如果点击运行按钮，则机器人会依次去到“PlacePos1”→“PickPos1”→“app3_a”→“app3_b”。

```
//运行
void RobotAppDlg::OnBnClickedEditApp2Run()
{
    // TODO: 在此添加控件通知处理程序代码
    try
    {
        size_t count = App2RunPointList.GetCount();
        vector<PointType> points;

        //获取所有要运动的点位
        for (size_t i = 0; i < count; i++)
        {
            CString tmp;
            App2RunPointList.GetText(i, tmp);
            vector<CString> cmd = SplitCString(tmp, L" ");
            if (cmd.size() > 0)
            {

```

图 3-15 避障运行按钮调用的函数

第4章 运动学算法调用接口

4.1 概述

在第三章的内容中，我们介绍了运用轨迹规划功能，执行的点位动作，那么本章以功能一：轨迹规划为例说明调用的接口函数及代码。

运动学的相关代码放在 **MotionPlan** 文件内，展开如下图所示

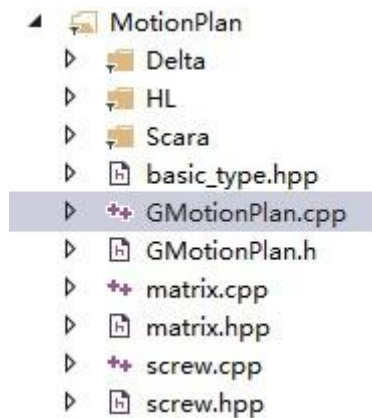


图 4-1 运动学相关代码

模块介绍列表：

名称	介绍
Delta	里面包含了 Delta 并联机器人轨迹规划代码
HL	里面包含了 HL 六轴机器人轨迹规划代码
SCARA	里面包含了 SCARA 四轴机器人轨迹规划代码
GMotionPlan	运动学规划调用的入口函数
Matrix	矩阵计算的方法
screw	欧拉角等相关计算方法

4.2 轨迹规划调用流程

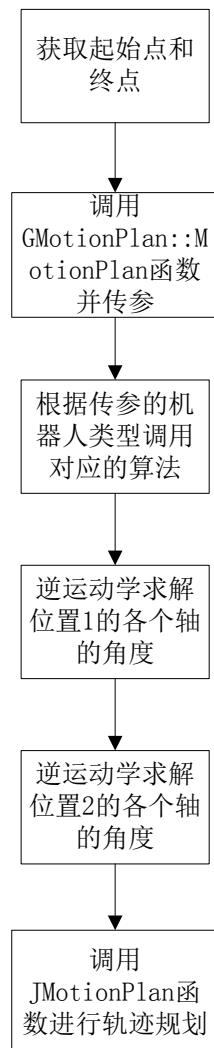


图 4-2 执行轨迹规划的流程

调用轨迹规划的函数前需要告知起始点和终点，并传参机器人的类型，最终会生成 **TXT** 文件保存规划的数据。

1. 获取起始点和终点

起始点和终点通过示教的方式获取，如果之前有示教过，点击界面的刷新即可显示可用点位

起始点位

刷新

终止点位

PlacePos1	[228.769,-286.403,-4.141,0.000,180.000,180.000]	PlacePos1	[228.769,-286.403,-4.141,0.000,180.000,180.000]
PickPos1	[-127.954,278.000,-81.572,0.000,180.000,180.000]	PickPos1	[-127.954,278.000,-81.572,0.000,180.000,180.000]
app3_a	[332.842,-139.168,-87.000,0.000,180.000,180.000]	app3_a	[332.842,-139.168,-87.000,0.000,180.000,180.000]
app3_b	[339.489,145.549,-87.000,0.000,180.000,180.000]	app3_b	[339.489,145.549,-87.000,0.000,180.000,180.000]
app3_c	[219.368,144.685,-87.000,0.000,180.000,180.000]	app3_c	[219.368,144.685,-87.000,0.000,180.000,180.000]
app3_work	[216.781,-141.399,-87.941,0.000,180.000,180.000]	app3_work	[216.781,-141.399,-87.941,0.000,180.000,180.000]
SafePoint1	[162.013,318.264,-33.239,0.000,180.000,180.000]	SafePoint1	[162.013,318.264,-33.239,0.000,180.000,180.000]
PlaceAbove	[339.349,-151.909,-37.671,0.000,180.000,180.000]	PlaceAbove	[339.349,-151.909,-37.671,0.000,180.000,180.000]
PlacePoint	[339.349,-151.915,-71.176,0.000,180.000,180.000]	PlacePoint	[339.349,-151.915,-71.176,0.000,180.000,180.000]
Avoid1	[344.781,205.586,-30.205,0.000,180.000,180.000]	Avoid1	[344.781,205.586,-30.205,0.000,180.000,180.000]
Avoid2	[419.309,193.543,-29.261,0.000,180.000,180.000]	Avoid2	[419.309,193.543,-29.261,0.000,180.000,180.000]
Avoid3	[430.301,65.896,-33.791,0.000,180.000,180.000]	Avoid3	[430.301,65.896,-33.791,0.000,180.000,180.000]
Avoid4	[340.259,54.200,-34.760,0.000,180.000,180.000]	Avoid4	[340.259,54.200,-34.760,0.000,180.000,180.000]
Pos1	[364.467,205.592,-30.206,0.000,180.000,180.000]	Pos1	[364.467,205.592,-30.206,0.000,180.000,180.000]

	X	Y	Z	Yaw	Pitch	Roll	config
点位1	-127.954	278.000	-81.572	0.000	180.000	140.603	1
点位2	228.769	-286.403	-4.141	0.000	180.000	-44.252	1

笛卡尔运算

图 4-3 点位选择

当点击“笛卡尔运算”时则会调用函数获取点位一和点位二的数据，并存入中间变量。

```

1. double p1x, p1y, p1z, p1Yaw, p1Pitch, p1Roll;
2. double p2x, p2y, p2z, p2Yaw, p2Pitch, p2Roll;
3.
4. //获取点位 1
5. Point1X.GetWindowTextW(tmp);
6. p1x = _ttof(tmp);
7. Point1Y.GetWindowTextW(tmp);
8. p1y = _ttof(tmp);
9. Point1Z.GetWindowTextW(tmp);
10. p1z = _ttof(tmp);
11. Point1Yaw.GetWindowTextW(tmp);
12. p1Yaw = _ttof(tmp);
13. Point1Pitch.GetWindowTextW(tmp);
14. p1Pitch = _ttof(tmp);
15. Point1Roll.GetWindowTextW(tmp);
16. p1Roll = _ttof(tmp);
17. Point1Config.GetWindowTextW(tmp);
18. int config1 = _ttoi(tmp);
19.
20. //获取点位 2
21. Point2X.GetWindowTextW(tmp);
22. p2x = _ttof(tmp);
23. Point2Y.GetWindowTextW(tmp);
24. p2y = _ttof(tmp);

```

```

25. Point2Z.GetWindowTextW(tmp);
26. p2z = _ttof(tmp);
27. Point2Yaw.GetWindowTextW(tmp);
28. p2Yaw = _ttof(tmp);
29. Point2Pitch.GetWindowTextW(tmp);
30. p2Pitch = _ttof(tmp);
31. Point2Roll.GetWindowTextW(tmp);
32. p2Roll = _ttof(tmp);
33. Point2Config.GetWindowTextW(tmp);
34. int config2 = _ttoi(tmp);

```

2. 调用 GMotionPlan::MotionPlan 函数并传参

获得点位一和点位二，已经机器人的类型后，调用 MotionPlan 执行规划。

```

1. //进行轨迹规划
2. QkmMotionPlan::GMotionPlan::MotionPlan(robotType, vel, acc, dec, sampleTime,
    p1x, p1y, p1z, p1Yaw, p1Pitch, p1Roll, config1,
    p2x, p2y, p2z, p2Yaw, p2Pitch, p2Roll, config2,
    "data.txt");

```

3. 根据传参的机器人类型调用对应的算法

Switch 语句下有 3 种 Case 语句构成，分别是 ScaraRobot、DeltaRobot、HlRobot。根据传参的类型调用对应的函数。

```

void GMotionPlan::MotionPlan(RobotType robotType, double vel, double acc, double dec, double sampleTime,
double x1, double y1, double z1, double yaw1, double pitch1, double roll1, int config1,
double x2, double y2, double z2, double yaw2, double pitch2, double roll2, int config2,
std::string fileName)
{
    switch (robotType)
    {
        case ScaraRobot:
        {
            ...
        }
        case DelataRobot:
        {
            ...
        }
        case HlRobot:
        {
            ...
        }
        default:
            break;
    }
}

```

图 4-4 根据传参类型调用对应的函数

4. 逆运动学求解位置 1 的各个轴的角度

下面以 Scara 为例介绍。通过逆运动学获得的对应的关节角度

```
1. case ScaraRobot:
2.     {
3.         double j11, j12, j13, j14;
4.         double j21, j22, j23, j24;
5.
6.         //选择 scara 机器人
7.         ScaraInv mAH500Inv(250, 250);
8.
9.         std::vector<double> pos1JointVal;
10.        std::vector<double> pos2JointVal;
11.
12.        //通过逆运动学求解位置 1 的各个轴的角度
13.        mAH500Inv.SetRobotEndPos(x1, y1, z1, roll1);
14.        mAH500Inv.GetRobotAngle(j11, j12, j13, j14, config1);
```

5. 逆运动学求解位置 2 的各个轴的角度

```
1. //通过逆运动学求解位置 2 的各个轴的角度
2.     mAH500Inv.SetRobotEndPos(x2, y2, z2, roll2);
3.     mAH500Inv.GetRobotAngle(j21, j22, j23, j24, config2);
```

6. 调用 JMotionPlan 函数进行轨迹规划

将逆解得到的起始点和终点的关节角度传入 JMotionPlan 进行轨迹规划。

```
1. JMotionPlan(robotType, vel, acc, dec, sampleTime,
2.     j11, j12, j13, j14, -1, -1, config1,
3.     j21, j22, j23, j24, -1, -1, config2,
4.     fileName);
```

下面继续以 Scara 为例，介绍 JMotionPlan 函数，一共分为五步

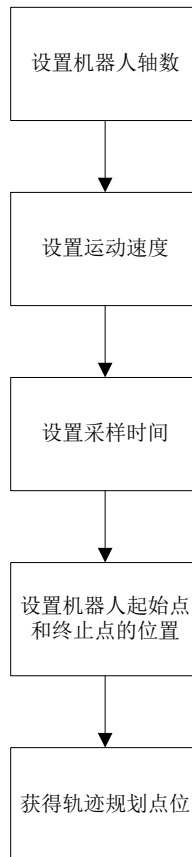


图 4-5 JMotionPlan 执行流程

```
1. void GMotionPlan::JMotionPlan(RobotType robotType, double vel, double a
   cc, double dec, double sampleTime,
2.     double j11, double j12, double j13, double j14, double j15, double
   j16, int config1,
3.     double j21, double j22, double j23, double j24, double j25, double
   j26, int config2,
4.     std::string fileName)
5. {
6.     std::vector<double> pos1JointVal;
7.     std::vector<double> pos2JointVal;
8.
9.     switch (robotType)
10.    {
11.    case ScaraRobot:
12.    {
13.        std::vector<std::vector<double>> planPos;
14.        std::vector<std::vector<double>> planVel;
15.        std::vector<std::vector<double>> planAcc;
16.
17.        ScaraMotionPlan mAH500Plan;
```

```

18.
19.     pos1JointVal.push_back(j11);
20.     pos1JointVal.push_back(j12);
21.     pos1JointVal.push_back(j13);
22.     pos1JointVal.push_back(j14);
23.
24.     pos2JointVal.push_back(j21);
25.     pos2JointVal.push_back(j22);
26.     pos2JointVal.push_back(j23);
27.     pos2JointVal.push_back(j24);
28.
29.     //轨迹规划
30.     mAH500Plan.SetRobotType(4, 0);
31.     mAH500Plan.SetProfile(vel, acc, dec);
32.     mAH500Plan.SetSampleTime(sampleTime / 1000.0);    //设置采样
        间隔, 要≥0.001S
33.     mAH500Plan.SetPosJointVal(pos1JointVal, pos2JointVal);
34.     mAH500Plan.GetPlanPoints(fileName, planPos, planVel, planAcc);
35.     break;
36. }

```

1) 设置机器人轴数

mAH500Plan.SetRobotType(4, 0)设置机器人的轴数

```

1.  /*****
2.  ABSTRACT:   设置机器人轴的个数
3.
4.  INPUTS:     num           轴的个数
5.  type        0: 串联,   1: 并联
6.
7.  OUTPUTS:    <none>
8.
9.  RETURN:     <none>
10. *****/
11. /
12. void ScaraMotionPlan::SetRobotType(int num, int type)
13. {
14.     if (num >= 4)
15.     {
16.         mAxisNum = num;
17.         mRbtType = type;

```

```

17.     for (int i = 0; i < mAxisNum; i++)
18.     {
19.         mStartPosJointVal.push_back(0);
20.         mEndPosJointVal.push_back(0);
21.         mTotalCnt.push_back(0);
22.     }
23. }
24. }

```

2) 设置运动速度

mAH500Plan.SetProfile(vel, acc, dec)设置运行速度

```

1.  /*****
2.  ABSTRACT:   设置机器人的运动参数
3.
4.  INPUTS:    vel           速度 (m/s)
5.  acc        加速度 (m/s/s)
6.  dec        减速度 (m/s/s)
7.
8.  OUTPUTS:   <none>
9.
10. RETURN:    <none>
11. *****/
12. void ScaraMotionPlan::SetProfile(double vel, double acc, double dec)
13. {
14.     mVel = vel;
15.     mAcc = acc;
16.     mDec = dec;
17. }

```

3) 设置采样时间

mAH500Plan.SetSampleTime(sampleTime / 1000.0); //设置采样间隔，要 $\geq 0.001S$

```

1.  /*****
2.  ABSTRACT:   设置采样间隔周期
3.
4.  INPUTS:    sampleTime    采样间隔周期，单位 S(>=0.001)
5.
6.  OUTPUTS:   <none>
7.

```

```

8. RETURN:      <none>
9. *****/
10. void ScaraMotionPlan::SetSampleTime(double sampleTime)
11. {
12.     // 如果用户设置小于 1ms, 强制变成 1ms
13.     if (sampleTime < 0.001)
14.     {
15.         mSampleTime = 0.001;
16.     }
17.     else
18.     {
19.         mSampleTime = sampleTime;
20.     }
21. }

```

4) 设置机器人起始点和终止点的位置

mAH500Plan.SetPosJointVal(pos1JointVal, pos2JointVal)

```

1. /***/
2. ABSTRACT:    设置机器人起始点和终止点的位置（关节角度）
3.
4. INPUTS:      startPos          起始点位关节角度，单位为度
5. endPos          终止点位关节角度，单位为度
6.
7. OUTPUTS:     <none>
8.
9. RETURN:      0                  设置成功
10. -1           设置失败
11. *****/
12. int ScaraMotionPlan::SetPosJointVal(vector<double>startPos, vector<double>endPos)
13. {
14.     if (startPos.size() == mAxisNum)
15.     {
16.         for (int i = 0; i < mAxisNum; i++)
17.         {
18.             // 串联四轴机器人第三个轴为直线轴，转换成米而不是度
19.             if ((i == 2) && (mAxisNum == 4) && (mRbtType == 0))
20.             {
21.                 mStartPosJointVal[i] = startPos[i] / 1000;

```

```

22.             mEndPosJointVal[i] = endPos[i] / 1000;
23.         }
24.     else
25.     {
26.         mStartPosJointVal[i] = startPos[i] * PI / 180;
27.         mEndPosJointVal[i] = endPos[i] * PI / 180;
28.     }
29.
30.     }
31. }
32. else
33.     return -1;
34.
35. return 0;
36. }

```

5) 获得轨迹规划点位

mAH500Plan.GetPlanPoints(fileName, planPos, planVel, planAcc)

```

1.  /*****
2.  ABSTRACT:   运动轨迹规划部分
3.
4.  INPUTS:    pos           二维位置向量
5.  vel        二维速度向量
6.  acc        二维加速度向量
7.
8.  OUTPUTS:   pos           二维位置向量（第一维：位置个数，第二维：
   每个轴的关节角度，单位弧度）
9.  vel        二维速度向量（第一维：速度个数，第二维：每个轴的速度，单
   位 m/s）
10. acc        二维加速度向量（第一维：加速度个数，第二维：每个轴的
   加速度， 单位 m/s/s）
11.
12. RETURN:    <none>
13. *****/
14. void ScaraMotionPlan::GetPlanPoints(string fileName, vector<std::vector
   <double>>> &pos, vector<std::vector<double>>> &vel, vector<std::vector<do
   ouble>>> &acc)
15. {
16.     double p, v, a;
17.     double cnt;

```

```

18.     int index;
19.     ofstream outfile;//创建文件
20.
21.     outfile.open(fileName);
22.
23.     mCount = 1;
24.     cnt = 0;
25.     index = 0;
26.
27.     for (int i = 0; i < mAxisNum; ++i)
28.     {
29.         //梯形轨迹规划//
30.         MoveAbsolute(mCount, mStartPosJointVal[i], mEndPosJointVal[i]
31.             , mVel * mSampleTime, mAcc *mSampleTime *mSampleTime, mDec
32.             *mSampleTime *mSampleTime
33.             , p, v, a, mTotalCnt[i]);//预先获取每个关节运动所需的 count
34.         数
35.     }
36.
37.     std::vector<Size> total_count_vec;
38.     total_count_vec.resize(mAxisNum, 0);
39.     for (int i = 0; i < mAxisNum; ++i)
40.         total_count_vec[i] = mTotalCnt[i];
41.     mMaxTotalCnt = *std::max_element(total_count_vec.begin(), total_cou
42.         nt_vec.end());//获取最多的 count 数
43.
44.     for (int i = 0; i < mMaxTotalCnt; i++)
45.     {
46.         pos.push_back(vector <double>());
47.         vel.push_back(vector <double>());
48.         acc.push_back(vector <double>());
49.     }
50.     for (int i = 0; i < mMaxTotalCnt; i++)
51.     {
52.         for (int j = 0; j < mAxisNum; j++)
53.         {
54.             pos[i].push_back(0);
55.             vel[i].push_back(0);
56.             acc[i].push_back(0);
57.         }
58.     }
59.
60.     while (mMaxTotalCnt > mCount)

```

```

58.     {
59.         mTime = mCount * mSampleTime;//时间
60.                                     //outfile << mTime << ", ";
61.         for (int i = 0; i < mAxisNum; ++i)
62.         {
63.             cnt = static_cast<double>(mCount) * mTotalCnt[i] / mMaxTotalCnt;
64.             //梯形轨迹规划//
65.             MoveAbsolute(cnt,
66.                 mStartPosJointVal[i], mEndPosJointVal[i],
67.                 mVel * mSampleTime, mAcc * mSampleTime * mSampleTime, mDec
68.                 * mSampleTime * mSampleTime,
69.                 p, v, a, mTotalCnt[i]); //每 1 毫秒更新位置指令 p, 保证所有关节同步启动同步停止, 把位置 p 发送给机器人各关节即可
70.                                     // 第一个点位是不对的
71.             if (mCount > 1)
72.             {
73.                 if ((i == 2) && (mAxisNum == 4) && (mRbtType == 0))
74.                 {
75.                     pos[mCount][i] = p * 1000; // S
76.                     CARA 第三轴为直线轴, 转成 mm
77.                 }
78.                 else
79.                 {
80.                     pos[mCount][i] = p * 180 / PI; // 由弧度转成度
81.                 }
82.                 vel[mCount][i] = v;
83.                 acc[mCount][i] = a;
84.                 outfile << pos[mCount][i] << " "; //保存数据, v 的单位 m/ms, a 的单位 m/ms/ms
85.             }
86.         }
87.         if ((mCount != 1) && ((mCount + 1) < mMaxTotalCnt))
88.         {
89.             outfile << endl;
90.         }
91.         mCount++;

```

第5章 力矩数据计算

5.1 总体概述

本章节主要是讲解如何在机器人运动过程中采集数据，对采集的数据使用公式换算得出控制卡输出控制电机的电流和反馈电流，再计算出输出力矩和反馈力矩。

5.1.1 操作流程框图

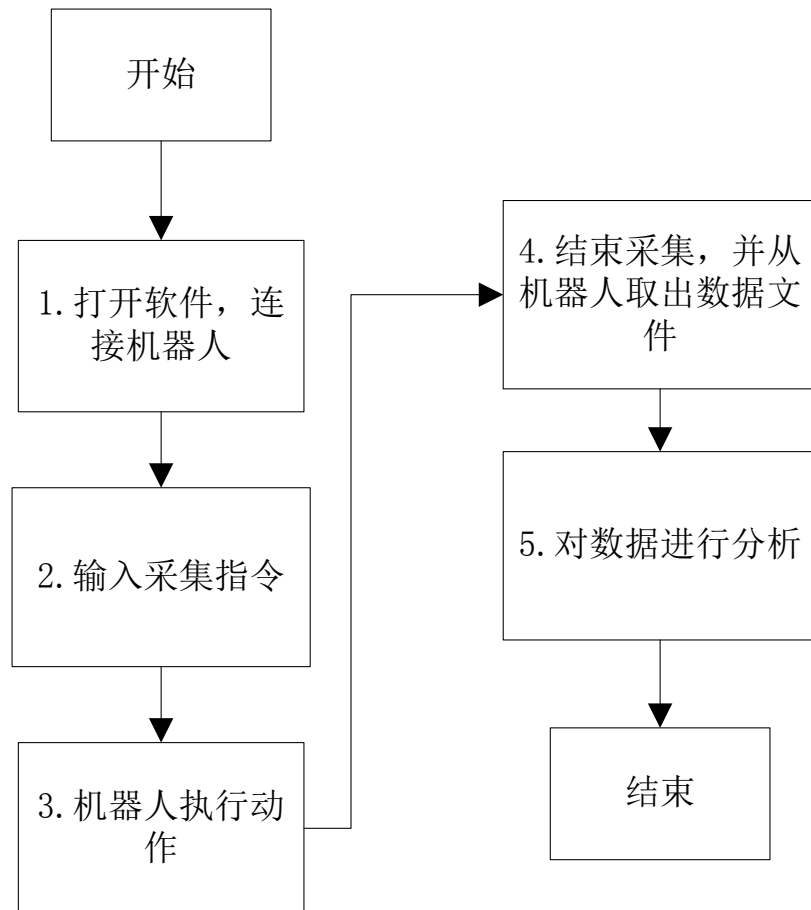


图 5-1 操作流程框图

5.1.2 公式介绍

1. 换算电流公式:

电流=采集的数据/ 15000 * 通道值;

2. 力矩计算公式

力矩=电流值*力矩常数*减速比*传动效率;

3. 丝杆推力计算

由于丝杆的特殊性只有推力, 不是力矩

推力=2 π *电流值*力矩常数*减速比*传动效率/丝杆导程;

5.1.3 指令介绍

1. log_cmd

log_cmd 1 2 0 ---→ 1 代表频率为 32K 即采样周 31.25us ;

log_cmd 2 2 0 ---→ 2 代表频率为 16K 即采样周期 62.5us;

log_cmd 4 2 0 ---→ 4 代表频率为 8K 即采样周 125us ;

log_cmd 1 2 0 ---→ 2 代表抓取的数据个数是两个;

0 代表驱动器 cell 结束数据抓取动作;

log_cmd 1 2 1 ---→ 1 代表 驱动器 cell 开始数据抓取动作;

2. P-0-0008.0.20 电流指令数据

P-0-0008.0.20 ---→ 0 代表的是 Cell1 控制的机器人 J1 轴的电流;

P-0-0008.1.20 ---→ 1 代表的是 Cell1 控制的机器人 J2 轴的电流;

若 Putty 连接的是 cell1 即 192.168.10.105 则

P-0-0008.0.20 ---→ 0 代表的是 Cell2 控制的机器人 J3 轴的电流;

P-0-0008.1.20 ---→ 1 代表的是 Cell2 控制的机器人 J4 轴的电流;

3. P-0-0008.0.18 电流反馈数据

P-0-0008.0.18 ---→ 0 代表的是 Cell1 控制的机器人 J1 轴的电流;

P-0-0008.1.18 ---→ 1 代表的是 Cell1 控制的机器人 J2 轴的电流;

若 Putty 连接的是 cell1 即 192.168.10.105 则

P-0-0008.0.18 ---→ 0 代表的是 Cell2 控制的机器人 J3 轴的电流;

P-0-0008.1.18 ---→ 1 代表的是 Cell2 控制的机器人 J4 轴的电流;

4. log_idn

log_idn "P-0-0008.0.20" "P-0-0008.0.18" 0

0 表示抓取从第 0 个数据开始，此处即从 P-0-0008.0.20 开始

5.2 连接机器人

5.2.1 打开 Putty 软件

在桌面上找到如下图的软件图标，打开

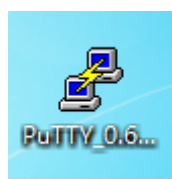


图 5-2 软件图标

5.2.2 连接设置

设置界面如下图所示

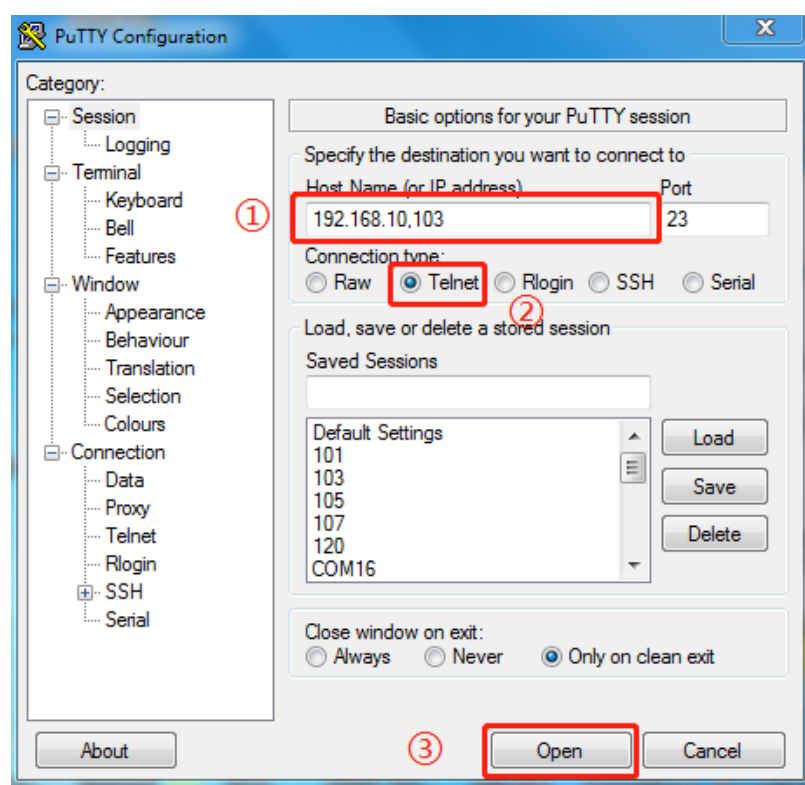


图 5-3 连接设置

①在 Host name(or IP address) 内输入 192.168.10.103;

说明：**192.168.10.103** 是机器人驱动器 **Cell1** 的 ip 地址，本机器人的一个驱动器驱动两个电机，即驱动器 **Cell1** 驱动的是机器人的第一和第二轴。关于本机器人驱动器更详细的说明请查看《用户手册》

【备注：必须先将操作的电脑 IP 更改为同一网段，例如为：192.168.10.10】

②Connection type 选择 Telnet;

③点击 open，以连接 Cell1;

完成上述三个步骤成功连接机器人驱动器如下图所示：

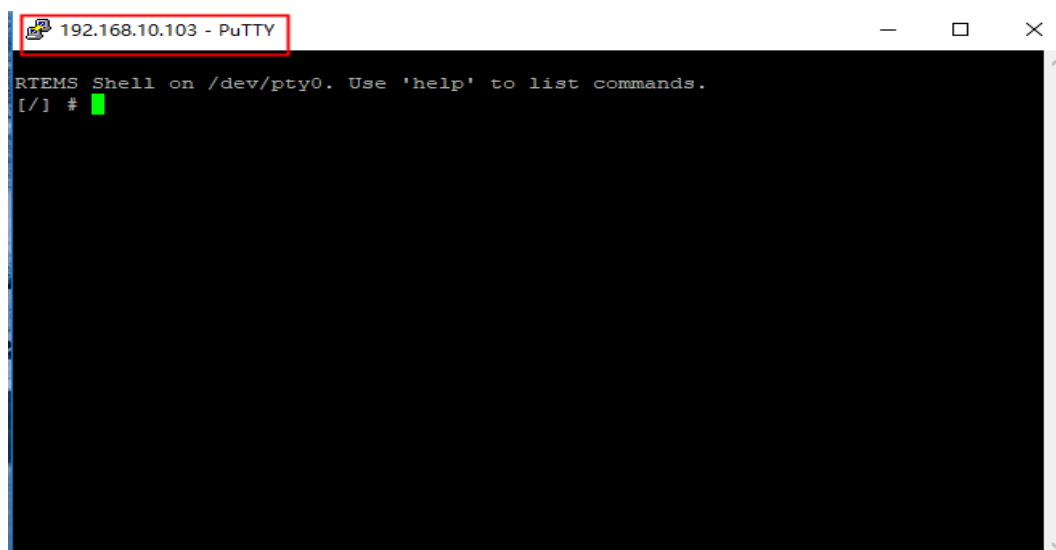


图 5-4 命令输入界面

5.3 采集数据设置

在 Putty 输入如下指令,如下图所示

```
192.168.10.103 - PuTTY
RT-DEMS Shell on /dev/ptyl. Use 'help' to list commands.
[/] # log_cmd 1 2 0
[/] # log_idn "P-0-0008.0.20" "P-0-0008.0.18" 0
The value is: 114.
The type is: INT16.
The address of this data is: 0x3495f02.
The value is: 109.
The type is: INT16.
The address of this data is: 0x3495efe.
[/] # log_cmd 1 2 1
[/] #
[/] # log_cmd 1 2 0
[/] #
```

图 5-5 指令输入

设置采样两组数据，并且从第 0 个开始。两组数据分别是第一轴的电流指令和第一轴反馈电流。

5.4 采集数据

1. 编写运动程序，并执行运动

【备注：此程序只做为教程范例且只适用四轴 SCARA 机器人，用户可根据实际需求编写程序】

```
指令输入
1 System.ClearVariables;
2 System.Speed 100;
3 LocationJ p1=90,0,10,0;
4 LocationJ p2=-90,0,10,0;
5
6 Move.Joint p1;
7 WaitTime 500;
8 Move.Joint p2;
9 WaitTime 500;
10
11 Move.Joint p1;
12 WaitTime 500;
13 Move.Joint p2;
14 WaitTime 500;
15
16 Move.Joint p1;
17 WaitTime 500;
18 Move.Joint p2;
19 WaitTime 500;
```

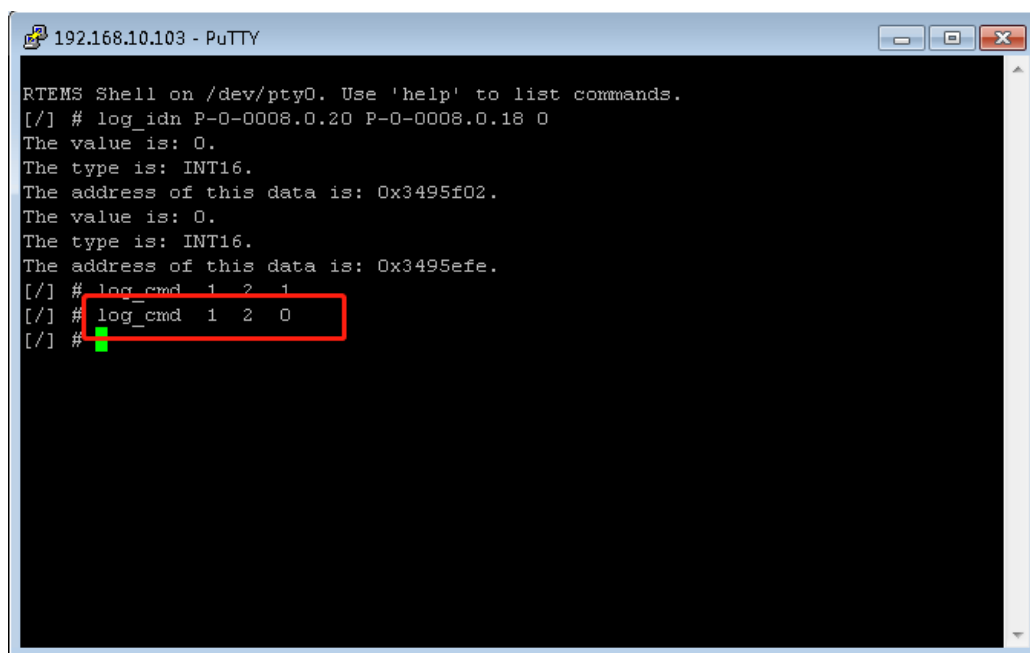
2. 开始采集

输入 log_cmd 1 2 1, 开始采集数据

```
192.168.10.103 - PuTTY
RTMS Shell on /dev/pty0. Use 'help' to list commands.
[/] # log_idn P-0-0008.0.20 P-0-0008.0.18 0
The value is: 0.
The type is: INT16.
The address of this data is: 0x3495f02.
The value is: 0.
The type is: INT16.
The address of this data is: 0x3495efe.
[/] # log_cmd 1 2 1
[/] #
```

3. 结束采集

等待运动结束后, 输入 log_cmd 1 2 0 停止采集



```
192.168.10.103 - PuTTY
RTMS Shell on /dev/pty0. Use 'help' to list commands.
[/] # log_idn P-0-0008.0.20 P-0-0008.0.18 0
The value is: 0.
The type is: INT16.
The address of this data is: 0x3495f02.
The value is: 0.
The type is: INT16.
The address of this data is: 0x3495efe.
[/] # log_cmd 1 2 1
[/] # log_cmd 1 2 0
[/] #
```

5.5 数据导出

如下所示，输入 <ftp://192.168.10.103/data>，并按下回车，可以看到生成的波形数据为 `logged_data.txt` 文本文件，将其复制至本地；

【备注:请勿直接在 <ftp://192.168.10.103/data> 内打 `logged_data.txt`】

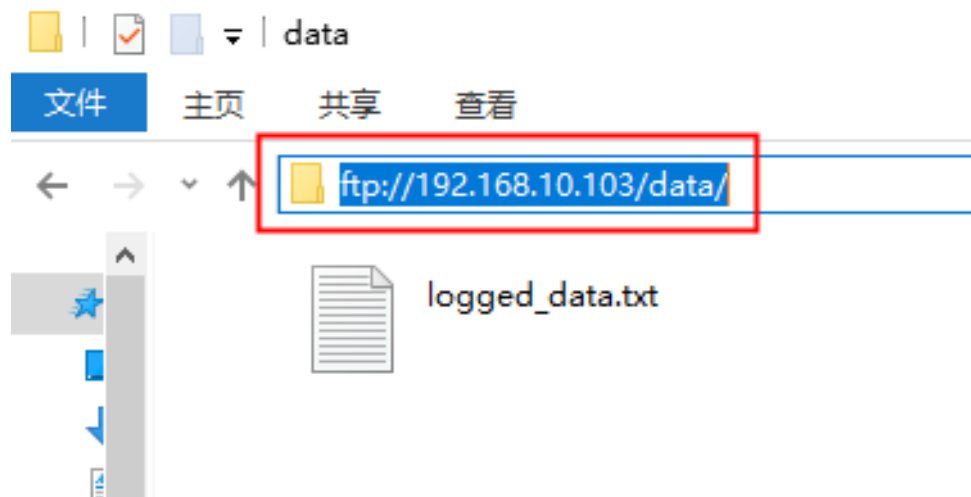


图 5-6 采集数据

5.6 数据分析

根据公式：

$$\text{采集数据} / 15000 * \text{通道值} = \text{电流 A}$$

需要两个数据-采集数据和通道值，带入公式即可。

5.6.1 采集数据获取

打开存在 logged_data.txt 本地的文件，如下图所示：

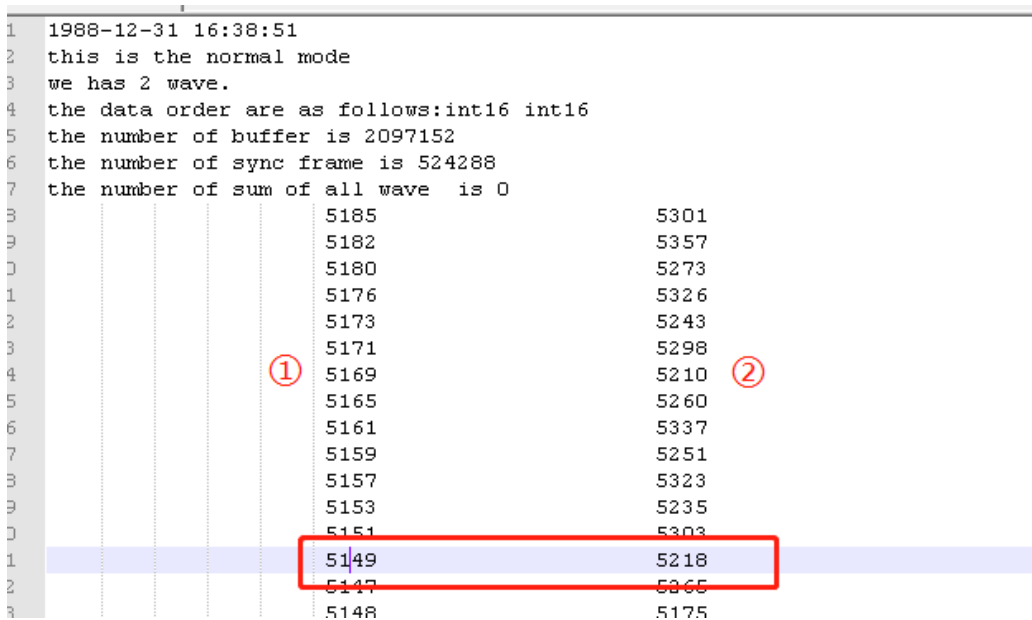


图 5-7 文件数据

上图中的①列为第一个数据（电流指令值），②列为第二个数据（电流反馈值）。下面进行公式计算时将会使用红色方框内的数据：

5.6.2 通道值获取

进入机器人驱动器 cell1 内，将电机参数配置文件拷出到本地

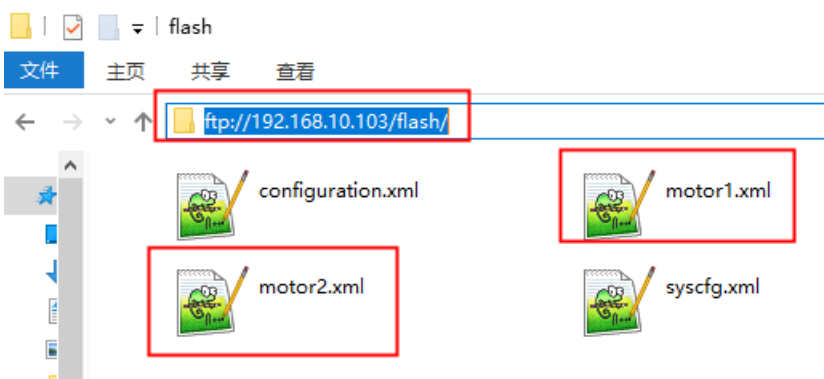


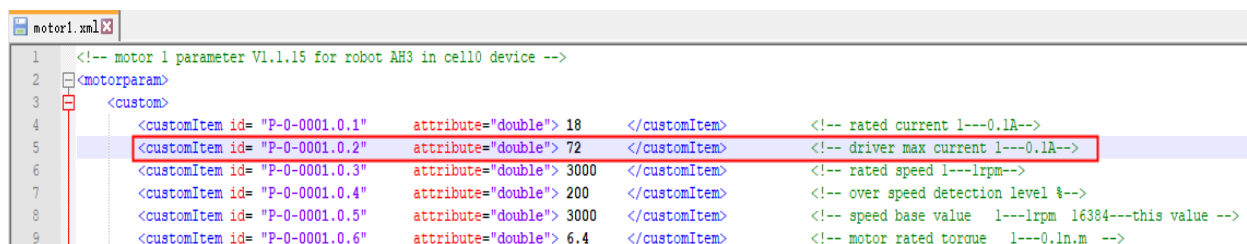
图 5-8 电机配置文件夹

备注：

驱动器 Cell0 控制的 J1、J2 轴，电机配置参数请从地址
<ftp://192.168.10.103/flash/> 获取：

驱动器 Cell1 控制的 J3、J4 轴，电机配置参数请从地址
<ftp://192.168.10.105/flash/> 获取：

打开配置文件如下图所示



```
1 <!-- motor 1 parameter V1.1.15 for robot AH3 in cell0 device -->
2 <motorparam>
3   <custom>
4     <customItem id= "P-0-0001.0.1" attribute="double"> 18 </customItem> <!-- rated current 1---0.1A-->
5     <customItem id= "P-0-0001.0.2" attribute="double"> 72 </customItem> <!-- driver max current 1---0.1A-->
6     <customItem id= "P-0-0001.0.3" attribute="double"> 3000 </customItem> <!-- rated speed 1---1rpm-->
7     <customItem id= "P-0-0001.0.4" attribute="double"> 200 </customItem> <!-- over speed detection level $-->
8     <customItem id= "P-0-0001.0.5" attribute="double"> 3000 </customItem> <!-- speed base value 1---1rpm 16384---this value -->
9     <customItem id= "P-0-0001.0.6" attribute="double"> 6.4 </customItem> <!-- motor rated torque 1---0.1n.m -->
```

由于单位是 0.1A，所以实际计算通道值是要除以 10，带入计算的通道值为 7.2

5.6.3 公式计算

1. 电流值计算

将获取的数据（图 1-7 红色方框内）带入公式计算出实际的输出电流和电流反馈：

输出电流： $5149 / 15000 * 7.2 \approx 2.471 \text{ A}$ ；

反馈电流： $5218 / 15000 * 7.2 \approx 2.504 \text{ A}$ ；

2. 力矩计算

上一节已经计算出机器人在某一时刻的电流值，本次实验是以 AH 机器人的一轴为例，查阅附录 1 找到对应的机器人参数表，将力矩常数和减速比，带入力矩计算公式：

输出指令力矩： $2.471 * 0.4 * 80 * 0.9 = 71.1648 \text{ N}\cdot\text{m}$ ；

实际输出力矩： $2.504 * 0.4 * 80 * 0.9 = 72.1152 \text{ N}\cdot\text{m}$ ；

第6章 注意事项

6.1 问题及解决办法

1. 视觉无法打开，报错

a) 报错信息

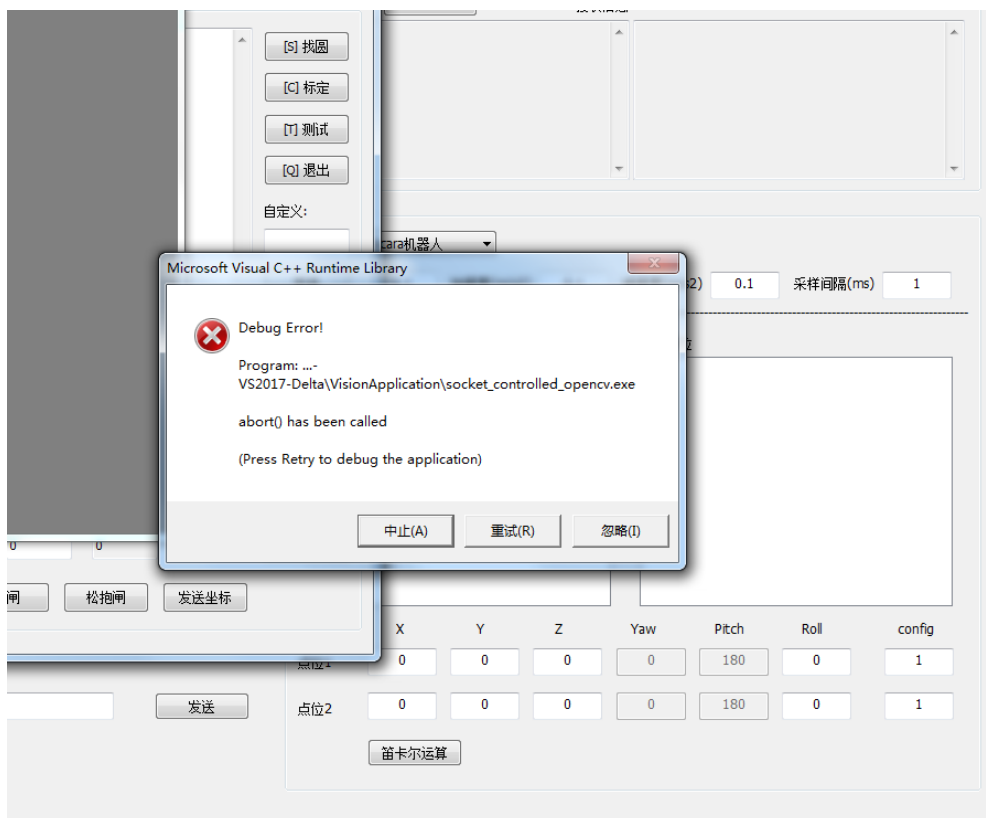


图 6-1 视觉报错信息

b) 原因

相机驱动安装失败，打开“控制面板”→“硬件和声音”→“设备和打印机”，此界面可以查看视觉安装驱动的信息，如果安装失败，如下图所示：“未知设备”。

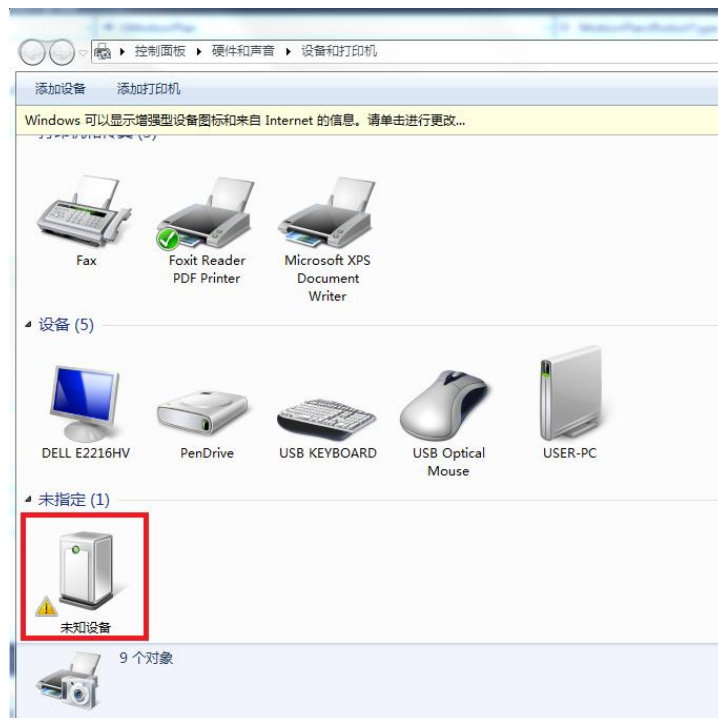


图 6-2 驱动安装失败

c) 解决办法

将相机的电源线驱动线拔掉重新插上，即可重新加载驱动



图 6-3 电源驱动线

安装 OK 的图如下：所示



图 6-4 相机驱动安装成功

1. C++运动学界面视觉光源触发时序不对

a) 问题

点击“获取视觉结果”，拍的照片是暗色的，但是光源有闪烁，只是时序不对。

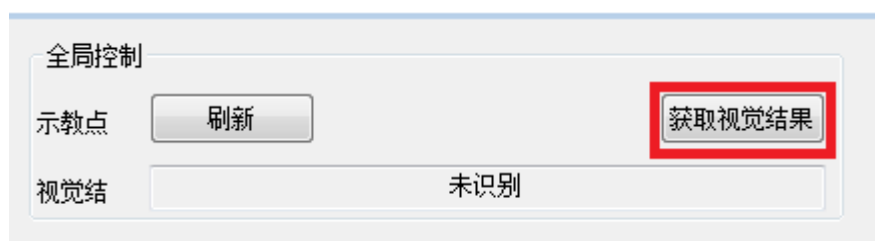


图 6-5 获取视觉结果

b) 原因

拍照的程序逻辑：“光源 IO 置为 True” → “拍照” → “光源 IO 置为 False”。一般 AH 机台不会出现时序不对问题，AP 机台有光源控制器会导致这种问题出现。

AP 的光源控制器有两种模式“REMO”和“MANU”，当开关处于 REMO 模式时，IO 置为 True，开启光源，如果开关置为“MANU”时，则反之。



图 6-6 模式选择

c) 解决办法

将开关置于 REMO 即可。

附录 A

ping-pong buffer 功能

指令 ID	关键字	描述
03850	PPB	ping-pong buffer
103851	PPB.Enable	进入和退出 ping-pong buffer 模式
103852	PPB. ReadFile	读取用户点位文件
103853	PPB. Prepare	机器人通过 Joint 运动到用户点位文件的第一个点
103854	PPB.Run	机器人开始执行 ping-pong buffer 中的点位
103855	PPB.Stop	机器人中断运动动作，执行减速停止
103856	PPB.Insert	宏指令方式向 ping-pong buffer 中插入点位
103857	PPB.FreeRoom	查询 ping-pong buffer 的剩余空间
103857	PPB.ClearBuffer	清理 ping-pong buffer 的剩余空间

103851 PPB.Enable

格式：

(1) **PPB. Enable** [robot_id] ,[mode]

参数：

参数 1：

robot_id：机器人 id

类型： Uint8

参数 2:

mode : mode = 0 表示退出 ping-pong buffer 模式。mode = 1 表示进入 ping-pong buffer 模式。

类型： Uint8

返回值：

返回执行结果:

描述：

进入和退出 ping-pong buffer 模式, 退出 ping-pong buffer 模式后会清除 buffer 内容并释放用户点位文件。如果机器人正在运动, 不能切换进入 ping-pong buffer 模式; 退出 ping-pong buffer 模式机器人会快速停止。

示例：

PPB.Enable 1,1

103852 PPB.ReadFile

格式：

(2) **PPB.ReadFile** [robot_id], [FileName1]

参数:

参数 1:

Robot_id : 机器人 id

类型： Uint8

参数 3:

FileName1 文件名

类型： 字符串

返回值：

返回读取结果:

描述：

读取用户点位文件。

示例：

PPB.ReadFile 1, cart_pos.txt

103853 PPB.JStartPos

格式：

(3) PPB. JStartPos [robot_id]

参数:

参数 1:

Robot_id：机器人 id

类型： Uint8

返回值：

返回执行结果:

描述：

机器人通过 Joint 运动到用户点位文件的第一个点。

示例：

PPB. JStartPos

103854 PPB.Run

格式：

(4) PPB. Run

参数:

参数 1:

Robot_id : 机器人 id

类型: Uint8

返回值 :

返回执行结果:

描述 :

机器人开始执行 ping-pong buffer 中的点位，用户点位文件执行结束后，清除 buffer 内容，释放用户点位文件。（执行该指令前必须先执行 JStartPos 到达起始点）。

示例 :

PPB.Run 1

103855 PPB.Stop

格式 :

(5) PPB. Stop

参数:

参数 1:

Robot_id : 机器人 id

类型: Uint8

返回值 :

返回执行结果:

描述：

机器人中断运动动作，执行减速停止，清除 **buffer** 内容并释放用户点位文件。

示例：

PPB.Stop 1

103855 PPB.Insert

格式：

(6) **PPB. Insert [Robot_id] ,[L1] ,[L2] ,...,[L20]**

参数：

参数 1:

Robot_id：机器人 id

类型： Uint8

参数 2:

L1:点位

类型:Location 或 LocationJ 类型

返回值：

返回执行结果:

描述：

向 ping-pong buffer 中压入数据

示例：

PPB.Insert 1,loc1,loc2,...loc20

103856 PPB.FreeRoom

格式：

(7)**PPB. FreeRoom [Robot_id]**

参数:

参数 1:

Robot_id : 机器人 id

类型: Uint8

返回值 :

Free_room: 剩余空间

类型: **Uint16**

描述 :

查询 ping-pong buffer 中剩余空间

示例 :

PPB. FreeRoom 1

103856 PPB.ClearBuffer

格式 :

(1) **PPB. ClearBuffer [Robot_id]**

参数:

参数 1:

Robot_id : 机器人 id

类型: Uint8

返回值 :

描述 :

清理 ping-pong buffer,机器人运动过程中不能执行该操作。

示例 :

PPB. ClearBuffer 1

附录 B

Ping-pong buffer 设计文档

1. Ping-pong buffer 的概念

Ping-pong buffer (PPB) 功能是，使用者 bypass Pallas 内部默认的运动 planner 和 trajectory generator，以实时网络通信周期 (e.g 1ms) 把指令讯息直接给到 servo，然后执行 robot 动作。或者，用户给出一些关键点位，在底层进行样条曲线拟合，轨迹规划，最后将实时指令讯息给到伺服。

2. 前提条件

序号	条件
1	robot 在自动模式下，静止状态下，才可以进入 PPB 模式，进入后分配内存（大小可配），退出时释放内存
2	PPB 模式下支持坐标系统，robot 根据当前坐标系解析点位。
3	PPB 模式下 Retry, Continue 指令不支持，Pause 指令不支持。
4	不支持传送带跟踪模式。
5	进出 PPB 模式不支持 blending 功能
6	PPB 模式下不支持 Move 和 Jog 运动。
7	进入 PPB 模式后，motion planning 的工作由使用者负责，内部的部分操作检查就相对失效。例如 PPB 走到“尽头”如果系统接收到的轨迹没有适当的减速，在没有新的 setpoint 情形下 robot 动作会骤然停止，这时候非常可能引发系统 fault（例如 tracking error 过大）。

3. 使用方法

步骤 1：将点位文件通过 FTP 拷入 192.168.10.101/data。

步骤 2：运行如下指令

- 1) 如果是笛卡尔点位文件，姿态用 zyz 欧拉角表示的：

```
PPB.Enable 1,1
Robot.Frame 1,2
PPB.ReadFile 1,/data/cart_zyz.txt
PPB.J2StartPoint 1,cfg,0
PPB.Run 1
```

其中，cfg 表示机器人 config。例如，SCARA 机器人，cfg=1，表示左手。

- 2) 如果是笛卡尔点位文件，姿态用固定角表示的：

```
PPB.Enable 1,1
Robot.Frame 1,2
PPB.ReadFile 1,/data/cart_xyz.txt
PPB.J2StartPoint 1,cfg,1
PPB.Run 1
```

- 3) 如果是关节点位文件。

```
PPB.Enable 1,1
Robot.Frame 1,1
PPB.ReadFile 1,/data/joint.txt
PPB.J2StartPoint 1,0,1
PPB.Run 1
```

- 4) 如果需要终止运动。

```
System.Abort 0
```

点位文件格式

1. 文件格式说明

- d) 点位文件每一行表示一组点位，每组点位中的元素用空格隔开。
- e) 笛卡尔空间维度为 6，分别表示位置和姿态。
- f) 位置：x,y,z，单位都是 mm。
- g) 固定角姿态：rx,ry,rz，单位是 degree；欧拉角姿态：rz,ry,rz，单位是 degree。
- h) 每组点位的开头不能有空格。例如：

```
200.0 10.0 20.0 180.0 0.0 30.0
```

应当修改为，

```
200.0 10.0 20.0 180.0 0.0 30.0
```

-
- i) 当点位文件为关节点位时，列数随关节维度而定，单位是 degree。

文件实例

1. 笛卡尔空间点位文件

```
200.0  10.0  20.0  180.0  0.0  30.0
200.0  10.0  20.1  180.0  0.0  30.0
200.0  10.0  20.2  180.0  0.0  30.0
200.0  10.0  20.4  180.0  0.0  30.0
.....
```

2. 关节空间点位文件（4 轴机器人）

```
30.0    30.1  30.0  30.0
30.0    30.2  30.0  30.0
30.0    30.4  30.0  30.0
30.0    30.9  30.0  30.0
.....
```

注意文件不能有空行，否则保存

附录 C

AH-5020-054S 电机减速机参数							
轴名称	减速比 i	电机功率 (W)	电机力矩常数 K (N·m/A)	电机 电 流 I (A)	电机输出扭矩 $T_m(N \cdot m) = K \cdot I$	关节输出扭矩 $T_j(N \cdot m) = T_m \cdot i \cdot \eta$ 丝杠推力 $P(N) = 2\pi \eta T_m \cdot i / L$	备注 η : 传动效率, 大约为 0.9~0.95, 按 0.9 计算
J1	80	400	0.4	1	0.4	28.8	
J2	50	200	0.39	1	0.39	17.55	
J3	1.8	200	0.39	1	0.39	198.4858239	推力 $P = 2\pi \eta T / L$ η : 丝杆传动效率, 大约为 0.9~0.95, 按 0.9 计算 T: 转矩 L: 丝杆导程 = 0.02m
J4	16.4	100	0.32	1	0.32	4.7232	

AP-1130-465S 电机减速机参数							
轴名称	减速比	电机功率 (W)	电机力矩常数	电机 电 流 I (A)	电机输出扭矩 $T_m(N \cdot m) = K \cdot I$	关节输出扭矩 $T_j(N \cdot m) = T_m \cdot i \cdot \eta$	备注
J1	50	750	0.442	1	0.442	19.89	
J2	50	750	0.442	1	0.442	19.89	
J3	50	750	0.442	1	0.442	19.89	
J4	20	400	0.4628	1	0.4628	8.3304	

HL6-0900-3656 电机减速机参数							
轴名称	减速比	电机功率(W)	电机力矩常数	电机 电 流 I (A)	电机输出扭矩 $T_m(N \cdot m) = K \cdot I$	关节输出扭矩 $T_j(N \cdot m) = T_m \cdot i \cdot \eta$	备注
J1	75	750	0.5	1	0.5	33.75	
J2	120	400	0.4628	1	0.4628	49.9824	
J3	100	400	0.4628	1	0.4628	41.652	
J4	81.6	100	0.2885	1	0.2885	21.18744	
J5	80	100	0.25	1	0.25	18	
J6	50	100	0.25	1	0.25	11.25	