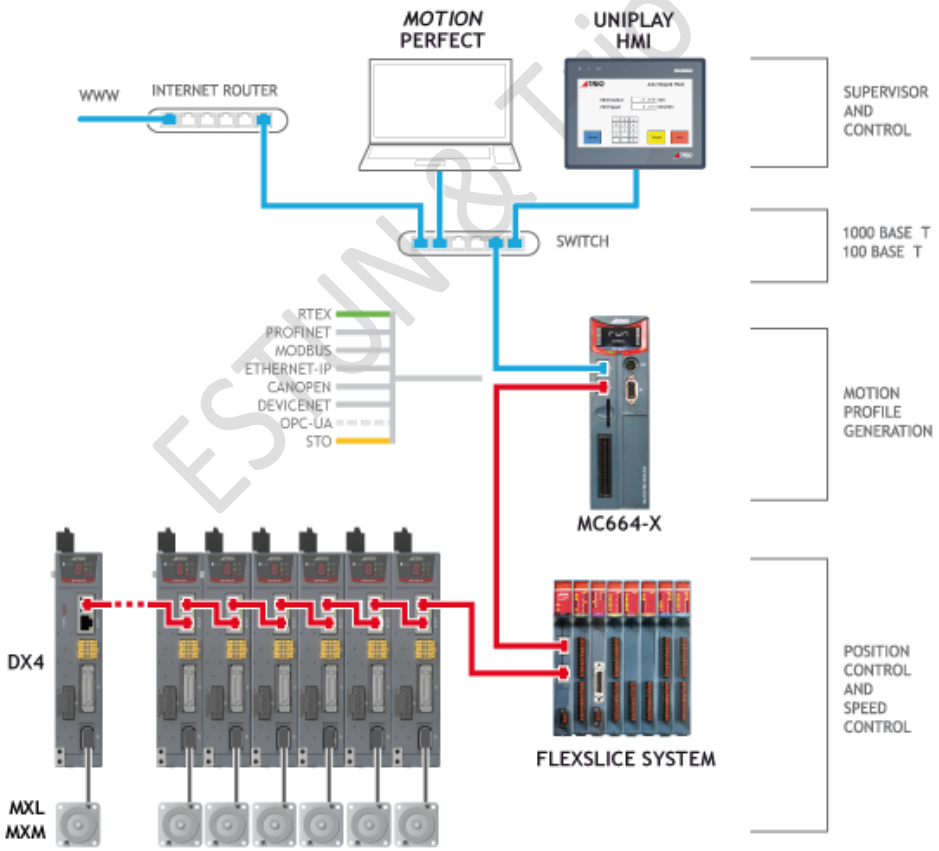
运动控制系统与应用技术

应用技术部　编制



前言

众所周知，机械装备的制造加工过程和功能一般都是通过其部件的运动来实现的。尽管不同设备的制造加工原理有很大差异（如冷加工的金属切削、剪切、折弯，热加工的锻造、焊接等），都离不开对机械部件的运动需求。获得特定的机械运动可以实现更为优质的加工需求（如加工的精度、效率、品质等）是机械装备的核心需求。

运动控制泛指通过某种驱动部件（如伺服电机、步进电机、液压泵）针对机械设备（或其部件），在运动过程中按照制造加工功能要求，对其运动的基本要素（如位置、速度、加速度、加加速度、转矩等）进行控制。运动控制技术已被广泛应用到各行各业中，它是一门综合性学科，集计算机技术、微电子技术、电力电子技术、信息技术、传感器技术等，将控制理论与机械工程的应用相结合。

本资料以 Trio 运动控制产品为背景，介绍运动控制的基本功能和使用，为运动控制的技术人员提供系统的应用层所需知识。

本资料由五篇构成，分别是通用篇、TrioBASIC 篇，IEC61131-3 篇，PC Motion 篇（标准规范篇）和实践篇构成。通用篇介绍运动控制系统的构成、各构成组件的工作原理、以及应用选型，介绍了运动控制系统的各个组件的基本知识和 MotionPerfect V5 的集成开发环境的功能的使用。TrioBASIC 篇，介绍了 TrioBASIC 的编程语言， Trio 的相关指令的功能与使用，以及机器人编程系统的使用。IEC61131-3 篇，简要介绍了 IEC61131-3 的标准内容，并结合 MotionPerfect V5 对 IEC61131-3 标准的支持，介绍了如何在 MotionPerfect V5 中使用 IEC61131-3 编程标准进行应用开发，同时介绍了 PLCopen 标准的相关内容，以及 Trio PLCopen MC 运动控制功能块库。PC Motion 篇，结合国内的应用情况，介绍了 Trio 运动控制器在 PC 端实现运动控制的支持，重点介绍了 ActiveX 和.Net DLL 的功能与使用方法。

本资料在 ABG 营销中心的支持下，由应用技术部团队（张圣、汪振涛、叶伟中、沈汉林、许涛）编制，胡乃龙提供机器人编程系统章节的相关资料。编制过程参考了相关的书籍、论文材料，在此向有关作者表示感谢。

由于编者水平所限，错漏及不足之处在所难免，敬请读者不吝指正。

您的意见和建议请发送邮件至 zhangsheng@estun.com，我们将不断完善书本的内容。

编者

2020 年 12 月 25 日

ESTUN & Trio

文档修订记录

序号	版本号	修订日期	修订概述	修订人	审批人	备注
1	V1.0	2020-12-25	新建	张圣，汪振涛、叶伟中、沈汉林、许涛，胡乃龙	许乐平	

ESTUN & Trio

目 录

前言
通用篇 - 1 -
1 运动控制系统 - 1 -
1.1 运动控制器 - 1 -
1.1.1 PLC - 1 -
1.1.2 专用运动控制器 - 6 -
1.1.3 具有运动控制功能的 PLC - 8 -
1.1.4 PC-Based 运动控制器 - 8 -
1.1.5 运动控制器的发展趋势 - 9 -
1.2 驱动控制器 - 11 -
1.2.1 变频器 - 11 -
1.2.2 伺服系统 - 13 -
1.2.3 步进电机驱动器 - 16 -
1.3 执行器 - 20 -
1.3.1 电机的种类 - 20 -
1.3.2 步进电机 - 20 -
1.3.3 伺服电机 - 22 -
1.4 传感器 - 23 -
1.4.1 编码器 - 24 -
1.4.2 模拟量位移传感器 - 26 -
1.4.3 部分技术参数 - 27 -
1.4.4 传感器的选择原则 - 27 -
1.5 控制器硬件接口 - 28 -
1.5.1 数字量输入/输出 - 28 -
1.5.2 模拟量输入/输出 - 30 -
1.5.3 脉冲输入输出 - 31 -
1.5.4 通信接口 - 33 -
1.5.5 现场总线协议 - 43 -
1.6 HMI - 60 -
2 Motion Perfect V5 - 63 -
2.1 软件的安装及基本使用 - 63 -
2.1.1 安装环境要求 - 63 -
2.1.2 语言设置 - 64 -
2.2 软件的连接 - 66 -
2.2.1 Ethernet 连接设置 - 66 -
2.2.2 仿真器连接设置 - 68 -
2.2.3 连接模式 - 69 -
2.3 控制器窗体 - 72 -
2.3.1 主菜单 - 72 -
2.3.2 Controller（控制器） - 72 -
2.3.3 工具栏 - 75 -
2.3.4 控制树 - 76 -

ESTUN & Trio

2.4 解决方案管理 - 79 -
2.5 工程操作 - 81 -
2.5.1 创建工程 - 81 -
2.5.2 上传工程 - 82 -
2.5.3 同步工程 - 83 -
2.5.4 修改工程 - 84 -
2.5.5 软件包操作 - 85 -
2.6 程序文件 - 91 -
2.6.1 文件类型 - 91 -
2.6.2 文件操作 - 93 -
2.6.3 新建文件 - 93 -
2.6.4 删除文件 - 94 -
2.6.5 导入文件 - 95 -
2.6.6 导出文件 - 96 -
2.7 工具 - 97 -
2.7.1 命令窗口 - 97 -
2.7.2 轴参数 - 100 -
2.7.3 数字量输入输出 - 103 -
2.7.4 模拟量输入输出 - 105 -
2.7.5 TABLE 监控 - 106 -
2.7.6 VR 监控 - 108 -
2.7.7 变量监控 - 108 -
2.7.8 轴点动 - 110 -
2.7.9 示波器 - 111 -
2.7.10 智能轴驱动 - 117 -
2.8 控制器工具 - 118 -
2.8.1 功能码激活 - 118 -
2.8.2 SD 卡管理 - 119 -
2.8.3 加载固件版本 - 120 -
2.8.4 重编程 FPGA - 123 -
2.8.5 文件路径查看 - 125 -
2.8.6 任务状态查看 - 125 -
2.8.7 数值导入/导出 - 126 -
2.8.8 控制器锁定/解锁 - 128 -
2.8.9 控制器时钟设置 - 130 -
2.8.10 备份管理 - 130 -
2.9 Mpv5 常用设置 - 131 -
2.9.1 刷新率设置 - 131 -
2.9.2 初始化设置 - 132 -
2.9.3 程序编辑设置 - 132 -
2.9.4 语言设置 - 133 -
2.9.5 工程设置 - 133 -
2.10 系统参数设置（MC_CONFIG） - 134 -
2.10.1 概述 - 134 -

ESTUN & Trio

2.10.2 IP 设置 - 134 -
2.10.3 ETHERCAT 设置 - 135 -
2.10.4 轴地址设置 - 135 -
2.10.5 轴类型设置 - 136 -
2.10.6 IO 地址设置 - 136 -
2.10.7 CPU 分配设置 - 136 -
2.10.8 任务分配设置 - 136 -
2.10.9 编译设置 - 136 -
2.11 程序的运行管理 - 137 -
Trio BASIC 篇 - 138 -
3 Trio BASIC 编程基础 - 138 -
3.1 局部变量 - 138 -
3.1.1 描述 - 138 -
声明-DIM - 138 -
3.1.2 字符串变量 - 139 -
3.1.3 数组变量 - 141 -
3.1.4 作用域详述 - 143 -
3.1.5 INCLUDE - 146 -
3.2 全局变量-VR - 148 -
3.2.1 描述 - 148 -
3.2.2 定义-GLOBAL - 148 -
3.2.3 VR 操作 - 149 -
3.2.4 使用场合 - 150 -
3.3 全局变量-TABLE - 151 -
3.3.1 描述 - 151 -
3.3.2 数据操作 - 151 -
3.3.3 使用场合 - 152 -
3.3.4 相关指令 - 154 -
3.4 常量 - 157 -
3.4.1 CONSTANT - 157 -
3.4.2 DEFCONST - 158 -
3.4.3 PI - 159 -
3.4.4 TRUE - 159 -
3.4.5 FALSE - 159 -
3.4.6 ON - 159 -
3.4.7 OFF - 159 -
3.5 运算符 - 160 -
3.5.1 = - 160 -
3.5.2 + - 160 -
3.5.3 – - 160 -
3.5.4 * - 160 -
3.5.5 / - 160 -
3.5.6 MOD（整数取余） - 160 -
3.5.7 FMOD（浮点数取余） - 161 -

ESTUN & Trio

3.5.8 ^（多次方） - 161 -
3.6 判断 - 161 -
3.6.1 > - 161 -
3.6.2 < - 161 -
3.6.3 >= - 161 -
3.6.4 <= - 162 -
3.6.5 <> - 162 -
3.7 位运算 - 162 -
3.7.1 >> - 162 -
3.7.2 << - 162 -
3.7.3 AND - 162 -
3.7.4 OR - 162 -
3.7.5 NOT - 163 -
3.7.6 XOR - 163 -
3.8 特殊符号 - 163 -
3.8.1 ： - 163 -
3.8.2 _(换行符) - 163 -
3.8.3 $（16 进制） - 163 -
3.8.4 # - 164 -
3.8.5 ‘（注释符号） - 164 -
3.9 结构指令 - 164 -
3.9.1 IF 语句 - 164 -
3.9.2 CASE 语句 - 165 -
3.9.3 WHILE 语句 - 166 -
3.9.4 REPEAT 语句 - 168 -
3.9.5 FOR 循环 - 168 -
3.9.6 EXIT_LOOP - 170 -
3.9.7 CONTINUE - 170 -
3.9.8 GOSUB - 170 -
3.9.9 GOTO - 171 -
3.9.10 FUNCTION - 172 -
3.9.11 WAIT - 174 -
3.10 IO 指令 - 176 -
3.10.1 数字量输入读取-IN - 176 -
3.10.2 数字量输出-OP - 176 -
3.10.3 数字量输出读取-READ_OP - 177 -
3.10.4 数字量输入取反-INVERT_IN - 178 -
3.10.5 位置比较输出-PSWITCH - 178 -
3.11 时间相关指令 - 181 -
3.11.1 延时指令-WA - 181 -
3.11.2 周期设置-SERVO_PERIOD - 181 -
3.11.3 任务计时-TICKS - 181 -
3.11.4 控制器时间-TIME - 183 -
3.11.5 控制器时间-TIME$ - 183 -

ESTUN & Trio

3.12 字符串相关指令 - 183 -
3.12.1 ASC - 183 -
3.12.2 CHR - 184 -
3.12.3 DATE$ - 184 -
3.12.4 DAY$ - 184 -
3.12.5 HEX - 184 -
3.12.6 INSTR - 185 -
3.12.7 LCASE - 185 -
3.12.8 LEFT - 185 -
3.12.9 LEN - 186 -
3.12.10 MID - 186 -
3.12.11 RIGHT - 186 -
3.12.12 STR - 186 -
3.12.13 UCASE - 187 -
3.12.14 VAL - 187 -
3.13 多任务 - 188 -
3.13.1 相关概念介绍 - 188 -
3.13.2 上电自启动任务设置 - 189 -
3.13.3 手动运行任务设置 - 190 -
3.13.4 任务的自动运行（RUN） - 192 -
3.13.5 任务停止-STOP - 193 -
3.13.6 任务停止-HALT - 194 -
3.13.7 任务的状态查看 - 194 -
3.13.8 任务资源的分配原则 - 200 -
3.13.9 常见任务操作错误及故障排查 - 208 -
3.13.10 特殊任务的设置 - 209 -
3.14 运动加载原则 - 211 -
3.14.1 任务缓存区 - 211 -
3.14.2 轴运动缓存区 - 212 -
3.14.3 加载规则及实例 - 214 -
3.15 单轴基本操作 - 215 -
3.15.1 轴类型设置 - 215 -
3.15.2 轴选择 - 217 -
3.15.3 运行单位设置 - 219 -
3.15.4 安全区设置 - 222 -
3.15.5 跟随误差的设置 - 227 -
3.15.6 使能操作 - 229 -
3.15.7 回零操作 - 233 -
3.15.8 JOG 实现 - 242 -
3.15.9 INC（寸动）实现 - 243 -
3.15.10 绝对运动 - 244 -
3.15.11 相对运动 - 245 -
3.15.12 运动取消 - 245 -
3.15.13 暂停保护功能 - 246 -

ESTUN & Trio

3.15.14 总线轴运动异常排查 - 248 -
3.15.15 运动平滑功能 - 249 -
3.15.16 速度控制的实现 - 251 -
3.15.17 力矩控制的实现 - 253 -
3.16 轴状态的监控 - 256 -
3.16.1 位置监控 - 256 -
3.16.2 速度监控 - 256 -
3.16.3 运动状态监控 - 257 -
3.17 多轴插补运动 - 260 -
3.17.1 插补轴组指定 - 260 -
3.17.2 插补速度设置 - 260 -
3.17.3 直线插补 - 262 -
3.17.4 圆弧插补 - 263 -
3.18 复杂运动 - 268 -
3.18.1 轴叠加 - 268 -
3.18.2 轴叠加 2 - 275 -
3.18.3 位置锁存功能 - 277 -
3.18.4 电子齿轮 - 281 -
3.18.5 电子凸轮 - 286 -
3.19 系统参数 - 302 -
3.19.1 IP 配置 - 302 -
3.19.2 系统运行状态信息 - 302 -
3.19.3 设备相关信息 - 302 -
3.19.4 COORDINATOR_DATA - 303 -
3.19.5 SCHEDULE_TYPE - 304 -
3.19.6 IP_PROTOCOL_CTRL - 305 -
4 通信接口 - 307 -
4.1 Modbus TCP - 307 -
4.1.1 应用场合 - 307 -
4.1.2 客户端 - 322 -
4.1.3 数据转换 - 326 -
4.2 TCP/IP - 327 -
4.2.1 客户端 - 327 -
4.2.2 服务器 - 335 -
4.3 UDP - 336 -
4.3.1 描述 - 336 -
4.3.2 连接建立 - 336 -
4.3.3 连接断开 - 336 -
4.3.4 连接状态检测 - 336 -
4.3.5 数据发送 - 336 -
4.3.6 数据侦听 - 336 -
4.3.7 数据接收 - 337 -
4.3.8 实例 - 337 -
4.4 EtherCAT - 337 -

ESTUN & Trio

4.4.1 相关概念 - 337 -
4.4.2 智能轴驱动使用 - 339 -
4.4.3 PDO 配置 - 345 -
4.4.4 SDO 操作 - 356 -
4.4.5 总线初始化 - 359 -
4.4.6 伺服节点设置 - 362 -
4.4.7 常见总线错误 - 364 -
5 机器人编程系统 RPS - 367 -
5.1 如何建立一个机器人工程 - 367 -
5.1.1 创建一个机器人模型 - 367 -
5.1.2 ROBOT_DEFINITIONS 文件 - 377 -
5.2 建立 3D 可视化 - 378 -
5.3 RPS 编程介绍 - 381 -
5.3.1 RPS 语言扩展 - 381 -
5.3.2 Behaviour - 381 -
5.3.3 机器人选择和程序类型 - 382 -
5.3.4 Frames and Tools - 383 -
5.3.5 TARGET data type - 386 -
5.3.6 Robot Position - 390 -
5.3.7 机器人 JOG - 390 -
5.4 运动指令 - 392 -
5.4.1 总体概述 - 392 -
5.4.2 过渡精度 - 394 -
5.4.3 MOVEJ - 395 -
5.4.4 MOVEL - 398 -
5.4.5 MOVEC - 400 -
5.4.6 MOVEJREL - 402 -
5.4.7 MOVELREL - 403 -
5.5 碰撞区域 - 403 -
5.5.1 定义 - 403 -
5.5.2 语法 - 404 -
5.6 动态抓取 - 405 -
5.6.1 语法 - 405 -
5.6.2 Attach Object Frame - 405 -
5.6.3 Sync to object frame - 407 -
IEC61131-3 篇 - 411 -
6 概述 - 411 -
6.1 IEC61131-3 标准 - 411 -
6.2 PLCopen 组织 - 411 -
6.3 认证等级 - 412 -
6.4 公用元素 - 413 -
6.4.1 数据类型 - 413 -
6.4.2 变量 - 415 -
6.4.3 软件模型（配置、资源、任务） - 417 -

ESTUN & Trio

6.4.4 程序组织单元 - 418 -
6.4.5 顺序功能图 SFC - 421 -
6.5 编程语言 - 422 -
7 Motion Perfect 中的 IEC61131-3 的使用 - 423 -
7.1 MotionPerfect 编程主界面 - 423 -
7.2 创建 IEC61131-3 任务 - 423 -
7.3 数据类型 - 426 -
7.4 创建变量 - 427 -
7.5 创建用户功能库 - 429 -
7.6 选择编程语言 - 431 -
7.7 Defines 的应用 - 433 -
7.7.1 预处理指令 - 433 -
7.7.2 条件编译指令 - 433 -
7.7.3 声明异常处理程序 - 434 -
8 PLCopen MC 库 - 436 -
8.1 PLCopen 规范概述 - 436 -
8.2 PLCopen 运动控制功能块 - 437 -
8.2.1 边沿触发功能块 - 437 -
8.2.2 电平控制功能块 - 439 -
8.2.3 共性特性 - 439 -
8.2.4 ContinuousUpdate - 440 -
8.2.5 Re-execution - 441 -
8.2.6 功能块的命名 - 441 -
8.3 Trio PLCopen MC 介绍 - 443 -
8.3.1 状态图 - 443 -
8.3.2 库的构成 - 445 -
8.3.3 定义的数据类型 - 446 -
8.3.4 相关的规则说明 - 450 -
8.3.5 轴参数 - 453 -
8.3.6 错误处理 - 454 -
8.4 使用 Trio PLCopen MC 功能块 - 455 -
8.4.1 轴参数的设置 - 455 -
8.4.2 轴状态查询 - 456 -
8.4.3 轴使能与禁能 - 456 -
8.4.4 回零 - 457 -
8.4.5 运行轴 - 458 -
8.4.6 停止轴 - 458 -
8.4.7 程序示例 - 459 -
PC Motion 篇 - 461 -
9 PC Motion - 461 -
9.1 PC Motion 的特点 - 462 -
9.2 适用场景 - 463 -
9.3 ActiveX - 465 -
9.3.1 ActivXe 控件 - 465 -

ESTUN & Trio

9.3.2 ActiveX 的使用 ..... - 465 -
9.4 DLL ..... - 469 -
9.4.1 DLL 简介 ..... - 469 -
9.4.2 DLL 的简单使用 ..... - 469 -
9.5 应用介绍 ..... - 471 -
9.5.1 控制器初始化 ..... - 472 -
9.5.2 系统及轴参数初始化 ..... - 473 -
9.5.3 轴运动 ..... - 475 -
9.5.4 轴跟随运动 ..... - 479 -
9.5.5 输入/输出 ..... - 485 -
9.5.6 VR/TABLE 变量操作 ..... - 488 -
9.5.7 Execute 指令 ..... - 491 -
参考文献 ..... - 492 -

ESTUN & Trio

ESTUN & Trio

通用篇

1 运动控制系统

运动控制技术是工业自动化领域的一个重要分支，主要应用于机床设备、自动化流水线与机器人控制等领域。运动控制系统中的主要部件通常包括运动控制器，驱动控制器、执行机构和传感器。



图 1-1 运动控制系统主要部件

典型的运动控制系统是一类闭环控制系统，通常，运动控制器包含闭合系统所需（如位置、速度、力矩）的控制算法/策略，同时实现对输入/输出信号的处理，实现与被控对象的接口控制。驱动控制器或放大器中实现运动控制器的控制指令（通常为速度或电流）的转换，实现执行机构的驱动控制。通常执行机构通过电机来实现运动，电机的类型包括有刷型或无刷型，旋转的或线性的。传感器反馈元件,例如旋转变压器、线性位移电阻器、转速计、光栅尺、高压计、编码器等，安装在驱动电机或负载上，为控制器提供运动状态信息，实现闭合的伺服回路。

1.1 运动控制器

运动控制器是运动控制系统的核心部件，负责产生运动路径控制指令、设备的逻辑控制，并对被控对象的外部环境变化及时做出响应。通常，运动控制器通过一定的通信手段将控制信号或命令发送给驱动控制器，驱动控制器器为执行结构（通常为电机）提供转动能源动力，同时将来自执行机构（电机）和负载的反馈传回控制器，实现闭环。运动控制器接收并且分析反馈，通过更新控制信号和命令来校正误差。典型的控制器有下几类:

1.1.1 PLC

可编程逻辑控制器(Programmable Logic Controller，PLC)，最初的目的是替代机械开关装置(继电模块)，其内部具有微处理器的用于自动化控制的数字运算控制器，可以将控制指令

随时载入内存进行储存、执行。PLC 由 CPU、指令及数据内存、输入/输出接口、电源、数字模拟转换等功能单元组成。早期的可编程逻辑控制器只有逻辑控制的功能，所以被命名为可编程逻辑控制器，后来随着不断地发展，这些当初功能简单的计算机模块已经有了包括逻辑控制、时序控制、模拟控制、多机通信等各类功能，名称也改为可编程控制器(Programmable Controller)，但是由于它的简写 PC 与个人电脑(Personal Computer)的简写相冲突，加上习惯的原因，人们还是经常使用可编程逻辑控制器这一称呼，并仍使用 PLC 这一缩写。



图 1-2 PLC 产品图片

1.1.1.1 PLC 的基本组成

可编程逻辑控制器实质是一种专用于工业控制的计算机，其硬件结构基本上与微型计算机相同，由中央处理模块(CPU 板)、输入输出模块(I/O)部件、编程器模块和电源模块四部分组成。基本组成如图 1-3 所示，基本构成详细描述如下：

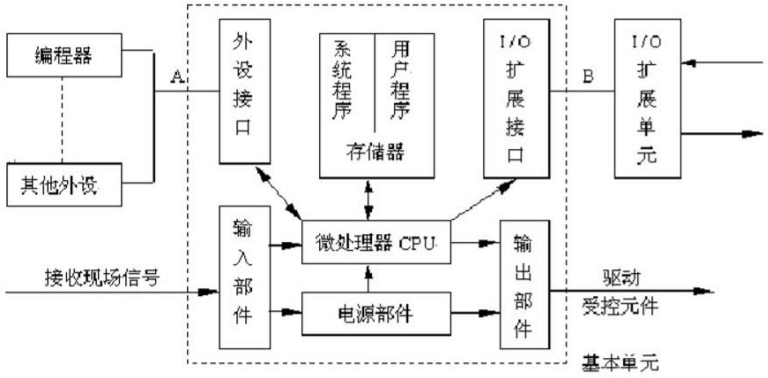


图 1-3 PLC 的基本组成

电源

电源用于将交流电转换成 PLC 内部所需的直流电，目前大部分 PLC 采用开关式稳压电源供电。

中央处理单元

中央处理器（CPU）是 PLC 的控制中枢，也是 PLC 的核心部件，其性能决定了 PLC 的性能。

中央处理器由控制器、运算器和寄存器组成，这些电路都集中在一块芯片上，通过地址总线、控制总线与存储器的输入/输出接口电路相连。中央处理器的作用是处理和运行用户

程序，进行逻辑和数学运算，控制整个系统使之协调。

存储器

存储器是具有记忆功能的半导体电路，它的作用是存放系统程序、用户程序、逻辑变量和其他一些信息。其中系统程序是控制 PLC 实现各种功能的程序，由 PLC 生产厂家编写，并固化到只读存储器(ROM)中，用户不能访问。

输入/输出部件

输入单元是 PLC 与被控设备相连的输入接口，是信号进入 PLC 的桥梁，它的作用是接收主令元件、检测元件传来的信号。输入的类型有直流输入、交流输入、交直流输入。 [6]

输出单元也是 PLC 与被控设备之间的连接部件，它的作用是把 PLC 的输出信号传送给被控设备，即将中央处理器送出的弱电信号转换成电平信号，驱动被控设备的执行元件。输出的类型有继电器输出、晶体管输出、晶闸门输出。

PLC 除上述几部分外，根据机型的不同还有多种外部设备，其作用是帮助编程、实现监控以及网络通信。常用的外部设备有编程器、打印机、盒式磁带录音机、计算机等。

1.1.1.2 工作原理

当可编程逻辑控制器投入运行后，其工作过程一般分为三个阶段，即输入采样、用户程序执行和输出刷新三个阶段。完成上述三个阶段称作一个扫描周期。在整个运行期间，可编程逻辑控制器的 CPU 以一定的扫描速度重复执行上述三个阶段。

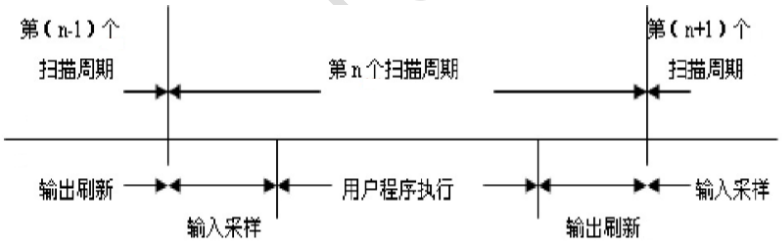


图 1-4 PLC 程序执行过程

输入采样

在输入采样阶段，可编程逻辑控制器以扫描方式依次地读入所有输入状态和数据，并将它们存入 I/O 映象区中的相应的单元内。输入采样结束后，转入用户程序执行和输出刷新阶段。在这两个阶段中，即使输入状态和数据发生变化，I/O 映象区中的相应单元的状态和数据也不会改变。因此，如果输入是脉冲信号，则该脉冲信号的宽度必须大于一个扫描周期，才能保证在任何情况下，该输入均能被读入。

用户程序执行

在用户程序执行阶段，可编程逻辑控制器总是按由上而下的顺序依次地扫描用户程序(梯形图)。在扫描每一条梯形图时，又总是先扫描梯形图左边的由各触点构成的控制线路，并按先左后右、先上后下的顺序对由触点构成的控制线路进行逻辑运算，然后根据逻辑运算的结果，刷新该逻辑线圈在系统 RAM 存储区中对应位的状态；或者刷新该输出线圈在 I/O 映象区中对应位的状态；或者确定是否要执行该梯形图所规定的特殊功能指令。

即，在用户程序执行过程中，只有输入点在 I/O 映象区内的状态和数据不会发生变化，而其他输出点和软设备在 I/O 映象区或系统 RAM 存储区内的状态和数据都有可能发生变化，而且排在上面的梯形图，其程序执行结果会对排在下面的凡是用到这些线圈或数据的梯形图起作用；相反，排在下面的梯形图，其被刷新的逻辑线圈的状态或数据只能到下一个扫描周期才能对排在其上面的程序起作用。

在程序执行的过程中如果使用立即 I/O 指令则可以直接存取 I/O 点。即使用 I/O 指令的话，输入过程映像寄存器的值不会被更新，程序直接从 I/O 模块取值，输出过程映像寄存器会被立即更新，这跟立即输入有些区别。

程序的执行遵循从左到右、从上到下的原则。扫描工作方式从第一条程序开始，在无中断和跳转的情况下，逐条执行程序，直到结束。然后再从头开始扫描执行，周而复始。

输出刷新

当扫描用户程序结束后，可编程逻辑控制器就进入输出刷新阶段。在此期间，CPU 按照 I/O 映象区内对应的状态和数据刷新所有的输出锁存电路，再经输出电路驱动相应的外设。这时，才是可编程逻辑控制器的真正输出。

小结

根据上述过程的描述，可以对 PLC 工作过程的特点小结如下：

①PLC 采用集中采样、集中输出的工作方式，这种方式减少了外界干扰的影响。

②PLC 的工作过程是循环扫描的过程，循环扫描时间的长短取决于指令执行速度、用户程序的长度等因素。

③输出对输入的影响有滞后现象。PLC 采用集中采样、集中输出的工作方式，当采样阶段结束后，输入状态的变化将要等到下一个采样周期才能被接收，因此这个滞后时间的长短又主要取决于循环周期的长短。此外，影响滞后时间的因素还有输入滤波时间、输出电路的滞后时间等。

④输出映像寄存器的内容取决于用户程序扫描执行的结果。

⑤输出锁存器的内容由上一次输出刷新期间输出映像寄存器中的数据决定。

⑥PLC 当前实际的输出状态由输出锁存器的内容决定。

1.1.1.3 功能特点

(1)可靠性高。由于 PLC 大都采用单片微型计算机，因而集成度高，再加上相应的保护电路及自诊断功能，提高了系统的可靠性。

(2)编程容易。PLC 的编程多采用继电器控制梯形图及命令语句，其数量比微型机指令要少得多，除中、高档 PLC 外，一般的小型 PLC 只有 16 条左右。由于梯形图形象而简单，因此容易掌握、使用方便，甚至不需要计算机专业知识，就可进行编程。

(3)组态灵活。由于 PLC 采用积木式结构，用户只需要简单地组合，便可灵活地改变控制系统的功能和规模，因此，可适用于任何控制系统。

(4)输入/输出功能模块齐全。PLC 的最大优点之一，是针对不同的现场信号(如直流或交

ESTUN & Trio

流、开关量、数字量或模拟量、电压或电流等)，均有相应的模板可与工业现场的器件(如按钮、开关、传感电流变送器、电机启动器或控制阀等)直接连接，并通过总线与 CPU 主板连接。

(5)安装方便。与计算机系统相比，PLC 的安装既不需要专用机房，也不需要严格的屏蔽措施。使用时只需把检测器件与执行机构和 PLC 的 I/O 接口端子正确连接，便可正常工作。

(6)运行速度快。由于 PLC 的控制是由程序控制执行的，因而不论其可靠性还是运行速度，都是继电器逻辑控制无法相比的。

近年来，微处理器的使用，特别是随着单片机大量采用，大大增强了 PLC 的能力，并且使 PLC 与微型机控制系统之间的差别越来越小，特别是高档 PLC 更是如此。

1.1.1.4 PLC 的选型

当运动控制系统使用到 PLC 作为运动控制系统的控制器时，选择合适、可靠的 PLC 设备至关重要。其基本原则是在功能满足应用的前提下，选择可靠的、便于维护以及性价比高的机型。在此原则下主要考虑如下因素：

满足应用的功能要求：

1，满足所需接口的要求。如数字量输入/输出，模拟量输入/输出，脉冲接口等用于控制的接口数量要满足应用的需求，且有一定的扩展余量，避免应用实际开发过程或后期的维护升级时，需要增加接口而导致的麻烦。需要做到：

（1）输入模块的输入电路应与外部传感器或电子设备（例如变频器）的输出电路的类型相配合，最好能使二者直接相连。对于需要大电流驱动的应用场合，需要考虑如何实现大电流的方案是否在方案中可实现，如选择继电器输出且采用外部供电。对于需要快速的输出输出的场合，需要考虑端口的响应速度（如选择带有 MOSFET 驱动的输出模块）。

（2）选择模拟量模块时，应考虑使用放大器，以及执行机构的量程是否能与 PLC 的模拟量输入/输出模块的量程匹配。

（3）使用旋转编码器时，应考虑 PLC 的高速计数器的功能和工作频率是否能满足要求。

（4）使用脉冲输出时，要考虑与被连接设备的接口对应，满足所需工作频率匹配。

2，PLC 处理性能。

对 PLC 在的响应速度有要求的情况，需要选用如高速 I/O 接口的 PLC， 或选用具有快速响应的模块，或选用带有中断输入的 PLC。程序的处理时间也会影响响应的速度，对于需要处理复杂、大量数据的快响应要求场合，选择具有高性能处理能力的 PLC。如实现 PID 运算、闭环控制、联网通信等功能的应用场合。

3， 存储容量的选择

PLC 程序存储器的容量通常以字或步为单位，用户程序存储器的容量可以作粗略的估算。一般情况下用户程序所需的存储器容量可按照如下经验公式计算：

程序容量 = K × 总输入点数/总输出点数

对于简单的控制系统，K=6；若为普通系统，K=8；若为较复杂系统，K=10；若为复

ESTUN & Trio

杂系统，则 K=12。在选择内存容量时同样应留有裕量，一般是运行程序的 25%。不应单纯追求大容量，在大多数情况下，满足 I/O 点数的 PLC，内存容量也能满足。

4，联网通信的选择。网络化的趋势在 PLC 的产品中也日渐显现。若 PLC 控制系统需要与外部设备、传感器进行通信连接，需要选择有培训物理接口和通信协议的 PLC 产品。

5，满足安装的需要。如结构形式（整体式或模块式）、体积、安装方式（如 DIN 导轨式安装，是否支持分布式 IO 和 PLC）是否符合应用需要。

6，供电电源的选择。结合电气系统的供电来看，一般选择纹波较小的、不宜受外部设备干扰的电源，如使用稳压器或带有隔离的变压器的电源进行供电。

满足应用的非功能性需求：

1，基于当前的经验和人员技能情况。对于企业来讲，选择的 PLC 不仅从上述的功能满足层面触发，还要考虑使用人员的技能情况，还要结合公司的软件管理整合考虑。优选当前技术人员普遍熟悉的产品是通常的做法。

2，可维护性考虑。产品投入运行后，维护工作将占主导。可以快速、方便地解决运行过程中出现的故障，可以降低控制系统的维护成本。具备完善的 PLC 硬件自诊断功能或便于发现故障的措施，有助于提升产品的维护性。

3，产品的可持续性。对于一些长期存在、持续改进的方案，选择 PLC 时，需要考虑 PLC 产品在软件的模块化、灵活性、扩展性方面的因素。为解决方案的持续改进、持续发展提供保障。

阅读 PLC 的产品规格书、选型手册，了解 PLC 的选型参数和指标。阅读 PLC 的使用说明书了解 PLC 的指令集、功能和开发方法。

ESTUN & Trio

1.1.2 专用运动控制器

专用运动控制器具有较为全面的运动控制功能，可实现单轴、多轴运动的协调控制功能，通常具备有电子齿轮、电子凸轮、二维/三维空间的插补运动。这类运动控制器的实现形式多样，多以基于嵌入式的软/硬件实现，且厂商提供了大量的运动控制功能和对应的开发、调试、仿真环境。如 Trio Motion Technology 提供的 MC4xx 系列运动控制器，为用户提供 Trio BASIC 的运动控制指令和 Motion Perfect 集成开发环境，极大地方便了用户的运动控制系统的开发。此类控制器厂商对其硬件和软件的兼容性和可控性的掌握，在产品的稳定性上相对更好，更适用一些稳定性要求高的工控场合。



图 1-5 专用运动控制器

运动控制是对机械传动装置的计算机控制，就对机械运动部件的位置、速度等进行实施控制和管理，使其按预先规定的运动参数和规定的轨迹完成相应的动作。

运动规划功能

运动控制器的目的是产生指令，使被控对象按照特定的规律运行。通用运动控制器通常都提供基于对冲击、加速度和速度等这些可影响动态轨迹精度的量值加以限制的运动规划方法。专用运动控制器提供对运动控制过程的运动参数的设置和运动相关的指令，实现运动的过程控制。

通常速度规划曲线包括梯形速度曲线、S 型速度曲线，通过限制加速度和加加速度来实现冲击的限制。对于高加速度、小行程运动的快速定位系统，其定位时间和超调量都有严格的要求，往往需要高阶导数连续的运动规划方法。

多轴插补、连续插补功能

通用运动控制器提供二维、三维空间的插补功能，以满足一些多轴联动的运动需求。可以支持直线、圆弧插补功能。对于一些轨迹类加工的应用，结合 CAD/CAM 的工具，运动控制器还需要支持由大量小线段组成的路径加工，此类运动控制要求各加工段间速度波动尽可能小，在路径的拐点处实现平滑过渡。为此，专用运动控制器逐步支持了速度前瞻、连续插补的功能。

电子齿轮与电子凸轮功能

电子齿轮和电子凸轮通过软件的方式简化了设备机械的设计，获得更为灵活的运动功能。具备电子齿轮和电子凸轮的功能是通用运动控制器的标配功能。电子齿轮可以实现轴与轴/多轴间建立设定的齿轮比同步运动，使得运动控制器在定长剪切和无轴转动的套色印刷方面获得很好地应用。

电子凸轮可以通过函数或编辑凸轮形状，无需实际的凸轮机械设备，实现从轴的特定运动轨迹，获得了灵活的应用，被广泛应用于飞剪、锂电加工设备中。

位置比较输出功能

在轴的运动过程中，实现运行轴的特定位置区域使指定的数字量输出端口（一个或多个）实时、快速地输出信号。该输出常用于如在 AOI 的飞行检测中，运动控制器的比较输出功能使系统运行到设定的位置即启动 CCD 快速摄像，而运动并不受影响，这极大地提高了效率。

位置锁存功能

可以将位置锁存信号产生的时刻，记录运动轴在此时刻的位置。在如具有色标捕捉的包装设备中良好的应用。

阅读相关技术文档，了解运动控制相关的路径规划、插补原理、坐标系及坐标变换。以及多轴机器人的运动学、动力学的相关理论与数学知识。

1.1.3 具有运动控制功能的 PLC

在工业自动化领域中，传统的专用运动控制器，在编程方式和功能方面，各个品牌的供应产品的使用上有较大的差别，需要投入大量的学习精力，且相互间的互换性和兼容性通常都存在困难。选择一个通用的编程语言是解决这个问题的关键。目前 PLC 语言是使用最为普遍的编程语言，可以被广大自动化工程师所接受、使用。PLCopen 组织考虑到用户在运动控制软件标准化的强烈需求，促进了 PLC 兼容软件的开发和使用，与 IEC 组织合作，共同发展、提高和完善了 IEC61131-3 标准。



图 1-6 具备 IEC61131-3 的 Trio 运动控制器

具备支持 IEC6113-3 标准的编程方式逐步成为各大运动控制器供应商的配置。不仅为传统 PLC 的用户快速使用运动控制器提供简便，而且通过标准化编程语言和提供 PLCopen 运动控制标准的运动控制功能块，使得产品更为通用、标准，使用户更易学习和使用。

1.1.4 PC-Based 运动控制器

伴随着软件技术的发展，实时操作系统（Real-time Operation System, RTOS）被广泛应用，使得对实时性高要求的运动控制应用在 PC 端的实现成为可能。再加上实际应用对 PC 端软件的兼容需要，基于 PC 的运动控制需求，使得 PC-based 运动控制得以快速发展。这类控制器通常分为两类：一类是以传统的 PCI 接口与 PC/IPC 建立通信，让运动控制功能仍然由 PC 外的硬件（PCI 卡）实现；另一类是以软件的形式将运动控制功能植入到 PC 中，使用 PC 本身的处理器实现。



PCI运动控制卡



基于PC的软件运动控制器

图 1-7 PC-Based 运动控制器

基于 PCI 运动控制卡的 PC-Based 运动控制器，厂商通过提供运动控制卡的 DLL 或 Lib 文件，使得用户可以再高级语言中使用 PCI 运动控制卡具备的运动控制功能。可让用户的应用程序与运动控制融合、集成在一起，实现更有用户特色的应用方案。Trio Motion Technology 提供了 ActiveX 和.Net DLL 的运动控制，通过以太网与运动控制器建立通信，无需 PC 带有 PCI 接口，将低对 PC 的体积要求。用户仍可在运动控制器中编写应用程序，使得应用更为开放。

采用以纯软件的形式将运动控制功能集成到 PC 中，使用实时操作系统来进行运动控制的实时任务的控制与管理。PC-MCAT 使用多核处理器的其中一个核完成运动控制功能，共享了客户的 PC 硬件资源。这种软件的集成，具备更强大的处理能力。使用户的应用更为灵活，可以方便地集成视觉等设备。

阅读 Trio 运动控制器的选型手册，了解各个型号的运动控制器的资源，功能特点，了解控制器选型所关注的指标。

ESTUN & Trio

1.1.5 运动控制器的发展趋势

运动控制器伴随着科技的发展，尤其是计算机软件的发展，结合市场需要，其发展趋势有如下的体现

编程语言统一化。运动控制器起初多以不同的开发环境和不相兼容的软件为用户提供使用接口。这会导致使用者需要花费时间学习编程环境、编程语言、厂家提供的特有运动指令等内容，来使用特定品牌/型号的运动控制器。因此，为控制器的发展和推广带来的困难。通过 PLCopen 组织的建立，为推动运动控制器的编程语言统一化，推出了基于 IEC61131-3 标准软件模型的编程语言规范，同时针对运动控制的应用需要，制定了 PLCopen 运动控制规范。

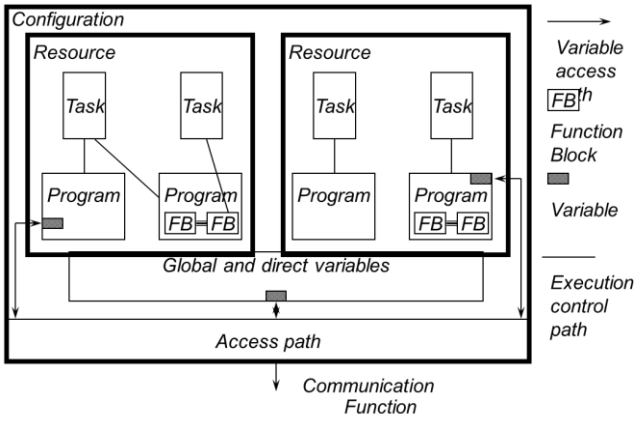


图 1-8 IEC61131-3 标准软件模型

网络化与信息化。由于运动控制系统的逐渐复杂，对控制器的扩展性要求越来越高。伴随现场总线的发展，以及现场总线的抗干扰性特点。目前市场上的运动控制器厂家几乎都推出了基于现场总线的运动控制器，在运动控制器的控制层常见的现场总线协议有 CANopen, EtherCAT 等，且以 EtherCAT 为通用运动控制器的主流总线协议。运动控制器还需要通信接口与外部实现信息交互，常见的通信接口有 RS232, RS485, Ethernet。PLCopen 组织也为该部分的标准化提供了规范文件，如 OPC UA 的应用等。

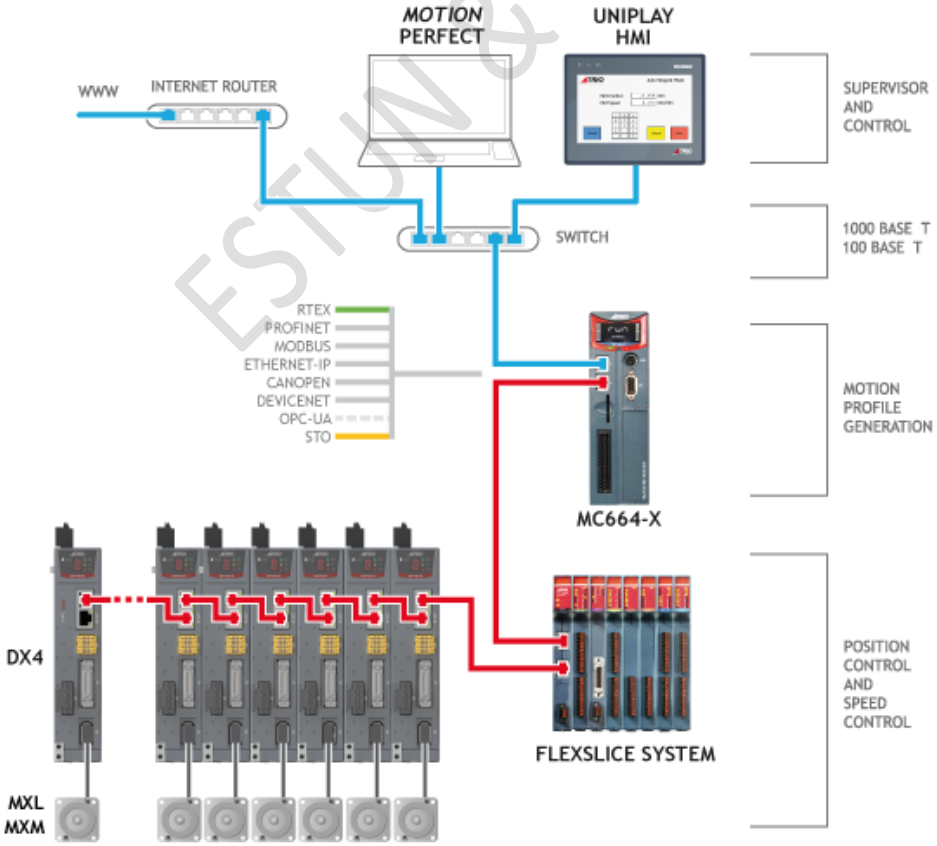


图 1-9 运动控制器的网络化和信息化示例

智能化。伴随人工智能技术、传感器技术的发展，运动控制系统越来越多地与视觉结合

应用。实现如飞拍、物品分类检测、品质检测等智能化工作，实现控制器的数据采集、分析决策功能。

1.2 驱动控制器

驱动控制器是运动控制系统的转换装置，用于将来自运动控制器的控制信号转换为执行器的运动。更为先进的智能化驱动可以自身闭合电流环、速度环和位置环，实现更高精度的控制。典型的驱动控制器如变频器、步进驱动器、伺服驱动器。

1.2.1 变频器

变频器（Variable-frequency Drive，VFD）是应用变频技术与微电子技术，通过改变电机工作电源频率方式来控制交流电动机的电力控制设备。

变频器主要由整流（交流变直流）、滤波、逆变（直流变交流）、制动单元、驱动单元、检测单元、微处理单元等组成。变频器使用内部电路（当前通常使用 IGBT）的通断来调整输出电源的电压和频率，根据电机的实际需要来提供其所需要的电源电压，进而达到节能、调速的目的。另外，变频器还有很多的保护功能，如过流、过压、过载保护等等。随着工业自动化的发展，产品在成熟度和成本方面都得到了很大进展，变频器目前已经得到了非常广泛的应用。



图 1-10 变频器产品

变频器的选用

按照生产机械的类型、调速范围、静态速度精度、起动转矩的要求，决定选用哪种控制方式的变频器。所谓合适是既要好用，又要经济，以满足工艺和生产的基本条件和要求。

- 需要控制的电机及变频器自身

1）电机的极数。一般电机极数以不多于 9 极为宜，否则变频器容量就要适当加大。

2）转矩特性、临界转矩、加速转矩。在同等电机功率情况下，相对于高过载转矩模式，变频器规格可以降额选取。

3）电磁兼容性。为减少主电源干扰，使用时可在中间电路或变频器输入电路中增加电抗器，或安装前置隔离变压器。一般当电机与变频器距离超过 50m 时，应在它们中间串入电抗器、滤波器或采用屏蔽防护电缆。

- 变频器功率的选用

系统效率等于变频器效率与电动机效率的乘积，只有两者都处在较高的效率下工作时，则系统效率才较高。从效率角度出发，在选用变频器功率时，要注意以下几点：

1）变频器功率值与电动机功率值相当时最合适，以利变频器在高的效率值下运转。

2）在变频器的功率分级与电动机功率分级不相同时，则变频器的功率要尽可能接近电动机的功率，但应略大于电动机的功率。

3）当电动机属频繁起动、制动工作或处于重载起动且较频繁工作时，可选取大一级的变频器，以利用变频器长期、安全地运行。

4）经测试，电动机实际功率确实有富余，可以考虑选用功率小于电动机功率的变频器，但要注意瞬时峰值电流是否会造成过电流保护动作。

5）当变频器与电动机功率不相同时，则必须相应调整节能程序的设置，以利达到较高的节能效果。

- 变频器箱体结构的选用

变频器的箱体结构要与环境条件相适应，即必须考虑温度、湿度、粉尘、酸碱度、腐蚀性气体等因素。常见有下列几种结构类型可供用户选用：

1）敞开型 IP00 型本身无机箱，适用装在电控箱内或电气室内的屏、盘、架上，尤其是多台变频器集中使用时，选用这种型式较好，但环境条件要求较高；

2）封闭型 IP20 型适用一般用途，可有少量粉尘或少许温度、湿度的场合；

3）密封型 IP45 型适用工业现场条件较差的环境；

4）密闭型 IP65 型适用环境条件差，有水、尘及一定腐蚀性气体的场合。

- 变频器容量的确定

合理的容量选择本身就是一种节能降耗措施。根据现有资料和经验，比较简便的方法有三种：

1）电机实际功率确定法。首先测定电机的实际功率，以此来选用变频器的容量。

2）公式法。当一台变频器用于多台电机时，应满足：至少要考虑一台电动机启动电流的影响，以避免变频器过流跳闸。

3）电机额定电流法选用变频器。

变频器容量选定过程，实际上是一个变频器与电机的最佳匹配过程，最常见、也较安全的是使变频器的容量大于或等于电机的额定功率，但实际匹配中要考虑电机的实际功率与额定功率相差多少，通常都是设备所选能力偏大，而实际需要的能力小，因此按电机的实际功率选择变频器是合理的，避免选用的变频器过大，使成本增加。对于轻负载类，变频器电流一般应按 1.1N（N 为电动机额定电流）来选择，或按厂家在产品中标明的与变频器的输出功率额定值相配套的最大电机功率来选择。

对于一些特殊的应用场合，如高温，高海拔，此时会引起变频器的降容，变频器容量要放大一档。

- 主电源

1）电源电压及波动。应特别注意与变频器低电压保护整定值相适应，因为在实际使用中，电网电压偏低的可能性较大。

2）主电源频率波动和谐波干扰。这方面的干扰会增加变频器系统的热损耗，导致噪声增加，输出降低。

3）变频器和电机在工作时，自身的功率消耗。在进行系统主电源供电设计时，两者的功率消耗因素都应考虑进去。

1.2.2 伺服系统

伺服驱动器（servo drives）又称为“伺服控制器”、“伺服放大器”，属于伺服系统的一部分，主要应用于高精度的定位系统。一般是通过位置、速度和力矩三种方式对伺服电机进行控制，实现高精度的传动系统的控制。尤其是应用于交流永磁同步电机控制的伺服驱动器已经成为国内广泛采用的产品。当前交流伺服驱动器普遍采用基于矢量控制（又称磁场定向控制 Field Oriented Control）的电流、速度、位置闭环控制算法。直接转矩控制是交流永磁同步电机的另一种常见的控制方法。



图 1-11 典型的伺服驱动控制器

交流伺服系统被广泛的应用于,如:数控机床、医疗机械、汽车制造业、工业机器人等领域。现代交流伺服系统主要由四部分组成:伺服电机、功率变换器、控制器和检测器件。

目前主流的伺服驱动器多采用数字信号处理器（DSP）作为控制核心（随着 ARM 处理器的发展，其丰富的外设资源和数据处理能力，基于 ARM 处理器的伺服驱动器也越来越多），伺服驱动器可以实现比较复杂的控制算法，实现数字化、网络化和智能化。对于小功率伺服驱动器，功率器件普遍采用以智能功率模块（IPM Intelligent Power Module）为核心设计的驱动电路，IPM 内部集成了驱动电路，具有过电压、过电流、过热、欠压等故障检测、保护电路，通过在主回路中加入软启动电路，以减小启动过程对驱动器的冲击。功率驱动单元首先通过三相全桥整流电路对输入的三相电或者市电进行整流，得到相应的直流电。经过整流

好的三相电或市电，再通过三相正弦 PWM 电压型逆变器变频来驱动三相永磁式同步交流伺服电机。功率驱动单元的整个过程可以简单的说就是 AC-DC-AC 的过程。整流单元（AC-DC）主要的拓扑电路是三相全桥可控整流电路。

1.2.2.1 交流伺服系统主要性能指标

位置伺服系统的主要控制目标是输出值迅速跟踪指令值的变化。应用场合不同，对伺服系统的具体要求也会有所差异，但是大体要求是基本一致的，具体来说，在机电一体化产品中，对伺服系统的性能指标要求主要包括

（1）定位精度

系统最终定位点与指令目标值之间的静态误差即为定位精度，定位精度是评价位置伺服系统定位准确度的一个关键指标。这与伺服系统的位置反馈元件（如编码器）的分辨率和精度有关，对性能优异的交流伺服系统而言，应当满足±1 个脉冲的定位精度要求。

（2）调速范围

即电机最高转速与最低转速之比，用 D 表示。

$$D = n_{max} / n_{min}$$

上式中，n_{max} maxn 为最高转速，n_{min} 为最低转速。通常应满足 D≥10000 才能满足低速加工和高速返回的要求。

（3）调速静态特性

对绝大多数负载来说，机械特性越硬，负载变化时速度瞬态变化越小，工作越稳定，所以希望机械特性越硬越好。

（4）调速动态特性

动态特性，即速度变化的暂态特性，主要包括两个方面：一为升速和降速过程是否快捷、灵敏且无超调。这就要求电机转子惯量小，转矩/惯量比大，单位体积有较大的电机转矩输出。二是当负载突增突减时，系统的转速能否自动调节而迅速恢复。

1.2.2.2 伺服系统的选型

伺服系统在选型时，通常与伺服电机和伺服驱动器的上位控制系统整体考虑，会涉及到具体的应用需求（负载情况、性能需求、机械结构等）、技术方案、产品的可维护性、性能价格比等。通常，选择伺服驱动器从如下因素考虑：

1）额定输出功率/输出电流。该指标主要根据所选的伺服电机的功率和转速来确定。通常，厂家提供的是驱动和电机是有特定的组合配置。可以询问厂家技术人员，获取推荐配置。

2）供电电压。驱动器的供电电压，通常与电气方案和成本相关。通常有单相/三相 220V/110V 50/60HZ，三相 380V50/60Hz 的选择。驱动器还需要承受一定的电压波动范围，防止电压的波动带来的不利影响。

3）转速。该指标通常与效率和功率相关，获得满足应用的转速，选择对应的伺服系统（驱动器和电机）。调速比（如 ProNet Plus 的调速比为 1：5000）也是反映伺服驱动性能的一个关键指标。在一些高要求的情况下，有时会对电机的速度波动率有要求，以实现过程的精准控制。

4）控制接口的支持。运动控制器需要通过一些信号、指令实现对伺服系统的控制，通常有脉冲接口、模拟量接口、现场总线接口实现对伺服系统的运动控制。当伺服驱动器具有对应的控制接口，还需要确认驱动器所支持的协议和对应的功能，以满足控制系统对伺服系统实现应用的需要。

5）响应速度。该指标反映伺服系统的性能指标，实现指令的高速、高精度。为了保证生产率和加工质量，除了要求有较高的定位精度外，还要求有良好的快速响应特性，即要求跟踪指令信号的响应要快。如数控系统在启动、制动时，要求加、减加速度足够大，缩短进给系统的过渡过程时间，减小轮廓过渡误差。

5）过载能力。在负载不稳定的应用环境下，选择更高过载能力的伺服系统，可以使被控对象稳定的运行。通常，伺服系统（中小功率）具备 2~3 倍的过载能力。

6）结构与安装。当前伺服驱动器在向小而精致的方向发展，以减少对客户应用环境中的空间要求。伺服驱动器的结构、散热需求，会对电气柜的布局、布线，甚至是方案的需要考虑的。需要结合应用的需要，选择合适结构和安装需求的伺服系统。

此外，可能会考虑到：防护等级、工作温度范围、硬件接口、控制功能的支持，产品的易用性等。

阅读一些伺服系统的手册、样本，了解伺服系统的技术、指标参数，以及各自的意义，了解选型时的关注点。

ESTUN & TIO

1.2.2.3 发展趋势

工业的应用发展，伴随电子技术、计算机技术、信息技术的发展，伺服系统在数字化、网络化方面获得快速发展，同时也为伺服系统在智能化方面提供条件，随着应用的复杂性和客户对易用性的要求，伺服智能化需求越来越明显。功能安全是近年来获得市场的重视，具备有功能安全的伺服系统的产品在市场上越来越多。

（1）数字化

数字控制技术以其高可靠性和可控性，使其取代模拟信号控制伺服系统成为一种必然趋势。采用全数字处理器的伺服控制单元成为当前市场产品的主流，使产品更符合技术和市场的发展需要。通过伺服控制系统的数字化，使得先进的控制算法得以应用，使伺服的功能得到方便、灵活的实现。数字化控制使得伺服系统在响应性、控制精度、产品的可维护性方面得到全面提升。

（2）智能化

目前，高速、高性能的处理器发展，使得伺服驱动器具备有更多智能化的功能实现条件。如当前研究较多的参数自整定功能，使得伺服驱动器根据应用工况，设定自身的工作参数，降低了使用的复杂度。另外还有如故障自诊断与分析、在线惯量辨识、在线共振点辨识、低频震动抑制等功能，为伺服的易用性等用户体验提供提升途径。

（3） 网络化

现场总线技术的发展和技术的成熟为伺服系统的网络化提供了发展控制键，构建网络型、总线型伺服系统是当前各大主流伺服系统供应商的发展趋势和方向。现场总线是使得构建控制系统更为灵活、接线简便，实现现场设备和控制装置之间进行双向、串形、多结点的数字通信。网络化是实现信息化和获取伺服执行端信息化数据的途径，也是复杂和智能化的伺服系统的必然选择。

（4）高效化

更高位数的高精度编码器在伺服系统中被越来越广泛使用，采用更高精度的编码器，提高系统的采样精度，提高伺服驱动器的产品性能，获得更快数据处理速度和伺服系统的加减速性能，以满足是否应用场合对伺服性能要求的提高。

（5）功能安全

功能安全被越来越多的具有功能安全要求的场合所需要，应用也越来越广泛。例如，安全转矩截止 STO（Safe Torque Off），实现如飞车、短路、过压等保护功能对伺服安全使用至关重要，尤其在当前人机协作日益增多的情况下，STO 功能变得必不可少。例如 ESTUN Summa 系列伺服系统可满足 STO SIL3 安全等级很 Cat4 标准。其他的安全功能如：安全停车 SS1（Safe Stop 1），安全停车 SS2（Safe Stop 2），安全抱闸控制 SBC（Safe Break Control），安全速度限制 SLS（Safe Limit of Speed）等。

阅读伺服产品手册和相关书籍，了解伺服系统的关键技术指标（响应频率、高低速比、扭矩、额定转速、过载率）。

阅读伺服用户手册、了解伺服的规格参数、功能、接线、参数意义及配置、配置工具的使用。

阅读一份功能安全的论文/著作，如邢江红，刘艳强的著作。

1.2.3 步进电机驱动器

步进电机驱动器是将接收到的运动指令转换为步进电机的角位移（对应步距角）的执行

机构。通常情况下接收对应位移的脉冲信号时，当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个步距角，它的旋转是以固定的角度一步一步运行的。外部控制器可以通过控制脉冲个数来控制步进电机的角位移量，从而达到准确定位的目的。同时可以通过控制驱动器输入控制脉冲的频率来实现步进电机转动的速度和加速度，从而达到调速和定位的目的。步进系统被广泛应用于雕刻机、电脑绣花机、数控机床、包装机械、点胶机、切料送料系统、测量仪器等设备上。



图 1-12 步进电机驱动器

步进电动机不能直接接到直流或交流电源上工作，必须使用专用的驱动电源（步进电动机驱动器）。步进电机的相数是指电机内部的线圈组数，常用的有二相、三相、四相、五相步进电机。电机相数不同，其步距角也不同，一般二相电机的步距角为 1.8 度、三相为 1.2 度、五相的为 0.72 度。在没有细分驱动器时，用户主要靠选择不同相数的步进电机来满足步距角的要求。如果使用细分驱动器，则相数将变得没有意义，用户只需在驱动器上改变细分数，就可以改变步距角。

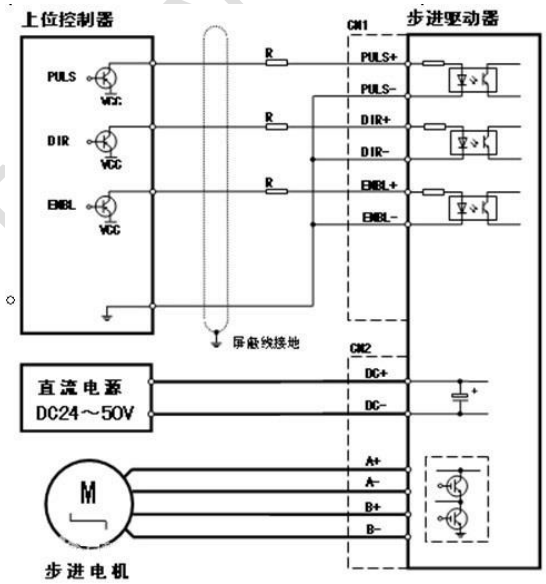


图 1-13 步进电机驱动器的常见连接

细分驱动技术通过控制各相绕组中的电流,使它们按一定的规律上升或下降，即在零电流到最大电流之间形成多个稳定的中间电流状态，相应的合成磁场矢量的方向也将存在多个稳定的中间状态，且按细分步距旋转。其中合成磁场矢量的幅值决定了步进电机旋转力矩的大小，合成磁场矢量的方向决定了细分后步距角的大小。细分驱动技术进一步提高了步进电机转角精度和运行平稳性。

1.2.3.1 主要性能指标

步进电机驱动器的基本参数和特性

a、供电电源，可据所驱动步进电机的电源规格进行选择。二相电机的供电一般为 12～48V。交流电源供电的，如 AC80V，可用 220V 市电经降压变压器，提供给驱动器。选用变压器时，须同时考虑电压和电流两方面的工作参数。如电流值 3A，由算式 80V*3A=240VA 得出的是视在功率值，达不到 3A 的实际输出能力，应将计算结果再乘以 1.5 以上的系数，选用 220/80V，400VA 以上的变压器，作为电源输入。

b、输出电流值。产品标注值往往为峰值输出能力，选用时，最低应按步进电机额定电流值的 2 倍以上。输出电流的档位，一般操作面板上的拨码开关进行人工整定，如 0.9A-3A。这也是一种过载保护整定，整定值可参考步进电机的工作电流值。一般有 8 级整定档次。

c、励磁方式：整步、半步、4 细分、8 细分、16 细分、32 细分、64 细分。半步实际上是 2 细分，细分级别越高，步矩角越小，而电机转速越低。步进驱动器的控制面板，也设有细分拨码开关，也对细分值进行设置。

d、保持转矩（HOLDING TORQUE）：是指步进电机通电但没有转动时，定子锁住转子的力矩。它是步进电机最重要的参数之一，通常步进电机在低速时的力矩接近保持转矩。由于步进电机的输出力矩随速度的增大而不断衰减，输出功率也随速度的增大而变化，所以保持转矩就成为了衡量步进电机最重要的参数之一。比如，当人们说 2N.m 的步进电机，在没有特殊说明的情况下是指保持转矩为 2N.m 的步进电机。

e、总线接口支持：总线型伺服成为当前新推出的产品主流，如支持 RS485 的 Modbus 协议，支持 CANopen CiA402 协议，以及当前应用广泛的 EtherCAT 的工业以太网协议。选择具备总线接口的驱动器，接口的类型和控制所支持的通信协议，与所选择的上层控制器有关，需要与之匹配。

1.2.3.2 步进电机驱动器的选型

由于步进电机及驱动器型号较多、种类较多，用户需要根据应用的需要选择合适的产品，这样才能获得相对最优的性能、最低的价格的所需产品。步进电机驱动器的选择时，通常要与所需步进电机相匹配。此外，还要考虑如下的一些因素：

1）步进电机驱动器的驱动电流。驱动电流是判断步进电机驱动能力的大小，是选择驱动器的重要指标之一。通常，要保证驱动器的最大驱动电流要略大于电机标称电流，驱动器的驱动电流通常有 2.0A，3.5A，6.0A，8.0A 等规格。

2） 步进驱动器的供电电压。供电电压是判断驱动器的升速能力的重要参数，通常有 24VDC，40VDC，80VDC，110VAC，220VAC 等。通常输入电压都是有一定的范围，且和电气方案相关。

3）步进驱动器的细分。通过细分的方法可以实现更高的步进电机转角精度，提升电机运行的平稳性。细分驱动技术已被广泛应用到步进驱动器中，成为普遍的功能配置。

4）确定步进电机的最高运行转速。转速指标在步进电机的选取时至关重要的考虑指标，步进电机的扭矩是随着电机转速的升高而下降，其下降的快慢和很多参数有关，如：驱动器的驱动电压、电机的相电流、电机的相电感、电机大小等等。一般规律是：驱动电压越高，

力矩下降越慢；电机的相电流越大，力矩下降越慢。需要根据负载最大力矩和最高转速这两个重要指标，再参考步进电机的矩频特性曲线，就可以选择出适合自己的步进电机和对应的步进驱动器。

5）控制模式。控制模式是指步进驱动器支持接收运动指令的形式。需要结合控制系统的整体方案、性价比来选择。通常有脉冲控制、模拟量控制、现场总线控制（如 CANopen, EtherCAT, Profinet 等）、I/O 控制等。当前以脉冲为主，且总线型驱动器正在被广泛应用。

6）通信接口和通信协议需求。需要步进伺服系统与控制器间进行信息交互的需求时，对驱动器支持的通信接口和通信协议要与控制器匹配。通常驱动器的通信接口有 Ethernet, RS-485 等，支持的协议有 TCP/IP, Modbus TCP, Modbus 等。

伺服驱动的选型还可能考虑到：降额使用（如散热不好、为复杂负载提供一定的波动裕度）、对产品的熟悉程度（选择熟悉的产品，可有降低人力投入、减少风险）、位置反馈需求（对于高精度要求场合，有时需要驱动器具有闭环控制要求，需要位置反馈接口。）、产品的市场接受度（具有普遍的市场接受度的产品，通常具备较好的产品可靠性、可获得性和易用性等）。

1.2.3.3 步进电机驱动器的发展趋势

步进电机驱动器涉及到多学科的技术，如计算机、电力电子、材料、信息技术、自动控制等，其发展和进步会随着这些领域的进步而更新、迭代。目前，驱动器的发展趋势有：

（1）不断顺着小而精致的方向发展。

（2）功能组一体化设计。如把转子位置传感器和电动机本体设计在一起组成一个闭环系统，把减速齿轮以及滑动或滚动丝杠等和电动机本体设计在一起组成变速机构和直线驱动机构，因而具有更加优越的控制性能。还有一些将驱动器和电机本体设计在一起，简化了用户使用时在驱动和电机间的接线，适用一些驱动与电机距离远的场合。

（3）进一步提高运行速度和输出功率，进一步提高响应速度。

（4）采用细分驱动控制技术及专用的控制芯片。通过步进电机的细分驱动技术，使得电机获得更高精度、运行的平稳性提升。专用的控制芯片为步进电机的控制提供为稳定、简便的方案，集成了如欠压、过流等保护功能。提升了驱动器的可靠性和性能。

（5）集成现场总线通信接口：现场总线的控制优势，可以降低采用脉冲控制和电压控制时干扰引起的问题，且接线简便。近年来得到大力的发展和广发的应用。

阅读步进电机驱动器手册，了解步进电机驱动器的规格参数、功能与使用。

阅读一份步进电机细分驱动技术相关技术资料或论文，了解技术原理。

ESTUN & Trio

1.3 执行器

执行器是运动控制系统中的控制对象，用于将驱动信号转换为位移、旋转等，典型的执行器如图 1-14 所示。



图 1-14 典型的执行器

通常，执行器通过控制直线式或旋转式的设备来实现被控对象的运动。常见的如各种类型的电机、液压、启动设备。这些执行器通过一些机械机构连接，实现控制对象的运动。常见的传动机构有：齿轮传动、带传动、丝杆传动，链传动、液压传动、气压传动等。本章节主要介绍部分常见旋转类电机执行器。

1.3.1 电机的种类

电机为运动控制系统的动力来源，将电能转化为动能，实现所需的运动和动力。其种类繁多分类方法也多种多样。从运动形式上来说，运动式的电机分为旋转运动和直线运动的电机。从结构上来说，电机包含定子与转子两大部分。有的电机定子在外侧转子在内侧；而有的电机则正好相反。从电机工作时所需的是直流电还是交流电，可分为直流电机和交流电机。

对于各种电机的具体结构和工作原理，在此就不做具体介绍，大家可以查阅相关的资料了解。

目前运动控制系统中采用的电机种类主要有交流异步电机、步进电机、交流伺服电机、直流伺服电机等。对于非驱控一体的系统，在运动控制系统中，为了实现被控对象的速度、位置或力矩控制，还需要电机对应的驱动器实现电机的驱动与控制，使其运转（如步进电机和伺服电机）。

这里重点介绍具有位置控制功能的步进电机、交流伺服电机这类在通用运动控制系统中常用的电机，重点介绍电机的控制方法、关键技术参数、关键选型指标。

1.3.2 步进电机

步进电机是一种将电脉冲信号转换成相应角位移或线位移的电动机。每输入一个脉冲信号，转子就转动一个角度或前进一步，其输出的角位移或线位移与输入的脉冲数成正比，转速与脉冲频率成正比。因此，步进电动机又称脉冲电动机。

ESTUN & Trio

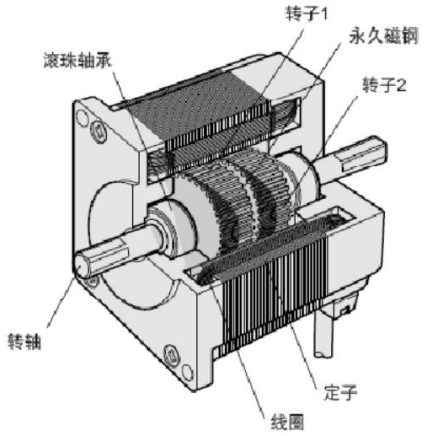


图 1-15 步进电机的构造图

步进电机相对于其它控制用途电机的最大区别是，它接收数字控制信号（电脉冲信号）并转化成与之相对应的角位移或直线位移，它本身就是一个完成数字模式转化的执行元件。而且它可开环位置控制，输入一个脉冲信号就得到一个规定的位置增量，这样的所谓增量位置控制系统与传统的直流控制系统相比,其成本明显减低，几乎不必进行系统调整。步进电机的角位移量与输入的脉冲个数严格成正比，而且在时间上与脉冲同步。因而只要控制脉冲的数量、频率和电机绕组的相序,即可获得所需的转角、速度和方向。

步进电机的转速取决于控制信号的脉冲频率、转子齿数和拍数。其角速度与脉冲频率成正比，而且在时间上与脉冲同步。在启动或加速时如果步进脉冲变化太快，转子由于惯性而跟随不上电信号的变化，产生堵转或失步。在停止/减速过程中，由于同样原因，可能产生过冲（又称超步）。为防止堵转、失步和超步，提高/降低工作频率（电机速度），要对步进电机进行升/降速控制。

因而在转子齿数和运行拍数一定的情况下，只要控制脉冲频率即可获得所需速度。步进电机的输出力矩随着控制信号脉冲频率的上升而下降，启动频率越高，启动力矩就越小，带动负载的能力越差。由于步进电机是借助它的同步力矩而启动的，为了不发生失步，启动频率是不高的。特别是随着功率的增加,转子直径增大，惯量增大，启动频率和最高运行频率可能相差十倍之多。

因此,步进电机的运行过程一般要经过加速阶段、匀速阶段、减速阶段三个阶段。为了达到快速完成运行距离。需要是加减速过程时间尽量的短，恒速时间尽量长。

选择步进电机时，结合应用需求，关注电机的技术规格是否满足需求。主要的参数包括：

1）确定步进电机的步距角。电机的步距角取决于负载精度的要求，将负载的精度要求换算到电机轴的角度，使得该角度大于或等于步距角。目前市场上常见的步距角有 0.36 度/0.72 度（五相电机）、0.9 度/1.8 度（二、四相电机）、1.5 度/3 度（三相电机）等。

2）确定电机的静力矩。步进电机的动态力矩比较难以确定，往往先确定电机的静力矩。静力矩的选择依据电机工作的负载，而负载可分为惯性负载和摩擦负载两种，单一的惯性负载和摩擦负载是不存在的，直接启动时（一般低速）时，两种负载均要考虑。加速阶段主要考虑惯性负载，匀速阶段主要考虑摩擦负载。一般情况下，选择静力转矩为摩擦负载的 2~3 倍，精力转矩确定后，电机的机座及长度即可确定下来。

3）保证负载惯量与电机转惯量匹配。电机的惯量匹配性关系到电机的稳定性和精确度，惯量越小，精度遇到高，惯量越大，稳定性越高。在电机选型时，需要结合负载惯量选择合适的惯量比电机，来获得系统稳定性和精度，甚至成本的平衡。通常需要保证电机的转子惯量>负载惯量/（J*i^2）,其中 J 为惯量匹配系数，一般为 3~20，具体咨询电机供应商，i 为减速比，直联是 i=1。

了解电机惯量和负载惯量的意义和计算方法。

1.3.3 伺服电机

交流伺服电机目前主要有两种：异步电机和永磁同步电机。异步电机又名感应电机，具有制造简单、价格低廉和维护方便等优点。但异步电机存在如下缺点:转子为通电线圈，因此增加了电机的损耗；异步电机需要吸收滞后的励磁电流，因此功率因素较低；异步电机在矢量控制时，需要直接测量或者间接计算磁通量，因此控制相对复杂。永磁同步电机根据反电动势的形式可以分为两种：一种为正弦波，即狭义的永磁同步电机；另一种为梯形波，即无刷直流电机。它们都具有以下优点：由于转子采用铁氧体和高性能的稀土钕铁硼永磁材料，因此转子无励磁损耗,提高了效率,减少了能耗；同时由于转子无励磁损耗,发热主体在定子侧，电机的散热效果更加好；从控制角度看，永磁同步电机的励磁磁通大小不变且与电枢电流有着固定的相位关系，因而比异步电机的矢量控制简单。由于以上诸多优点,永磁同步电机逐渐在伺服系统中占据主导地位。

这里主要介绍工业自动化中应用最为广泛的交流永磁同步电机。从结构上，交流伺服电动机主要由两部分组成，即定子部分和转子部分。其中定子的结构与旋转变压器的定子基本相同，在定子铁心中绕制了励磁线圈绕组。而转子却不同，按不同的转子结构又分电磁式及非电磁式两大类。非电磁式又分为磁滞式、永磁式和反应式多种。其中磁滞式和反应式同步电动机存在效率低、功率因数较差、制造容量不大等缺点。永磁式伺服电机由于其制造工艺的成熟和电子技术、计算机技术的发展，获得了广泛的市场应用。



图 1-16 交流永磁同步电机

交流永磁同步伺服电机一般在本体上集成了编码器的位置采集传感器，为伺服驱动器提供位置反馈信息，实现位置闭环系统，获得精准的位置控制和性能。

选择电机主要从以下几方面考虑:

(1)系统的精度要求；

(2)连续运转速度和最高速度，最高加速度要求；

(3)负载情况；

(4)惯量匹配情况。合适的惯量匹配可以获得合适的系统响应性。通常使伺服电机的惯量比（折算到电机轴上的负载惯量/电机转子惯量，最佳惯量匹配比为 1）控制在 5~10 倍;；

(5)机械传动结构；

(6)初步选定电机；

(7)计算所选电机的峰值力矩和额定力矩是否满足要求。

伺服系统选型之力矩匹配主要考虑:

(1)峰值力矩:加减速要求,电机转子惯量!负载惯量"

(2)额定力矩:负载力矩要求,速度运行图,最佳速度运行图,梯形曲线,

永磁同步电机 PMSM 的工作原理、不同结构的 PMSM 电机的特点和应用场合。

阅读公司伺服电机数据手册、理解各个规格参数的意义、对应用的影响

1.4 传感器

传感器是能感受规定的被测量并按照一定的规律转换成可用的输出信号的器件或装置。传感器的输出信号通常是电量，它便于传输、转换、处理、显示等。电量有很多类型，如电压、电流、电容、电阻等，输出的信号的形式有传感器的原理确定。

通常，传感器由敏感元件和转换元件组成，如图 1-17 所示， 其中敏感元件是指传感器中能直接感受或响应被测量的部分；转换元件是指传感器中能将敏感元件感受或响应的被测量转换成适合传输或测量的电信号部分。由于传感器输出信号一般都很微弱，需要有信号调理与转换电路，进行放大、运算调制等，此外信号调理转换以及传感器的工作必须有辅助的电源，因此信号调理转换电路以及所需的电源都应作为传感器组成的一部分。随着半导体器件与集成技术在传感器中的应用，传感器的信号调理转换电路与敏感元件一起集成在同一芯片上，安装在传感器的壳体里。

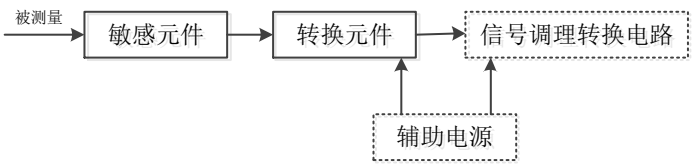


图 1-17 传感器的组成框图

传感器的种类繁多，分类方法也很多，但目前一般采用两种分类方法。一种是按被测参数分类，如温度、压力、位移、速度等；另一种是按传感器的工作原理分类，如应变式、电

容式、压电式、光电式、磁电式等。这里我们主要以工程应用进行介绍，以被测参数来分类说明，重点介绍运动控制系统中常用的传感器，如位移传感器、压力传感器等、包括其工作原理、主要技术指标与选型。

1.4.1 编码器

将机械转动的模拟量（位移）转换成以数字代码形式表示的电信号，这类传感器称为编码器，编码器以其高精度、高分辨率和高可靠性被广泛应用于各种位移的测量。

编码器的种类很多，主要分为脉冲盘式（增量式编码器）和码盘式编码器（绝对值编码器），码盘式编码器目前有接触式编码器、电磁式编码器和光电式编码器。

脉冲盘式编码器的输出是一系列脉冲，需要一个计数系统对脉冲进行加减（正向或反向旋转时）累积计数，一般还需要一个基准数据即零位基准，才能完成角位移测量。绝对编码器不需要基准数据及计数系统，它在任意位置都可给出与位置相对应的固定数字码输出，能方便地与数字系统连接。

编码器按其结构有接触式、光电式、电磁式等，后两种为非接触式编码器。非接触式编码器具有非接触、体积小和寿命长，且分辨率高的特点。三种编码器相比较，光电编码器的性价比最高，它作为精密位移传感器在自动测量和自动控制技术中得到了广泛的应用。

1.4.1.1 增量式编码器

增量式编码器是将角位移转换成周期性的脉冲信号或正余弦信号，根据编码器线数不同，一个周期信号对应的角位移量也不同，一般有 512 线、1024 线、2048 线等，线即是每圈产生的周期信号数。编码器主轴和被测对象（如电机）等旋转轴链接到一起，并随之旋转。对于常用的光电编码器，其内部光栅码盘上刻有等间距的缝隙，固定光栅上也刻有等间距的狭缝，光栅码盘随着主轴一起旋转，当与固定光栅的狭缝重叠时，发光二极管透过棱镜与狭缝形成的莫尔条纹被光敏元件接收，经过光电转换电路形成两路相位相差 90° 的 TTL 或正余弦周期性电信号。

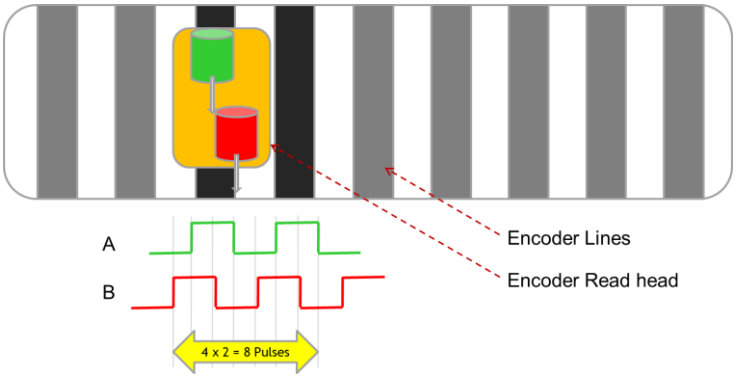


图 1-18 增量式编码器工作原理

增量式编码器是目前使用最为广泛的编码器。通常，其成本较其他类型编码器低。增量式编码器随着编码器主轴的旋转，产生一些列的脉冲信号，实现对旋转装置旋转量的检测。通常，除输出 A,B 相的脉冲信号外，每圈/全量程（对线性检测传感器而言，如光栅尺）固定位置还会输出一个 C 脉冲（有时称为 Z 脉冲，或标志信号）。C 脉冲通常用于特定参考位

置检测，如回零。有时，为了实现编码器信号的远距离传输或共模干扰的抑制，会将编码器信号通过差分方式传输。

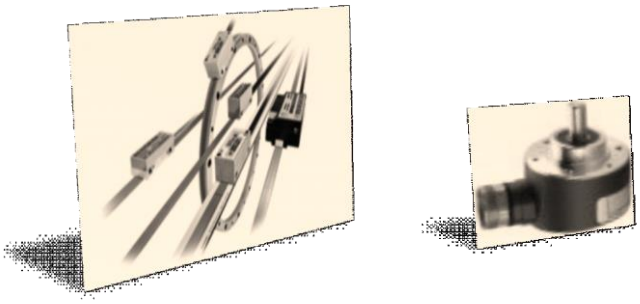


图 1-19 增量式编码器产品

1.4.1.2 绝对值编码器

绝对值编码器是将角位移转换为相应数字量，每个位置对应一个确定的数字码，按测量范围可分为单圈式和多圈式，前者测量范围在 360°以内，后者可测量大于 360°的范围。对于 n 位绝对式编码器，其码盘上有 n 道刻线，每道刻线依次为 2 线、4 线、8 线等，以读取每道刻线明暗的方式，获取一组唯一的二进制格雷码。



图 1-20 绝对值编码器码盘示例

绝对值编码按其机构分为：单圈和多圈绝对值编码器。按原理分为磁绝对值编码器，光电绝对值编码器。在选择编码器时还要关注编码器的尺寸、安装方式，出线位置、防护等级、精度、传输协议等因素。

绝对式编码器精度分为：单圈精度和多圈精度，加起来是总精度，也就是通常的多少位（常规 24 位，25 位，30 位，32 位。。。。）。

绝对值编码器的信息传输协议多样，如 SSI、4-20MA、profibus-dp、DEVicenet、并行、二进制码、BiSS、ISI、CANopen、Endat 及 Hiperface 等。较为常用的如 SSI 和 Endat。

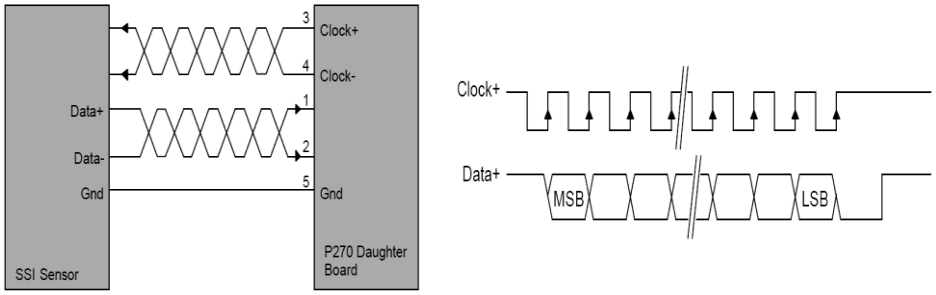


图 1-21 绝对值编码器的 Synchronous Serial Interface (SSI)传输

位置检测传感器，除了常见的编码器外，光栅、磁栅、电容式、电感式、电阻式位移传感器在运动控制系统中使用，进行位置检测、反馈。

旋转变压器是一种基于磁性原理的旋转编码器，通常简称旋变。它环境适应性强、响应速度快、可靠性高。其输出绕组的端电压随转子位置发生变化。可以通过测量该电压获取转子位置信息。旋变也常用于运动控制系统中，如检测电机的转动角度等。

学习光栅、磁栅、电容式、电感式传感器的工作原理、特性及选型知识。

ESTUN & Trio

1.4.2 模拟量位移传感器

模拟量的位移传感器常用于有限行程的应用中，以电压方式提供绝对的位置反馈信息。其分辨率取决于所测量的长度和模拟量信号的位分辨率。典型的模拟量输入为 12bits。提供 2^{12}=4096 个增量，若 100mm 代表所有的变化量，则其分辨率为 100/4095 = 0.0244mm 的分辨率。



图 1-22 模拟量传感器

模拟量传感器的信号传输非常容易受到一些电气噪声的影响，在布线和传输线的线的选择上，需要注意。通常需要采用低压差分信号的方式进行传输，以抑制共模干扰。

1.4.3 部分技术参数

1.4.3.1 分辨率

在运动控制中，分辨率是指反馈器件（通常指传感器）能够测量的最小单位。例如，当1000 个脉冲对应 1mm 的线性编码器，那么该线性编码器的分辨率是 1μm。对于连接在 4 毫米螺距的丝杠上的旋转编码器，若旋转编码器每转输出 4000 脉冲，则其分辨率 1μm。

对于输出正交的编码器脉冲信号，通常经过解码，可以实现 4 倍的分辨率。对于闭环系统，系统会对反馈系统获取的信息，获取当前的误差，然后更具误差的大小进行控制、纠正。

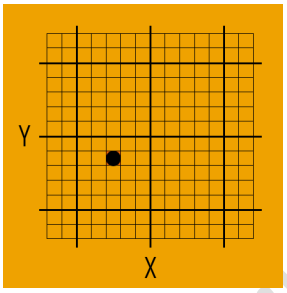


图 1-23 分辨率示例说明

上图的网格代表充传感器反馈的 XY 平面的位置。对于低分辨率的传感器，图中的黑点在其中移动较大范围时，才能够在传感器的反馈信号汇总反映出来。对于高分辨率的传感器，黑点的微小移动都会被传感器检测到。传感器的分辨率越高，将以利于系统误差识别。但从经验来看，分辨率的选择需要与所需性能和器件的分辨率做一个平衡选择。

1.4.3.2 可重复性与精度

在运动控制系统中，可重复性是指系统能够重复到达相同点的能力。到达目标位置的偏移即使可重复性误差。运动的精度为到达目标位置的精确程度。例如：如果需要在一个直径为 1mm 的孔位于 XY 位置，插入一个直径为 0.998mm 的引脚，必须有一个比 1μm 更好的 XY 运动的综合精度和重复性，才能保证引脚一直有效地插入孔中。

1.4.4 传感器的选择原则

传感器的选择原则如下：

- 技术要求。 传感器应满足一定的静态和动态特性的要求，主要有线性度、测量范围、灵敏度、分辨率、重复性等静态性能要求和响应速度、稳定性以及可靠性等动态特性要求。
- 使用环境要求。 使用环境包括环境温度、湿度、振动、磁场/电场及周围气体环境等，对传感器的使用精度、寿命等有影响、应予以考虑。
- 电源要求。 包括电源电压等级、形式、纹波系数等，一些应用场合还需要考虑频率和功耗的影响。
- 安全要求。 包括安装场所的安全性、是否需要采用本安设备、隔爆或采取其他安全

措施，此外，也需要考虑操作人员的安全，例如，对传感器防护外壳的防护等级考虑等。

- 维护和管理的要求。 包括维护的方便性、备品备件的可获得性、是否有自诊断功能、是否采用模块结构、更换的复杂度和对维护人员的技术要求等。
- 性能价格比的要求。在满足检测要求的前提下，应具备较高的性能价格比、较长的平均无故障时间和较短的平均维修时间等。
- 与可获得性相关的要求。包括产品的交货期、保修期、生命周期和备件的交货期等。
- 操作简单、安装方便，节能，性价比高。

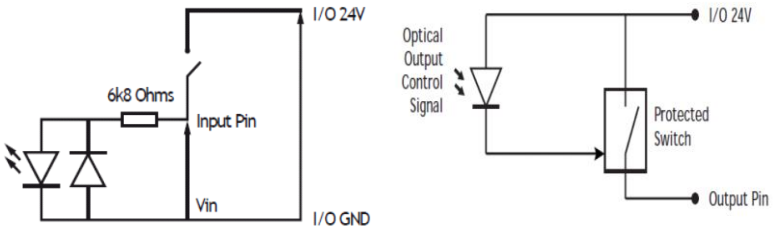
学习了解传感器的静态特性和动态特性。

1.5 控制器硬件接口

运动控制器的控制过程需要通过其硬件接口来采集和输出信号，实现控制目的。

1.5.1 数字量输入/输出

数字量输入/输出接口是 PLC 和运动控制系统中最为常用、且可以说是不可或缺的。 为了降低外部电气干扰信号对接口控制器内部电路的影响，通常采用光耦隔离的方式将数字量输入/输出信号隔离。如图 1-24 所示的数字量输入和数字量输出的常用接口设计。



数字量输入接口　　数字量输出接口

图 1-24 数字量输入/输出接口

数字量输入和输出为系统提供开关量信号输入和输出，具有 0 和 1 两种状态。对于接入数字量输入接口的信号，需要一定的电压来驱动内部电路。对于作为系统中断输入的信号，系统内部可以检测数字量输入的高/低电平、上升沿/和下降沿来触发对应的中断处理程序。对于数字量输入和输出接口，在使用时需要注意其接口的类型（如 PNP 和 NPN 型）与接口的驱动能力。通常，对于直接控制电磁阀的输出接口，需要较大电流驱动的接口，会在输出接口增加驱动，如继电器或 MOSFET，实现大电流的驱动，使用 MOSFET 的场合，还可以实现对设备的快速控制，如激光器的光栅输出控制接口。输入输出接口的响应速度有时也是一

个重要的参数。

对于继电器输出的端口，往往需要提供系统外的供电，实现外部设备的控制。端口的输出电流参数需要参考产品的技术参数，避免因过流或过压导致端口或外部设备损坏。继电器输出可以实现常开和常闭的输出，如图 1-25 所示。

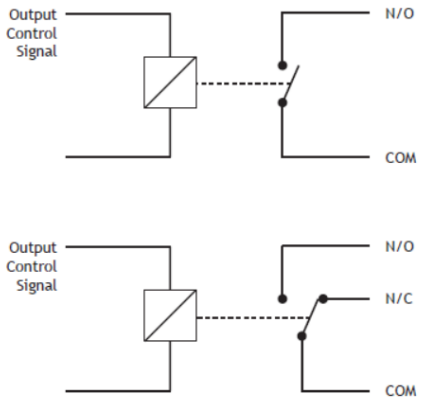


图 1-25 继电器输出的数字量输出端口

数字量输入和输出的配置数量是在做系统选型时考虑的一个相对重要和频繁的因素，为此有些灵活的控制系统，通过提供可以让用户自由选择的数字量方向（输入/输出）的接口，让用户灵活设置。

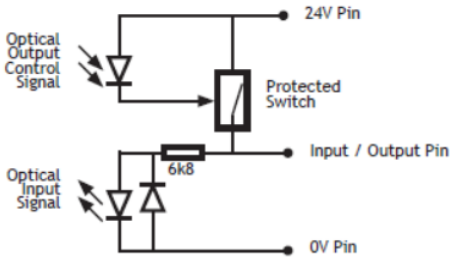


图 1-26 双向（输入/输出）数字量信号接口

常见的连接到数字量输入的外部信号有：限位开关信号、接近开关信号、气缸的位置检测信号、光幕报警输出信号、急停开关信号、模式开关信号灯；

常见的连接到数字量输出的外部信号有：系统准备好输出、报警信号输出、指示灯控制、开关阀控制信号，气缸的控制信号等。

阅读技术手册/使用手册，了解数字量输入/输出接口的参数，如何选择输入/输出端口，以及如何连接设备（接线）。

1.5.2 模拟量输入/输出

模拟量输入接口通常用于连接外部传感器输出信号，如温度传感器、压力传感器调理后的输出信号等。控制系统的模拟量输入接口，有相应的电压/电流范围、分辨率等参数，传感器的输出信号范围要与接口的信号类型和信号采集范围相对应。

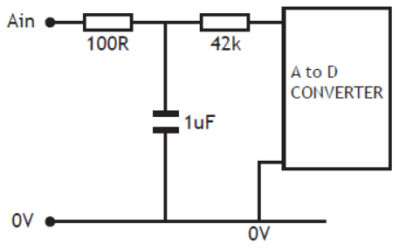


图 1-27 模拟量输入接口

获取外部输入的模拟量信号后，在控制器内部处理该信号时，需要考虑外部信号的噪声、线性度、以及自身的转换进度和转换速度。然后再转换为实际的控制量，如长度、温度、压力等。

模拟量输出在运动控制系统中，通常用于伺服轴转速/扭矩、液压比例阀等的控制。输出电压/电流的范围需要结合被控对象的接口参数设置，避免超出被控对象的模拟量输入信号的输入范围，而失去控制作用。

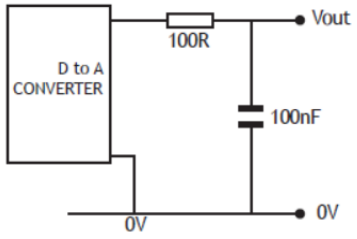


图 1-28 模拟量输出接口

模拟量输入和输出信号都容易受到外部干扰，在应用中要注意这些信号的抗干扰处理。如使用多层屏蔽线缆、远离干扰源的路径进行布线。干扰往往是不可避免的，因此对于采样获得的模拟量信号，必要时在软件里做去干扰处理。

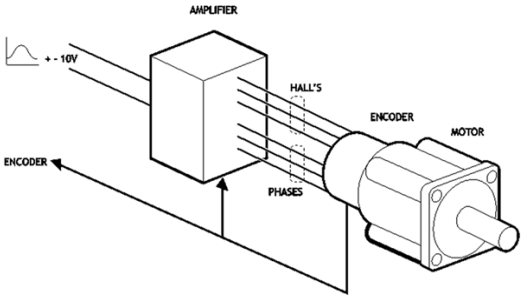


图 1-29 模拟量输出的电机控制示例

在使用伺服电机实现速度控制的场合，我们可以选用模拟量来实现电机的速度控制，模拟量的值决定了电机的运行速度。模拟量有两种方式可以选择，电流或电压。电压方式，只需要在控制信号端加入一定大小的电压即可。实现简单，在有些场景使用一个电位器即可实现控制。但选用电压作为控制信号，在环境复杂的场景，电压容易被干扰，造成控制不稳定；电流方式，需要对应的电流输出模块。但电流信号抗干扰能力强，可以使用在复杂的场景。

阅读技术文档，了解模拟量输入 AD 转换器的分辨率、量化误差、转换精度、转换时间与转换速率等概念。

1.5.3 脉冲输入输出

在一些小型单机设备，选用脉冲控制实现电机的定位，应该是最常见的应用方式，这种控制方式简单，易于理解。基本的控制思路：脉冲总量确定电机位移，脉冲频率确定电机速度。选用了脉冲来实现伺服电机的控制，通常有如下的指令脉冲状态，实现脉冲的驱动，如图 1-30。

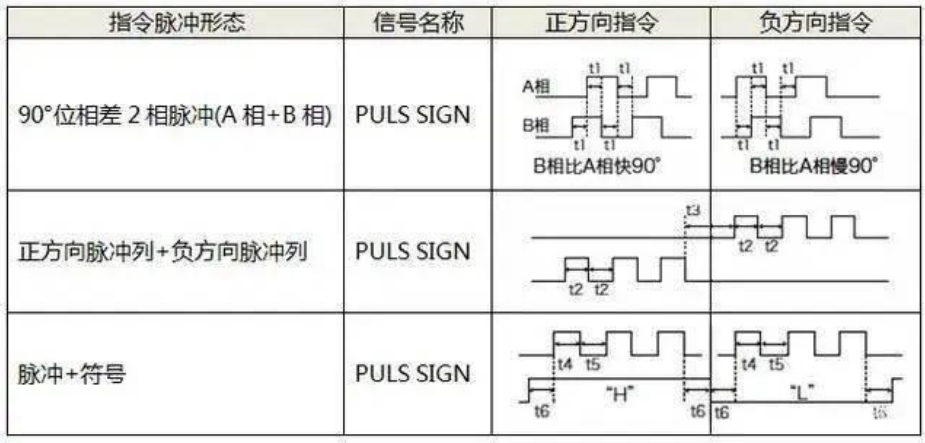


图 1-30 脉冲输出控制方式

第一种，驱动器接收两路(A、B 路)高速脉冲，通过两路脉冲的相位差，确定电机的旋转方向。如上图中，如果 B 相比 A 相快 90 度，为正转；那么 B 相比 A 相慢 90 度，则为反转。运行时，这种控制的两相脉冲为交替状，因此我们也叫这样的控制方式为差分控制。具有差分的特点，那也说明了这种控制方式，控制脉冲具有更高的抗干扰能力，在一些干扰较强的应用场景，优先选用这种方式。但是这种方式一个电机轴占用两路高速脉冲端口。对于脉冲的输出频率或接收频率范围，是应用时选择的一个重要指标。

第二种，驱动器依然接收两路高速脉冲，但是两路高速脉冲并不同时存在，一路脉冲处于输出状态时，另一路必须处于无效状态。选用这种控制方式时，一定要确保在同一时刻只有一路脉冲的输出。两路脉冲，一路输出为正方向运行，另一路为负方向运行。和上面的情况一样，这种方式也是一个电机轴需要占用两路高速脉冲端口。

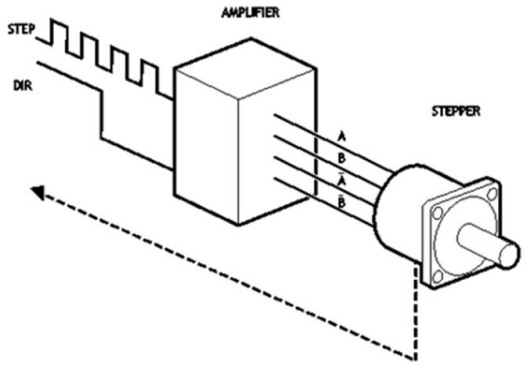


图 1-31 步进电机的脉冲控制示例

第三种，只需要给驱动器一路脉冲信号，电机正反向运行由一路方向 IO 信号确定。这种控制方式控制更加简单，高速脉冲端口资源占用也最少。在一般的小型系统中，可以优先选用这种方式。

在运动控制中，脉冲输入接口通常用于位置检测/速度检测。在使用脉冲输入接口时，需要了解接口所支持的信号、接口的电气参数，如支持的输入脉冲频率范围，使外部连接的信号与接口兼容。有时为了设备的体积、成本考虑，会将将输入接口的信号复用，根据用户的需要选择对应的功能。在 Trio 的运动控制器中，会将脉冲输入和输出接口做成灵活的，可以根据用户的需要，选择。

SYNC ENCODER

The sync encoder port is bidirectional. It can be used as a reference encoder input or as an encoder simulation output to act as a master reference for other parts of the system.

Pin	Encoder	Absolute	Pulse & Direction
1	Enc. A	Clk +	Step +
2	Enc. /A	Clk -	Step -
3	Enc. B	N/C	Direction +
4	Enc. /B	N/C	Direction -
5	0V Encoder	0V Enc.	0V Stepper
6	Enc. Z	Data +	Enable +
7	Enc. /Z	Data -	Enable -
8	5V*	5V	5V*
9	Registration Input (5V)		
*5V supply is limited to 150mA (shared with serial port)			

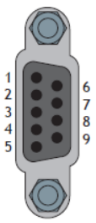


图 1-32 脉冲输入/输出接口示例 1

FLEXIBLE AXIS PORT

Pin	Encoder	Stepper Axis	Absolute Encoder
1	Enc. A	Step +	Clock
2	Enc. /A	Step -	/Clock
3	Enc. B	Direction +	--------
4	Enc. /B	Direction -	--------
5	0V Serial/Encoder	0V Serial/Encoder	0V 0V Serial/Encoder
6	Enc. Z	Enable +	Data
7	Enc. /Z	Enable -	/Data
8	5V*	5V*	5V*
9	Not Connected	Not Connected	Not Connected

*Current limit is 150mA max. Shared with serial port.

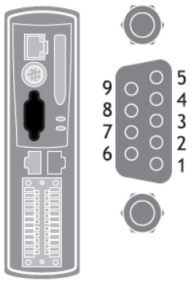


图 1-33 脉冲输入/输出接口示例 2

采用远程 IO 模块的方式，可以使现场总线的运动控制系统更具灵活性和扩展性。便于布线和远距离的设备控制。



图 1-34 EtherCAT 远程 IO 模块

1.5.4 通信接口

具有通信接口的自动化控制设备，几乎是运动控制和伺服驱动器不可或缺的配置接口。以实现设备与外部进行信息交互，是实现信息化的支持。常见的接口有 RS232, RS485，RS422, CAN, USB, Ethernet。

1.5.4.1 RS-232

RS-232 接口符合美国电子工业联盟（EIA）制定的串行数据通信的接口标准，原始编号全称是 EIA-RS-232（简称 232，RS232）。它被广泛用于计算机串行接口外设连接。连接电缆和机械、电气特性、信号功能及传送过程。而工业控制的 RS-232 口一般只使用 RXD、TXD、GND 三条线。

RS-232 是现在主流的串行通信接口之一。由于 RS232 接口标准出现较早，难免有不足之处，主要有以下四点：

（1）采用负逻辑传输。规定逻辑“1”的电平为-5V~-15 V，逻辑“0”的电平为+5 V～+15 V。选用该电气标准的目的在于提高抗干扰能力，增大通信距离。RS -232 的噪声容限为 2V，接收器将能识别高至+3V 的信号作为逻辑“0”，将低到-3V 的信号作为逻辑“1”。

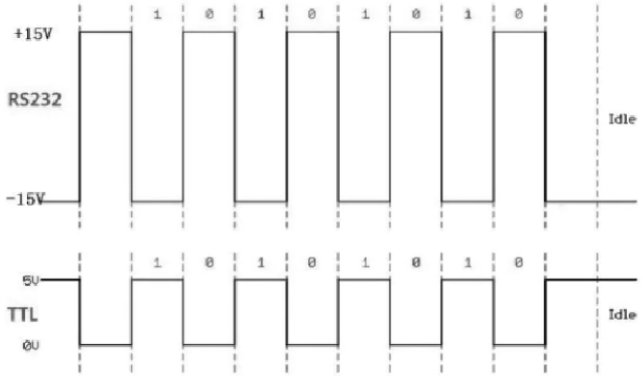


图 1-35 RS232 电平与 TTL 电平

TTL 电平为 5V 为逻辑正，0 为逻辑负。与 TTL 电平不兼容故需使用电平转换电路方能与 TTL 电路连接。

（2）传输速率较低。RS-232-C 标准规定的数据传输速率为每秒 1200、2400、4800、9600、19200，38400,115200 等波特率。通常波特率越高，其传输的距离越短。

（3）接口使用一根发送信号线和一根接收信号线，构成共地的传输形式，这种共地传输容易产生共模干扰，所以抗噪声干扰性弱。因此传输距离短，通常在 15 米左右。

（4）RS-232-C 接口在总线上只允许连接一个收发器，即整个网络只有两个设备。且数据可以双向传输，可以实现全双工。但必须保证收发数据双方的波特率一致。

1.5.4.2 RS485

RS-485 采用平衡发送和差分接收，使用差分信号进行数据传输，有效抑制共模干扰。使得通信距离达到几十米到上千米。加上总线收发器具有高灵敏度，能检测低至 200mV 的电压，故传输信号能在千米以外得到恢复，通过增加中继器可以传递得更远。因此 RS-485 串行总线在工业环境中获得了广泛的应用。

RS-485 可以采用二线与四线方式，二线制可实现真正的多点双向通信。而采用四线连接时，与 RS-422 一样只能实现点对多的通信，即只能有一个主（Master）设备，其余为从设备。在工业自动化应用中，以二线方式的 RS485 为主。且主要以总线型网络拓扑为绝大部分应用。如图 1-36 所示的 RS485 网络拓扑图。

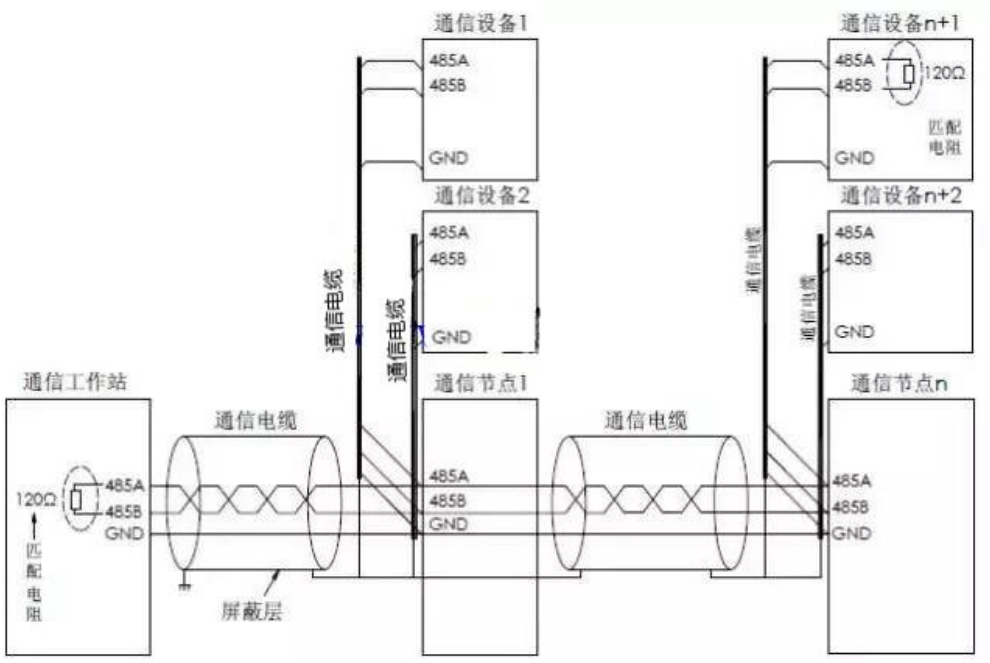


图 1-36 RS485 常用网络拓扑图

在总线型拓扑网络布线时，需要注意：

（1）尽可能地缩短分支的长度（即通信电缆到通信设备 1,2,，…N-1, N 的长度）

（2）工业应用中，建议选择具有屏蔽层的双绞线作为通信电缆。

（3）屏蔽层采用单端接地措施。

（4）长距离通信时，需要增加终端电阻，以抑制信号的反射，提升通信稳定性。终端电阻的大小需要结合通信线路的阻抗特性选择，示例中为常用的 120Ω。

采用二线方式的 RS-485 为半双工工作方式，任何时候只能有一点处于发送状态，因此在 RS485 中通常只有一个主设备来控制网络中设备的数据通信。RS-485 需要 2 个终接电阻，其阻值要求等于传输电缆的特性阻抗。在短距离传输时可不需终接电阻，即一般在 300 米以下不需终接电阻。终接电阻接在传输总线的两端。相对于 RS-232-C 的不足， RS-485 具有以下特点：

（1）RS-485 的电气特性：逻辑“1”以两线间的电压差+2V~+6V 表示，逻辑“0”以两线间的电压差-6V~-2V 表示。当在收端 AB 之间有大于+200mV 的电平时，输出正逻辑电平，小于-200mV 时，输出负逻辑电平。接收器接收平衡线上的电平范围通常在 200mV 至 6V 之间。接口信号电平比 RS-232-C 降低了，就不容易损坏接口电路芯片，且该电平与 TTL 电平兼容，可方便与 TTL 电路连接。

（2）数据最高传输速率为：10Mbps

（3）RS-485 接口采用平衡驱动器和差分接收器的组合，抗共模干扰能力强，即抗噪声性能好。可以实现更远距离的数据传输，实际上可达 3000 米甚至更远距离。

（4）而 RS-485 接口在总线上只允许连接多达 128 个收发器，即具有多站能力，这样用

户可以利用单一的 RS-485 接口方便地建立设备网络。

阅读材料，了解如何计算 RS485 终端电阻阻值，以及通信故障的原因与排除办法。

1.5.4.3 RS422

RS-422 标准全称是“平衡电压数字接口电路的电气特性”，它定义了接口电路的特性。实际上还有一根信号地线，共 5 根线。由于接收器采用高输入阻抗和发送驱动器相较于 RS232 具有更强的驱动能力，可以实现相同传输线上连接多个接收节点。



图 1-37 RS422 信号接口

RS-422 和 RS-485 电路原理基本相同，都是以差动方式发送和接收数据。差分工作使同速率条件下传输距离远的根本原因，这正是二者与 RS232 的根本区别，因为 RS232 是单端输入输出，双工工作时至少需要数字地线、发送信号线和接收信号线三条线（异步传输），还可以加其它控制线完成同步等功能。

RS422 与 RS485 的主要区别在于：RS-422 通过两对双绞线可以全双工工作收发互不影响，而 RS485 只能半双工工作，发收不能同时进行，但它只需要一对双绞线。

由于通信的需要，运动控制器具备如上的串行通信接口，使控制器与外界进行信息互联提供硬件支持。Trio 的运动控制器中，串行通信接口的定义如下。

Pin	Function	Note
1	RS485 Data In A Rx+	Serial Port #2
2	RS485 Data In B Rx-	
3	RS232 Transmit	Serial Port #1
4	Serial 0V	
5	RS232 Receive	
6	5V OUT	Serial Port #2
7	RS485 Data Out Z Tx-	
8	RS485 Data Out Y Tx+	

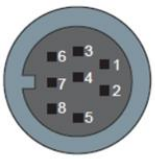


图 1-38 Trio 控制器串行通信接口定义

1.5.4.4 CAN

CAN[Control（Controller） Area Network]是控制(器)局域网的简称。CAN 是一种

有效支持分布式控制或实时控制的串行通信网络。CAN 接口以其高可靠性被广泛用户汽车本体的局域网通信，在工业应用中也获得了广泛应用。Trio 运动控制器上具备有该物理接口，主要用户系统的输入、输出接口扩展。

CAN 总线规范已被国际标准化组织 ISO 制订为国际标准 ISO11898，工业应用中，以 CAN_H 和 CAN_L 的两个信号线进行差分传输。总线上的数据采用不归零编码方式（NRZ），可具有两种互补的逻辑值之一：显性及隐性。隐性位表示 1，显性为表示 0。同一 CAN 网络各节点使用相同的位速率。它的每位时间由同步段、传播段、相位缓冲段 1 及相位缓冲段 2 组成。发送器在同步段前改变输出的位数值，接收器在两个相位缓冲段间采样输入位值，而两个相位缓冲段长度可自由调节，以保证采样的可靠性。另外，CAN 总线采用时钟同步技术来保证通讯的同步。

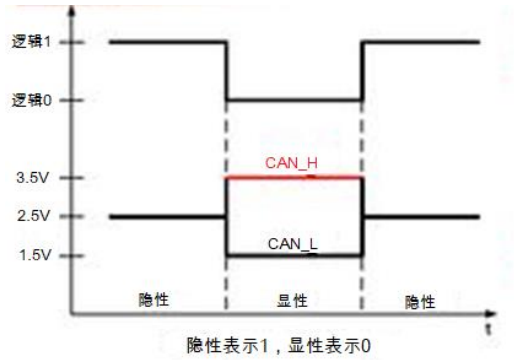


图 1-39 CAN 的物理信号

CAN 总线的速度可以达到 1M 波特率。其传输距离与传输速度相关。通常情况下，不同的传输速度和传输距离如下图所示。

Bit rate [kbit/s]	Bus length [m]	Nominal bit time [µs]	Range for sample point (2) [%]	Max. stub length [m]	Accumulated stub length [m]
1000	25	1	75 to 90	1,5 m	7,5 m
800	50	1,25	75 to 90	2,5 m	12,5 m
500	100	2	85 to 90	5,5 m	27,5 m
250	250 (1)	4	85 to 90	11 m	55 m
125	500 (1)	8	85 to 90	22 m	110 m
50	1000 (1)	20	85 to 90	55 m	275 m
20	2500 (1)	50	85 to 90	137,5 m	687,5 m
10	5000 (1)	100	85 to 90	275 m	1375 m
(1) The total device internal delay is considered as 210 ns @ 1 Mbit/s and 800 kbit/s; 310 ns (includes 2 x 40-ns optocoupler) @ 500 and 250 kbit/s; 450 ns (2 x 100-ns optocoupler) @ 125 kbit/s; 1,5 time quanta @ ≤ 50 kbit/s					
(2) It is highly recommended to place the sample point as close as possible to 87,5 % of the bit time					

图 1-40 CAN 的传输速度和距离的关系以及采样点的设置推荐

在工业应用中，CAN 总线的网络拓扑以总线型为主。其他类型的拓扑可以通过 CAN 中继器、集线器来拓展实现。对于总线型拓扑的 CAN 网络，在布置网络时，需要考虑：

（1）注意通信速率与通信距离的关系，保证通信的稳定；

（2）线缆布置上，控制分支的长度；

（3）终端电阻的配置，实现阻抗匹配，降低信号反射对通信质量的影响；

（4）建议采用带屏蔽的双绞线作为通信线缆，屏蔽层单点接地处理；

（5）总线上连接的设备设置为相同的通信速度和不同的 ID。

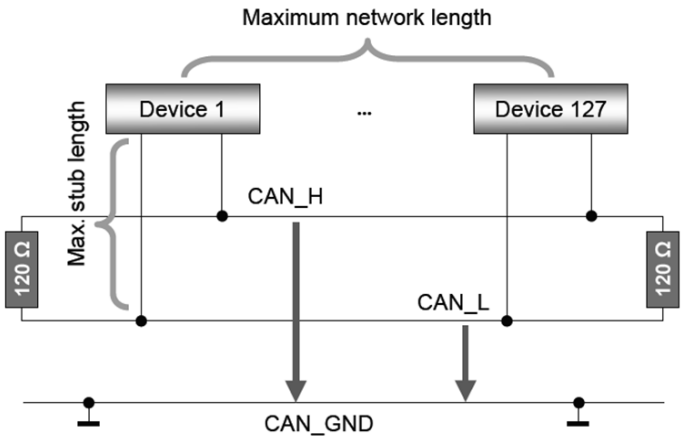


图 1-41 CAN 总线型网络拓扑

上图以基于 CAN 总线 CANopen 协议的单段网络（最多支持 127 个设备节点）布线为例，展现了总线型网络的布置。

CAN 支持四类信息帧类型。

（1）数据帧。 CAN 协议有两种数据帧类型标准 2.0A 和标准 2.0B。两个标准的主要差别是帧标识 ID 的长度不同。在 2.0A 类型中，ID 的长度为 11 位；在 2.0B 类型中 ID 为 29 位。CAN 数据帧由帧起始、仲裁域、控制域、数据区（最大可传输 8 字节的数据）校验域、应答域、帧结束。

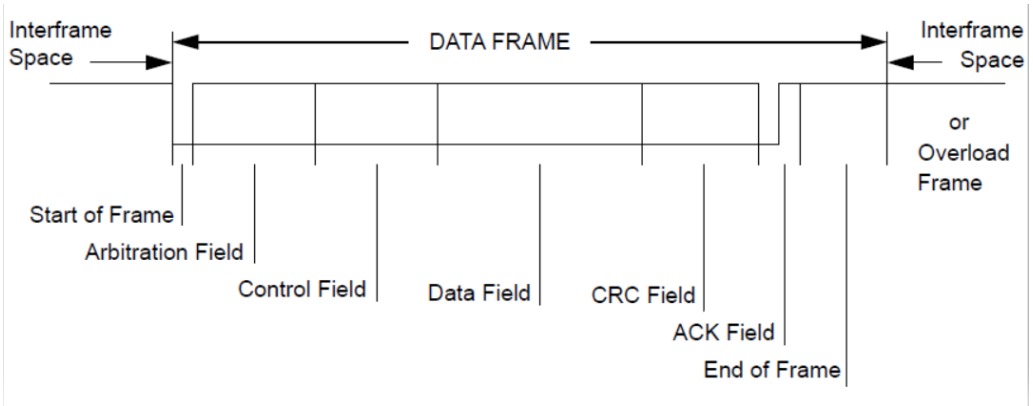


图 1-42 CAN 数据帧结构

（2）远程帧。 接收数据的 CAN 节点可通过发远程帧请求源节点发送数据。它由 6 个域组成：帧起始、仲裁域、控制域、校验域、应答域、帧结束。

（3）错误指示帧 由错误标志和错误分界两个域组成。接收节点发现总线上的

报文有误时，将自动发出“活动错误标志”，其他节点检测到活动错误标志后发送“错误认可标志”。

（4）超载帧。 由超载标志和超载分隔符组成。超载帧只能在一个帧结束后开始。当接收方接收下一帧之前，需要过多的时间处理当前的数据，或在帧问空隙域检测到显性电平时，则导致发送超载帧。

（5）帧间空隙。位于数据帧和远地帧与前面的信息帧之间，由帧间空隙和总线空闲状态组成。帧间空隙是必要的，在此期间， CAN 不进行新的帧发送，目的是使 CAN 控制器在下次信息传递前有时间进行内部处理操作。当总线空闲时，CAN 控制器方可发送数据。

CAN 总线由于具有优秀的检错机制，广泛应用于汽车内部的通信。而后被拓展应用到工业控制领域，并获得了广泛的应用。其主要的特点有：

（1） CAN 总线以报文为单位进行信息交换，报文中含有标示符（ID），它既描述了数据的含义又表明了报文的优先权。因此，对于实时性高、重要的数据，可以通过设置较小的 ID 来获取高的传输优先级。

（2）CAN 总线上的各个节点都可主动发送数据。当同时有两个或两个以上的节点发送报文时，CAN 控制器采用 ID 进行仲裁，较小 ID 的节点将抢得总线使用权。ID 控制节点对总线的访问。发送具有最高优先权报文的节点获得总线的使用权，其他节点自动停止发送，总线空闲后，这些节点将自动重发报文。因此，CAN 网络可以构成多主网络，被应用于分布式系统中。

（3）必须保证网络中的各个节点的通信波特率设置相同。不同的波特率节点存在后，网络将不能进行通信。

（4）节点的发送数据若未成功发送，将等待总线空闲，继续进行重发操作。

在运动控制系统中，CAN 接口被广泛应用到控制器、电机驱动器、远程 IO 设备。其接口标准参见 CiA303 标准定义。Trio 运动控制系统的产品遵守开放端子接口的标准，如图 1-43 所示。

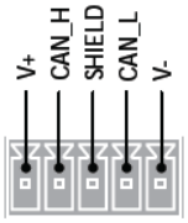


图 1-43 CAN 总线接口信号定义

ESTUN & Trio

随着控制信息传输的数量增大，普通 CAN 总线的传输能力已经不能满足一些应用，CAN-FD 通过提升和增加帧数据区域的长度来解决部分应用需求。具备 CAN-FD 控制器和接口器件不断涌现，在汽车领域已经开始广泛应用。在工控领域的应用还相对较少。

阅读 CAN 总线协议 2.0， MODBUS 协议，CANopen 协议，CiA402， EtherCAT CoE 协议相关的技术资料。

1.5.4.5 Ethernet

以太网(Ethernet)是当前应用最广泛、最普遍的局域网通信协议标准。在 IEEE802.3 对以太网的标准协议做出了具体的定义。以太网因适用各种网络协议、传输效率高、能耗低等优势，在工业控制和数字信息系统中被广泛使用。

在工业应用中，使用以太网接口作为设备对更上层的设备提供通信接口是当前普遍的做法。尤其是在工业计算机（IPC）中，具备以太网接口几乎是必须的配置。在工业中以太网以接口以 RJ45 为主，部分接口采用航空插头实现更高的连接可靠性和防护等级。

以太网接口在工业设备上，通常作用有：

1）通过以太网接口向外部传输设备的信息（如监控信息）。

2）与外部设备进行信息交互，实现设备间的信息交互。

3）设备使用的接口，如实现对设备的开发、在设备内编程、调试等。

4）提供设备的维护接口，如程序的升级、设备的自检等功能。

1.5.4.6 TCP

传输控制协议（TCP，Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议。简单来说，TCP 每发出一个数据包都要求确认，如果有一个数据包丢失，就收不到确认，发送方就必须重发这个数据包。为了保证传输的可靠性，TCP 协议建立了三次对话的确认机制，即在正式收发数据前，必须和对方建立可靠的连接。TCP 数据包和 UDP 一样，都是由首部和数据两部分组成，唯一不同的是，TCP 数据包没有长度限制，理论上可以无限长，但是为了保证网络的效率，通常 TCP 数据包的长度不会超过 IP 数据包的长度，以确保单个 TCP 数据包不必再分割。

在使用以太网接口进行 TCP/IP 通信时，需要做如下的一些设置：

（1）设置设备的 TCP/IP 属性。对于使用设定 IP 地址的情况，需要使各个网络 IP 在同一个网段，且地址不冲突。例如：PC 端的 IP 地址设置为 192.168.0.100，子网掩码设置为 255.255.255.0，网关为空，或设置为 192.168.0.1。在控制器的 IP 地址设置为 192.169.0.102。

ESTUN & PR

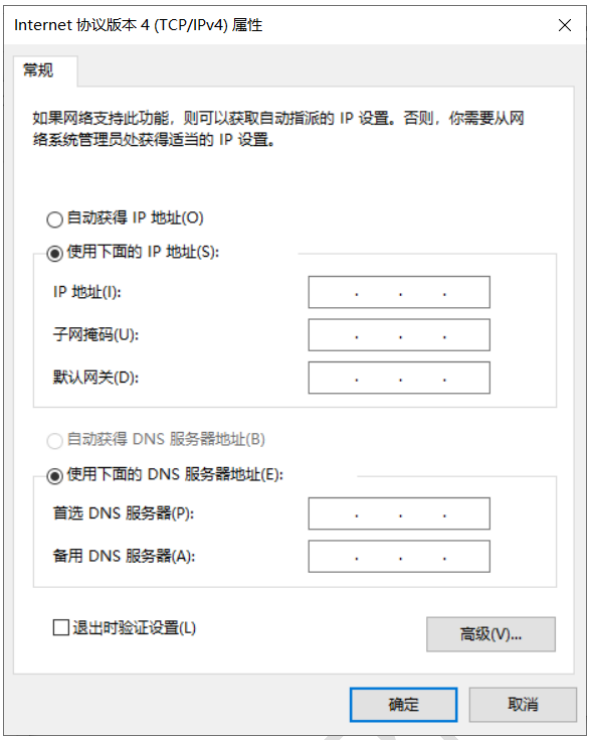


图 1-44 TCP/IP 属性配置页面

（2）设置连接设备（如计算机）的 TCP/IP 属性。

完成两个设备的配置后，即可在 PC 端通过命令窗口中使用 PING 指令访问中特定的 IP 地址，判断是否可以建立连接。

1.5.4.7 UDP

用户数据报协议（UDP，User Datagram Protocol）较 TCP 协议之前出现，UDP 为应用程序提供了一种无需建立连接就可以发送封装的 IP 数据包的方法。UDP 协议定义了端口，同一个主机上的每个应用程序都需要指定唯一的端口号，并且规定网络中传输的数据包必须加上端口信息，当数据包到达主机以后，就可以根据端口号找到对应的应用程序了。UDP 协议基本上是 IP 协议与上层协议的接口。UDP 协议适用端口分别运行在同一台设备上的多个应用程序。

UDP 协议比较简单，实现容易，但它没有确认机制，数据包一旦发出，无法知道对方是否收到，因此可靠性较差。为了解决这个问题，提高网络通信的可靠性，在 UDP 协议的基础上增加了建立连接和握手机制，TCP 协议诞生了。

UDP 用来支持那些需要在计算机之间传输数据的网络应用，常被用于对传输性能要求高，而对传输内容的完整性不是特别重要的场合，如多媒体数据流，不会因为数据的损坏而对已传输的数据进行重发、纠正。由于 UDP 是一种无需建立连接的通信，报文没有可靠性保证、顺序保证和流量控制字段等。但 UDP 协议的控制选项较少，在数据传输过程中延迟小、数据传输效率高，适合对可靠性要求不高的应用程序，或者可以保障可靠性的应用程序，如 DNS、TFTP、SNMP 等。

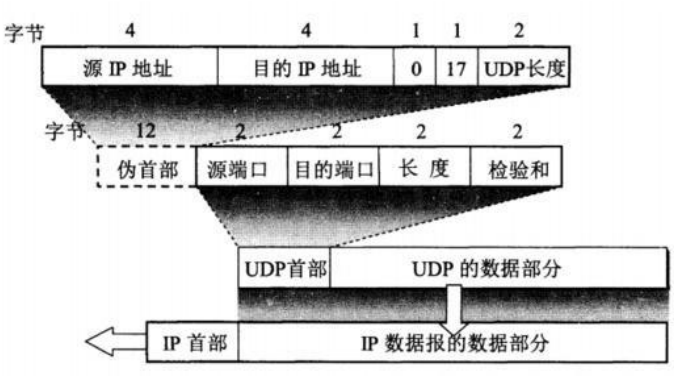


图 1-45 UDP 报文结构

在 UDP 协议层次模型中，UDP 位于 IP 层之上。应用程序访问 UDP 层然后使用 IP 层传送数据报。IP 数据包的数据部分即为 UDP 数据报。IP 层的报头指明了源主机和目的主机地址，而 UDP 层的报头指明了主机上的源端口和目的端口。UDP 传输的段（segment）有 8 个字节的报头和有效载荷字段构成。

UDP 报头由 4 个域组成，其中每个域各占用 2 个字节，具体包括源端口号、目标端口号、数据报长度、校验值。

端口号

UDP 协议使用端口号为不同的应用保留其各自的数据传输通道。UDP 和 TCP 协议正是采用这一机制实现对同一时刻内多项应用同时发送和接收数据的支持。数据发送一方（可以是客户端或服务器端）将 UDP 数据包通过源端口发送出去，而数据接收一方则通过目标端口接收数据。有的网络应用只能使用预先为其预留或注册的静态端口；而另外一些网络应用则可以使用未被注册的动态端口。因为 UDP 报头使用两个字节存放端口号，所以端口号的有效范围是从 0 到 65535。一般来说，大于 49151 的端口号都代表动态端口。UDP 端口号指定有两种方式：由管理机构指定端口和动态绑定的方式。

长度

数据报的长度是指包括报头和数据部分在内的总字节数。因为报头的长度是固定的，所以该域主要被用来计算可变长度的数据部分（又称为数据负载）。数据报的最大长度根据操作环境的不同而各异。从理论上说，包含报头在内的数据报的最大长度为 65535 字节。不过，一些实际应用往往会限制数据报的大小，有时会降低到 8192 字节。

校验值

UDP 协议使用报头中的校验值来保证数据的安全。校验值首先在数据发送方通过特殊的算法计算得出，在传递到接收方之后，还需要再重新计算。如果某个数据报在传输过程中被第三方篡改或者由于线路噪音等原因受到损坏，发送和接收方的校验计算值将不会相符，由此 UDP 协议可以检测是否出错。这与 TCP 协议是不同的，后者要求必须具有校验值。

TCP 与 UDP 的区别

UDP 和 TCP 协议的主要区别是两者在信息传输过程中是否有信息传输机制可靠性保证措施。TCP 协议是面向连接的传输控制协议，包含了专门的传递保证机制，当数据接收方收到发送方传来的信息时，会自动向发送方发出确认消息；发送方只有在接收到该确认消息之后才继续传送其它信息，否则将一直等待直到收到确认信息为止。因此，TCP 具有高可靠性，确保传输数据的正确性，不出现丢失或乱序

与 TCP 不同，UDP 协议提供了无连接的数据报服务，并不提供数据传送的保证机制。如果在从发送方到接收方的传递过程中出现数据包的丢失，协议本身并不能做出任何检测或提示。UDP 在传输数据前不建立连接，不对数据报进行检查与修改，无须等待对方的应答，所以会出现分组丢失、重复、乱序，应用程序需要负责传输可靠性方面的所有工作。因此，通常人们把 UDP 协议称为不可靠的传输协议。

UDP 具有较好的实时性，工作效率较 TCP 协议高；UDP 段结构比 TCP 的段结构简单，因此网络开销也小。

TCP 协议可以保证接收端毫无差错地接收到发送端发出的字节流，为应用程序提供可靠的通信服务。对可靠性要求高的通信系统往往使用 TCP 传输数据。

以太网技术的成熟后，工业以太网技术得以快速发展，逐渐成为了当前工业通信中的主流，常见的工业以太网协议有 EtherCAT，Profinet, EtherNet/IP, PowerLink 等等。工业以太网技术得以快速发展的主要原因与其特点有关：

（1）工业以太网具有广泛的应用范围。以太网技术作为计算机网络技术的重要基础，可以有效利用已有的计算机网络开发资源，更便于应用开发。

（2）工业以太网具有较高的通信速率。工业以太网的 100Mbps、1000Mbps 的传输技术都已十分成熟。其数据的吞吐效率可以满足当前工业应用的数据负载和数据实时性方面的要求。

（3）工业以太网技术具有良好的应用经济性。工业以太网技术的高速发展和技术的成熟，在以太网应用的的设计、应用方面已经十分成熟。在应用开发中，具有相当丰富的资源和设计案例，可以有效降低开发难度和开发成本，也便于推广。

（4）可以利用以太网的技术实现数据的共享和兼容性。

1.5.5 现场总线协议

上述几种接口是工业自动化应用的主要通信接口。为了实现信息的传输，基于这些物理层的传输接口，相应的通信协议基于这些通信接口逐步发展，以满足特定的应用需求。在工业自动化控制系统中，被广泛使用的工业现场总线协议有 Modbus (RTU 和 ASCII 两种)，CANopen， EtherCAT，以及一些自动化设备厂商推出的协议，如 Siemens 的 Profibus, Profinet, 贝加莱推出的 Powerlink，安川的 Mechatrolink，三菱机电推出的 CC-link 总线，固高科技推

出 g-link 总线等。这些总线随着行业、技术的发展而改进、丰富和完善功能。

在运动控制系统中，采用通信方式实现伺服电机控制的常见通信协议有 CANopen、EtherCAT、Modbus、Profibus。使用通信的方式来对电机控制，是目前一些复杂、大系统应用场景首选的控制方式。采用通信方式，系统的大小、电机轴的多少都易于裁剪，没有复杂的控制接线。搭建的系统具有极高的灵活性。

接下来，简单介绍被广泛使用的工业现场总线协议 Modbus (RTU 和 ASCII 两种)，CANopen， EtherCAT 的基本概念的特点。

1.5.5.1 Modbus RTU/ASCII

Modbus 是一种串行通信协议，已经成为工业领域通信协议的业界标准。被广泛应用在工业控制领域，实现设备间的通信。Modbus 协议采用主从方式，在同一段 Modbus 网络中，通常只有一个主节点（Master），其它都为从节点（Slave）。每一个 Slave 节点都有一个唯一的地址。只有主节点才可以启动一个命令/发起通信，从节点对主节点的信息作出响应或无响应。

Modbus 信息以帧的方式进行传输，每帧有确定的起始点和结束点，是接收设备能够识别帧的起始和结束，在帧的起点开始接收帧信息，帧的结束信息完成整帧的信息接收。Modbus 协议有 RTU 和 ASCII 模式，其数据格式及帧的结构如下。

1） ASCII

ASCII 的每个字节的格式：

编码系统：采用 16 进制，ASCII 字符 0~9，A~F

数据位： 1bit 起始位

7bit 数据位，低位先传

奇/偶校验 1bit：无奇偶校验位时 0 bit

停止位： 无校验时 2bit，有校验时 1bit

校验区：采用 LRC 校验 8bit

ASCII 模式采用 LRC(Longitudinal Redunancy Check)帧误值。LRC 帧误值是从设备地址开始到数据区结束进行计算所得到的（累加后取低字节值的补码）。数据帧中的信息以字符形式进行传输，即每 8bit 的字节按照其 16 进制的数值传输两个字符。

ASCII数据帧格式							
起始标志	设备地址	功能码	数据区	校验区		结束标志	
':' (0x3A)	1字节	1字节	数据1， 数据2，… 数据N	LRC高字节	LRC低字节	'CR'（0x0D）	'LF'（0x0A）

图 1-46 ASCII 模式帧结构

示例：向从设备地址 0x01 的伺服驱动器 0x0200 地址写入数值 100。

指令信息	
起始标志	':'
设备地址	'0'
	'1'
功能码	'0'
	'6'
写入地址	'0'
	'2'
	'0'
	'0'
写入内容	'0'
	'0'
	'6'
	'4'
LRC校验	'9'
	'3'
结束符	CR
	LF

响应信息	
起始标志	':'
设备地址	'0'
	'1'
功能码	'0'
	'6'
写入地址	'0'
	'2'
	'0'
	'0'
写入地址的内容	'0'
	'0'
	'6'
	'4'
LRC校验	'9'
	'3'
结束符	CR
	LF

图 1-47 ASCII 数据帧通信示例

ASCII 采用字符进行传输，其传输效率较低，但其起始标志和结束符的存在，可以很容易判断数据帧的开始与结束。且字符间的传输间隔没有严格要求（如长达 1 秒）。

2） RTU

RTU 模式中每个字节的格式：

编码系统：8bit 二进制，十六进制 0~9，A~F

数据位：起始位： 1bit

数据位： 8bit

校验位： 有校验时 1 bit，无校验时 0 bit

停止位： 有校验时 1bit， 无校验时 2bit

校验区：采用循环冗余校验 CRC。

RTU 模式采用 CRC(Cyclical Redundancy Check)帧误值实现对数据帧传输的正确性进行校验。

RTU数据帧格式						
起始位	设备地址	功能码	数据区	CRC校验		结束符
>=3.5bit	8bit	8bit	N*8bit	8bit低字节	8bit高字节	>=3.5bit

图 1-48 RTU 模式的数据帧结构

示例：向从设备地址 0x01 的伺服驱动器 0x0200 地址写入数值 100。

ESTUN & Trio

指令信息	
设备地址	01H
功能码	06H
起始地址	02H
	00H
写入内容	00H
	64H
CRC校验低字节	89H
CRC校验高字节	99H

响应信息	
设备地址	01H
功能码	06H
起始地址	02H
	00H
写入内容	00H
	64H
CRC校验低字节	89H
CRC校验高字节	99H

图 1-49 RTU 数据帧通信示例

RTU 模式中每 8bit 字节分成 2 个 4 位 16 进制的字符，可以实现在相同通信速率（波特率）下，较 ASCII 模式有更为高效的数据传输。但这种模式，数据帧的开始和结束时通过总线的空闲时间来判断的。帧间隔需要大于等于 3.5 个位传输时间，因此时间的检测/控制会对设备处理数据传输的实时性有要求。

使用 RTU 模式和 ASCII 模式读取从设备指定地址的数据帧的示例。从设备地址为 0x01 的伺服驱动器 0x0101 的地址读 2 个字（WORD）的内容。从设备地址、读取起始地址、读取数量计算所得的 LRC=0Xf8，CRC=0x3794。两种模式的数据帧如图 1-50：

ASCII模式读取	
起始标志	':'
设备地址	'0'
	'1'
功能码	'0'
	'3'
起始地址	'0'
	'1'
	'0'
	'1'
读取个数（以WORD为单位）	'0'
	'0'
	'0'
	'2'
LRC校验	'F'
	'8'
结束符	CR
	LF

RTU模式读取	
设备地址	01H
功能码	03H
起始地址	01H
	01H
读取个数	00H
	02H
CRC校验低字节	89H
CRC校验高字节	99H

ESTUN & Trio

图 1-50 两种模式的读取指令

Modbus 数据帧中，地址的范围时 0~255，地址 0 用于广播帧，所有从站接收，且无需应答。

功能码代表帧的操作目的。如下内容以 RTU 模式说明几种常用功能码的意义与使用。

01 读线圈

如：要检测变频器运行方向（假如变频器反转运行）（CD160=CD161=1，CD162=3）

应发送数据： 01 01 00 05 00 01 CRC（16 进制），

变频器返回数据： 01 01 01 01 CRC（16 进制）

返回的数据位为“01”（16 进制）化为 2 进制数为“0000 0001”末位表示运行状

态为“1”表示变频器正在反转运行（如是正转或停止状态则为“0”详见线圈说明）

03 读保持寄存器

如：要读变频器功能码 CD000 中的设定值（假如为 30.00HZ）（CD160=CD161=1 CD162=3）

应发送数据： 03 00 00 00 01 CRC （16 进制）

变频器返回数据： 01 03 02 0B B8 CRC（16 进制）

返回的数据位为“0BB8”（16 进制）转换为 10 进制数为 3000 表示 CD000 设置值为 30.00

04 读取输入寄存器

如：要读变频器温度显示值（假如温度为 36.2）（CD160=CD161=1 CD162=3）

应发送数据： 01 04 00 06 00 01 CRC（16 进制）

变频器返回数据： 01 04 02 01 6A CRC（16 进制）

返回的数据位为“016A”（16 进制）转换为 10 进制为“362”表示温度为 36.2

05 写单个线圈状态

如：要使变频器的正转无效，反转有效（CD160=CD161=1 CD162=3）

应发送数据： 01 05 00 49 00 00 FF 00 CRC（16 进制）

变频器返回数据：01 05 00 49 00 00 FF 00 CRC（16 进制）

“0000”代表无效; “FF00”代表有效

06 写单个保持寄存器值

如：要对变频器功能码 CD000 写入 30.00（CD160=CD161=1 CD162=3），

应发送数据： 01 06 00 00 0B B8 36 0D 0A（16 进制）

变频器返回数据： 01 06 00 00 0B B8 36 0D 0A（16 进制）

返回的数据位为“0BB8”（16 进制）转换为 10 进制为“3000”表示 CD000 设置值为 30.00

0F 写多个线圈状态

如：要使变频器的正转无效（CD160=CD161=1 CD162=3）

应发送数据： 01 0F 00 48 00 08 01 00 02 CRC（16 进制）

变频器返回数据：01 0F 00 48 00 08 CRC（16 进制）

“0002”即 0000 0010; 即线圈 0x49 置为 1，给变频器发正转命令。

10 写多个保持寄存器

ESTUN & Trio

如要同时对 CD000 写入 30.00，对 CD001 写入 110.0，（CD160=CD161=1

CD162=3）

应发送数据： 01 10 00 00 00 02 04 0B B8 04 4C CRC（16 进制）

变频器返回数据： 01 10 00 00 00 02 45CRC（16 进制）

发送的要写入 16 进制数据“0BB8”和“04 4C”转换为 10 进制数分别为 3000 和

1100 表示对 CD000 和 CD001 写入的数据分别为 30.00 和 110.0

在 Modbus 通信过程中，从站接收到异常的指令信息帧时，会向主站发送带有异常信息的数据帧。以 RTU 模式为例，从站发现自身地址的异常信息帧时，会将向主站响应对应的错误帧，错误帧的设备地址为从设备的地址，功能码将接收到的功能码+0x80，以表示这是一个异常，紧接着发送一个字节的异常代码。其异常帧的结构如表 1-1 所示。对于 ASCII 模式，其区别在传输的帧数据采用字符传输。

表 1-1 RTU 模式异常帧格式及距举例

报文	内容含义	备注（举例）
1 字节	设备地址	01H
1 字节	功能码	85H
1 字节	异常代码	02H
1 字节	CRC 高字节	C3H
1 字节	CRC 低字节	51H

异常代码反映错误的原因。常见的异常代码及其含义如表 1-2 所示。

表 1-2 Modbus 异常代码及含义

异常代码	含义
1	非法功能码。子站无法处理主站报文中的功能码。
2	非法数据地址。子站无法处理主站报文中的数据地址。
3	非法数据内容。主站报文中的数据不能满足子站的要求格式。
4	子站设备错误。
5	确认帧。
6	子站忙。

Modbus 协议被广泛应用于工业自动化设备间的通信，构建主从（客户端和服务器）设备间的通信网络。其发展也会随应用的需求变化而增加和优化。本章节以介绍 Modbus 的基本功能和使用方法，需要深入了解 Modbus 协议，可进入其官网进行了解。

Mobus 官网：https://modbus.org/tech.php

1.5.5.2 CANopen

CANopen 是一种架构在控制局域网络（Controller Area Network, CAN）上的高层通讯协议，包括通讯子协议和设备子协议[14]。设备子协议已经广泛应用于工业控制场合，如用于 I/O 模组的 CiA401 协议、用于运动控制的 CiA402 协议等。CANopen 协议是工业控制中常用的一种现场总线协议。它不仅被应用到基于 CAN 的物理层通讯的应用层协议，还被应用到工业以太网总线协议中，如 EtherCAT 和 Powerlink。

CiA301 是 CANopen 的通信协议栈，设备子协议（如 CiA401，CiA402）是建立在该通信协议栈的上层。通信协议栈处理连接到 CAN 网络上的其他模块（CANopen 设备）所需的通信规约。CANopen 协议的数据是基于 CAN 总线的数据帧构建的，其帧结构定义如图 1-51 所示。

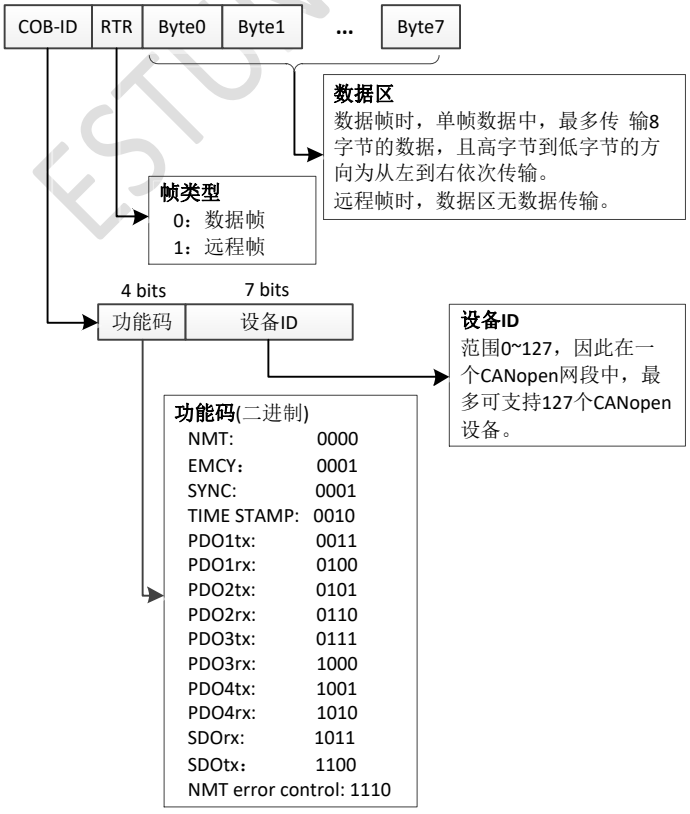


图 1-51 CANopen 数据帧结构

CANopen 设备需要具备设备的网络状态机（NMT）管理，网络状态机包括四种：初始状态(Initialisation)、预操作状态(Pre-operational)、操作状态(Operational)、停止状态(Stopped)。

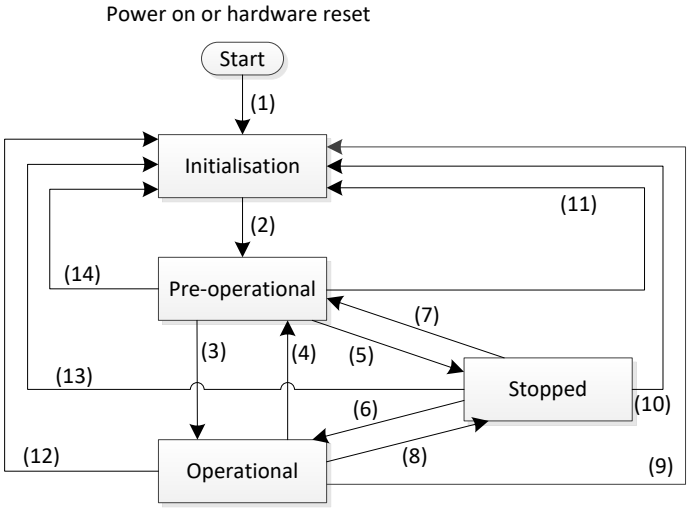


图 1-52 CANopen 网络状态机

CANopen 网络状态机的转换，需要遵循图 1-52 的状态机转换规则。中的各个状态转化路径如下表 1-3 中定义

表 1-3 NMT 状态机切换说明

状态切换路径代号	说明
(1)	设备上电或硬件复位时，自动进入初始化状态(Initialisation)。
(2)	完成 NMT 初始化状态(Initialisation)的处理工作后，自动地进入预操作状态(Pre-operational)。
(3)	通过发送启动远程节点指令进入。
(4)、(7)	通过发送进入预操作状态(Pre-operational)指令进入。
(5)、(8)	通过发送停止远程节点指令进入。
(6)	通过发送启动远程节点指令进入。
(9)、(10)、(11)	通过发送复位节点指令进入。
(12)、(13)、(14)	通过发送复位通信指令进入。

CANopen 网络中，只有主设备才能进行网络状态机管理，发送网络状态管理命令。网络状态命令数据帧格式如图 1-53：

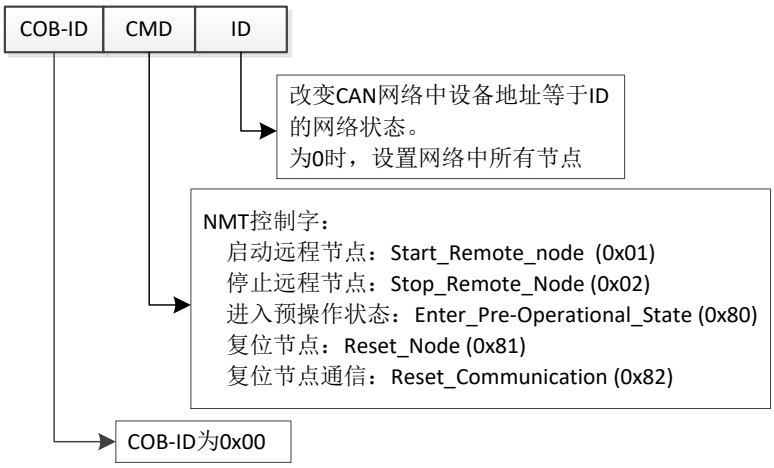


图 1-53 网络状态机管理命令格式

设备初始化结束后，将自动进入预操作状态（Pre-Operational），并发送 Boot-up 消息。网络中的主站（或管理机）在任何时候都可以通过 NMT 服务都可使所有或者部分节点进入不同的工作状态。NMT 服务的 CAN 报文由 CAN 头(COB-ID)和两字节数据组成；第一个字节表示请求的服务类型('NMT command specifier')，第二个字节是节点 ID，或者 0（此时寻址所有节点）。

仅支持最小化 boot-up 的设备叫最小能力设备。最小能力设备在设备初始化结束后自动进入 Pre-Operational 状态。在这个状态，网络中的主设备可以通过 SDO 进行参数配置和 COB-ID 分配。

当设备进入 Stopped 停止状态后，除了 NMT 服务和节点保护服务（如果支持并且激活的话）外，节点设备将停止通讯。

在不同的网络状态下，所支持的通讯对象是不同的，见表 1-4 所示。

表 1-4 不同网络状态支持的通讯对象

网络状态	支持的通讯对象
初始化状态(Initialization)	Boot-up
预操作状态(Pre-operational)	NMT, Node Guard, SDO, Emergency
操作状态(Operational)	NMT, Node Guard, SDO, Emergency, PDO
停止状态(Stopped)	NMT, Node Guard

对象字典 (Object Dictionary) 是 CANopen 协议的一个核心内容，它是由一系列变量构成的对象组，这些对象组提供设备的信息数据、参数、功能，使设备能够通过总线操作的控制数据和状态信息等。这些构成对象组的变量通过一个 16 位索引 (Index)和一个 8 位子索引(Subindex)构成的访问地址组织起来[14]。索引的范围在 0x1000 到 0x9FFF 之间，子索引的范围为为 0 到 255 之间。CANopen 的对象字典使用电子数据文件 EDS（Electronic Data Sheet）来记录对象字典中的数据。EDS 也成为 CANopen 设备信息的交互电子资料。

CANopen 对象字典中的对象由一系列子协议来描述[14]。子协议为对象字典中的每个对

象的各个属性进行描述，包括它的功能、名字、索引、子索引、数据类型，以及这个对象可选性、读写属性等等，这样可保证不同厂商的同类型设备兼容。CANopen 协议的核心描述子协议是 CiA301，其包括了 CANopen 协议应用层及通信结构描述，其它的协议子协议都是对 CiA301 协议描述文本的补充与扩展。CiA 组织对不同行业都会组织起草一份 CANopen 设备子协议，如 CiA401，CiA402 等。

CANopen 设备间的通讯可分为三种通讯模型[14]：

在 Master/Slave 模型中，一个 CANopen 设备为 Master，负责传送或接收其他设备（称为 Slave）的数据。网络状态机管理 NMT 协定就使用了 Master/Slave 模型。

客户机/服务器（Client/Server）模型定义在 SDO 协定中，SDO Client 将对象字典的索引及子索引传送给 SDO Server，因此会产生一个或数个需求数据（对象字典中，索引及子索引对应的内容）的 SDO 数据包。

生产者/消费者（Producer/Consumer）模型用在 Heartbeat and Node Guarding 协定。由一个生产者送出数据给消息者，同一个生产者的数据可能给一个以上的消息者。又可分为二种： push-model：生产者会自动送出数据给消费者。 pull-model：消费者需送出请求讯息，生产者才会送出数据。

CANopen 协议被广泛应用于工控领域，CiA 组织针对不同工控领域，制定相应的设备子协议，CiA402 协议就是其中之一，是伺服运动控制的协议标准，属于 CANopen 协议的应用层协议规范，位于 CANopen 参考模型的最上层。CANopen 参考模型如图 1-54。

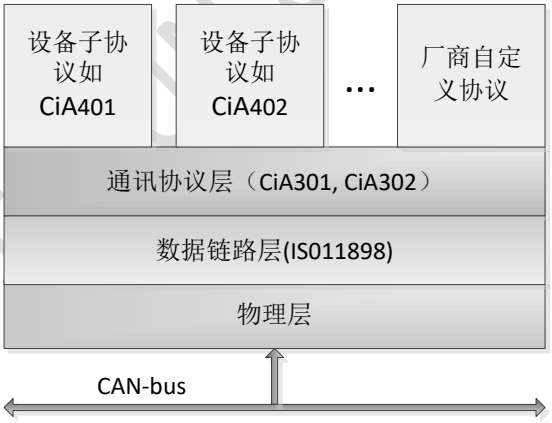


图 1-54 CANopen 参考模型

可以看出，CANopen 协议包含许多的子协议，其中通信协议层描述了 CANopen 的应用层及通信结构，是其他设备子协议和厂商自定义协议的基础。由此可见，CANopen 协议的子协议主要划分为三类[14]：

1. 通讯子协议（Communication Profile）：描述对象字典的主要形式、通讯对象以及参数。该子协议应用于所有 CANopen 设备中，其索引值范围为：0x1000~0x1FFF。

2. 厂商自定义子协议（Manufacturer-specific Profile）：CANopen 对象字典范围内，预留特定段给厂商根据其产品需要，自行定义对象子弹的对象，如添加设备子协议未定义的特殊

功能。因此这段对象字典内的对象，存在不统一。即不同厂商，相同的对象字典项的定义可能不同。用于厂商自定义子协议的索引值范围为 0x2000~0x5FFF。

3. 设备子协议(Device Profile)：这类子协议是为各种不同类型设备而制定的，用于定义对象字典中索引值范围为 0x6000~0x9FFF 的对象，例如 CiA401、CiA402、CiA406 等。因此，不同厂商间，这部分对象字典基本是相互兼容的。

设备子协议 CiA402 是针对伺服运动控制行业制定的，从目前的协议版本中，定义了伺服驱动器运动控制的多种工作模式：包括轨迹位置模式（Profile Position Mode）、位置插补模式（Interpolated Position Mode）、轨迹速度模式（Profile Velocity Mode）、速度模式（Velocity Mode）、轨迹扭矩模式（Profile Torque Mode）、回零模式（Homing Mode）。在该协议中，对各个不同工作模式下的参数（对应对象字典中的对象）做了定义。

为了实现对伺服设备的控制，CiA402 协议中对伺服设备中的控制状态机（Device Control State Machine）做了规定，规范了设备的状态机转换规则，控制伺服驱动器的启动与停止，执行各个运动控制模式下的相关指令，并对设备的错误作出响应。CANopen 要求支持协议的伺服驱动器都必须满足这种状态机制[16]。其状态机转换机制如图 1-54 所示：

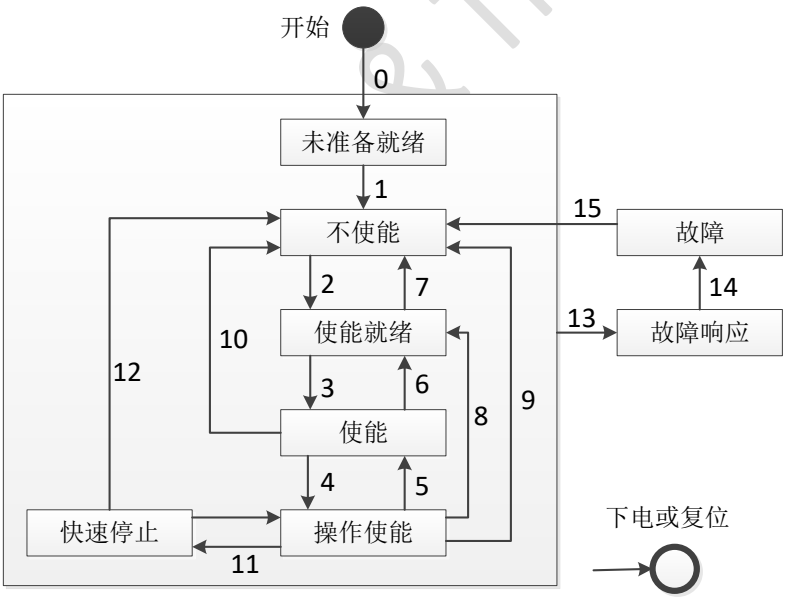


图 1-55 伺服驱动器状态机转换规则

为了获取伺服设备的状态机，可以从伺服设备中的对象字典中获取状态字 Statusword（索引为 0x6041，子索引为 0x00）的值来判断伺服当前所处状态。状态字 Statusword 的值与对应的状态机关系如表 1-5 所示。

表 1-5 Statusword 的值对应的状态机

Statusword 的值	状态机
xxxx xxxx x0xx 0000b	未准备就绪

Statusword 的值	状态机
xxxx xxxx x1xx 0000b	不使能
xxxx xxxx x01x 0001b	使能就绪
xxxx xxxx x01x 0011b	使能
xxxx xxxx x01x 0111b	操作使能
xxxx xxxx x00x 0111b	快速停止
xxxx xxxx x00x 1111b	故障响应
xxxx xxxx x0xx 1000b	故障

状态字 Statusword 中的 BIT10 表示 target reached，在不同工作模式时，用于判断提供的运行指令是否完成。

伺服驱动器状态机的各个状态切换是通过设置设备中的控制字 Controlword（索引为 0x6040，子索引为 0x00）来实现的。图 1-55 的各个状态切换箭头连线边的数字表示从一个转换箭头指向的状态的命令，控制字的转换命令定义如表 1-6 所示。

表 1-6 控制字 Controlword 状态转换命令对应的位设置

Controlword 的值	控制字的位					状态转换
	Bit7	Bit3	Bit2	Bit1	Bit0	
禁能	0	×	1	1	0	2,6,8
使能	0	0	1	1	1	3
使能操作	0	1	1	1	1	3+4（*）
禁用电压	0	×	×	0	×	7,9,10,12
快速停止	0	×	0	1	×	7,10,11
禁能操作	0	0	1	1	1	5
使能操作	0	1	1	1	1	4,16
故障复位	↑	×	×	×	×	15
*执行使能状态转换后，自动转换到使能操作状态。 *“×”表示与该位的设置值无关						

设备的状态字和控制字，在采用 CANopen 协议控制伺服设备运转时所必须掌握和运用的。此外还有不同的运行模式设置和运行模式状态，位置、速度、扭矩等命令和状态信息供访问。此外，还有不同工作模式下的参数设置和信息反馈对象字典供访问。CiA402 协议为了实现单设备多轴控制的需要，对对象字典索引范围 0x6000~0x9FFF 区域进行划分，具体如下：

索引范围 0x6000~0x67FF 用于轴 0；

索引范围 0x6800~0x6FFF 用于轴 1；

索引范围 0x7000~0x77FF 用于轴 2；

索引范围 0x7800~0x7FFF 用于轴 3；

索引范围 0x8000~0x87FF 用于轴 4；

索引范围 0x8800~0x8FFF 用于轴 5；

索引范围 0x9000~0x97FF 用于轴 6；

索引范围 0x9800~0x9FFF 用于轴 7；

因此，单个节点的 CANopen 伺服驱动可支持 8 个轴的应用需求。

CANopen 协议在运动控制应用中，主要涉及 CiA301, CiA401, DSP402 的内容。如何使用 SDO 和 PDO，及 PDO 的配置，需要更深入地了解协议内容和使用，请访问：https://www.can-cia.org/ 获取更多资源。

ESTUN & Trio

1.5.5.3 EtherCAT

2003 年，德国 Beckhoff 公司提出了一种开放、灵活的工业以太网技术 EtherCAT，在工业控制领域获得了广泛应用。近年来，EtherCAT 技术的发展迅速，具有 EtherCAT 协议的产品也不断涌现。尤其在运动控制领域、伺服系统、运动控制系统、远程 IO 模块、传感器等产品，支持 EtherCAT 几乎是行业的主流、趋势。

EtherCAT 使用标准以太网传输介质连接主从站构成系统，以介质访问解决信道竞争与分配，采用数据收发同时进行的全双工模式。EtherCAT 具备优异的实时性，因此被广泛应用与工控领域。主站设备利用标准以太网接口硬件发送 0x88A4 类型数据帧报文，报文依次经过网络中的每一个从站设备时，由硬件直接读取插入数据帧中属于自己的子报文。从站采用“On the fly”技术，利用硬件动态地完成报文的读写操作，以提高数据传输速率，数据在从站中的停留时间缩短至 200ns。

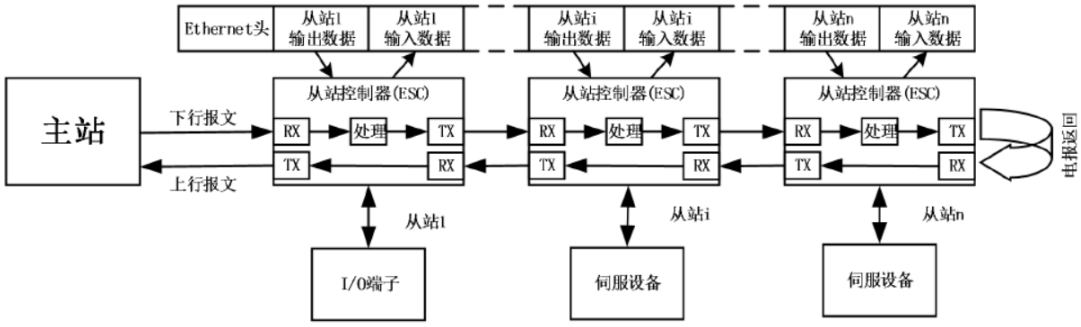


图 1-56 EtherCAT 报文传输过程

主站是网络通信的协调者与协议运行的管理者，是 EtherCAT 总线的重要组成部分。主站可以运行在运行在工业计算机平台，甚至普通 PC 上，也可以运行在基于嵌入式平台上。EtherCAT 主站采用常规的网络适配器作为接口与从站互联，现阶段网卡已经成为 PC 机的标准配件之一，因此以工业计算机为硬件运行平台的主站无需特殊的硬件（对同步要求高的场合，有些网口可能不能满足要求）。对于嵌入式主站平台系统而言，一般都设有通信控制单元 CCU。

EtherCAT 网络可以构成多种形式的拓扑结构，用户可根据应用需要灵活布置。

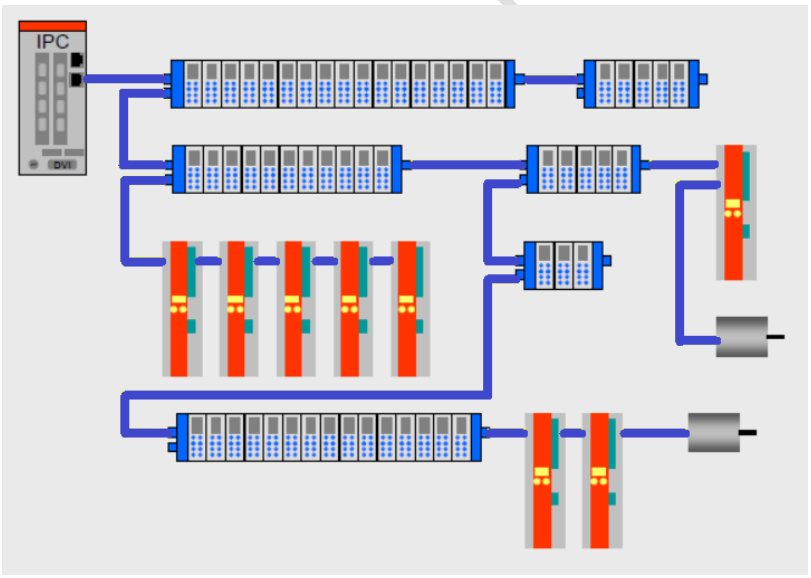


图 1-57 EtherCAT 的拓扑结构示例

EtherCAT 状态机（EtherCAT State Machine，简称 ESM）表征、转换应用层的五种工作状态，如图 1-58 所示，其中启动状态 Init、预工作状态 Pre-OP、安全工作状态 Safe-OP、工作状态 OP 这四种状态为 EtherCAT 设备必须支持的，另外一种可选状态是自举状态 Boot-Strap。EtherCAT 规定系统的某些操作只能在相应的工作状态进行，用以协调主从站间的通信关系，确保通信的顺畅与安全。

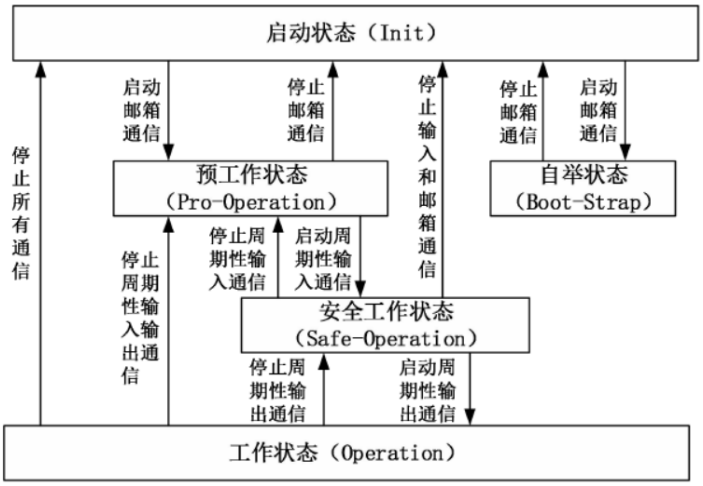


图 1-58 EtherCAT 网络状态机

从启动状态向工作状态转变时，必须遵守网络状态机的转化流程进行。需经历启动状态、预工作状态、安全工作状态、工作状态，逐级转化，反向状态转换时允许跨越中间状态进行转化，如图 1-58 所示。EtherCAT 主站发出新状态申请指令，从站检查自身运行状态执行指令，并改变从站的状态指示变量，完成状态转化。

表 1-7 EtherCAT 网络状态机说明

状态	说明
启动状态（Init）	设备初始化（如 ESC 寄存器初始化），无法使用过程数据通信及邮箱通信
预工作状态(Pre-Operational)	允许邮箱通信，禁止过程数据通信
安全工作状态(Safe-Operational)	可以读取过程数据对象的输入数据(TxPDO) 不能接收过程数据对象的输出数据(RxPDO)
工作状态(Operational)	通常工作状态 可以进行过程数据对象的输入/输出数据通信以及邮箱通信（可读写周期性/非周期性数据对象的读写操作）。
自举状态(Boot-Strap)	可使用邮箱通信，使用 FoE 更新固件数据

EtherCAT 应用层协议位是 EtherCAT 协议的顶层，是直接面向控制任务的一层，负责为上层应用函数读写网络数据提供协议接口，为控制程序程序提供服务。应用层不包括控制程序，它只是定义了控制程序与网络的交互接口，使符合此应用层协议的各种应用程序可以协同工作。目前 EtherCAT 应用层协议包括 CoE、SoE、EoE、FoE 等，通过这些协议 EtherCAT 技术实现对其他现场总线设备的兼容，方便设备的生产方及使用方的过渡，丰富可挂接到 EtherCAT 总线的设备种类。

在运动控制应用中，目前以使用 CoE 实现从站数据的输入与输出为主流。以下部分对 CoE 协议做简要介绍。CoE 是 EtherCAT 为兼容 CANopen 协议而制定的应用层协议。CANopen 是 CAN 总线的应用层协议，定义了基于 CAN 总线系统的高级通信协议和设备配

置规范。作为 CoE 协议核心概念的对象字典，本质上是一张包含系统各种参数、数据、描述的组，按照实时性不同可以分为 PDO（过程数据对象）和 SDO（服务数据对象）。前者的刷新频率高、优先级高，用于状态参数和指令数据的周期性传输；后者以邮箱通信方式读写寄存器配合参数，实现网络初始化操作。CoE 协议结构如图 1-59 所示。

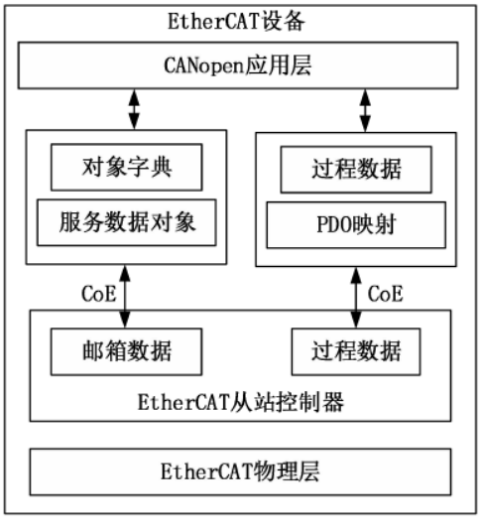


图 1-59 EtherCAT 应用层协议 CoE

在 CoE 的运动控制系统应用中，采用 SDO 的目的主要对象字典的参数配置，如 PDO 的映射设置，以及一些设备的参数设置。而 PDO 通常是进行周期性数据的传输，其周期通常与控制器的控制周期相同。因此，在运动控制系统中，对 EtherCAT 从设备的 PDO 映射配置，通常是需要做的工作，以满足控制器的控制需要。

1.5.5.4 MODBUS/TCP

随着以太网的高速发展，Modbus 也需适应于以太网的 TCP/IP 协议，在 1996 年，施耐德公司提出了基于 TCP/IP 协议的 Modbus 协议，该协议在 Modbus 协议的基础上，加入 TCP/IP 协议的消息头部。兼容了 TCP/IP 协议的 Modbus 入 TCP/IP 协议的消息头部。兼容了 TCP/IP 协议的 Modbus 协议能够在以太网中适用。在以太网日益发达的今天，这是对 Modbus 协议的极大进展与突破，使得它成为工业控制领域最广泛应用的协议。

Modbus 数据在 TCP/IP 以太网上传输，支持 Ethernet II 和 802.3 两种帧格式，Modbus TCP 数据帧包含报文头、功能代码和数据 3 部分，MBAP（Modbus Application Protocol ，Modbus 应用协议）报文头 MBAP 分 4 个域，共 7 个字节，如图 1-60 所示。ModbusTCP 的协议标识符为 0x00.

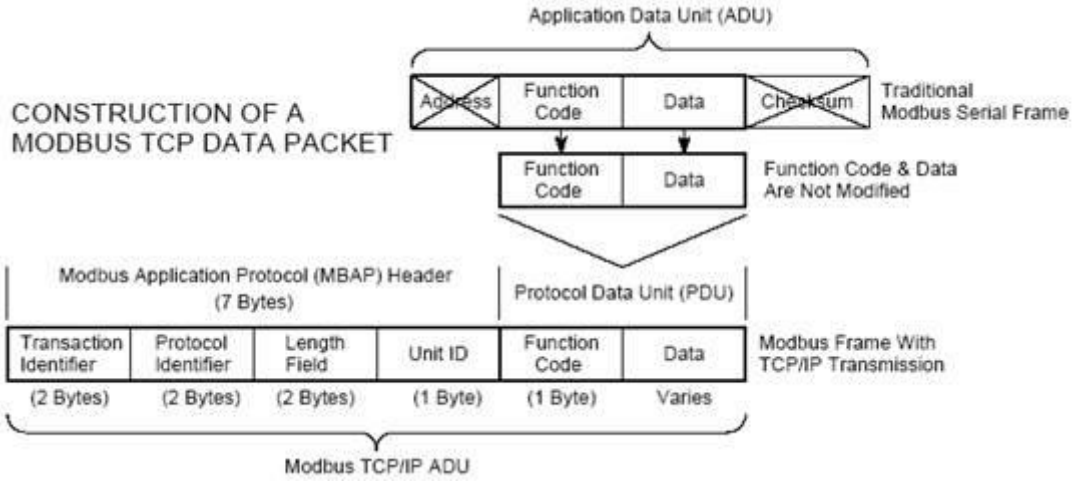


图 1-60 Modbus TCP 数据帧结构

Modbus TCP 使 Modbus RTU 协议运行于以太网，Modbus TCP 使用 TCP/IP 和以太网在站点间传送 Modbus 报文，Modbus TCP 结合了以太网物理网络和网络标准 TCP/IP 以及以 Modbus 作为应用协议标准的数据表示方法。Modbus TCP 通信报文被封装于以太网 TCP/IP 数据包中。由于 Modbus TCP 使用以太网 TCP/IP 数据链路层的校验机制而保证了数据的完整性，元 Modbus 报文中的校验数据 Checksum 被去除，原报文中的设备地址“Address”也被“Unit ID”替代。与传统的串口方式，Modbus TCP 插入一个标准的 Modbus 报文到 TCP 报文中，不再带有数据校验和地址。Modbus TCP 协议的最通用用途是为诸如 PLC，I/O 模块，以及连接其它简单域总线或 I/O 模块的网关服务的。

Modbus TCP 传输过程中使用了 TCP/IP 以太网参考模型的 5 层：

第一层：物理层，提供设备物理接口，与市售介质/网络适配器相兼容

第二层：数据链路层，格式化信号到源/目硬件址数据帧

第三层：网络层，实现带有 32 位 IP 址 IP 报文包

第四层：传输层，实现可靠性连接、传输、查错、重发、端口服务、传输调度

第五层：应用层，Modbus 协议报文

在 Modbus 服务器中按缺省协议使用 Port 502 通信端口（这是目前在仪表与自动化行业中唯一分配到的端口号），在 Modbus 客户器程序中设置任意通信端口，为避免与其他通讯协议的冲突一般建议 2000 开始可以使用。

Modbus TCP 设备可分为主站(Master)和从站(Slave)。同一网段中主站只有一个，从站可以有多个，主站向各从站发送请求帧，从站给予接收到的主站信息响应。在使用 TCP 通信时，主站为客户端（Client），主动与从设备建立连接；从站为服务器（Server），等待连接并给与请求响应信息。通信任务结束时，关闭 TCP 连接

建立连接后的 Modbus TCP 通信过程与 Modbus RTU 的通信过程类似：

- 主站请求：功能码+数据
- 从站正常响应：请求功能码+响应数据

- 从站异常响应：异常功能码 + 异常码，其中异常功能码即将请求功能码+0x80，异常码指示差错类型
- 注意：需要超时管理机制，避免无期限的等待可能不出现的应答。

1.6 HMI

HMI (Human Machine Interface)普遍用于自动化设备中，为用户提供与设备的交互接口。可以通过 HMI 实现设备运行状态监控、设备参数设置、将设备内的数据直观地显示出来。通常 HMI 设备通过通信接口与控制器建立信息交互，常见的通信接口有 RS232，RS485 和以太网接口。连接运动控制器、可编程序控制器（PLC）、变频器、交流伺服驱动器、仪表等工业控制设备，利用显示屏显示，通过输入单元（如触摸屏、键盘、鼠标等）写入工作参数或输入操作命令，实现人与机器信息交互。



图 1-61 HMI 触摸屏设备

在运动控制系统中，常见的 HMI 设备有触摸屏或专用的显示设备（如带显示的工控机）。一般而言，HMI 系统需要具备如下的基本功能：

- 设备工作状态显示，如指示灯、按钮、文字、图形、曲线等
- 数据、文字输入操作，打印输出
- 生产配方存储，设备生产数据记录
- 简单的逻辑和数值运算
- 可连接多种工业控制设备组网

为了实现人机交互的功能，触摸屏提供商会提供与产品对应的触摸屏组态软件开发工具。用户需要使用 HMI 的画面组态软件制作“工程文件”，再通过 PC 机和 HMI 产品的通信接口，把编制好的“工程文件”下载到 HMI 的触摸屏中运行。

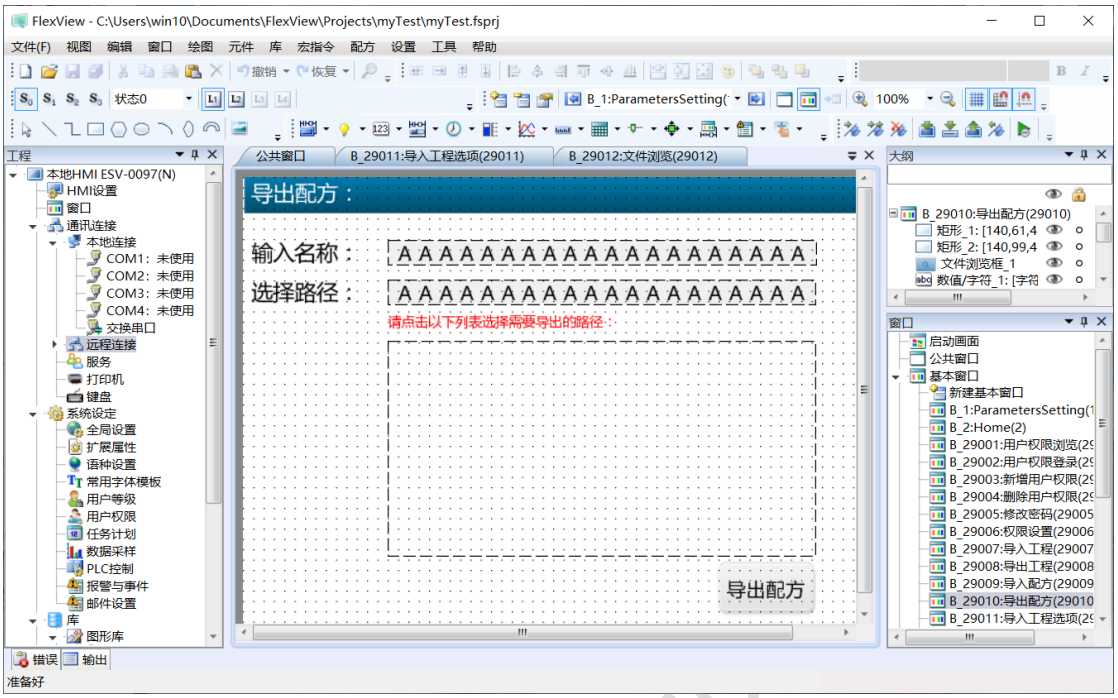


图 1-62 HMI 触摸屏软件开发工具

人机界面 HMI 触摸屏产品由硬件和软件两部分组成，硬件部分包括中央处理器 CPU、显示单元、输入单元、通讯接口、数据存贮单元等构成，其中 CPU 的性能通常决定了 HMI 产品的性能高低，反映产品的处理能力，是 HMI 的核心单元。在应用中选择 HMI 产品时，结合应用，需要考虑如下指标：

- 显示屏的尺寸及色彩、分辨率
- HMI 处理器的性能指标
- 输入方式：触摸屏（电阻式、电容式等）或薄膜键盘
- HMI 的用户程序内存容量及支持的扩展方式
- 应用环境参数，如工作温度、输入电压、抗震情况、接口电气防护等级等
- 通讯口种类及数量
- 安装方式与结构参数，如防护等级
- 需要满足的认证需求，如 CE, RoHS 等

此外，在选型时，还得考虑应用的其他需求、以及供货、性价比等方面的因素，参考产品的数据手册和应用需求进行选型。

人机界面 HMI 触摸屏厂商会提供对应的 PC 端应用开发组态软件，其开发过程通常如下：

- 确定 HMI 触摸屏的主要任务，选择适合的 HMI 产品
- 在 PC 机上，使用应用开发组态软件，编辑“工程文件”
- 调试并保存“工程文件”

- 使用 PC 机连接 HMI 硬件，下载“工程文件”到 HMI 中
- 连接 HMI 和控制器（如运动控制器、PLC、仪表等），实现人机交互

本章内容主要介绍运动控制系统的各个构成部件，对于各个部件的介绍，更倾向于实际的应用，对于运动控制技术的理论为展开介绍。

对于 HMI 的使用，用户需通过查阅 HMI 的数据手册、帮助文档等渠道来学习。

ESTUN & Trio

2 Motion Perfect V5

基于微软 Windows 的 PC 端应用 Motion Perfect V5 为 Trio 提供的机器自动化开发平台，用于连接 Trio 运动控制技术公司的多任务控制器，伺服驱动器、HMI 和 IO 扩展产品。支持离线仿真、可视化、集成开发环境、系统 IO 配置和 DX4 伺服驱动配置等功能。

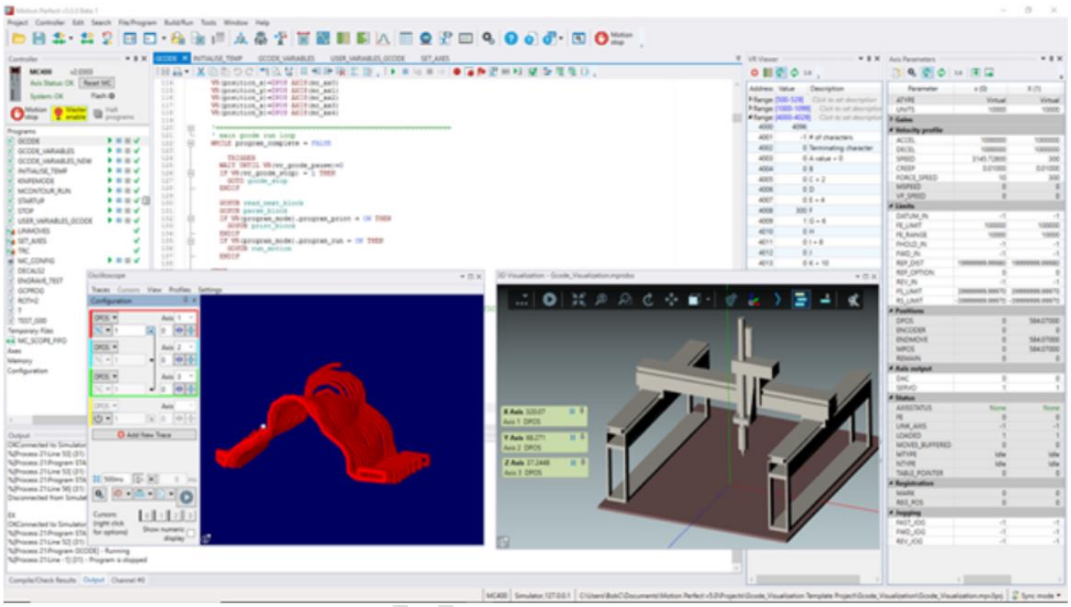


图 2-1 Motion Perfect V5 窗口

2.1 软件的安装及基本使用

2.1.1 安装环境要求

操作系统	Windows xp、windows 7	Windows 10
. Net Library	4.6.2	
处理器	双核或更高	双核或更高
RAM	2G	4G+
硬盘空间	200Mb+	500Mb+
3d Graphics	DirectX 9	DirectX 11
通讯方式	Ethernet	Ethernet

2.1.2 语言设置

1. 软件安装完成，双击桌面图标  ，启动软件；

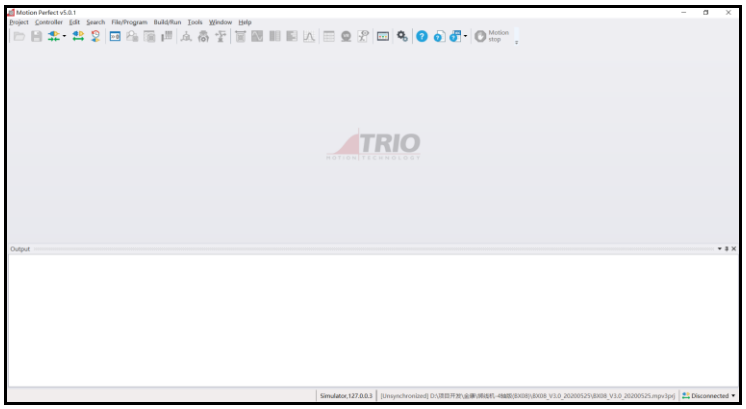


图 2-2 软件启动画面

2. 点击菜单栏“TOOLS/工具”按钮，

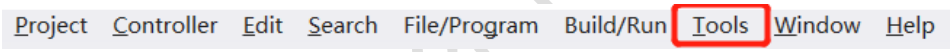


图 2-3 菜单栏

3. 下拉列表点击“Options”按钮，进入软件设置界面；

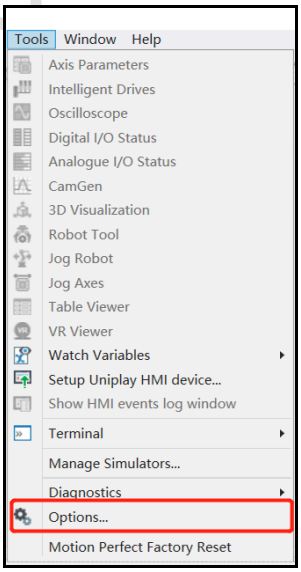


图 2-4 工具栏选项

4. 在弹出的对话框中，选择“Language”选项，选择 Mpv5 软件环境语言，点击“Apply and Reset”按钮生效；

ESTUN & Trio

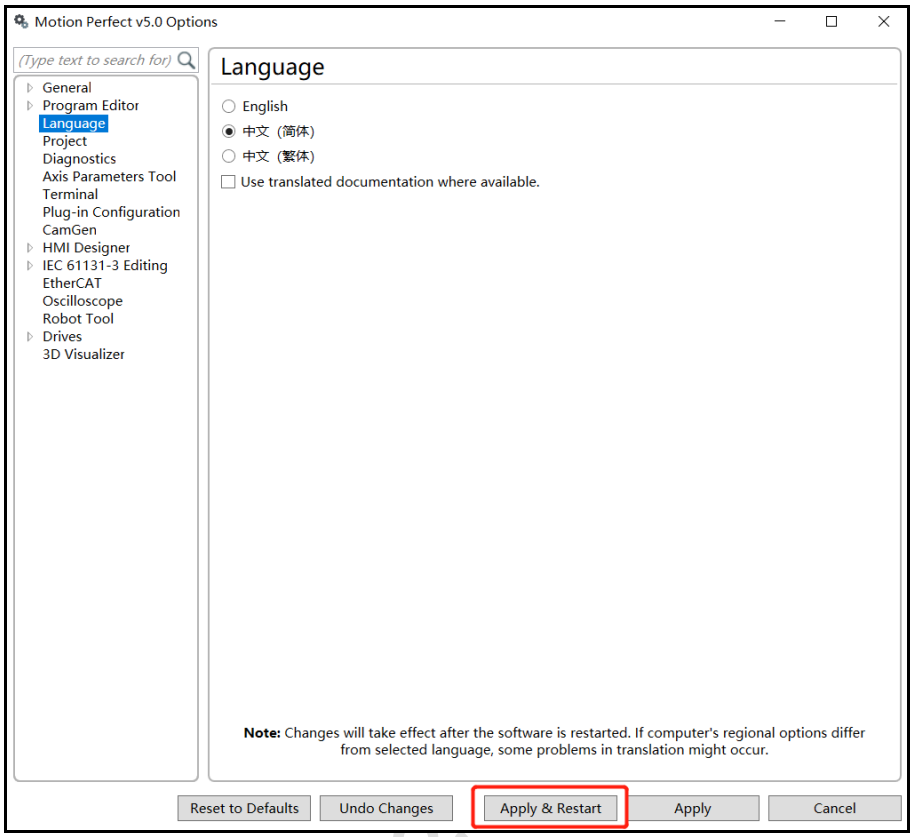


图 2-5 语言设置界面

5. 语言设置为中文简体，软件重启，Mpv5 界面如图 2-6，设置生效。

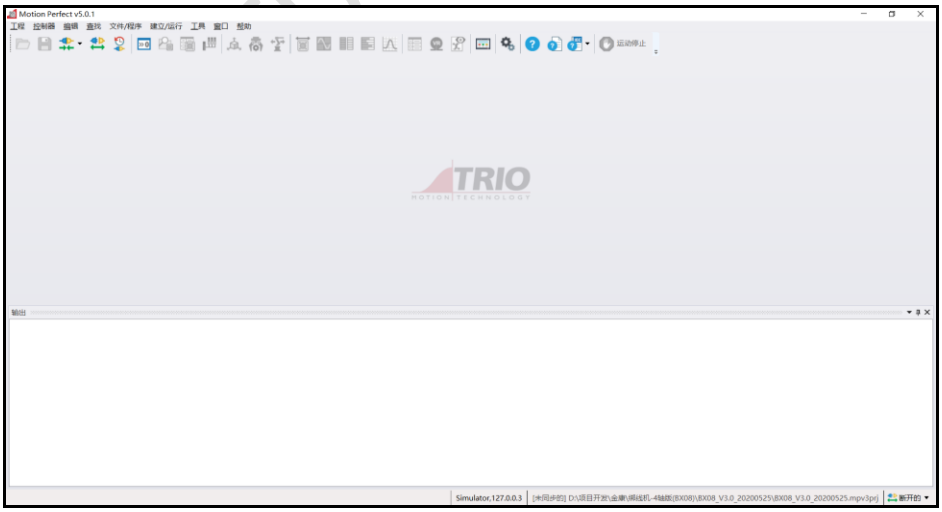


图 2-6 中文启动界面

2.2 软件的连接

2.2.1 Ethernet 连接设置

1. 确保控制器已通电，控制器 Ethernet 网口与电脑网口已连接；
2. 确认控制器 IP 地址，默认 Ip 地址为 192.168.0.250；通过插拔网线，可在控制器液晶显示面板查看 IP 地址；
3. 设置 IPV4 地址，进入网络和 INTERNET 设置界面；



图 2-7 PC 网络和 INTERNET 设置界面

4. 选择“更改适配器设置”，进入网络连接界面；



图 2-8 网络连接界面

5. 双击控制器所连接的网络端口，双击“Internet 协议版本 4（TCP/IPv4）”，设置 IPv4 地址；

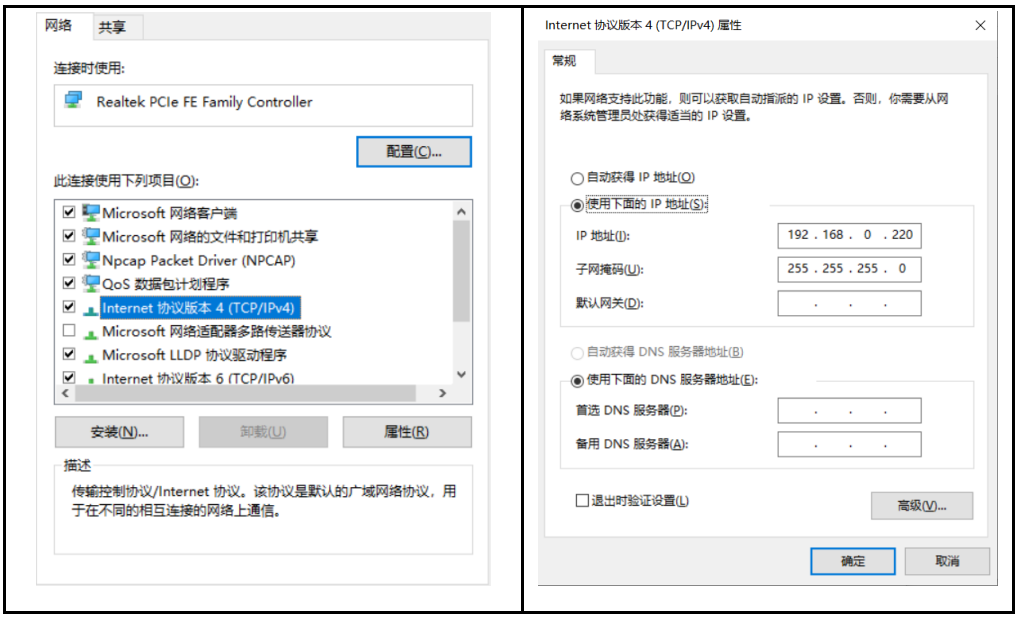


图 2-9 IPV4 设置界面

6. 参考图 2-9，将 IP 地址设置为与控制器相同网段；

7. 打开 Motion Perfect V5；

8. 点击工具栏“Controller/控制器”按钮，下拉列表点击“连接设置...”按钮，弹出连接设置对话框；

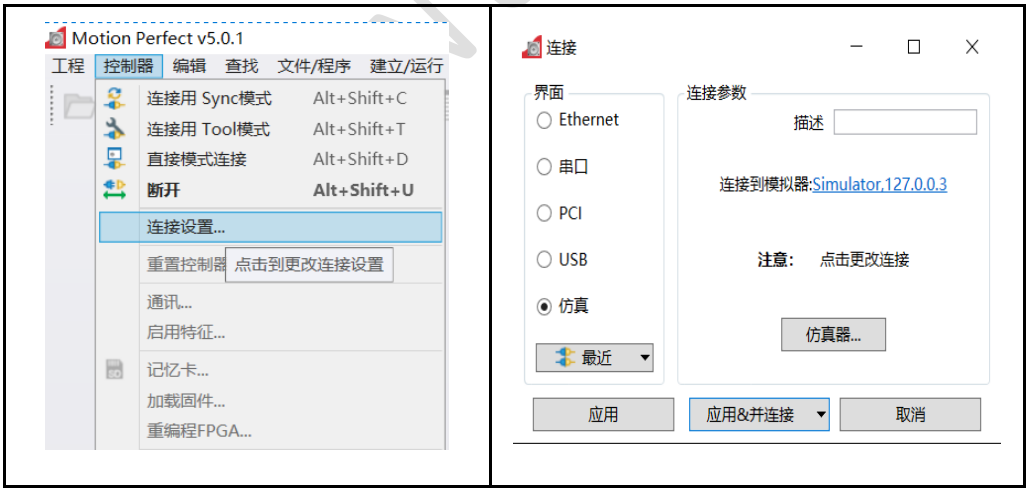


图 2-10 连接设置界面

9. 选择“Ethernet”选项，设置控制器 IP 地址（参考控制器面板），IP 端口设置为 23；

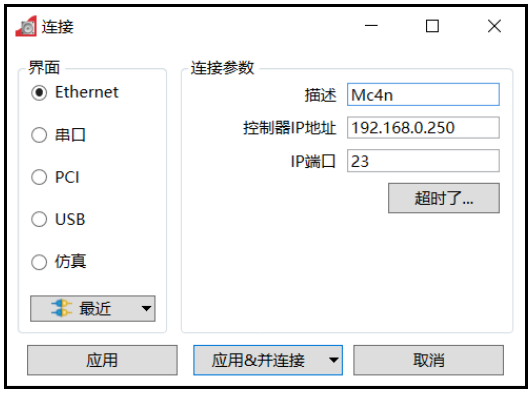


图 2-11 控制器 IP 设置

10. 点击“应用”按钮或者“应用&并连接”按钮，确定 IP 设置；

2.2.2 仿真器连接设置

用户在使用 Mpv5 软件编写工程代码时，如无实际控制器，可借助仿真控制器编写、调试程序。

1. MC400 Simulater 为虚拟控制器，用于离线编程，MPV5 安装时会将 MC400 Simulater 同步安装；Mpv5 软件运行时，窗体上会显示仿真器图标；



图 2-12 虚拟仿真器图片

2. 参考“ETHERNET 连接设置”，打开连接设置对话框；

3. 选择“仿真”选项；点击“连接到仿真器”，选择仿真器 IP 地址及配置网络端口；

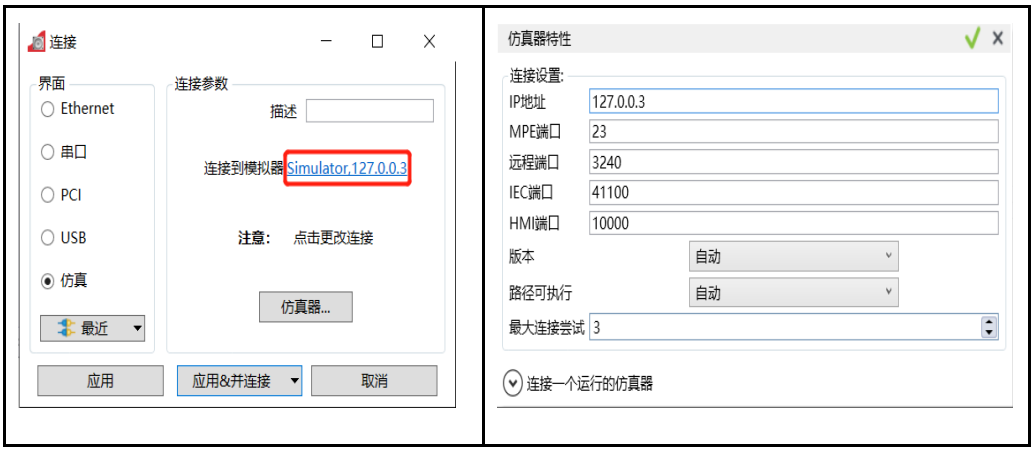


图 2-13 仿真器的设置

ESTUN & Trio

4. 点击“仿真器...”按钮，可增加新的仿真器；



图 2-14 添加新的仿真器

5. 仿真器可在此界面进行管理；

Button	Function
	运行仿真器
	停止仿真器
	编辑仿真器
	删除仿真器

6. 点击“应用”按钮或者“应用&并连接”按钮。

2.2.3 连接模式

Mpv5 与控制器的连接包含四种，分别为：

- 断开模式（Disconnected Mode）；
- 直接模式（Direct Mode）；
- 工具模式（Tool Mode）；
- 同步模式（Sync Mode）。

不同模式下，用户可操作权限不一致。

2.2.3.1 断开模式

断开模式下，用户无法对控制器进行任何操作。

ESTUN & TYO

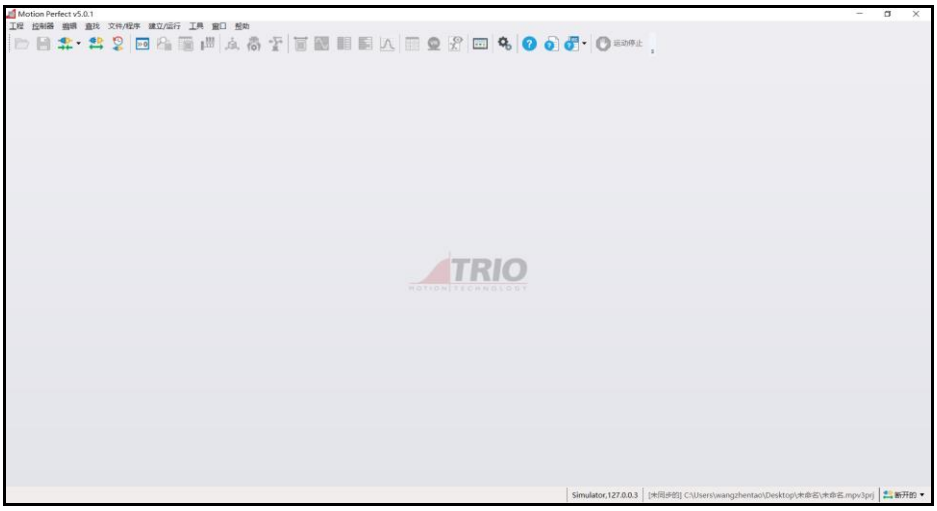


图 2-15 断开模式

2.2.3.2 直接模式

直接模式下，用户可在命令窗口 0 进行操作，执行简单的命令操作。

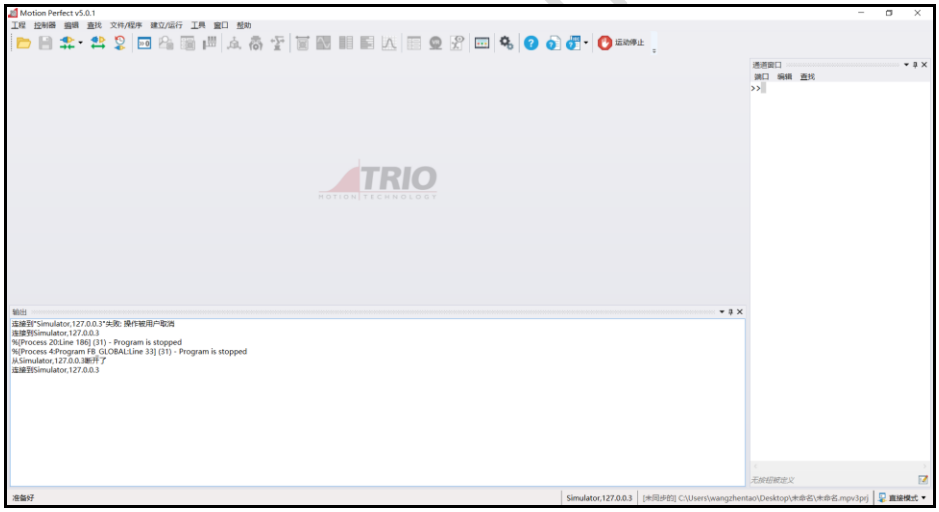


图 2-16 直接模式

2.2.3.3 工具模式

工具模式下，用户可进行以下操作：

1) 命令窗口 0 可进行操作
2) 工具栏工具可以被操作
3) 控制器树窗口可以查看

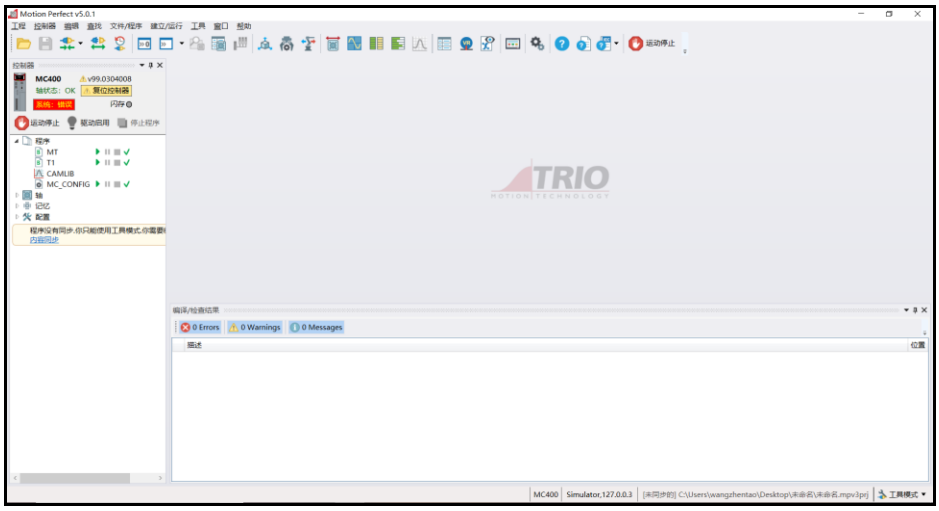


图 2-17 工具模式

2.2.3.4 同步模式

同步模式下，用户可进行以下操作：

1) 命令窗口 0 可进行操作；

2) 工具栏工具可以被操作；

3) 程序树可以查看；

4) 程序文本可以被打开、编辑；

5) 用户编写程序，需要在同步模式下进行；如果当前无实体控制器，可基于 Mpv5 提供的仿真器编写程序；

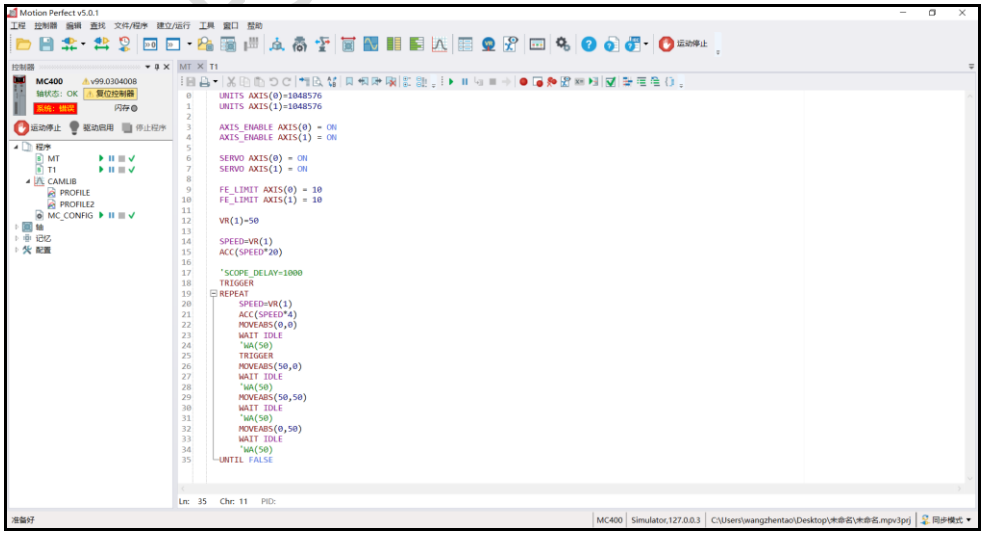


图 2-18 同步模式

ESTUN & Trio

2.3 控制器窗体

2.3.1 主菜单



图 2-19 主菜单

2.3.1.1 Project（工程）

点击主菜单栏“工程”按钮，弹出工程选项列表。

选项	描述
New（新建）	创建新的工程，并清除控制器所有文件
Load（调取）	加载其他工程至控制器
Change（改变）	改变同步路径
Create from controller（从控制器创建）	创建新的工程文件夹，读取控制器程序
Save（保存）	保存当前工程
Save As（保存为）	保存当前工程为其他名称
Export（导出）	导出工程为.zip 格式.RPJ 格式
Manage Encryption（加密工程）	工程加密及加密钥生成
Project Check（工程检查）	检查当前工程文件
Backups（备份）	打开备份管理窗口，创建或者备份工程
Close（关闭）	关闭当前工程，返回直接模式
Import file/program …（导入文件/程序）	导入程序文件至当前工程
Modify Startup Program（修改 STARTUP）	修改 STARTUP 程序
Recent Projects（最近的工程）	便捷打开最近编辑的工程
Solution Manager（解决方案管理器）	打开解决方案管理器，便于用户操作多个控制器
Print（打印）	打印当前页面
Exit（退出）	退出软件

2.3.2 Controller（控制器）

点击主菜单栏“控制器”按钮，弹出控制器选项列表。

ESTUN 内部资料

选项	描述
Connect in Sync Mode（连接用 Sync Mode）	连接到同步模式
Connect in Tool Mode （连接用 Tool Mode）	连接到工具模式
Connect in Direct Mode（连接用 Direct Mode）	连接到直接模式
Disconnect（断开）	断开控制器连接
Connection Settings（连接设置）	设置 Mpv5 连接对象
Reset Controller（重置控制器）	将控制器复位
Communications（通讯……）	打开控制器所有通讯接口的设置子界面
Enable Features（启用特征……）	打开控制器软件激活码的子界面
Memory Card（记忆卡……）	打开内存卡管理界面
Load Firmware（加载固件……）	打开控制器更新固件子界面
Reprogram FPGA（重编程 FPGA）	打开控制器 FPGA 程序更新界面
Directory	打开文件在控制器内部存储路径子窗口
Process	打开任务状态子窗口
Import Values	从文件中导入数值至 VR 或者 TABLE
Export Values	将 VR/TABLE 的数值导出至文件
Download All Files	下载控制器的所有文件
Lock Controller	锁定控制器
Unlock Controller	解锁控制器
Date and Time	打开控制器时间设置界面
HMI	打开 HMI 管理界面

ESTUN &TRIO

2.3.2.1 Edit（编辑）

点击主菜单栏“编辑”按钮，弹出编辑选项列表。

选项	描述
Undo（撤销）	取消最新的一次操作
Redo（重做）	恢复最新的一次修改
Cut（剪切）	剪切选中的文本
Copy（拷贝）	拷贝选中的文本
Paste（粘贴）	复制选中的文本
Select All（选择全部）	选择全部文本
Select None（选择没	取消选择全部文本

选项	描述
有）	
Delete（删除）	删除全部文件

2.3.2.2 Search（查找）

点击主菜单栏“查找”按钮，弹出查找选项列表。

选项	描述
查找	查找字符串
查找下一个	查找搜寻字符串的下一个位置
查找上一个	查找搜寻字符串的上一个位置
替换	替换字符串文本
在工程中查找	在工程所有文件中查找字符串
在工程中替换	在工程所有文件中替换字符串

2.3.2.3 文件/程序

点击主菜单栏“文件/程序”按钮，弹出文件/程序选项列表。

选项	描述
新建	新建一个空的程序文件
调取	从电脑中加载程序文件至工程
保存到磁盘	将工程中的文件保存到磁盘
编辑	编辑当前工程中的文件
调试	调试当前工程中的程序文件
保存	保存修改
复制	复制工程里的文件
重命名	修改工程中的文件名称
D 删除	删除选中的工程文件
删除所有	删除工程中所有文件

2.3.2.4 建立/运行

点击主菜单栏“建立/运行”按钮，弹出建立/运行选项列表。

选项	描述
编译	编译选定的程序文件
编译所有	编译工程中的所有程序文件
编译/检查结果窗口	打开/关闭编译结果窗口

ESTUN & Trio

选项	描述
运行	运行选择的程序文件
单步	单步运行选择的程序文件
介入	单步运行时，跳转至函数块中单步运行
跳出	单步运行时，跳出函数块中单步运行
暂停	选择的程序文件暂停执行
停止	选择的程序文件停止执行
停止所有	所有正在运行的程序文件停止执行

2.3.3 工具栏

用户可以通过工具栏选项，快速使用 Mpv5 的功能。

图标	功能	描述
	打开工程	打开工程并与控制器内容同步
	保存工程	保存当前工程到磁盘（仅同步模式有效）
	连接	打开子菜单，选择不同连接模式
	断开连接	断开连接
	最近的工程	打开最近的工程子窗口，用户可选择最近的工程或者最近的连接
	窗口 0	如果处于同步模式、工具模式或者直接模式，打开终端窗口 0
	终端窗口	如果处于同步模式、工具模式，用户选择打开终端窗口 5、6、7
	工程查找	打开工程查找对话框
	轴参数	打开轴参数工具
	智能轴驱动	打开智能轴驱动界面
	3D 可视化	打开 3D 可视化界面
	机器人工具	打开机器人工具调试界面

ESTUN & TYC

图标	功能	描述
	机器人点动	打开机器人点动调试界面
	轴点动	打开轴点动调试界面
	示波器	打开示波器工具
	数字量输入输出	打开数字量输入输出监控工具
	模拟量输入输出	打开模拟量输入输出监控工具
	凸轮曲线	创建或者编辑凸轮曲线
	TABLE 监控	打开 TABLE 监控工具
	VR 监控	打开 VR 监控工具
	局部变量监控	打开变量监控工具
	HMI 模拟运行	仿真平台运行 HMI 界面程序
	选项	打开选项窗体
	Motion Perfect 帮助	显示 Motion Perfect 帮助文件
	TrioBasic 帮助	显示 TrioBASIC 帮助文件
	IEC 帮助	显示 IEC 帮助文件
	运动停止	停止所有运动及所有程序

ESTUN & Trio

2.3.4 控制树

控制器树窗口在工具模式或者同步模式下，用户可以打开。控制器树窗口包含控制器的相关信息。

2.3.4.1 显示或者关闭

用户可以操作“菜单栏-窗口-控制器树窗口”，关闭或者显示控制器树窗口。

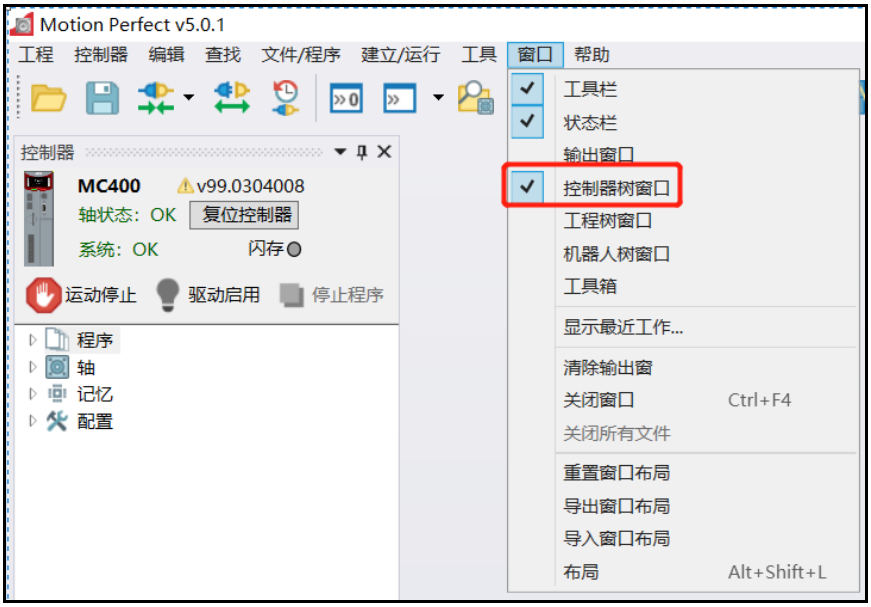


图 2-20 控制器树窗口显示

2.3.4.2 工程树头

控制器工程树头部显示连接对象的基本信息，包含常用操作按钮。



图 2-21 控制器工程树头部信息

图标或按钮	功能
	当前连接控制器图标，将光标仿真于该图标，显示控制器详细信息； 
MC400	显示控制器型号

图标或按钮	功能
v99.0304008	显示控制器的固件版本
轴状态：OK	显示轴状态； 如果轴报警，显示为红色；
系统：OK	显示系统状态； 如果系统报警，显示为红色；点击该按钮，弹出系统故障对话框；
复位控制器	点击该按钮，复位控制器； 如果 MC_CONFIG 文件发生修改，该按钮显示黄色； Reset MC
运动停止	运动停止按钮； 停止所有用户程序和轴运动；
驱动启用	WDOG（轴使能总开关）开关按钮；
停止程序	停止程序按钮； 停止所有用户程序；

2.3.4.3 工程树

控制器树包含工程的基本信息，程序文件，轴信息，变量信息，配置信息。



图 2-22 工程树系统

ESTUN & Trio

控制器树内容由以下几个部分组成：

名称	含义
程序	显示存储在控制器中的程序文件或者其他文件
轴	显示控制器的基本轴信息
记忆	显示控制器的内存信息
配置	显示控制器的基本信息及通讯配置

2.4 解决方案管理

用户使用解决方案管理工具，可便捷的将多个工程与不同地址的控制器进行管理。

用户可以同时连接多个控制器，且能编辑多个控制器的内部程序。

1. 点击“菜单栏-工程-解决方案管理器”，打开解决方案管理器；

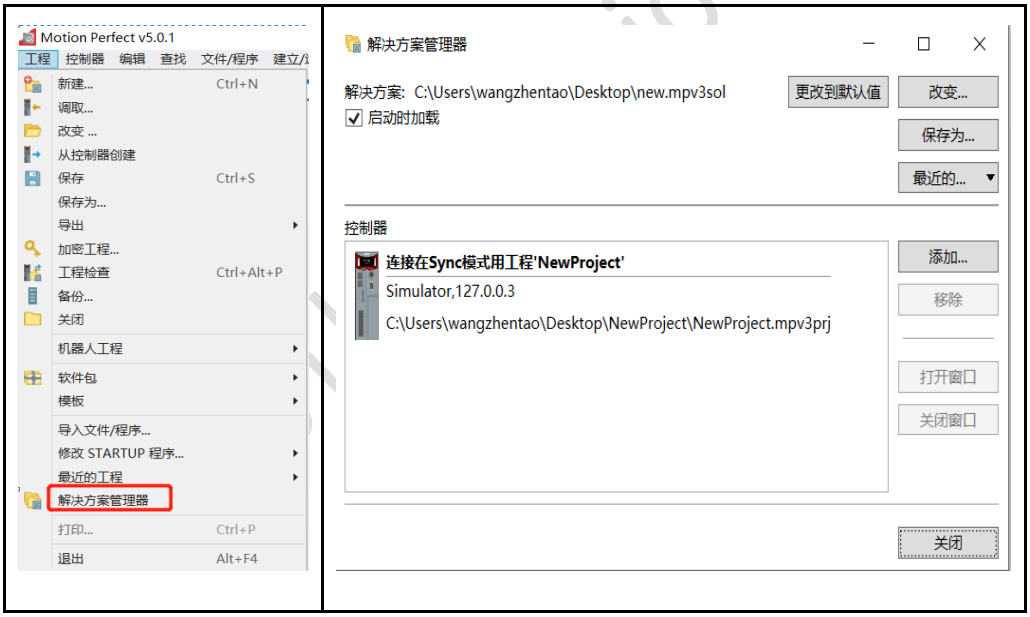


图 2-23 解决方案管理器

2. 点击“添加...”按钮，可以添加新的连接；并弹出新工程的连接选项；

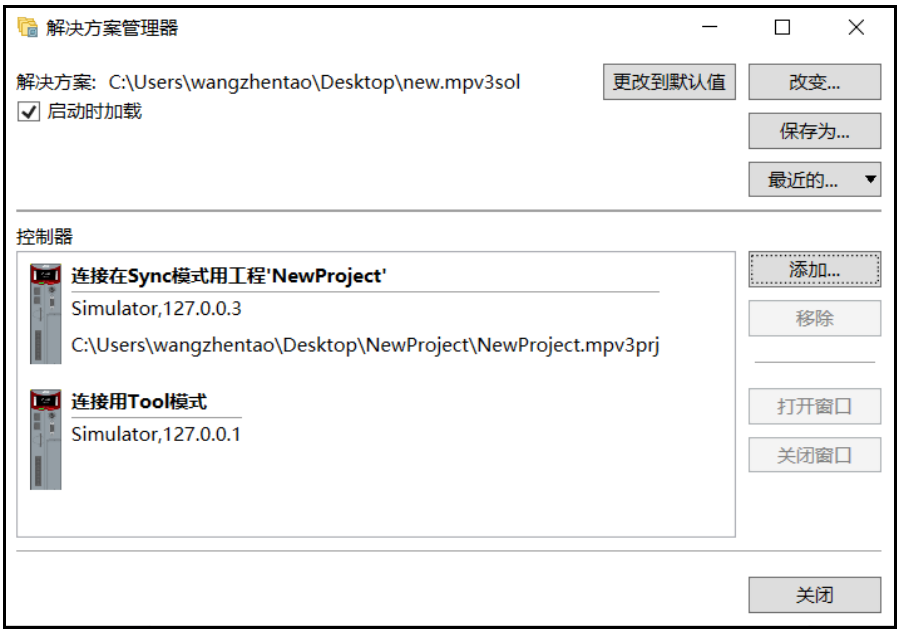


图 2-24 添加新的控制器

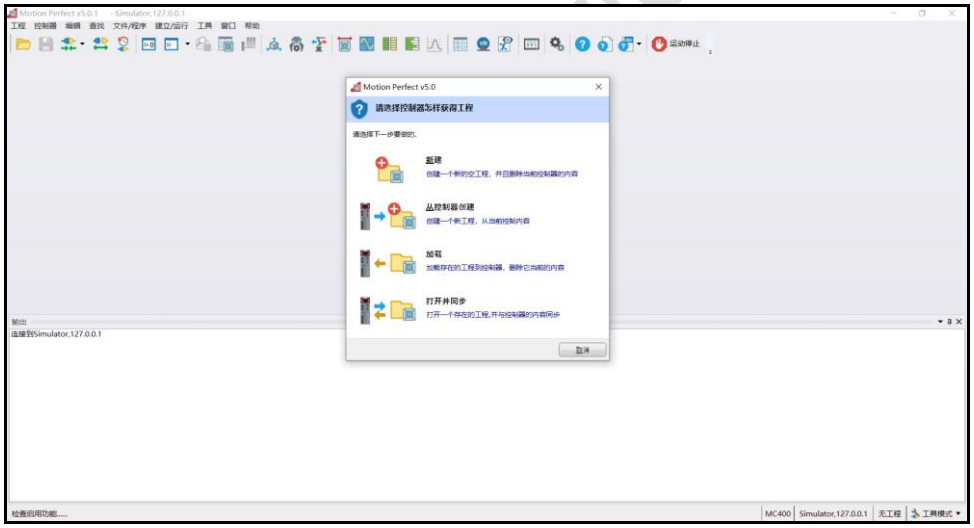


图 2-25 新的控制器连接选项

3. 点击“移除...”按钮，可删除连接；

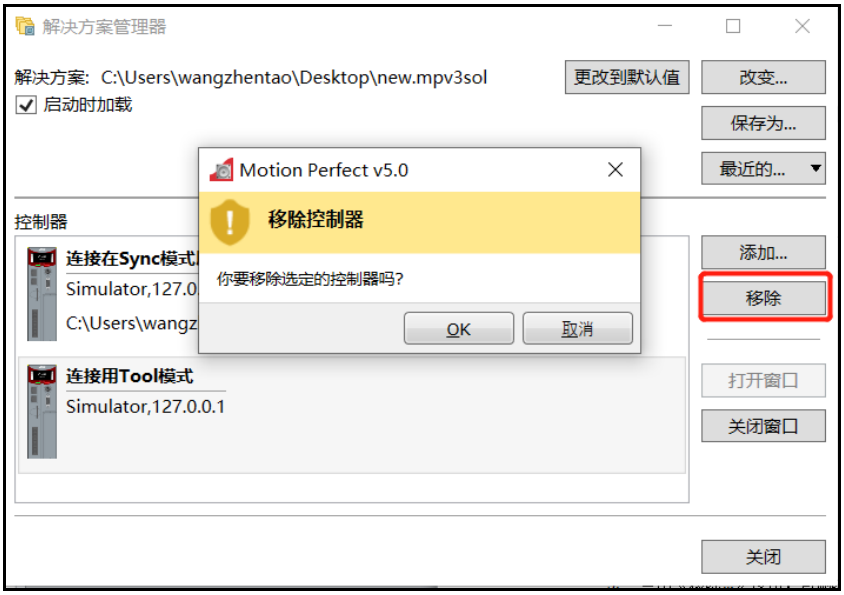


图 2-26 移除控制器

2.5 工程操作

2.5.1 创建工程

用户可以创建一个新的工程或者基于模板的工程。

新建工程操作会清空控制器内部的所有文件。

1. 点击按钮“菜单栏-工程-新建...”，弹出新建工程对话框；

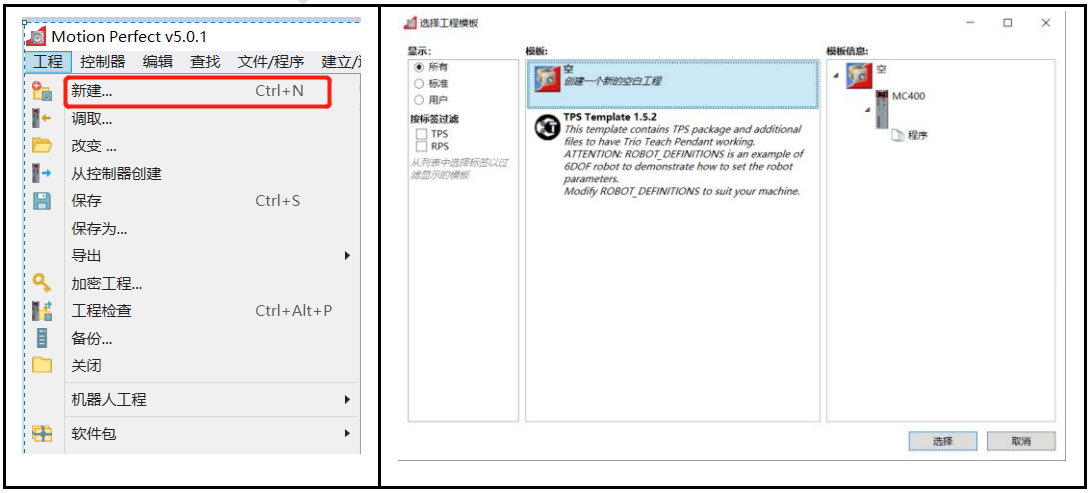


图 2-27 打开新建工程

2. 点击“选择”按钮，创建新的工程；在弹出对话框选择工程的存储路径和工程名称；

ESTUN & Trio

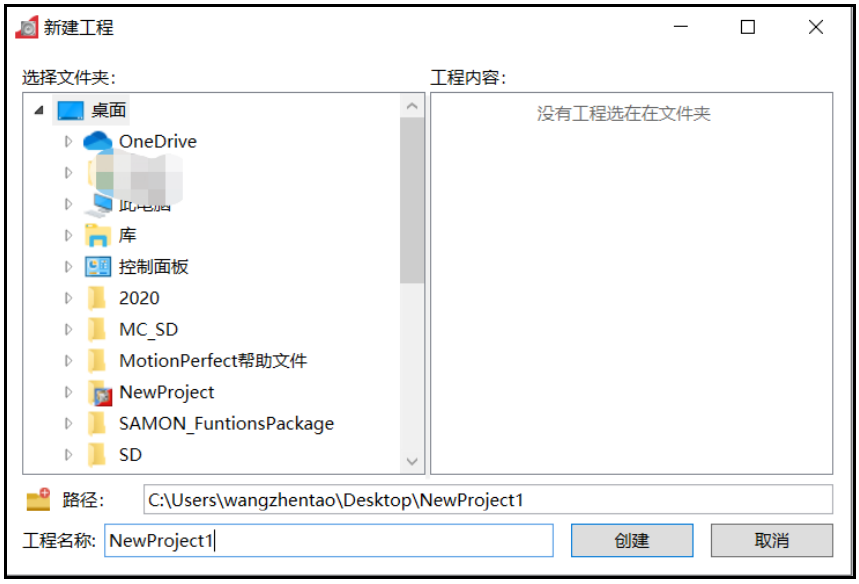


图 2-28 新建工程对话框

3. 点击 “创建”按钮，创建新的工程；

2.5.2 上传工程

用户可将控制器内的程序文件上传至 PC 端的工程文件夹，并进入同步模式。

1. 点击按钮“菜单栏-工程-从控制器创建...”，弹出保存工程对话框；

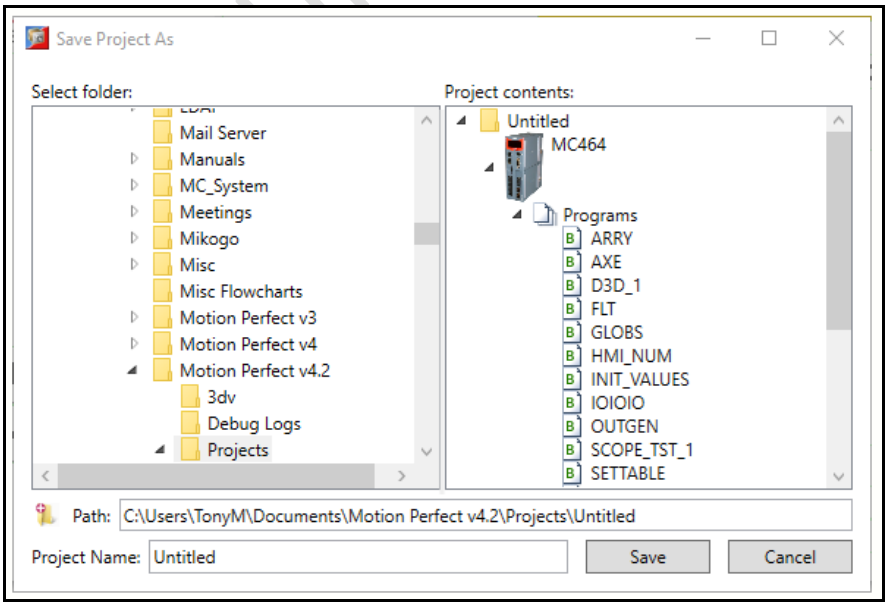


图 2-29 上传工程对话框

2. 选择存储路径，命名工程名，点击“保存”按钮确认；

2.5.3 同步工程

用户选择磁盘中的项目文件夹，同步过程时：

- 用户可选择将部分程序文件加载至控制器；
- 可以选择将控制器文件读取至工程文件夹；
- 同步完成，工程文件夹与控制器内容一致，进入同步模式；

具体操作如下：

1. 点击按钮“菜单栏-工程-改变”，弹出改变工程对话框；

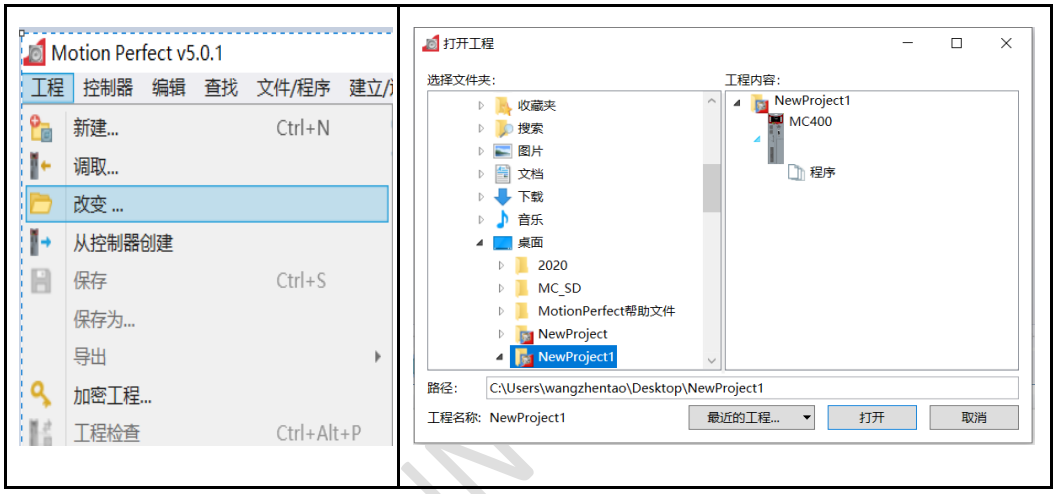


图 2-30 打开改变工程对话框

2. 选择需要同步的文件夹，点击“打开”按钮，弹出同步窗口；

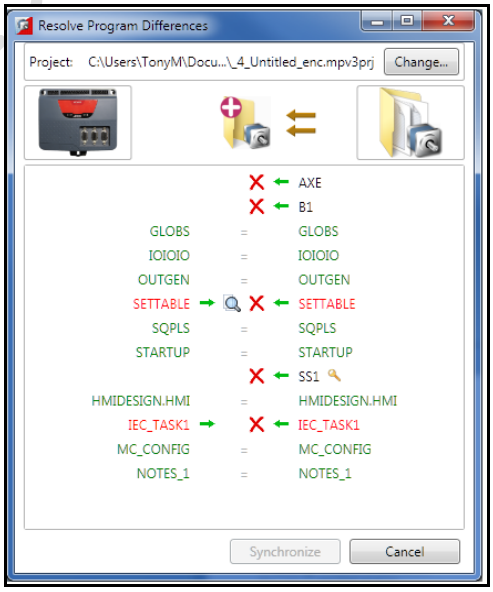


图 2-31 同步工程

3. 根据实际情况选择文件，点击“同步”按钮，进入同步模式。

Icon/Button	含义
Change	改变需要同步的文件夹
	创新新的空文件夹
	将文件夹内的所有文件同步至控制器
	将控制器内的所有文件同步至文件夹
	将文件夹内的指定文件同步至控制器
	将控制器内的指定文件同步至文件夹
	删除指定文件
	打开文件对比软件（文件比对设置），比较相同名称文件内容的差异性

2.5.4 修改工程

修改工程：用户将磁盘中选中的工程文件夹文件加载至控制器。

此操作会删除控制器内先前工程的所有内容。

1. 点击按钮“菜单栏-工程-调取”，弹出修改工程对话框；

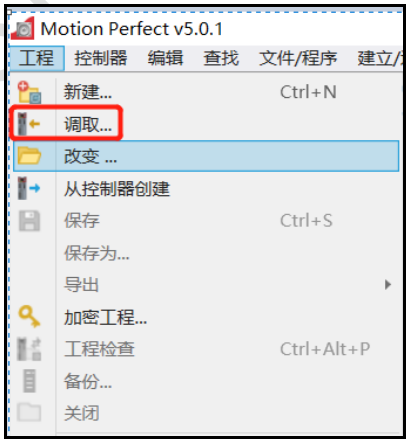


图 2-32 打开修改工程对话框

2. 选择需要加载至控制器的工程文件夹，点击“加载”按钮，完成操作；

ESTUN & Trio

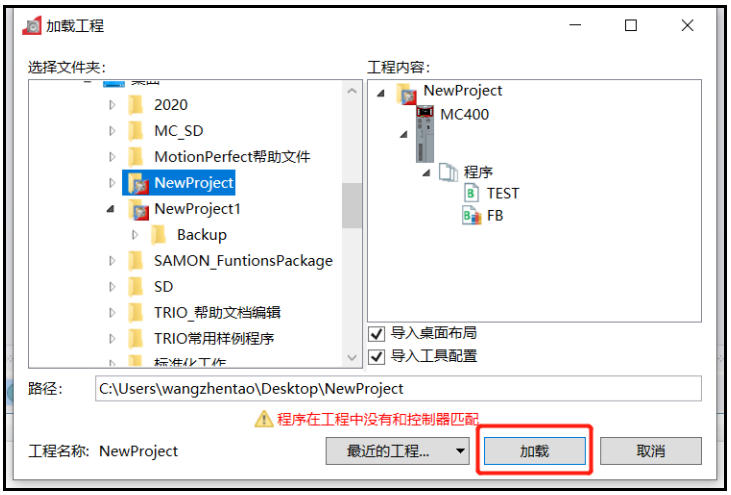


图 2-33 选择工程文件夹

2.5.5 软件包操作

软件包将一组程序文件（BASIC、BASIC LIB 等文件）通过处理，形成软件包，使用户不能看到程序文件的源码，具有保护源码的功能。开发者可将需要保密或可以模块化复用的程序文件发布成软件包，实现对软件包内的程序为文件的保密（Mpv5 中不显示软件包内程序文件的源代码），也可以对软件包的复用。

2.5.5.1 软件包添加到工程

开发者可以将已有的软件包加载至当前工程，具体操作如下：

1. 点击“菜单栏-工程-软件包”按钮，弹出选项；

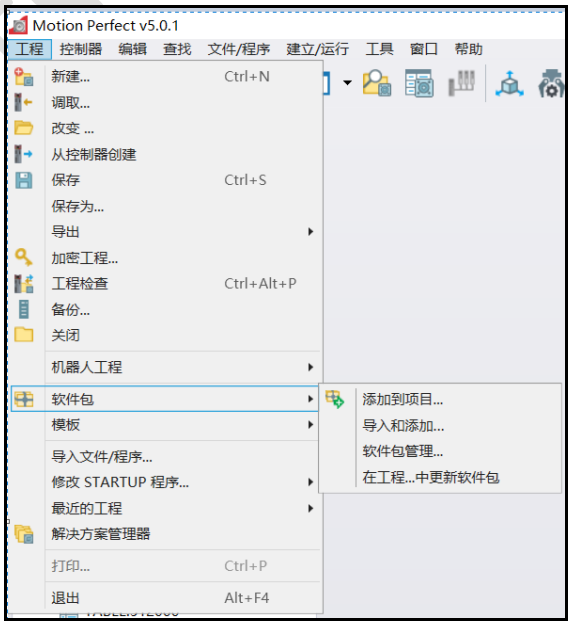


图 2-34 打开“添加到项目”选项

2. 点击“添加到项目”按钮，弹出“管理软件包”对话框；

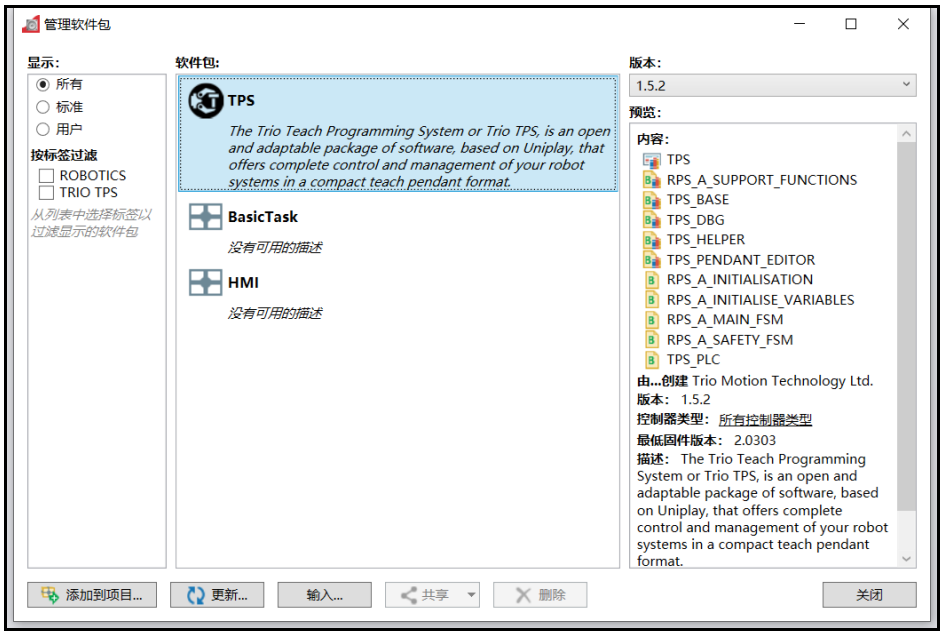


图 2-35 管理软件包窗口

3. 选择需要添加的软件包，点击“添加到项目……”按钮，完成操作；

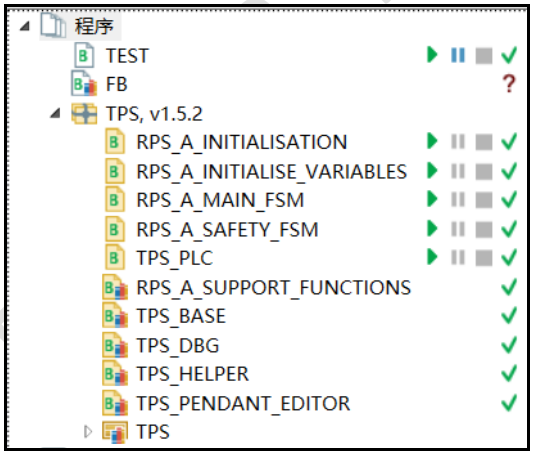


图 2-36 软件包添加成功

2.5.5.2 软件包生成

开发者可将工程中的部分代码文件加密，生成软件包。

具体操作如下：

1. 将需要加密打包的程序添加至相同组。选中文件，进行添加至组操作；如果无组别，可创建新组；

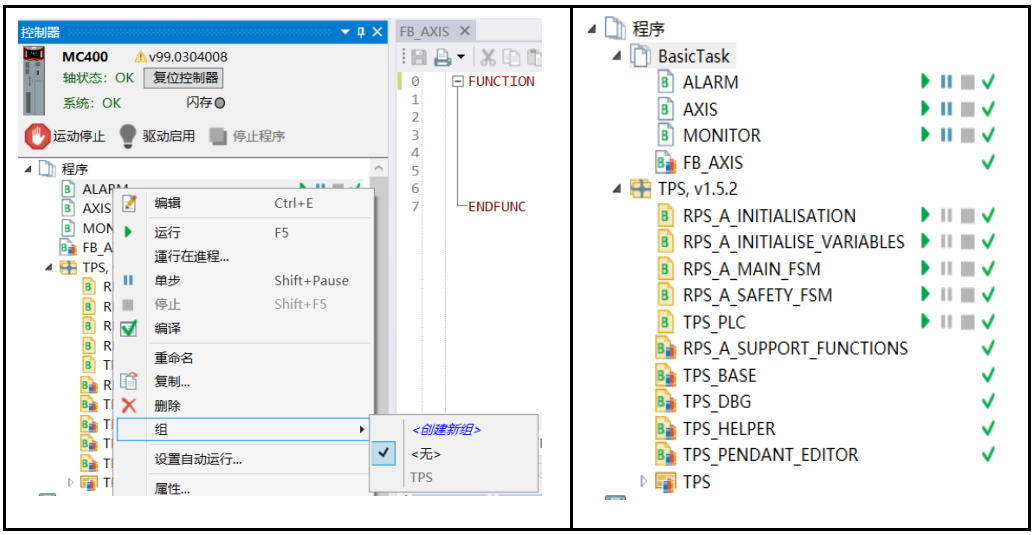


图 2-37 添加文件至组

2. 选中组，右键，点击“从组中创建软件包……”；

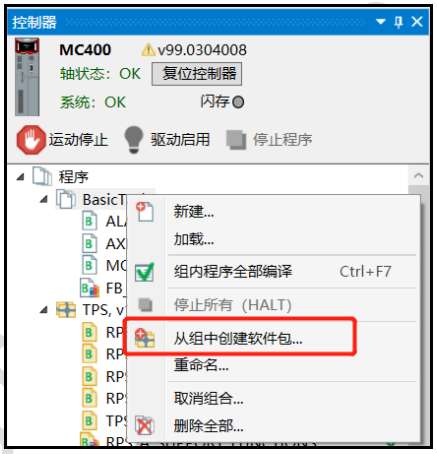


图 2-38 从组中创建软件包

3. 填写软件包的名称，版本号，描述；如有帮助文件，可添加；

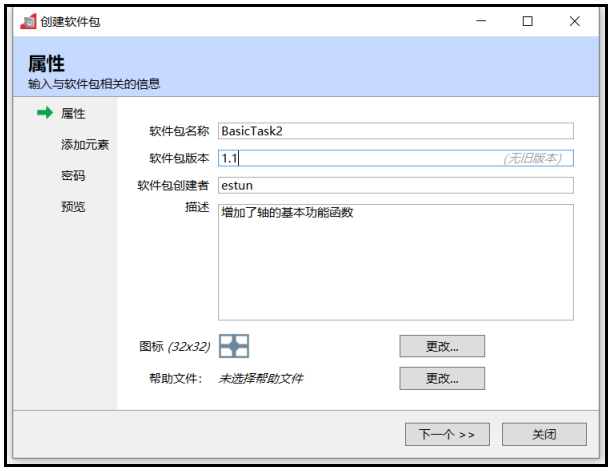


图 2-39 设置软件包信息

4. 设置软件包的使用限制；

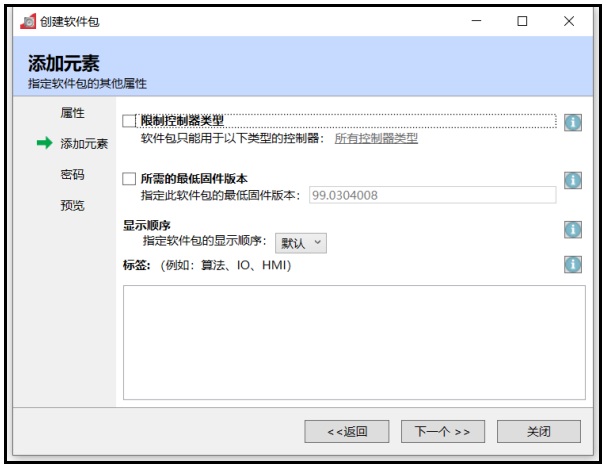


图 2-40 设置软件包使用限制

5. 设置软件包的密码。

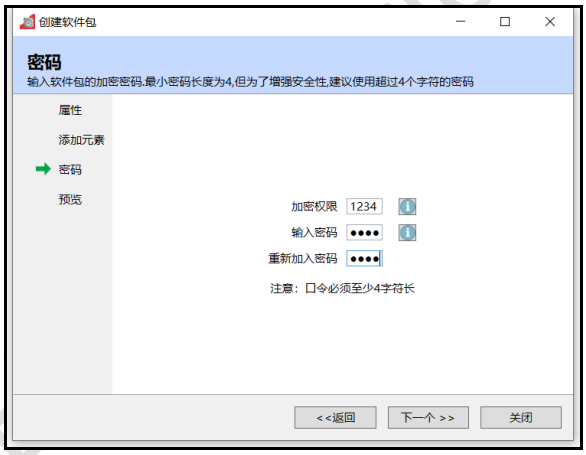


图 2-41 设置软件包密码

6. 点击“完成”按钮，生成软件包；

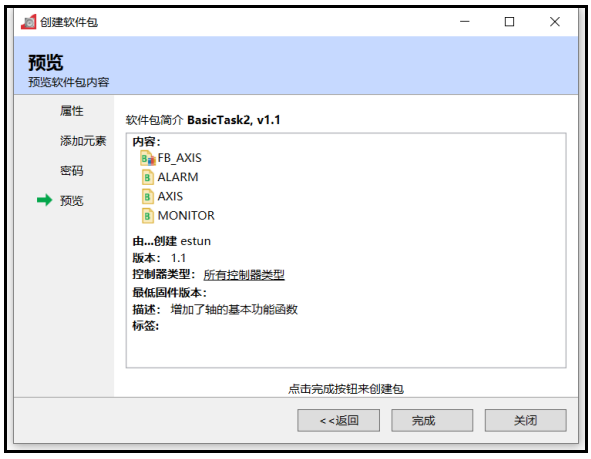


图 2-42 生成软件包

7. 可将软件包导出成文件，分享给其他开发者；



图 2-43 软件包导出界面

8. 点击“菜单栏-工程-软件包-软件包管理”按钮，可看到新增的软件包；

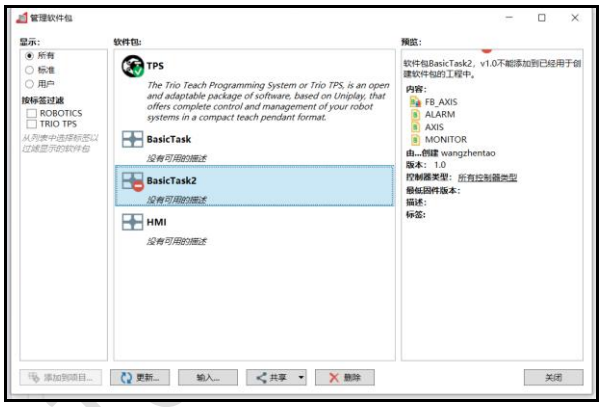


图 2-44 查看新增软件包

2.5.5.3 软件包管理

点击“菜单栏-工程-软件包-软件包管理”按钮，可以打开管理软件包界面。

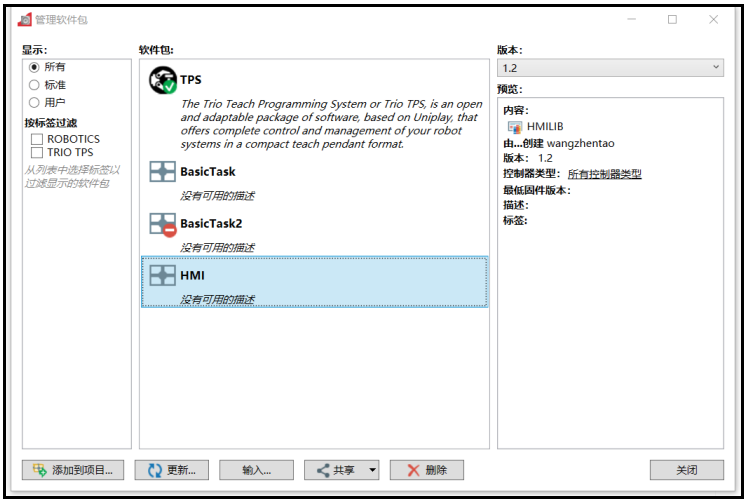


图 2-45 管理软件包窗口

包含功能有：

1. 切换软件包的显示；

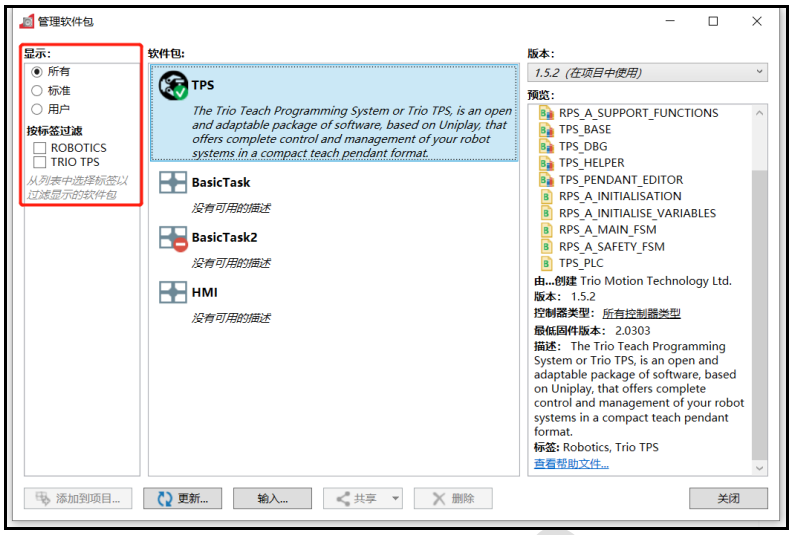


图 2-46 软件包显示选项

2. 选中软件包，进行软件包的操作；

按钮	功能描述
添加到项目	将选中的软件包添加到项目工程
更新	更新选中的软件包
输入.....	导入软件包文件，在列表中显示
共享	将选中的软件包生成文件，分享给其他开发者
删除	删除选中的软件包

3. 软件包的简要描述；如有帮助文件，可点击“查看帮助文件...”按钮；

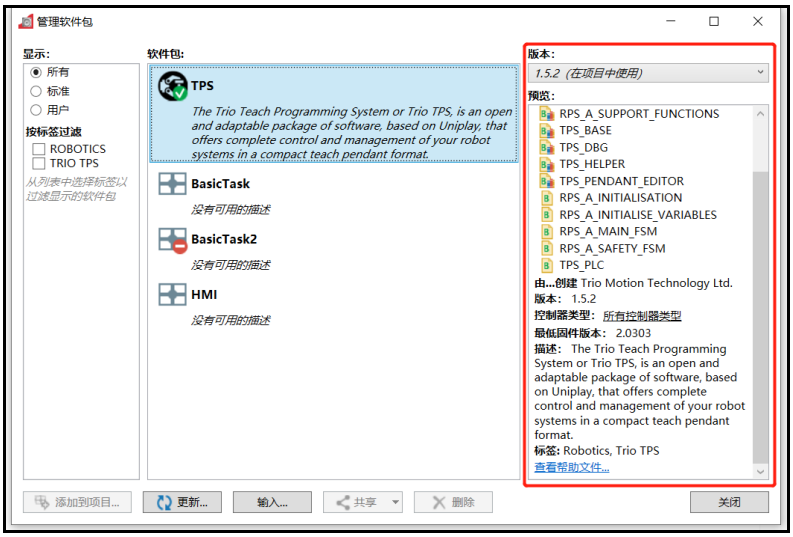


图 2-47 软件包简述

2.6 程序文件

一个工程由多个程序文件组成，开发者可以在 Motion Perfcet 软件中对程序文件进行新建、删除、复制、编辑等操作。

2.6.1 文件类型

控制器内部支持不同类型的程序文件。

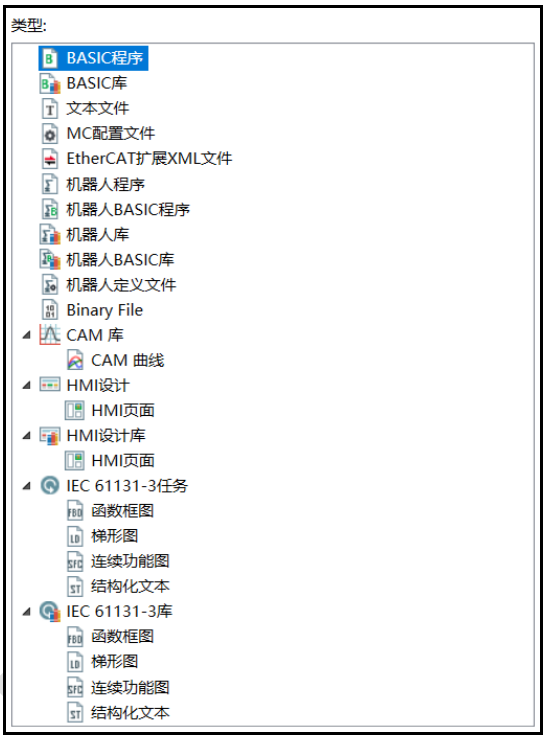


图 2-48 程序文件类型

每个文件类型的含义及功能参考下表：

文件类型	功能
BASIC 程序	可执行文件； 文件名称用户自定义； 开发者可在文件中编写基于 TRIOBASIC 语言的代码； 一个工程中可以包含多个 BASIC 程序；
BASIC 库	文件名称用户自定义； 开发者可在文件中编写基于 TRIOBASIC 语言的功能块代码； 一个工程中可以包含多个 BASIC 库程序；
文本文件	不可执行文件；

文件类型	功能
	文件名称用户自定义； 一个工程中可以包含多个文本文件；
MC 配置文件	系统参数设置文件； 一个工程中最多包含一个 MC_CONFIG 文件； MC_CONFIG 发生修改，控制器需要重启生效； 文件名称固定为 MC_CONFIG;
EtherCat 扩展 XML 文件	Ethercat PDO 配置文件； 一个工程中最多包含一个 EtherCat 扩展 XML 文件； 文件名称固定为 EC_EXTEND；
机器人程序	机器人程序包含有限的关于机器人操作的 BASIC 指令
机器人 BASIC 程序	可以包含所有的 BASIC 指令和机器人的操作指令。
CAM 库	编辑凸轮曲线文件； 一个工程中可包含多个凸轮文件；
HMI 设计	HMI 任务； 可执行； 一个工程中最多包含一个 HMI 任务；
HMI 界面	HMI 任务的子文件； 一个工程中可包含多个 HMI 界面；
IEC61131-1 任务	可执行文件； 文件名称用户自定义； 一个工程中可以包含多个 IEC61131-1 任务文件；
IEC61131-1 库	文件名称用户自定义； 一个工程中可以包含多个 IEC61131-1 库文件；
函数框图	函数框图文件； IEC61131-1 任务或者 IEC61131-1 库的子文件；
梯形图	梯形图文件； IEC61131-1 任务或者 IEC61131-1 库的子文件；
连续功能图	连续功能图文件； IEC61131-1 任务或者 IEC61131-1 库的子文件；
结构文本	结构文本文件； IEC61131-1 任务或者 IEC61131-1 库的子文件；

ESTUN & Trio

2.6.2 文件操作

开发者在同步模式下可以执行新建文件、删除文件、导入文件、导出文件、文件编辑等操作。

2.6.3 新建文件

1. 点击“菜单栏-文件/程序-新建”按钮，或者双击“控制器树-程序”按钮，弹出新建文件对话框文件；

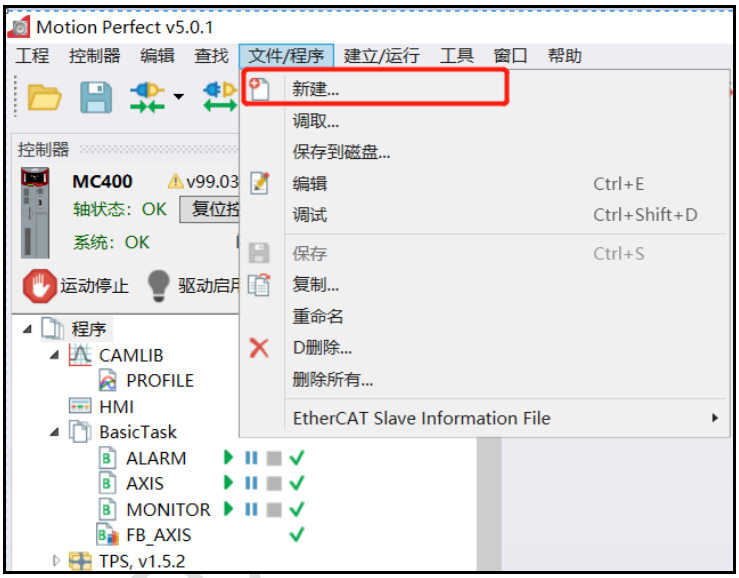


图 2-49 新建文件方法 1

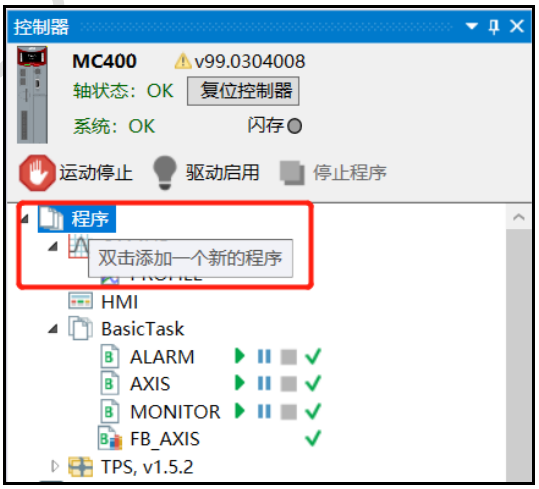


图 2-50 新建文件方法 2

2. 选中文件类型，编辑文件名称，设置文件组，点击确认按钮，生成新的文件；

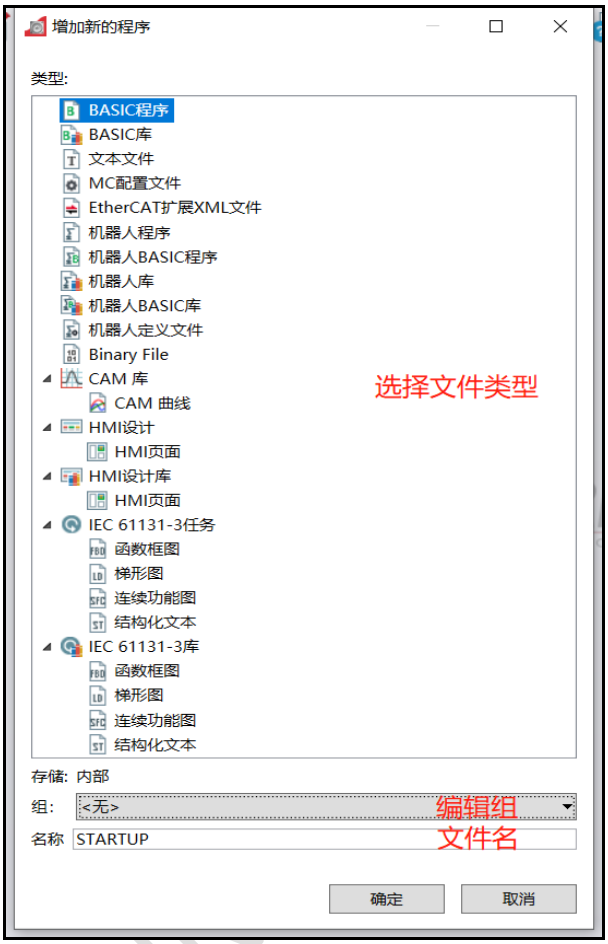


图 2-51 新文件命名

3. 完成程序的新建工作。

2.6.4 删除文件

1. 点击“菜单栏-文件/程序-D 删除”按钮，弹出文件删除对话框；



图 2-52 删除文件

2. 选择需要删除的文件，点击“删除”按钮；

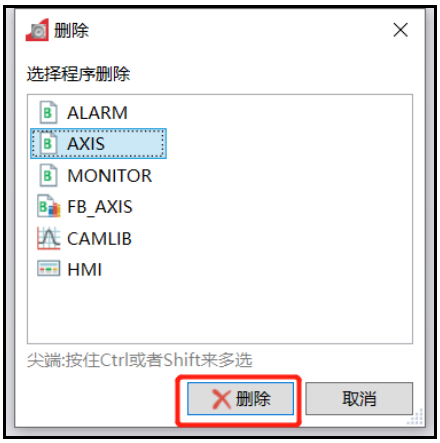


图 2-53 选择需要删除文件

3. 右键需要删除的程序文件，点击“删除”按钮，也可对单个文件进行删除处理；

2.6.5 导入文件

点击“菜单栏-文件/程序-调取”按钮，打开导入文件对话框，选择文件导入至当前工程。



图 2-54 导入文件

从磁盘中选择需要导入的文件，点击“打开”按钮。

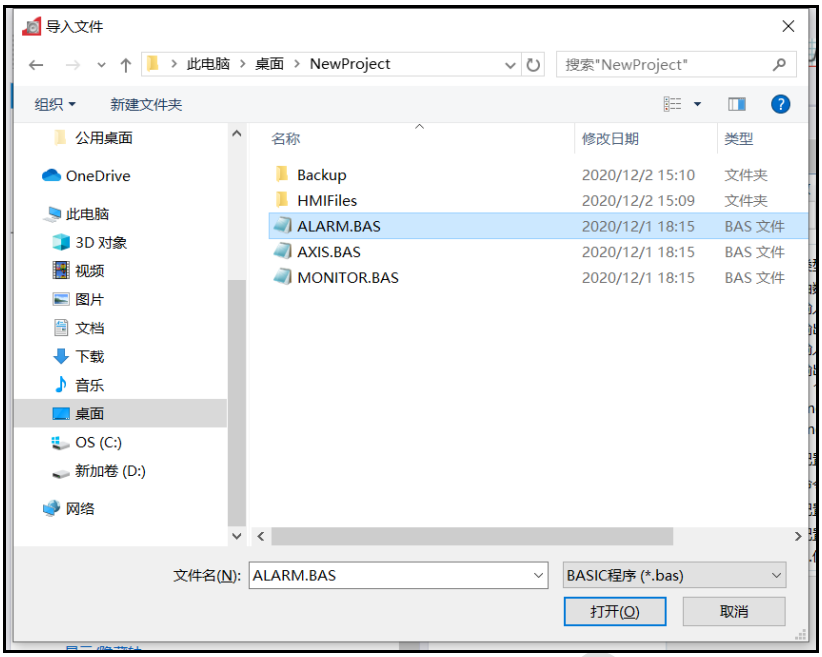


图 2-55 选择导入文件

2.6.6 导出文件

点击“菜单栏-文件/程序-保存到磁盘...”按钮，弹出保存程序对话框；



图 2-56 导出文件至磁盘

选择需要保存的文件，点击“保存...”按钮；



图 2-57 选择导出文件

选择磁盘存储路径，点击确认按钮。

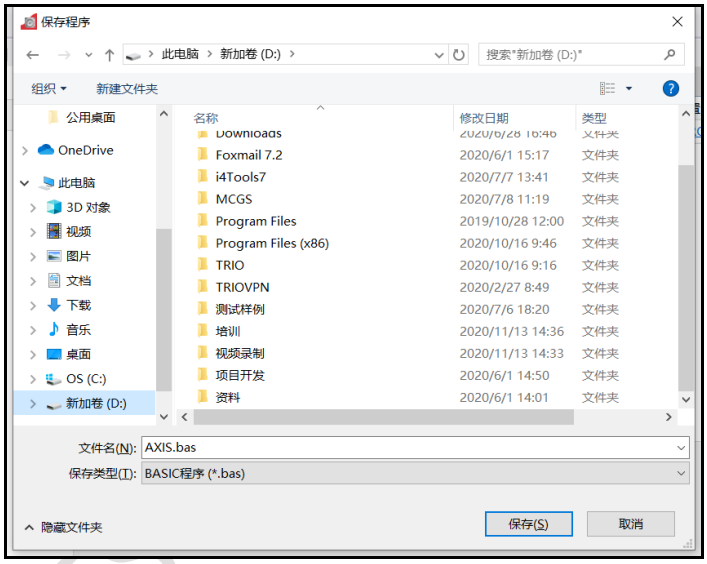


图 2-58 选择存储路径

2.7 工具

Motion Perfect 为用户提供了丰富的工具，简化调试操作。在同步模式和工具模式下可使用工具栏。

2.7.1 命令窗口

点击工具栏图标，可以选择打开窗口 0、5、6、7。

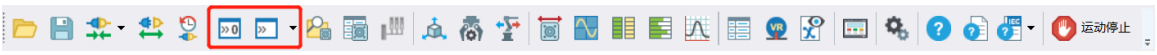


图 2-59 工具栏图标

窗口为用户与程序之间提供了关联，开发者可以通过执行 PRINT 指令，将信息打印至指定窗口。

窗口 0 支持用户输入 Basic 指令并执行，窗口 5、6、7 仅支持信息的显示。

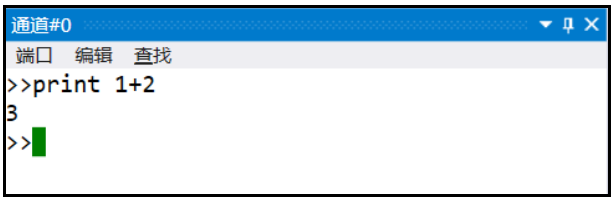


图 2-60 命令窗口

选项	描述
记录	将窗口信息存储为 log 文件至磁盘
脚本	执行简单的脚本命令
关闭	关闭命令窗口



2.7.1.1 设置日志文件

打开命令窗口，点击“端口-记录-设置日志文件”；

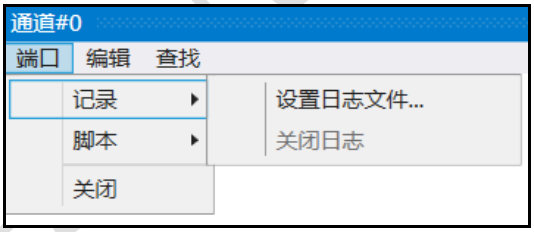


图 2-61 命令窗口记录选项

在磁盘中选择或者新建 LogFile 文件；

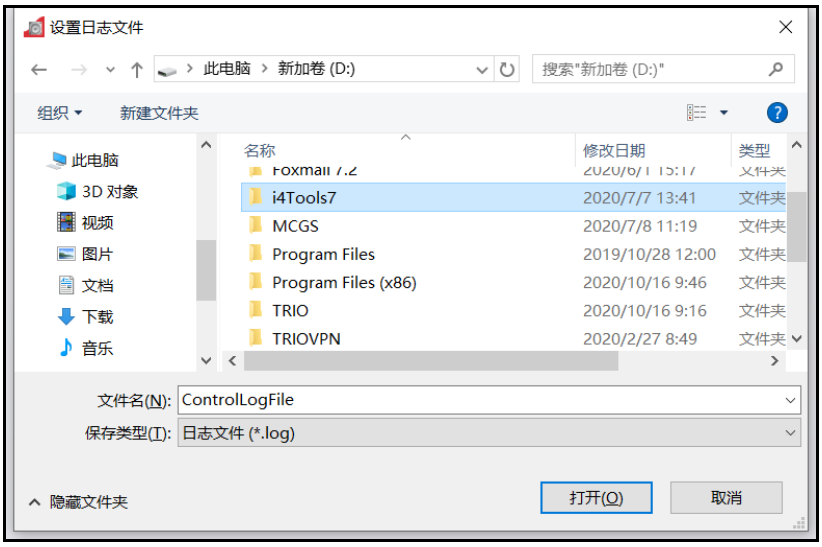


图 2-62 选择记录文件

点击“保存”按钮，设置 LOG 文件生效；

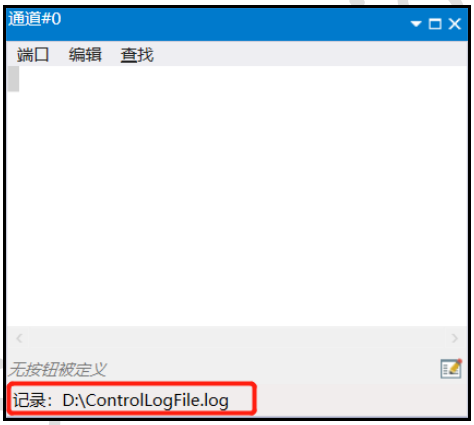


图 2-63 Log 文件设置生效

关闭窗口，log 文件设置失效。

2.7.1.2 设置窗体快捷命令

点击命令窗口 0 右下方“添加/移除按钮”，打开按钮编辑界面；

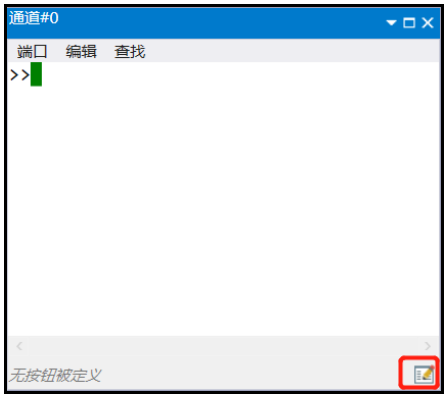


图 2-64 快捷按钮设置

编辑快捷指令，关联执行代码；

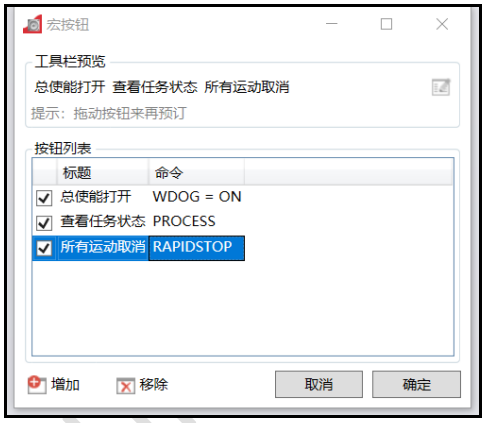


图 2-65 新增编辑按钮

点击按钮，便可在命令窗口 0 快捷执行代码；

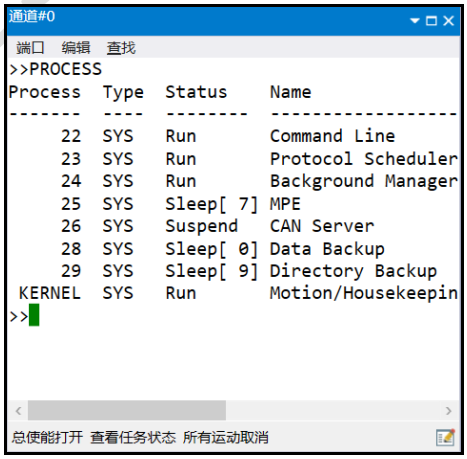


图 2-66 执行快捷按钮

2.7.2 轴参数

点击工具栏图标，可以打开轴参数界面。



图 2-67 工具栏图标—轴参数

用户使用轴参数界面，可以实现对轴参数的设置及状态的监控。



图 2-68 轴参数工具栏图标

图标	描述
	添加更多监控轴参数
	添加更多监控轴
	轴监控界面停止刷新/周期刷新
	轴监控界面立即刷新
	设置数据的显示格式（10 进制或者 16 进制）
	导出轴参数文件
	导入轴参数设置文件

2.7.2.1 添加监控轴

点击“轴参数-选择轴”按钮，弹出选择轴对话框；

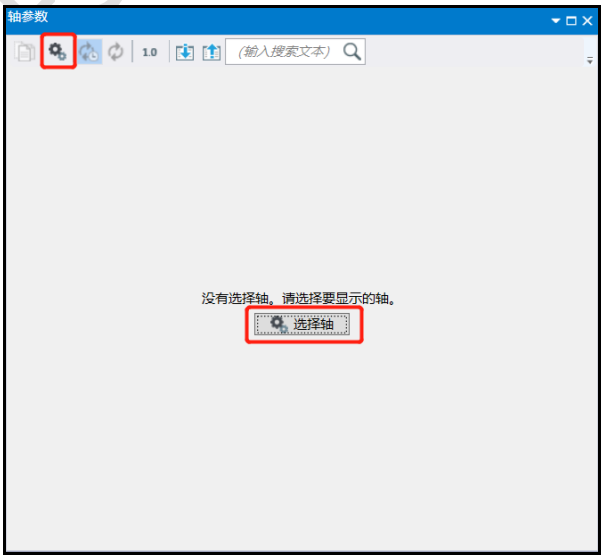


图 2-69 选择轴窗口

勾选需要监控的轴，点击“确认”按钮；

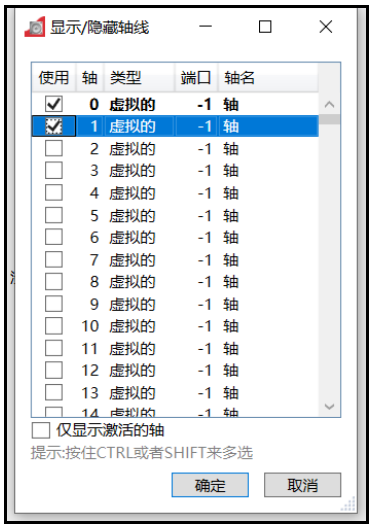


图 2-70 勾选监视轴

2.7.2.2 添加更多参数

点击“轴参数-显示所有参数”按钮，显示所有轴参数；

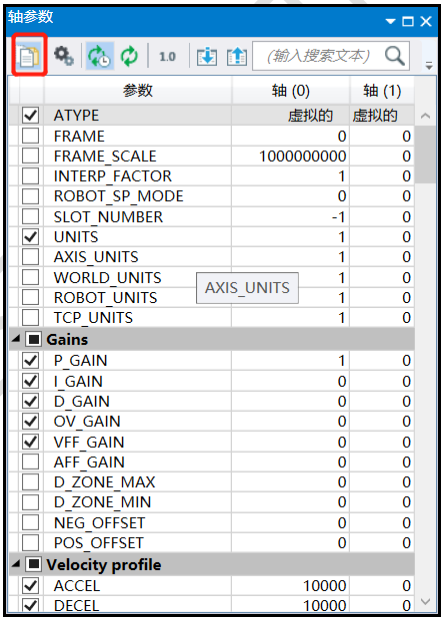


图 2-71 显示更多轴参数

勾选需要添加到常用监控的轴参数，点击“轴参数-显示所有参数”按钮；

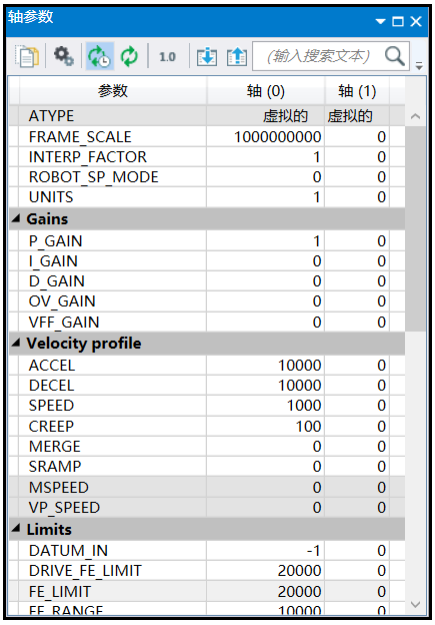


图 2-72 添加完成轴参数

2.7.3 数字量输入输出

点击工具栏图标，可以打开数字量输入输出界面。

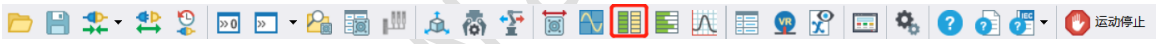


图 2-73 工具栏图标：数字量输入/输出

数字量输入输出工具用于显示 IN、OP 的状态，用户也可借助此工具快速切换输出点状态。

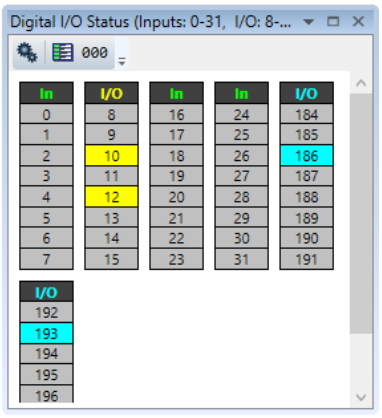


图 2-74 数字量输入/输出监控窗口



图 2-75 数字量输入输出的工具栏图标

图标	描述
	添加更多 IO 监控
	显示/隐藏 IO 描述
000	强制关闭所有输出
	导出 IO 描述文件
	导入 IO 描述文件

2.7.3.1 添加更多监控

点击“选择数字量 I/O”按钮，弹出 I/O 配置界面；

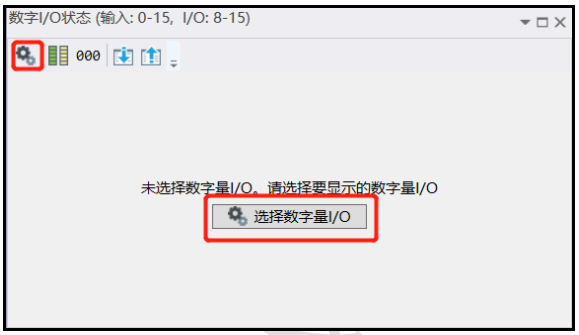


图 2-76 添加更多数字量 I/O

勾选需要监控的输入输出点；点击“列表”或者“设备”按钮，切换显示模式；

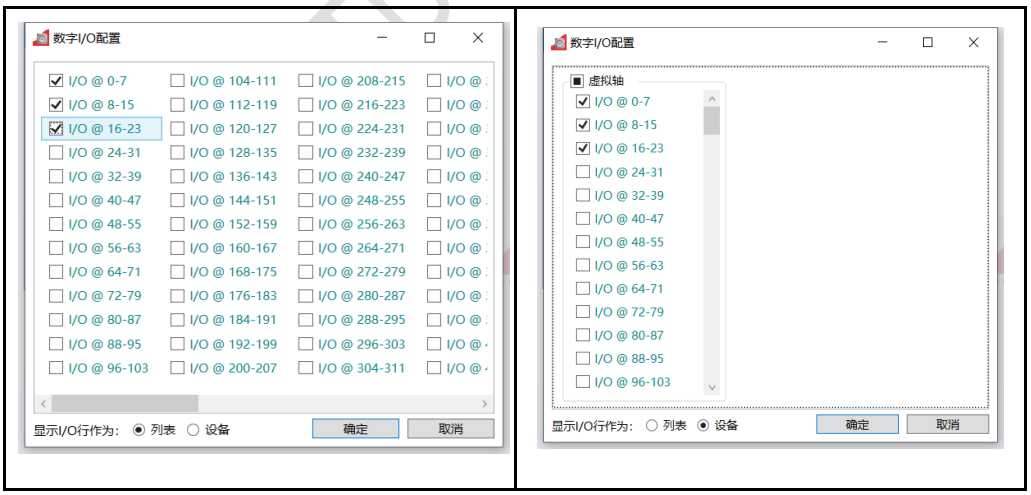


图 2-77 IO 显示模式

点击确定按钮，完成添加。

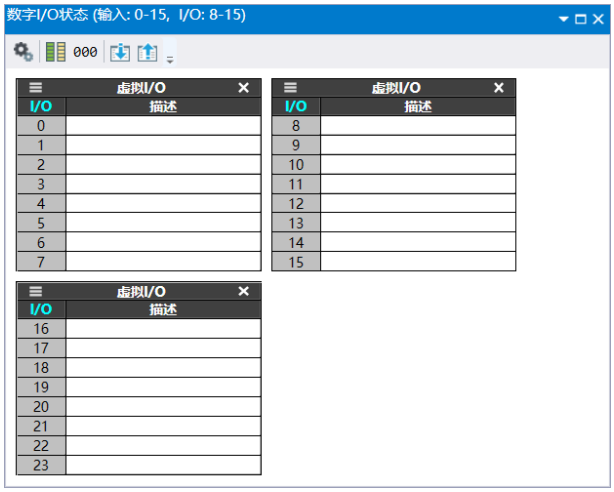


图 2-78 IO 监控画面

2.7.4 模拟量输入输出

点击工具栏图标，可以打开模拟量输入输出界面。

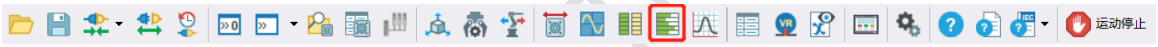


图 2-79 工具栏图标—模拟量输入输出

模拟量输入输出工具用于显示 AIN、AOUT 的状态。

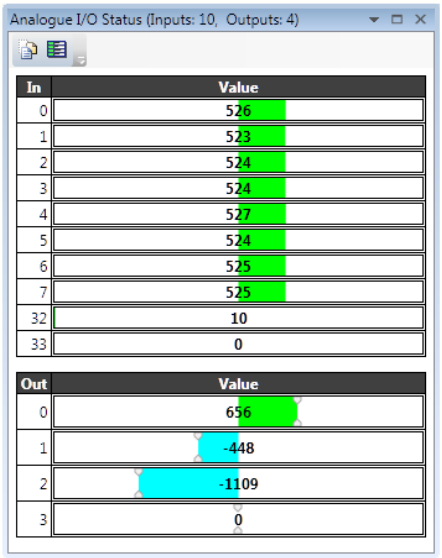


图 2-80 模拟量输入输出监控窗口

菜单栏：



图 2-81 模拟量输入输出监控窗口工具栏图标

图标	描述
	添加更多模拟量监控
	显示/隐藏 AIO 描述
	导出 AIO 描述文件
	导入 AIO 描述文件

2.7.5 TABLE 监控

点击工具栏图标，可以 TABLE 变量监控界面。

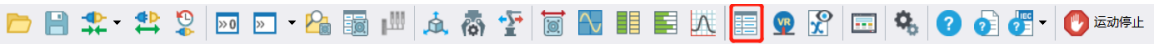


图 2-82 工具栏图标—TABLE 变量

查看 TABLE 窗口用于监控控制器 TABLE 变量的数值及便捷设置。

菜单栏：



图 2-83 TABLE 变量监控窗口菜单栏图标

图标	描述
	添加更多 TABLE 变量监控
	显示/隐藏 TABLE 变量描述
	停止刷新/周期刷新
	立即刷新
	10 进制显示/16 进制显示

2.7.5.1 设置一维监控范围

点击窗口工具栏“定义新的数值范围”按钮，弹出“定义新范围”窗口；

在“定义新范围”窗口，设置监控范围。

ESTUN & Trio

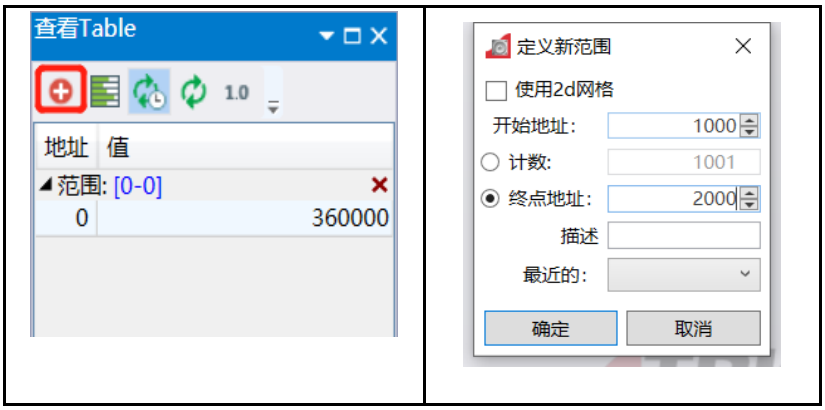


图 2-84 设置 TABLE 变量监控范围

如图 2-84 ，点击确认按钮，在 TABLE 监控窗口添加了 TABLE(1000)-TABLE(2000)的监控；

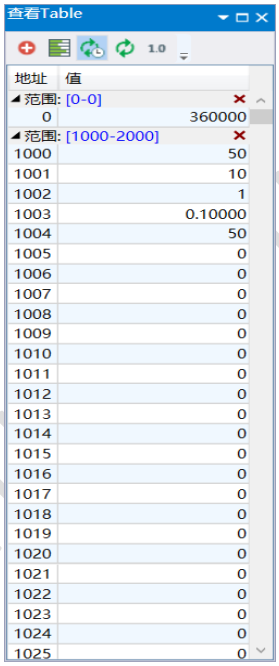


图 2-85 TABLE 一维监控界面

2.7.5.2 添加二维监控范围

点击窗口工具栏“定义新的数值范围”按钮，弹出“定义新范围”窗口；

在“定义新范围”窗口，勾选“使用 2D 网络”选项，设置监控范围；生成新的 2D 监控新窗口。

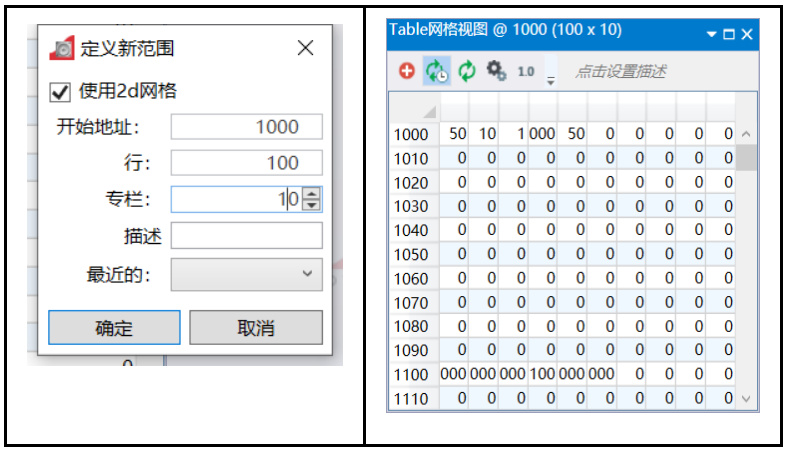


图 2-86 新增 TABLE 二维监控

2.7.6 VR 监控

点击工具栏图标，可以打开 VR 变量监控界面。

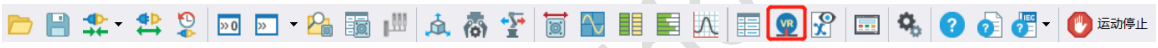


图 2-87 工具栏图标—VR 变量

查看 VR 窗口用于监控控制器 VR 变量的数值及便捷设置。



图 2-88 VR 变量监控窗口菜单栏

图标	描述
	添加更多 VR 变量监控
	显示/隐藏 VR 变量描述
	停止刷新/周期刷新
	立即刷新
1.0	10 进制显示/16 进制显示

2.7.7 变量监控

点击工具栏图标，可以打开变量监控界面。

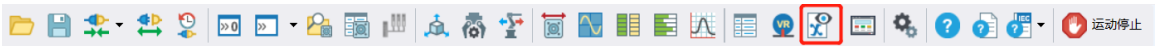


图 2-89 工具栏图标—变量监控

观察变量窗口可用于监控全局变量，也可用于监控正在运行的 BASIC 程序的局部变量。

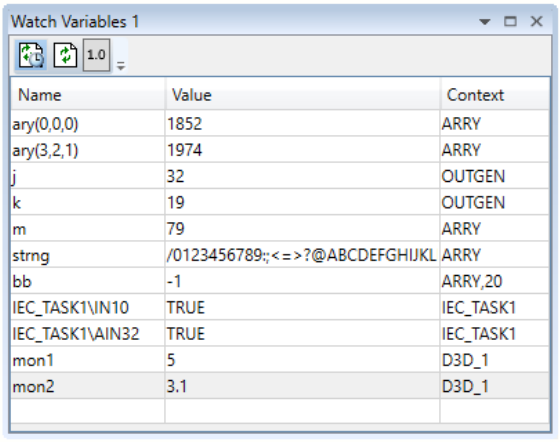


图 2-90 变量监控窗口



图 2-91 变量监控窗口菜单栏

图标	描述
	复制选中的变量
	粘贴选中的变量
	停止刷新/周期刷新
	立即刷新
1.0	10 进制显示/16 进制显示

2.7.7.1 添加局部变量监控

光标选中需要添加监控的变量；

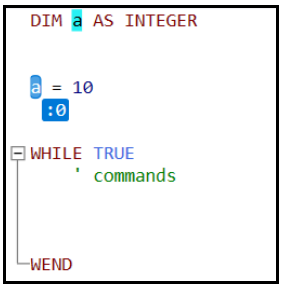


图 2-92 程序中添加监控变量

右键，弹出选项，点击“观看变量”按钮；

ESTUN & Trio

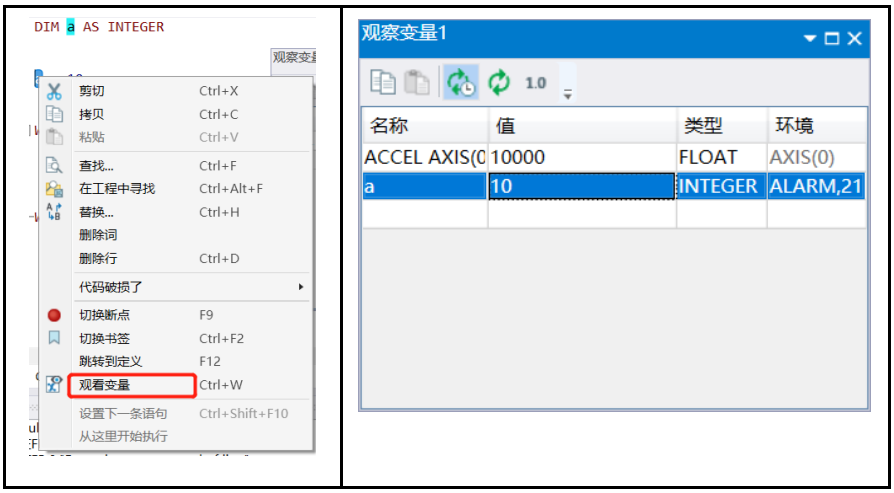


图 2-93 添加监控变量

2.7.8 轴点动

点击工具栏图标，可以 Jog 轴窗口。

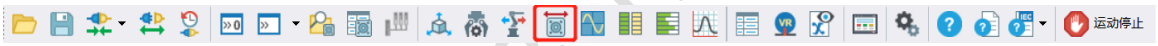


图 2-94 工具栏图标—JOG 轴

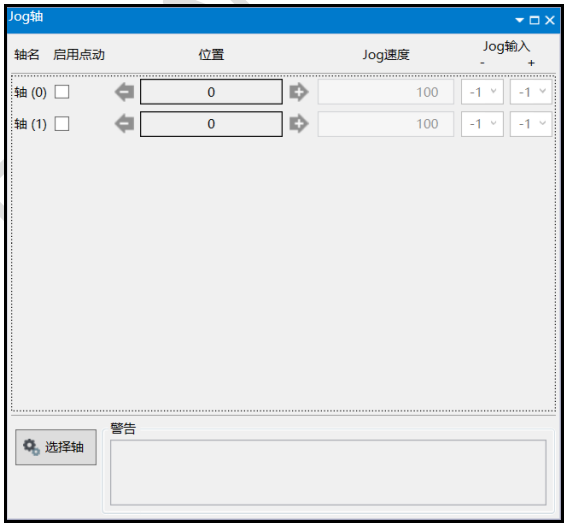


图 2-95 JOG 轴窗口

点击“选择轴”按钮，可以添加更多轴进行 JOG 操作；

使用 JOG 功能，需要 JOG 输入处轴关联虚拟 IO；

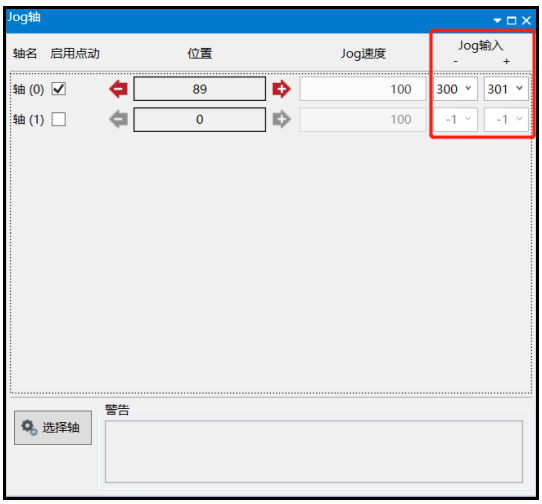


图 2-96 设置 JOG 轴参数

2.7.9 示波器

Motion Perfect 提供了示波器功能，为用户提供观察运动参数，采集过程数据。辅助用户进行程序调试。

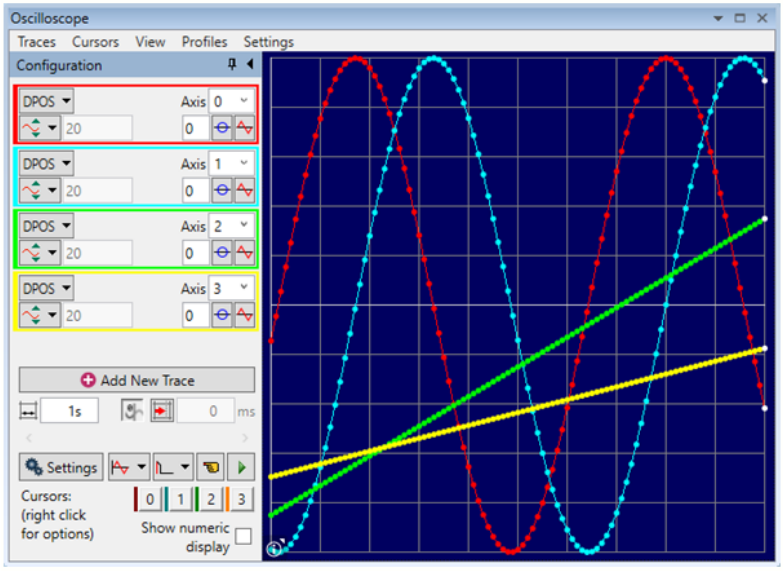


图 2-97 示波器监控界面

示波器可用于监控各种轴变量、系统变量、输入输出等，用于程序的测试及设备的监控。

- 示波器最大支持 8 路通道同时采集；
- 示波器最大支持 1s 中收集 1000 个信息值；
- 示波器支持手动触发采集、程序触发采集；

2.7.9.1 主菜单

示波器的主菜单包含轨迹、光标、查看、配置文件、设置等选项。

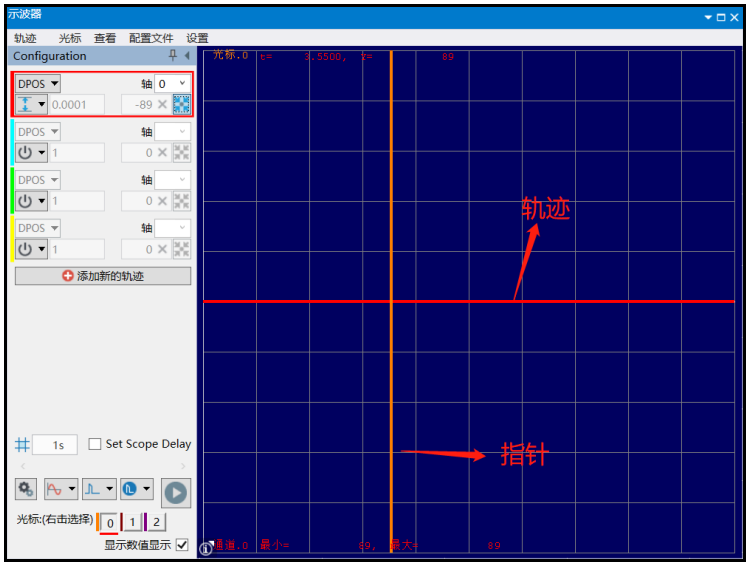


图 2-98 示波器窗口示意图

轨迹：监控数据生成的变化曲线	
选项	**描述**
添加新的轨迹	添加新的监控通道，最大至 8
加载	加载轨迹至指定通道（从.list 文件）
保存	保存指定通道轨迹至文件
重置	将轨迹重置
移除	移除通道
输出跟踪	导出轨迹至文件

光标	
选项	**描述**
添加新的指针	添加新的指针
MAP TO	将指针与轨迹进行关联
重置	将指针恢复至初始位置
移除	关闭指针显示
视野中心	将指针移动至视野中心

查看	
选项	**描述**
快照	示波器窗口截图

配置文件	
选项	**描述**
增加新的配置文件	将当前示波器窗体的配置存储，便于修改后快速恢复布局

设置	
选项	**描述**
输出设置	将示波器的设置导出成文件
导入设置	导入已有的示波器设置文件
重设设置	将示波器设置恢复成默认设置
设置	打开设置界面

2.7.9.2 通道设置

用户可根据需求，设置每个采集通道的采集对象、采集模式、轨迹显示单位、是否锁定、轨迹显示偏移。

1. 采集对象



用户可以通过下拉列表选择需要监视的对象，可选择对象包括：轴参数及轴状态、系统状态、输入输出、TABLE 连续数据区域；



通过通道选择，确定监控参数的轴号或者顺序号；

2. 采集模式



模式	描述
	通道不采集数据
	通道根据采集数据信息的大小，自动设置轨迹显示单位
	轨迹显示单位通过右侧进行设置
	冻结显示该通道的信息

ESTUN & Trio

3. 轨迹显示单位



自动测量模式下，该设定值跟随数据大小变化；手动测量模式下，该值可设置。

2.7.9.3 采集设置

用户可以设置采样参数，包括采样时基设置、采样时间偏移设置、显示模式设置、触发模式设置、手动/程序触发设置。

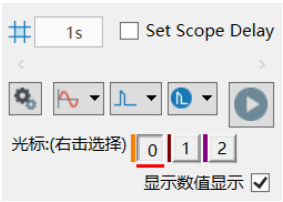
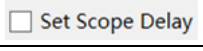


图 2-99 采集设置内容

模式	描述
	用户可以设置示波器单元格采集数据的时间
	程序触发模式下，设置提前或者滞后的开始采样时间
	打开示波器设置窗口
	选择不同的数据显示模式： Yt 模式：X 坐标为时间，Y 坐标为采集信息； XY 模式：X 坐标显示第一通道采集信息，Y 坐标显示第二通道采集信息； XY 模式（相同量纲） XYZ 模式 XYZD 模式 XYZUVW 模式
	单触发模式 重复触发模式 连续触发模式
	手动触发 程序触发
	开始/停止采集按钮

ESTUN e-Trio

模式	描述
光标:(右击选择) 0 1 2	指针的显示/关闭按钮

2.7.9.4 采集模式

示波器包含多种采集模式。

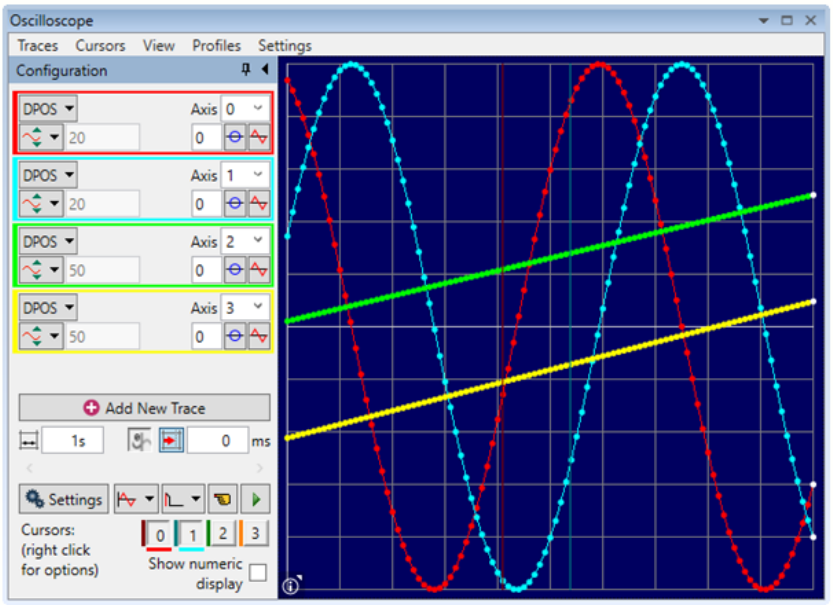


图 2-100 YT 模式

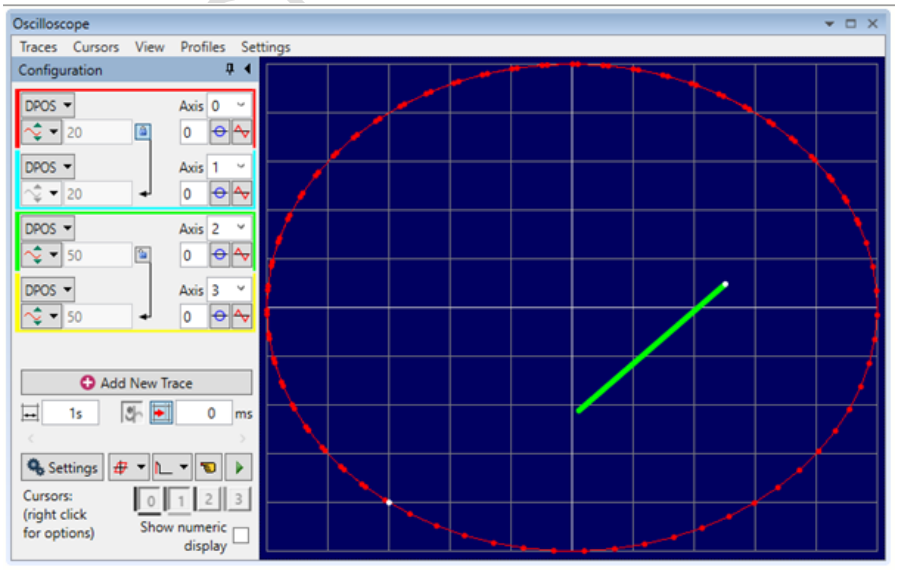


图 2-101 XY 模式

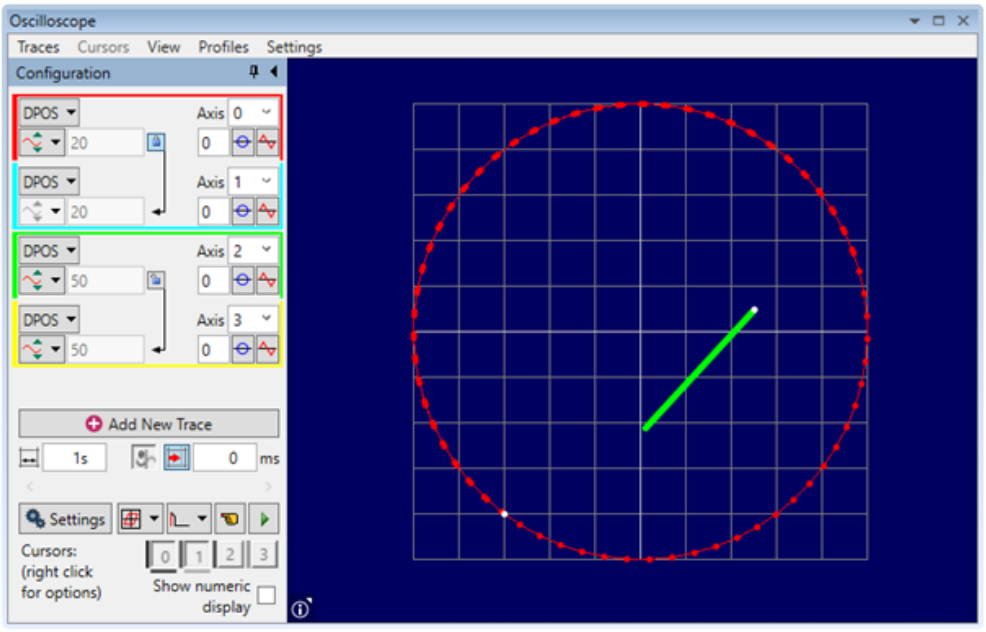


图 2-102 XY 模式（相同量纲）

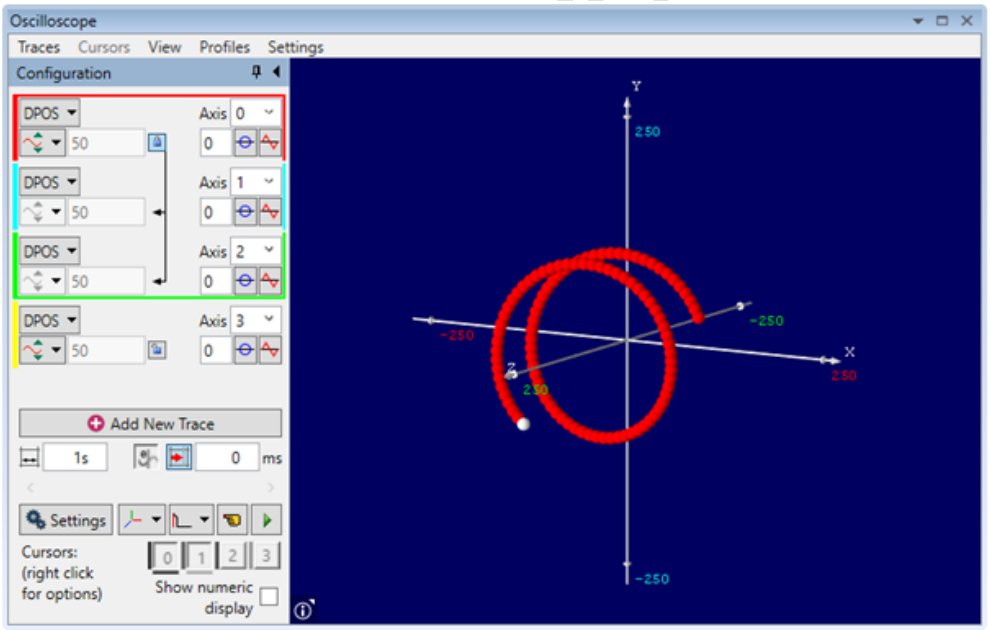


图 2-103 XYZ 模式

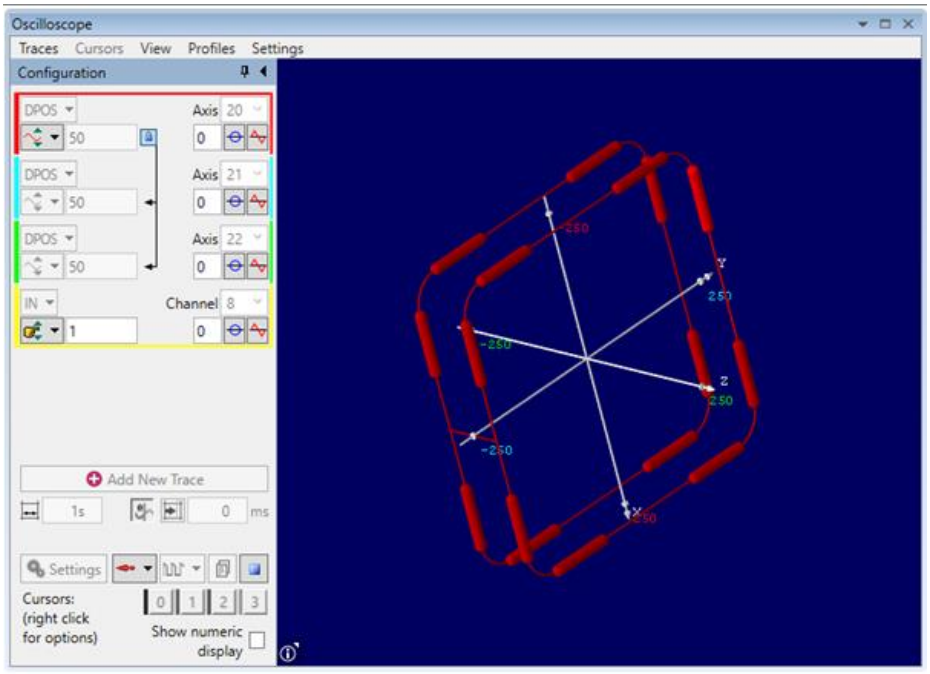


图 2-104 XYZD 模式

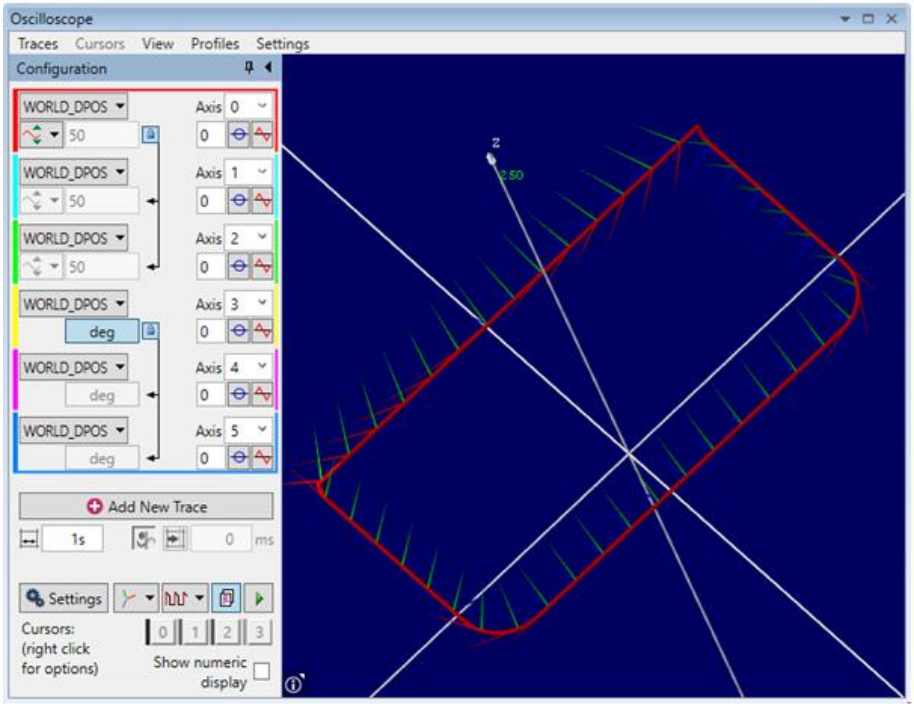


图 2-105 XYZUVW 模式

2.7.10智能轴驱动

参考章节 ETHERCAT。

2.8 控制器工具

2.8.1 功能码激活

用户在使用控制器时，如果需要新增功能（开通机器人软件包、轴数升级），则需要通过功能码激活功能。

获取功能码，查看控制器序列号，通过供应商或者厂家获取软件升级码；查询方式有：

1. 控制器树查询（控制器树窗口-配置-序列号）；

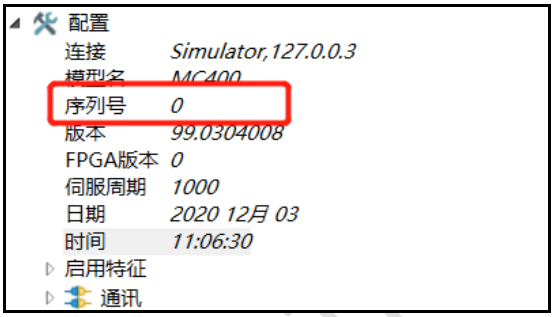


图 2-106 控制器树查询序列号

2. 通过系统参数（SERIAL_NUMBER）查询；

```
>>print serial_number
0
```

图 2-107 参数查询序列号

3. 通过控制器标签查询；



图 2-108 控制器标签查询序列号

打开功能码激活窗口；点击菜单栏“控制器-启用特征”按钮；



图 2-109 输入特征激活吗

1. 输入功能激活码；
2. 点击“确认”按钮，复位控制器生效。

2.8.2 SD 卡管理

用户可以通过内存卡管理窗口对 SD 卡的文件进行保存，读取等操作。

点击菜单栏“控制器-记忆卡...”，打开 SD 卡管理窗口。

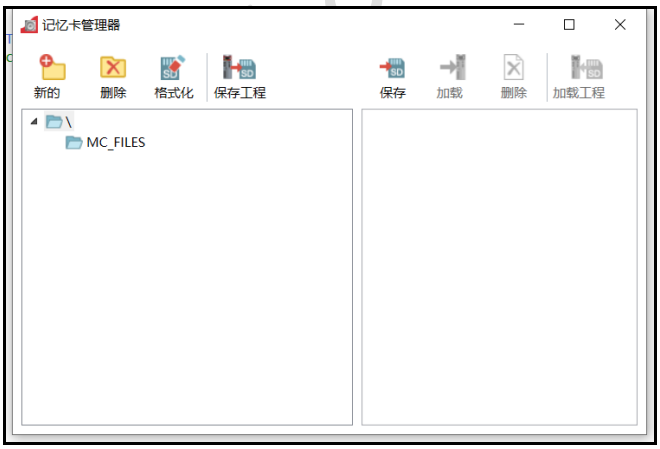


图 2-110 SD 卡管理窗口

如果未安装 SD 卡，弹出未安装提醒界面。



图 2-111 记忆卡提醒窗口

图标	描述
	SD 卡中创建新的文件夹
	删除指定的 SD 卡文件夹
	格式化 SD 卡
	将工程保存至 SD 卡
	保存指定文件存储至 SD 卡
	将 SD 卡中指定文件加载至控制器
	删除指定文件
	将 SD 卡中的工程加载至控制器

2.8.3 加载固件版本

控制器的软件固件版本可以通过 Motion Perfect V5 进行升级，新的固件版本可以为用户提供更多的指令集、控制功能。

使用者最好在供应商或者厂家的建议下，选择固件版本，进行升级。

加载新的固件版本会删除控制器内的所有程序文件，请做好工程的备份工作。（VR 变量数值不会被清除）

1. 查看当前控制器的固件版本；

4. 通过 MPV5-控制器树查阅；



图 2-112 控制器树查询固件版本

5. 通过系统参数 VERSION 获取控制器固件版本；

```
>>print version
99.0304
```

图 2-113 参数查询固件版本

2. 从 TRIO 官网下载控制器的最新固件版本、最新固件版本更新说明；

3. 更新固件版本

4. 点击菜单栏“控制器-加载固件…”按钮，打开加载固件对话框；

ESTUN & TRIO

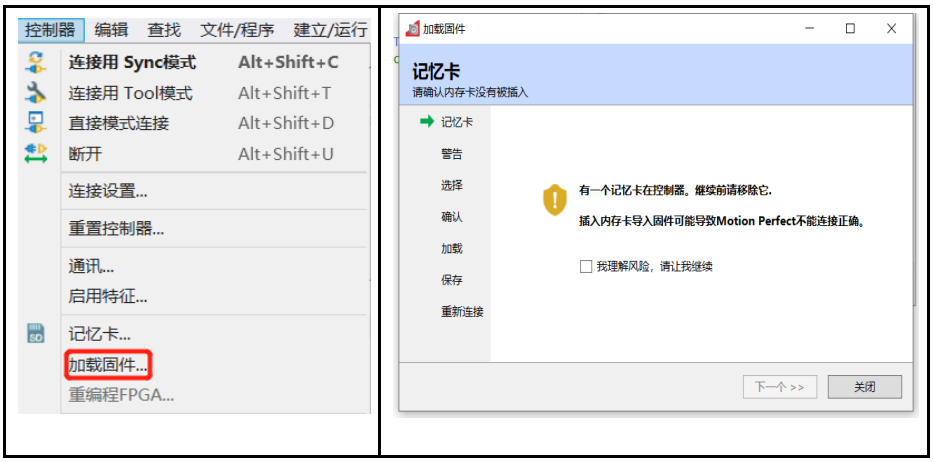


图 2-114 打开加载固件窗口

5. 如果控制器已插入 SD 卡，请拔出 SD 卡，点击下一步继续；

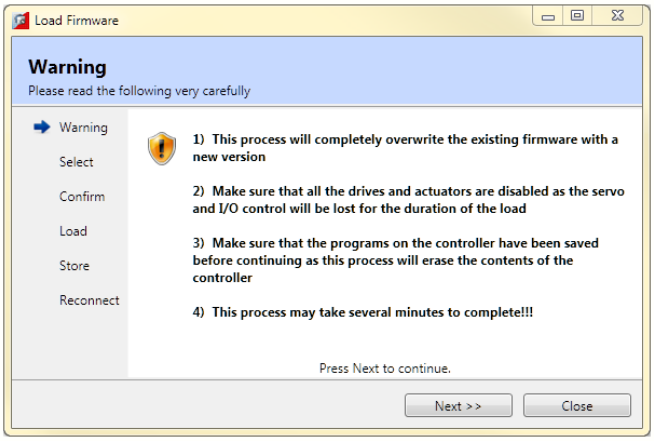


图 2-115 提示窗口

6. 点击“下一步”按钮，选择固件版本文件；

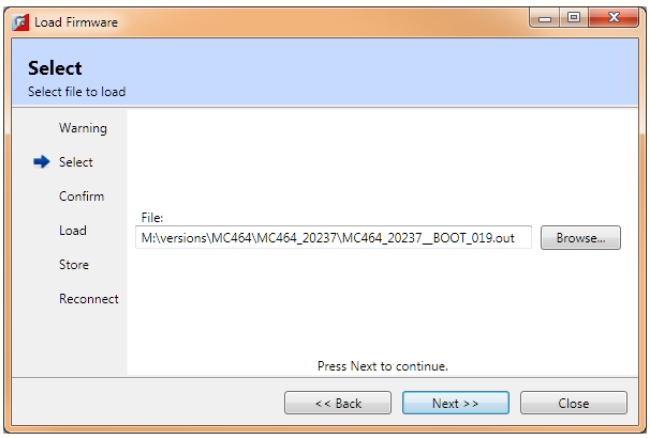


图 2-116 选择固件

7. 点击“下一步”；

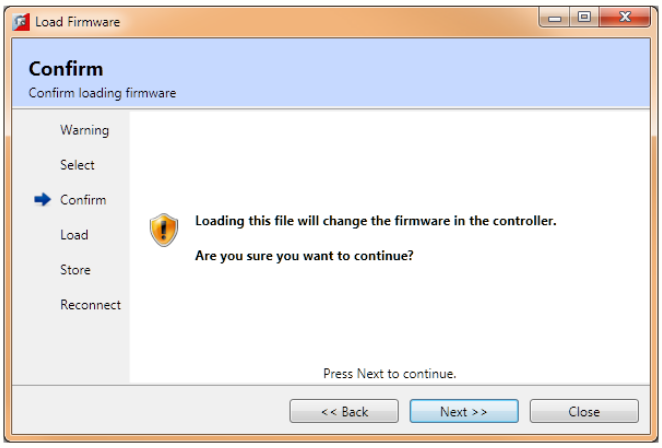


图 2-117 确认界面

8. 点击“下一步”，开始加载固件；

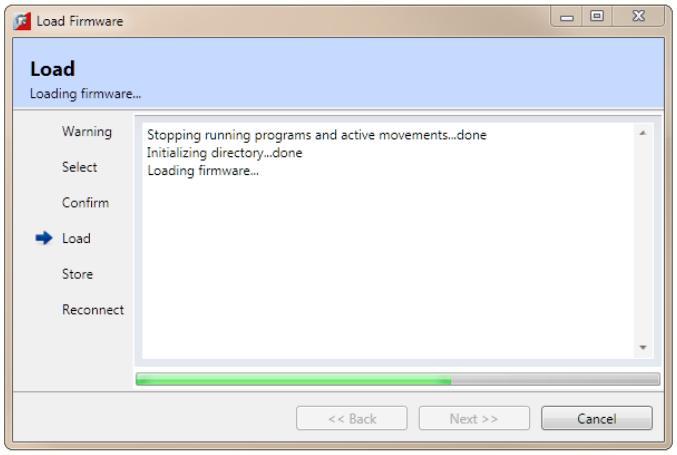


图 2-118 固件加载中

9. 等待加载完成，点击“下一步”按钮；

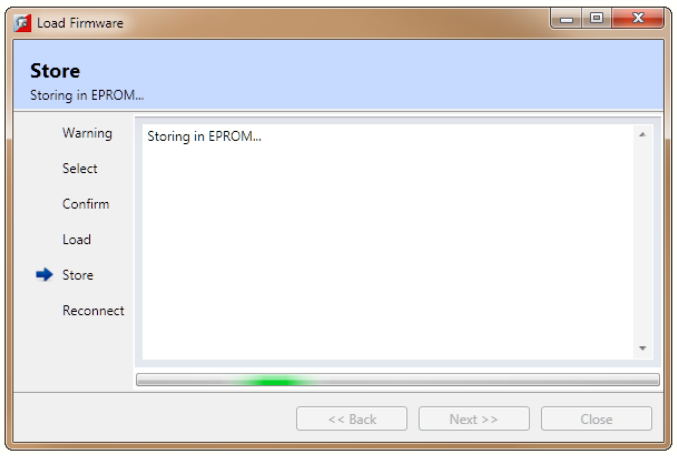


图 2-119 等待固件加载完成

10. 点击“下一步”，等待保存成功，控制器重新连接成功；

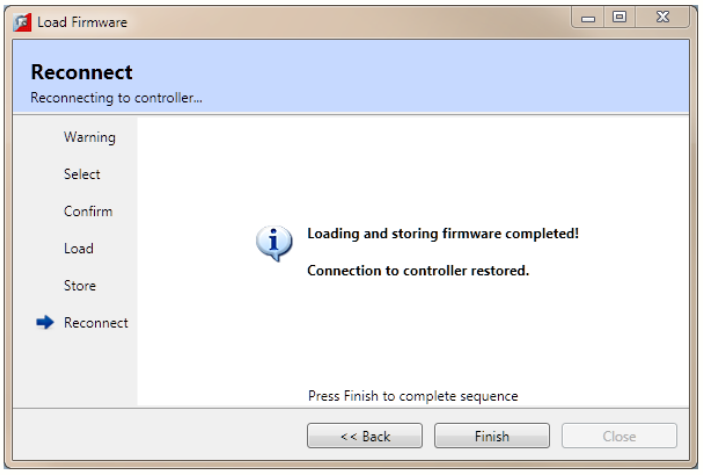


图 2-120 固件加载完成

11. 重新下载工程。

2.8.4 重编程 FPGA

固件版本程序内可能会包含不同的 FPGA 程序，以便支持不一样的控制功能。

用户可以通过 MVP5 修改控制器的 FPGA 程序，具体操作如下：

1. 点击菜单栏“控制器-重编程 FPGA...”按钮，打开对话框（部分控制器（NANO/MC400）不支持此功能）；

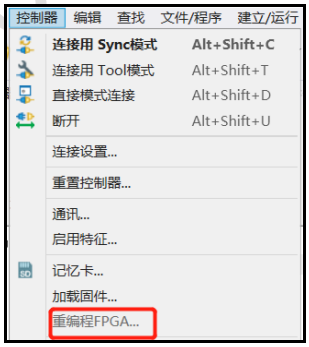


图 2-121 打开 FPGA 重编程界面

2. 选择需要修改到的 FPGA 程序，点击“下一步”按钮；

ESTUN & Trio

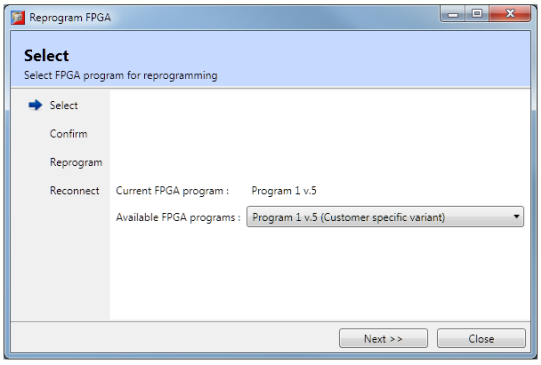


图 2-122 选择 FPGA 程序

3. 点击 “确认”按钮，开始更新；

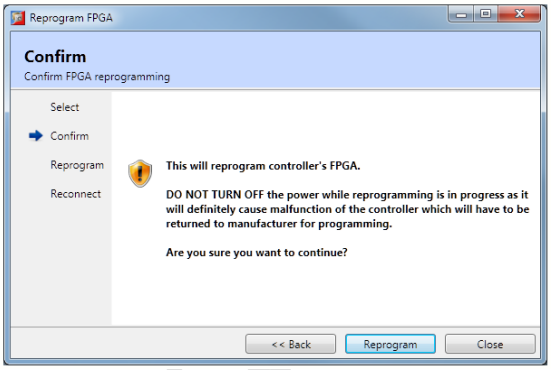


图 2-123 确认界面

4. 等待更新完成，点击“下一步”；

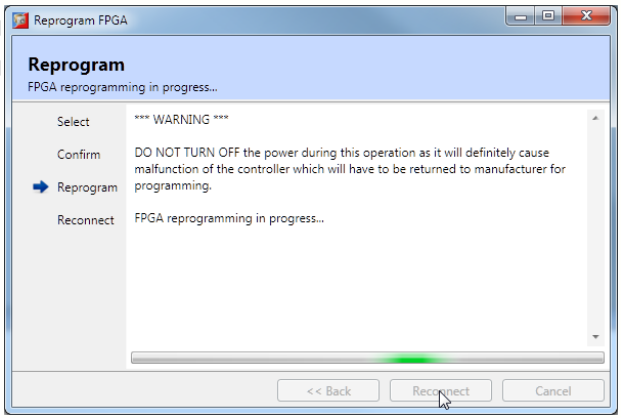


图 2-124 等待更新完成

5. 完成后将控制器重新上电，点击“重新连接”按钮；

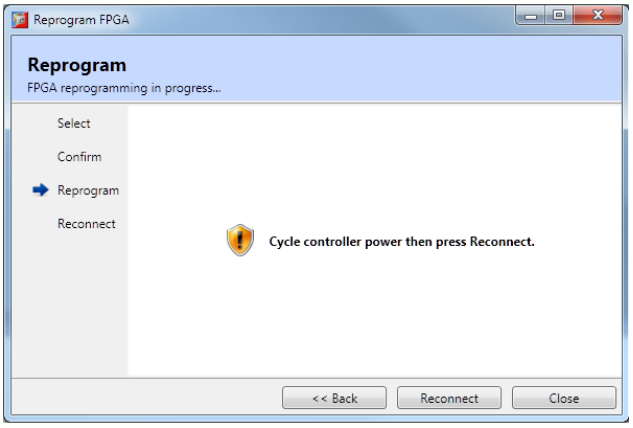


图 2-125 更新完成，重连控制器

6. 连接成功，完成 FPGA 的程序加载。

2.8.5 文件路径查看

用户可以单击菜单栏“控制器-D 目录...”按钮，打开文件路径查看对话框。

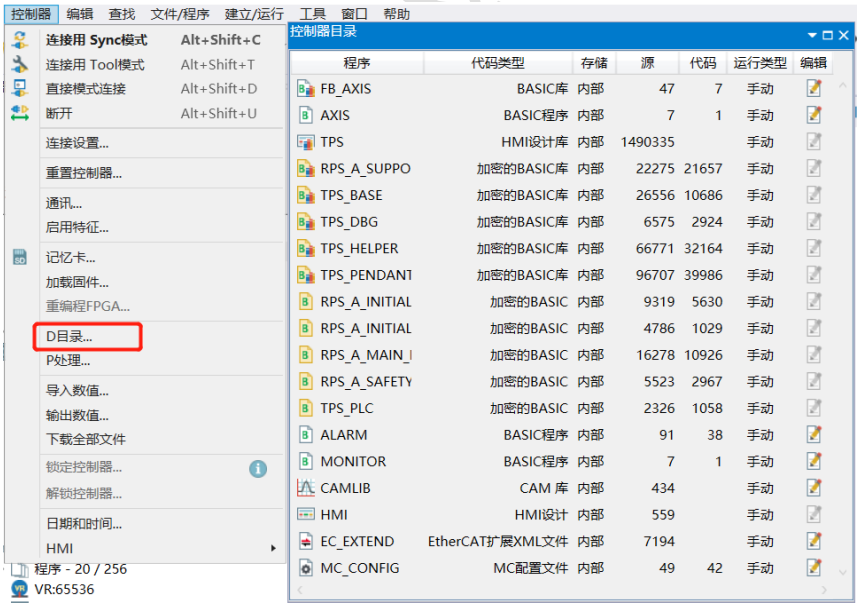


图 2-126 查看文件路径窗口

2.8.6 任务状态查看

参考 PROCESS。

2.8.7 数值导入/导出

开发者可以将 VR/TABLE 变量的数值导入至磁盘文件，也可以将磁盘文件中的信息加载至 VR/TABLE 变量。

1. 支持的文件格式

- BASIC 程序文件；
- .CSV 文件；
- .List 文件；

2. 导出操作

- 点击菜单栏“控制器-输出数值”按钮，弹出输出选项对话框；



图 2-127 打开输出数值窗口

- 选择需要导出的数据类型及范围；



图 2-128 选择需要导出的数值

- 点击保存按钮，选择存储文件类型及设置文件名；

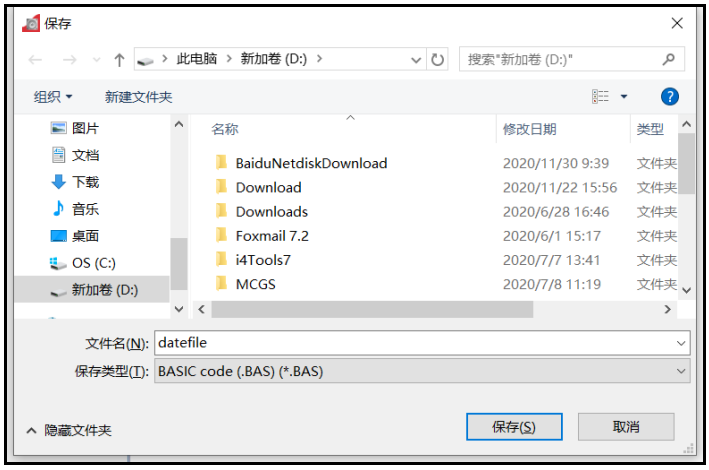


图 2-129 设置存储路径

- 点击“保存”按钮，完成操作。

3. 导入操作

- 点击菜单栏“控制器-导入数值”按钮，弹出输出选项对话框；



图 2-130 打开导入数值窗口

- 选择需要导入的文件，点击“打开”文件；

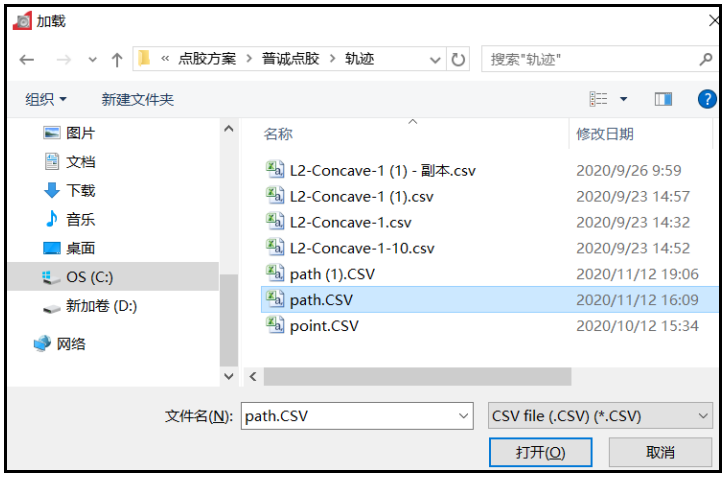


图 2-131 选择导入的文件

- 点击“加载”按钮，确认导入。



图 2-132 确认导入

2.8.8 控制器锁定/解锁

用户可以对控制器进行锁定操作，避免非开发人员进入同步模式，修改程序文件，对设备造成不必要的伤害。



图 2-133 控制器处于锁定状态

控制器如果锁定，用户需要解锁控制器，方可进入同步模式，进行程序文件编辑工作。

控制器的锁定有两种方式：

- 方式 1，点击菜单栏“控制器-锁定控制器”，弹出对话框；



图 2-134 控制器锁定对话框

- 方式 2，如果方式 1 无法完成锁定，可使用 LOCK 指令进行锁定；开发者退出至直接模式，打开窗口 0，执行锁定命令；（括号内为锁定密码示意）



图 2-135 使用指令锁定控制器

控制器的解锁也包含两种方式：

- 方式 1，点击菜单栏“控制器-解锁控制器”，弹出对话框，输入密码；

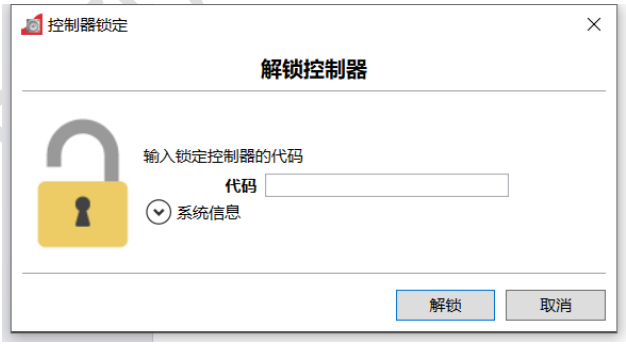


图 2-136 控制器解锁对话框

- 方式 2，直接模式下，在窗口 0 执行 UNLOCK 命令（括号内为解锁密码示意）。解锁密码为锁定时的锁定密码。



图 2-137 使用指令解锁控制器

2.8.9 控制器时钟设置

控制器出厂时间为 2000 年 1 月 1 日，用户可对控制器的时钟、日期进行设置。

点击菜单栏《控制器-日期和时钟》按钮，弹出时钟设置对话框。



图 2-138 控制器日期和时间设置界面

用户可选择设置当前时间，也可将控制器与 PC 时间进行同步。

2.8.10备份管理

开发者在进行工程编辑时，Mpv5 会对工程的文件进行自动备份操作，存储至工程文件夹中的 Backups 文件夹。



图 2-139 工程文件夹

用户也可以打开备份管理界面，查看当前备份的工程，或者执行恢复备份、新建备份等操作。

点击菜单栏“工程-备份...”，弹出备份管理器。

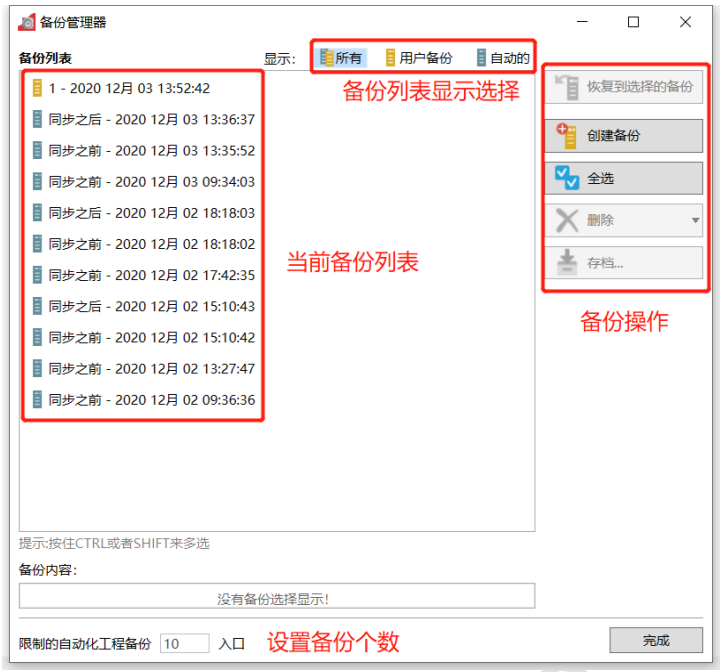


图 2-140 备份管理器窗口

2.9 Mpv5 常用设置

点击菜单栏“工具-选项”按钮，打开 MPV5 软件设置界面。

2.9.1 刷新率设置

设置轴参数监控、VR/TABLE 监控工具的刷新时间。

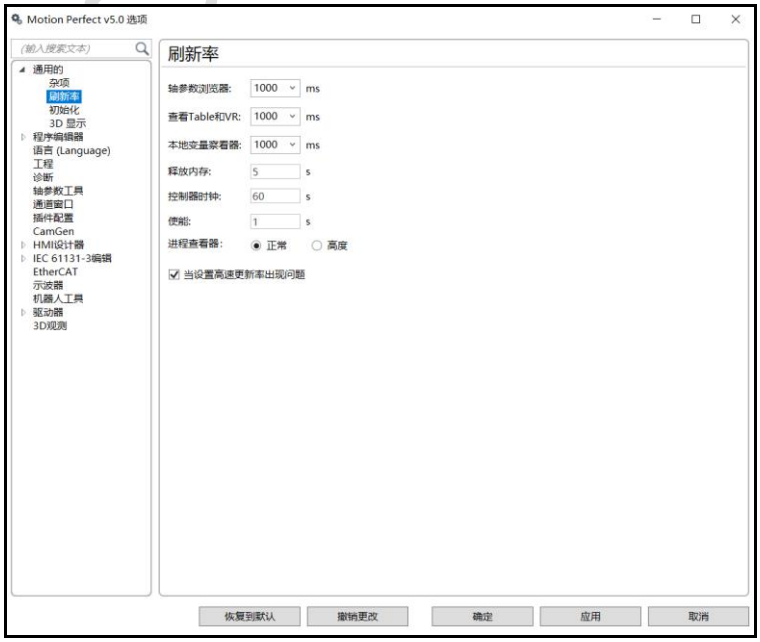


图 2-141 刷新率设置界面

2.9.2 初始化设置

设置 Mpv5 软件打开的相关操作。

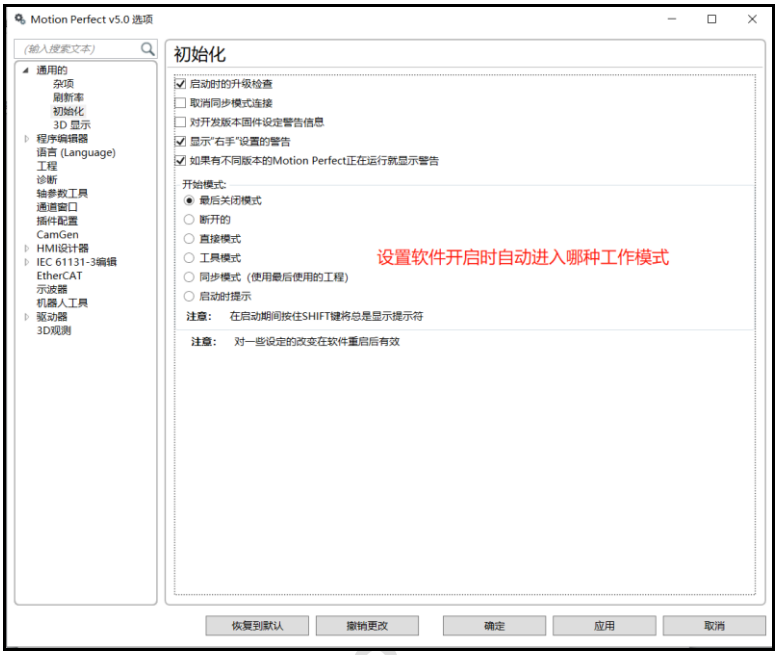


图 2-142 初始化设置界面

2.9.3 程序编辑设置

用户可设置 BASIC 文本程序的字体、字体大小、字体颜色、缩放设置等。

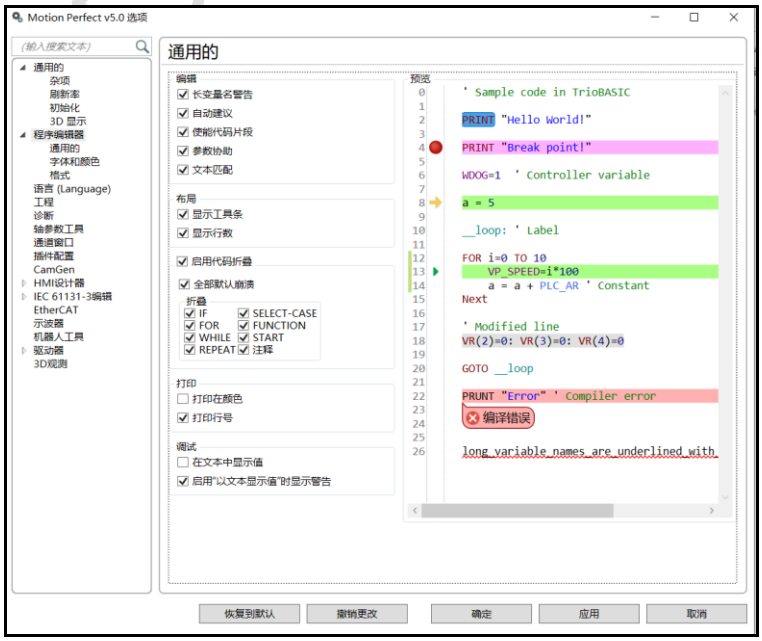


图 2-143 程序编辑设置界面

2.9.4 语言设置

用户可设置 MPV5 软件的环境语言。



图 2-144 语言设置界面

2.9.5 工程设置

用户可设置文本比对软件、工程自动备份最大个数。

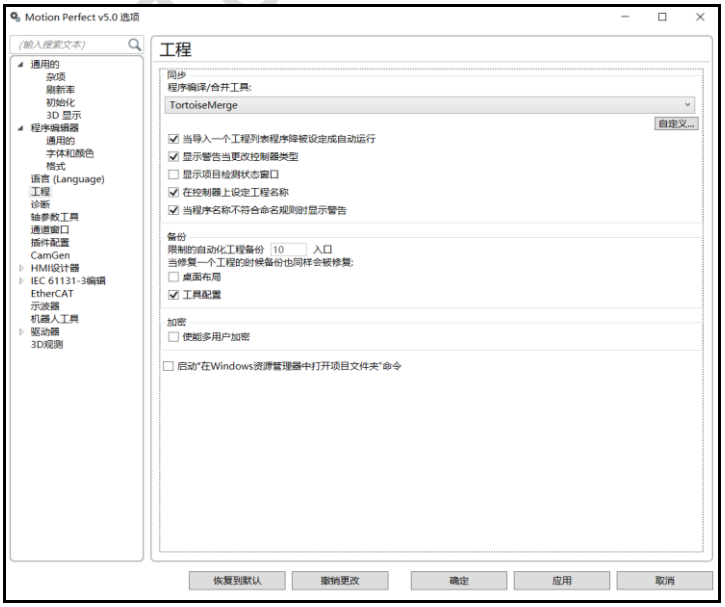


图 2-145 工程设置界面

2.10 系统参数设置（MC_CONFIG）

2.10.1 概述

MC_CONFIG 文件用于设置控制器的系统参数，参数设置完成后需要复位控制器生效。

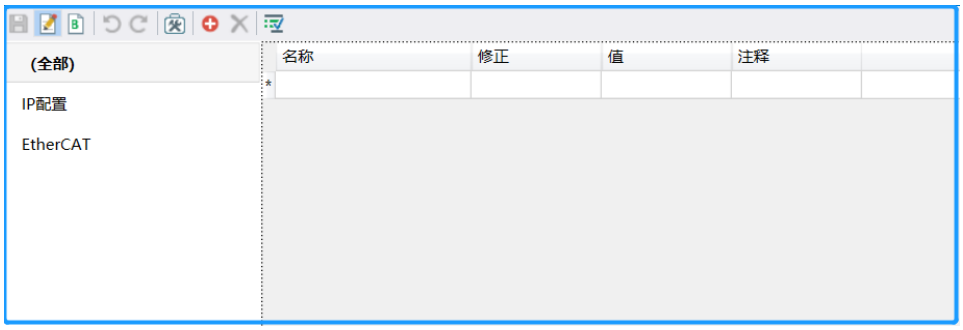


图 2-146 MC_CONFIG 界面

用户可在“IP 配置”中配置 ETHERNET 端口 IP 地址；

用户可在“EtherCAT”配置 EtherCAT 总线相关参数（轴地址设置、伺服周期设置等）；

用户可在“全部”里面添加其他配置；

2.10.2 IP 设置



图 2-147 IP 设置界面

用户可在 MC_CONFIG 文件“IP 配置”中设置控制器 IP 地址；

控制器 IP 默认地址为 192.168.0.250；

用户也可在 MC_CONFIG 文件中“全部”设置 IP 地址；

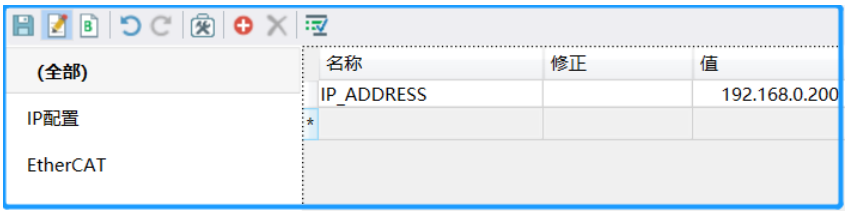


图 2-148 IP 地址设置

IP 相关的系统参数包括：

参数名称	含义
IP_ADDRESS	设置控制器的 IP 地址（例：192.168.0.200）
IP_GATEWAY	设置控制器的网关地址
IP_NETMASK	设置控制器的子网掩码
IP_PROTROL_CONFIG	参考 MODBUS TCP 设置

2.10.3 ETHERCAT 设置



图 2-149 ETHERCAT 设置界面

用户可在 MC_CONFIG 文件“EtherCAT”中设置 EtherCAT 总线相关参数；

EtherCAT 相关系统参数：

参数名称	含义
AUTO_ETHERCAT	设置控制器上电 EC 总线进入状态
SERVO_PERIOD	设置 EtherCAT 总线通讯周期
DRIVE_MODE	设置总线轴类型
DRIVE_PROFILE	设置总线轴 PDO 配置
NODE_AXIS	设置节点的轴地址
NODE_INDEX	设置节点轴 PDO 关联的 VR 地址
NODE_IO	设置节点的 IO 起始地址

2.10.4 轴地址设置

轴地址设置相关参数：

参数名称	含义
NODE_AXIS	设置节点的轴地址
AXIS_OFFSET	设置轴偏移，MC4N/MC6N 等控制器需要设置，将本体编码器轴进行偏移；参考 EC 初始化无法指定轴号；

2.10.5 轴类型设置

轴类型设置相关参数：

参数名称	含义
ATYPE	设置轴类型

2.10.6 IO 地址设置

IO 地址设置相关参数：

参数名称	含义
NODE_IO	设置 ETHERCAT 节点的 IO 起始地址

2.10.7 CPU 分配设置

CPU 资源分配相关系统参数：

参数名称	含义
SLOTn_TIME	设置不同类型任务 CPU 占比
SCHEDULE_TYPE	特殊设置，参考多任务章节

2.10.8 任务分配设置

任务分配设置相关系统参数：

参数名称	含义
REMOTE_PROC	设置远程程序（基于 Active/Dll 链接）任务号
HMI_PROC	设置 Trio Hmi 程序运行任务号
PCMACT_PROC	设置基于 API 远程程序运行任务号

2.10.9 编译设置

编译设置相关系统参数：

参数名称	含义
COMPILE_MODE	编译时检查变量是否定义

ESTUN Trio

2.11 程序的运行管理

Motion Perfect 的工程可以在 Simulator 或实际的控制器运行/调试。由于应用程序采用 Trio 的多任务机制进行管理，可实现多个进程（Process）/任务并行执行。在 Trio BASIC 的编程环境中，单个 BASIC 主程序可作为一个任务/进程执行；在 IEC61131-3 的编程环境中，每个 IEC 任务可作为一个任务/进程进行管理。

每个工程在实际应用时，需要设置一个自启动任务，然后在启动起来的程序中，进行任务的运行管理（运行任务、停止任务）。根据各个任务的运行需要，设置的进程号，实现快速任务和慢速任务的设置。

ESTUN & Trio

Trio BASIC 篇

3 Trio BASIC 编程基础

Trio BASIC 是运行于 Trio 可编程运动控制器的多任务编程语言。 语法类似于其他 BASIC 家族语言。 用户通过运行在 Microsoft Windows 操作系统的 Motion Perfect 开发和测试应用程序，协调所有所需的运动和机器功能。 MotionPerfect 提供所有编辑和调试功能，支持使用 Trio BASIC 编写和调试应用程序。其特点如下：

- 简易的 BASIC 语言，便于独立的机器编程
- 完美集成于 Motion Perfect 应用开发软件
- 全面的多轴运动控制功能
- 程序的多任务处理，有益于提升软件结构和维护便利性
- 支持传统的伺服或步进轴以及现场总线 (SERCOS、EtherCAT 等) 轴的控制
- 支持全面的多轴协调运动、单轴运动指令类型。包括直线、圆弧和螺旋线插补，电子凸轮、电子齿轮等主从轴运动。
- 高精度数值表示（最多 64 位），支持位操作
- 支持硬件位置捕获
- 支持高速输出、位置比较输出

Trio BASIC 有 300 多个命令，旨在使编程运动功能快速和简单。

ESTUN & Trio

3.1 局部变量

3.1.1 描述

局部变量的作用范围为单个任务，因此在不同任务定义相同名称的局部变量，其所代表的含义不一致。

局部变量的类型包含：布尔型、整型、浮点型、字符串型。

声明-DIM

指令语法	DIM name AS type DIM name AS FLOAT [(size1 [,size2 [,size3]])] DIM name AS INTEGER [(size1 [,size2 [,size3]])] DIM name AS STRING(length)

功能描述	变量的声明 **BOOLEAN** 布尔型(TRUE or FALSE) **INTEGER** 64 位整型 **FLOAT** 64 位浮点 **STRING** 文本型
示例	DIM machine_ok AS BOOLEAN DIM position AS FLOAT(10) DIM i,j,k AS INTEGER DIM character AS INTEGER DIM message AS STRING(200) DIM me AS INTEGER (20 , 20 , 20)
备注	局部变量的声明为可选的，在默认情况下，COMPIL_MODE 设定值为 0，局部变量可在未定义的情况下直接使用。为减小程序的编写错误，可在 MC_CONFIG 文件中将 COMPIL_MODE 设定为 1，则所有的局部变量均需要定义后使用，否则编译错误； 默认情况下，如果一个局部变量未使用 DIM 语句进行定义，该变量的数据类型为 64 位浮点型；如果该变量未赋值，其值为 0； 如果将浮点数赋值给整型变量，则会截取浮点数的整数部分；

编译设置-COMPILE_MODE

功能描述	系统参数； **设定值** / **含义** 0：BASIC 文件中变量未使用 DIM、GLOBAL、CONSTANT、DEFCONST 进行定义，编译不报错，变量类型为 64 位浮点型； 1：BASIC 文件中变量未使用 DIM、GLOBAL、CONSTANT、DEFCONST 进行定义，编译报错； MC_CONFIG 文件中设置。

3.1.2 字符串变量

功能描述	字符串变量是局部变量的类型之一。 这意味着在不同任务定义相同名称的字符串变量，其所代表的含义不一致。
定义	字符串变量，使用 DIM 语句进行定义，用户需要设置字符串的最大长度；

ESTUN & Trio

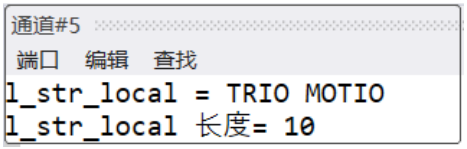
DIM string_local AS STRING(100)

表示定义变量名称为 string_local 的字符串变量，最大长度为 100 个字符；

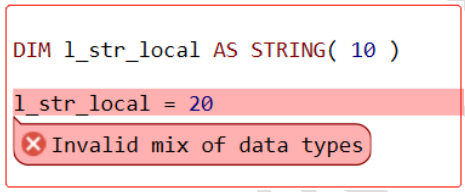
当给字符串赋值超过定义的长度时，编译运行不会报错，会自动截取最大长度对应的字符串赋值给变量；

```
DIM l_str_local AS STRING( 10 )
l_str_local = "TRIO MOTION TECHNOLOOY"
'窗口 5 打印相关信息(字符串内容及长度)
PRINT #5, "l_str_local = "; l_str_local
PRINT #5, "l_str_local  长度= "; LEN( lstr_local )
```

运行结果：



不可以将数值赋值给字符串变量，编译时报错；



可以通过 VR 变量在不同任务中传输字符串信息；

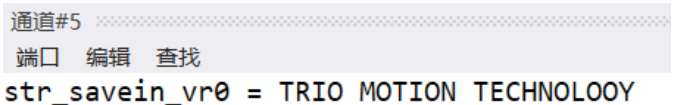
PROG1:

```
VR( 0 ) = "TRIO MOTION TECHNOLOOY"
```

PROG2:

```
DIM str_savein_vr0 AS STRING( 100 )
str_savein_vr0 = VRSTRING( 0 )
PRINT #5, "str_savein_vr0 = "; str_savein_vr0
```

运行结果：



<table>
<tr><td>相关操作函数</td><td>
<table>
<tr><td>ASC</td><td>返回字符对应的 ASCII 码</td></tr>
<tr><td>CHR</td><td>返回 ASCII 码对应的字符</td></tr>
<tr><td>INSTR</td><td>返回查询字符在字符串中的位置</td></tr>
<tr><td>LCASE</td><td>返回字符串的小写格式</td></tr>
<tr><td>LEFT</td><td>返回字符串指定数字左边的字符</td></tr>
<tr><td>LEN</td><td>返回字符串的长度</td></tr>
<tr><td>MID</td><td>返回字符串指定起始位置及指定长度的字符</td></tr>
<tr><td>RIGHT</td><td>返回字符串指定数字右边的字符</td></tr>
<tr><td>STR</td><td>将数值转换成字符串</td></tr>
<tr><td>UCASE</td><td>返回字符串的大写格式</td></tr>
<tr><td>VAL</td><td>将字符串转换成数值</td></tr>
<tr><td>VRSTRING</td><td>从 VR 区域读取连续字符串</td></tr>
</table>
</td></tr>
<tr><td>应用场合</td><td>字符串变量一般用于 TCP 通讯。与第三方设备进行通讯，可以按照相互约定的协议，发送及接收字符串。</td></tr>
</table>

3.1.3 数组变量

<table>
<tr><td>功能描述</td><td>控制器的 4 种变量都可以定义为数组变量。数组变量的最大维度为 3 维，每一维的最大长度为 1024。</td></tr>
<tr><td>声明</td><td>数组变量需要使用 DIM 语句进行定义方可使用；
DIM local_bool AS BOOLEAN(50)
DIM local_int AS INTEGER(50, 50)
DIM local_float AS FLOAT(50, 50, 50)
DIM local_str AS STRING(500)</td></tr>
<tr><td>定义规则</td><td>1) 4 种类型变量均可以定义为 3 维数组变量；
2) 每一维的最大长度为 1024，如果定义时超过 1024，编译出错；
3) 字符串变量数组括号内第一个数值表示字符串变量的最大长度；</td></tr>
<tr><td>赋值操作</td><td>1) 方法 1-单个赋值
DIM a AS INTEGER(10)
a(0) = 10
a(1) = 20
a(2) = 30
DIM b AS INTEGER(10, 10)
b(0, 0) = 10.3</td></tr>
</table>

ESTUN & Trio

```
b(0, 1) = 20.9
b(1, 0) = 30
b(1, 1) = 30.5
DIM c AS STRING(20, 10, 10)
c(0, 0) = "aaa"
c(0, 1) = "bbb"
```

2) 方法 2-批量赋值

```
DIM a AS INTEGER(10)
a = 10, 20, 30
DIM b AS INTEGER(10, 10)
b(0) = 10.3, 20.9
b(1) = 30, 30.5
DIM c AS STRING(20, 10, 10)
c(0) = "aaa", "bbb"

PRINT "a(0)="; a(0)
PRINT "a(1)="; a(1)
PRINT "b(0,0)="; b(0, 0)
PRINT "b(0,1)="; b(0, 1)
PRINT "b(1,0)="; b(1, 0)
PRINT "b(1,1)="; b(1, 1)
PRINT "c(0,0)="; c(0, 0)
PRINT "c(0,1)="; c(0, 1)
```

运行结果：



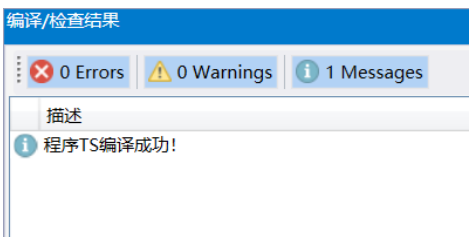
使用注意

数组索引越界，编译时不报错，运行时报错；

错误写法：

```
DIM a AS INTEGER( 10 )
a( 10 ) = 100
```

编译结果：



ESTUN & Trio

	运行结果： DIM a AS INTEGER(10) a(10) = 100 Parameter out of range
应用场合	1) BASIC 功能块的多次封装； 2) BASIC 功能块的多返回值功能实现； 3) 简化相同功能变量的重复定义；

3.1.4 作用域详述

本节通过几个例子描述局部变量作用域。

1) 两个任务定义相同的变量，含义不一样；

PROG1:

```
DIM a AS INTEGER

WHILE TRUE
  a = 100
  PRINT #5, "PROG1[A]="; a
  WA( 50000 )
WEND
```

PROG2:

```
DIM a AS INTEGER
RUN "PROG1" '运行 PROG1 程序
WHILE TRUE
  PRINT #5, "PROG2[A]="; a
  WA( 50000 )
WEND
```

运行结果：

通道#5
端口 编辑 查找
PROG1[A]=100
PROG2[A]=0

结果分析：

在任务 PROG1 中将变量 a 赋值为 100，不会影响 PROG2 中变量 a 的大小。

ESTUN & Trio

2) 使用 INCLUDE 指令，两个任务中包含相同变量定义文件，在不同任务中含义同样不一样；

PROG1:

```
INCLUDE "VAR"

WHILE TRUE
  a = 100
  PRINT #5, "PROG1[A]="; a
  WA( 50000 )
WEND
```

PROG2:

```
INCLUDE "VAR"

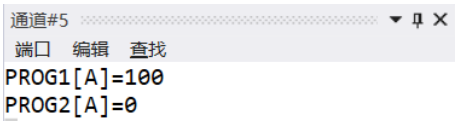
RUN "PROG1" '运行 PROG1 程序

WHILE TRUE
  PRINT #5, "PROG2[A]="; a
  WA( 50000 )
WEND
```

VAR:

```
DIM a AS INTEGER
```

运行结果：



结果分析：

使用 INCLUDE 功能，变量定义文件中的变量不是全局变量，仍是局部变量。

3) 函数变量的作用域实例 1

在 FB.LIB 文件中定义两个函数，如下：

```
FUNCTION write_val(value AS INTEGER)
  DIM a AS INTEGER
  a = value
ENDFUNC
```

ESTUN & Trio

```
FUNCTION print_val( )
   DIM a AS INTEGER
   PRINT "a="; a
ENDFUNC
```

在 PROG3.BAS 文件中调用两个函数

```
write_val( 10 )
print_val( )
```

结果分析：

在功能函数中定义的局部变量，其作用域仅在该函数中；

4) 函数变量的作用域实例 2

将 FB.LIB 文件中的两个函数进行修改，如下：

```
DIM a AS INTEGER

FUNCTION write_val(value AS INTEGER)
   a = value
ENDFUNC

FUNCTION print_val( )
   PRINT "a="; a
ENDFUNC
```

在 PROG3.BAS 文件中调用两个函数

```
write_val( 10 )
print_val( )
```

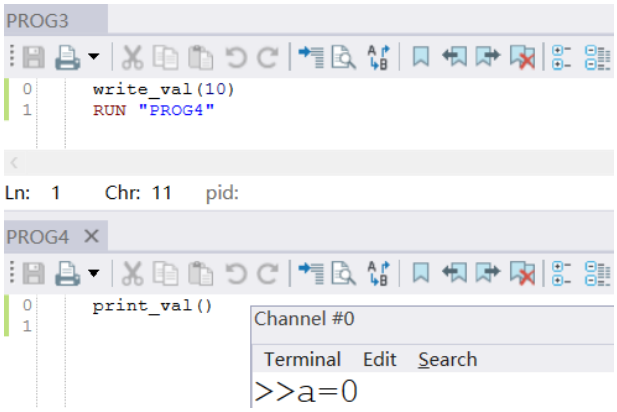
结果分析：

将变量定义在功能块外部，当函数被同一个任务调用时，该变量在两个函数间为全局的。

5) 函数变量的作用域实例 3

定义的两个函数如实例 2，在 PROG3.BAS 和 PROG4.BAS 文件中分别调用两个函数，执行效果如下：

ESTUN & Trio



综上可知，不同函数使用函数文件中定义的变量时，函数在不同任务被调用，变量表达含义不一致。

3.1.5 INCLUDE

功能描述	将多个任务中定义的相同名称的局部变量统一在一个文件中声明，减少变量声明重复工作；
使用注意	1) INCLUDE 的文件中只包含变量的定义和赋值； 2) 最大支持 5 层嵌套使用； 3) 不允许交叉调用，文件 A 包含文件 B,那么文件 B 不可以包含文件 A； 4) INCLUDE 仅适用于.BAS 文件，不适用 BASIC LIB 文件； 5) 一个.BAS 文件可以包含多个变量定义文件。

使用实例	将输入输出变量和轴名称变量分别定义在不同文件；
	VAR_IO
	`DIM di_startbtn AS INTEGER`
	`DIM di_estopbtn AS INTEGER`
	`DIM do_greenled AS INTEGER`
	
	`di_startbtn = 10`
	`di_estopbtn = 11`
	`do_greenled = 15`
	VAR_AXISNAME
	`DIM ax_x AS INTEGER`
	`DIM ax_y AS INTEGER`
	
	`ax_x = 0`
	`ax_y = 1`
	MAIN
	`INCLUDE "VAR_IO"`
	`INCLUDE "VAR_AXIS_NAME"`

ESTUN & Trio

```
WHILE TRUE
  ' 状态机代码
WEND
```

MONITOR

```
INCLUDE "VAR_IO"
INCLUDE "VAR_AXIS_NAME"

WHILE TRUE
    ' 监控代码
WEND
```

ESTUN & Trio

3.2 全局变量-VR

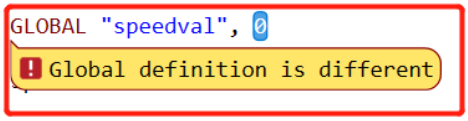
3.2.1 描述

功能描述	VR 变量为 TRIO 控制器内部定义的全局变量
特点	1) 默认情况下，VR 变量掉电保存； 2) VR 变量的长度为 64 位，遵循 IEEE-754 标准（53 位有效位，1 符号位和 11 指数位）； VR(0) -54 = 1 LOOKAHEAD_FACTOR Bit number out of range 1) VR 变量可用于在不同任务间传送字符串信息； 2) 不同型号控制器最大支持 VR 个数不一致； MC4N/MC403/MC405/MC508：4096 NANO/MC6N：16384 MC664/MC664-X：65536

3.2.2 定义-GLOBAL

指令格式	GLOBAL ["name"[, vr_number]]
功能描述	GLOBAL 指令用于定义全局 VR 变量，用于提高程序可读性；
实例	VR(1)表示速度参数； GLOBAL "speedval", 1 speedval = 100
使用注意	1) 不同型别控制器支持不同个数 GLOBAL 变量的声明； MC464：1024 MC403/MC405/Euro404/Euro408/MC4N-ECAT/MC4N-RTEX：4096 MC508/NANO/MC6N/MC664/MC664-X：8192 2) GLOBAL 变量的声明一般编写在初始化程序当中，因为当有任务在执行时，执行 GLOBAL 语句，运行出错； GLOBAL "speedval", 0 More than one program running when trying to set GLOBAL/CONSTANT 3) 如果已经将变量关联了 VR，修改关联，运行出错；

ESTUN & Trio



4) 在窗口 0 执行 LIST_GLOBAL 可以查看声明的 GLOBAL 变量个数；

```
list_global
Global                          VR
----------------                ----
speedval                        1
Constant                        Value
----------------                --------
```

5) GLOBAL 变量声明的变量其作用域是全局的，不同任务，不同函数均可读写；

3.2.3 VR 操作

3.2.3.1 SET_BIT

指令格式	SET_BIT(bit, variable)
功能描述	将 variable 指定的 VR 对应的 bit 位置 1； bit：指定 bit 位（范围：0-51） Variable：指定操作的 VR 地址
实例	'将 VR(10)的 bit9 位置 1 v_addr = 10 bitnum = 9 SET_BIT(bitnum, v_addr)

3.2.3.2 CLEAR_BIT

指令格式	CLEAR_BIT(bit, variable)
功能描述	将 variable 指定的 VR 对应的 bit 位清 0； **bit**：指定 bit 位（范围：0-51） **Variable**：指定操作的 VR 地址
实例	'将 VR(10)的 bit9 位清 0 v_addr = 10 bitnum = 9 CLEAR_BIT(bitnum, v_addr)

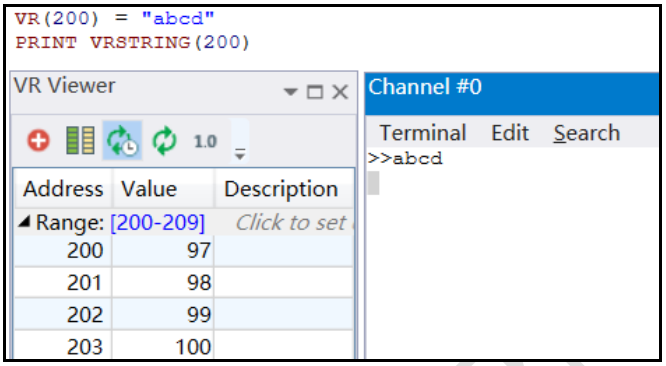
3.2.3.3 READ_BIT

指令格式	READ_BIT(bit, variable)
功能描述	读取 variable 指定的 VR 对应的 bit 位状态； **bit**：指定 bit 位（范围：0-51） **Variable**：指定操作的 VR 地址

ESTUN & Trio

实例	'读取 VR(10)的 bit9 位 v_addr = 10 bitnum = 9 value = READ_BIT(bitnum, v_addr)

3.2.3.4 VRSTRING

指令格式	VRSTRING(variable)
功能描述	返回 variable 开始地址存储的字符
实例	 说明： 使用 VR 传输字符串，从括号内的首地址开始，依次存放每个字符的 ASC 码，如例子中,VR(200)=97=ASC(a), VR(201)=98=ASC(b), VR(202)=99=ASC(c), VR(203)=100=ASC(d); 使用 VRSTRING 读取字符串时，从括号内指定的首地址开始，依次解出 ASC 码对应的字符，当遇到 VR 内部存储的值为 0 时，停止解析，并将字符串返回；

3.2.3.5 其他方式

置位	VR(0).10 = 1'将 VR(0)的 BIT10 位置 1
复位	VR(0).10 = 0 '将 VR(0)的 BIT10 位清 0
位状态读取	Value = VR(0).10

3.2.4 使用场合

VR 变量主要用于以下场合：

- 与 HMI 的通讯，VR 变量在默认状况下，对应 MODBUS 通讯 4 区；
- 由于 VR 变量即可对位操作，可用于关联 HMI 的操作按钮；
- 由于 VR 变量的掉电保存特性，可用于设备的参数存储；

- 如果当前 TRIO 控制器作为客户端，对其他控制器进行 MODBUS 通讯（包括另外的 TRIO 控制器），VR 变量可用于数据的传输，对应 MODBUS 通讯 4 区；
- 由于 VR 变量的易操作性，TRIO 不同任务之间需要传输数据信息或者 BIT 位信息，可使用 VR 变量进行交互；

3.3 全局变量-TABLE

3.3.1 描述

功能描述	TABLE 变量为 TRIO 控制器内部定义的全局变量
特点	默认情况下，TABLE 变量掉电不保存； TABLE 变量的长度为 64 位 TABLE 变量的个数为 512000；

3.3.2 数据操作

1) BIT 位操作

TRIO 控制器提供 TABLE 变量的 BIT 位操作函数。

例如：

读位的值 bit_value = TABLE(index).bit_num	Bit_value = TABLE(100).3
置位位 TABLE(index).bit_num = 1	TABLE(100).3 = 1
清除位 TABLE(index).bit_num = 0	TABLE(100).3 = 0

2) 数值操作

- 数值写

'赋值方式 1 TABLE(1000, 10, 20, 30.5)	'赋值方式 2 TABLE(1000, 10) TABLE(1001, 20) TABLE(1002, 30.5)
'赋值方式 3	'赋值方式 4

ESTUN & Trio

'地址参数可以为变量 DIM tb_hd_addr AS INTEGER tb_hd_addr = 1000 TABLE(tb_hd_addr, 10, 20, 30.5)	'赋值可以为变量 DIM tb_hd_addr AS INTEGER tb_hd_addr = 1000 TABLE(tb_hd_addr, VR(0), 20, 30.5)

- 数值读

'赋值方式 1 TABLE(1000, 10, 20, 30.5)	'赋值方式 2 TABLE(1000, 10) TABLE(1001, 20) TABLE(1002, 30.5)

3.3.3 使用场合

3.3.3.1 存储凸轮表数据

TABLE 变量可用于存储电子凸轮表点位，CAM、CAMBOX 相关指令索引该区域值，实现凸轮运动控制。

CAM

Type:

Axis Command

Syntax:

```
CAM(start point, end point, table multiplier, distance)
```

start point:	The start position of the cam profile in the TABLE
end point:	The end position of the cam profile in the TABLE
multiplier:	The table values are multiplied by this value to generate the positions.
distance:	The distance parameter relates the speed of the axis to the time taken parameter (which are in user units).

CAMBOX

Type:

Axis Command

Syntax:

```
CAMBOX(start_point, end_point, table_multiplier, link_distance , link_axis[, link_options][, link_pos][, offset_start])
```

start_point:	The start position of the cam profile in the TABLE
end_point:	The end position of the cam profile in the TABLE
table_multiplier:	The table values are multiplied by this value to generate the positions.
link_distance:	The distance the link axis must move to complete CAMBOX profile.
link_axis:	The axis to link to.

3.3.3.2 存储机器人机械参数

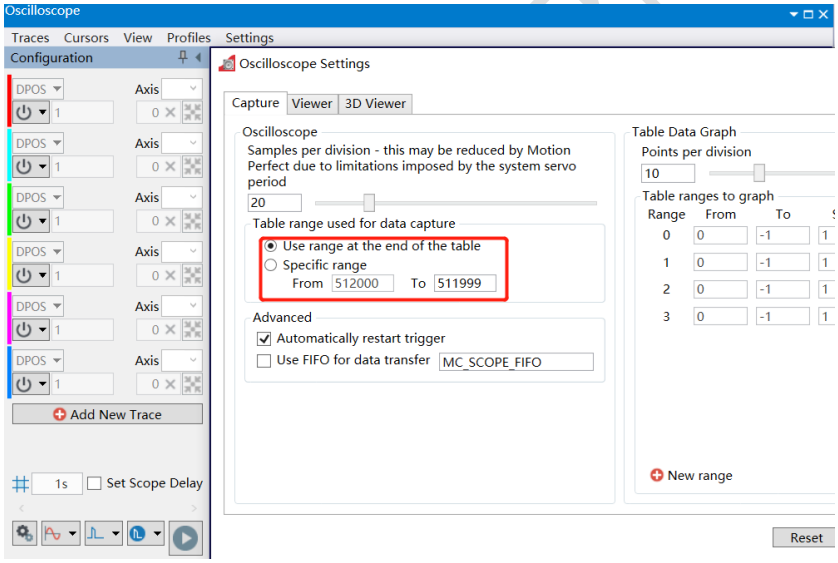
使用 FRAME 功能激活机械手的控制，需要在 TABLE 区存储机械手相关的机械参数。



3.3.3.3 示波器占用

TRIO 内部的示波器及 3D 可视化功能，将采集信息存储至 TABLE 区，然后显示波形或者动画。

在示波器的设置中，默认情况下，会将采集信息自动存储至 TABLE 区域的末端区域。



NOTE:鉴于此，我们在调试时，需要避免程序中使用的 TABLE 区间不要与此区间重合，避免出现意外的程序 BUG。

3.3.3.4 MODBUS 通讯

可以将 TABLE 区设置为 MODBUS 的 4 区，用于与触摸屏通讯。参考 ETHERNET 指令使用。

```
ETHERNET( 1, -1, 14, 0, 7, -1 )' cancel all previous maps
ETHERNET( 1, -1, 14, 0, 7, 100, 200, 10, 0, 0 ) ' VRs 200 to 209 size = Int 16
ETHERNET( 1, -1, 14, 0, 7, 110, 300, 10, 0, 1 ) ' VRs 300 to 209 size = Int 32
ETHERNET( 1, -1, 14, 0, 7, 130, 400, 10, 0, 2 ) ' VRs 400 to 409 size = FP 32
ETHERNET( 1, -1, 14, 0, 7, 150, 400, 10, 1, 2 ) ' TABLE 400 to 409 size = FP 32
```

```
ETHERNET( 0, -1, 14, 0, 7 ) ' display the map to terminal #0
```

3.3.3.5 不同任务间的数据交互

使用同 VR 一致。

3.3.3.6 配方功能

开辟一段连续区域，存储设备的多组参数，通过 FLASH_TABLE 指令实现。

3.3.4 相关指令

3.3.4.1 TABLE 写保护（TABLE_PROTECT）

<table>
<tr><td>指令形式</td><td>TABLE_PROTECT (first [, last/lock [, lock]])</td></tr>
<tr><td>功能描述</td><td>用于锁定/解锁 TABLE 变量区域的写操作；
<table>
<tr><td>first</td><td>首地址</td></tr>
<tr><td>Last</td><td>尾地址</td></tr>
<tr><td>LOCK</td><td>0：可读写（默认）；1：仅读；</td></tr>
</table></td></tr>
<tr><td>实例</td><td>机械手参数的写保护；
<pre>
'Start Robot 0 configuration
TABLE_PROTECT(2000, 2100, 0) 'Protected table range for robot definition

KINEMATIC_GROUP(0,-1)
TABLE(2000,325,275,20,100000*(80/1)/(2*PI),10000*(50/1)/(2*PI),100000*(30/1)/20)
TABLE(2006,10000*(1/1)/(2*PI),1,0,0,0)

WORLD_POS_REDUCED_SPEED = 10
WORLD_POS_MAX_SPEED = 100
WORLD_POS_MAX_ACC = 1000
WORLD_POS_MAX_JERK = 1000
WORLD_ORI_REDUCED_SPEED = 10
WORLD_ORI_MAX_SPEED = 10
WORLD_ORI_MAX_ACC = 100
WORLD_ORI_MAX_JERK = 100
WORLD_POS_JOG_SPEED = 10
WORLD_POS_JOG_ACC = 10
WORLD_ORI_JOG_SPEED = 10
WORLD_ORI_JOG_ACC = 10
JOG_SPEED_FACTOR = 1
JOG_ACC_FACTOR = 1

Axis 0 configuration

Axis 1 configuration

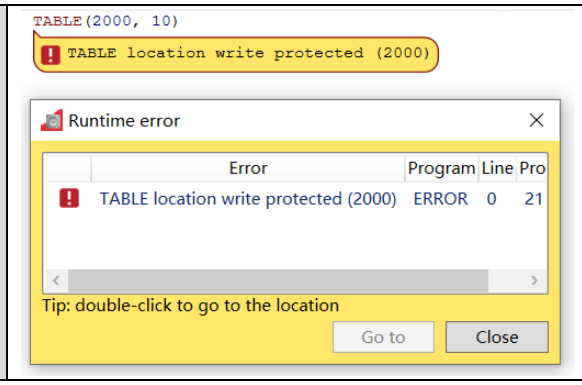
Axis 2 configuration

Axis 3 configuration

Axis 4 configuration

Axis 5 configuration

KINEMATIC_GROUP(0,2000,41,0,1,2,3,4,5)
TABLE_PROTECT(2000, 2100, 1) 'Protected table range for robot definition
'Stop Robot 0 configuration
</pre>
当对 TBALE 区域执行写保护后，对该区域进行写操作，任务报错，如下：</td></tr>
</table>



3.3.4.2 FLASH_TABLE

指令形式	FLASHTABLE(function,flashpage,tablepage)		
功能描述	将 TABLE 区域内的值存储至 FLASH 区域，或者将 FLASH 区域的数值读取至 TABLE 区；		
	function	1	将 TABLE 区域值写入 FLASH EPROM
		2	读 FLASH EPROM，写入 TABLE
		3	读 FLASH EPROM（该区域有存储信息）
	flashpage	Flash 区域选择（0-31），每个区域可存储 16000 数值	
	tablepage	Table 区域选择（0-31），每个区域可存储 16000 数值	

ESTUN 埃斯顿

实例	1. VR(1000)-VR(1499)存储设备的当前参数，设备的配方存储至 TABLE 区域 TABLE(16000)-TABLE(31999)，共 48 组； '配方存储 FOR i = 0 TO 499 TABLE(16000 + 500 * forluma_write_index, VR(1000 + i)) NEXT i flash_writecmd = 1 flashpage = 5 tablepage = INT(16000 / 16000) FLASHTABLE(flash_writecmd, flashpage, tablepage) '配方读取 flash_readcmd = 3 flashpage = 5 tablepage = INT(16000 / 16000) FLASHTABLE(flash_readcmd, flashpage, tablepage) FOR i = 0 TO 499 VR(1000 + i) = TABLE(16000 + 500 * forluma_read_index) NEXT i

3.3.4.3 FLASH_DATA

功能描述

系统参数；

MC_CONFIG 文件中设置；

FLASH_DATA 控制是否将 VR 或 TABLE 数据自动备份到闪存；

MC4N	0	Vr 变量掉电保存
	1	TABLE 变量掉电保存
MC664	0	VR 变量和前 196608 个 TABLE 掉电保存
	1	VR 变量和 TABLE 变量掉电保存
MC6N/NANO/ MC508	0	Vr 变量掉电保存
	1	Vr 变量和前 64000 个 TABLE 掉电保存

ESTUN & Trio

3.4 常量

3.4.1 CONSTANT

<table>
<tr><td>指令形式</td><td>CONSTANT ["name"[, value]]</td></tr>
<tr><td>功能描述</td><td>声明全局常量，提高可读性；
<table>
<tr><td>name</td><td>常量名称</td></tr>
<tr><td>value</td><td>常量大小，可以为数值，也可以为运算公式，</td></tr>
</table></td></tr>
<tr><td>实例</td><td>1. 声明常量 one、servo_to_ms，其大小分别为 1 和 SERVO_PERIOD / 1000
CONSTANT "one", 1
CONSTANT "servo_to_ms", SERVO_PERIOD / 1000</td></tr>
<tr><td>使用注意</td><td>声明常量时，常量数值可以为数字，也可以为表达式，但不可以为字符串变量；
<pre>CONSTANT "one", 1
CONSTANT "servo_to_ms", SERVO_PERIOD / 1000
CONSTANT "test", "aa"
Invalid mix of data types</pre>
不同型别控制器，最大支持的常量定义个数不一致；
<table>
<tr><td>MC464</td><td>1024</td></tr>
<tr><td>MC403/MC405/Euro404/Euro408/MC4N-ECAT/C4N-RTEX</td><td>4096</td></tr>
<tr><td>MC508/NANO/MC6N/MC664/MC664-X</td><td>8192</td></tr>
</table>
CONSTANT 变量的声明一般编写在初始化程序当中，当有任务在执行时，执行 CONSTANT 语句，运行出错；
<pre>CONSTANT "state_manu", 0
More than one program running when trying to set GLOBAL/CONSTANT</pre>
如果已经将常量关联了数值，修改关联，运行出错；如需修改常量变量关联数值，必须重启控制器；
<pre>CONSTANT "state_manu", 20
Global definition is different</pre>
在窗口 0 执行 LIST_GLOBAL 可以查看声明的 CONSTANT 变量个数；
<pre>>>LIST_GLOBAL
Global   VR
----------------   ----
Constant   Value
----------------   -------
one   1.00000
servo_to_ms   1.00000</pre></td></tr>
</table>

	CONSTANT 指令声明的常量其作用域是全局的，不同任务，不同函数均可读，但不可写；
	`state_manu = 20`
	`Value is read only`
	`CONSTANT "state_auto", 10`

3.4.2 DEFCONST

指令形式	DEFCONST name value
功能描述	声明任务常量，提高可读性； **name**：常量名称 **value**：常量大小，可以为数值，也可以为运算公式，
实例	DEFCONST state_manu 10 DEFCONST state_auto 20 DEFCONST mysize 20 DEFCONST fval SIN(90)

应用场合	在定义数组变量时，数组变量的维度不可以为变量，可以使用 DEFCONST，便捷修改数组维度；
	`DIM array_size AS INTEGER`
	`array_size = 100`
	`DIM a AS INTEGER( array_size )`
	`DIM b AS FLOAT( array_size )`
	SELECT_CASE 语句中使用，提高程序可读性；
	`DEFCONST state_manu 10`
	`DEFCONST state_auto 20`
	`DEFCONST state_error 30`
	`SELECT_CASE sys_state`
	`  CASE state_manu`
	`    ' commands`
	`  CASE state_auto`
	`    ' commands`
	`  CASE state_error`
	`    ' commands`
	`  CASE ELSE`
	`    ' commands`
	`END_CASE`

使用注意	常量可读，但不可写；

ESTUN & Trio

	DEFCONST state_manu 10 DEFCONST state_auto 20 DEFCONST mysize 20 DEFCONST fval SIN(90) state_manu = 30 Extra characters at end of line DEFCONST 声明的常量作用域不是全局的；仅在当前任务有效； DEFCONST 声明可在.LIB 文件中使用； DEFCONST 指令需要固件版本 2.300 以上支持；

3.4.3 PI

功能描述	关键字； 只读； 表示 π 的大小；

3.4.4 TRUE

功能描述	关键字； 只读； 布尔状态：真；值为-1；

3.4.5 FALSE

功能描述	关键字； 只读； 布尔状态：假；值为 0；

3.4.6 ON

功能描述	关键字； 只读；值为 1；

3.4.7 OFF

功能描述	关键字； 只读；值为 0；

ESTUN & Trio

3.5 运算符

3.5.1 =

功能描述	赋值；
实例	a = 100 '表示将 100 赋值给变量 a;
功能描述 2	比较

实例	IF IN(7) = ON THEN
	
	ENDIF

3.5.2 +

功能描述	加法运算
实例	a = b + c

3.5.3 –

功能描述	减法运算
实例	a = b - c

3.5.4 *

功能描述	乘法运算
实例	a = b * c

3.5.5 /

功能描述	除法运算
实例	a=10/(2.1+9)

3.5.6 MOD（整数取余）

功能描述	整数取余运算
实例	>>PRINT 18 MOD 12 6.0000 >>

ESTUN & Trio

3.5.7 FMOD（浮点数取余）

功能描述	浮点数取余运算
实例	>>PRINT 2023.4 FMOD 356 243.4000 >>PRINT -204.56 FMOD 60 -24.5600 >>PRINT 32760 FMOD 204.8 196.8000

3.5.8 ^（多次方）

表达式	<expression1> ^ <expression2>
功能描述	多次方运算
实例	x = 2 ^ 6 'X=2 的 6 次方

3.6 判断

3.6.1 >

表达式	<expression1> > <expression2>
功能描述	如果 expression1（表达式 1）大于 expression2（表达式 2），则返回 TRUE ，否则返回 FALSE 。

3.6.2 <

表达式	<expression1> < <expression2>
功能描述	如果 expression1（表达式 1）小于 expression2（表达式 2），则返回 TRUE ，否则返回 FALSE 。

3.6.3 >=

表达式	<expression1> >= <expression2>
功能描述	如果 expression1（表达式 1）大于或者等于 expression2（表达式 2），则返回 TRUE ，否则返回 FALSE 。

ESTUN & Trio

3.6.4 <=

表达式	<expression1> <= <expression2>
功能描述	如果 expression1（表达式 1）小于或者等于 expression2（表达式 2），则返回 TRUE ，否则返回 FALSE 。

3.6.5 <>

表达式	<expression1> <> <expression2>
功能描述	如果 expression1（表达式 1）不等于 expression2（表达式 2），则返回 TRUE ，否则返回 FALSE 。

3.7 位运算

3.7.1 >>

表达式	<expression1> >> <expression2>
功能描述	右移位指令

3.7.2 <<

表达式	<expression1> << <expression2>
功能描述	左移位指令

3.7.3 AND

表达式	<expression1> AND <expression2>
功能描述	与运算

3.7.4 OR

表达式	<expression1> OR <expression2>
功能描述	或运算

ESTUN & Trio

3.7.5 NOT

表达式	NOT expression
功能描述	取反运算

3.7.6 XOR

表达式	<expression1> XOR <expression2>
功能描述	异或运算

3.8 特殊符号

3.8.1 :

功能描述	分隔符
实例	标签使用 sub_lab: RETURN 用于在多语句行上分隔 TrioBASIC 语句 BASE(0): UNITS = 1000: SPEED = 100

ESTUN & TrIO

3.8.2 _(换行符)

功能描述	换行符
实例	IF IN(0) = ON _ AND IN(1) = OFF _ AND IN(2) = ON THEN ENDIF

3.8.3 $（16 进制）

功能描述	16 进制符
实例	DIM a AS INTEGER a = $FF11

3.8.4

功能描述	指定端口号
实例	PRINT #5, "MESS"

3.8.5 ‘（注释符号）

功能描述	注释语句，注释字符默认为绿色
实例	'PROGRAM TO ROTATE WHEEL turns = 10 'turns contains the number of turns required MOVE(turns)' the movement occurs here

3.9 结构指令

通过本节的学习，能够掌握 TrioBasic 编程环境中，不同结构指令的使用及应用场合。

3.9.1 IF 语句

指令语法	IF condition THEN
	commands
	ELSEIF expression THEN
	commands
	ELSE
	commands
	ENDIF

功能描述	判断语句，当判断条件成立时，执行相应语句；当判断条件不成立时，执行 else 语句对应的内容；

示例	读取系统 WDOG 数值，控制灯输出；
	warning = 8
	IF WDOG = ON THEN
	OP(warning, ON)
	ELSE
	OP(warning, OFF)
	ENDIF
	读取不同状态，打印不同字符
	IF operating_state = 0 THEN
	PRINT#5, "Machine Running"
	ELSEIF operating_state = 1 THEN

ESTUN & Trio

```
    PRINT#5, "Machine Idle"
  ELSEIF operating_state = 2 THEN
    PRINT#5, "Machine Jammed"
  ELSE
    PRINT#5, "Machine in unknown state"
  ENDIF
```

多个条件判断

```
IF IN(0) = ON AND IN(1) = OFF AND VR(10) = 100 THEN
  OP(10, ON)
ELSE
  OP(10, OFF)
ENDIF
```

当 3 个条件存在不成立条件时，使用该写法不利于监控具体哪个条件不成立，可按照以下写法:

```
DIM conditon AS BOOLEAN
conditon = (IN(0) = ON)
conditon = conditon AND (IN(1) = OFF)
conditon = conditon AND (VR(10) = 100)
IF conditon THEN
  OP(10, ON)
ELSE
  OP(10, OFF)
ENDIF
```

备注	如上例 2，对于分支结构较多的 IF 语句，建议使用 CASE 语句进行替换； 如上例 3，对于判定条件较多的 IF 语句，建议使用拆分方式，提高可调试性、可读性； IF 语句主要用于条件判定，类似于 PLC 当中的触点功能。

ESTUN & Trio

3.9.2 CASE 语句

指令语法	SELECT_CASE variable
	  CASE 1
	commands
	  CASE 3,5,7
	Commands
	  CASE 8,10,12
	commands
	  CASE ELSE
	  commands
	END_CASE

功能描述	分支选择语句，读取变量 variable 数值，执行当前值内代码，不需 BREAK 语句，

	即可跳出至 END_CASE 语句
示例	CASE 语句可用于条件较多的判断 variable = IN(0, 1) SELECT_CASE variable CASE 1 PRINT "Input 0 is ON" CASE 2 PRINT "Input 1 is ON" CASE 3 PRINT "Inputs 0 and 1 are ON" CASE ELSE PRINT "Inputs 0 and 1 are OFF" END_CASE CASE 语句可用于流程控制
备注	1. 对于分支结构较多的 IF 语句，使用 CASE 语句替代； 2. CASE 语句的分支，可使用 DEFCONST 定义的局部常量，可提高程序可读性； 3. CASE 语句不需要编写额外的 BREAK 语句； 4. 一个选择语句最大支持 200 个 CASE; 5. CASE 语句主要用于设备的流程控制；

3.9.3 WHILE 语句

指令语法	WHILE condition Commands WEND
功能描述	循环语句，当 condition 成立（非 0），执行 While 循环内部指令；当 condition=0 时，跳出循环；

ESTUN & TRIO

示例

使用 WHILE 语句构建死循环，实现类似 PLC 自上而下循环扫描功能

```
WHILE TRUE
      ' 代码部分
WEND
```

使用 WHILE 语句构建查询语句，类似于 FOR 循环功能；查询 VR(0)-VR(99)中哪个值为 10，检索到第一个，跳出循环；

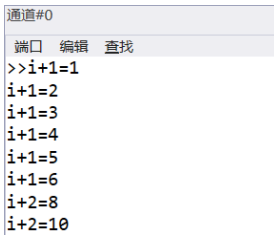
```
 i = 0
WHILE i < 100
  IF VR(i) = 10 THEN
    EXIT_LOOP
  ENDIF
  i = i + 1
WEND
```

WHILE 语句结合 CONTINUE 语句使用，执行到 CONTINUE 语句，可以直接返回条件判断部分；

```
i = 0
WHILE i < 10
  IF i > 5 THEN
    i = i + 2
    PRINT "i+2="; i
    CONTINUE
  ENDIF

  i = i + 1
  PRINT "i+1="; i
WEND
```

执行效果：



备注

1. WHILE 语句一般用于构建循环任务；
2. WHILE 语句可以使用 EXIT_LOOP 指令跳出循环；
3. WHILE 语句可以结合 CONTINUE 语句使用；

ESTUN & Trio

3.9.4 REPEAT 语句

指令语法	REPEAT commands UNTIL expression
功能描述	循环语句，当 expression 成立（非 0），跳出循环，当 expression 为 false 时，继续执行 commands；

示例	使用 REPEAT 语句构建死循环，实现类似 PLC 自上而下循环扫描功能：
	`REPEAT`
	`  '代码部分`
	`UNTIL FALSE`
	REPEAT 语句功能与 WHILE 语句类似，不同点在于 REPEAT 语句为先执行再判断，WHILE 语句为先判断再执行；
	REPEAT 语句结合可结合 EXIT_LOOP 指令使用，如下，进行查询工作；
	`REPEAT`
	`  IF VR(i) = 10 THEN`
	`  EXIT_LOOP`
	`  ENDIF`
	`  i = i + 1`
	`UNTIL i >= 99`

3.9.5 FOR 循环

指令语法	FOR variable = start TO end [STEP increment] commands NEXT variable
功能描述	FOR 循环，一般用于重复执行多次操作，当执行该语句时，variable 会被赋值为 start，按照 STEP 设定的值逐步变化，直到变化至 end，退出循环。

示例	FOR 循环的 STEP 值默认为 1；
	`FOR op_num = 10 TO 18`
	`  OP(op_num, ON)`
	`NEXT op_num`
	执行效果：

ESTUN & Trio



FOR 循环的 STEP 值可修改，可以为正值，也可以为负值；

```
DIM i AS FLOAT

FOR i = 2 TO 0 STEP - 0.4
   PRINT "i="; i
NEXT i
```

执行结果：

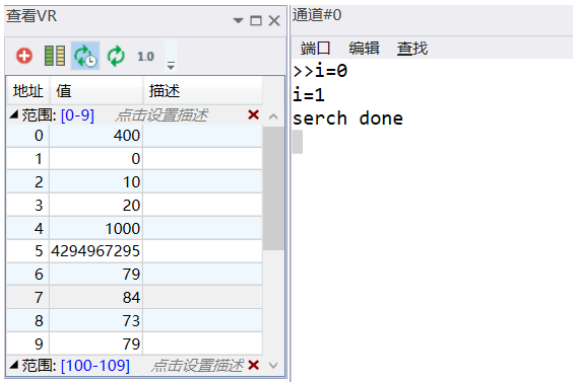
```
>>i=2.0000
i=1.6000
i=1.2000
i=0.8000
i=0.4000
i=1.1102e-16
```

FOR 循环语句可以配合 EXIT_LOOP 语句使用，可提前退出循环

```
DIM i AS INTEGER

FOR i = 0 TO 10
   IF VR(i) = 10 THEN
      PRINT "serch done"
      EXIT_LOOP
   ENDIF
   PRINT "i="; i
NEXT i
```

执行结果：



备注

默认情况下，STEP 的值为 1；

STEP 的值可以为正值，则 END 的设定值应当大于 START 的设定值；

STEP 的值可以为负值，则 END 的设定值应当小于 START 的设定值；

FOR 循环最大支持 8 层嵌套；

ESTUN & TIM

3.9.6 EXIT_LOOP

功能描述	结合 FOR 循环、WHILE 循环、REPEAT 循环使用，可用于提前跳出循环
示例	详见 WHILE 循环、REPEAT 循环、FOR 循环

3.9.7 CONTINUE

功能描述	结合 FOR 循环、WHILE 循环、REPEAT 循环使用，一旦触发 CONTINUE 语句，将直接跳转到下一次循环
示例	详见 WHILE 循环、REPEAT 循环、FOR 循环

3.9.8 GOSUB

指令语法

```
GOSUB label
...
label:
  commands
RETURN
```

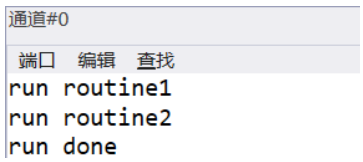
功能描述

调用标签，执行到 GOSUB 语句，将跳转至指定标签内执行程序，执行到 RETURN 语句，返回至 GOSUB 语句；

示例

1. 参考下例，可知标签跳转执行的顺序；

```
GOSUB routine1
GOSUB routine2
PRINT "run done"
STOP
routine1:
PRINT "run routine1"
RETURN
routine2:
PRINT "run routine2"
RETURN
```



2. 一个标签可以关联多个 return 语句；

```
WHILE TRUE
  GOSUB lab_check_error
  IF b_sys_error THEN
    OP(red_led, ON)
  ELSE
    OP(red_led, OFF)
```

ESTUN & Trio

```
    ENDIF
WEND

lab_check_error:
b_sys_error = FALSE

IF SYSTEM_ERROR <> 0 THEN
    b_sys_error = TRUE
    RETURN
ENDIF

IF MOTION_ERROR <> 0 THEN
    b_sys_error = TRUE
    RETURN
ENDIF

RETURN
```

3. 标签支持嵌套，一个标签程序可以调用另一个标签；

```
GOSUB routine1
PRINT "run done"
STOP

routine1:
PRINT "run routine1"
GOSUB routine2
RETURN

routine2:
PRINT "run routine2"
RETURN
```

```
通道#0
端口  编辑  查找
run routine1
run routine2
run done
```

备注	标签最大支持 8 层嵌套；

3.9.9 GOTO

指令语法	`GOTO label`
	`…`
	`label:`

功能描述	跳转标签，执行到 GOTO 语句，将跳转至指定标签内执行程序，不返回，GOTO 指令和标签指令之间的代码将跳过；
示例	GOTO 语句一般应用于初始化程序中，不建议在流程程序中使用； `IF FLAG(f_ini_done) = ON THEN`

ESTUN & Trio

```
    GOTO sys_run
ENDIF
GOSUB ectercat_ini
GOSUB axisnum_check
FLAG(f_ini_done, ON)

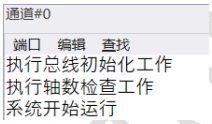
sys_run:
PRINT "系统开始运行"
RUN "MAIN"
RUN "MONITOR"
STOP

ectercat_ini:
PRINT "执行总线初始化工作"
RETURN

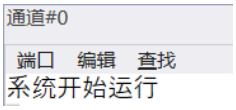
axisnum_check:
PRINT "执行轴数检查工作"
RETURN
```

执行效果：

第一次：



第二次



备注	GOTO 语句不建议在流程程序中使用；

3.9.10FUNCTION

指令语法	FUNCTION name([param1 AS type[, param2 AS type[, param3 AS type ….]]]) …. RETURN x ENDFUNC
功能描述	用户定义函数，完成工程项目中重复使用代码的封装，提高工程的可读性、易维护性; FUNCTION 文件在 BASIC Library 文件中编辑；

示例	用户定义函数可以不带返回值，也可以不带返回值；
	不带返回值：
	`FUNCTION set_rs232(baudrate AS INTEGER)`
	`  SETCOM(baudrate, 8 , 1 , 0 , 1 , 0)`
	`ENDFUNC`

ESTUN & Trio

带返回值：

```
FUNCTION get_decel_distance(f_decel AS FLOAT , f_vel AS FLOAT) AS
FLOAT
  set_rs232( 9600 )
  DIM dec_dist AS FLOAT
  ' calculate the deceleration distance using v^2 = u^2 + 2as
  dec_dist = f_vel ^ 2 / (2 * f_decel)
  RETURN dec_dist
ENDFUNC
```

用户函数返回值类型不支持数组类型；

```
FUNCTION math(inval1 , inval2 AS FLOAT) AS FLOAT()
```

Extra characters at end of line

```
ENDFUNC
```

数组变量可以作为函数参数；

```
FUNCTION max_i(data AS INTEGER()) AS INTEGER
  DIM i, max AS INTEGER
  max = data(0)

  FOR i = 1 TO data.dimsize(0) - 1
    IF data(i) > max THEN
      max = data(i)
    ENDIF
  NEXT i
  RETURN max

ENDFUNC
```

ESTUN & Trio

如上，数组变量可作为参数传递至函数，定义函数时不能定义数组的维度，可以通过 dimsize 获取数组维度的长度；

数组变量可以作为函数参数，用于返回多个值；

```
FUNCTION math_cal(a, b AS FLOAT, result AS FLOAT())
  result(0) = a + b
  result(1) = a - b
ENDFUNC
```

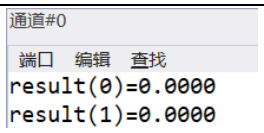
任务调用：

```
DIM result AS FLOAT( 10 )

math_cal( 100, 200, result )
PRINT "result(0)="; result( 0 )
PRINT "result(1)="; result( 1 )
```

```
通道#0
端口  编辑  查找
result(0)=300.0000
result(1)=-100.0000
```

使用非数组变量无法将函数内部运算值带回调用任务；

	FUNCTION math_cal2(a, b AS FLOAT, result, result1 AS FLOAT) result = a + b result1 = a - b ENDFUNC 任务调用： DIM result AS FLOAT(10) math_cal2(100, 200, result(0), result(1)) PRINT "result(0)="; result(0) PRINT "result(1)="; result(1)  关于函数变量作用域知识，可参考局部变量作用域章节。
备注	函数支持被其他函数调用，但不支持递归调用； 函数输入参数最大为 16 个； 函数嵌套调用最大为 5 层； GOSUB /GOTO 指令不可在函数中编辑； 2.301 固件版本及以上支持编辑 1024 个用户函数，之前版本支持 256 个用户函数；

3.9.11WAIT

指令语法	WAIT UNTIL expression
功能描述	等待表达式值为 TRUE
示例	1. WAIT 语句会造成程序堵塞，当运行到该行时，如果条件不满足，将无法执行当前任务的其他代码； DIM i_startbtn AS INTEGER i_startbtn = 8 WHILE TRUE WAIT UNTIL IN(i_startbtn) = ON GOSUB io_monitor WEND io_monitor: '将输入状态映射至 VR VR(0).0 = IN(10) VR(0).1 = IN(11) RETURN 运行结果：  运行分析： 程序堵塞在 WAIT UNTIL 语句，输入信号 10 和 11 状态发送改变，由于该部分代码未扫描，VR(0)的数值未发送改变；
备注	1. 使用 WAIT 语句，当程序执行到该行，如果条件不满足，则将会停止在这一步，直到条件满足；

ESTUN & Trio

	2. WAIT 语句会造成程序堵塞；

ESTUN & Trio

3.10 IO 指令

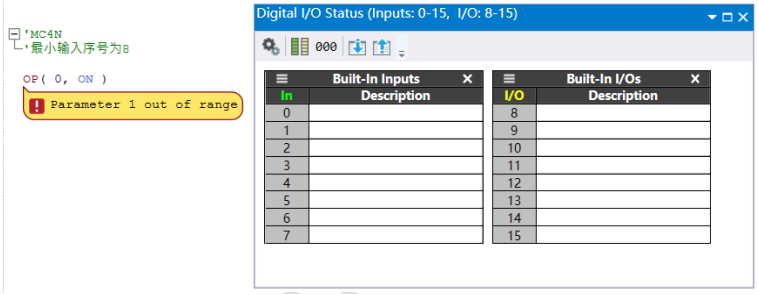
3.10.1数字量输入读取-IN

<table>
<tr><td>指令语法</td><td>value = IN[(input_no[,final_input])]</td></tr>
<tr><td>功能描述</td><td>读取单个数字量输入或者多个数字量输入的状态；
<table>
<tr><td>Value</td><td>返回读取输入点的状态</td></tr>
<tr><td>Input_no</td><td>读取输入起始点</td></tr>
<tr><td>Final_input</td><td>读取输入结束点（可选参数）</td></tr>
</table></td></tr>
<tr><td>示例</td><td>1. 读取单个输入
<pre>'读取单个输入
'判断 0 号输入是否导通，导通则打开 10 号输出，否则关闭
IF IN(0) = ON THEN
  OP(10, ON)
ELSE
  OP(10, OFF)
ENDIF</pre>
2. 读取多个输入
<pre>'读取多个输入
'将 0 号输入到 31 号输入存储至 VR(0)，用于 HMI 显示
'VR(0).0--IN(0);VR(0.1)--IN(1);...;VR(0).31 = IN(31)
VR(0) = IN(0, 31)</pre>
3. 使用 ETHERNETIP 通讯，如果通讯通道设置为 DI，可以通过读取多个输入的方式快捷的将开关量转换成数字量；</td></tr>
<tr><td>备注</td><td>1. 如果读取多个输入，单条指令最大读取输入为 32 位；</td></tr>
</table>

3.10.2数字量输出-OP

指令语法	OP(output, state)

ESTUN & Trio

<table>
<tr><td>功能描述</td><td>写单个输出
<table><tr><td>Output</td><td>指定操作数字量输出</td></tr><tr><td>state</td><td>OFF：关闭；ON：打开</td></tr></table></td></tr>
<tr><td>示例</td><td>'读取单个输入
'判断 0 号输入是否导通，导通则打开 10 号输出，否则关闭
IF IN(0) = ON THEN
OP(10, ON)
ELSE
OP(10, OFF)
ENDIF</td></tr>
<tr><td>备注</td><td>不同型号控制器，最小输出序号不一致；比如 MC4N，最小输出序号为 8，当对不存在的输出（0-7）进行操作，程序运行错误；
 
</td></tr>
<tr><td>指令语法</td><td>OP(start, end, state)</td></tr>
<tr><td>功能描述</td><td>写多个输出
<table><tr><td>start</td><td>指定操作数字量输出起始点</td></tr><tr><td>end</td><td>指定操作数字量输出结束点</td></tr><tr><td>state</td><td>各 BIT 位关联各输出</td></tr></table></td></tr>
<tr><td>示例</td><td>'将输出10-13都打开
OP(10, 13, $F) '$表示16进制</td></tr>
</table>

3.10.3数字量输出读取-READ_OP

指令语法	value = READ_OP(output [,finaloutput])

<table>
<tr><td>功能描述</td><td>读取单个输入、或者多个输入状态
<table><tr><td>value</td><td>存储输出状态</td></tr><tr><td>output</td><td>读取需要读取输出的起始点</td></tr><tr><td>finaloutput</td><td>读取需要读取输出的结束点（可选参数）</td></tr></table></td></tr>
<tr><td>示例</td><td>读取单个输出
<pre>'读取单个输出

IF READ_OP(10) = ON THEN
   PRINT "10 号输出已经打开"
ENDIF</pre>读取多个输出
<pre>'读取多个输入
'将 0 号输出到 31 号输出存储至 VR(1)，用于 HMI 显示
'VR(1).0--OP(0);VR(1).1--OP(1);...;VR(1).31 = OP(31)

VR(1) = READ_OP(0, 31)</pre></td></tr>
</table>

3.10.4数字量输入取反-INVERT_IN

<table>
<tr><td>指令语法</td><td>INVERT_IN(input, state)</td></tr>
<tr><td>功能描述</td><td>将数字量输入的状态进行软取反；
<table><tr><td>input</td><td>指定操作的输入通道</td></tr><tr><td>state</td><td>ON:激活取反
OFF:关闭取反</td></tr></table></td></tr>
<tr><td>示例</td><td>对 10 号数字量输入通道取反；
<pre>INVERT_IN(10, ON)</pre></td></tr>
<tr><td>使用</td><td>参考 DATUM_IN、FWD_IN 等指令</td></tr>
</table>

3.10.5位置比较输出-PSWITCH

<table>
<tr><td>指令语法</td><td>PSWITCH(switch, enable [,axis, output, state, setpos, resetpos])
PSWITCH(switch, OFF [, hold])</td></tr>
</table>

功能描述

位置比较输出功能

switch	选择，范围为 0-63，即最多有 64 个 PSWITCH 功能可以使用	
enable	0/OFF	激活位置比较功能
	1/ON	关闭位置比较功能
	5	基于 DPOS(目标位置)的位置比较功能
axis	参考轴（可以为实轴，也可以为虚拟轴）	
output	关联输出点	
state	0/OFF	到达 SetPos，输出点状态变为 OFF
	1/ON	到达 SetPos，输出点状态变为 ON
setpos	位置比较点 1，单位为 UNITS	
Resetpos	位置比较点 2，单位为 UNITS	
hold	0	输出点状态在执行 PSWITCH(,OFF,)保持不变
	1	输出点状态在执行 PSWITCH(,OFF,)变为 OFF

示例

位置比较例 1

```
'已知轴 1 为旋转轴，其位置变化为 0-360°
'要求输出点 20,在角度 30 度-100 度之间输出为 ON
axisnum = 1
BASE( axisnum )
UNITS = 100
SPEED = 100
ACCEL = 1000
DECEL = 1000
'设置旋转轴
REP_OPTION = ON
REP_DIST = 360

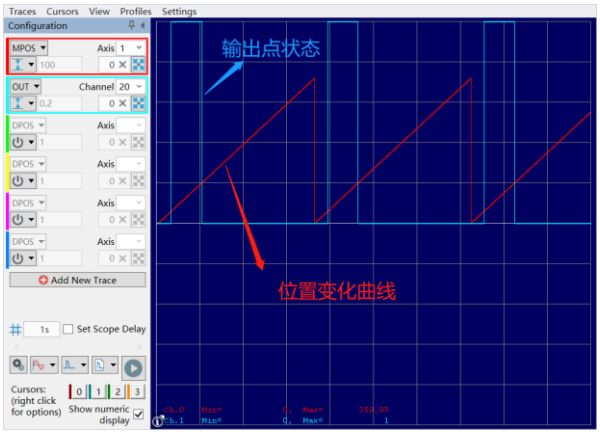
DIM out_num AS INTEGER
out_num = 20
DIM start_angle, end_angle AS FLOAT
start_angle = 30
end_angle = 100

'加载位置比较指令
PSWITCH( 0, ON, axisnum, out_num, ON, start_angle, end_angle )

FORWARD AXIS( axisnum ) '加载轴运动-轴一直正转
TRIGGER     '触发示波器采集
```

运行结果：

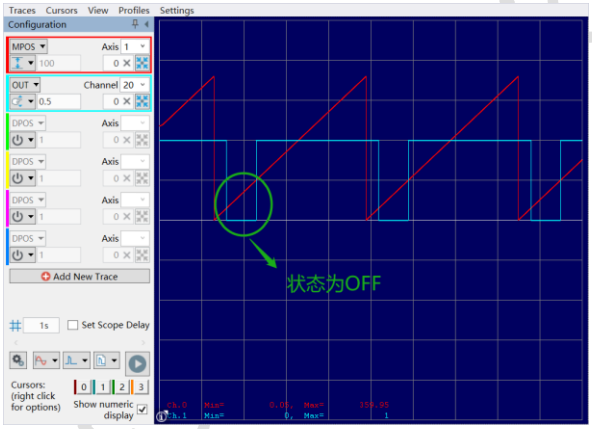
ESTUN & TMD



State=off，当位置值在【setpos,resetpos】之间，输出为 OFF,其余范围，输出为 ON;

```
'加载位置比较指令
PSWITCH( 0, ON, axisnum, out_num, OFF, start_angle, end_angle )
```

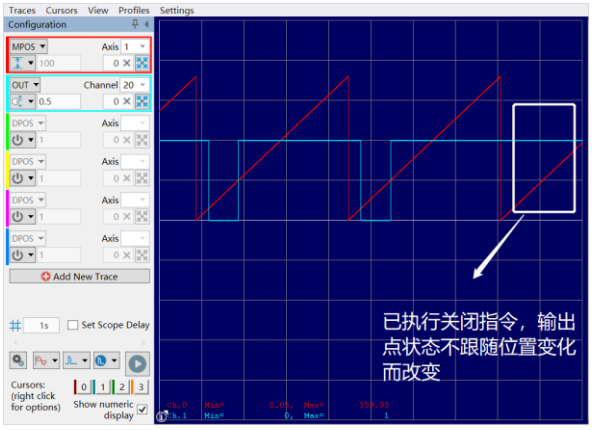
运行结果：



执行 PSWITCH(,OFF,)可以关闭位置比较功能；

```
PSWITCH( 0, OFF, 0 )
```

运行结果：



3.11 时间相关指令

3.11.1延时指令-WA

指令语法	WA(time)
功能描述	延时指定时间，执行下一行 **Time** 延时时间，单位为 ms
示例	'打开输出点 10，延时 100ms 打开输出点 11 OP(10, ON) WA(100) OP(11, ON)
备注	WA 指令会造成任务在设定时间内处于 SLEEP 状态，无法扫描其他程序

3.11.2周期设置-SERVO_PERIOD

功能描述	系统参数，设置控制器的伺服周期； 在 MC_CONFIG 文件中设置有效； 单位为微妙； 功能： 控制器与轴的通讯周期，设定值可以为 250us、500us、1000us、2000us SERVO_PERIOD 设置越小，系统负载越大； SERVO_PERIOD 设置越小，轴运动效果更好； 需要根据控制器性能及实际负载能力合理设置 SERVO_PERIOD；

3.11.3任务计时-TICKS

功能描述	任务参数，一个伺服周期 TICKS 值自动减 1； 单位为伺服周期（SERVO_PERIOD）； TICKS 数据为 64 位； 可读，可写；

ESTUN & Trio

示例

1. 使用 TICKS 计算程序运行消耗时间

```
'通过 TICKS 计算代码运行耗时
DIM l_i_ticks_preval AS INTEGER
DIM l_f_usetime AS FLOAT

WHILE TRUE
  l_i_ticks_preval = TICKS

  FOR i = 0 TO 1000
    VR( 0 ) = i
  NEXT i
  l_f_usetime = ABS( l_i_ticks_preval - TICKS ) * SERVO_PERIOD / 1000 '单位为 ms
WEND
```

2. 使用 TICKS 完成定时器功能，不会造成任务堵塞

```
'打开输出点 10，延时 100ms 打开输出点 11
'由于该任务还包含其他功能，要求不能使用 WA 语句（造成堵塞）

DIM l_i_delaytime AS INTEGER
l_i_delaytime = 100
DIM l_i_step AS INTEGER

WHILE TRUE

  SELECT_CASE l_i_step
    CASE 0 '打开 OP10
      OP( 10, ON )
      l_i_ticks_preval = TICKS
      l_i_step = 10
    CASE 10 '延时
      l_i_waittime = ABS( l_i_ticks_preval - TICKS ) * SERVO_PERIOD / 1000
      IF l_i_waittime > l_i_delaytime THEN
        OP( 11, ON )
        l_i_step = 20
      ENDIF
  END_CASE
WEND
```

ESTUN & TRIO

3.11.4控制器时间-TIME

功能描述	读写系统时间，单位为 s； 可读，可写；
示例	写控制器时间 '写系统时间 0 时 10 分 50s TIME = 00: 10: 50 读控制器时间 '读系统时间，将小时存储至 VR(0) 分存储至 VR(1) 秒存储至 VR(2) l_i_systime = TIME VR(0) = INT(l_i_systime / 3600) '除以 3600 取整 VR(1) = INT((l_i_systime - VR(0) * 3600) / 60) '减去小时部分，除以 60 取整 VR(2) = (l_i_systime - VR(0) * 3600) MOD 60 '减去小时部分，除以 60 取模

3.11.5控制器时间-TIME$

功能描述	读写系统时间，返回值为字符串； 可读，可写；

示例	读控制器时间
	'窗口 0 中打印 LOG 信号，需要添加发生时间
	IF IN(0) = ON THEN
	PRINT TIME$ + "0 号输入导通"
	ENDIF

3.12 字符串相关指令

3.12.1ASC

指令格式	value = ASC("string")
功能描述	返回 ASC 码字符对应的数值
实例	返回字符 A 的 ASC 码存储至 VR(0); VR(0) = ASC("A") 返回字符串变量中的第 3 个字符的 ASC 码存储至 VR(0);

ESTUN & Trio

	VR(0) = ASC("ABC", 2)

3.12.2CHR

指令格式	value = CHR(number[, format])
功能描述	返回数值对应的 ASC 码字符； **value**：字符串 **number**：数值 **format**：转换方式 0：ASCII 1：UNICODE UTF-8 encoded
实例	输入回车加换行的 ASC 码，返回回车换行； DIM send_mes AS STRING(100) send_mes = "MESSAGE" + CHR(13) + CHR(10)

3.12.3DATE$

功能描述	返回控制器当前日期
实例	PRINT DATE$ 22/Nov/2020

3.12.4DAY$

功能描述	返回当前日期（周几）
实例	>>print day$ Sunday

3.12.5HEX

指令格式	value = HEX(number)
功能描述	返回数值的 16 进制表达，并存储至字符串中； **number**：数值 **value**：字符串变量存储数值的 16 进制表达式
实例	>>print hex(255) FF

ESTUN & Trio

3.12.6INSTR

<table>
<tr><td>指令格式</td><td>INSTR(<offset index,>string, search string<,wild card char>)</td></tr>
<tr><td>功能描述</td><td>在字符串中查询指定字符的位置；
<table>
<tr><td>返回值</td><td>查询对象在被查询字符串中的位置，如果未查询到，返回值为-1</td></tr>
<tr><td>Offset index</td><td>可选参数，指定开始查询位置，未设置则从第一个字符开始查询</td></tr>
<tr><td>String</td><td>被查询字符串</td></tr>
<tr><td>Search string</td><td>查询对象</td></tr>
<tr><td>Wild card char</td><td>屏蔽符号</td></tr>
</table></td></tr>
<tr><td>实例</td><td>DIM str1 AS STRING(32)
str1 = "TRIO MOTION TECHNOLOGY"
PRINT INSTR(str1, "MOTION") 'value = 5
PRINT INSTR(6, str1, "MOTION") 'value = -1
PRINT INSTR("Value = 123.45E10", "###.##E##", "#") 'value = 8
PRINT INSTR("this is my string", "is * y", 42) 'value = 5
PRINT INSTR(3, str1, "IO") 'value = 8</td></tr>
</table>

3.12.7LCASE

指令格式	LCASE(string)
功能描述	将字符串字符转换为小于字符；
实例	>>print LCASE("AAAA") aaaa

3.12.8LEFT

<table>
<tr><td>指令格式</td><td>LEFT(string, length)</td></tr>
<tr><td>功能描述</td><td>从字符串左侧开始截取字符，长度由 length 参数指定；
<table>
<tr><td>String</td><td>操作字符串</td></tr>
<tr><td>length</td><td>长度</td></tr>
</table></td></tr>
<tr><td>实例</td><td>>>PRINT LEFT("TRIO MOTION TECHNOLOGY", 4)
TRIO</td></tr>
</table>

ESTUN & Trio

3.12.9LEN

指令格式	LEN(string)
功能描述	返回字符串的长度；
实例	>>PRINT LEN("TRIO MOTION TECHNOLOGY") 22

3.12.10 MID

指令格式	MID(string, start[, length])
功能描述	从字符串中间开始截取字； **String** 操作字符串 **start** 开始位置 **length** 截取长度
实例	>>PRINT MID("TRIO MOTION TECHNOLOGY", 5, 6) MOTION

3.12.11 RIGHT

指令格式	RIGHT(string, length)
功能描述	从字符串右侧开始截取字符，长度由 length 参数指定； **String** 操作字符串 **length** 截取长度
实例	>>PRINT RIGHT("TRIO MOTION TECHNOLOGY", 10) TECHNOLOGY

3.12.12 STR

指令格式	STR(value[,precision[,width]])
功能描述	将数值转换成字符串； **Value** 需要转换的数值 **Precision** 字符总长度（默认值=5） **width** 小数部分长度（默认值=0）

ESTUN & Trio

实例	>>print str(1.2,10) 1.2000000000

3.12.13 UCASE

指令格式	UCASE(string)
功能描述	所有字符转换成大写形式；
实例	>>PRINT UCASE("Trio Motion Technology") TRIO MOTION TECHNOLOGY

3.12.14 VAL

指令格式	VAL(string)
功能描述	将字符串转换成数值； 如果字符串中无数值，则返回 0；
实例	>>PRINT VAL("123.456") 123.4560 >>PRINT VAL("AA") 0.0000

ESTUN & Trio

3.13 多任务

3.13.1相关概念介绍

3.13.1.1 多任务

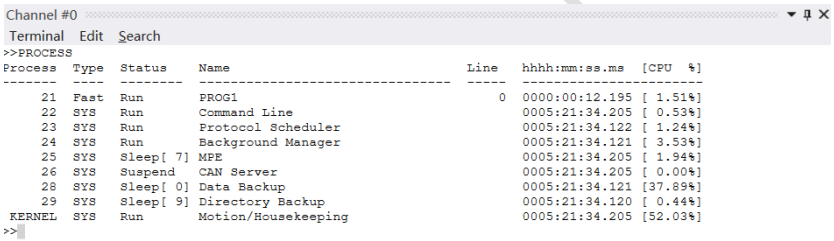
多任务处理是指控制器可以将全部任务按照内部的设定规则，在一个伺服周期内执行多个任务。

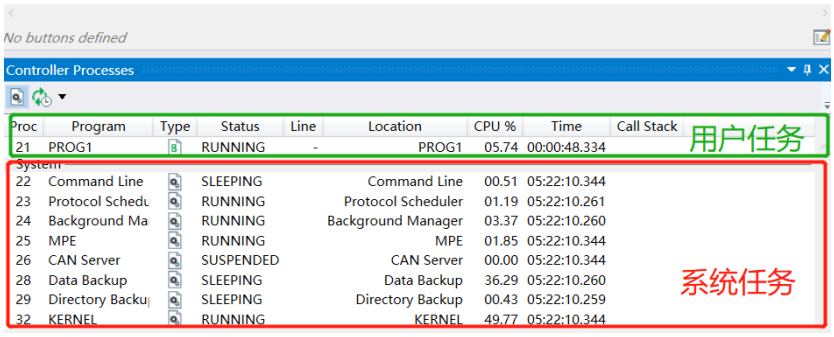
好处是用户可以根据项目需求，将一个大的工程细化成不同功能，合理的分配至不同任务进行处理，实现控制器资源的最大利用。

控制器任务由系统任务、用户任务组成。

按照功能分类，用户任务又可以分为上电初始化任务（执行一次），循环任务（一直运行）。

按照优先级，用户任务又可以分为快速任务、慢速任务。





3.13.1.2 用户任务

用户任务，运行用户程序的载体，可运行.BAS 程序、IEC 程序、HMI 程序、远程程序。遵循以下规则：

- 每个任务最多可运行一个程序；
- 一个程序文件可以在多个用户任务中同时运行；
- 程序文件可以手动运行在任务，也可以通过指令运行在任务，见任务的运行；

- 用户任务可分为快速任务和慢速任务；
- 用户任务的资源分配遵循一定规则，详见任务资源分配；
- 用户任务的状态可以通过相关指令读取，详见任务状态读取；
- 可以将部分任务设置为上电自动运行，详见操作见任务自启动设置；
- 不同控制器，用户任务最大数不一致；

表 3-1 不同控制器支持的任务数量和进程号

控制器型号	最大用户任务/进程号
MC403	6/0-5
MC405/EU404	10/0-9
其他	22/0-21

3.13.1.3 快速任务和慢速任务

默认设置下，控制器用户任务号最高的两个任务为快速任务，其他为慢速任务。

表 3-2 不同控制器的快速任务进程号

控制器型号	快速任务号
MC403	4/5
MC405/EU404	8/9
其他	20/21

当同时有程序在快速任务和慢速任务上执行时，系统会将用户任务资源分配为两大部分，一部分用于快速任务之间的分配，一部分用于慢速任务的分配，因为快速任务个数少，平均分配到每个任务的资源相对较多。

快速任务还有一个特点就是每个伺服周期均会执行。

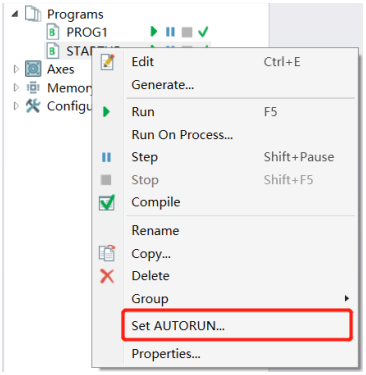
3.13.1.4 可执行文件

可独立运行的文件，包括.bas 文件、.hmi 文件、.iec 文件。比如，.basic lib 文件只是函数集，无法独立运行。

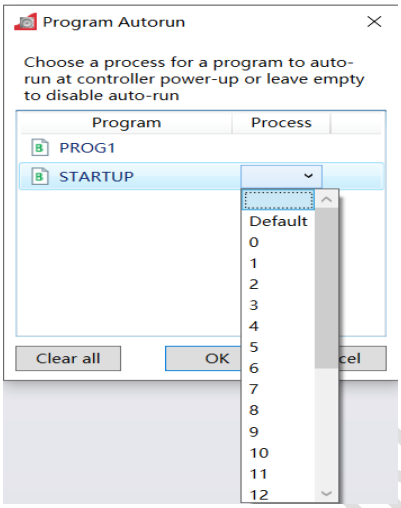
3.13.2上电自启动任务设置

上电自动运行任务由用户指定，具体操作如下：

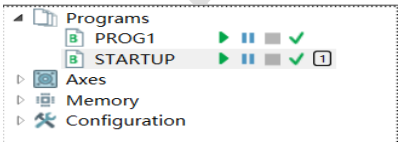
1) 选中需要设置控制器上电后自动运行的程序文件，右键弹出菜单，点击”Set AUTORUN …”；



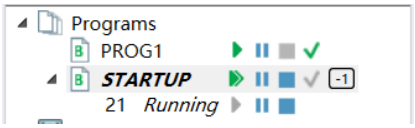
2) 选择程序上电自动执行的进程号；



3) 点击 OK 按钮确认，返回界面，查看是否设置成功。如下图，表示控制器上电后将，运行程序 STARTUP，其他程序的运行可以通过该程序调度（启动、停止），或在其他运行的程序中进行调度。



4) 如果将任务上电运行号码设置为默认，如下图，该程序在当前空闲最高任务号运行；



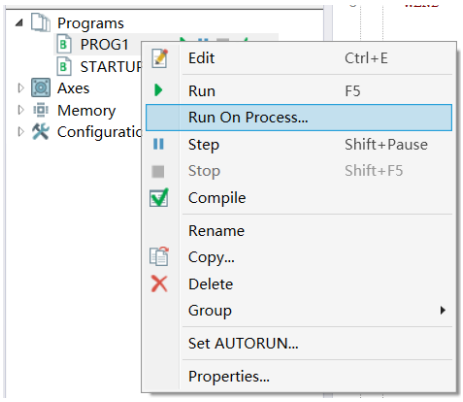
3.13.3手动运行任务设置

调试过程中，需要让某个程序手动运行在某个任务。

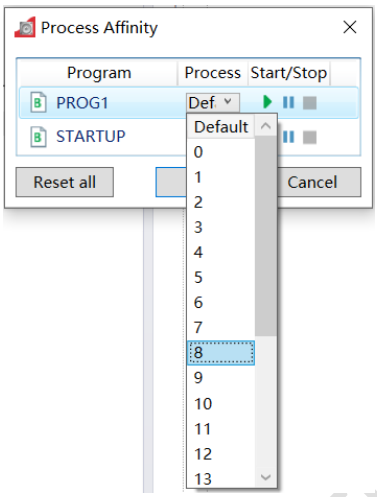
比如：程序文件 PROG1.BAS 手动运行在任务 8，具体操作过程如下；

1) 选中程序文件，右键弹出菜单，点击” Run On Process… ”；

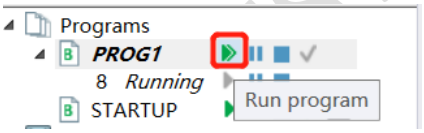
ESTUN & Trio



2) 手动设置运行的任务号；

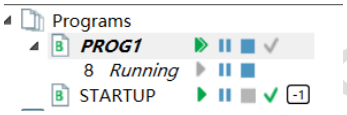


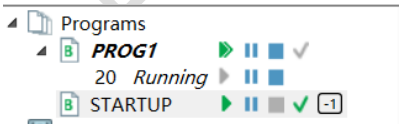
3) 点击 OK 按钮，确认；点击运行按钮，程序运行在 8 号任务。



ESTUN & Trio

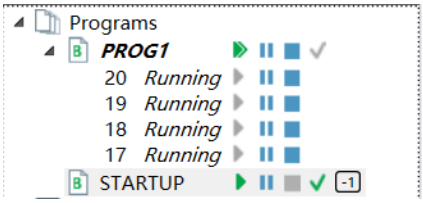
3.13.4任务的自动运行（RUN）

<table>
<tr><td>指令语法</td><td>RUN ["program" [, process]]</td></tr>
<tr><td>功能描述</td><td>运行指定程序文件在指定任务；
<table>
<tr><td>program</td><td>需要运行的程序名</td></tr>
<tr><td>process</td><td>分配的任务号，如果未写该参数，程序将在当前可执行任务中的最高任务运行</td></tr>
</table></td></tr>
<tr><td>示例</td><td>1. 需要程序初始化完成后运行 PROG1 程序在 8 号任务；
STARTUP 程序中：
RUN "PROG1", 8
运行效果：
 

2. 运行 PROG1 程序，但不指定任务号；
STARTUP 程序：
RUN "PROG1"
执行效果：
 

因为在执行 RUN 指令时，STARTUP 程序运行在 21 号任务，所以当前可执行的最高任务号为 20，所以 PROG1 在 20 号任务执行；
3. 当对一个程序多次触发执行时，如果不分配任务号，那么该程序将在多个任务同时执行；
RUN "PROG1"
RUN "PROG1"
RUN "PROG1"
RUN "PROG1"
执行结果：</td></tr>
</table>

ESTUN & Trio



因为一个程序文件可以同时在多个任务执行，可以通过 PROCNUMBER 知道程序文件当前正在执行的任务号是多少。

```
通道#0
端口  编辑  查找
>>当前正在执行的任务号是：20
当前正在执行的任务号是：19
当前正在执行的任务号是：18
当前正在执行的任务号是：17
```

4. 当一个任务正在执行时，使用 RUN 指令往这个任务加载程序，运行会出错。

```
RUN "PROG1",8

RUN "PROG1",8
```

运行结果：

```
RUN "PROG1", 8
RUN "PROG1", 8
Program(s) running
```

关于如何避免该错误，将在 PROC_STATUS 中介绍。

3.13.5任务停止-STOP

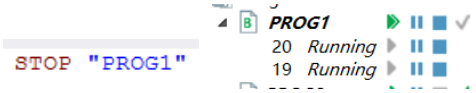
指令语法	`STOP "progname",[process_number]`
功能描述	停止指定程序运行，当在这个程序中执行 STOP 命令，程序在 STOP 这一行立即停止； 当在一个程序文件中对其他程序触发停止，其他程序立即停止； 如果一个程序文件在多个任务同时执行，可以通过指定 process_number 停止某个任务运行； 如果一个程序文件仅在 1 个任务执行，无需指定 process_number；

示例	如何在 PROG2 程序中提前停止 PROG2 的运行；
	`PRINT "当前执行行数1"`
	`PRINT "当前执行行数2"`
	`PRINT "当前执行行数3"`
	`STOP`
	`PRINT "当前执行行数4"`
	Channel #0
	Terminal Edit Search
	>>当前执行行数1
	当前执行行数2
	当前执行行数3

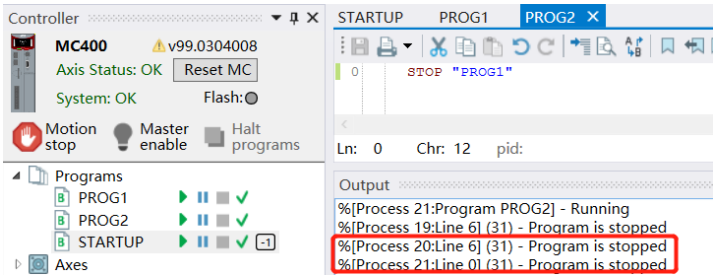
ESTUN & Trio

1) 程序执行到 STOP 就停止，后续命令将不执行.

在 PROG2 程序中停止 PROG1 程序运行；

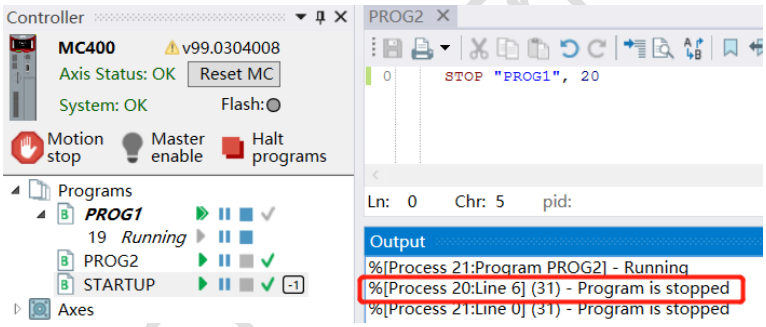


执行该语句，运行效果为：



2) 所有任务中，如果正在执行 PROG1，全部停止；

PROG1 程序正在任务 20 和任务 21 中执行，需要在 PROG2 中触发停止命令，停止在 20 号任务中执行的程序 PROG1,21 号任务中的继续运行；



3.13.6任务停止-HALT

指令语法	Halt
功能描述	停止当前正在运行的所有用户任务，不包含远程任务、HMI 任务；
备注	HALT 指令只会停止所有用户任务，但不会停止轴运动；

3.13.7任务的状态查看

3.13.7.1 PROCESS

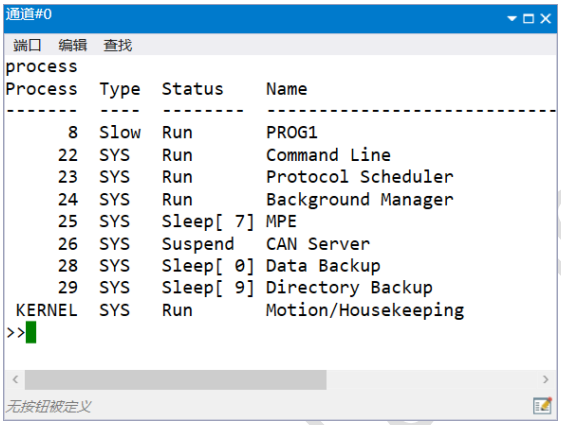
指令语法	PROCESS

功能描述

在窗口 0 中打印当前正在运行任务的状态，该命令仅窗口 0 运行有效；

PROCESS	任务号码
Type	任务类型（系统任务、快速任务、慢速任务）
Status	任务状态
Program	任务运行的程序文件名
Line	执行 PROCESS 命令时，当前任务正在执行行
Time	当前程序在该任务的运行时间
CPU	当前任务 CPU 资源占比

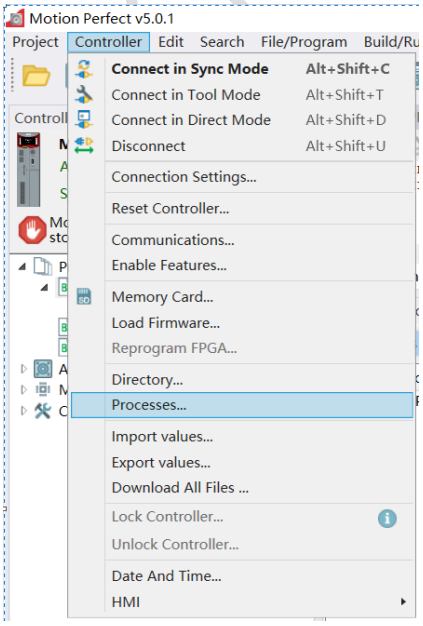
示例



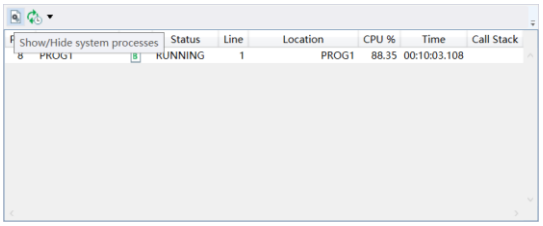
其他

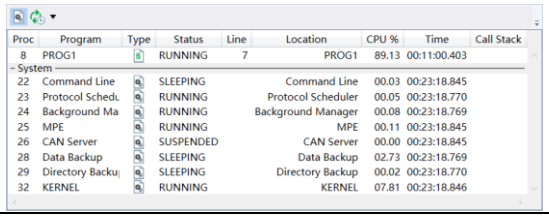
用户也可以打开 Montion Perfect 软件提供的对话框，监控任务运行状态，步骤如下：

工具栏“controller”，在弹出选项中，点击“Process…”

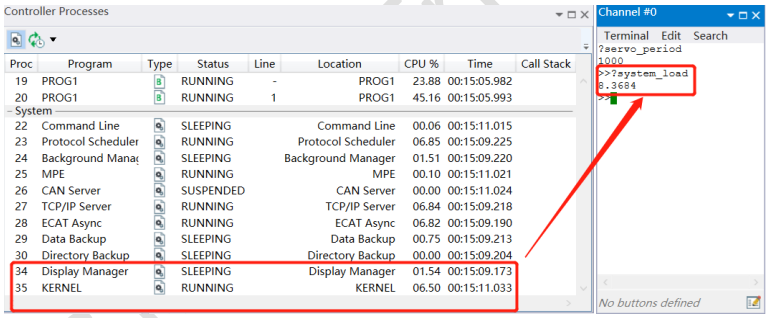
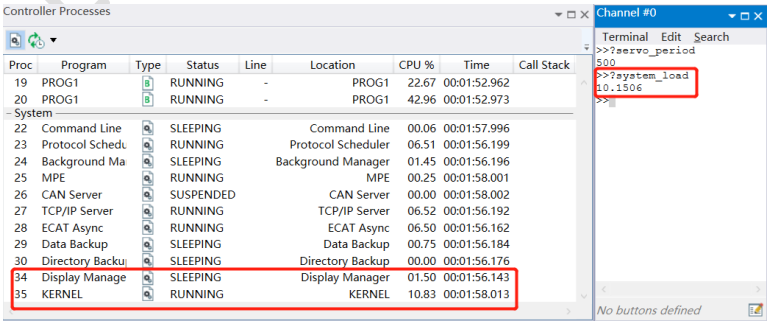


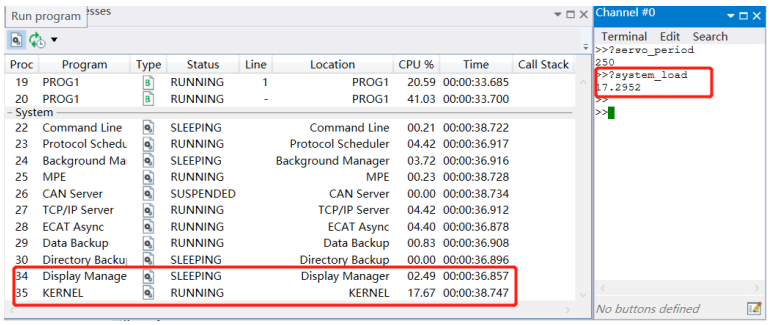
点击 按钮，可显示系统任务；





3.13.7.2 SYSTEM_LOAD

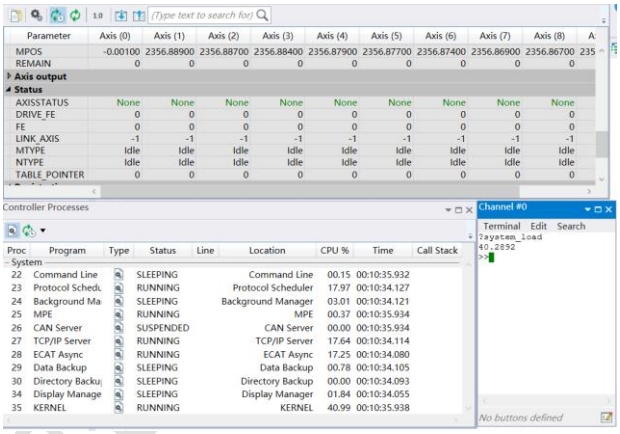
功能描述	系统参数，只读； 反馈当前运动内核及 ETHERCAT 实时通讯占用当前处理核的 CPU 比例；
遵循规则	1. 相同条件下，伺服周期（SERVO_PERIOD）设定值越小，系统负载越大；  Servo_period =1000  Servo_period =500

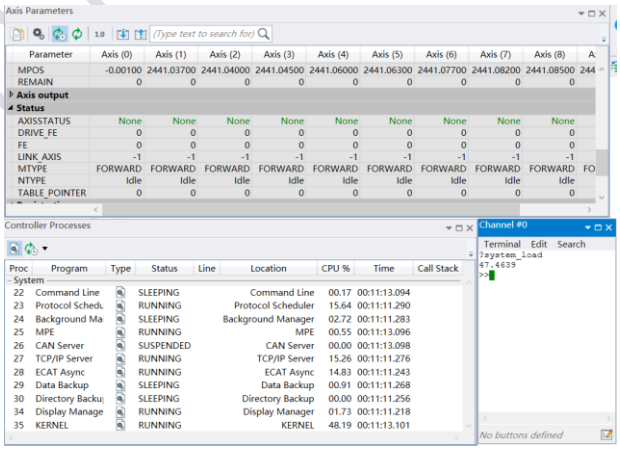


Servo_period =250

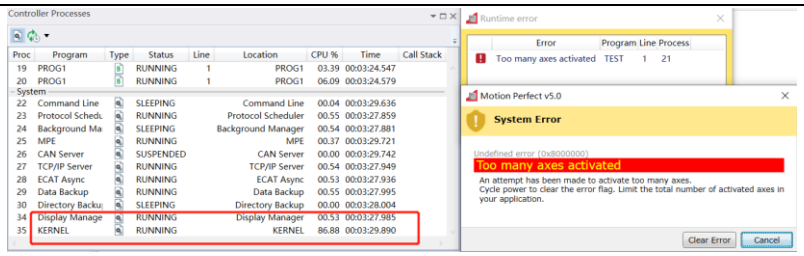
2. 相同情况下，EC 总线轴数越多，系统负载越大；

3. 相同情况下，激活运动轴数越多，系统负载越大；即使轴当前无运动，系统负载率不会降低；

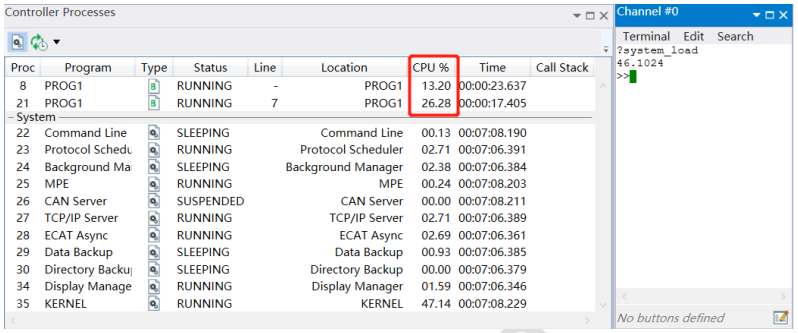




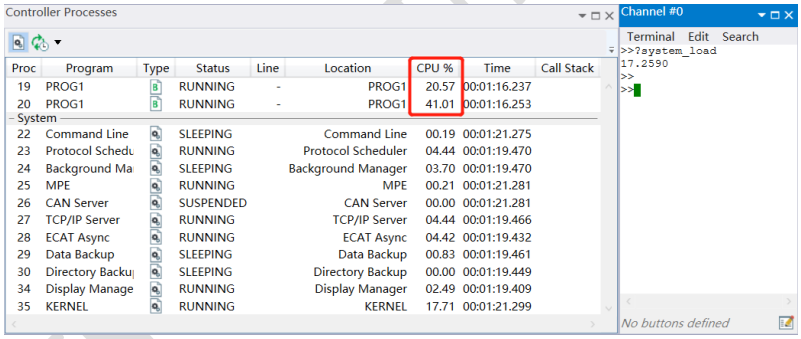
4. 单核控制器系统负载率应该保证在 55%以内，否则，将出现系统故障；



5. 单核控制器系统负载率高，将会使用户任务执行效率降低；



System_load = 46.1024



System_load = 17.259

3.13.7.3 SYSTEM_LOAD_MAX

功能描述	系统参数，只读； 反馈当前运动内核及 ETHERCAT 实时通讯占用当前处理核的最大 CPU 比例；

3.13.7.4 PROC_STATUS

功能描述

任务参数（只读），

返回指定任务的运行状态，通过 PROC(X)指定读取任务号；

返回值	含义
0	STOPPED-无程序在指定任务运行
1	RUNNING-存在程序或者 IEC TASK 在指定任务运行
2	STEPPING-单步运行中
3	PAUSED-暂停状态（比如：执行到断点处）
4	PAUSING-准备进入暂停状态
5	STOPPING-准备进入 STOP 状态
6	SUSPEND
7	SUSPENDED
8	SLEEP-任务休眠状态（比如：执行 WA, WAIT IDLE）

示例

1. 通过读取任务状态，判断是否执行任务；

正确写法：通过判断任务 8 的状态，控制 RUN 语句的执行；

PROG_A.BAS

```
'1 ）要求在程序 a 中，当输入信号 0 状态为 on 时，触发程序 b 执行在任务 10 中执行，_
'  已知程序 a 文件名为 prog_a.bas, 程序 b 文件名为 prog_b.bas;
'2 ）当 prog_b 正在运行时，输入信号为 on，程序 prog_a 和 prog_b 的状态均不受影响;
WHILE TRUE
  IF IN( 0 ) = ON THEN
    IF PROC_STATUS PROC( 8 ) = 0 THEN
      PRINT "运行程序 PROG_B"
      RUN "PROG_B", 8
    ELSE
      'print "程序 PROG_B 正在运行中"
    ENDIF
  ENDIF
WEND
```

错误写法：

```
WHILE TRUE
  IF IN( 0 ) = ON THEN
    PRINT "运行程序 PROG_B"
    RUN "PROG_B", 8
```

ESTUN & Trio

```
  ENDIF
WEND
```

运行结果：

```
WHILE TRUE

    IF IN( 0 ) = ON THEN

        PRINT "运行程序PROG_B"
        RUN "PROG_B", 8
Program(s) running
    ENDIF
```

如上，如果 8 号任务正在运行，再次往 8 号任务加载新的程序，当前正在执行触发运行的程序报错，停止运行。

3.13.7.5 PROC_LINE

功能描述	任务参数（只读）； 返回指定任务当前正在运行行数，通过 PROC(X)指定读取任务号；

3.13.8任务资源的分配原则

控制器不同任务对于系统资源 CPU 的占用率查看，可参考 PROCESS。

不同型别控制器 CPU 核数不一致，不同核数的控制器任务对系统资源的占用原则存在一定差异。用户需要根据分配原则，正确的分配不同任务 CPU 的占用率。

单核控制器	MC403/MC405/MC508/EU404/MC4N-ECAT/MC4N-RTEX/MC664
双核控制器	NANO/MC6N
四核控制器	MC664-X

NOTE:MC664 与 MC664-X 最大的区别在于 CPU 核数不一致，属于硬件差别，所以无法通过软升级方式将 MC664 升级至 MC664-X。

3.13.8.1 单核控制器任务 CPU 分配原则

1) 单核控制器的单位为 1 个伺服周期，一个伺服周期内，控制器将处理以下任务；

ESTUN & Trio



NOTE:单核控制器优先分配系统负载部分；

2) 用户任务和系统任务部分遵循一定的分别原则，现假设系统负载部分比例为 A，用户任务为 B,系统任务为 C，那么他们之间遵循的原则是：

$$A + B + C = 100$$
$$B = 3C$$

```
>>PROCESS
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  -------------------
      6  Slow  Run       PROG_B                                4  0000:00:05.797 [23.87%]
     21  Fast  Run       PROG_A                                3  0000:00:16.030 [45.07%]
     22  SYS   Run       Command Line                             0000:00:27.990 [ 0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:00:26.198 [ 6.86%]
     24  SYS   Sleep[ 0] Background Manager                       0000:00:26.193 [ 1.40%]
     25  SYS   Sleep[ 7] MPE                                      0000:00:27.995 [ 0.11%]
     26  SYS   Suspend   CAN Server                               0000:00:27.996 [ 0.00%]
     27  SYS   Run       TCP/IP Server                            0000:00:26.190 [ 6.86%]
     28  SYS   Run       ECAT Async                               0000:00:26.162 [ 6.84%]
     29  SYS   Sleep[ 0] Data Backup                              0000:00:26.183 [ 0.75%]
     30  SYS   Sleep[ 0] Directory Backup                         0000:00:26.174 [ 0.01%]
     34  SYS   Sleep[ 0] Display Manager                          0000:00:26.141 [ 1.54%]
KERNEL   SYS   Run       Motion/Housekeeping                      0000:00:28.002 [ 6.64%]
```

如上：

$$A = 6.64 + 1.54 = 8.18$$
$$B = 23.87 + 45.07 = 68.94$$
$$C = 0.07 + 6.86 + 1.4 + 0.11 + 0 + 6.86 + 6.84 + 0.75 = 22.85$$

可验证：

$$A + B + C = 100$$
$$B = 3C$$

3) 用户任务占比为 B，快速任务占比为 B1，慢速任务占比为 B2，默认情况下，满足以下规则：

$$B = B1 + B2$$
$$B1 \approx 2 * B2$$

```
PROCESS
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
      6  Slow  Run       PROG_B                                1  0000:07:12.795 [ 7.95%]
     18  Slow  Run       PROG_B                                1  0000:00:03.975 [ 7.95%]
     19  Slow  Run       PROG_B                                1  0000:00:06.385 [ 7.97%]
     20  Fast  Run       PROG_B                                1  0000:00:09.366 [21.21%]
     21  Fast  Run       PROG_A                                3  0000:07:23.031 [23.84%]
     22  SYS   Run       Command Line                             0000:07:34.992 [ 0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:07:33.200 [ 6.82%]
     24  SYS   Sleep[ 0] Background Manager                      0000:07:33.196 [ 1.52%]
     25  SYS   Sleep[ 7] MPE                                     0000:07:34.996 [ 0.10%]
     26  SYS   Suspend   CAN Server                             0000:07:34.999 [ 0.00%]
     27  SYS   Run       TCP/IP Server                          0000:07:33.193 [ 6.83%]
     28  SYS   Run       ECAT Async                             0000:07:33.165 [ 6.80%]
     29  SYS   Run       Data Backup                            0000:07:33.186 [ 0.77%]
     30  SYS   Sleep[ 9] Directory Backup                       0000:07:33.177 [ 0.00%]
     34  SYS   Sleep[ 0] Display Manager                        0000:07:33.145 [ 1.53%]
 KERNEL  SYS   Run       Motion/Housekeeping                    0000:07:35.006 [ 6.66%]
```

如上：

$B1 = 21.21 + 23.84 = 45.05$

$B2 = 7.95 + 7.95 + 7.97 = 23.87$

可知：

$B1 \approx 2 * B2$

4) 如果程序均处于 RUNNING 状态，快速任务平均分配快速任务占用资源，慢速任务平均分配占用资源；

```
PROCESS
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
      6  Slow  Run       PROG_B                                1  0000:07:12.795 [ 7.95%]
     18  Slow  Run       PROG_B                                1  0000:00:03.975 [ 7.95%]
     19  Slow  Run       PROG_B                                1  0000:00:06.385 [ 7.97%]
     20  Fast  Run       PROG_B                                1  0000:00:09.366 [21.21%]
     21  Fast  Run       PROG_A                                3  0000:07:23.031 [23.84%]
     22  SYS   Run       Command Line                             0000:07:34.992 [ 0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:07:33.200 [ 6.82%]
     24  SYS   Sleep[ 0] Background Manager                      0000:07:33.196 [ 1.52%]
     25  SYS   Sleep[ 7] MPE                                     0000:07:34.996 [ 0.10%]
     26  SYS   Suspend   CAN Server                             0000:07:34.999 [ 0.00%]
     27  SYS   Run       TCP/IP Server                          0000:07:33.193 [ 6.83%]
     28  SYS   Run       ECAT Async                             0000:07:33.165 [ 6.80%]
     29  SYS   Run       Data Backup                            0000:07:33.186 [ 0.77%]
     30  SYS   Sleep[ 9] Directory Backup                       0000:07:33.177 [ 0.00%]
     34  SYS   Sleep[ 0] Display Manager                        0000:07:33.145 [ 1.53%]
 KERNEL  SYS   Run       Motion/Housekeeping                    0000:07:35.006 [ 6.66%]
```

如上，20 和 21 号任务占比均约等于 $B1/2 \approx 22$，6、18/19 任务 CPU 占比均约等于 $B2/3 \approx 7.95$。

5) 如果系统仅一个快速任务正在运行，那么该快速任务将全部占用快速任务资源；

```
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
      6  Slow  Run       PROG_B                                1  0000:15:24.780 [ 7.95%]
     18  Slow  Run       PROG_B                                1  0000:08:15.961 [ 7.95%]
     19  Slow  Run       PROG_B                                1  0000:08:18.371 [ 7.97%]
     21  Fast  Run       PROG_A                                3  0000:15:35.015 [45.06%]
     22  SYS   Run       Command Line                             0000:15:46.976 [ 0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:15:45.184 [ 6.82%]
     24  SYS   Sleep[ 0] Background Manager                       0000:15:45.179 [ 1.51%]
     25  SYS   Sleep[ 7] MPE                                      0000:15:46.979 [ 0.10%]
     26  SYS   Suspend   CAN Server                               0000:15:46.980 [ 0.00%]
     27  SYS   Run       TCP/IP Server                            0000:15:45.174 [ 6.82%]
     28  SYS   Run       ECAT Async                               0000:15:45.147 [ 6.80%]
     29  SYS   Sleep[ 0] Data Backup                              0000:15:45.169 [ 0.77%]
     30  SYS   Sleep[ 9] Directory Backup                         0000:15:45.159 [ 0.00%]
     34  SYS   Run       Display Manager                          0000:15:45.126 [ 1.55%]
 KERNEL  SYS   Run       Motion/Housekeeping                      0000:15:46.987 [ 6.64%]
```

6) 如果系统中只有慢速任务在运行，无快速任务运行，那么慢速任务平均分配所有用户任务；

```
process
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
      6  Slow  Run       PROG_B                                1  0000:17:19.875 [21.30%]
     18  Slow  Run       PROG_B                                1  0000:10:11.056 [23.90%]
     19  Slow  Run       PROG_B                                4  0000:10:13.469 [23.88%]
     22  SYS   Run       Command Line                             0000:17:42.072 [ 0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:17:40.282 [ 6.89%]
     24  SYS   Sleep[ 0] Background Manager                       0000:17:40.277 [ 1.39%]
     25  SYS   Sleep[ 7] MPE                                      0000:17:42.077 [ 0.10%]
     26  SYS   Suspend   CAN Server                               0000:17:42.078 [ 0.00%]
     27  SYS   Run       TCP/IP Server                            0000:17:40.273 [ 6.89%]
     28  SYS   Run       ECAT Async                               0000:17:40.245 [ 6.87%]
     29  SYS   Sleep[ 0] Data Backup                              0000:17:40.266 [ 0.75%]
     30  SYS   Sleep[ 9] Directory Backup                         0000:17:40.257 [ 0.00%]
     34  SYS   Run       Display Manager                          0000:17:40.224 [ 1.51%]
 KERNEL  SYS   Run       Motion/Housekeeping                      0000:17:42.087 [ 6.47%]
```

7) 如果任务处于 SLEEP/PAUSE/STEP 状态，其 CPU 占比约为 0；

```
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
     19  Slow  Sleep[ 0] PROG_E                               2  0000:00:06.115 [ 0.00%]
     20  Fast  Pause     PROG_D                               0  0000:00:45.498 [ 0.00%]
     21  Fast  Pause     PROG_C                               0  0000:00:50.024 [ 0.00%]
     22  SYS   Run       Command Line                             0000:20:31.539 [ 0.25%]
     23  SYS   Run       Protocol Scheduler                       0000:20:29.746 [29.77%]
     24  SYS   Sleep[ 0] Background Manager                       0000:20:29.740 [ 3.87%]
     25  SYS   Sleep[ 7] MPE                                      0000:20:31.539 [ 0.17%]
     26  SYS   Suspend   CAN Server                               0000:20:31.540 [ 0.00%]
     27  SYS   Run       TCP/IP Server                            0000:20:29.733 [28.21%]
     28  SYS   Run       ECAT Async                               0000:20:29.704 [28.86%]
     29  SYS   Sleep[ 0] Data Backup                              0000:20:29.724 [ 0.84%]
     30  SYS   Sleep[ 9] Directory Backup                         0000:20:29.714 [ 0.00%]
     34  SYS   Sleep[ 0] Display Manager                          0000:20:29.681 [ 1.59%]
 KERNEL  SYS   Run       Motion/Housekeeping                      0000:20:31.542 [ 6.44%]
```

8) 通过 SLOTn_TIME 可以调节系统任务、快速任务、慢速任务的占比，详见 SLOT_TIME 的设置。

3.13.8.2 双核控制器任务 CPU 分配原则

双核控制器包含 MC6N/NANO 两种型号。

1) 单核控制器的单位为 1 个伺服周期，一个伺服周期内，控制器将处理以下任务；

   如下，双核处理器将 EC 实时处理及运动内核在内核 0 上运行，用户任务及系统任务在内核 1 运行；

   SYSTEM_LOAD 的大小不会影响用户任务的执行速率；

   可以通过 SCHEDULE_TYPE 将部分系统任务移动至内核 0，提高 CPU 的利用率，详见 SCHEDULE_TYPE 设置；




```
>>process
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU:       %]
-------  ----  --------  --------------------------------  -----  -------------------------
     19  Slow  Run       PROG_B                                1  0000:00:03.768 [0 :   24.84%]
     20  Fast  Run       PROG_D                                0  0000:00:11.605 [0 :   24.76%]
     21  Fast  Run       PROG_C                                3  0000:00:12.762 [0 :   24.82%]
     22  SYS   Run       Command Line                             0000:03:22.785 [0 :    0.07%]
     23  SYS   Run       Protocol Scheduler                       0000:03:21.678 [0 :    5.54%]
     24  SYS   Sleep[ 0] Background Manager                       0000:03:21.677 [0 :    1.92%]
     25  SYS   Sleep[ 7] MPE                                      0000:03:22.788 [0 :    0.25%]
     26  SYS   Run       TCP/IP Server                            0000:03:21.675 [0 :    5.53%]
     27  SYS   Run       ECAT Async                               0000:03:21.660 [0 :    5.53%]
     28  SYS   Run       Data Backup                              0000:03:21.672 [0 :    5.52%]
     29  SYS   Sleep[ 9] Directory Backup                         0000:03:21.667 [0 :    0.00%]
 KERNEL  SYS   Run       Housekeeping                             0000:03:22.792 [0 :    1.22%]
 KERNEL  SYS   Run       Servo Scheduler                          0000:03:22.792 [1 :    2.37%]
>>
```

2) 用户任务和系统任务部分遵循一定的分别原则，现假设系统负载部分比例为 A，用户任务为 B,系统任务为 C，那么他们之间遵循的原则是：

$$A + B + C = 100$$
$$B = 3C$$

3) 用户任务占比为 B，快速任务占比为 B1，慢速任务占比为 B2，默认情况下，满足以下规则：

$$B = B1 + B2$$
$$B1 \approx 2 * B2$$

4) 如果程序均处于 RUNNING 状态，快速任务平均分配快速任务占用资源，慢速任务平均分配占用资源；

5) 如果系统仅一个快速任务正在运行，那么该快速任务将全部占用快速任务资源；

6) 如果系统中只有慢速任务在运行，无快速任务运行，那么慢速任务平均分配所有用户任务；

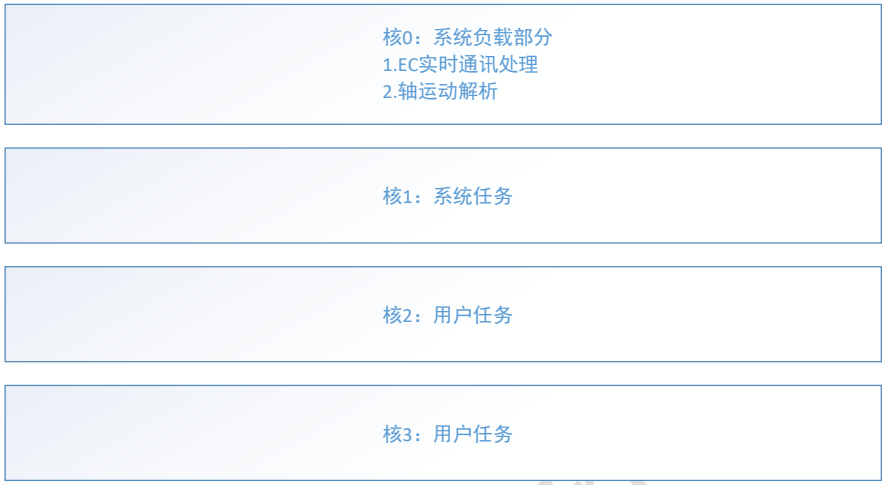
7) 如果任务处于 SLEEP/PAUSE/STEP 状态，其 CPU 占比约为 0；

8) 通过 SLOTn_TIME 可以调节系统任务、快速任务、慢速任务的占比，详见 SLOT_TIME 的设置。

3.13.8.3 四核控制器任务 CPU 分配原则

四核控制器包含 MC-664X。

1) 四核控制器的单位为 1 个伺服周期，一个伺服周期内，控制器将处理以下任务。四核控制器按照下图将各任务分配至不同内核；



如上，系统任务和运动内核、通讯处理运行在独立核，与用户任务相互不影响。

```
Process  Type  Status     Name                          Line   hhhh:mm:ss.ms  [CPU:      %]
     20  Fast  Run        PROGB                            0  0000:00:03.733 [3 : 100.00%]
     21  Fast  Run        PROGA                            0  0000:01:13.691 [2 : 100.00%]
     22  SYS   Run        Command Line                        0000:06:32.586 [0 :   0.25%]
     23  SYS   Run        Protocol Scheduler                  0000:06:28.474 [0 :  21.95%]
     24  SYS   Sleep[ 0]  Background Manager                  0000:06:28.472 [0 :   7.51%]
     25  SYS   Sleep[ 7]  MPE                                 0000:06:32.587 [0 :   1.70%]
     26  SYS   Suspend    CAN Server                          0000:06:32.587 [0 :   0.00%]
     27  SYS   Run        TCP/IP Server                       0000:06:28.469 [0 :  21.98%]
     28  SYS   Run        ECAT Async                          0000:06:28.453 [0 :  21.88%]
     29  SYS   Run        Data Backup                         0000:06:28.465 [0 :  21.95%]
     30  SYS   Sleep[ 9]  Directory Backup                    0000:06:28.460 [0 :   0.00%]
     33  SYS   Sleep[ 0]  Display Manager                     0000:06:28.447 [0 :   1.53%]
KERNEL   SYS   Run        Servo Scheduler                     0000:06:32.588 [1 :  14.70%]
```

2) 四核控制器 SLOTn_TIME 参数设置无效；

3) 四核控制器中，快速任务、慢速任务对 CPU 的占比受启动顺序影响；当运行第一个任务时，该运动加载至核 2，下一个任务加载至核 3，依次类推；

```
RUN "PROGA", 21
RUN "PROGB", 20
RUN "PROGC", 19
RUN "PROGD", 18
RUN "PROGE", 17
RUN "PROGF", 16
```

```
PROCESS
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU:      %]
-------  ----  --------  --------------------------------  -----  -----------------------------
     16  Slow  Run       PROGF                                 0  0000:00:05.502 [3 :  12.50%]
     17  Slow  Run       PROGE                                 0  0000:00:05.520 [2 :  12.50%]
     18  Slow  Run       PROGD                                 2  0000:00:05.530 [3 :  12.50%]
     19  Slow  Run       PROGC                                 0  0000:00:05.541 [2 :  12.50%]
     20  Fast  Run       PROGB                                 2  0000:00:05.547 [3 :  75.00%]
     21  Fast  Run       PROGA                                 0  0000:00:05.553 [2 :  75.00%]
     22  SYS   Run       Command Line                             0000:13:41.559 [0 :   0.25%]
     23  SYS   Run       Protocol Scheduler                       0000:13:37.446 [0 :  21.88%]
     24  SYS   Run       Background Manager                       0000:13:37.446 [0 :   7.68%]
     25  SYS   Sleep[ 7] MPE                                      0000:13:41.561 [0 :   1.75%]
     26  SYS   Suspend   CAN Server                               0000:13:41.561 [0 :   0.00%]
     27  SYS   Run       TCP/IP Server                            0000:13:37.443 [0 :  21.90%]
     28  SYS   Run       ECAT Async                               0000:13:37.426 [0 :  21.82%]
     29  SYS   Run       Data Backup                              0000:13:37.438 [0 :  21.90%]
     30  SYS   Sleep[ 9] Directory Backup                         0000:13:37.434 [0 :   0.00%]
     33  SYS   Sleep[ 0] Display Manager                          0000:13:37.421 [0 :   1.53%]
 KERNEL  SYS   Run       Housekeeping                             0000:13:41.562 [0 :   1.29%]
 KERNEL  SYS   Run       Servo Scheduler                          0000:13:41.562 [1 :  14.83%]
```

如果连续加载两个快速任务，两个快速任务被分配至两个核，那么在一个核内，快速任务的 CPU 占比是慢速任务的 3 倍。

3.13.8.4 SLOTn_TIME

功能描述	系统参数；MC_CONFIG 文件中设置； 包含 SLOT0_TIME、SLOT1_TIME、SLOT2_TIME、SLOT3_TIME；
设置原则	SLOTn_TIME 仅适用于单核控制器、双核控制器，不适用于四核控制器； SLOTn_TIME 在双核控制器中使用，要求控制器的固件版本在 2.300 及以上； SLOT1_TIME、SLOT2_TIME、SLOT3_TIME、SLOT4_TIME 的默认值为 25； SLOT0_TIME 决定了慢速任务的 CPU 占比，SLOT1_TIME +SLOT3_TIME 决定了快速任务的占比，SLOT2_TIME 决定了系统任务的占比； 伺服周期 SERVO_PERIOD=1000、2000 时，设定情况满足以下条件，设定值生效； $SLOT0_{TIME} + SLOT1_{TIME} + SLOT2_{TIME} + SLOT3_{TIME} = 100$ $SLOT0_{TIME} \geq 10$ $SLOT1_{TIME} \geq 10$ $SLOT2_{TIME} \geq 10$ $SLOT3_{TIME} \geq 10$ 伺服周期 SERVO_PERIOD=500 时，设定情况满足以下条件，设定值生效； $SLOT0_{TIME} + SLOT1_{TIME} = 100$ $SLOT2_{TIME} + SLOT3_{TIME} = 100$
实例	默认设置下，两个快速任务，两个慢速任务同时执行，CPU 占用率如下： Process Type Status Name Line hhhh:mm:ss.ms [CPU %] 18 Slow Run PROG_E 0 0000:00:07.415 [11.94%] 19 Slow Run PROG_D 0 0000:00:37.210 [11.94%] 20 Fast Run PROG_C 3 0000:00:39.770 [21.26%] 21 Fast Run PROG_B 1 0000:00:41.282 [23.85%] 22 SYS Run Command Line 0000:02:12.153 [0.07%] 23 SYS Run Protocol Scheduler 0000:02:10.362 [6.86%] 24 SYS Sleep[0] Background Manager 0000:02:10.358 [1.36%] 25 SYS Sleep[7] MPE 0000:02:12.158 [0.11%] 26 SYS Suspend CAN Server 0000:02:12.159 [0.00%] 27 SYS Run TCP/IP Server 0000:02:10.353 [6.87%] 28 SYS Run ECAT Async 0000:02:10.324 [6.86%] 29 SYS Sleep[0] Data Backup 0000:02:10.349 [0.78%] 30 SYS Sleep[9] Directory Backup 0000:02:10.339 [0.00%] 34 SYS Run Display Manager 0000:02:10.305 [1.54%] KERNEL SYS Run Motion/Housekeeping 0000:02:12.167 [6.57%] 降低快速任务占比至 30%，慢速任务提升至 45%，系统任务保持 25%，设

置如下：

Name	Modifier	Value
SLOT1_TIME		15
SLOT0_TIME		45
SLOT2_TIME		25
SLOT3_TIME		15

运行效果：

```
>>process
Process  Type  Status    Name                              Line   hhhh:mm:ss.ms  [CPU  %]
-------  ----  --------  --------------------------------  -----  ---------------------
     18  Slow  Run       PROG_E                                0  0000:00:03.272 [21.42%]
     19  Slow  Run       PROG_D                                0  0000:00:04.103 [21.42%]
     20  Fast  Run       PROG_C                                0  0000:00:04.822 [11.84%]
     21  Fast  Run       PROG_B                                4  0000:00:05.884 [14.38%]
     22  SYS   Run       Command Line                             0000:00:20.745 [ 0.08%]
     23  SYS   Run       Protocol Scheduler                       0000:00:18.953 [ 6.83%]
     24  SYS   Run       Background Manager                       0000:00:18.948 [ 1.51%]
     25  SYS   Sleep[ 7] MPE                                      0000:00:20.748 [ 0.09%]
     26  SYS   Suspend   CAN Server                               0000:00:20.750 [ 0.00%]
     27  SYS   Run       TCP/IP Server                            0000:00:18.945 [ 6.83%]
     28  SYS   Run       ECAT Async                               0000:00:18.916 [ 6.80%]
     29  SYS   Sleep[ 0] Data Backup                              0000:00:18.936 [ 0.76%]
     30  SYS   Sleep[ 9] Directory Backup                         0000:00:18.928 [ 0.00%]
     34  SYS   Sleep[ 0] Display Manager                          0000:00:18.895 [ 1.54%]
 KERNEL  SYS   Run       Motion/Housekeeping                      0000:00:20.757 [ 6.50%]
```

3.13.8.5 SCHEDULE_TYPE 的设置

功能描述

系统参数；

MC_CONFIG 文件中编辑；

BIT3	0	系统任务中的 TCP/IP 任务与其他系统任务在一个核
	1	系统任务中的 TCP/IP 任务在运动核运行
BIT4	0	系统任务中的 PROTROL Scheduler 任务与其他系统任务在一个核
	1	系统任务中的 PROTROL Scheduler 任务在运动核运行
BIT5	0	系统任务中的 ECAT Async 任务与其他系统任务在一个核
	1	系统任务中的 ECAT Async 任务在运动核运行
BIT6	0	2 个最高任务为快速任务
	1	所有任务均为慢速任务

注意事项

1. BIT3/BIT4/BIT5 功能均在 NANO 和 MC6N 控制器生效；
2. BIT3/BIT4/BIT5 功能设置需要在固件版本 2.300 及以上，如果在低版本中设置，会导致控制器无法与 PC 连接；
3. 将部分系统任务移动至运动内核，可以提高该任务的 CPU 占用率；
   - 将 ECAT Async 任务移动至运动内核，可以提高 ETHERCAT SDO 的读写速度；
   - 将 TCP/IP 任务移动至运动内核，控件通讯速率将得到提升，且通讯速率不受用户任务的 CPU 占比影响；

ESTUN & Trio

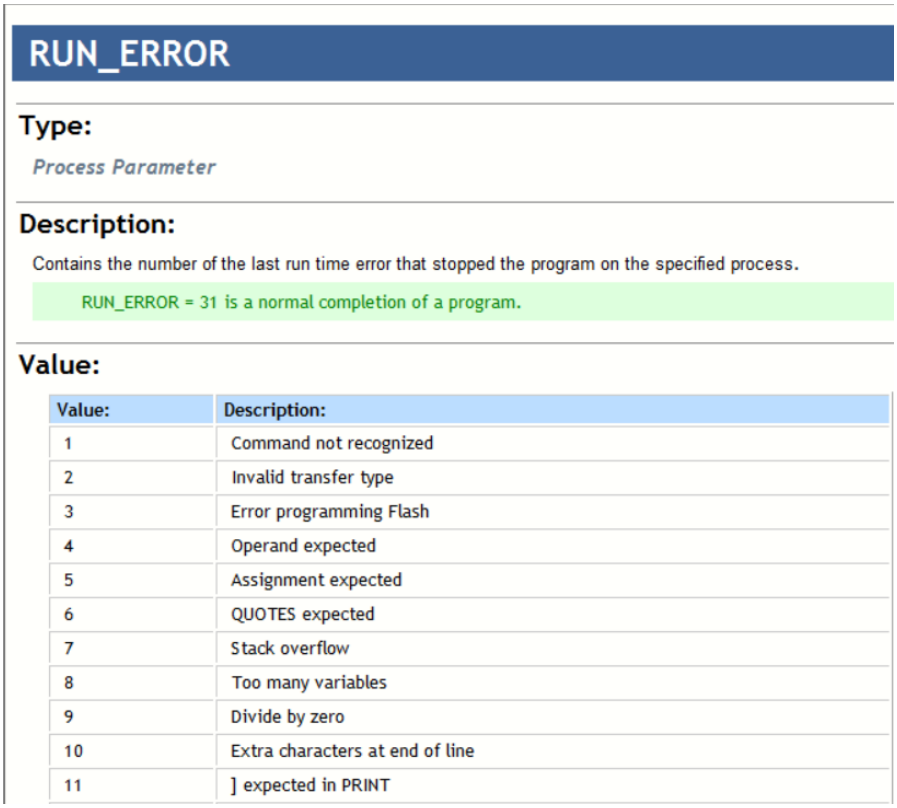
3.13.9常见任务操作错误及故障排查

3.13.9.1 任务的重复执行

任务的重复执行，在 PROC_STATUS 中讲解，并提供解决方法，请参考 PROC_STATUS。

3.13.9.2 RUN_ERROR

当程序在运行过程中，由于指令参数错误等原因，当前程序被停止运行，可以通过 RUN_ERROR 查看停止原因。



RUN_ERROR

Type:

Process Parameter

Description:

Contains the number of the last run time error that stopped the program on the specified process.

RUN_ERROR = 31 is a normal completion of a program.

Value:

Value:	Description:
1	Command not recognized
2	Invalid transfer type
3	Error programming Flash
4	Operand expected
5	Assignment expected
6	QUOTES expected
7	Stack overflow
8	Too many variables
9	Divide by zero
10	Extra characters at end of line
11	] expected in PRINT

更多请详见 TRIO BASIC 帮助文件。

常用的故障有：

1) 任务的重复触发；

2) 指令参数超限，比如说对虚拟轴执行 SDO 读写操作、数值操作错误（除 0、对 VR 的 BIT53 位及以上赋值）；

3) 数组操作超过数组定义长度；

3.13.9.3 ON BASICERROR GOTO

当程序运行时，发生错误，导致程序异常停止，为便于用于处理该错误或者定位编写 BUG，可使用 ON BASICERROR GOTO 语句。

参考下例：

```
'当系统发送错误，要求存储错误至VR，方便排查故障

'当发送运行错误，将跳转至标签error_store
ON BASICERROR GOTO error_store



WHILE TRUE

    '循环程序

    a = 10
    c = a / b

WEND


error_store:

'存储错误至VR(0)

VR( 0 ) = RUN_ERROR

STOP
```



上图中，b 由于未赋值为 0，所以出现错误，VR(0)中的值为 9，通过查询 RUN_ERROR，知道出现了除 0 错误。

9	Divide by zero

3.13.9.4 FLASH_LOG

通过 ON BASICERROR GOTO,用户可以知道该任务无故停止的原因，但无法定位至具体哪一行代码。

FLASH_LOG，功能为在窗口 0 打印系统日志。

系统日志内最多可存储最近的 4096 条，可存储程序异常停止的原因。同时也便于用户定位问题位置。

```
1232 07/Jan/2000 23:22:39 [    1280.968] MPE_MODE change (4)
1233 07/Jan/2000 23:22:39 [    1280.983] MPE_MODE change (0)
1234 07/Jan/2000 23:22:41 [    1282.143] Soft Restart (EX)
1235 07/Jan/2000 23:22:51 [       5.595] 432:03264@1000[0.22 99.0304026000 0109
1236 07/Jan/2000 23:22:52 [       6.438] MPE_MODE change (1)
1237 07/Jan/2000 23:22:53 [       7.053] MPE_MODE change (0)
1238 07/Jan/2000 23:22:53 [       7.068] MPE_MODE change (4)
1239 07/Jan/2000 23:22:53 [       7.677] MPE_MODE change (5)
1240 07/Jan/2000 23:23:07 [      21.416] Program 'BASICERROR' created
1241 07/Jan/2000 23:23:07 [      21.425] Program 'BASICERROR' saved to flash
1242 07/Jan/2000 23:32:26 [     580.799] BASICERROR@13[21]  Divide by zero
<END OF LOG>
```

3.13.10 特殊任务的设置

TRIO 的 HMI 界面程序、基于控件的远程任务、基于控件的文件传输任务、基于 API 的远程任务，均占有用户任务。

3.13.10.1 HMI_PROC

功能描述	系统参数：MC_CONFIG 文件中设置； 指定使用 MOTION PERFECT 编辑的 HMI 页面程序运行任务号； 如果不设置，HMI 程序运行时，将运行在当前空闲的最高任务； 如果未使用 TRIO HMI，该参数无需设置；

3.13.10.2 REMOTE_PROC

功能描述	系统参数：MC_CONFIG 文件中设置； 指定采用控件或者 DLL 形式访问控制器程序的任务号； 如果不设置，远程程序运行时，将运行在当前空闲的最高任务； 如果未使用控件，该参数无需设置；

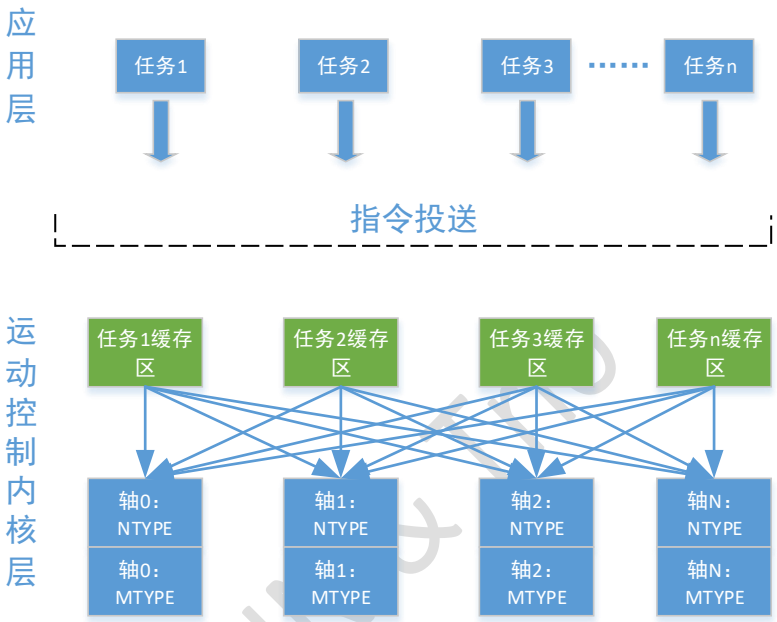
3.13.10.3 PCMCAT_PROC

功能描述	系统参数：MC_CONFIG 文件中设置； 指定采用 API 函数访问控制器程序的任务号； 如果不设置，远程程序运行时，将运行在当前空闲的最高任务； 如果未使用控件，该参数无需设置； 该参数仅适用于 PC_MCAT/PC_MCAT_S 控制器；

ESTUN & Trio

3.14 运动加载原则

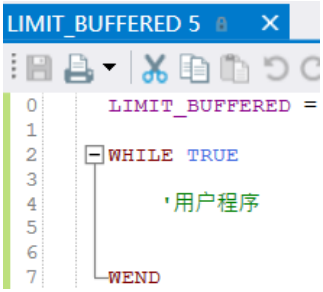
多任务系统中，每个任务均有独立的缓存，用于存储扫描到的轴运动指令。每个轴有两个独立的轴缓存区，可存储两条运动指令。MTYPE 表示当前正在执行的运动，NTYPE 表示即将执行的运动指令。



3.14.1任务缓存区

3.14.1.1 LIMIT_BUFFERED

指令语法	LIMIT_BUFFERED = N(1~66)
功能描述	设置任务缓存区大小，范围为 1~63，默认值为 1； 缓存区的大小，决定了任务同时最大存储轴运动指令。
示例	设置程序运行的任务的轴缓存区大小； LIMIT_BUFFERED 21 0 LIMIT_BUFFERED = 10 1 2 WHILE TRUE 3 4 '用户程序 5 6 7 WEND

	表示将 21 号任务的运动缓存区大小设置为 10；  表示将 5 号任务的运动缓存区设置为 10。
备注	当缓存区加载运动指令已满，同时该任务又执行到新的运动指令，任务会自动在这行休眠，直到缓存区有空间加载该运动指令； 应当根据实际情况设置缓存区大小，如果当前任务用于处理连续运动，应当根据连续运动的最大加载数设置缓存区；

3.14.1.2 PMOVE

指令语法	任务参数，可读
功能描述	反馈指定查询任务缓存区是否已满，通过 PROC 指令设置查找任务号；

示例	查看任务 0 当前缓存区是否已满，存储至 VR(0)
	`DIM serch_task AS INTEGER`
	`serch_task = 0`
	`DEFCONST task_full 1`
	`DEFCONST task_notfull 0`
	`WHILE TRUE`
	`VR( 0 ) = PMOVE PROC( serch_task )`
	`IF VR( 0 ) = task_full THEN`
	`PRINT "0号缓存区轴运动满"`
	`ENDIF`
	`IF VR( 0 ) = task_notfull THEN`
	`PRINT "0号缓存区轴运动未满"`
	`ENDIF`
	`WEND`

3.14.2轴运动缓存区

3.14.2.1 MTYPE

指令语法	轴参数，只读
功能描述	反馈轴当前正在执行的运动；

	用户可根据查看此参数，判断当前运动类型； 运动类型返回值详见表格轴运动类型反馈；

示例	如果当前轴缓存（MTYPE/NTYPE）为空，任务中执行一条运动指令，需要一个伺服周期，运动才能加载至 MTYPE；
	`BASE( 1 )`
	`UNITS = 100`
	`SPEED = 100`
	`ACCEL = 1000000`
	`DECEL = 1000000`
	`TICKS = 0 '任务时钟`
	`MOVE( 100 )`
	`WAIT IDLE '判断MTYPE中无运动`
	`PRINT "运动耗时="; ABS( TICKS )`
	上图中的写法，加载 MOVE(100)后立即判断 IDLE，很可能在当前伺服周期指令还未加载至 MTYPE，导致时序错误；
	`BASE( 1 )`
	`UNITS = 100`
	`SPEED = 100`
	`ACCEL = 1000000`
	`DECEL = 1000000`
	`TICKS = 0 '任务时钟`
	`MOVE( 100 )`
	`WA( 2 ) '增加延时，等待指令加载成功后判断`
	`WAIT IDLE '判断MTYPE中无运动`
	`PRINT "运动耗时="; ABS( TICKS )`

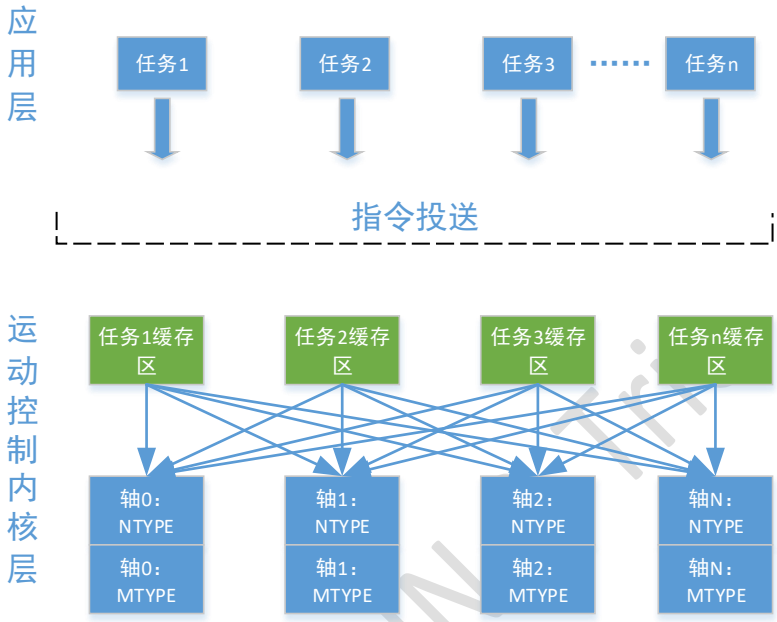
3.14.2.2 NTYPE

指令语法	轴参数，只读
功能描述	反馈轴即将执行的运动； 用户可根据查看此参数，判断即将运动类型； 运动类型返回值详见表格轴运动类型反馈；

3.14.2.3 MOVES_BUFFERED

指令语法	轴参数，只读
功能描述	反馈轴缓存区运动总数；

3.14.3加载规则及实例



1) 每个任务的缓存区大小可以通过 LIMIT_BUFFERED 进行设置；

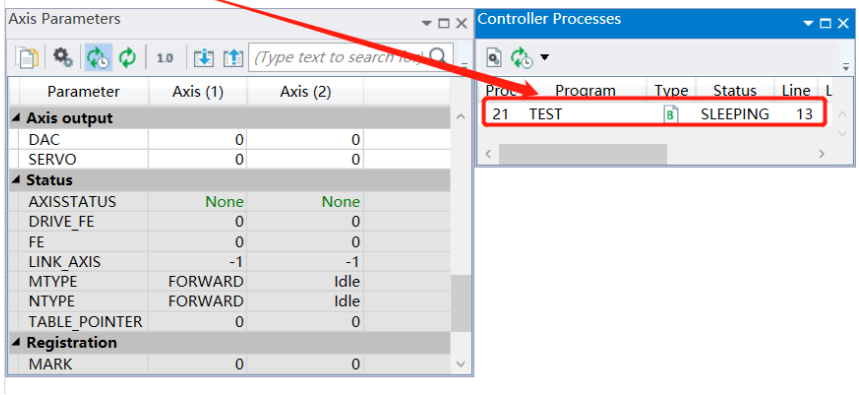
2) 每个任务的缓存区大小决定了最大加载轴运动，当加载运动超过设置值，将会自动触发任务暂停；

```
'该程序任务缓存区大小设置为2

LIMIT_BUFFERED = 2

'加载FORWARD运动进行测试
'FORWARD运动为正转运动，无运动取消指令或者触发报警，将不会自动取消

FORWARD AXIS( 1 ) 'AXIS(1) MTYPE
FORWARD AXIS( 1 ) 'AXIS(1) NTYPE
FORWARD AXIS( 1 ) 'TASK BUFFERED+1 = 1
FORWARD AXIS( 1 ) 'TASK BUFFERED+1 = 2 FULL
FORWARD AXIS( 2 )
```

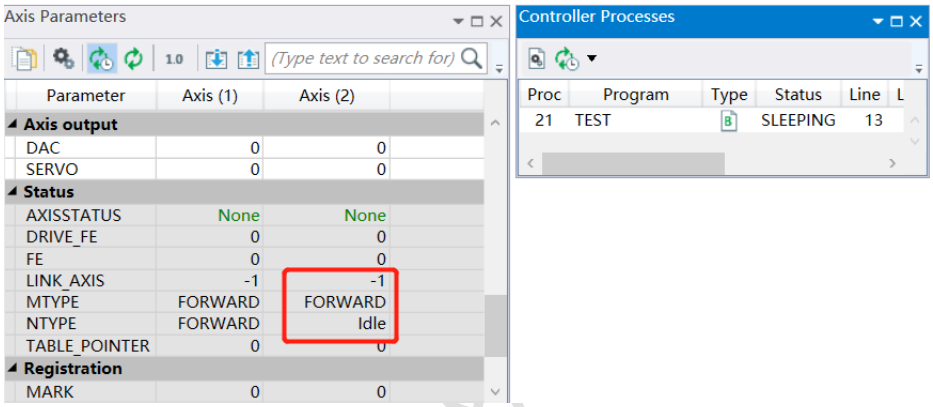


上图中使用 FORWARD 指令进行测试，将缓存区大小设置为 2，轴 1 执行 4 条运动指令，轴 2 执行 1 条运动指令；

其执行效果为：轴 1 的 4 条指令加载至轴 0 的 MTYPE/NTYPE，缓存区，并占满任务的运动缓存区，当执行到轴 2 运动时，由于运动缓存区已满，导致程序暂停在该行，无法继续执行；

3) 每个任务都可以加载轴运动，接上例，任务 21 中执行到加载轴 2 运动，由于 21 号任务缓存满，所以轴 2 运动无法加载；但并不影响其他任务加载轴 2 运动；

```
'另一个任务加载轴2运动
FORWARD AXIS( 2 )
```



3.15 单轴基本操作

3.15.1轴类型设置

3.15.1.1 ATYPE

功能描述	设置控制轴的类型； 在 MC_CONFIG 文件中进行设置； 完整 ATYPE 设置可参考 TRIO BASIC 帮助文件；

ESTUN &

常用类型

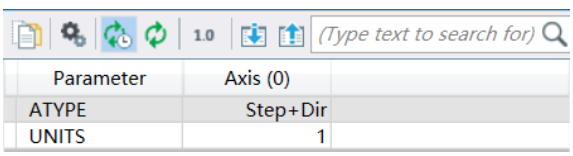
设定值	轴类型	匹配硬件
0	虚拟轴	
35	PWM 输出	MC4N/MC6N/MC664(X) 本体 DP9 用于 PWM 控制 MC403/MC405/MC508 本体轴用于 PWM 控制
43	脉冲＋方向	MC4N/MC6N/MC664(X) 本体 DP9 控制脉冲型伺服 MC403/MC405/MC508 本体轴用于控制脉冲型伺服
44	模拟量闭环	MC403/MC405/MC508 用于控制模拟量型伺服位置控制（伺服工作在速度环，位置环在控制器） MC664(X)本体 DP9 针口和本体模拟量输出构成闭环
65	ETHERCAT 位置控制	所有 ETHERCAT 型控制器，总线轴的默认设置
66	ETHERCAT 速度控制	所有 ETHERCAT 型控制器，6060 = 9
67	ETHERCAT 力矩控制	所有 ETHERCAT 型控制器，6060 = 10
69	ETHERCAT 增量式编码器输入	P375 外接增量式编码器
71	ETHERCAT 脉冲控制	P375 外接脉冲型伺服/步进
76	增量式编码器输入	MC6N/MC4N/MC664(X)本体 DP9 针外接增量式编码器

实例

将 MC4N 本体上 DP9 接口轴类型设置为脉冲+方向，设置如下：

	Name	Modifier	Value
	ATYPE	AXIS(0)	43
*			

重启控制器，执行效果如下：



部分 ATYPE 的设置语句写在.bas 文件中，运行报错，建议 ATYPE 的设置在 MC_CONFIG 文件；

ESTUN & TMO

	ATYPE AXIS(4) = 65 ATYPE must be configured using MC_CONFIG parameter DRIVE_MODE 总线型伺服可能会涉及到控制模式的切换，需要在.BAS 文件中对 ATYPE 进行修改；

3.15.1.2 DRIVE_MODE

功能描述	设置 ETHERCAT 总线轴的控制方式； 轴参数； MC_CONFIG 文件中设置；	
设定值	1	位置控制模式（6060:00=8）
	2	速度控制模式（6060:00=9）
	3	转矩控制模式（6060:00=10）

3.15.1.3 DRIVE_PROFILE

功能描述	设置 ETHERCAT 总线轴的 PDO 配置； 轴参数； MC_CONFIG 文件中设置；
使用注意	不同产品的预设 PDO 选项可能不一致； 预设 PDO 可以被 EC_EXTEND 文件中替换；

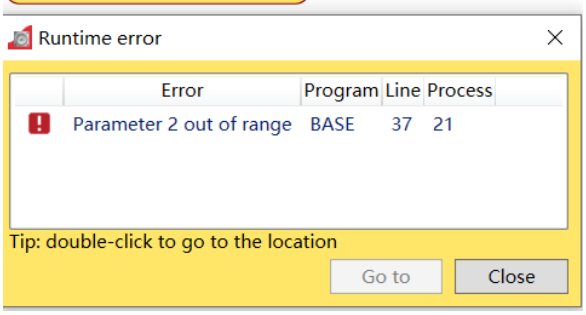
3.15.2轴选择

3.15.2.1 AXIS

指令语法	AXIS(expression)
功能描述	用于运动指令、轴参数的轴指定
示例	设置轴参数时，指定操作轴； SPEED AXIS(0) = 100 '0号轴的速度设定为100 SPEED AXIS(1) = 200 '1号轴的速度设定为200 执行运动命令时，指定操作轴 MOVE(100) AXIS(0) '0号轴移动100 MOVEABS(200) AXIS(1) '1号轴移动到位置200

ESTUN & Trio

3.15.2.2 BASE

指令语法	BASE(axis no<,second axis><,third axis>...)
功能描述	用于运动指令、轴参数的轴指定; 用于指定轴组进行插补运动;
示例	设置轴参数时，指定操作轴; '对轴0及轴1进行基本参数的设置 BASE(0) UNITS = 1000 SPEED = 10 ACCEL = 100 DECEL = 100 FE_LIMIT = 10 FE_RANGE = 10 BASE(1) UNITS = 1000 SPEED = 10 ACCEL = 100 DECEL = 100 FE_LIMIT = 10 FE_RANGE = 10 执行运动命令时，指定操作轴 '轴0移动100 BASE(0) MOVE(100) '轴1移动至200 BASE(1) MOVEABS(200) 指定多轴执行插补运动 '0号轴和1号轴执行插补运动到（100,100） BASE(0, 1) MOVEABS(100, 100) 默认情况下，BASE 内部的轴顺序不可为降序; '以1号轴作为基准轴，1号轴和0号轴执行插补运动 BASE(1, 0) Parameter 2 out of range  设置 COORDINATOR_DATA，可以改变顺序，解决案例 4

	'以1号轴作为基准轴，1号轴和0号轴执行插补运动 COORDINATOR_DATA(16, 1) ' Do not check axis order BASE(1, 0) MOVEABS(100, 100)

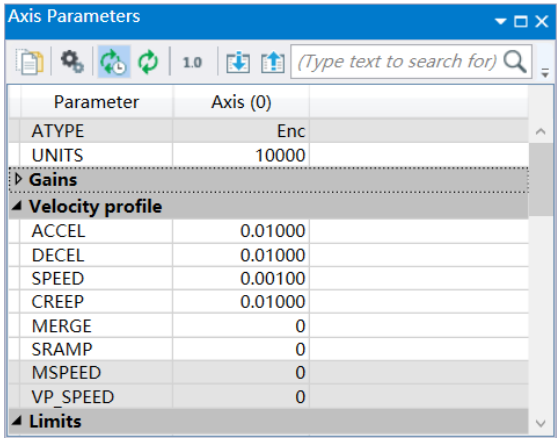
3.15.3运行单位设置

3.15.3.1 UNITS

功能描述	轴参数； 表示一个运动单位，电机的脉冲数；
计算方法	1. 直线机构 假设电机转动一圈脉冲数为 A，减速机减速比为 B（如果无减速机，则 B=1），丝杆导程为 C（单位为 mm），如果需要控制器运动指令单位为 1mm，那么 UNITS 的设定值为： UNITS=A(单圈脉冲数)*B(减速比)/C(导程) 2. 旋转机构 假设电机转动一圈脉冲数为 A，减速机减速比为 B（如果无减速机，则 B=1），如果需要控制器运动指令单位为 1°，那么 UNITS 的设定值为： $UNITS = A(\text{单圈脉冲数}) * B(\text{减速比})/360$
实例	已知 电机型号：EM3A-04AFA22（20 位增量式编码器） 减速机：3:1 的减速比 丝杆：10mm 导程 UNITS 设定值：(2^20)*(3/1)/(10)
使用注意	1. UNITS 设定值大于 0； 2. UNITS 更改，轴参数会跟随 UNITS 的变化比例而变化，所以一般轴参数的设置需要放在 UNITS 设置后；

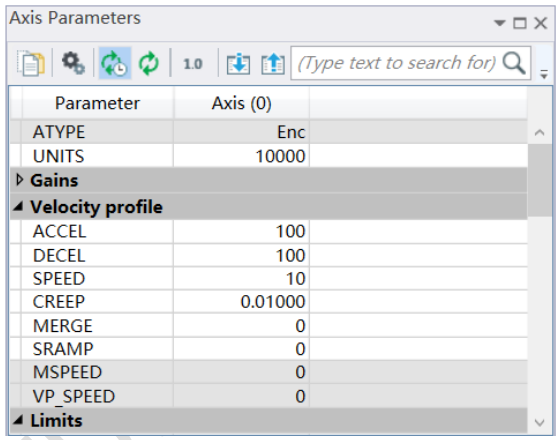
ESTUN & Trio

```
BASE( 0 )
SPEED = 10
ACCEL = 100
DECEL = 100
UNITS = 10000
```



UNITS 设置在其他参数之后

```
BASE( 0 )
UNITS = 10000
SPEED = 10
ACCEL = 100
DECEL = 100
```



UNITS 设置在其他参数之前

3. 在 FRAME=0 时，UNITS 表示关节轴的单位量纲，在 FRAME<>0 时,UNITS 为世界坐标系轴的单位量纲，AXIS_UNITS 为关节轴的单位量纲；

4. 在 FRAME<>0 时，UNITS 的计算方式按照对应 FRAME 要求进行设置。

3.15.3.2 AXIS_UNITS

功能描述	轴参数； 表示一个运动单位，电机的脉冲数； FRAME<>0 时生效； AXIS_UNITS 的计算方法同 UNITS 一致；

3.15.3.3 ENCODER_RATIO/STEP_RATIO

指令语法	ENCODER_RATIO(mpos_count, input_count) STEP_RATIO(output_count, dpos_count)

功能描述

控制器内部第二齿轮比功能，默认值为 1；

$$\text{MPOS (轴位置)}=\frac{ENCODER}{UNITS}*\frac{mpos_count}{input_count}$$

Mpos_count	默认值为 1，设定值为整型
Input_count	默认值为 1，设定值为整型

$$\text{dac_out}_{(\text{目标位置})}=\text{DPOS}*\text{UNITS}*\frac{output_count}{dpos_count}$$

Out_count	默认值为 1，设定值为整型
Dpos_count	默认值为 1，设定值为整型

示例

已知 X/Y 两轴做插补运动，X/Y 两轴的机械参数如下：

	X	**Y**
电机脉冲数	2^20	2^20
减速比	3	1
丝杆导程	10mm	10mm
计算得出 UNITS（1mm）	$\frac{(2^{20})*3}{10}$	$\frac{(2^{20})*1}{10}$

插补运动，要求插补对象的两轴 UNITS 需设置一致，则可以通过 ENCODER_RATIO 和 STEP_RATIO 进行设置。

```
DIM axisx, axisy AS INTEGER
axisx = 1
axisy = 2
BASE( axisx )
UNITS = ( 2 ^ 20 ) * 3 / 10
ENCODER_RATIO( 1, 1 )
STEP_RATIO( 1, 1 )
BASE( axisy )
UNITS = ( 2 ^ 20 ) * 3 / 10
ENCODER_RATIO( 3, 1 )
STEP_RATIO( 1, 3 )
```

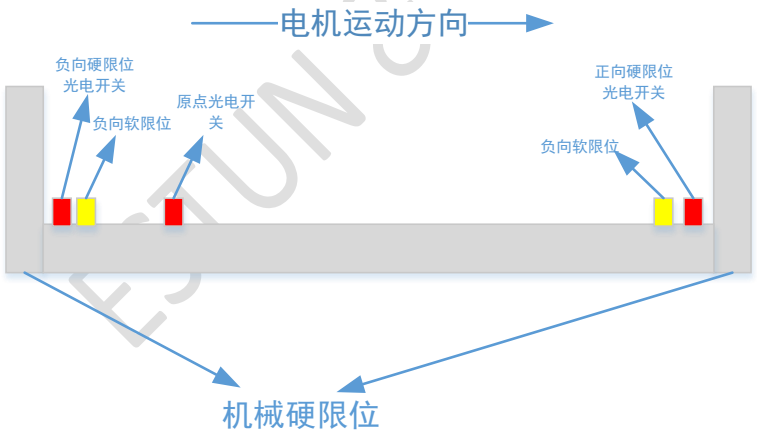
用于更改点机运动方向；

	'已知轴3的运动方向与设备要求运动方向相反 '要求不改变伺服驱动器设置 '可以通过ENCODER_RATIO进行设置 BASE(3) ENCODER_RATIO(-1, 1) STEP_RATIO(-1, 1) 对于增量式编码器轴，如需使用第二齿轮比功能，仅需修改 ENCODER_RATIO；
使用注意	如果不想通过该方法进行修改，可在伺服驱动器端设置齿轮比，实现两轴的 UNITS 一致；

3.15.4安全区设置

3.15.4.1 描述

电机驱动负载，在直线机构上运动，负载的运动范围有限，需要通过软限位或者硬限位的方式，保护负载不会撞到机械限位。以直线机构为例，保护示意图如下：



1) 限位保护功能的目的是保护负载在运动过程中，不会运动至机械硬限位，进而造成设备损坏；

2) 硬限位是在导轨处安装感应光电信号实现，光电信号作为输入信号接入控制器，控制器将输入信号分配为轴的限位保护信号；总线型伺服，光电信号可以接在控制器的数字量输入，也可以接在驱动器的输入端子（通过 PDO 方式将驱动器输入映射至控制器输入），具体操作详见伺服外接原点、限位信号；

3) 软限位的设置一般在设备回零后进行设置，软限位为位置值，当电机当前位置超过软限位区间，停止电机运动；

4) 软限位的设定值保护范围应当小于硬限位保护范围；

5) 绝对值电机，如果没有硬限位保护，那么一定要设置软限位保护；

6) 增量式电机，一般设备会安装硬限位开关，可以不进行软限位设置；

3.15.4.2 正向软限位-FS_LIMIT

功能描述	轴参数； 单位为 UNITS； 设置轴的正向软限位；
示例	1. 正向软限位的设置，将轴 1 的正向软限位设置为 1000，轴 2 的正向软限位设置为 2000； 2. 轴在正向运动时，到达正向限位，轴会减速停止，并将当前运动指令清除，轴状态字 AXISSTATUS 的 BIT9 置 1；因为轴的减速过程，所以轴停止的位置会超过正向限位； 3. 当轴状态为正向软限位警示时，加载正向运动，运动指令不执行，被清除；加载反向运动指令，运动执行；

```
FS_LIMIT AXIS( 1 ) = 1000
FS_LIMIT AXIS( 2 ) = 2000
```

```
BASE( 3 )
UNITS = 1000
FS_LIMIT = 1000
SPEED = 400
ACCEL = 4000
DECEL = 4000
FASTDEC = 10000000

FORWARD
WA( 2 )
PRINT "MTYPE AXIS(3)="; MTYPE AXIS( 3 )
WAIT UNTIL AXISSTATUS AXIS( 3 ).9 = 1 '等待触发软限位
WA( 100 )
PRINT "MPOS AXIS(3)="; MPOS AXIS( 3 )
PRINT "MTYPE AXIS(3)="; MTYPE AXIS( 3 )
```

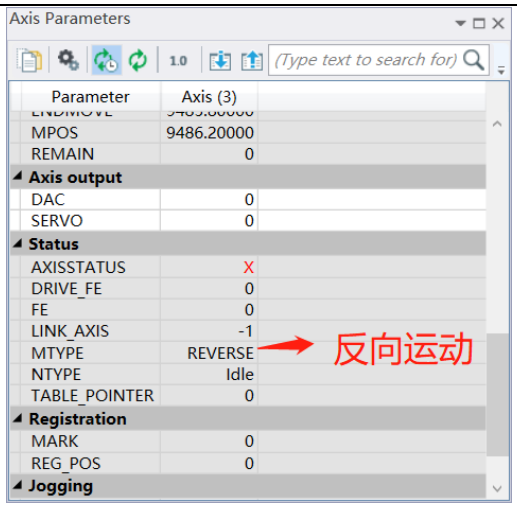
运行结果：

```
MTYPE AXIS(3)=0
MPOS AXIS(3)=1000.8000
MTYPE AXIS(3)=0
```

当触发正向软限位，AXISSTATUS AXIS(3).9=1；轴当前运动被清除，MTYPE=0；轴停止位置为 1000.8>1000；

$$1000.8 = 1000 + \frac{speed^{\wedge}2}{fastdec} = 1000 + \frac{400^{\wedge}2}{2*100000} = 1000.8$$

ESTUN & Trio



3.15.4.3 负向软限位-RS_LIMIT

功能描述	轴参数； 单位为 UNITS； 设置轴的反向软限位；
示例	1) 反向软限位的设置同上； 2) 轴在负向运动时，到达负向限位，轴会减速停止，并将当前运动指令清除，轴状态字 AXISSTATUS 的 BIT10 置 1； 3) 因为轴的减速过程，所以轴停止的位置会超过负向限位； 4) 当轴状态为软负限位警示时，加载负向运动，运动指令不执行，会被直接清除；加载正向运动指令，运动执行；

3.15.4.4 ERRORMASK

功能描述	轴参数； 轴报警设置		
设置	BIT9	0	默认值，到达正向软限位，轴减速停止，不断开使能
		1	到达正向软限位，轴断开使能
	BIT10	0	默认值，到达负向软限位，轴减速停止，不断开使能
		1	到达负向软限位，轴断开使能

3.15.4.5 正向硬限位设置-FWD_IN

功能描述	轴参数； 默认值为-1；设定范围为（-1-1023）； 指定轴的正向硬限位的输入通道； 低电平有效；

ESTUN & Trio

使用注意	1) 正向硬限位低电平有效（硬限位光电感应时，输入点状态为 OFF）；如果正向硬限位光电为常开型，可以通过 INVERT_IN 指令取反； 2) 轴如果无需配置正向硬限位，或者取消轴的正向硬限位，设置为-1； 3) 轴正向运动感应到正向硬限位，AXISSTATUS 的 BIT4 置 1，按照 FASTDEC 设定值减速，所以轴会停止在硬限位以外；FASTDEC 值设定越大，减速距离越小； 4) 当轴当前的状态为硬正限位警示时，加载正向运动，运动指令不执行，会被直接清除；加载反向运动指令，运动执行；

3.15.4.6 负向硬限位设置-REV_IN

功能描述	轴参数； 默认值为-1；设定范围为（-1-1023）； 指定轴的负向硬限位的输入通道； 低电平有效；
使用注意	同上

3.15.4.7 ERRORMASK

功能描述	轴参数； 轴报警设置		
设置	BIT4	0	默认值，到达正向硬限位，轴减速停止，不断开使能
		1	到达正向硬限位，轴断开使能
	BIT5	0	默认值，到达负向硬限位，轴减速停止，不断开使能
		1	到达硬向软限位，轴断开使能

3.15.4.8 旋转机构设置

1) REP_OPTION

功能描述	结合 REP_DIST，用于设置轴运动范围； 默认值为 0；

Bit0	0	轴位置范围为-REP_DIST--+REP_DIST
	1	轴位置范围为 0--+REP_DIST

示例	轴 1 带动旋转机构，需要轴 1 的位置范围为 0-360°；
	`BASE( 1 )`
	`REP_OPTION.0 = 1`
	`REP_DIST = 360`

ESTUN & Trio

2) REP_DIST

功能描述	设置轴运动范围，结合 REP_OPTION 使用
示例	默认情况下，REP_DIST 的设定值小于 FS_LIMIT(正向软限位)的设定值； 当使用控制器控制皮带结构，皮带一直朝一个方向做定位运动，运动过程中要求皮带不会触发限位报警，那么可以把皮带轴的 REP_DIST 设置值小于 FS_LIMIT; '皮带轴 '朝一个方向一直运动 BASE(2) REP_DIST = 10000 REP_OPTION = 0 FS_LIMIT = REP_DIST + 100

3.15.4.9 控制器限位与驱动器限位关联

一般伺服产品自身也具备硬限位保护功能，使用过程中，由于 TRIO 控制器与驱动器在硬限位保护上存在冲突，处理方式不一致，所以应用上需要注意以下几点。

1) 硬限位直接接入控制器的输入模块，可以屏蔽 ETHERCAT 伺服的限位保护功能（因为多数伺服产品硬限位为低电平有效，未接入信号，伺服会报限位警告）；以 PRONET-EC 系统伺服为例，可将参数 PN000.1 和 PN000.2 设置为 1；

参数号	名称及说明	单位	设定范围	出厂值	重新上电
Pn000	二进制位参 Pn000.0：参数伺服 ON Pn000.1：禁止正转输入信号（P-OT） Pn000.2：禁止反转输入信号（N-OT） Pn000.3：瞬间停电时输出伺服报警输出（ALM）	—	0~1111	0	需要

2) 硬限位如果接在伺服 CN1 端子，通过总线方式进行读取映射至 TRIO 上（参考 PDO DI 映射），建议参考 1 中将伺服端限位功能屏蔽，打开控制器端限位功能；

默认设置下，伺服端的设置是感应到限位信号，立即停止运动，控制器端感应到信号，减速停止，那么在感应到硬限位信号时，可能会误触发轴的跟随误差过大报警；

控制器端和伺服端如果将正负限位的配置弄反，导致轴无法运动；

参数号	名称及说明	单位	设定范围	出厂值	重新上电
Pn000	二进制位参 Pn000.0：参数伺服 ON Pn000.1：禁止正转输入信号（P-OT） Pn000.2：禁止反转输入信号（N-OT） Pn000.3：瞬间停电时输出伺服报警输出（ALM）	—	0~1111	0	需要

3) 如果不使用控制器的限位保护功能，则需要打开伺服端的限位保护功能；

4) ETHERCAT 增量轴的回零实现有两种途径，1 是使用控制器的运动指令（DATUM/DATUM+FORWARD 等），2 是使用驱动器的回零功能（控制模式（6060=6））。如果使用的是运动指令回零，那么应该屏蔽控制器的限位保护功能；如果使用的是驱动器内部回零功能，应该关闭控制器的限位保护功能，使用驱动器的限位保护功能；

5) 通过设置轴参数 ERRORMASK 可以将运动碰到限位，报警，自动断开使能；

ERRORMASK.4=1	触发正向硬限位，伺服使能断开
ERRORMASK.5=1	触发负向硬限位，伺服使能断开
ERRORMASK.9=1	触发正向软限位，伺服使能断开
ERRORMASK.10=1	触发负向软限位，伺服使能断开

6) 如果需要使用运动到限位后找原点（FORWARD+DATUM）的方式进行回零，则 ERRORMASK.4 和 ERRORMASK.5 设置为 0；

7) 常用故障：当硬限位接驱动器，通过总线方式映射，轴在运动过程中突然运动被中止，轴未反馈任何错误，原因可能为：硬限位焊线存在虚焊。

3.15.5跟随误差的设置

3.15.5.1 FE

功能描述	轴参数，反馈轴的跟随误差；可读； 单位为 UNITS； $\text{FE AXIS}(x) = DPOS\ AXIS(x) - MPOS\ AXIS(x)$ 跟随误差，表示理想目标位置值和实际位置值的偏差，体现了伺服的跟随性； 影响跟随误差的因素有： - 伺服产品的刚性，刚性越大，跟随误差越小； - 跟随误差与运行速度成正比； - 机械负载卡顿，也会造成跟随误差过大；

ESTUN & ITM

3.15.5.2 FE_RANGE

功能描述	轴参数； 可读写； 单位为 UNITS； 当轴跟随误差 FE>FE_RANGE，轴状态字 AXISSTATUS 的 BIT1 置 1； 默认情况下，发生跟随误差警告，轴不会断使能； ERRORMASK.2 设置为 1，发生生跟随误差警告，轴将断使能；
如何设置	参考 FE_LIMIT

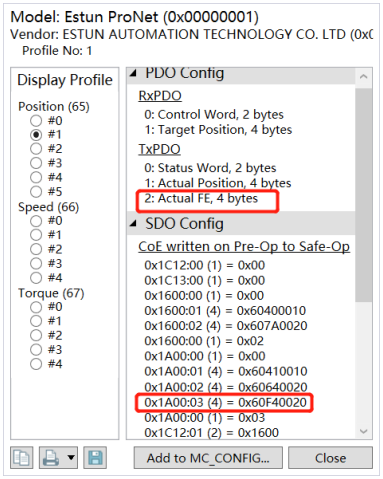
3.15.5.3 FE_LIMIT

功能描述	轴参数；可读写； 单位为 UNITS； 当轴跟随误差 FE>FE_LIMIT，轴状态字 AXISSTATUS 的 BIT8 置 1； 默认情况下，发生跟随误差警告，轴将断使能； ERRORMASK.8 设置为 0，发生生跟随误差警告，轴将不会断使能；
如何设置	如果是增量式伺服，在使用 DATUM 指令回零时，应当根据设置的回零速度设置 FE_LIMIT，避免回零过程中目标位置一直增加，轴因为卡顿等因素造成飞车；回零过程因为速度一般较小，可以将 FE_LIMIT 设置较小，起到保护作用； 轴自动运行过程中，运行速度较大，调试阶段将 FE_LIMIT 设置较大，待伺服位置环等参数调节完毕，通过示波器采集电机在高速运动时的跟随误差（FE），根据该值设置自动运行时的跟随误差，可以为最大跟随误差的 1.2 倍；

3.15.5.4 DRIVE_FE

功能描述	轴参数，反馈轴的跟随误差; 可读; 单位为脉冲; 通过 PDO 读取伺服端跟随误差（60F4:00）; 如果未将 60F4:00 配置进 PDO，那么该值一直为 0； 在控制器内置 PDO 中，DRIVE_PROFILE=1 和 DRIVE_PROFILE=2 中配置了 60F4:00;

ESTUN&Trio

3.15.5.5 DRIVE_FE_LIMIT

功能描述	轴参数，设置轴的跟随误差阈值；可读写； 单位为脉冲； 设定值一般为 DRIVE_FELIMIT AXIS(X)=FE_LIMIT AXIS(X)*UNITS AXIS(X); 常见错误： 将 60F4:00 配置进 PDO，但未对 DRIVE_FE_LIMIT 赋值，导致轴在低速运动下也会报跟随误差过大错误；（在 MC_CONFIG 中轴参数 DRIVE_PROFILE 设置为 1 或者为 2；一般状况下，如果轴回零需要使用 Z 脉冲，需要将 DRIVE_PROFILE 设置为 2）

3.15.5.6 FE_LIMIT_MODE

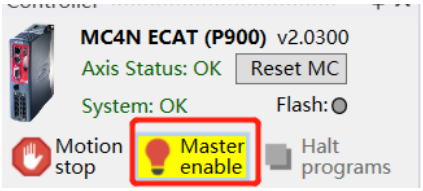
功能描述	轴参数，跟随误差报警滤波设置； **0** 默认值，跟随误差 FE 大于 FE_LIMIT，即触发报警 **1** 连续两个伺服周期 FE 大于 FE_LIMIT，触发报警 建议设置该参数，避免由于编码器某个伺服周期错误的位置反馈，误触发跟随误差报警；

3.15.6使能操作

3.15.6.1 WDOG

功能描述	系统参数； 轴使能总开关； WDOG = ON:打开轴使能总开关；

ESTUN & Trio

	WDOG = OFF:关闭轴使能总开关； 可以通过控制器数窗口按钮手动打开/关闭 WDOG; 
实例	WDOG = ON，不一定所有轴均使能；每个轴使能均有独立的分开关 AXIS_ENABLE; WDOG=OFF,所有轴均使能断开； 默认情况下，如果某个轴发生故障，将产生系统故障，导致 WDOG = OFF; 通过 DISABLE_GROUP 将轴进行分组，同时通过 SYSTEM_ERROR_MASK 指令屏蔽相应系统报警，当其中某个轴发生故障，WDOG 保持为 ON，其他轴组的使能状态不受影响；

3.15.6.2 AXIS_ENABLE

功能描述	轴参数；可读写； AXIS_ENABLE AXIS(X) = ON :打开 X 轴的使能分开关； 默认值为 ON（用户容易认为轴状态的使能控制仅由 WDOG 决定）；

实例	使用 AXIS_ENABLE 可以实现对单个轴的使能通断控制；
	`'要求操作VR(0).0，轴0的使能状态进行切换`
	`IF VR( 0 ).0 = 1 THEN`
	`  IF AXIS_ENABLE AXIS( 0 ) = ON THEN`
	`  AXIS_ENABLE AXIS( 0 ) = OFF`
	`  ELSE`
	`  AXIS_ENABLE AXIS( 0 ) = ON`
	`  ENDIF`
	`  VR( 0 ).0 = 0`
	`ENDIF`

3.15.6.3 SERVO

功能描述	轴参数；可读写； 位置闭环功能；
注意事项	1) 需要根据控制轴类型及控制方式合理设置 SERVO；

	a) 轴类型为脉冲方向控制（ATYPE=43），SERVO 设置为 0； b) 轴类型为总线位置轴（ATYPE=65）,SERVO 设置为 1； c) 轴类型为总线速度轴（ATYPE=66）,控制轴匀速运行，SERVO 设置为 0，控制轴定位，SERVO 设置为 1； d) 轴类型为总线力矩轴（ATYPE=67）,控制轴恒力矩输出，SERVO 设置为 0，控制轴定位，SERVO 设置为 1； e) 使用 P375 脉冲加方向控制，SERVO 设置为 1； f) 轴类型为模拟量闭环（ATYPE=44）,SERVO 设置为 1； 2) SERVO=0 时，控制器会将 MPOS 值设定为 DPOS 值； 3) SERVO=0 时，跟随误差被清 0；

3.15.6.4 DISABLE_GROUP

指令格式	DISABLE_GROUP(parameter[,parameters…])
功能描述	轴分组功能

ESTUN & Trio

实例

实例 1

```
'已知设备由3个部分组成
'第一部分有3个伺服轴：0,1,2
'第二部分有4个伺服轴：3,4,5,6,
'第三部分有3个伺服轴：7,8,9
'要求某部分中的其中某个轴报警，同一部分的其他轴使能断开
'要求某部分中的其中某个轴报警，其他部分不受影响

DISABLE_GROUP( -1 ) '取消原先分组
DISABLE_GROUP( 0, 1, 2 )
DISABLE_GROUP( 3, 4, 5, 6 )
DISABLE_GROUP( 7, 8, 9 )
```

实例 2

```
'已知设备中共有10个轴，轴号从0-9
'要求任何一个轴故障，不影响其他轴
DISABLE_GROUP( -1 ) '取消原先分组

FOR i = 0 TO 9

    DISABLE_GROUP( i )

NEXT i
```

总线轴报警，会触发系统报警，导致 WDOG=OFF，如果轴故障不影响其他轴组，除 DISABLE_GROUP 需要设置，仍然需要设置 SYSTEM_ERROR_MASK;

```
'已知设备中共有10个轴，轴号从0-9
'要求任何一个轴故障，不影响其他轴
DISABLE_GROUP( -1 ) '取消原先分组

SYSTEM_ERROR_MASK.20 = 0 '轴单元站错误，WDOG不会变为OFF

FOR i = 0 TO 9

    DISABLE_GROUP( i )

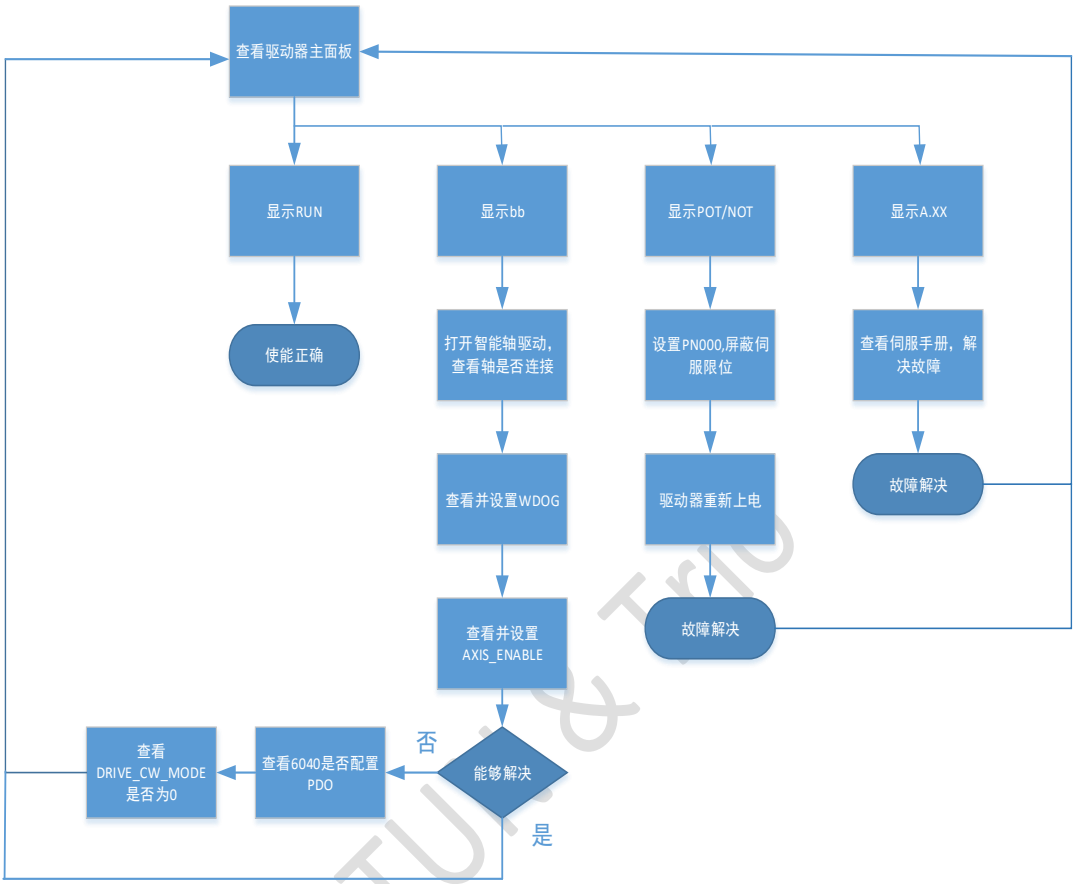
NEXT i
```

3.15.6.5 使能条件

1) 脉冲控制轴，硬件连接正确；
2) ETHERCAT 总线轴，通讯需要正确，控制字(6040:00)及状态字(6041:00)需要配置在 PDO;
3) WDOG = ON;
4) AXIS_ENABLE AXIS(X)=ON；
5) 驱动器无故障（埃斯顿伺服面板，不报 A.XX 报警）；
6) 伺服面板不显示 POT/NOT(可以通过 PN000 进行屏蔽)；

3.15.6.6 ETHERCAT 总线轴如何确认轴已使能

以埃斯顿 PRONET-EC 系统为例，简述如何确认轴使能状态及原因。



3.15.7回零操作

3.15.7.1 含义

回零，建立轴运动的参考点。

	增量式伺服	绝对值伺服
上电编码器位置	电机位置清 0	电机位置不清 0
回零模式	多种	将当前位置定义为 0
有无动作	有	无
编码器故障检测	不需要	需要（发生故障，重新标 0）
回零精度	回零速度影响回零精度	不受回零速度影响
原点开关	需要	不需要
限位保护	硬限位保护	软限位保护

3.15.7.2 增量式回零-控制器实现

3.15.7.2.1 原点开关 DATUM_IN

功能描述	轴参数； 默认值为-1； 设置指定轴的原点开关输入通道； 低电平有效，如果外接原点开关为常开型，需要使用 INVERT_IN 取反；

3.15.7.2.2 回零速度 SPEED

功能描述	轴参数； 设置回零中的速度； 单位为 UNITS/s;

3.15.7.2.3 蠕动速度 CREEP

功能描述	轴参数； 设置回零运动中的蠕动速度； 单位为 UNITS/s; 蠕动速度一经确认，不可随意修改；

3.15.7.2.4 回零加速度 ACCEL

功能描述	轴参数； 设置回零运动中的加速度； 单位为 UNITS/s^2;

3.15.7.2.5 回零减速度 DECEL

功能描述	轴参数； 设置回零运动中的减速度； 单位为 UNITS/s^2;

3.15.7.2.6 回零运动 DATUM

指令形式	DATUM(sequence)

ESTUN & Trio

<table>
<tr><td>功能描述</td><td>轴命令；
触发轴执行找原点动作；
<table>
<tr><th>sequence</th><th>描述</th></tr>
<tr><td>1</td><td>轴以 CREEP 速度正向运动直到发现 Z 脉冲，并将 Z 脉冲位置设置为 0</td></tr>
<tr><td>2</td><td>轴以 CREEP 速度反向运动直到发现 Z 脉冲，并将 Z 脉冲位置设置为 0</td></tr>
<tr><td>3</td><td>轴以 SPEED 速度正向运动直到发现原点开关，然后以 CREEP 速度反向离开原点开关，离开原点开关，回零完成，将检测原点开关上升沿时的位置定义为 0</td></tr>
<tr><td>4</td><td>轴以 SPEED 速度反向运动直到发现原点开关，然后以 CREEP 速度正向离开原点开关，离开原点开关，回零完成，将检测原点开关上升沿时的位置定义为 0</td></tr>
<tr><td>5</td><td>轴以 SPEED 速度正向运动直到发现原点开关，然后以 CREEP 速度反向直到发现 Z 脉冲，并将 Z 脉冲位置设置为 0</td></tr>
<tr><td>6</td><td>轴以 SPEED 速度反向运动直到发现原点开关，然后以 CREEP 速度正向直到发现 Z 脉冲，并将 Z 脉冲位置设置为 0</td></tr>
</table></td></tr>
</table>

3.15.7.2.7 Z 脉冲找原点

<table>
<tr><td>问题描述</td><td>使用 DATUM 指令回零，模式 1,2,5,6 都需要找电机 Z 脉冲确定原点位置，控制器应当能够接受到 Z 脉冲反馈，该模式方可正常运动</td></tr>
<tr><td>使用注意</td><td>脉冲方向轴无法使用 Z 脉冲回零；
模拟量闭环轴必须接 Z 脉冲；
ETHERCAT 总线轴必须将 TOUCH PROBE 相关的对象配置至 PDO;
<table>
<tr><th>Index</th><th>Name</th><th>Type</th><th>Attr.</th><th>PDO Mapping</th><th>M/O</th></tr>
<tr><td>60B8h</td><td>Touch Probe Function</td><td>UINT16</td><td>RW</td><td>YES</td><td>M</td></tr>
<tr><td>60B9h</td><td>Touch Probe Status</td><td>UINT16</td><td>RO</td><td>YES</td><td>M</td></tr>
<tr><td>60BAh</td><td>TouchProbePos1PosValue</td><td>INT32</td><td>RO</td><td>YES</td><td>M</td></tr>
<tr><td>60BBh</td><td>TouchProbeNeg1PosValue</td><td>INT32</td><td>RO</td><td>YES</td><td>M</td></tr>
<tr><td>60BCh</td><td>TouchProbePos2PosValue</td><td>INT32</td><td>RO</td><td>YES</td><td>M</td></tr>
<tr><td>60BDh</td><td>TouchProbeNeg2PosValue</td><td>INT32</td><td>RO</td><td>YES</td><td>M</td></tr>
</table>
NOTE:使用 ESTUN 系列伺服，可以设置 DRIVE_PROFILE=2；</td></tr>
</table>

ESTUN & TRIO

3.15.7.2.8 复杂回零方式

描述	很多应用，单轴执行 DATUM 里面的某个模式无法满足所有的回零需求，比如，直线机构原点开关安装在中间位置，上电后无法确定负载停止在原点开关左边或是右边，单个 DATUM 运动指令就无法满足需求
方法	多运动指令结合。 使用 FORWARD 指令或者 REVERSE 指令找限位，保证负载停止在原点开关的某一侧； 执行 DATUM 指令回零，确定零点

3.15.7.3 增量式回零-伺服驱动器实现

该方法仅适用于 ETHERCAT 总线型伺服；

3.15.7.3.1 伺服控制模式

在对象 6060:00 设置伺服工作模式，对象 6061:00 查看伺服当前工作模式，值见下表：

值	说明
0	未设置控制模式
1	位置控制模式
3	速度控制模式
4	转矩控制模式
6	回零模式
8	位置插补控制模式（CSP）DRIVE_MODE=1
9	位置插补控制模式（CSV）DRIVE_MODE=2
10	位置插补控制模式（CST）DRIVE_MODE=3

回零模式下相关对象有：

对象索引	说明	对象索引	说明
6040:00	控制字	**6041:00**	状态字
6098:00	回零模式	**6099:00**	回零速度
609A:00	回零加速度	607C:00	零点偏移

3.15.7.3.2 DRIVE_CW_MODE

功能描述	轴参数； 开放控制字（6040:00）的写权限；	
设定值	0	系统根据 WDOG 和 AXIS_ENABLE 的状态设置控制字

	1	用户可以通过 DRIVE_CONTROLWORD 控制该控制字
	2	用户可以通过 DRIVE_CONTROLWORD 控制该控制字位 11 到 15

3.15.7.3.3 DRIVE_CONTROLWORD

功能描述	轴参数； DRIVE_CW_MODE=1 时有效； 设置 6040:00 值
BIT 位含义	

Bit	Description
0	打开
1	使能电压
2	快速停止
3	启用操作
4	回零操作开始
5	特定的操作模式
6	特定的操作模式
7	故障复位
8	停止

3.15.7.3.4 DRIVE_STATUWORD

功能描述	轴参数； 读取 6041:00 值；
BIT 位含义	

15	14	13	12	11	10	9 ~ 0
Homeflag	*	homing_error	homing_attained	*	target_reached	*

Name	Value	Description
Targetreached	0	• Halt = 0: Home position not reached • Halt = 1: Axle decelerates
	1	• Halt = 0: Home position reached • Halt = 1: Axle has velocity 0
Homing attained	0	Homing mode not yet completed
	1	Homing mode carried out successfully
Homing error	0	No homing error
	1	Homing error occurred; Homing mode carried out not successfully; The error cause is found by reading the error code
Homeflag*	0	Home position not find
	1	Home position has found (only for absolute encoder)

NOTE:具体含义详见《ETHERCAT 总线手册》

3.15.7.3.5 使用注意

- 原点开关、限位开关需要设置在伺服驱动器端；
- 回零速度、回零模式需要通过 SDO 写的方式进行设置，参考 CO_WRITE_AXIS 指令；

ESTUN & Trio

3.15.7.3.6 案例

```
BASE( 2 )
SERVO = OFF
DRIVE_CW_MODE = 1 ' set the control word to be user mode
DRIVE_CONTROLWORD = $06 ' disable the drive
' Set the drive to DS402 homing mode
CO_WRITE_AXIS( ax, $6060, $00, 2, -1, 6 )
' wait for the homing mode to be accepted
VR( 100 ) = 0
REPEAT
   CO_READ_AXIS( ax, $6061, $00, 2, 100 )
UNTIL VR( 100 ) = 6
' set the homing method (1 for +ve direction, 2 for -ve)
fwd = 1
rev = 2
CO_WRITE_AXIS( ax, $6098, $00, 2, -1, fwd )
DRIVE_CONTROLWORD = $1f 'start homing
WA( 20 )
' wait for Homing Done flag (bit 12)

REPEAT
   WA( 1 )
UNTIL DRIVE_STATUS.12 = 1

WA( 20 )
DEFPOS( ENCODER ) ' set the axis position to drive's value
SERVO = ON
WDOG = ON
' Set the drive to position mode
CO_WRITE_AXIS( ax, $6060, $00, 2, -1, 8 )
' Set control word to normal enabled state
DRIVE_CONTROLWORD = $2f
DRIVE_CW_MODE = 0 ' set the control word back to wdog mode
```

ESTUN & Trio

3.15.7.4 绝对值回零-控制器实现

3.15.7.4.1 DEFPOS

指令格式	DEFPOS(pos1 [,pos2[, pos3[, pos4...]]])
功能描述	轴命令；

	将轴的位置设置为 POS1，轴无运动； 单位为 UNITS;

案例	绝对值电机标零，将 ENCODER 存储至 VR 中
	IF l_i_hm_mode(j) = 0 THEN
	' abs drive
	' zero define
	IF VR(l_controlword + j).l_cmd_home = 1 THEN
	VR(c_v_axis_absoffset_saveaddr + j) = ENCODER AXIS(j)
	DEFPOS(0) AXIS(j)
	VR(c_v_axis_statusword + j).c_v_axisstatus_homedone = 1
	VR(c_v_axis_statusword + j).c_v_axisstatus_abs_bd_done = 1
	VR(c_v_axis_statusword + j).c_v_axisstatus_encoder_error = 0
	VR(l_controlword + j).l_cmd_home = 0
	ENDIF
	ENDIF

3.15.7.4.2 OFFPOS

功能描述	轴命令； 设置当前位置的偏移； 单位为 UNITS；
案例	绝对值电机标零，上电之后设置 UNITS 之前，执行一次 OFFPOS 操作，将存储在 VR 中的值补偿给轴； IF l_b_ax_isabs(i) _ AND VR(c_v_axis_statusword + i).c_v_axisstatus_abs_bd_done = 1 _ AND FLAG(c_flag_axis_ini_done) = OFF THEN OFFPOS AXIS(i) = -VR(c_v_axis_absoffset_saveaddr + i) ENDIF

3.15.7.4.3 注意点

控制器复位或者重新上电，OFFPOS 补偿只能执行一次；

3.15.7.5 绝对值回零-驱动器实现

参考增量式回零-驱动器实现

回零模式为 35，可将电机位置清 0，控制器读取到的 ENCODER 也清 0。

方式	方向	目标位置	参考点位置	DS402
30	负	参考点开关	参考点开关	30
33	负	当前位置	C 脉冲	33
34	正	当前位置	C 脉冲	34
35	--	当前位置	当前位置	35
-4	正	目标扭矩	C 脉冲	厂家自定义
-3	负	目标扭矩	C 脉冲	厂家自定义
-2	正	目标扭矩	目标扭矩	厂家自定义
-1	负	目标扭矩	目标扭矩	厂家自定义

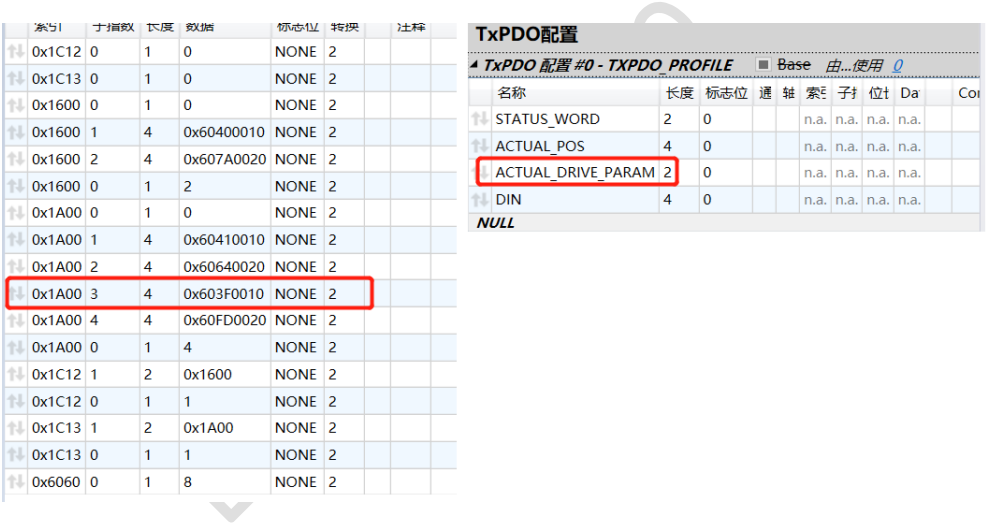
3.15.7.6 绝对值编码器故障检测（ESTUN 产品为例）

1) 通过 SDO 或者 PDO 的方式读取对象 603F:00 的数值；SDO 读取参考指令

CO_READ_AXIS,PDO 读取参考 PDO 配置

603F	0	Error code	UINT16	RW	YES

A．45	×	绝对值编码器多圈信息出错	多圈信息出错
A．46	×	绝对值编码器多圈溢出	多圈信息溢出
A．47	×	电池电压低于 2.5v	多圈信息已丢
A．48	×	电池电压低于 3.1v	电池电压偏低
A．49	×	编码器位置跳变	未接电池，电池电压不足或编码器损坏
A．50	×	串行编码器通讯超时	编码器没连接、编码器信号受干扰，编码器损坏或编码器解码电路损坏。



索引	子指数	长度	数据	标志位	转换	注释
0x1C12	0	1	0	NONE	2	
0x1C13	0	1	0	NONE	2	
0x1600	0	1	0	NONE	2	
0x1600	1	4	0x60400010	NONE	2	
0x1600	2	4	0x607A0020	NONE	2	
0x1600	0	1	2	NONE	2	
0x1A00	0	1	0	NONE	2	
0x1A00	1	4	0x60410010	NONE	2	
0x1A00	2	4	0x60640020	NONE	2	
0x1A00	3	4	0x603F0010	NONE	2	
0x1A00	4	4	0x60FD0020	NONE	2	
0x1A00	0	1	4	NONE	2	
0x1C12	1	2	0x1600	NONE	2	
0x1C12	0	1	1	NONE	2	
0x1C13	1	2	0x1A00	NONE	2	
0x1C13	0	1	1	NONE	2	
0x6060	0	1	8	NONE	2	

PDO 映射

2) 通过函数将 603F:00 的数值转成故障代码；

```
FUNCTION inter_fb_ax_is_abserror(ax AS INTEGER,drive_code AS STRING) AS BOOLEAN

    ' 用户可以根据需求新增新的品牌驱动器
    ' 在_C_AXIS文件中配置轴的驱动器型别,
    DIM tmp,tmp2 AS INTEGER

    IF drive_code = "ESTUN_PRONET" THEN
        tmp = (DRIVE_PARAMETER AXIS(ax) AND $00FF)
        IF tmp>=45 AND tmp<=56 THEN
            RETURN TRUE
        ELSE
            RETURN FALSE
        ENDIF
    ENDIF

    IF drive_code = "ESTUN_SUMMA" THEN
        tmp2 = (DRIVE_PARAMETER AXIS(ax) AND $00FF)
        tmp = VAL(HEX(tmp2))
        IF tmp>=45 AND tmp<=56 THEN
            RETURN TRUE
        ELSE
            RETURN FALSE
        ENDIF
    ENDIF

    RETURN FALSE

ENDFUNC
```

3) 检测到绝对值故障，报警，重新标零；

```
IF l_i_hm_mode(i) = 0 THEN
    IF inter_fb_ax_is_abserror(i, l_str_drive_pid(i)) THEN
        VR(c_v_axis_statusword + i).c_v_axisstatus_homedone = 0
        VR(c_v_axis_statusword + i).c_v_axisstatus_abs_bd_done = 0
        VR(c_v_axis_statusword + i).c_v_axisstatus_encoder_error = 1
    ENDIF
ENDIF
```

3.15.8JOG 实现

3.15.8.1 FWD_JOG

功能描述	轴参数； 设置轴正向 JOG 的输入信号，可以为虚拟 IO; 低电平有效； 默认值为-1； 设定范围为 0-1023；

3.15.8.2 REV_JOG

功能描述	轴参数； 设置轴反向 JOG 的输入信号，可以为虚拟 IO; 低电平有效； 默认值为-1； 设定范围为 0-1023；

3.15.8.3 JOGSPEED

功能描述	轴参数； 设置 JOG 运动的速度； 单位为 UNITS/S;

ESTUN & Trio

3.15.8.4 实例

```
'通过 VR(0)实现 0-15 轴的正向 JOG VR(0).0-轴 0---VR(0).15-轴 15
'通过 VR(1)实现 0-15 轴的反向 JOG VR(1).0-轴 0---VR(1).15-轴 15
'VR(10)-VR(25)存储各轴的 JOG 速度
'JOG 加减速是速度的 10 倍

DIM di_jog_virtual AS INTEGER
di_jog_virtual = 500 '虚拟 IO
OP( di_jog_virtual, OFF ) '低电平有效

WHILE TRUE
  'JOG 实现
  FOR i = 0 TO 15
    IF VR( 0 ).i = 1 THEN
      GOSUB write_speed
```

```
        FWD_JOG AXIS( i ) = di_jog_virtual
        REV_JOG AXIS( i ) = -1
      ELSEIF VR( 1 ).i = 1 THEN
        GOSUB write_speed
        FWD_JOG AXIS( i ) = -1
        REV_JOG AXIS( i ) = di_jog_virtual
      ELSE
        FWD_JOG AXIS( i ) = -1
        REV_JOG AXIS( i ) = -1
      ENDIF
    NEXT i
WEND

write_speed:
BASE( i )
JOGSPEED = VR( 10 + i )
ACCEL = JOGSPEED * 10
DECEL = JOGSPEED * 10
RETURN
```

3.15.9INC（寸动）实现

3.15.9.1 MOVE

指令形式	MOVE(distance1 [,distance2 [,distance3 [,distance4...]]])
功能描述	轴命令； Distance 的单位为 UNITS； 运动速度为 SPEED; 加速度为 ACCEL; 减速度为 DECEL; 功能 1：单轴相对运动； 功能 2：多轴相对运动插补；

distance1:	从当前位置沿基准轴移动的距离。
distance2:	从当前位置开始在 BASE 数组中的下一个轴所移动的距离。
distance3:	从当前位置开始在 BASE 数组中的下二个轴所移动的距离。
distance4:	从当前位置开始在 BASE 数组中的下三个轴所移动的距离。

ESTUN & Trio

3.15.9.2 寸动实现

```
'通过 VR(0)实现 0-15 轴的正向寸动  VR(0).0-轴 0---VR(0).15-轴 15
'通过 VR(1)实现 0-15 轴的反向寸动  VR(1).0-轴 0---VR(1).15-轴 15
'VR(10)-VR(25)存储各轴的寸动速度
'寸动加减速是速度的 10 倍
'VR(30)-VR(45)存储各轴的寸动长度

WHILE TRUE
  'JOG 实现
  FOR i = 0 TO 15
    IF VR( 0 ).i = 1 THEN
      IF MTYPE AXIS( i ) = 0 THEN
        GOSUB write_speed
        MOVE( ABS( VR( 30 + i ) ) ) AXIS( i )
      ENDIF
      VR( 0 ).i = 0
    ENDIF
    IF VR( 1 ).i = 1 THEN
      IF MTYPE AXIS( i ) = 0 THEN
        GOSUB write_speed
        MOVE( -ABS( VR( 30 + i ) ) ) AXIS( i )
      ENDIF
      VR( 1 ).i = 0
    ENDIF
  NEXT i
WEND

write_speed:
BASE( i )
SPEED = VR( 10 + i )
ACCEL = JOGSPEED * 10
DECEL = JOGSPEED * 10
RETURN
```

ESTUN & Trio

3.15.10 绝对运动

指令形式	MOVEABS(position1[, position2[, position3[, position4...]]])

功能描述	轴命令； Position 单位为 UNITS; 运动速度为 SPEED; 加速度为 ACCEL; 减速度为 DECEL; 功能 1：单轴绝对运动； 功能 2：多轴绝对运动插补；
实例	单轴绝对运动 '轴 2 以速度 100，加速度 1000，减速度 1000 移动至点 A,A 点坐标为 500 BASE(2) SPEED = 100 ACCEL = 1000 DECEL = 1000 MOVEABS(500)

3.15.11 相对运动

参考 MOVE 指令。

3.15.12 运动取消

3.15.12.1 CANCEL

指令形式	CANCEL([mode])
功能描述	用于取消轴运动命令； 用于取消轴组的运动命令；

mode	0	清除 MTYPE 中的运动
	1	清除 NTYPE 中的运动
	2	取消轴的所有运动指令

实例	1. 执行 CANCEL 指令，将以 FASTDEC 设定值减速停止； 2. CANCEL 指令对特殊运动指令无效（SYNC）；

ESTUN & Trio

3.15.12.2 RAPIDSTOP

指令形式	RAPIDSTOP [(mode)]
功能描述	用于取消所有轴运动命令； 用于取消所有轴组的运动命令； mode: 0 清除所有轴 MTYPE 中的运动；1 清除所有轴 NTYPE 中的运动；2 取消所有轴的所有运动指令
实例	执行 RAPIDSTOP 指令，将以 FASTDEC 设定值减速停止； RAPIDSTOP 指令对特殊运动指令无效（SYNC）；

3.15.13 暂停保护功能

3.15.13.1 FHOLD_IN

功能描述	轴参数； 默认值为-1； 分配输入信号为轴的暂停命令，可以为虚拟 IO;设置范围 0-1023； 低电平有效； 轴暂停功能激活，AXISSTATUS 的 BIT7 置 1； 轴暂停状态，轴以 FHSPEED 速度运行；
实例	使用 FHOLD_IN 实现暂停功能； 如果需要轴在运动过程中，轴暂停，可将未使用的虚拟 IO 分配给轴，由于低电平有效，轴将暂停，但运动指令不取消；如果需要继续工作，将该参数恢复为-1；

```
'要求系统暂停状态，所有轴要立即停止，但运动指令不取消
'轴包含：0-10
DEFCONST sys_run 10
DEFCONST sys_pause 20
DEFCONST sys_estop

WHILE TRUE
  SELECT_CASE sys_statamachine
    CASE sys_run
      GOSUB axis_pause_cancel
    CASE sys_pause
      GOSUB axis_pause_set
      ' commands
    CASE sys_estop
  END_CASE
```

ESTUN & Trio

```
WEND

axis_pause_set:
OP( 500, OFF )
FOR i = 0 TO 10
   FHSPEED AXIS( i ) = 0
   FHOLD_IN AXIS( i ) = 500 '虚拟 IO
NEXT i
RETURN

axis_pause_cancel:
FOR i = 0 TO 10
   FHSPEED AXIS( i ) = 0
   FHOLD_IN AXIS( i ) = -1 '虚拟 IO
NEXT i
RETURN
```

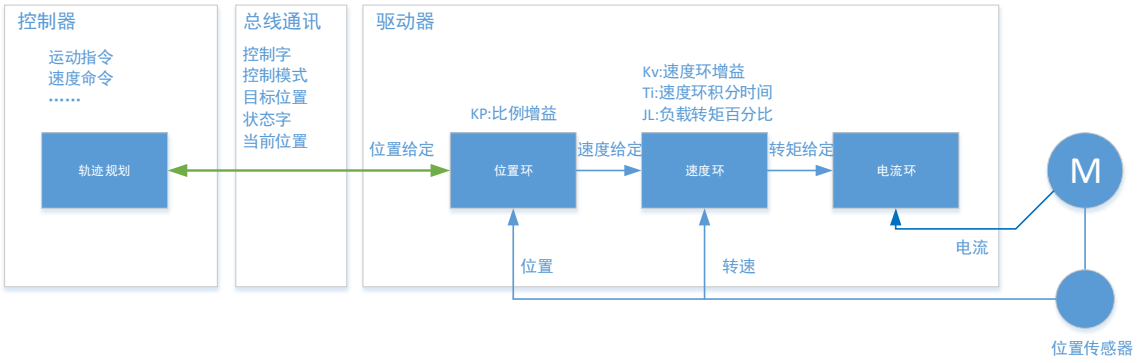
3.15.13.2 FHSPEED

功能描述	轴参数； 默认值为 0； 单位为 UNITS/s; 当 feedhold 输入有效时，运动降低到 FHSPEED。

ESTUN & Trio

3.15.14 总线轴运动异常排查

单轴系统拓扑图：



3.15.14.1 伺服端

- 查看驱动器端是否处于限位保护状态，当处于限位保护，驱动器面板显示 POT 或者 NOT;（常见错误：伺服端未屏蔽正负限位保护）
- 如果伺服驱动器能够使能，查看伺服的位置环、速度环增益参数；（常见错误：伺服端位置环参数设置不合理）

Pn101	负载刚性设定	—	0~36	5	不需要
Pn102	速度环增益	rad/s	1~4000	250	不需要
Pn103	速度环积分时间	0.1ms	1~4096	200	不需要
Pn104	位置环增益	1/s	0~1000	40	不需要

3.15.14.2 总线端

- 在智能轴驱动里面查看伺服控制模式（6061:00）是否正确，或者通过 SDO 读的方式查看控制模式是否正确；位置控制模式，6061:00 的值应当等于 8；

该对象用来显示伺服从站当前所在的控制模式。

索引	子索引	名称	数据类型	可访问性	PDO 映射	取值
6061h	0	Modes of Operation display	UINT8	RO	TxPDO	Default：0

- PDO 配置是否正确，在通过采集诊断报告，获取当前的 PDO 映射；

RPDO	TPDO	说明
6040h：Controlword	6041h：Statusword	必须
607Ah：Target position	6064h：Position Actual Value	必须
6060h：mode of operation	6061h：Modes of operation display	可选

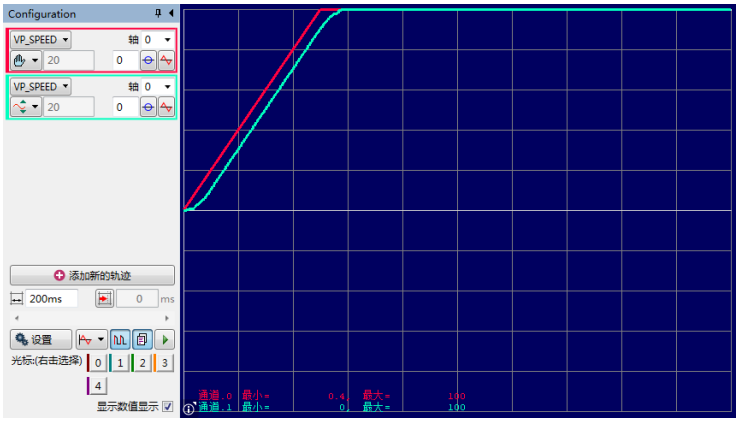
3.15.14.3 控制器端

- UNITS 设置是否合理，是否过小；
- 使能命令是否给定（WDOG/AXIS_ENABLE）；
- SERVO 设定值是否正确；
- 速度（SPEED）,加速度（ACCEL）,减速度（DECEL）是否正确；
- 轴状态是否处于限位保护；
- 暂停保护功能是否激活；

ESTUN & Trio

3.15.15 运动平滑功能

3.15.15.1 VP_MODE

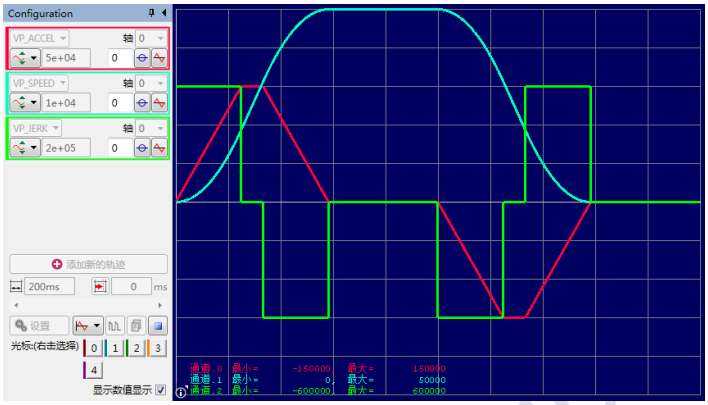
功能描述	通过设置 VP_MODE，以及设置时间参数 SRAMP、JERK 等轴参数，实现 S 型运动及多阶运动。
实例	VP_MODE=0（默认值） SRAMP:定义 S 型曲线的时间（ms 为单位），必须在运动指令下发前进行设置；  BASE(0) SRAMP=100 SPEED=100 ACCEL=200

DECEL=200

MOVE(200)

上图所示：红线为 SRAMP=0 时设置的理想速度曲线，加速时间为 100/200=0.5s，绿线为 SRAMP=100 时的理想速度曲线，加速时间为 100/200+0.1=0.6s

VP_MODE=1



VP_MODE = 1

UNITS = 1000

SPEED = 50000

ACCEL = 150000

DECEL = 150000

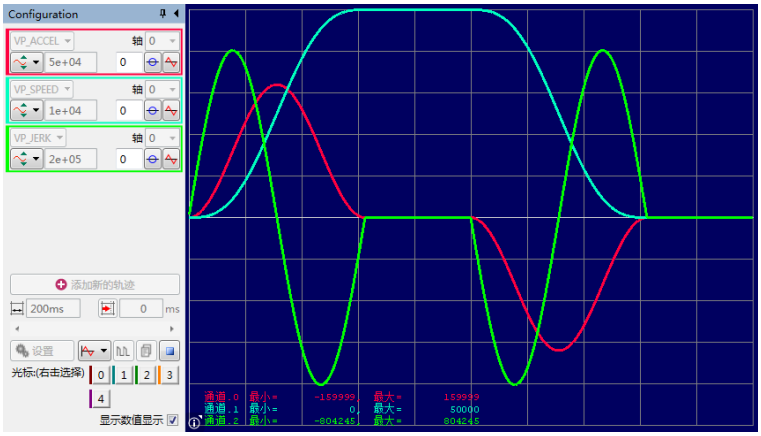
JERK = 600000

MOVE (50000)

如上图所示，加速度值的变化由 JERK 参数决定，加速至设定加速度值所需时间为 ACCEL/JERK=150000/600000=0.25S,加速至设定速度需要 0.6S。

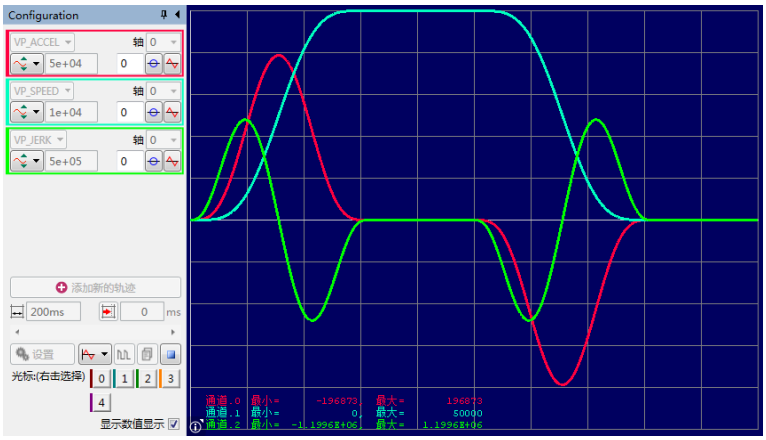
VP_MODE=2

加速度曲线为 sin 曲线；



VP_MODE=3

ESTUN & Trio

3.15.15.2 SRAMP

功能描述	轴参数； S 曲线时间；单位为 ms; VP_MODE=0 有效；

3.15.15.3 JECK

功能描述	轴参数； 加加速度；单位为 units/s^3; VP_MODE=1 有效；

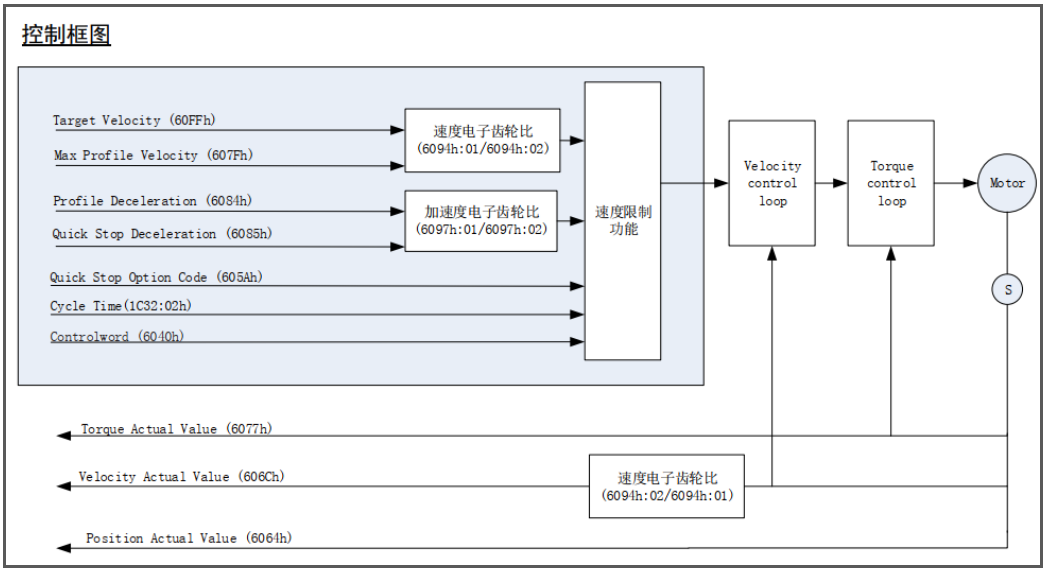
3.15.16 速度控制的实现

控制器可实现对 ETHERCAT 总线轴速度模式（6060:00=9）的控制。

- 选择轴类型为 ETHERCAT 总线速度轴；
- 使用轴参数 DRIVE_MODE 设置速度控制模式；
- 使用轴参数 DRIVE_PROFILE 选择 PDO 配置；
- 关闭位置闭环功能；
- 通过 WDOG/AXIS_ENABLE 控制轴使能；
- 通过 DAC 控制给定速度；
- 通过 SDO 写实现其他速度模式相关对象的读写；

ESTUN & Trio

3.15.16.1 ETHERCAT 速度模式



速度模式控制图

3.15.16.2 PDO 配置

速度模式下，PDO 的配置需要遵循下表。

RPDO	TPDO	说明
6040:00：控制字	6041:00：状态字	必须
60FF:00：目标速度		必须
6060:00：控制模式	6061:00：实际控制模式	可选
	6064:00：电机位置	可选
	606C:00：实际速度	可选

使用 ESTUN 驱动器，轴参数 DRIVE_PROFILE 设置为 0，其关联的 PDO 配置如下：

RPDO	TPDO
6040:00：控制字	6041:00：状态字
60FF:00：目标速度	6064:00：电机位置

如需数据库内无此配置，参考《PDO 配置》，添加总线配置文件。

3.15.16.3 速度模式选择

功能描述	MC_CONFIG 文件中作如下设置： ATYPE = 66 DRIVE_MODE = 2

ESTUN & Trio

3.15.16.4 关闭位置闭环

功能描述	CSV 模式下，如果只是速度控制，SERVO=0;

3.15.16.5 激活使能

功能描述	使用 WDOG、AXIS_ENABLE 对轴进行使能；

3.15.16.6 速度给定-DAC

功能描述	轴参数； DAC 给定值，关联 PDO 中的目标速度 60FF:00; 单位：0.1r/min;

3.15.16.7 速度模式相关对象

功能描述	其他速度控制相关的参数可以通过 CO_WRITE_AXIS 指令写入；	
对象表	607F:00	最大速度，单位为 0.1r/min
	6084:00	加速度
	6085:00	快速停止减速度
	6064:00	实际速度

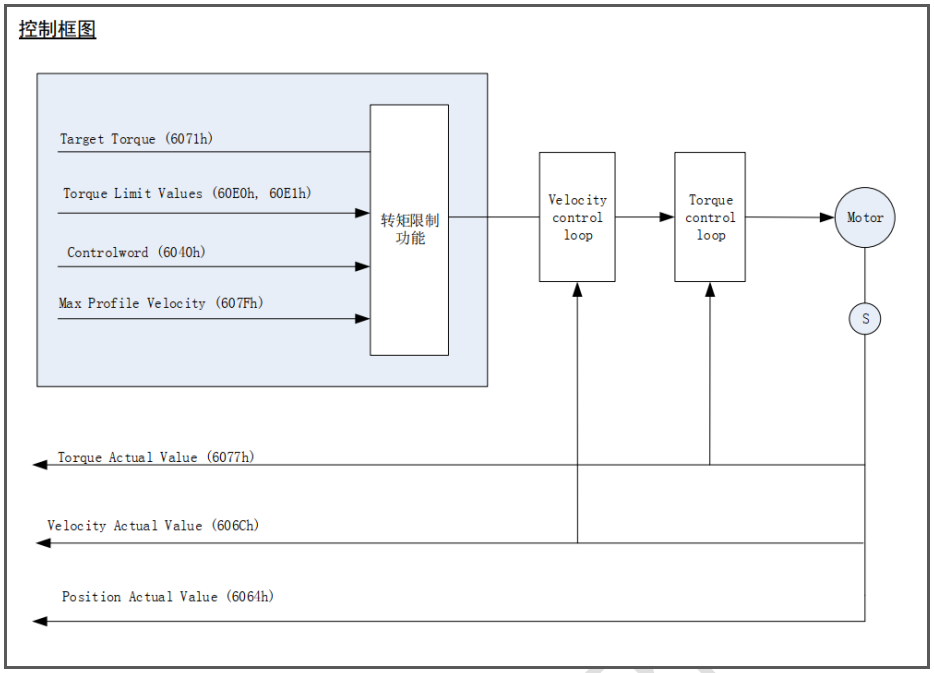
3.15.17 力矩控制的实现

控制器可实现对 ETHERCAT 总线轴力矩模式（6060:00=10）的控制。

- 选择轴类型为 ETHERCAT 总线力矩轴；
- 使用轴参数 DRIVE_MODE 设置速度控制模式；
- 使用轴参数 DRIVE_PROFILE 选择 PDO 配置；
- 关闭位置闭环功能；
- 通过 WDOG/AXIS_ENABLE 控制轴使能；
- 通过 DAC 控制给定转矩；
- 通过 SDO 写实现其他转矩模式相关对象的读写；

ESTUN & Trio

3.15.17.1 ETHERCAT 力矩模式



转矩模式控制框图

3.15.17.2 PDO 配置

速度模式下，PDO 的配置需要遵循下表。

RPDO	TPDO	说明
6040:00：控制字	6041:00：状态字	必须
6071:00：目标转矩		必须
6060:00：控制模式	6061:00：实际控制模式	可选
	6064:00：电机位置	可选
	6077:00：实际转矩	可选

ESTUN 驱动器，DRIVE_PROFILE 设置为 0，其关联的 PDO 配置如下：

RPDO	TPDO
6040:00：控制字	6041:00：状态字
6071:00：目标转矩	6064:00：电机位置

如需数据库内无此配置，参考《PDO 配置》，添加总线配置文件。

3.15.17.3 转矩模式选择

功能描述	MC_CONFIG 文件中作如下设置： ATYPE = 67 DRIVE_MODE = 3

3.15.17.4 关闭位置闭环

功能描述	CST 模式下，如果只是速度控制，SERVO=0;

3.15.17.5 激活使能

功能描述	使用 WDOG、AXIS_ENABLE 对轴进行使能；

3.15.17.6 转矩给定

功能描述	轴参数； 关联 PDO 中的目标速度 6071:00; 单位：0.1%*额定转矩； 范围：-3000--3000

3.15.17.7 速度限幅设置

功能描述	对象索引为 607F:00； 单位为 0.1r/min； 可使用 SDO 方式写该对象，参考 CO_WRITE_AXIS; 可使用 PDO 方式，通过 VR 写该对象，参考《PDO 对象如何关联 VR》

3.15.17.8 力矩模式相关对象

功能描述	其他速度控制相关的参数可以通过 CO_WRITE_AXIS 指令写入；	
对象表	60E0:00	正向转矩限幅
	60E1:00	反向转矩限幅

ESTUN & Trio

3.16 轴状态的监控

用户可以通过工具栏-轴参数工具，监控各轴运行状态；也可以通过读取轴参数，监控各轴状态。

轴状态的监控包含：

- 位置信息监控；
- 速度信息监控；
- 运动状态信息监控；
- 总线信息监控；

3.16.1位置监控

参数名称	含义	单位	读写权限
MPOS	轴的测量位置	UNITS	只读
DPOS	FRAME=0:关节轴的目标位置； FRAME<>0：世界轴的目标位置	UNITS	只读
AXIS_DPOS	关节轴的目标位置	AXIS_UNITS	只读
REMAIN	轴的运动剩余量	UNITS	只读
ENCODER	编码器位置，ETHERCAT 总线下，关联对象索引为（6064:00）	脉冲	只读
ENDMOVE	轴的停止位置	UNITS	只读
DAC_OUT	CSP 模式下，给定的位置	脉冲	只读

3.16.2速度监控

参数名称	含义	单位	读写权限
MSPEED	轴的测量速度	UNITS/s	只读
MSPEEDF	一阶滤波后的测量速度，滤波系数由轴参数 MSPEED_FILTER 设置	UNITS/s	只读
SPEED	轴的设定速度	UNITS/s	读写
VP_SPEED	轴的理想给定速度	UNITS/s	只读
ACCEL	轴的设定加速度	UNITS/S^2	读写
DECEL	轴的设定减速度	UNITS/S^2	读写

JERK	轴的设定加加速度	UNITS/S^3	读写
VP_ACCEL	轴的理想给定加速度	UNITS/S^2	只读
VP_JECK	轴的理想给定加加速度	UNITS/S^3	只读

3.16.2.1 MSPEED_FILTER

功能描述	轴参数； 速度的一阶滤波系数； MSPEEDF=(1-MSPEED_FILTER)*MSPEEDF+MSPEED_FILTER*MSPEED

3.16.3运动状态监控

参数名称	含义	单位	读写权限
MTYPE	轴当前正在执行的运动，参考轴缓存表	--	只读
NTYPE	轴即将执行的运动指令，参考轴缓存表	--	只读
IDLE	TRUE:当前轴无运动； FALSE:当前轴有运动；	--	只读
AXISSTATUS	不同 BIT 位表示不同状态，参考 AXISSTATUS	--	只读
FE	跟随误差	UNITS	只读
LOADED	TRUE:运动指令加载成功	--	只读

3.16.3.1 轴缓存

轴缓存区（MTYPE/NTYPE）数值与运动类型关联

数值	运动类型	描述
0	IDLE	无运动
1	MOVE	相对运动
2	MOVEABS	绝对定位
3	MHELICAL	螺旋线插补
4	MOVECIRC	圆弧插补
5	MOVEMODIFY	运动位置修改
6	MOVESP	平滑相对运动
7	MOVEABSSP	平滑绝对值运动
8	MOVECIRCSP	平滑圆弧插补
9	MHELICALSP	平滑螺旋线插补
10	FORWARD	正向运动

ESTUN & Trio

数值	运动类型	描述
11	REVERSE	反向运动
12	DATUM	找零运动
13	CAM	单轴凸轮运动
14	FWD_JOG	正向 JOG 运动
15	REV_JOG	反向 JOG 运动
20	CAMBOX	电子凸轮耦合
21	CONNECT	电子齿轮
22	MOVELINK	追剪运动
23	CONNPATH	多轴电子齿轮
24	FLEXLINK	飞剪运动
25	CAMLINK	
26	COMPENSATE_XY	平面位置补偿
30	MOVETANG	
31	MSPHERICAL	空间圆弧运动
32	MTRANSCIRE	
33	MOVESEQ	多点运动
34	MOVEABSSEQ	多点绝对运动
40	MSPHERICALSP	平滑空间圆弧运动
41	MTRANSCIRESP	
42	MOVESEQSP	平滑多点运动
43	MOVEABSSEQSP	平滑多点绝对运动
44	MOVEJ	机器人关节运动
45	MOVEL	机器人世界直线运动
46	MOVEC	机器人世界圆弧运动
47	MOVELEND	

ESTUN & Trio

3.16.3.2 AXISSTATUS

BIT 位	描述
0	速度限制
1	跟随误差 FE 超过 FE_RANGE
2	远程通讯错误（轴连接断开）
3	驱动器故障
4	到达正向硬限位
5	到达反向硬限位
6	回零指令执行中
7	暂停保护激活
8	跟随误差 FE 超过 FE_LIMIT
9	正向软限位报警
10	反向软限位报警
11	运动指令清除中（执行 CANCEL/RAPIDSTOP）
12	脉冲输出超限
13	MOVETANG 减速
15	VOLUME_LIMIT 激活
更多	请查看 BASIC 手册

3.16.3.3 总线监控

参数名称	含义	单位	读写权限
DRIVE_CONTROL	控制字状态	--	读写
DRIVE_STATUS	状态字	--	只读
DRIVE_TORQUE	实时力矩（前提：PDO 配置）	--	只读
DRIVE_SPEED	实时速度（前提：PDO 配置）	--	只读
DRIVE_FE_LIMIT	轴跟随误差（前提：PDO 配置）	脉冲	只读

ESTUN & Trio

3.17 多轴插补运动

插补运动，指多个轴按照设置的合成速度同时启停运动。常用的插补运动有：

- 直线插补运动；
- 二轴圆弧插补；
- 三轴空间圆弧插补；
- 螺旋线插补；

为达到良好的插补运动效果，要求 UNITS 设置值一样。如果基于相同单位计算出的 UNITS 不一致，可通过 ENCODER_RATIO、STEP_RATIO 实现。

3.17.1插补轴组指定

用户可通过 BASE 指令指定插补运动的轴组。

如果插补轴的轴号存在倒序，需要执行 COORDINATOR_DATA 语句，开放该功能。

3.17.2插补速度设置

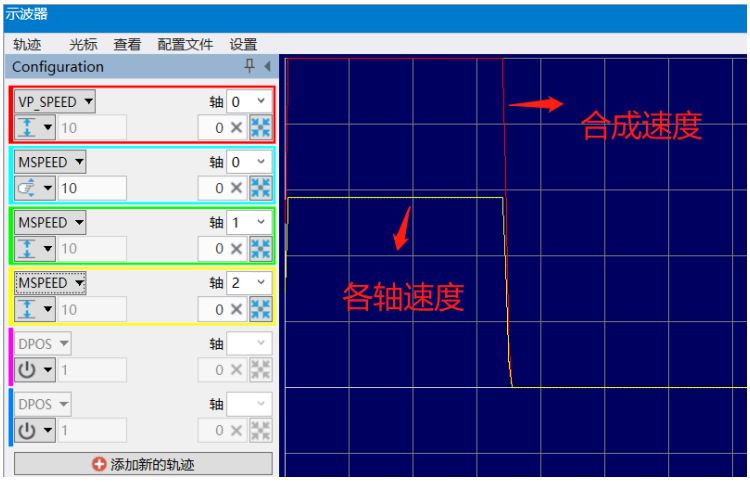
插补运动的合成速度、加速度、减速度等运动参数通过设置基准轴的轴参数 SPEED、ACCEL、DECEL 实现。如果轴 0 和轴 1 进行直线插补运动，设置轴 0 的速度参数即可。

使用 INTER_FACTOR 功能可以更改插补速度设置。

3.17.2.1 INTERP_FACTOR

功能描述	轴参数； 默认值为 1； 当轴组执行插补运动时，INTERP_FACTOR 设置为 0 的轴不参与插补速度的计算；
实例	X/Y/Z 三轴执行直线插补运动，相对运动，移动量为 100,100,100，运行速度为 50，加速度为 500，减速度为 500，所有轴均参与合成速度的运算； BASE(0, 1, 2) SPEED = 50 ACCEL = 500 DECEL = 500 MOVE(100, 100, 100) TRIGGER 监控速度曲线：

ESTUN & Trio

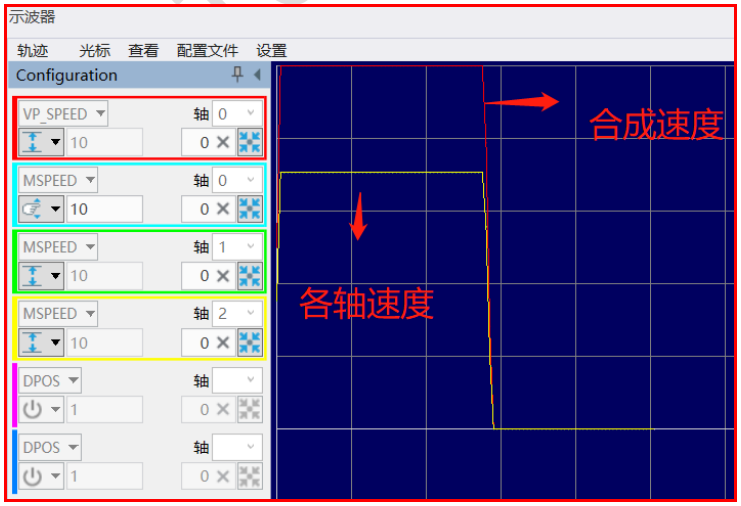


各轴速度均为：50 / SQR(3)=28.8675

如果使用 INTERP_FACTOR 屏蔽 Z 轴；

```
INTERP_FACTOR AXIS( 2 ) = 0
BASE( 0, 1, 2 )
SPEED = 50
ACCEL = 500
DECEL = 500
MOVE( 100, 100, 100 )
TRIGGER
WAIT IDLE
INTERP_FACTOR AXIS( 2 ) = 1
```

监控速度曲线：



各轴速度均为：50 / SQR(2)=35.3553

使用注意	如果运动时屏蔽了某轴，不参与合成速度运算，运动结束后需要将该轴的 INTERP_FACTOR 设置为 1，否则该轴无法单独运动； 使用 INTERP_FACTOR 时，如果参与运算轴的移动量远远小于未参与轴的移动量，可能会出现轴运动速度过大异常；

ESTUN & Trio

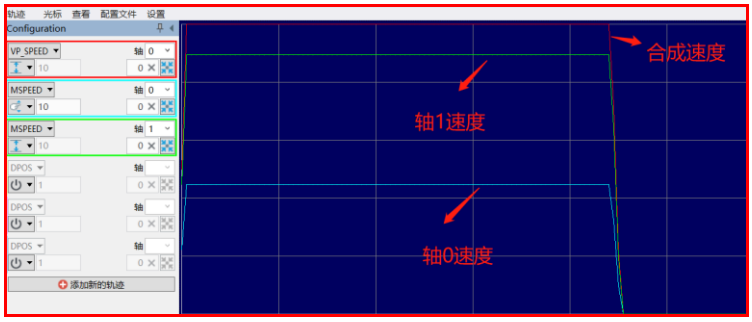
	比如 x/y/z 的移动量为（1,1,100），y/z 两轴的 inter_factor 设置为 0，运动速度为 10，那么 z 轴的运动速度则为 1000。 在使用 CANNPATH 指令实现轴与轴组之间的电子齿轮控制时，可能会需要使用 INTERP_FACTOR 指令屏蔽部分轴的变化量；

3.17.3直线插补

直线插补分为相对直线插补运动，绝对直线插补运动，支持 2-6 轴的直线插补运动。

3.17.3.1 MOVE

指令格式	MOVE(distance1 [,distance2 [,distance3 [,distance4...]]])
功能描述	多轴直线插补运动； 相对运动； 单位为 UNITS; 使用 BASE 语句指定轴组； 插补运动的合成速度由轴组的基准轴 SPEED 参数给定； 插补运动的加速度由轴组的基准轴 ACCEL 参数给定； 插补运动的加速度由轴组的基准轴 DECEL 参数给定；

实例	已知点 A 与点 B 之间 XY 方向坐标相差 100,200，要求点 A 到点 B，运动速度为 50，加速度为 500，减速度为 500，代码实现如下：
	`BASE( 0, 1 )`
	`SPEED = 50`
	`ACCEL = 500`
	`DECEL = 500`
	`MOVE( 100, 200 )`
	速度监控图：
	 
	分析：
	给定的速度为矢量速度，匀速时轴 0 和轴 1 的速度分别为：
	V0 = 50*X_INC/总移动量；
	V1 = 50*Y_INC/总移动量；

ESTUN & Trio

3.17.3.2 MOVEABS

指令格式	MOVEABS(position1[, position2[, position3[, position4...]]])
功能描述	多轴直线插补运动； 绝对运动； 原点建立参考轴回零部分； 单位为 UNITS; 使用 BASE 语句指定轴组； 插补运动的合成速度由轴组的基准轴 SPEED 参数给定； 插补运动的加速度由轴组的基准轴 ACCEL 参数给定； 插补运动的加速度由轴组的基准轴 DECEL 参数给定；
实例	已知点 A 的坐标为（100,200,40），需要从当前位置以速度 40，加速度 400，减速度 400 移动至 A 点，XYZ3 轴的轴号分别为 0,2,3， 则代码实现如下： BASE(0, 2, 3) SPEED = 40 ACCEL = 400 DECEL = 400 MOVEABS(100, 200, 40)

3.17.4圆弧插补

圆弧插补运动包含两轴圆弧插补，三轴空间圆弧插补，空间螺旋线插补。

3.17.4.1 MOVECIRC

指令格式	MOVECIRC(end1, end2, centre1, centre2, direction [,ta [,output]])

ESTUN & Trio

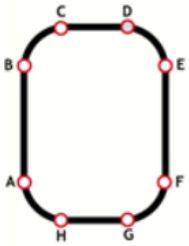
功能描述

平面圆弧插补；

End1	圆弧结束点相对于起点 X 方向的偏移量	
End2	圆弧结束点相对于起点 Y 方向的偏移量	
Centre1	圆弧圆心点相对于起点 X 方向的偏移量	
Centre2	圆弧圆心点相对于起点 Y 方向的偏移量	
diretion	0	逆时针圆弧
	1	顺时针圆弧
	2	最短路径圆弧
	3	最长路径圆弧
ta	过度角弧度，可降低机械振动	
Output	存储增加过渡角的修正圆参数	

实例

实现如下轨迹；



实现代码：

```
BASE( 0, 1 )
SPEED = 30
ACCEL = 300
DECEL = 300

ta = 0 * PI / 180 ' set the transition angle to 0 degrees
MOVEABS( 0, 40 ) 'move A -> B
MOVECIRC( 30, 30, 30, 0, 1, ta ) 'move B -> C
MOVE( 20, 0 ) 'move C -> D
MOVECIRC( 30, -30, 0, -30, 1, ta ) 'move D -> E
MOVE( 0, -40 ) 'move E -> F
MOVECIRC( -30, -30, -30, 0, 1, ta )'move F -> G
MOVE( -20, 0 ) 'move G -> H
MOVECIRC( -30, 30, 0, 30, 1, ta ) 'move H -> A
```

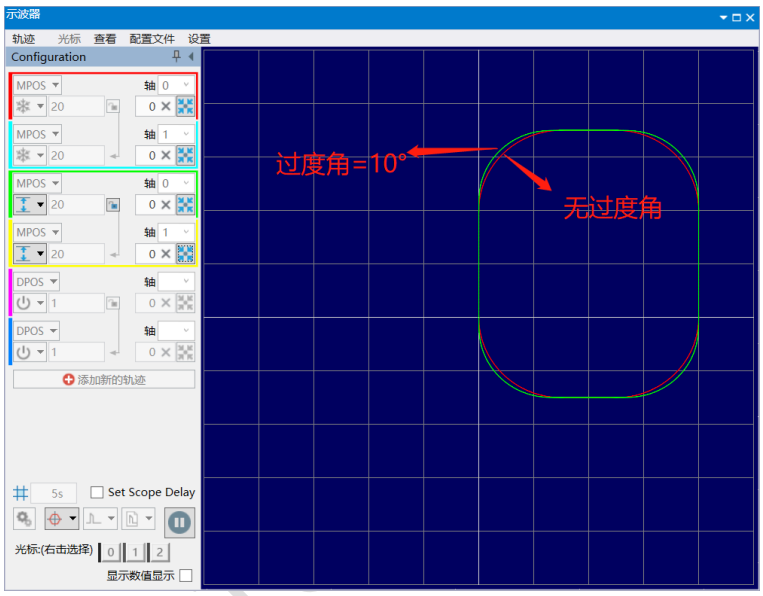
将过度角设置为 10 度，代码如下：

```
BASE( 0, 1 )
SPEED = 30
ACCEL = 300
DECEL = 300
```

ESTUN & Trio

```
ta = 10 * PI / 180 ' set the transition angle to 10 degrees
MOVEABS( 0, 40 ) 'move A -> B
MOVECIRC( 30, 30, 30, 0, 1, ta ) 'move B -> C
MOVE( 20, 0 ) 'move C -> D
MOVECIRC( 30, -30, 0, -30, 1, ta ) 'move D -> E
MOVE( 0, -40 ) 'move E -> F
MOVECIRC( -30, -30, -30, 0, 1, ta )'move F -> G
MOVE( -20, 0 ) 'move G -> H
MOVECIRC( -30, 30, 0, 30, 1, ta ) 'move H -> A
```

监控位置曲线：



在许多情况下，即使是几度的“过渡曲线”也可以显着降低机器振动，但过渡曲线工具会修改圆弧的位置和半径。

使用注意	如果指定的终点不在圆弧上。圆弧将以中心和终点之间的直线指定的角度结束； 在 MOVECIRC 期间，两个轴都不能越过设置的绝对重复距离（REP_DIST）；

3.17.4.2 MSPHERICAL

指令格式	MSPHERICAL({parameters}, mode [, gtpi][, rotau][, rotav][, rotaw])

<table>
<tr><td>功能描述</td><td>空间圆弧插补运动；
<table>
<tr><td>mode</td><td>0</td><td>设置圆弧的中间点和结束点</td></tr>
<tr><td></td><td>1</td><td>设置圆弧的圆心和结束点</td></tr>
<tr><td></td><td>2</td><td>设置两个中间点，绘制完整圆弧</td></tr>
<tr><td></td><td>3</td><td>设置中间点和圆心，绘制完整圆弧</td></tr>
<tr><td>Gtpi</td><td colspan="2">该值非 0 时，mode0 和 mode1 运动时绘制圆弧方向反向</td></tr>
<tr><td>Rotau</td><td colspan="2">U 轴旋转移动量</td></tr>
<tr><td>Rotav</td><td colspan="2">V 轴旋转移动量</td></tr>
<tr><td>rotaw</td><td colspan="2">R 轴旋转移动量</td></tr>
</table></td></tr>
<tr><td colspan="2">Mode=0：指定圆弧的中间点和结束点</td></tr>
<tr><td>指令格式</td><td>MSPHERICAL(endx, endy, endz, midx, midy, midz, 0)</td></tr>
<tr><td>指令描述</td><td>
<table>
<tr><td>endx</td><td>第一个轴运动结束点</td></tr>
<tr><td>endy</td><td>第二个轴运动结束点</td></tr>
<tr><td>endz</td><td>第三个轴运动结束点</td></tr>
<tr><td>midx</td><td>第一个轴运动中间点</td></tr>
<tr><td>midy</td><td>第二个轴运动中间点</td></tr>
<tr><td>midz</td><td>第三个轴运动中间点</td></tr>
</table></td></tr>
<tr><td colspan="2">Mode=1：指定圆弧的圆心和结束点，最短路径运动，方向可能为顺时针，也可能为逆时针；</td></tr>
<tr><td>指令格式</td><td>MSPHERICAL(endx, endy, endz, centrex, centrey, centrez, 1)</td></tr>
<tr><td>指令描述</td><td>
<table>
<tr><td>endx</td><td>第一个轴运动结束点</td></tr>
<tr><td>endy</td><td>第二个轴运动结束点</td></tr>
<tr><td>endz</td><td>第三个轴运动结束点</td></tr>
<tr><td>centrex</td><td>空间圆弧球心 x</td></tr>
<tr><td>centrey</td><td>空间圆弧球心 y</td></tr>
<tr><td>centrez</td><td>空间圆弧球心 z</td></tr>
</table></td></tr>
<tr><td>注意</td><td>1.球心、结束点和起点不能共线，使用 mode1 无法绘出半圆；</td></tr>
<tr><td colspan="2">Mode=2：从起点开始，经过第一个点，第二个点，回到起始点；</td></tr>
<tr><td>指令格式</td><td>MSPHERICAL(midx1, midy1, midz1, midx, midy, midz, 2)</td></tr>
</table>

<table>
<tr><td rowspan="2">指令描述</td><td><table>
<tr><td>midx1</td><td>第一个轴第二个中间点</td></tr>
<tr><td>midy1</td><td>第二个轴第二个中间点</td></tr>
<tr><td>midz1</td><td>第三个轴第二个中间点</td></tr>
<tr><td>midx</td><td>第一个轴第一个中间点</td></tr>
<tr><td>midy</td><td>第二个轴第一个中间点</td></tr>
<tr><td>midz</td><td>第三个轴第一个中间点</td></tr>
</table></td></tr>
<tr><td colspan="2">Mode=3：从起点开始，围绕球心，已最短距离到达中间点，返回起点；</td></tr>
<tr><td>指令格式</td><td>MSPHERICAL(midx, midy, midz, centrex, centrey, centrez, 3)</td></tr>
<tr><td>指令描述</td><td><table>
<tr><td>midx</td><td>第一个轴第一个中间点</td></tr>
<tr><td>midy</td><td>第二个轴第一个中间点</td></tr>
<tr><td>midz</td><td>第三个轴第一个中间点</td></tr>
<tr><td>centrex</td><td>空间圆弧球心 x</td></tr>
<tr><td>centrey</td><td>空间圆弧球心 y</td></tr>
<tr><td>centrez</td><td>空间圆弧球心 z</td></tr>
</table></td></tr>
<tr><td>使用注意</td><td>1.球心、结束点和起点不能共线</td></tr>
</table>

ESTUN & Trio

3.18 复杂运动

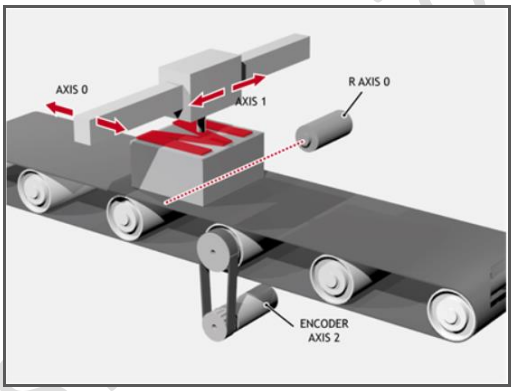
3.18.1轴叠加

轴叠加运动，指轴在执行某条运动指令的时候，可以同时叠加其他轴的运动。控制器支持两种轴叠加方式，分别为：

- ADDAX
- MOVEADD

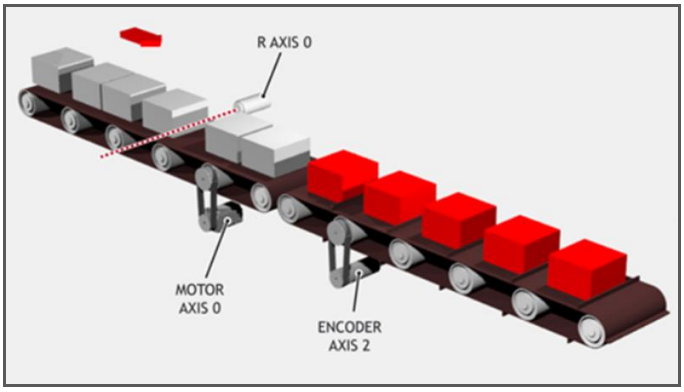
轴叠加运动常用的使用场合有：

- 对象处于运动状态下，对对象进行抓取或者轨迹的加工；比如，在一个移动的物体上进行喷涂；



动态轨迹运动图

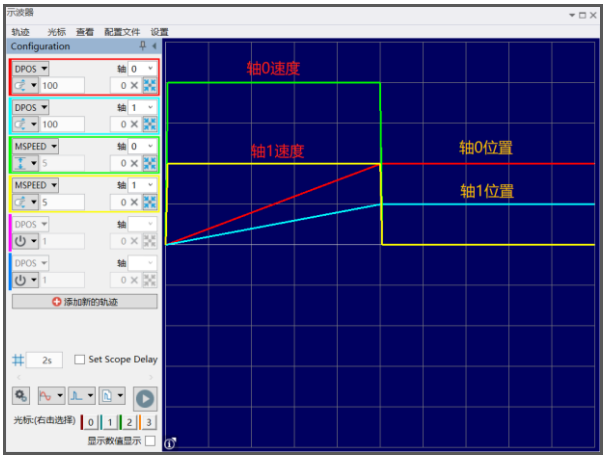
- 轴在按照预设的运动下，需要微调运动量，以达到控制效果，比如，物料在传送带上的等距传输；



物料的等距传输图

3.18.1.1 ADDAX

<table>
<tr><td>指令格式</td><td>ADDAX(axis)</td></tr>
<tr><td>功能描述</td><td>轴命令，通过 AXIS 或者 BASE 指令指定操作轴；
用于建立/取消轴连接；
可以通过轴参数 ADDAX_AXIS 查看当前轴所叠加的轴号；
使用 CANCEL、RAPIDSTOP 指令无法取消轴叠加运动，需要使用 ADDAX(-1) 指令方可取消；
轴运动叠加原则为：
叠加轴的移动量为 X，叠加轴的量纲单位 UNITS 设置为 A，被叠加轴的量纲单位 UNITS 设置为 B，那么被叠加轴的叠加量为：X*A/B
<table>
<tr><td>参数</td><td>描述</td></tr>
<tr><td>axis</td><td>叠加轴轴号；如果该值为-1，则表示取消叠加</td></tr>
</table></td></tr>
<tr><td>实例</td><td>将轴 1 的运动叠加到轴 0 上；
<pre>
FOR i = 0 TO 1
    BASE(i)
    DEFPOS(0)
    UNITS = 1000
    SPEED = 10
    ACCEL = 100
    DECEL = 100
NEXT i

ADDAX(1) AXIS(0)   ' 进行轴叠加
MOVE(100) AXIS(0)
MOVE(100) AXIS(1)
TRIGGER
</pre>
采集的位置-速度波形：


取消轴 0 当前所有的叠加运动；</td></tr>
</table>

ESTUN & Trio

ADDAX(-1) AXIS(0)

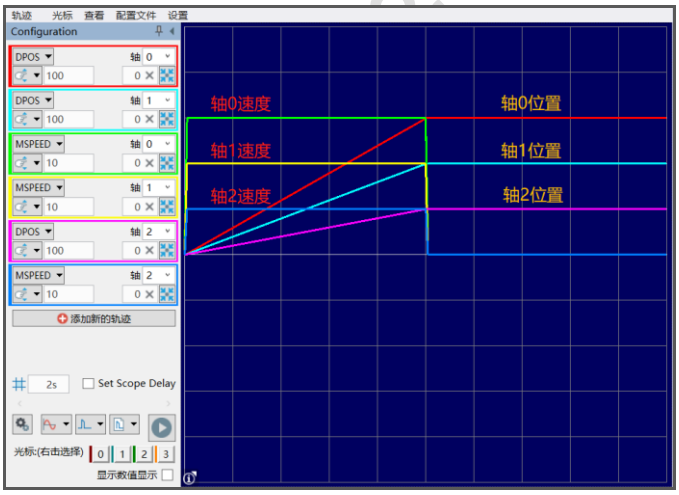
叠加多个轴运动；（轴 0 需要同时叠加轴 1 和轴 2 的运动）

```
FOR i = 0 TO 2
    BASE(i)
    DEFPOS(0)
    UNITS = 1000
    SPEED = 10
    ACCEL = 100
    DECEL = 100
NEXT i

ADDAX(2) AXIS(1) '轴 2 叠加至轴 1
ADDAX(1) AXIS(0) '轴 1 叠加至轴 0

MOVE(100) AXIS(0)
MOVE(100) AXIS(1)
MOVE(100) AXIS(2)
TRIGGER
```

采集的位置-速度波形：



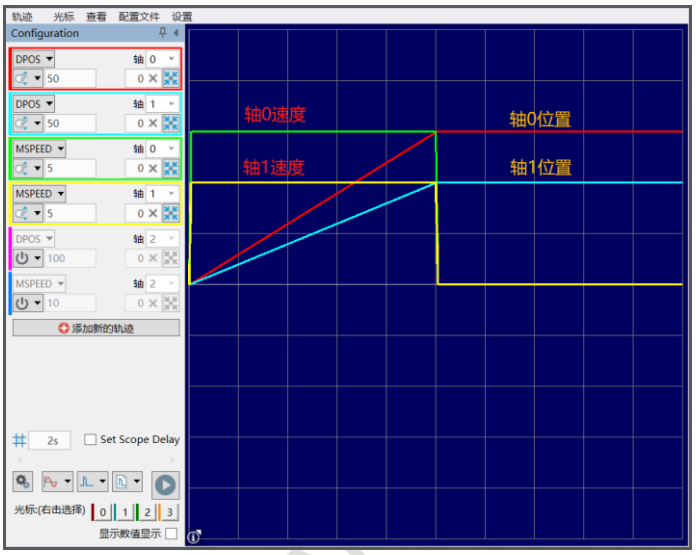
叠加量由叠加轴和被叠加轴的 UNITS 共同决定；

```
BASE(0)
DEFPOS(0)
UNITS = 1000
SPEED = 10
ACCEL = 100
DECEL = 100
BASE(1)
DEFPOS(0)
UNITS = 500
SPEED = 10
```

```
ACCEL = 100
DECEL = 100

ADDAX(1) AXIS(0) '进行轴叠加
MOVE(100) AXIS(0)
MOVE(100) AXIS(1)
TRIGGER
```

采集的位置-速度波形：



分析：

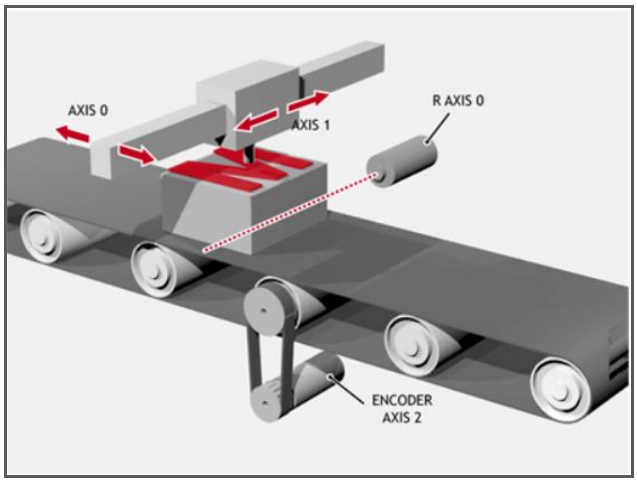
轴 1 的运动叠加至轴 0，轴 1 的移动量为 100，由于轴 1 的 UNITS 单位为轴 0 的 UNITS 的 0.5 倍，所有轴 0 叠加的移动量为：1000*0.5=500

3.18.1.2 ADDAX_AXIS

功能描述	轴参数，只读，通过 AXIS 或者 BASE 指令指定读取轴； 反馈指定轴当前叠加的轴号； 默认值为-1，表示当前无叠加轴；

3.18.1.3 案例分析

3.18.1.3.1 动态过程的轨迹加工



动态轨迹运动图

参考上图，要求在移动的箱体表面完成字母 M 的喷涂；整个系统包含 3 个实轴：

- 轴 0：运动轴，完成喷涂 X 轴方向的运动；
- 轴 1：运动轴，完成喷涂 Y 轴方向的运动；
- 轴 2：反馈轴，皮带位置的反馈；

具体要求：

- 当箱体感应到安装的皮带测的光电信号，开始执行喷涂运动；
- 箱体在喷涂的过程中，皮带处于运动状态；

实现思路：

- 轴 0 的运动由两部分组成，一部分是轨迹运动时 X 方向的变化值，一部分为皮带移动量；
- 轨迹运动时 X 的方向使用虚拟轴运动，通过轴叠加的方式至轴 0；
- 皮带移动量可使用电子齿轮方式实现，其中轴 2 为反馈轴，轴 0 为从动轴

实现代码：

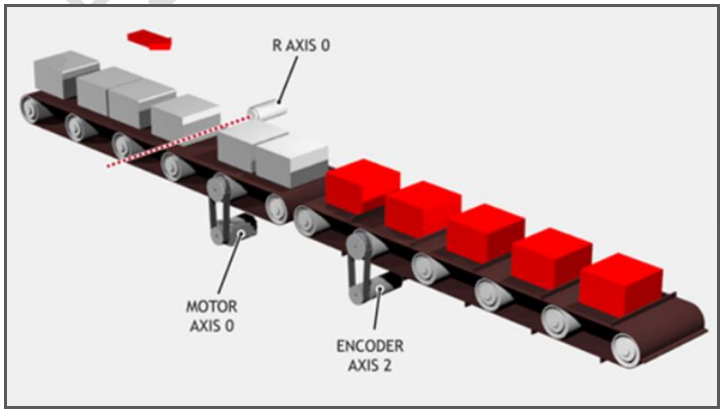
```
'将轴 3 的运动叠加至轴 0
UNITS AXIS(3) = UNITS AXIS(0)
ATYPE AXIS(3) = 0 'set axis 3 as virtual axis
SERVO AXIS(3) = ON
DEFPOS(0) AXIS(3)
ADDAX(3) AXIS(0) 'connect axis 3 requirement to axis 0
```

ESTUN & Trio

```
DIM di_sensor_channel AS INTEGER
di_sensor_channel = 2
'离开光电
WAIT UNTIL IN(di_sensor_channel) = ON
'开启色标捕捉功能
REGIST(3) 'registration input detects a box on the conveyor
WAIT UNTIL MARK OR IN(2) = OFF
'色标捕捉完成，开始加载轨迹运动
'将轴 0 和轴 2 建立 1:1 的电子齿轮比
CONNECT(1, 2) AXIS(0)
'加载轨迹运动
BASE(3, 1)
'Draw the M
MOVEABS(1200, 0)'move A > B
MOVEABS(600, 1500)'move B > C
MOVEABS(1200, 3000)' move C > D
MOVEABS(0, 0)'move D > E

'等待运动完成，取消电子齿轮连接，返回起始位置
WAIT IDLE
BASE(0)
CANCEL 'stop axis 0 from following the belt
WAIT IDLE
MOVEABS(0) 'move axis 0 to home position
```

3.18.1.3.2 物料等距传输



物料等距传输图

参考上图，传输带上箱体来料间距不等，需要物料在通过光电检测后，在下一段皮带上箱体之间等距排序；

系统硬件组成：

- 轴 0：运动轴，控制第一段皮带的运动；

ESTUN & Trio

- 轴 2：运动轴，控制第二段皮带的运动；
- 光电检测功能；

实现思路：

- 轴 2 作为主轴，轴 0 作为从轴，两轴之间电子齿轮比为 1:1；
- 建立虚拟轴 1，当感应到光电时，计算两次光电感应之间的位置差 A，求位置差 A 与料盒之间的设定间距 B 之差 C；
- 虚拟轴 1 执行相对运动，运动量为差值 C，同时将轴 1 的运动叠加至轴 0；

实现代码：

```
'间距设置
expected = 2000 'sets expected position
'建立轴叠加和电子齿轮关系
BASE(0)
ADDAX(1)
CONNECT(1, 2)

'记录第一次感应位置
BASE(0)
REGIST(3)
WAIT UNTIL MARK
ini_pos = REG_POS

WHILE TRUE

  '两次料盒位置差
  BASE(0)
  REGIST(3)
  WAIT UNTIL MARK
  seenat = REG_POS - ini_pos
  ini_pos = REG_POS

  '轴 1 执行补偿运动
  offset = expected - seenat
  MOVE(offset) AXIS(1)

WEND
```

ESTUN & Trio

3.18.2轴叠加 2

用户也可以使用 MOVEADD 指令实现轴的叠加运动控制，MOVEADD 和 ADDAX 的区别在于：MOVEADD 的指定轴号即为叠加轴，不占用虚拟轴（比如轴 0 正在运动，需要在轴 0 上实现叠加功能，则给轴 0 执行 MOVEADD 运动指令）。

3.18.2.1 MOVEADD

指令格式	MOVEADD(distance)
功能描述	轴命令；通过 AXIS 或者 BASE 指令指定操作轴； 用于在当前轴运动上叠加一次补偿运动； 使用 CANCEL(5)可对 MOVEADD 运动进行清除； 通过 FORCE_SPEED、FORCE_ACCEL、FORCE_JERK 设置叠加运动的速度、加速度、加加速度； 通过轴参数 AXISSTATUS 的 BIT28、BIT29 可查看 MOVEADD 运动的状态； 指令格式：

参数	描述
distance	补偿距离，单位为 UNITS

3.18.2.2 FORCE_SPEED

功能描述	轴参数；通过 AXIS 或者 BASE 指令指定操作轴； 设置 MOVEADD 运动的速度，单位为 UNITS/s;

3.18.2.3 FORCE_ACCEL

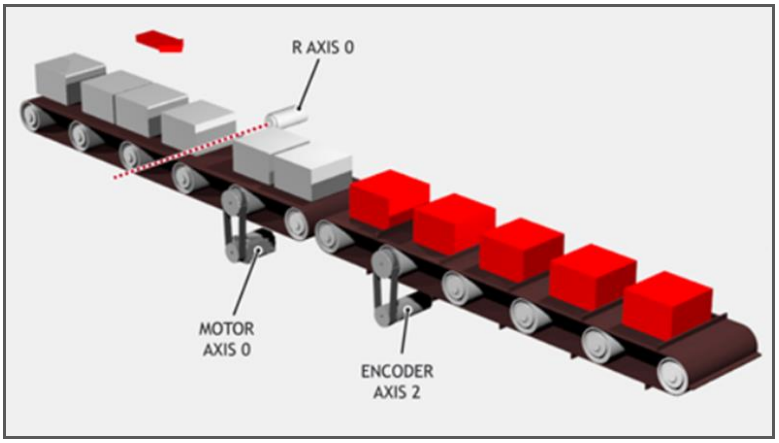
功能描述	轴参数；通过 AXIS 或者 BASE 指令指定操作轴； 设置 MOVEADD 运动的加速度，单位为 UNITS/s^2;

3.18.2.4 FORCE_JERK

功能描述	轴参数；通过 AXIS 或者 BASE 指令指定操作轴； 设置 MOVEADD 运动的加加速度，单位为 UNITS/s^3;

ESTUN eTrio

3.18.2.5 案例分析



物料的等距传输图

参考上图 ，传输带上箱体来料间距不等，需要物料在通过光电检测后，在下一段皮带上箱体之间等距排序；

系统硬件组成：

- 轴 0：运动轴，控制第一段皮带的运动；
- 轴 2：运动轴，控制第二段皮带的运动；
- 光电检测功能；

实现思路：

- 轴 2 作为主轴，轴 0 作为从轴，两轴之间电子齿轮比为 1:1；
- 当感应到光电时，计算两次光电感应之间的位置差 A，求位置差 A 与料盒之间的设定间距 B 之差 C；使用 MOVEADD 指令进行补偿处理；

代码实现：

```
'间距设置
expected = 2000 'sets expected position
'建立轴叠加和电子齿轮关系
BASE(0)
CONNECT(1, 2)

'记录第一次感应位置
BASE(0)
REGIST(3)
WAIT UNTIL MARK
ini_pos = REG_POS

WHILE TRUE
```

ESTUN & Trio

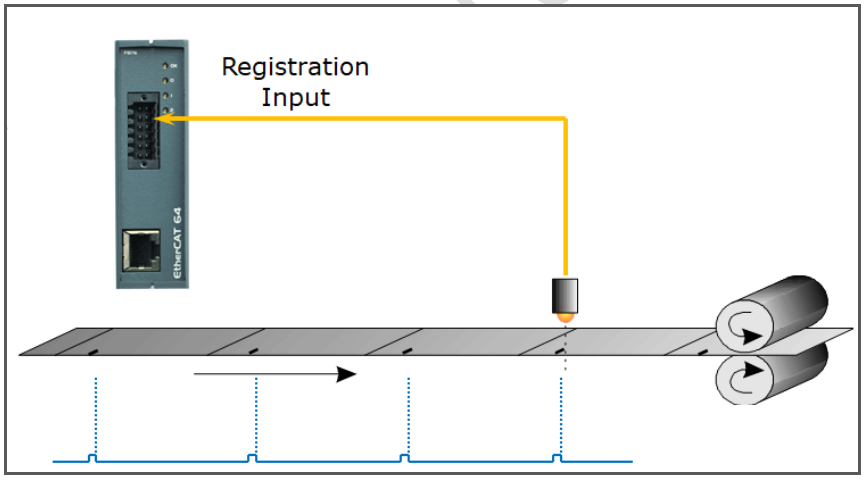
```
'两次料盒位置差
BASE(0)
REGIST(3)
WAIT UNTIL MARK
seenat = REG_POS - ini_pos
ini_pos = REG_POS

'轴 1 执行补偿运动
offset = expected - seenat
MOVEADD(offset) AXIS(0)

WEND
```

3.18.3 位置锁存功能

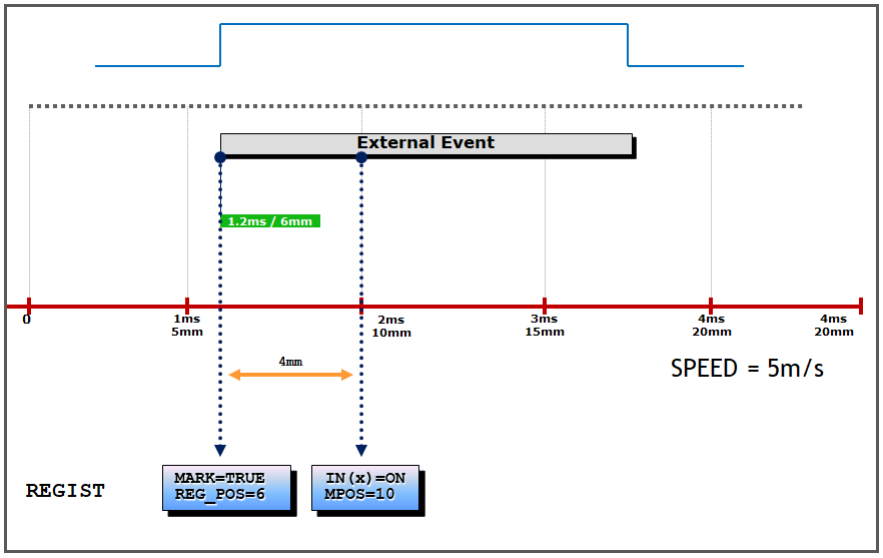
位置锁存功能，又称探针功能，色标捕捉功能。指轴在运动过程中，能锁存外部 DI 信号或电机 Z 信号发生变化时的位置信息。位置锁存功能，相较于使用程序逻辑实现位置存储，更加精确。



位置锁存功能在圆筒印刷上的应用

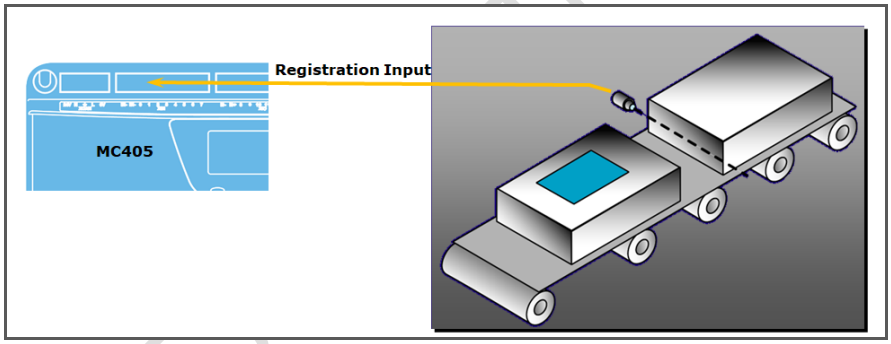
TRIO 控制器支持多种位置锁存功能，根据 DI 信号的接入对象，可分为：

- 控制器位置锁存功能，DI 信号连接至控制器的高速输入信号；
- 伺服驱动器位置锁存功能，DI 信号接入至驱动器的高速输入信号（ETHERCAT 总线控制模式下，控制器通过操作位置锁存控制字，实现位置锁存功能）；



位置锁存与逻辑位置存储区别

3.18.3.1 基于控制器的位置锁存

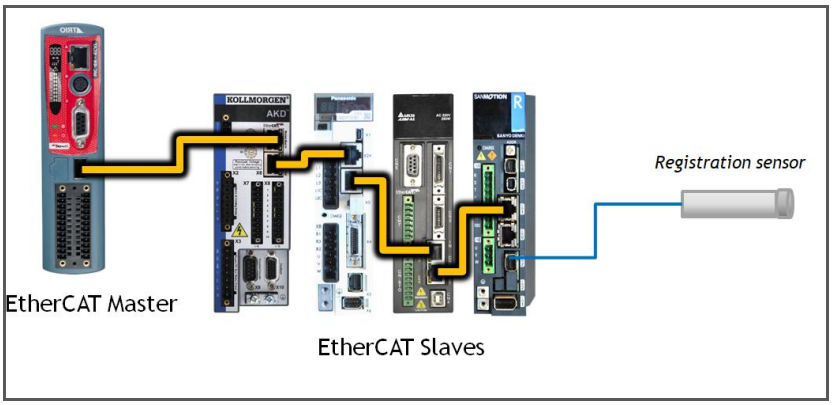


基于控制器的色标捕捉

当色标检测开关连接至控制器的输入端子：

- 可通过 RESIST 指令下方检测色标状态命令；
- 可通过 MARK、REG_POS、REG_SPEED 读取状态；
- 可通过 REGIST 设定色标输入通道；

3.18.3.2 基于驱动器的位置锁存



基于驱动器的位置锁存示意图

当色标检测开关连接至驱动器的输入端子，用户也可通过 REGIST 等指令实现位置锁存功能，但需要满足以下前提：

- 驱动器支持位置锁存功能；
- 位置锁存关联的对象需要配置 PDO；

对象地址	关键字	BASIC 指令
60B8:00	TORCH_PROBE_FUNCTION	REGIST 和 DATUM
60B9:00	TORCH_PROBE_STATUS	REGIST 和 DATUM
60BA:00	TORCH_PROBE1_POS1	REGIST 和 DATUM
60BB:00	TORCH_PROBE1_POS2	REGIST 和 DATUM
60BC:00	TORCH_PROBE2_POS1	REGIST 和 DATUM
60BD:00	TORCH_PROBE2_POS2	REGIST 和 DATUM

3.18.3.3 相关指令

3.18.3.3.1 REGIST

指令格式	REGIST(mode, channel, source, edge, window [,quantity, table_start])	
参数	名称	含义
	Mode	模式选择，20 或者 21；
	Channel	输入通道选择
	Source	输入通道类型选择 0：捕捉信号为第一个 REGIST_INPUT 信号； 1：捕捉信号为 Z 脉冲信号； 2：捕捉信号为第二个 REGIST_INPUT 信号；

		3：捕捉信号为普通的输入信号
	Edge	触发沿选择
	windoes	0:触发位置范围无限制； 1:触发位置范围在 OPEN_WIN----CLOSE_WIN 之间 2: 触发位置范围在 OPEN_WIN----CLOSE_WIN 之外；

3.18.3.3.2 MARK

指令描述	轴参数； 返回色标捕捉状态； 当执行 REGIST 指令时，MARK 状态变为 FALSE； 当色标输入信号生效时，MARK 状态变为 TRUE；

3.18.3.3.3 MARKB

指令描述	轴参数； 返回色标捕捉状态； 当执行 REGIST 指令时，MARKB 状态变为 FALSE； 当第二路色标输入信号生效时，MARK 状态变为 TRUE；

3.18.3.3.4 REGIST_POS

指令描述	轴参数； 返回第一路位置锁存信息，单位为 UNITS；

3.18.3.3.5 REGIST_POSB

指令描述	轴参数； 返回第二路位置锁存信息，单位为 UNITS；

3.18.3.3.6 REGIST_SPEED

指令描述	轴参数； 返回第一路速度锁存信息，单位为 UNITS/S；

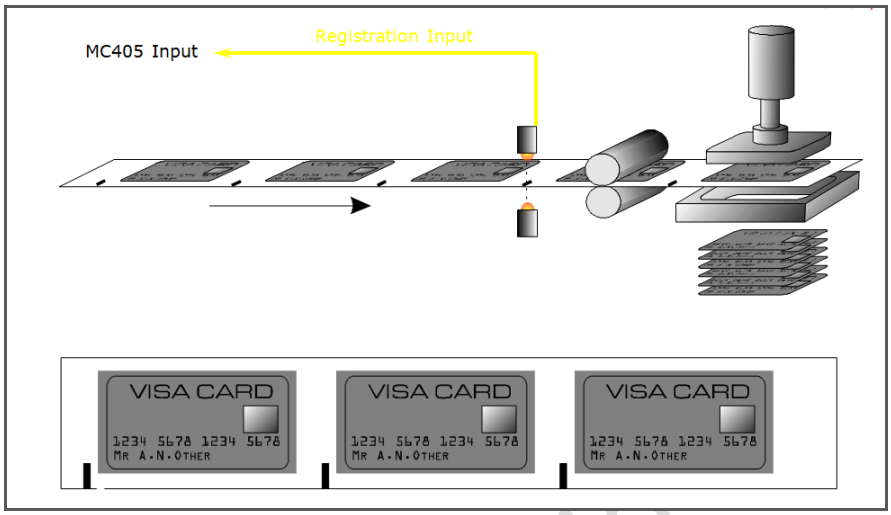
3.18.3.3.7 REGIST_SPEEDB

指令描述	轴参数； 返回第二路速度锁存信息，单位为 UNITS/S；

ESTUN & Trio

3.18.3.4 应用案例

3.18.3.4.1 印刷应用



位置锁存印刷应用示意图

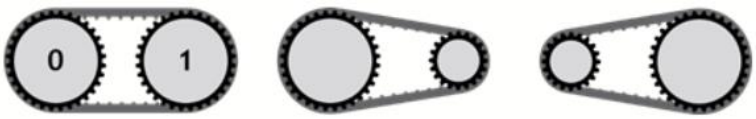
参考上图，传送带上方安装色标传感器，接入控制器，当感应到皮带上的黑色区域，锁存位置，并在此基础上向前运动一定偏移量。

代码实现：

```
loop:
WAIT UNTIL IN(punch_clr) = ON
MOVE(index_length)
REGIST(20, 0, 2, 0, 0)
WAIT UNTIL MARK
MOVEMODIFY(REG_POS + offset) '修改运动指令停止位置
WAIT IDLE
GOTO loop
```

3.18.4电子齿轮

电子齿轮功能，指从动轴按照设定的运动比例（齿轮比），跟随主轴运动。



电子齿轮运动示意图

主轴：可以为虚拟轴，也可以为实轴（ETHERCAT 位置轴、编码器轴等），每个伺服周期

主轴的变化量会叠加至从轴；主轴可以为单个轴，使用 CONNECT 指令连接；主轴也可以为运动轴组，使用 CONPATH 指令连接；

从轴：可以为虚拟轴，也可以为实轴，一般为 ETHERCAT 位置轴或者脉冲位置轴，每个周期的运动量为主轴上一个周期的变化量乘以齿轮比；

齿轮比：设定值可为正值，也可为负值，当执行一条新的 CONNECT 指令时，可以改变电子齿轮比；当电子齿轮比发生修改时，电子齿轮比的切换时间由轴参数 CLUTCH_RATE 决定；

运动特性：可使用 CANCEL 及 RAPIDSTOP 指令取消电子齿轮运动，默认情况下，当从轴触发限位保护时，运动也会被取消；如果需要在触发限位保护时，电子齿轮运动不受影响，可设置 AXIS_MODE；

状态查看：通过轴参数 LINK_AXIS 可查看连接的主轴轴号。

3.18.4.1 相关指令

3.18.4.1.1 CONNECT

指令格式	CONNECT(ratio, driving_axis)
功能描述	轴命令，通过 BASE 或者 AXIS 指定操作轴； 电子齿轮运动的从轴执行该指令； 建立从轴与主轴的电子齿轮比；

参数	描述
ratio	电子齿轮比
driving_axis	主轴轴号

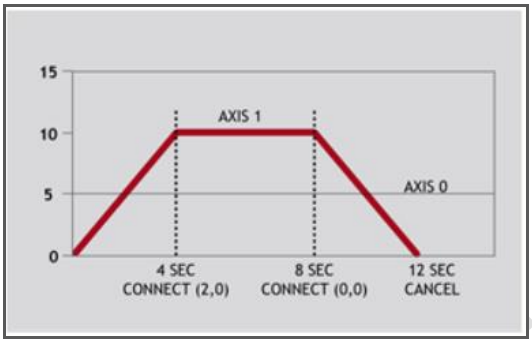
3.18.4.1.2 CONPATH

指令格式	CONNPATH(ratio , driving_axis)
功能描述	轴命令，通过 BASE 或者 AXIS 指定操作轴； 电子齿轮运动的从轴执行该指令； 建立从轴与轴组的电子齿轮比；

参数	描述
ratio	电子齿轮比
driving_axis	主轴轴号

ESTUN & Trio

3.18.4.1.3 CLUTCH_TARE

功能描述	轴参数，通过 BASE 或者 AXIS 指定操作轴； 电子齿轮运动的从轴执行设置该参数； 默认值为 1000000； 当使用 CONNECT 或者 CONPATH 修改电子齿轮比时，电子齿轮比修改至目标值的时间为：变化量/CLUTCH_TARE，单位为 s
样例	 轴 1 与轴 0 之间的速度比要求如图，代码实现： FORWARD AXIS(0) BASE(1) CONNECT(0, 0) 'set initial ratio to zero CLUTCH_RATE = 0.5 'set clutch rate CONNECT(2, 0) 'apply the required connect ratio WA(8000) CONNECT(0, 0) 'apply zero ratio to disconnect WA(4000) 'wait for deceleration to complete CANCEL 'cancel connect

3.18.4.1.4 AXIS_MODE

功能描述	轴参数，通过 BASE 或者 AXIS 指定操作轴； 电子齿轮运动的从轴执行设置该参数； AXIS_MODE 的 BIT2 置 1，当执行 CONNECT 运动时，触发限位保护，不会清除当前 CONNECT 指令

3.18.4.1.5 LINK_AXIS

功能描述	轴参数，通过 BASE 或者 AXIS 指定操作轴； 当执行 CONNECT、COMBOX、MOVELINK、FLEXLINK 等指令时，该参数可反馈当前轴连接的主轴；

ESTUN & Trio

3.18.4.2 案例分析

3.18.4.2.1 电子齿轮：主轴为单个控制轴

在龙门运动控制、收放卷的应用中，需要两轴的运动保持同步运动，可使用电子齿轮功能，主轴与从轴之间的电子齿轮比设置为 1。

以龙门结构为例，龙门结构由两个实轴组成，分别为轴 0 和轴 1。

- 方案 1:轴 1 作为从轴，轴 0 作为主轴；

```
DIM gantry_0, gantry_1 AS INTEGER
DIM gantry_diff AS INTEGER
gantry_0 = 0 '龙门轴 0
gantry_1 = 1 '龙门轴 1
gantry_diff = 2 '龙门轴位置偏差
BASE(gantry_diff)
ATYPE = 85
AXESDIFF(gantry_0, gantry_1)

'建立两轴电子齿轮比
DIM ratio AS FLOAT
ratio = 1
CONNECT(ratio, gantry_0) AXIS(gantry_1)

'主轴开始运动，带动从轴运动
BASE(gantry_0)
SPEED = 1000
ACCEL = 10000
DECEL = 10000
MOVE(10000)

TRIGGER
```

分析：由于轴 1 的运动是轴 0 上一个运动的变化量，所以在运动过程中，轴 0 与轴 1 的位置偏差为主轴一个伺服周期的移动量。

- 方案 2：轴 0 及轴 1 均作为从轴，连接相同的主轴，主轴为虚拟主轴；

```
DIM gantry_0, gantry_1 AS INTEGER
DIM gantry_diff AS INTEGER
DIM gantry_master AS INTEGER
gantry_master = 3
gantry_0 = 0 '龙门轴 0
gantry_1 = 1 '龙门轴 1
```

ESTUN & Trio

```
gantry_diff = 2 '龙门轴位置偏差
BASE(gantry_diff)
ATYPE = 85
AXESDIFF(gantry_0, gantry_1)

'建立两轴电子齿轮比
DIM ratio AS FLOAT
ratio = 1
CONNECT(ratio, gantry_master) AXIS(gantry_0)
CONNECT(ratio, gantry_master) AXIS(gantry_1)

'主轴开始运动，带动从轴运动
BASE(gantry_master)
SPEED = 1000
ACCEL = 10000
DECEL = 10000
MOVE(10000)

TRIGGER
```

分析：从轴与主轴之间均存在一个伺服周期的移动量偏差，两个从轴之间不存在偏差。

3.18.4.2.2 电子齿轮：主轴为轴组

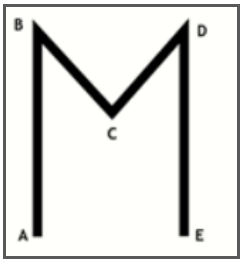
在点胶、高速拍照等应用场合，高速位置比较输出功能的位置由多轴坐标决定。用户可以通过轴组的电子齿轮比功能，将多轴坐标转换为从轴的单轴坐标，简化位置比较输出功能的控制

实现方法：

- 将虚拟轴与插补轴组的基准值使用 CONPATH 指令建立电子齿轮关系；
- 根据多轴的坐标，计算矢量位置，用于虚拟轴的位置比较输出；

样例描述：

- X、Y 轴（轴 0、轴 1）在平面上执行直线插补运动，其轨迹为：



- 要求在点 A、B、C、D、E 处触发拍照；

ESTUN & Trio

代码实现：

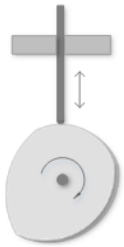
```
DIM expos AS FLOAT(10) '虚轴拍照坐标

CONNPATH(1, 0) AXIS(2)
DEFPOS AXIS(2)


expos(0) = 12
expos(1) = expos(0) + SQR(3 ^ 2 + (-6) ^ 2)
expos(2) = expos(1) + SQR(3 ^ 2 + (6) ^ 2)
expos(3) = expos(2) + 12
'加载触发位置至 TABLE 区
'执行高速位置比较输出指令
'参考 HW_PSWTICH 指令的使用
'执行轨迹运动
BASE(0, 1)
MOVE(0, 12) 'move A >  B
MOVE(3, -6) 'move B >  C
MOVE(3, 6) 'move C >  D
MOVE(0, -12)'move D > E
```

ESTUN & Trio

3.18.5电子凸轮



机械凸轮示意图

机械凸轮由主动件和从动件。主动件为在金属盘上加工轮廓曲线，一般为匀速旋转运动；从动件一般与凸轮轮廓接触，主动件在运动时，从动件往复运动，运动轨迹由凸轮轮廓决定。

由于凸轮机构属于高副机构，故凸轮与从动件之间为点或线接触，不便润滑、易于磨损，有噪声，盘片的加工制造要求较高，维修复杂。因此在很多应用上，使用电子凸轮替代机械凸轮。

电子凸轮，指的是用户通过构造凸轮曲线，以到达主动件与从动件之间的运动关系，不需要额外的机械结构辅助完成。

TRIO 控制器为用户提供了多种凸轮运动形式：

- CAM：凸轮表运动；

- CAMBOX：跟随凸轮表运动；
- MOVELINK：特定轨迹凸轮运动，又称追剪运动，一般应用于追剪应用；
- FLEXLINK: 特定轨迹凸轮运动，又称飞剪运动，一般应用于飞剪应用；

3.18.5.1 凸轮表运动

凸轮表运动，用户可以通过存储在 TABLE 区域的位置信息，形成运动轨迹。

3.18.5.1.1 指令格式-CAM

指令格式	CAM(start point, end point, table multiplier, distance)	
功能描述	轴命令，通过 BASE 或者 AXIS 指定操作轴； 指令轴按照凸轮表中的位置信息运动； 位置信息存储在连续的 TABLE 区域；	
参数	名称	含义
	start point	存储位置信息 TABLE 区域的开始地址
	end point	存储位置信息 TABLE 区域的结束地址
	table multiplier	TABLE 区域位置信息的放大倍数；
	distance	距离参数，用于决定凸轮运动的运动时间；
使用注意	TABLE 区域存储的位置信息为增量位置信息； TABLE 区域存储的位置信息单位为脉冲； CAM 指令运动时间为：distance/Speed; 执行 CAM 指令时，要求 ACCEL 的设定值为 SPEED 的 1000 倍及以上；	

3.18.5.1.2 应用案例

1. 使用 CAM 指令实现 Sin 曲线运动；

   代码：

```
'填写凸轮表
'SIN 曲线的波峰为 50 个量纲单位
DIM tb_start_addr AS INTEGER
DIM tb_end_addr AS INTEGER

tb_start_addr = 0
tb_end_addr = tb_start_addr + 360
```

```
DIM sin_maxval AS FLOAT
sin_maxval = 50

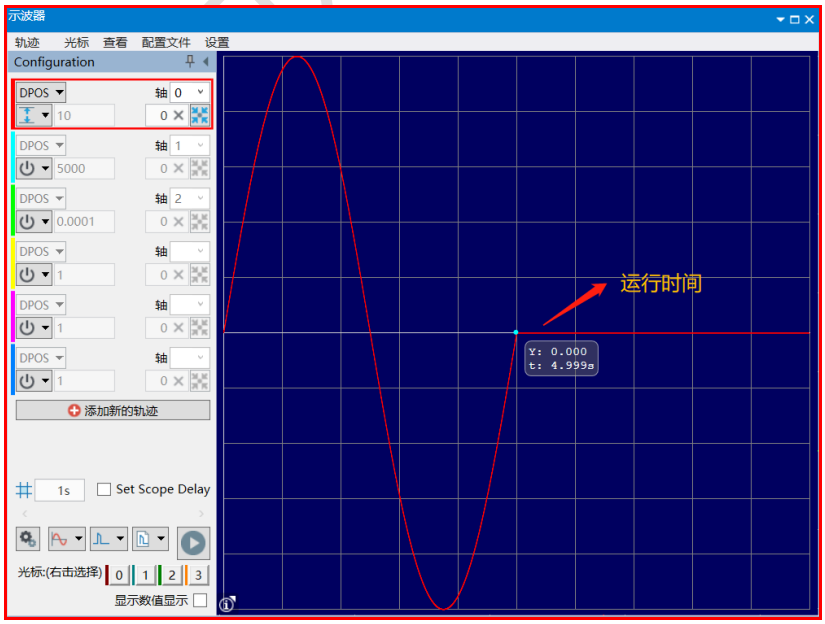
FOR i = 0 TO 360 'loop to fill in the table
   rad = i * 2 * PI / 360 'convert degrees to radians
   x = SIN(rad) * sin_maxval
   TABLE(tb_start_addr + i , x) 'place value of x in table
NEXT i

'触发单轴运动
'5S 内完成一次凸轮表运动
BASE(0)
UNITS = 1000
SPEED = 100
ACCEL = SPEED * 1000
DECEL = SPEED * 1000

DIM run_time AS FLOAT
run_time = 5

CAM(0, 360, UNITS AXIS(0), run_time * SPEED AXIS(0))
TRIGGER
```

运行效果图：



基于 CAM 实现的 SIN 曲线图

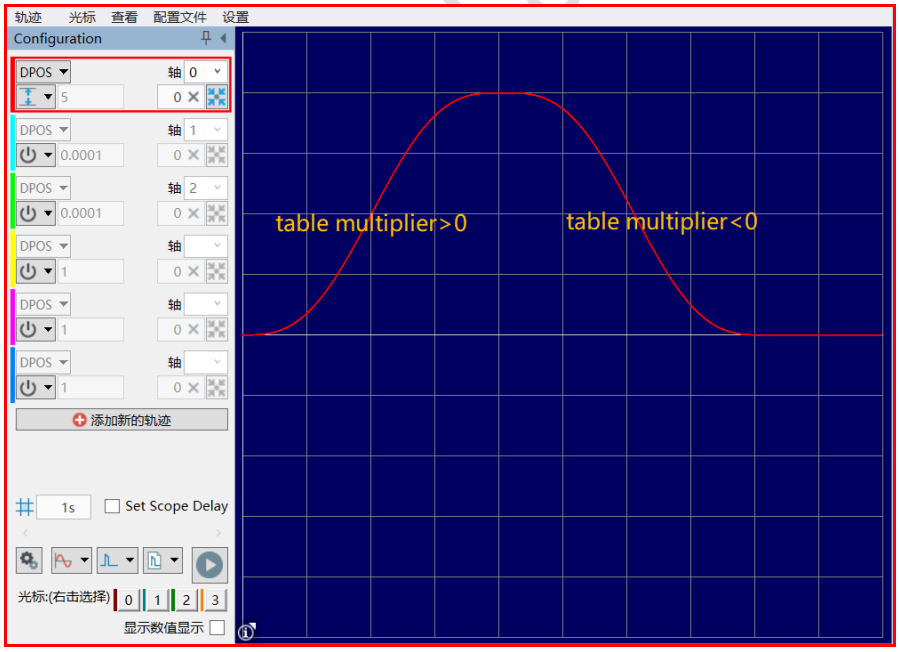
2. table multiplier 为负值，可实现反向凸轮运动；

代码：

```
GOSUB profile_gen 'loads the cam profile into the table
TRIGGER
BASE(0)
SPEED = 10000 : ACCEL = SPEED * 1000 : DECEL = SPEED * 1000
'capture load value
distance = SPEED * 4 'calculate the distance parameter
CAM(0 , 200 , 50 , distance) 'move 50mm forward in time calculated
WAIT IDLE
WA(100)
CAM(0 , 200 , -50 , distance) 'move back to pick up position
STOP

profile_gen:
num_p = 201
scale = 400 'set scale so that multiplier is in mm
FOR p = 0 TO num_p - 1
   TABLE(p, ((-SIN(PI * 2 * p / num_p) / (PI * 2)) + p / num_p) * scale)
NEXT p
RETURN
```

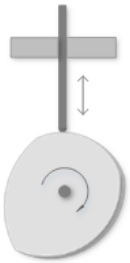
运行效果图：



CAM 执行效果图

3.18.5.2 跟随凸轮表运动

跟随凸轮表运动，指从轴跟随主轴运动，两轴之间的位置关系由凸轮表的位置信息决定。



机械凸轮示意图

参考机械凸轮示意图，跟随凸轮表运动中的主轴相当于旋转的金属盘，从轴相当于上下运动的连杆结构，凸轮表相当于金属盘的轮廓曲线。

主轴：可以为虚拟轴，也可以为实轴，通常为旋转结构，匀速运动；

从轴：一般为实轴，通常为直线结构；

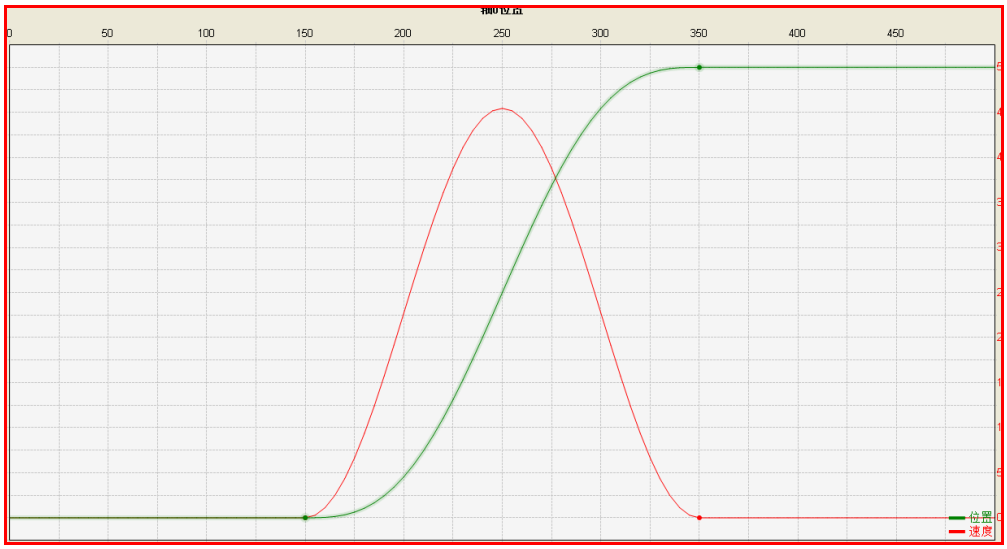
3.18.5.2.1 指令格式-CAMBOX

指令格式	CAMBOX(start_point, end_point, table_multiplier, link_distance , link_axis[, link_options][, link_pos][, offset_start])		
功能描述	轴命令，通过 BASE 或者 AXIS 指定操作轴； 从轴跟随主轴运动，位置关系存储于 TABLE 区域；		
参数	**名称**	**含义**	
	start point	存储位置信息 TABLE 区域的开始地址	
	end point	存储位置信息 TABLE 区域的结束地址	
	table multiplier	TABLE 区域位置信息的放大倍数；	
	Link_distance	当主轴移动距离到达该值时，从轴完成凸轮运动；	
	Link_axis	主轴轴号	
	Link_Option	可选参数	
		Bit0	当 MARK 信号发生时，开始凸轮运动
		Bit1	当主轴到达某个位置，开始凸轮运动
使用注意	TABLE 区域存储的位置信息为增量位置信息； TABLE 区域存储的位置信息单位为脉冲； CAM 指令运动时间为：distance/Speed; 执行 CAM 指令时，要求 ACCEL 的设定值为 SPEED 的 1000 倍及以上；		

3.18.5.2.2 应用案例

从轴跟随主轴运动，凸轮表信息存储至 TABLE(1000)-TABLE(1101)，主轴运动量为 500，

从轴运动量参考凸轮表，凸轮规划位置-速度曲线如下：



实现代码如下：

```
tablestart = 1000
points = 101
' store the table points
TABLE(tablestart, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000)
TABLE(tablestart + 8, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000)
TABLE(tablestart + 16, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000)
TABLE(tablestart + 24, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0103)
TABLE(tablestart + 32, 0.0818, 0.2745, 0.6451, 1.2460, 2.1241, 3.3192, 4.8635, 6.7805)
TABLE(tablestart + 40, 9.0845, 11.7805, 14.8635, 18.3192, 22.1241, 26.2460, 30.6451)
TABLE(tablestart + 47, 35.2745, 40.0818, 45.0103, 50.0000, 54.9897, 59.9182, 64.7255)
TABLE(tablestart + 54, 69.3549, 73.7540, 77.8759, 81.6808, 85.1365, 88.2195, 90.9155)
TABLE(tablestart + 61, 93.2195, 95.1365, 96.6808, 97.8759, 98.7540, 99.3549, 99.7255)
TABLE(tablestart + 68, 99.9182, 99.9897, 100.0000, 100.0000, 100.0000, 100.0000)
TABLE(tablestart + 74, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000)
TABLE(tablestart + 80, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000)
TABLE(tablestart + 86, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000)
TABLE(tablestart + 92, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000, 100.0000)
TABLE(tablestart + 98, 100.0000, 100.0000, 100.0000)
' set the variables
tablestop = tablestart + points - 1
m_axis = 1
l_axis = 0
l_dist = 500
' options = no reg_start + no pos_start + no repeat
options = 0 + 0 + 0
start_pos = 0
' perform the move
```

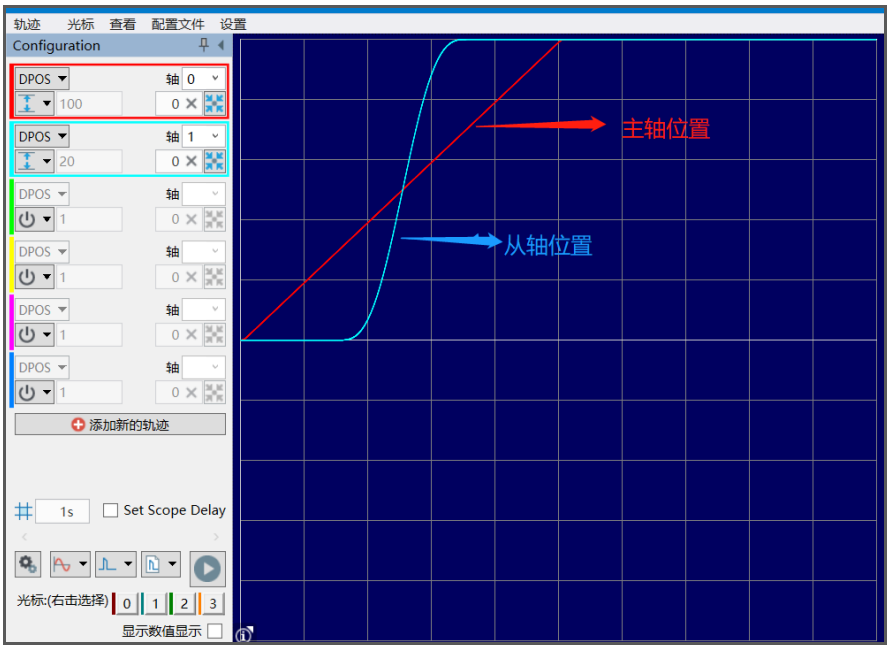
ESTUN & Trio

```
CAMBOX(tablestart, tablestop, UNITS AXIS(m_axis), l_dist, l_axis, options, start_pos) AXIS(m_axis)
'Stop CamGen Sector1

BASE(l_axis)
SPEED = 100
ACCEL = 1000
DECEL = 1000
MOVE(l_dist)

TRIGGER
```

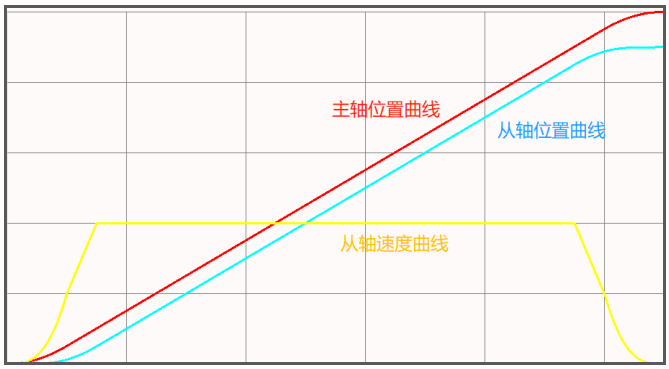
运行效果图：



3.18.5.3 特定轨迹凸轮运动-MOVELINK

MOVELINK 指令，是凸轮运动的一种，用户可以通过简单设置几个相关参数，便可以构建主轴与从轴之间的运动关系，位置关系不是存储于 TABLE 表中。

默认情况下，MOVELINK 的位置关系参考下图：



MOVELINK 主从默认位置-速度关系图

用户在使用 MOVELINK 编辑主-从轴之间的位置关系时，需要设置从轴的移动量、主轴的移动量、从轴加速阶段主轴的移动量，从轴减速阶段主轴的移动量等参数。

MOVELINK 指令的常用应用场合有：

- 追剪设备；
- 排线设备；
- 绕线设备；

3.18.5.3.1 指令格式-MOVELINK

指令格式	MOVELINK (distance, link_dist, link_acc, link_dec, link_axis[, link_options][, link_pos][,base_dist])	
参数	名称	含义
	distance	跟随轴的移动量（单位为 UNITS）
	Link_dist	主轴的移动量（单位为 UNITS）
	Link_acc	从轴加速阶段主轴的移动量（单位为 UNITS）
	Link_dec	从轴减速阶段主轴的移动量（单位为 UNITS）
	Link_axis	主轴轴号
	Link_options	连接选项
		BIT0 当 MARK 信号发生时，开始凸轮运动
		BIT1 当主轴到达某个位置，开始凸轮运动
		BIT2 自动凸轮重复运动，可以通过 REP_OPTION 的 BIT1 置位，关闭重复运动
		BIT4 设置 S 型曲线加减速
		BIT8 当 MARKB 信号发生时，开始凸轮运动
		BIT9 当 R_MARK 信号发生时，开始凸轮运动
		BIT10 设置速度曲线

ESTUN & Trio

		BIT11	0：SIN 曲线； 1：9 阶速度曲线 2：7 阶速度曲线； 3：5 阶速度曲线
	link_pos	BIT1=1 生效，主轴触发位置	
	base_dist	基准距离	

3.18.5.3.2 应用案例

1. MOVELINK 位置参数设置规则

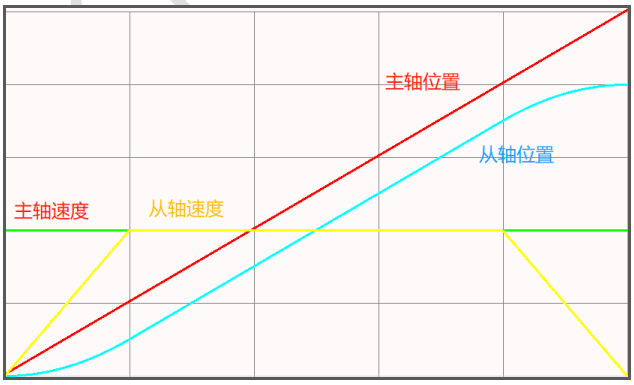
如果需要从轴和主轴在匀速阶段速度保持一致，则要求满足以下公式：

Distance+ Link_acc/2+ Link_dec/2 = Link_dist

样例代码：

```
BASE(0)
IF MTYPE = 0 THEN
   FORWARD
ENDIF
BASE(1)
MOVELINK(4, 5, 1, 1, 0)
```

运行效果图：



MOVELINK 运行效果图

可将一条集成的 MOVELINK 运动指令拆解成多条 MOVELINK 指令，参考上例，另外的写法为：

```
BASE(0)

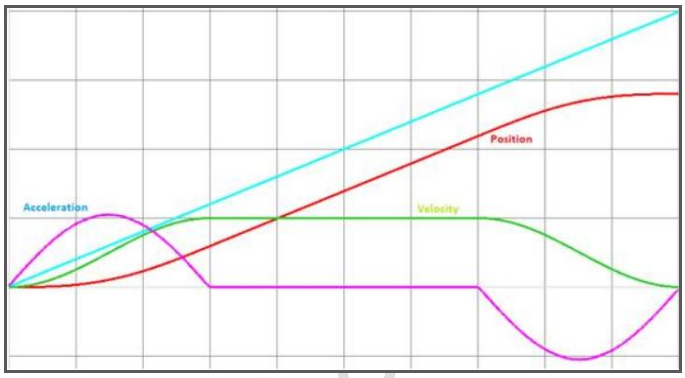
IF MTYPE = 0 THEN
   FORWARD
```

```
ENDIF

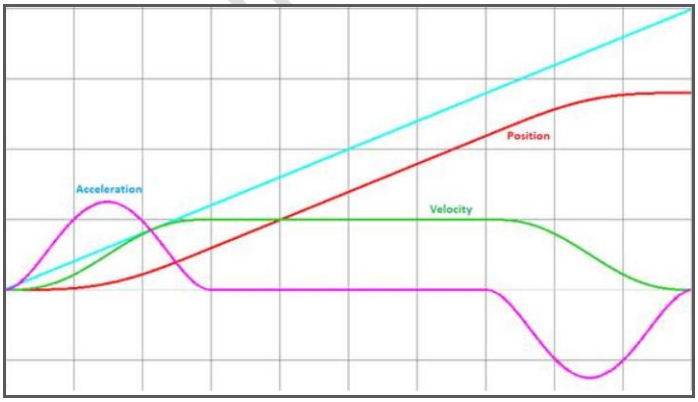
BASE(1)
MOVELINK(0.5, 1, 1, 0, 0)'ACCEL
MOVELINK(3, 3, 0, 0, 0)'SAME SPEED
MOVELINK(0.5, 3, 0, 1, 0)'DECEL
```

2. MOVELINK 速度参数设置

用户可以通过设置 Link_options 的 BIT4、或者 BIT9、BIT10，实现 MOVELINK 运动速度曲线的优化；



BIT4=1 &&BIT10=0&&BIT11=0（SIN）



BIT4=1 &&BIT10=1&&BIT11=1(5 阶)

3. MOVELINK 不同触发模式

MOVELINK 指令支持不同的触发模式，用户通过设置触发相关参数，提前加载运动指令，带触发调节满足，从轴自动跟随主轴运动。相比较通过编写逻辑程序实现条件触发，及时性更好。

MOVELINK 支持的触发模式有：

- 主轴位置触发：主轴到达某个位置，从轴开始跟随；

样例要求：

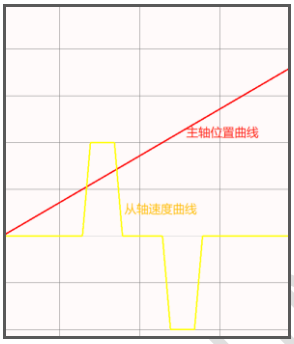
主轴为旋转机构，从轴在主轴位置 100-150 时，与主轴正向同步运动；主轴位置 200-250，与从轴反向同步运动；

实现代码：

```
BASE(0)
REP_DIST = 360
REP_OPTION.0 = 1

BASE(1)
option.1 = 1'设置位置触发
MOVELINK(40, 50, 10, 10, 0, option, 100)
MOVELINK(-40, 50, 10, 10, 0, option, 200)
TRIGGER
```

运行图片：



- 主轴 MARK 触发：MARK 事件发生、MARK_B 事件发生、R_MARK 事件触发。

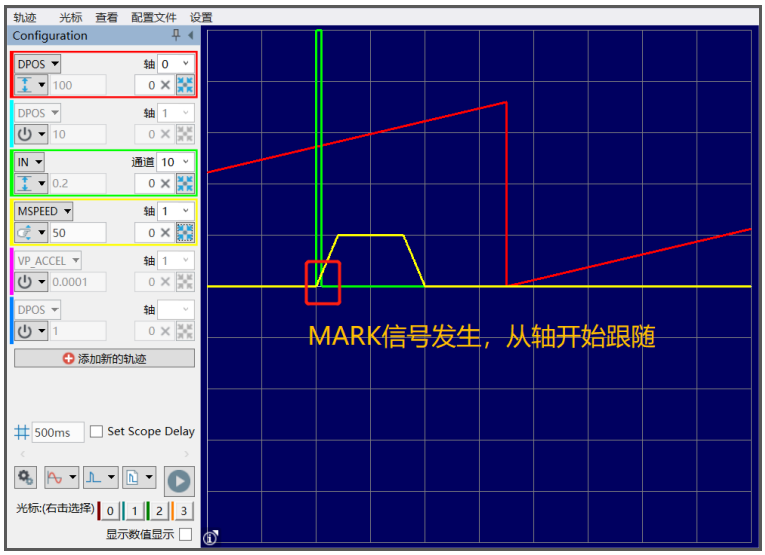
代码：

```
'10 号输入作为 MARK 输入
BASE(0)
REP_DIST = 360
REP_OPTION.0 = 1
REGIST(20, 10, 4, 0, 0)

BASE(1)
option.0 = 1'设置 MARK 点触发
MOVELINK(40, 50, 10, 10, 0, option)
```

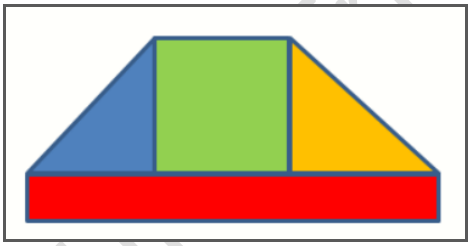
运行效果图：

ESTUN & Trio



MARK 触发 MOVELINK 跟随

4. MOVELINK 基准距离使用



基准位置应用-从轴速度曲线

当需要通过 MOVELINK 指令实现从轴类似于图 的速度曲线时，可使用 MOVELINK 指令的基准参数（base_dist）实现：

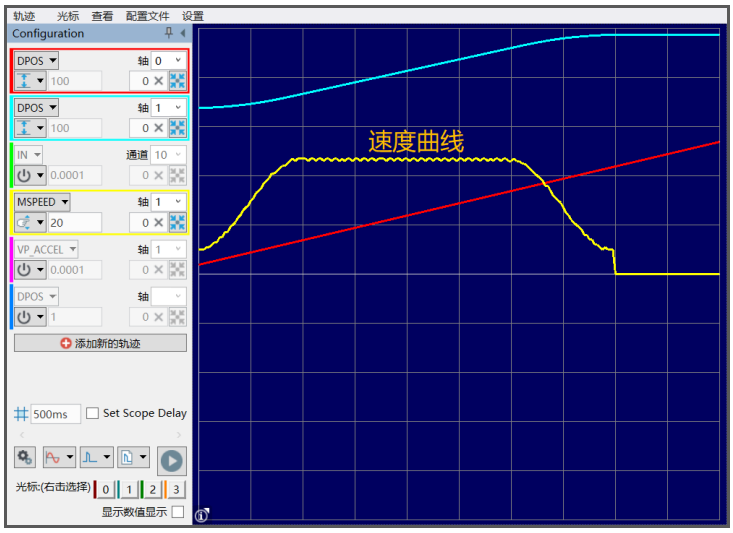
从轴移动量（Distance）= 跟随部分距离+基准参数（base_dist）

主轴移动量 =从轴移动量（Distance）+加速段/2（Link_acc）+减速段/2（Link_Dec）

代码：

```
' Starting at a base ratio of 1, increase the slave speed
' to cause it to advance by 110 mm
' base ratio is 40:40
' total slave distance = 110 + 40
' total link distance on master axis = 110 + ((50+50)/2) + 40
BASE(1)
MOVELINK(150, 200, 50, 50, 0, 16, 0, 40) ' bit set in option to use S curve
TRIGGER
```

运行效果图：



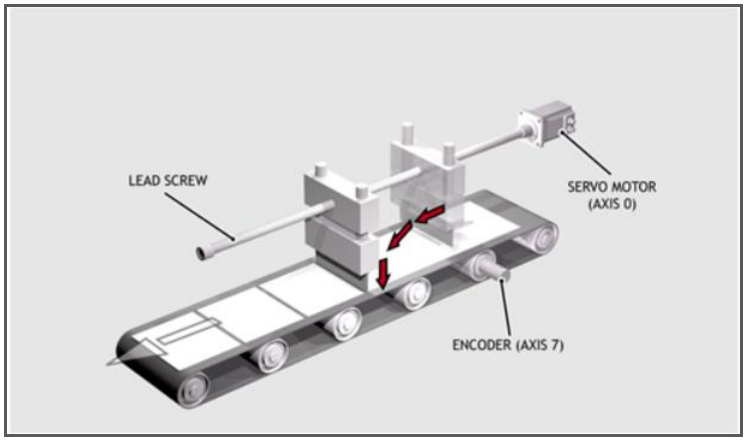
通过基准参数（base_dist）还可以实现更多的速度曲线。



代码实现如下：

```
BASE(1)
MOVELINK(200 , 250 , 100 , 0 , 0 , 16)'蓝思段+绿色段（加速+匀速）
MOVELINK(60 , 90 , 0 , 90 , 0 , 16 , 0 , 30)'橙色+红色（减速，但不减速到 0）
MOVELINK(110 , 360 , 0 , 60 , 0 , 16)'绿色+橙色（第二段匀速+减速）
```

5. 案例--追剪应用



追剪示意图

参考追剪示意图，7 号轴为皮带位置反馈轴，轴 0 为切刀轴，要求物料在皮带上运动时，

动态定长切物料。

具体要求：

- 物料切的长度为 160m；
- 切的过程中，切刀轴的速度应当与皮带运行速度一致；
- 速度同步过程中，控制输出点，进行切割；
- 第一次切长度不做要求；

代码实现：

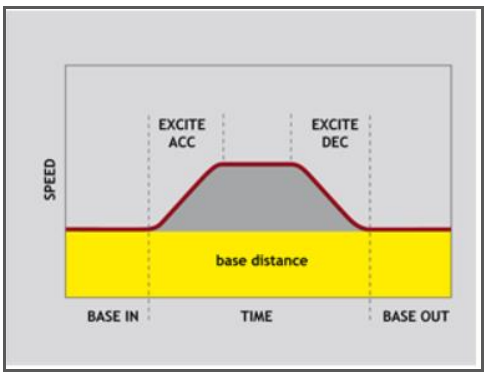
```
DIM master_axis AS INTEGER
master_axis = 7
'运动分解：主轴运动长度为 160 = 150（空跑） +0.6（加速）+1.0（匀速+减速）+8.4（空跑）
'运动分解：从轴运动长度为 000 = 000（空跑） +0.3（加速）+0.7（匀速+减速）+（-1）（返回）
WHILE IN(2) = ON '输入信号 2 控制循环
  BASE(0)
  MOVELINK(0, 150, 0, 0, master_axis) '命令 1：皮带轴空跑 150m，切刀轴不动作
  MOVELINK(0.3, 0.6, 0.6, 0, master_axis) '命令 2：切刀轴加速至与皮带同速
  MOVELINK(0.7, 1.0, 0, 0.6, master_axis) '命令 3：切刀轴与皮带保持同速+减速停止
  WAIT LOADED '表示命令 3 运动加载至 MTYPE，说明命令 2 执行完成，已经开始同速运动
  OP(8, ON) '切刀工作
  MOVELINK(-1.0, 8.4, 0.5, 0.5, master_axis) '命令 4：切刀轴返回运动，从轴返回移动量=前进移动量
  WAIT LOADED '表示命令 4 运动加载至 MTYPE，说明命令 3 执行完成，已经开始返回运动
  OP(8, OFF) '关闭切刀

WEND
```

ESTUN & TRIO

3.18.5.4 特定轨迹凸轮运动 2-FLEXLINK

FLEXLINK 指令，凸轮运动的一种。主-从轴之间的速度曲线由指令参数决定，遵循图 。



FLEXLINK 从轴速度曲线

FLEXLINK 指令相比较于 MOVELINK 指令，可构建更加复杂的速度曲线，也可完成 MOVELINK 指令能够实现的运动效果。

FLEXLINK 指令的常用应用场景有：

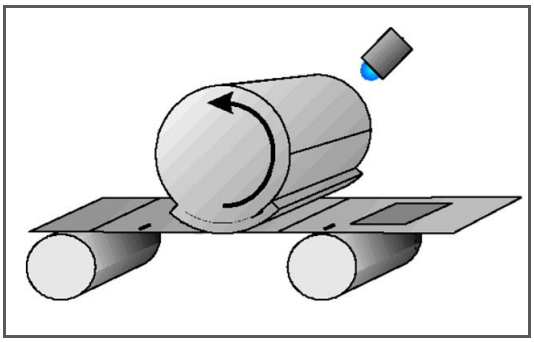
- 飞剪；

3.18.5.4.1 指令格式-FLEXLINK

指令格式	FLEXLINK(base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, link_axis, options, start_pos)		
参数	base_dist	从轴低速部分移动量	
	excite_dist	从轴变速部分移动量	
	link_dist	主轴移动量	
	base_in	从轴低速部分百分比-变速运动前	
	base_out	从轴低速部分百分比-变速运动后	
	Excite_acc	从轴变速部分—加速部分百分比	
	Excite_dec	从轴变速部分—减速部分百分比	
	Link_axis	主轴轴号	
	Options	BIT0	MARK 事件触发后，开始 FLEXLINK 运动
		BIT1	主轴到达设定位置，开始 FLEXLINK 运动
		BIT2	FLEXLINK 运动自动重复，可以通过 REP_OPTION 的 BIT1 置位，关闭重复运动
		BIT8	MARKB 事件触发后，开始 FLEXLINK 运动
		BIT10-12	配置变速部分速度曲线： 0：SIN 速度曲线 1：9 阶速度曲线 2：7 阶速度曲线 3：5 阶速度曲线 4：梯形速度曲线
	start_pos	Options BIT1=1 时生效，主轴触发位置	

ESTUN & Trio

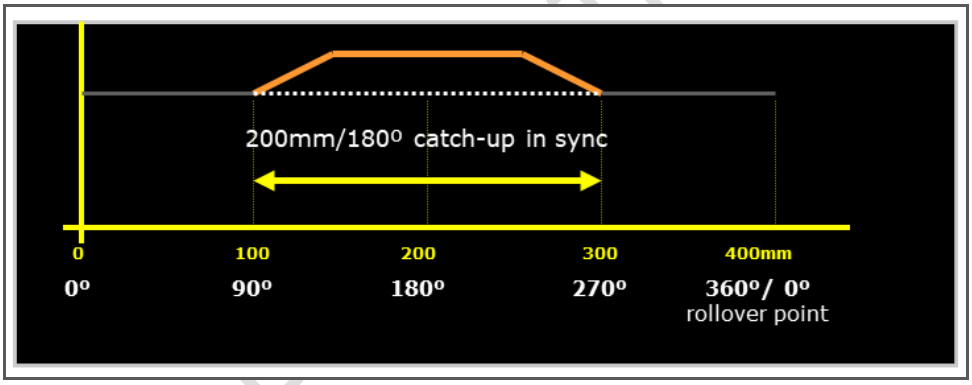
3.18.5.4.2 应用案例



圆筒间隔印刷

圆筒印刷，要求如下：

- 印刷间距可以与滚筒周长不同；
- 运动必须平滑；
- 印刷部分在接触工件部分，运动速度保持一致；



圆筒位置-速度图

参考圆筒-位置速度图，当印刷部分接触圆筒时（90-270），圆筒需要加速至与皮带速度一致，其余部分，应当保持一定低速运行。代码实现：

```
BASE(1)
base_d = 325
link_dist = 325
excite_d = circ - base_d
base_in = 100 * 90 / 360
base_out = 100 * 90 / 360
exacc = 100 * 30 / 360
exdec = 100 * 30 / 360
FLEXLINK(base_d, excite_d, link_dist, base_in, base_out, exacc, exdec, 0)
```

3.19 系统参数

该章节内容仅介绍部分系统参数，其他系统参数请参考 Trio BASIC 帮助文件。

3.19.1IP 配置

参数	说明
IP_ADDRESS	‘ MC_CONFIG script file IP_ADDRESS=192.168.2.100
IP_GATEWAY	‘ MC_CONFIG script file IP_GATEWAY=192.168.0.254
IP_MAC	获取控制器 Ethernet 接口到的 MAC 地址。 Get the MAC address in hexadecimal format >>?hex(ip_mac) 6700000F9 >> Converted to the 6 Octets format this is: 00 06 70 00 00 F9
IP_NET_MASK	‘ MC_CONFIG script file IP_NETMASK=255.255.255.0

3.19.2系统运行状态信息

参数	说明
SYSTEM_ERROR	固件运行过程中发现的错误，错误信息用 SYSTEM_ERROR 的位指示。详见帮助文件中各个位代表的错误信息。
SYSTEM_ERROR_MASK	使能 SYSTEM_ERROR 对应位的错误使 WDOG 改变为 OFF。
SYSTEM_LOAD	获取系统和用户软件运行时的负载率，表示当前伺服周期的负载率，以百分数表示。100-该值表示系统可用的剩余资源。
SYSTEM_LOAD_MAX	该值表示从上电或 SYSTEM_LOAD_MAX 设为 0 开始，系统的最大 SYSTEM_LOAD 。如果 SYSTEM_LOAD_MAX > 100 ，表示当前伺服周期不能处理完应用程序，需要加大 SERVO_PERIOD。
AXIS_ERROR	获取导致轴运动错误的轴号。

3.19.3设备相关信息

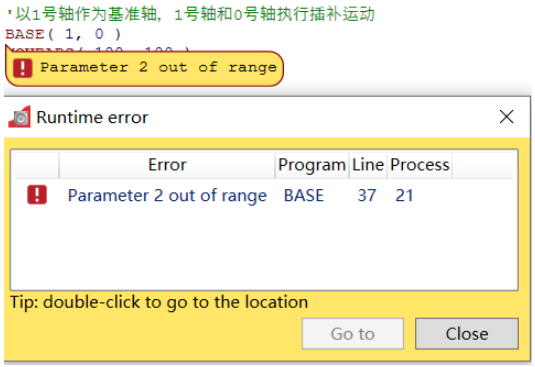
参数	说明
SERIAL_NUMBER	获取唯一的控制器序列号。

参数	说明
	>>PRINT SERIAL_NUMBER 325.0000
VERSION	获取控制器运行的固件版本号。 >>? VERSION 2.0100
FREE	获取剩余可用的程序内存空间字节数。 >>PRINT FREE 47104.0000
TIME	可通过 TIME 读取或设置控制器的实时时钟。 Sets the real time clock in 24 hour format; hh:mm:ss 'Set the real time clock >>TIME = 13:20:00
TIME$	TIME$用于读取实时时钟的时间，以字符串形式输出，格式为 Hour:Minute:Second。 >>print time$ 15:51:06
TSIZE	获取 TABLE 的字节数。 >>?tsize 500000.0000
NOP	获取系统可用的输出端口数量。 >>?NOP 80.0000

EST 川 & Trio

3.19.4COORDINATOR_DATA

<table>
<tr><td>指令格式</td><td>COORDINATOR_DATA(function_number, [param_values])</td></tr>
<tr><td>功能描述</td><td>
<table>
<tr><th>Function_number</th><th>Param_values</th></tr>
<tr><td rowspan="2">16</td><td>0:不允许轴倒序（默认值）</td></tr>
<tr><td>1：允许轴倒序</td></tr>
</table>
</td></tr>
<tr><td>实例</td><td>默认情况下：</td></tr>
</table>

<table>
<tr><td></td><td>


设置该参数功能码 16 等于 1：
<pre>'以1号轴作为基准轴，1号轴和0号轴执行插补运动
COORDINATOR_DATA(16, 1) ' Do not check axis order
BASE(1, 0)
MOVEABS(100, 100)</pre>
</td></tr>
<tr><td>注意</td><td>如果激活了 FRAME 功能，那么轴组不可倒序。</td></tr>
</table>

3.19.5SCHEDULE_TYPE

该参数用于改变多任务的调度方法。该参数在 MC_CONFIG 文件中进行设置。

位 0 用于使能/禁能当一个任务处于睡眠状态时，另外的任务可以使用 CPU 资源运行程序。睡眠状态可以通过暂停程序运行的指令进入该状态，如 WAIT 和 WA 指令。

当伺服周期 SERVO_PERIOD=2000,位 1 被设置为 1 时，每个伺服周期，固件将加倍进行任务调度。在 MC464 产品中，当 SERVO_PERIOD=2000 时，系统进程将每毫秒与通信进程进行一次信息交互。

在 MC464 中，SCHEDULE_TYPE 的值被存储到 FLASH 闪存中。

Bit	Operation		Value
0	0	充分利用 CPU 资源，当任务进入睡眠状态后，CPU 将运行其他任务。例如，任务调用了 WA 指令。该任务直至 WA 的等待时间结束，再进入运行状态。（默认值）	1
	1	即使任务进入睡眠状态，任务仍占用 CPU 的资源。该设置用于 Trio 的老控制器。	
1	0	使用 2000uS 的进程调度（默认）	2
	1	当 SERVO_PERIOD=2000，进程按 1mS 进行调度。在 MC464 中，使能更快速的通信周期（V2.0209 以上）。	
2	0	仅 MC664-X; 使用双核. (default)	4

ESTUN & Trio

	1	仅 MC664-X; 使 MC664-X 启动单核运行模式。通常用于评估 MC664 的性能是否满足要求。	
3	0	TCP/IP 与 SYS 进程共享一个核 (default)	8
	1	TCP/IP 进程运行在第二个核，与运动内核共享。*	
4	0	通信协议栈与 SYS 共享一个核。 (default)	16
	1	通信协议栈运行在第二个核，与运动内核共享资源。*	
5	0	EtherCAT 异步进程与 SYS 共享一个核. (default)	32
	1	EtherCAT 运行在第二个核，与运动内核共享资源。*	
6	0	2 个最高进程号进程为快速任务。 (default)	64
	1	所有的用户进程都为慢速任务，所有用户程序具有相同优先级。	

*（适用于多核处理器，如 MC6NE, Flex-nano）

3.19.6IP_PROTOCOL_CTRL

该参数用于禁能一个或多个通信以太网通信端口。位为 0 时表示端口使能，1 时端口禁能。该参数可以在控制器上电时暂时禁能端口，然后在程序中使能所需端口。默认情况下所有的端口都是使能的。

位	说明	值
0	Motion Perfect (Telnet)	n/a
1	PCMotion	n/a
2	Modbus	4
3	EthernetIP	8
4	IEC61131-3 programming	n/a
5	Uniplay	n/a
6	Transparent protocol text file loader	n/a
7	Reserved bit	n/a
8	Reserved bit	n/a
9	MC4xx protocol text file loader	n/a

禁能 Ethernet IP 端口，并在 BASIC Startup 程序中，设置数据通信接口后再使能 Ethernet IP。

a) MC_CONFIG 文件中设置:

IP_PROTOCOL_CTRL = 8

ESTUN & Trio

b) Startup BASIC 程序中设置:

```
'--Config *PLC INPUT* Instance (100), data to PLC from Trio.
ETHERNET(1, -1, 14, 1, 0, 200) '200 = set VR starting address
ETHERNET(1, -1, 14, 1, 1, 3) '3 = use VR for data location
ETHERNET(1, -1, 14, 1, 2, 1) '1 = use 32 bit integer data
ETHERNET(1, -1, 14, 1, 3, 120) '120 = number of data values

'--Config *PLC OUTPUT* Instance (101), data from PLC to Trio.
ETHERNET(1, -1, 14, 2, 0, 400) '400 = set VR starting address
ETHERNET(1, -1, 14, 2, 1, 3) '3 = use VR for data location
ETHERNET(1, -1, 14, 2, 2, 1) '1 = use 32 bit integer data
ETHERNET(1, -1, 14, 2, 3, 120) '120 = number of data values
'------------------------------------------------------------
IP_PROTOCOL_CTRL = 0 'enable Ethernet IP
```

ESTUN & Trio

4 通信接口

TRIO 控制器支持主流的通讯协议包含：MODBUS TCP、TCP/IP、UDP、ETHERNETIP、ETHERCAT。

MODBUS TCP 主要用于与 HMI 之间的数据传输；TCP/IP 主要用于与视觉、PC 端软件的数据传输；UDP 主要用于与第三方传感器之间的数据传输；ETHERNETIP 主要用于与其他品牌 PLC 之间的数据传输；ETHERCAT 通讯为实时通讯，主要用于与 ETHERCAT 总线型伺服之间的数据传输。

4.1 Modbus TCP

控制器所有型号均支持 MODBUS TCP 通讯协议；

	客户端	服务器
是否支持	是	是
控制器	所有型号控制器	所有型号控制器

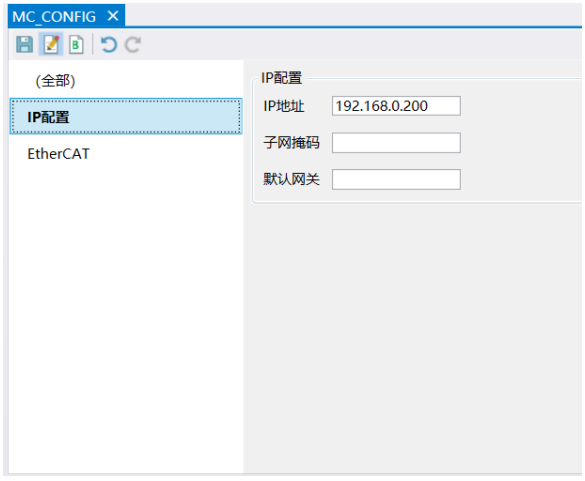
4.1.1 应用场合

1. HMI 作为 MODBUS TCP 客户端，访问控制器内部信息；
2. 被其他控制器访问；

4.1.1.1 16 位整型通讯

4.1.1.1.1 控制器设置

1. MC_CONFIG 文件中设置控制器 IP 地址；服务器和客户端的 IP 地址在同一网段；比如：控制器 IP 为 192.168.0.200，HMI 的 IP 地址为 192.168.0.100.



2. 控制器用于 MODBUS TCP 的端口号为 502；

ESTUN & Trio

3. 默认情况下，地址对应关系如下表：

TRIO 内部地址	MODBUS 地址区域	对应关系	实例
IN	1 区	IN(X)-1x(X+1)	IN(1)---1x2; IN(10)-1x11；
OP	0 区	OP(X)-0x(X+1)	OP(1)---0x2; OP(10)-0x11；
VR	4 区	VR(X)-4x(X+1) 32 位数据占用两个 VR	VR(1)---4x2; VR(10)-4x12；

4. 默认情况下 MODBUS TCP 的通讯格式为 16 位整数通讯，如需添加代码，如下：

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 3 ) '4 区关联 TRIO 地址为 VR
ETHERNET( 1, -1, 14, 0, 2, 0 ) '设置为 16 位通讯
ETHERNET( 1, -1, 14, 0, 6, 0 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

5. 可通过 ETHERNET 指令修改与 MODBUS 4 区关联的 TRIO 内部地址，可全部修改为 TABLE 区；

```
'4 区所有关联地址均为 TABLE

IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 4 ) '4 区关联 TRIO 地址为 TABLE
ETHERNET( 1, -1, 14, 0, 2, 0 ) '设置为 16 位通讯
ETHERNET( 1, -1, 14, 0, 6, 0 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

6. 参考自定义通讯格式，可通过 ETHERNET 指令修改与 MODBUS 4 区关联的 TRIO 内部地址，可部分修改为 TABLE 区；

ESTUN & Trio

4.1.1.1.2 触摸屏设置

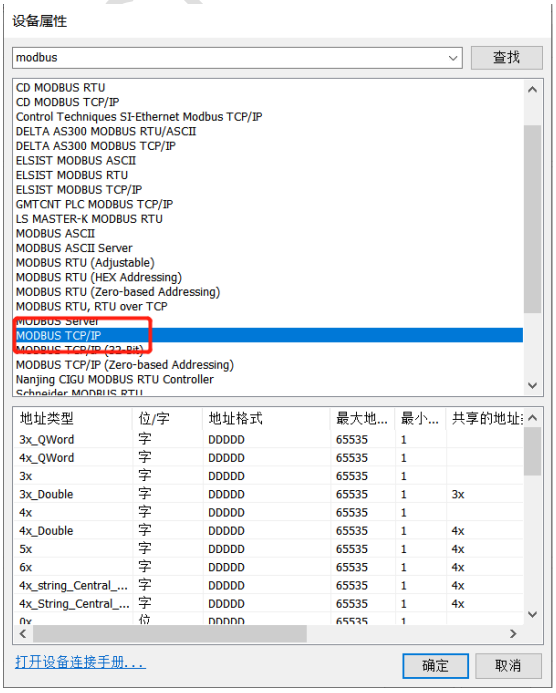
1. 触摸屏 IP 地址设置，触摸屏 IP 地址应当与控制器 IP 地址在相同网段；详见触摸屏操作文档；

2. 触摸屏设备设置，以威纶通触摸屏为例；

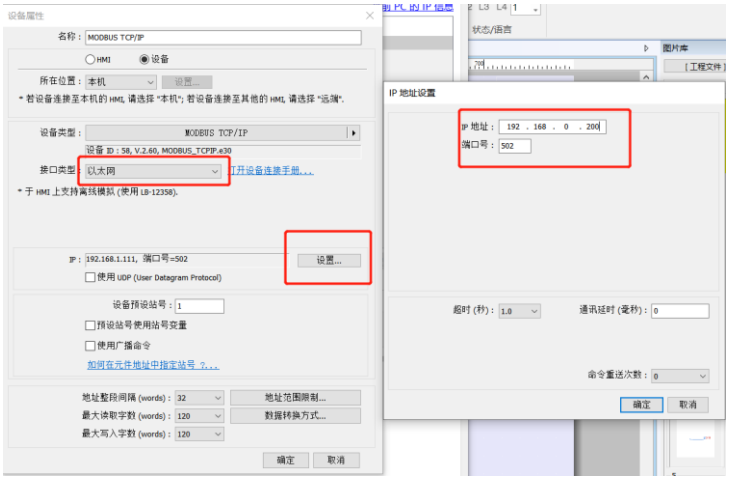
   1) 新建设备/服务器；



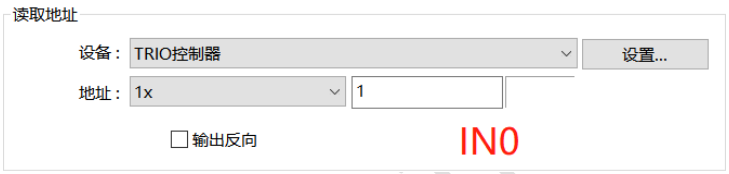
   2) 设备类型选择为 MODBUS TCP/IP



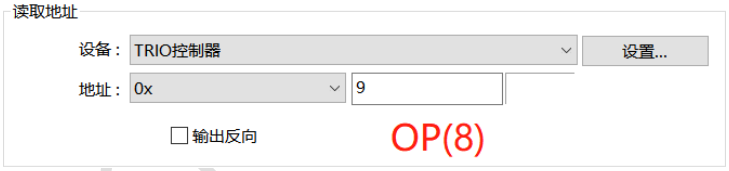
   3) 设置控制器 IP 地址及端口号，与控制器设置中相匹配；



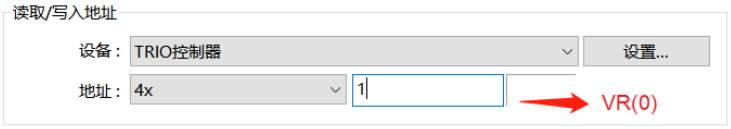
4) 编辑画面，建立连接；



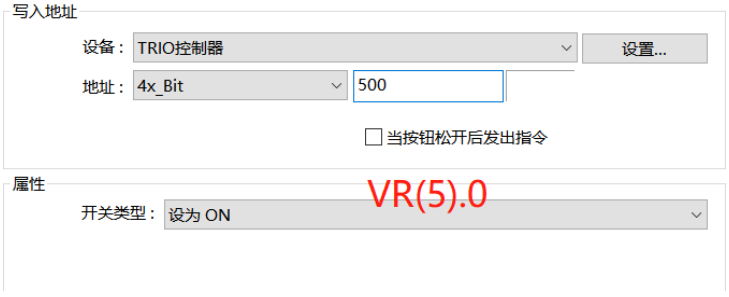
IN 关联



OP 关联



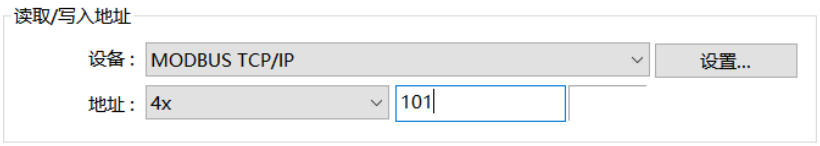
VR 关联



VR_BIT 关联

4.1.1.1.3 16 位通讯模式下传输 32 位整型数据

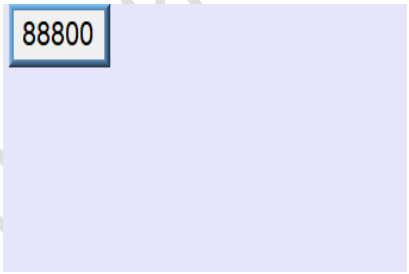
1. HMI 上设置数值输入控件数据类型选择为 32 位整型，地址为 4x101;





2. 运行效果





3. 结果分析

触摸屏上设置的 32 位整型数据，地址是 4X101，当数据的表达范围超出 16 位表示时，数据的高 16 位存储至 4X102，低 16 位存储至 4X101，所以：

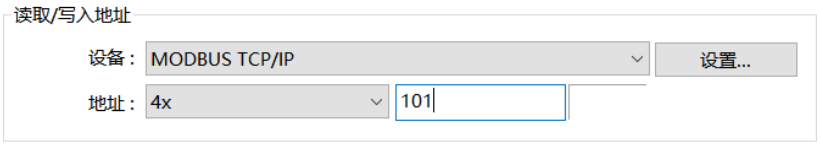
VR(101) = INT(88800/65536)=1

VR(100) = 88800 MOD 65536 = 23264 '取模运算

4. 在 16 位整型通讯的模式下，HMI 控件选择 32 位数据通讯，将出现占用两个 VR 变量情况；

4.1.1.1.4 16 位通讯模式下传输 32 位浮点型数据

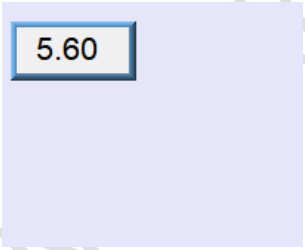
1. HMI 上设置数值输入控件数据类型选择为 32 位浮点，地址为 4x101;





2. 运行效果





3. 结果分析

触摸屏上设置的 32 位浮点数，地址是 4X101，5.6 将按照 32 位浮点数的格式转换成 2 进制数存储至 4X101（低 16 位）和 4X102（高 16 位）；



所以：

VR(100) =4X101=$3333=13107

VR(101) =4X012=$40B3=16563

4. 在 16 位整型通讯的模式下，HMI 控件选择 32 位浮点数通讯，将出现占用两个 VR 变量情况；

5. 可使用 IEEE_IN 指令将 16 进制数转换成浮点数。

```
DIM byte AS INTEGER( 4 )
```

```
byte( 0 ) = ( VR( 101 ) AND $0000ff00 ) >> 8 '取高 8 位
byte( 1 ) = ( VR( 101 ) AND $000000ff ) '取低 8 位
byte( 2 ) = ( VR( 100 ) AND $0000ff00 ) >> 8 '取高 8 位
byte( 3 ) = ( VR( 100 ) AND $000000ff ) '取低 8 位

result = IEEE_IN( byte( 0 ), byte( 1 ), byte( 2 ), byte( 3 ) )
```

4.1.1.2 32 位整型通讯

4.1.1.2.1 控制器设置

1. 设置控制器 IP 地址，同 16 位通讯；
2. 端口号为 502；
3. 如果需要设置全部 VR 对应 MODBUS 4 区，代码如下；

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 3 ) '4 区关联 TRIO 地址为 VR
ETHERNET( 1, -1, 14, 0, 2, 1 ) '设置为 32 位整型通讯
ETHERNET( 1, -1, 14, 0, 6, 1 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

地址对应关系		
TRIO 内部寄存器	MODBUS 地址区域	对应关系
IN	1 区	IN(X)-1x(X+1)
OP	0 区	OP(X)-0x(X+1)
VR	4 区	VR(X)-4x(2*X+1)

4. 如果需要设置全部 TABLE 对应 MODBUS 4 区，代码如下；

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 4) '4 区关联 TRIO 地址为 TABLE
ETHERNET( 1, -1, 14, 0, 2, 1 ) '设置为 32 位整型通讯
ETHERNET( 1, -1, 14, 0, 6, 1 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

地址对应关系		
TRIO 内部寄存器	MODBUS 地址区域	对应关系
IN	1 区	IN(X)-1x(X+1)
OP	0 区	OP(X)-0x(X+1)

ESTUN & Trio

TABLE	4 区	TABLE(X)-4x(2*X+1)

5. 如果需要设置部分 TABLE 对应 MODBUS 4 区，部分 VR 对应 MODBUS4 区，参考《自定义通讯格式》；

4.1.1.2.2 触摸屏设置

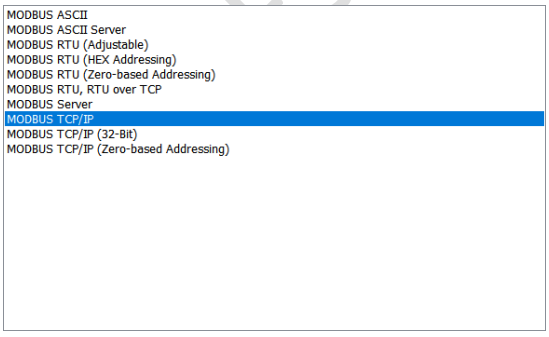
1) 触摸屏 IP 地址设置，触摸屏 IP 地址应当与控制器 IP 地址在相同网段；

2) 触摸屏设备设置，以威纶通触摸屏为例；

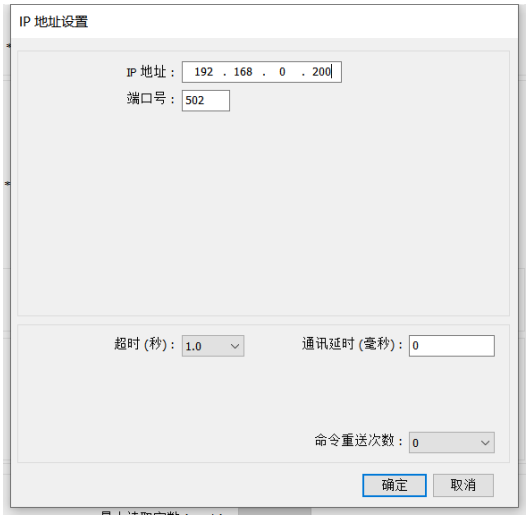
1) 新建设备/服务器；



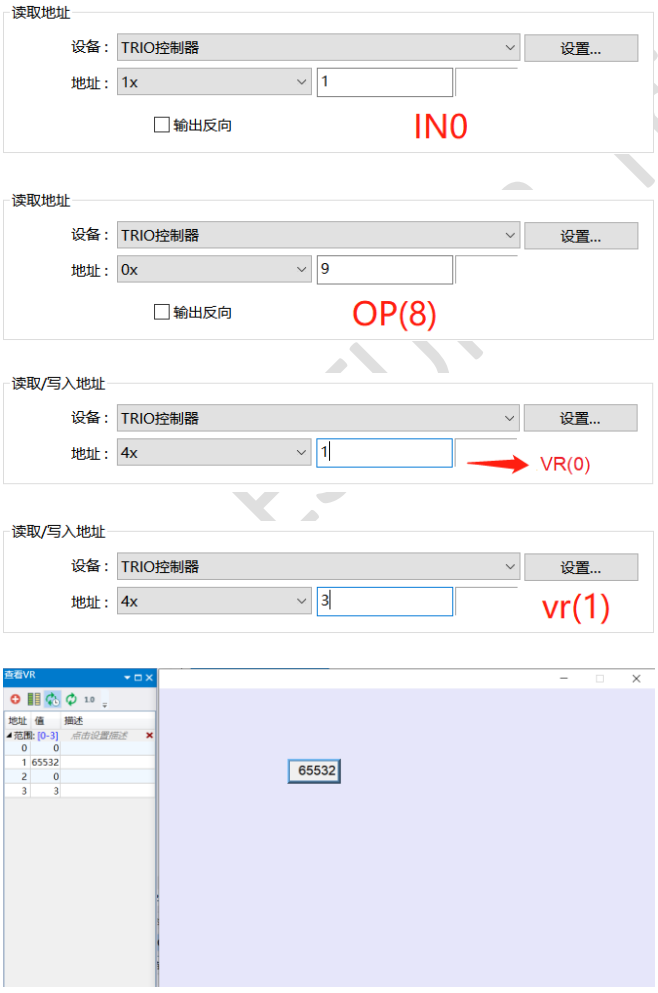
2) 设备类型选择为 MODBUS TCP/IP



3) 设置控制器 IP 地址及端口号；


IP 地址设置
IP 地址：
192 . 168 . 0 . 200
端口号：
502
超时 (秒)：
1.0
通讯延时 (毫秒)：
0
命令重送次数：
0
确定
取消

4) 编辑画面，建立连接；


读取地址
设备：
TRIO控制器
设置...
地址：
1x
1
输出反向
IN0
读取地址
设备：
TRIO控制器
设置...
地址：
0x
9
输出反向
OP(8)
读取/写入地址
设备：
TRIO控制器
设置...
地址：
4x
1
VR(0)
读取/写入地址
设备：
TRIO控制器
设置...
地址：
4x
3
vr(1)
查看VR
地址
值
描述
范围: [0-3]
点击设置描述
0 0
1 65532
2 0
3 3
65532

4.1.1.3 32 位浮点通讯

4.1.1.3.1 控制器设置

1. 设置控制器 IP 地址，同 16 位通讯；
2. 端口号为 502；
3. 如果需要设置全部 VR 对应 MODBUS 4 区，代码如下；

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 3 ) '4 区关联 TRIO 地址为 VR
ETHERNET( 1, -1, 14, 0, 2, 2 ) '设置为 32 位浮点通讯
ETHERNET( 1, -1, 14, 0, 6, 1 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

地址对应关系		
TRIO 内部寄存器	MODBUS 地址区域	对应关系
IN	1 区	IN(X)-1x(X+1)
OP	0 区	OP(X)-0x(X+1)
VR	4 区	VR(X)-4x(2*X+1)

4. 如果需要设置全部 TABLE 对应 MODBUS 4 区，代码如下；

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 4) '4 区关联 TRIO 地址为 TABLE
ETHERNET( 1, -1, 14, 0, 2, 2 ) '设置为 32 位整型通讯
ETHERNET( 1, -1, 14, 0, 6, 1 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

地址对应关系		
TRIO 内部寄存器	MODBUS 地址区域	对应关系
IN	1 区	IN(X)-1x(X+1)
OP	0 区	OP(X)-0x(X+1)
TABLE	4 区	TABLE(X)-4x(2*X+1)

5. 如果需要设置部分 TABLE 对应 MODBUS 4 区，部分 VR 对应 MODBUS4 区，参考《自定义通讯格式》；

ESTUN & Trio

4.1.1.3.2 触摸屏设置

同 32 位整型触摸屏设置

4.1.1.4 自定义通讯格式

4.1.1.4.1 控制器设置

1. 设置控制器 IP 地址，同 16 位通讯；
2. 端口号为 502；
3. 使用 ETHERNET 指令功能码 14，配置 4 区地址关联的变量类型（VR/TABLE）和数据类型（16 位整数、32 位整型、32 位浮点）；参考 ETHERNET 指令；

4.1.1.4.2 触摸屏设置

同 32 位整型触摸屏设置

4.1.1.4.3 实例

1. 需求如下表：

MODBUS 地址	TRIO 内部地址	通讯数据类型
4x1-4x100	VR(0)-VR(99)	16 位整型
4x101-4x299	VR(100)-VR(199)	32 位整型
4x301-4x499	VR(200)-VR(299)	32 位浮点
4x501-4x699	TABLE(0)-TABLE(99)	32 位浮点

2. TRIO 设置代码如下；

```
IP_PROTOCOL_CTRL = 12
ETHERNET( 1, -1, 14, 0, 7, -1 )' cancel all previous maps
ETHERNET( 1, -1, 14, 0, 7, 0, 0, 100, 0, 0 ) ' VRs 0 to 99 size = Int 16
ETHERNET( 1, -1, 14, 0, 7, 100, 100, 200, 0, 1 ) ' VRs 100 to 199 size = Int 32
ETHERNET( 1, -1, 14, 0, 7, 300, 200, 200, 0, 2 ) ' VRs 200 to 299 size = FP 32
ETHERNET( 1, -1, 14, 0, 7, 500, 0, 200, 1, 2 ) ' TABLE 0 to 99 size = FP 32
ETHERNET( 0, -1, 14, 0, 7 ) ' display the map to terminal #0
IP_PROTOCOL_CTRL = 0
```

3. HMI 设备类型选择为:MODBUS TCP/IP；
4. 根据关联地址绑定 HMI 控件；

4.1.1.5 TRIO MODBUS

仅部分 HMI 产品支持。

ESTUN & Trio

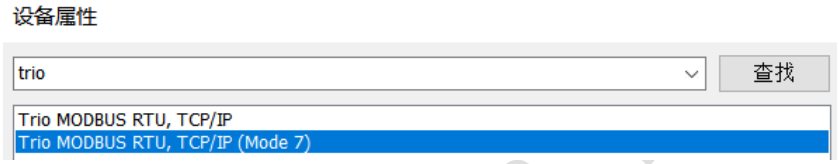
4.1.1.5.1 控制器设置

设置同 32 位浮点数设置，代码如下：

```
IP_PROTOCOL_CTRL = 12 '打开配置权限
ETHERNET(1,-1,10,502)
' Modbus TCP
ETHERNET( 1, -1, 14, 0, 1, 3 ) '4 区关联 TRIO 地址为 VR
ETHERNET( 1, -1, 14, 0, 2, 2 ) '设置为 32 位浮点通讯
ETHERNET( 1, -1, 14, 0, 6, 1 )
IP_PROTOCOL_CTRL = 0 '关闭配置权限
```

4.1.1.5.2 触摸屏设置

1. 设备类型选择为 TRIO MODBUS (MODE7)，其他设置同 32 位浮点设置；



2. 支持 VR 或者 TABLE 通讯，不支持 DI/DO 的传输；

地址类型	位/字	地址格式	最大地...	最小...	共享的地址类型
VR	字	DDDD	4096	0	
VR_INT	字	DDDD	4096	0	
Table	字	DDDDD	31999	0	
Table_INT	字	DDDDD	31999	0	
VR_Bit	位	DDDDdd	409631	0	
VR_INT_Bit	位	DDDDdd	409631	0	
Table_Bit	位	DDDDDdd	3199931	0	
Table_INT_Bit	位	DDDDDdd	3199931	0	

3. 同时支持 VR 整型和浮点型传输；
4. 整型数据地址类型为 VR_INT、TABLE_INT;
5. 浮点数据地址类型为 VR、TABLE；
6. BIT 位数据类型为 VR_INT_BIT；
7. DI/DO 的监控可通过控制器编写代码将 DI/DO 状态映射至 VR；

4.1.1.6 不同方式对比

	16 位整型	32 位整型	32 位浮点	自定义	TRIO MODBUS
DI	1 区	1 区	1 区	1 区	映射至 VR 实

					现
DO	1 区	1 区	1 区	1 区	映射至 VR 实现
VR	可设置为 4 区	可设置为 4 区	可设置为 4 区	可设置为 4 区	可设置为 4 区
TABLE	可设置为 4 区	可设置为 4 区	可设置为 4 区	可设置为 4 区	可设置为 4 区
4 区地址对应关系	4x（A+1）—VR(A)或者 4x（A+1）—TABLE(A)	4x（2*A+1）—VR(A)或者 4x（2*A+1）—TABLE(A)	4x（2*A+1）—VR(A)或者 4x（2*A+1）—TABLE(A)	用户自定义	VR_INT（X）对应整型 VR（X）； VR（X）对应浮点 VR（X）；
BIT 位	支持；0-15 位	支持；0-31 位	不支持	用户自定义，设置为 16 位整型或者 32 位整型区域可传输 BIT 位	支持； VR_INT_BT
32 位整数	不支持	支持	不支持	支持，需自定义	支持； VR_INT
32 位浮点	不支持	不支持	支持	支持，需自定义	支持； VR

ESTUN & Trio

4.1.1.7 相关指令

4.1.1.7.1 IP_PROTOCOL_CONFIG

功能描述	此参数允许用户禁用以太网上的一个或多个通信端口的操作。如果位为 0，则启用端口。如果位是 1，则端口被禁用，当客户端尝试打开时，端口将不响应。它可以用来暂时禁用端口。一个基本的程序配置的协议流程：上电后，在程序配置好以后，再启用端口。默认情况下，所有端口都已启用。

位	描述	值
0	Motion Perfect (Telnet)	n/a
1	PCMotion	n/a
2	Modbus	4
3	EthernetIP	8

	4	IEC61131-3 programming	n/a
	5	Uniplay	n/a
	6	Transparent protocol text file loader	n/a
	7	Reserved bit	n/a
	8	Reserved bit	n/a
	9	MC4xx protocol text file loader	n/a
	如果需要改变 Modbus TCP 传输数据类型，我们需要在 MC_CONFIG 文件中通过设置该参数的 bit2、bit3 为 1，在上电时刻关闭该端口的 MODBUS 功能；然后在 Startup 程序中完成设定工作及重新打开端口工作，通过上述所说的 ETHERNET 指令完成设定后，将 IP_PROTOCOL_CTRL 赋值为 0，重新打开 ModBus 功能。		

4.1.1.7.2 ETHERNET

表达形式：ETHERNET(rw, slot, function [,parameters…])		
rw（读写）	指定所需的操作。	
	0	读
	1	写
slot（槽）	-1 本体以太网口	
Function（功能）	功能	
	0	IP 地址
	1	保留
	2	子网掩码
	3	MAC 地址
	4	默认端口号
	5	令牌端口号
	6	PRP 固件版本(只读)
	7	Modbus TCP 模式
	8	默认网关
	9	数据配置
	10	Modbus TCP 端口号
	11	ARP 高速缓冲存储器
	12	保留
	13	保留

ESTUN & Trio

	14	为 Modbus TCP 或 ETHERNET IP 配置端口
	15	设置每个栈调用处理的最大数据包

用户可通过功能码 7、9、14 对端口进行配置，Startup 中根据设置生成的代码采用功能码 14 实现。

表达形式：ETHERNET(1, slot, 14, endpoint_id, parameter_index, parameter_value)		
endpoint_id（端口 id）	0	Modbus TCP 服务器（从站）
	1	Ethernet IP 对象，例 100 (输入)
	2	Ethernet IP 对象，例 101 (输出)
	3	Ethernet IP 对象，例 102 (输入)
	4	Ethernet IP 对象，例 103 (输出)
	6	Modbus TCP 客户端（主站）
parameter_index（参数索引）	0	地址
	1	数据位置
	2	数据格式
	3	长度
	4	类别
	5	Instance
	6	操作模式
	7	设置/读取内存映射
parameter_value（值）	见下表	

parameter_index	parameter_value	
0	数据位置的起始地址。	
1	数据在控制器上的位置	
	0	寄存器 (保留)
	1	IO 输入
	2	IO 输出
	3	VR (默认)

ESTUN & Trio

	4	Table
	5	数字 IO 输入
	6	数字 IO 输出
	7	模拟 IO 输入
	8	模拟 IO 输出
2	0	16 位整型 (默认)
	1	32 位整型
	2	32 位浮点型
	3	64 位浮点型
3	返回的数据位置的数目	
6	0	Modbus TCP 使用正常寻址
	1	Modbus TCP 使用“地址减半”寻址

4.1.2 客户端

4.1.2.1 应用场合

访问 MODBUS TCP SERVER，使用 MODBUS 指令实现连接的建立、断开、数据读写工作。

4.1.2.2 连接建立-MODBUS 功能码 0

指令格式	value = MODBUS(0,slot , ip address/network type 1...4 [,port number/connection type [,vr_index [,unit identifier [,CmdTimeout [,CharTimeout]]]]])

ESTUN & Trio

<table>
<tr><td>功能描述</td><td>建立与 MODBUS TCP SERVER 的连接
<table>
<tr><td>value</td><td colspan="2">TRUE：连接建立成功；</td></tr>
<tr><td></td><td colspan="2">FALSE：连接建立失败；</td></tr>
<tr><td>slot</td><td colspan="2">槽号，一般值为-1；</td></tr>
<tr><td rowspan="4">Ip address/network type</td><td>MODBUS TCP</td><td>服务器 IP 地址</td></tr>
<tr><td>RS232</td><td>-1</td></tr>
<tr><td>两线制 RS485</td><td>-2</td></tr>
<tr><td>四线制 RS485</td><td>-3</td></tr>
<tr><td rowspan="3">port number/connection type</td><td>MODBUS TCP</td><td>502</td></tr>
<tr><td>MODBUS RTU(CLINT)</td><td>0</td></tr>
<tr><td>MODBUS RTU(SERVER)</td><td>1</td></tr>
<tr><td>vr_index</td><td colspan="2">存储通讯 HANDLE 的 VR 地址</td></tr>
</table></td></tr>
<tr><td>连接实例</td><td>已知服务器 IP 地址为：192.168.0.185，建立的通讯 handle 存储在 VR(20)，实现代码如下：
<pre>'IP Address 192.168.0.185, Port Number 502
IF MODBUS(0, -1, 192, 168, 0, 185, 502, 20) = TRUE THEN
  PRINT "Modbus port opened OK"
    modbus_handle = VR(20)
  ELSE
  PRINT "Error, Modbus server not found"
ENDIF</pre></td></tr>
<tr><td>使用注意</td><td>控制器 IP 地址与服务器 IP 地址应当在相同网段；</td></tr>
</table>

4.1.2.3 连接断开-MODBUS 功能码 1

<table>
<tr><td>指令格式</td><td>value = MODBUS(1,slot,handle)</td></tr>
<tr><td>功能描述</td><td>断开与服务器之间的连接；
<table>
<tr><td>value</td><td>TRUE：指令执行成功；</td></tr>
<tr><td></td><td>FALSE：指令执行失败；</td></tr>
<tr><td>slot</td><td>槽号，一般值为-1；</td></tr>
<tr><td>handle</td><td>使用功能码 0 建立连接时存储在 VR 中的数值</td></tr>
</table></td></tr>
<tr><td>连接实例</td><td>参考连接建立实例，handle 存储至 VR(20);
<pre>MODBUS(1, -1, VR(20))</pre></td></tr>
</table>

ESTUN & Trio

4.1.2.4 连接状态读取-MODBUS 功能码 2

<table>
<tr><td>指令格式</td><td colspan="3">MODBUS(2, slot, handle [,VR index])</td></tr>
<tr><td>功能描述</td><td colspan="3">断开与服务器之间的连接；
<table>
<tr><td>value</td><td colspan="2">TRUE：指令执行成功；</td></tr>
<tr><td></td><td colspan="2">FALSE：指令执行失败；</td></tr>
<tr><td>slot</td><td colspan="2">槽号，一般值为-1；</td></tr>
<tr><td>handle</td><td colspan="2">使用功能码 0 建立连接时存储在 VR 中的数值</td></tr>
<tr><td>Vr index</td><td colspan="2">存储连接状态的 VR 地址，存储值为 0，表示关闭，
存储值为 1，表示打开；</td></tr>
</table></td></tr>
<tr><td>连接实例</td><td colspan="3">参考连接建立实例，handle 存储至 VR(20);
<pre>
MODBUS(2, -1, VR(20), 200)

IF VR(200) = 1 THEN
    PRINT "Modbus port is open"
  ELSE
    PRINT "Modbus port is closed"
  ENDIF
</pre></td></tr>
</table>

4.1.2.5 数据读写- MODBUS 功能码 3

<table>
<tr><td>指令格式</td><td colspan="3">value = MODBUS(3, slot, handle, modbus function code [, parameters])</td></tr>
<tr><td>功能描述</td><td colspan="3">对服务器的数据进行读写操作；
<table>
<tr><td>value</td><td colspan="2">TRUE：指令执行成功；</td></tr>
<tr><td></td><td colspan="2">FALSE：指令执行失败；</td></tr>
<tr><td>slot</td><td colspan="2">槽号，一般值为-1；</td></tr>
<tr><td>handle</td><td colspan="2">使用功能码 0 建立连接时存储在 VR 中的数值</td></tr>
<tr><td rowspan="8">modbus function code</td><td>1</td><td>读多个线圈（0 区）</td></tr>
<tr><td>2</td><td>读多个离散输入（1 区）</td></tr>
<tr><td>3</td><td>读多个寄存器</td></tr>
<tr><td>5</td><td>写单个线圈</td></tr>
<tr><td>6</td><td>写单个寄存器</td></tr>
<tr><td>15</td><td>写多个线圈</td></tr>
<tr><td>16</td><td>写多个寄存器</td></tr>
<tr><td>parameters</td><td colspan="2">根据 modbus function code 进行设置</td></tr>
</table></td></tr>
</table>

Modbus 当前其他功能码，请查阅 Modbus 相关的协议文档。

ESTUN & Trio

实例 1

```
my_slot = -1

open_modbus = $ 00
close_modbus = $ 01
get_status = $ 02
ex_modbus_func = $ 03
get_error_log = $ 10

' check if Modbus is already open
MODBUS( get_status, my_slot, 100 )
IF VR( 100 ) = 1 THEN
   ' close the connection so that it can be re-opened
   MODBUS( close_modbus, my_slot )
ENDIF

' open the modbus server (remote slave) & put handle in VR(20)
MODBUS( open_modbus, my_slot, 192 , 168 , 000 , 249, 502, 20 )

REPEAT
   ' get 10 values from holding registers 1000 to 1009
   MODBUS( ex_modbus_func, my_slot, VR( 20 ), 3 , 1000 , 10 )
   ' send 10 values to holding registers 1010 to 1019
   MODBUS( ex_modbus_func, my_slot, VR( 20 ), 16 , 1010 , 10 )
   WA( 200 )
UNTIL FALSE
```

ESTUN & Trio

实例 2

```
' Modbus registers are 16 bit. We are sending 32 bit values
' therefore 2 Modbus registers are used for each data value
ETHERNET( 1 , -1 , 14 , 6 , 1 , 3 ) 'VR
ETHERNET( 1 , -1 , 14 , 6 , 2 , 1 ) 'integer 32 bit on master

my_slot = -1
open_modbus = $00
close_modbus = $01
get_status = $02
ex_modbus_func = $03
get_error_log = $10
vr_handle = 100
vr_status = 101
```

```
default_handle = 0
' check if Modbus is already open
MODBUS( get_status, my_slot, default_handle, vr_status )
IF VR( vr_status ) = 1 THEN
   ' close the connection so that it can be re-opened
   MODBUS( close_modbus, my_slot, default_handle )
ENDIF
' open the modbus server (remote slave)
IF NOT MODBUS( open_modbus, my_slot, 192 , 168 , 008 , 250 , 502 , vr_handle ) THEN
   PRINT "Err: failed to open modbusTCP connection"
   STOP
ELSE
   PRINT "Successfully opened connection, handle : " , VR( vr_handle ) [ 0 ]
ENDIF

REPEAT
   ' get 2 values from holding registers 2048 to 2049 copy to VR(200).. VR(201)
   IF MODBUS( ex_modbus_func, my_slot, VR( vr_handle ), 3 , 4096 , 4 , 200 ) = FALSE THEN
      PRINT # 5 , "Error reading holding registers 2048..2049"
   ENDIF
      ' send 30 values to holding registers 500 to 529, get from VR(1200)...
   IF MODBUS( ex_modbus_func, my_slot, VR( vr_handle ), 16 , 1000 , 60 , 1200 ) = FALSE
THEN
      PRINT # 5 , "Error writing holding registers 500.."    ENDIF
   WA( 20 )
UNTIL FALSE
```

ESTUN & TIP

4.1.3 数据转换

4.1.3.1 IEEE_IN

指令格式	value = IEEE_IN(byte0,byte1,byte2,byte3)
功能描述	返回由 4 个字节表示的浮点数； **Byte0** 31-24 位 **Byte1** 23-16 位 **Byte2** 15-8 位 **Byte3** 7-0 位
实例	已知通过 MODBUS 指令从服务器获取浮点数值，数值的低 16 位存储在 VR(100),数值的高 16 位存储在 VR(101)，通过 IEEE_IN 指令可以获取该浮点数大小；

	DIM byte AS INTEGER(4) byte(0) = (VR(101) AND $0000ff00) >> 8 '取高 8 位 byte(1) = (VR(101) AND $000000ff) '取低 8 位 byte(2) = (VR(100) AND $0000ff00) >> 8 '取高 8 位 byte(3) = (VR(100) AND $000000ff) '取低 8 位 result = IEEE_IN(byte(0), byte(1), byte(2), byte(3))

4.1.3.2 IEEE_OUT

指令格式	byte_n = IEEE_OUT(value, n)
功能描述	获取浮点数的指定字节；
实例	需要将浮点数 A 的值使用 MODBUS 指令告知 MODBUS 服务器，VR(101)存储高 16 位，VR(100)中存储低 16 位； DIM byte AS INTEGER(4) byte(0) = IEEE_OUT(a, 0) byte(1) = IEEE_OUT(a, 1) byte(2) = IEEE_OUT(a, 2) byte(3) = IEEE_OUT(a, 3) VR(101) = byte(0) << 8 + byte(1) VR(100) = byte(2) << 8 + byte(3)

4.2 TCP/IP

4.2.1 客户端

4.2.1.1 描述

1. 控制器支持 TCP/IP 通讯协议，作为客户端访问服务器；
2. 最大支持同时与 10 个 TCP/IP 服务器通讯；
3. 通道 20-29 可用于建立 TCP/IP 客户端访问服务器；

4.2.1.2 连接建立

指令格式	value = OPEN # channel AS "[location:]name" FOR access

ESTUN & Trio

<table>
<tr><td>功能描述</td><td>建立与服务器的连接；
<table>
<tr><td>channel</td><td>端口号（20-29）</td></tr>
<tr><td>location</td><td>TCP</td></tr>
<tr><td>name</td><td>IP 地址＋端口号</td></tr>
<tr><td>access</td><td>READ_WRITE（该通道用于读写）</td></tr>
</table></td></tr>
<tr><td>实例</td><td>打开与 TCP 服务器的连接，服务器 IP 地址为 192.168.0.200，端口号为 8080；
控制器使用 20 号通道与服务器关联；
portnum = 20
OPEN #portnum AS "TCP:192.168.0.200(8080)" FOR READ_WRITE</td></tr>
<tr><td>注意</td><td>仿真器不支持 TCP/IP 通讯仿真；
端口号的范围为 20-29；
TCP 服务器 IP 地址应当与控制器 IP 地址在相同网段；</td></tr>
</table>

4.2.1.3 连接断开

<table>
<tr><td>指令格式</td><td>CLOSE channel</td></tr>
<tr><td>功能描述</td><td>关闭与服务器的连接；
<table>
<tr><td>channel</td><td>端口号（20-29）</td></tr>
</table></td></tr>
<tr><td>实例</td><td>关闭与 TCP 服务器的连接，通道为 20；
portnum = 20
CLOSE #portnum</td></tr>
</table>

4.2.1.4 连接状态检测

<table>
<tr><td>指令格式</td><td>value = IS_OPEN #channel</td></tr>
<tr><td>功能描述</td><td>检测通道是否打开；
<table>
<tr><td rowspan="2">value</td><td>TRUE</td><td>通道已打开</td></tr>
<tr><td>FALSE</td><td>通道已关闭</td></tr>
<tr><td>channel</td><td colspan="2">端口号（20-29）</td></tr>
</table></td></tr>
<tr><td>使用注意</td><td>连接成功建立后，IS_OPEN 状态为 TRUE；如果服务器关闭，但控制器未与服务器进行数据读写，IS_OPEN 的状态可能仍是 TRUE；如果服务器关闭，控制器端执行了读写操作，IS_OPEN 的状态将变为 OFF;</td></tr>
</table>

ESTUN & Trio

4.2.1.5 数据发送

4.2.1.5.1 PRINT

<table>
<tr><td>指令格式</td><td>PRINT [#channel,] print_expression</td></tr>
<tr><td>功能描述</td><td>往端口打印数据或者字符串，可用于数据发送；
<table>
<tr><td>channel</td><td colspan="2">TRIO 内部端口号</td></tr>
<tr><td>Print_expression</td><td colspan="2">发送信息</td></tr>
<tr><td colspan="3">Print_expression</td></tr>
<tr><td>;</td><td colspan="2">两个信息之间为分隔符</td></tr>
<tr><td>,</td><td colspan="2">两个信息之间为空格符</td></tr>
<tr><td rowspan="3">Number(w,x)</td><td colspan="2">可以发送数值，数值打印设置如下</td></tr>
<tr><td>w</td><td>数值的字符总长</td></tr>
<tr><td>x</td><td>小数点字符长度</td></tr>
<tr><td>“string”</td><td colspan="2">可以发送字符串</td></tr>
</table></td></tr>
<tr><td>实例</td><td>1. 使用引号发送字符串；
PRINT "CAPITALS and lower case CAN BE PRINTED"
2. 打印带有 4 个字符和 1 个小数位的 VR，然后在下一个选项卡中显示一个 2 个小数位的局部变量；
VR(1) = 6
variable = 410.5:
PRINT VR(1) [4, 1], variable[2]
3. 以十六进制格式将输入 8-16 的状态打印到 Motion Perfect 中的终端通道 5；
PRINT #5, HEX(IN(8, 16))
4. 特殊字符可以通过 CHR 指令表示；比如需要输出回车+换行符，查询 ASC 码表可在该特殊符号的 ASC 码为 13 和 10；
PRINT #5, "CAPITALS and lower case CAN BE PRINTED" + CHR(13) + CHR(10)</td></tr>
<tr><td>注意</td><td>1. 使用 PRINT 语句发送信息，无发送状态返回值；</td></tr>
</table>

ESTUN & Trio

4.2.1.5.2 CHANNEL_WRITE

<table>
<tr><td>指令格式</td><td>x = CHANNEL_WRITE(channel, storage_buffer[,number of bytes])</td></tr>
<tr><td>功能描述</td><td>往端口打印数据或者字符串，可用于数据发送；
<table>
<tr><td>channel</td><td>TRIO 内部端口号</td></tr>
<tr><td>Strrage_buffer</td><td>发送信息，可以为字符串，也可以为数组，也可以为 VR 地址</td></tr>
<tr><td>Number of bytes</td><td>传输的数据个数</td></tr>
</table></td></tr>
<tr><td>实例</td><td>1. 发送字符串变量存储的信息；
DIM output_string AS STRING(10)
output_string = "0123456789"
CHANNEL_WRITE(20 , output_string)
2. 发送数组存储的信息；
3. 发送 VR 变量存储的信息；</td></tr>
<tr><td>注意</td><td>1. 指令的返回值为发送数据的个数，可用于判断是否发送成功；</td></tr>
</table>

4.2.1.6 数据侦听

<table>
<tr><td>指令格式</td><td>value = KEY [#channel]</td></tr>
<tr><td>功能描述</td><td>反馈当前通道内的缓存字符数；</td></tr>
<tr><td>实例</td><td>当返回值为非 0 时，表示通道内有新的信息，使用 CHANNEL_READ 或者 GET 指令读取信息；
main:
WHILE TRUE
IF KEY#21 THEN
GOSUB read
ENDIF
WEND

read:
GET#1 k
RETURN</td></tr>
</table>

ESTUN & Trio

4.2.1.7 数据接收

4.2.1.7.1 GET

指令格式	GET [#channel,] variable
功能描述	将缓存区内的第一个字符取出，将其 ASC 码存储至 variable；

实例	读取某个输入字符；
	start:
	PRINT#5, "Press 'y' for YES or 'n' for NO."
	WAIT UNTIL KEY #5
	GET#5, char
	IF CHR(char) = "y" THEN
	PRINT#5, "YES selected"
	ELSEIF CHR(char) = "n" THEN
	PRINT#5, "NO selected"
	ELSE
	PRINT#5, "BAD selection"
	GOTO start
	ENDIF
	2.读取字符串信息；将通道 20 号的信息流读取，存储至字符串变量中；
	DIM l_s_recieve AS STRING(100)
	l_s_recieve = ""
	WAIT UNTIL KEY #20
	REPEAT
	GET#20, char
	l_s_recieve = l_s_recieve + CHR(char)
	UNTIL (KEY #20) = 0

4.2.1.7.2 CHANNEL_READ

指令格式	x = CHANNEL_READ(channel, storage_buffer[, delimiter_buffer[, escape_character[, crc]]])

ESTUN & Trio

功能描述

将通信通道内的信息读取出来存储至字符串变量或者 VR 变量中；

遇到以下 3 种情况，指令停止读取:

- 通道空；
- 读取信息超过字符串的最大长度；
- 读取到 delimiter_buffer 存储的结束符；

x	返回值	
value	通道号	
Storage_buffer	字符串变量名	存储读取信息的字符串变量
	数值	存储信息的 VR 首地址和长度
Delimiter_buffer	字符串变量名	存储结束符的字符串变量
	数值	存储结束符的 VR 首地址和长度
Escape_character	可选参数	
crc	可选参数	

实例

读取信息存储至字符串,信息存储至字符串变量 b，结束符存储在 d，结束符为空格加换行；

```
DIM b AS STRING( 100 )
DIM d AS STRING( 2 )
d = CHR( 13 ) + CHR( 10 ) ' set the delimiters
WAIT UNTIL KEY #20
CHANNEL_READ( 20 , b, d ) ' read channel 20.
```

读取信息存储至 VR，信息存储至 VR(100)-VR(199)，结束符存储在 VR(10,11)，结束符为空格加换行；

```
DIM portnum AS INTEGER
portnum = 20
DIM vr_rec_addr, vr_end_addr AS INTEGER
vr_rec_addr = 100
vr_end_addr = 10
DIM vr_rec_length, vr_end_length AS INTEGER
vr_rec_length = 100
vr_end_length = 2
' set the delimiters
VR( vr_end_addr ) = 13 'carriage return
VR( vr_end_addr + 1 ) = 10 'line feed
'读取信息
WAIT UNTIL KEY #portnum
CHANNEL_READ( portnum , vr_rec_addr , vr_rec_length , vr_end_addr ,
vr_end_length )
DIM b AS STRING( 100 )
```

ESTUN & Trio

	b = VRSTRING(vr_rec_addr + 1) '接收到的信息存储从设定的位置加 1 开始

4.2.1.8 实例

4.2.1.8.1 实例 1 建立与服务器的连接

```
DIM portnum AS INTEGER

portnum = 20

REPEAT
  CLOSE #portnum '关闭连接
  WA( 100 )
  OPEN #portnum AS "TCP:192.168.0.200(8080)" FOR READ_WRITE '建立连接
UNTIL IS_OPEN #portnum '判断连接状态
```

4.2.1.8.2 实例 2 交互实例

控制要求：

- 使用 TCP 通讯与 PC 软件进行数据交互；
- PC 软件 IP 地址为 192.168.0.220；
- PC 软件端口号为 8081；
- 请求命令（控制器—>PC）：GET:POS;
- 返回信息格式：POS:X,Y,Z,（POS:X1.2,Y2.3,Z22,）

代码实现：

交互逻辑：

```
DIM rec_string AS STRING(100)
DIM x ,y,z AS FLOAT
DIM x_s,y_s,z_s AS STRING(20)

DIM l_i_arr_pos_1 AS INTEGER(10)
DIM l_i_arr_pos_2 AS INTEGER(10)

'建立连接

OPEN #20 AS "TCP:192.168.0.220(8081)" FOR READ_WRITE

WHILE TRUE
```

ESTUN & Trio

```
  SELECT_CASE l_i_step
    CASE 0 '发送请求
      CHANNEL_WRITE(20, "GET:POS") '向上位机请求数据
      l_i_step = 3
    CASE 3 '接收信息
      '返回字符串格式
      'POS:X,Y,Z,
      IF KEY #20 THEN
        CHANNEL_READ(20,rec_string)
        l_i_step = 5
      ENDIF
    CASE 5 '查询逗号
      count_1 = fb_get_symbol_pos(rec_string,":",l_i_arr_pos_1)
      count_2 = fb_get_symbol_pos(rec_string,",",l_i_arr_pos_2)

      PRINT count_1,count_2
      PRINT l_i_arr_pos_1
      PRINT l_i_arr_pos_2
      l_i_step = 10
    CASE 10 '解析数据
      x_s = MID(rec_string,l_i_arr_pos_1(0)+1,l_i_arr_pos_2(0)-l_i_arr_pos_1(0)-1)
      y_s = MID(rec_string,l_i_arr_pos_2(0)+1,l_i_arr_pos_2(1)-l_i_arr_pos_2(0)-1)
      z_s = MID(rec_string,l_i_arr_pos_2(1)+1,l_i_arr_pos_2(2)-l_i_arr_pos_2(1)-1)
      l_i_step = 20
    CASE 20

      x = VAL(x_s)
      y = VAL(y_s)
      z = VAL(z_s)
  END_CASE
WEND
```

查询字符位置函数：

```
FUNCTION fb_get_symbol_pos(serch_string AS STRING, _
    serch_symbol AS STRING, returnval AS INTEGER()) AS INTEGER
  DIM len_string AS INTEGER
  DIM index,i AS INTEGER
  index = 0
  len_string = LEN(serch_string)
  FOR i = 0 TO len_string-1
    IF serch_symbol = MID(serch_string,i,1) THEN
      returnval(index)=i
      index = index+1
```

ESTUN & Trio

```
    ENDIF
  NEXT i
  RETURN index
ENDFUNC
```

4.2.1.9 使用注意

服务器不可以一直向控制器发送字符信息，如果一直发送，控制器未读取，会导致缓存区过大，触发控制器系统报警；

4.2.2服务器

4.2.2.1 描述

1. 控制器支持 TCP/IP 通讯协议，作为服务器被客户端访问；
2. 最大可建立一个服务器；
3. 通道 25 可用于建立 TCP/IP 服务器；

4.2.2.2 服务器建立

参考 OPEN 指令，通道 25 用于建立 TCP/IP 服务器；

```
DIM tcp_port AS INTEGER
DIM tcpserver_id AS STRING( 50 )
tcpserver_channel = 25
tcp_port = 790
tcpserver_id = "TCPSERVER:(" + STR( tcp_port, 0 ) + ")"
OPEN #tcpserver_channel AS tcpserver_id FOR READ_WRITE
```

4.2.2.3 服务器关闭

参考 CLOSE 指令

4.2.2.4 数据侦听

参考 TCP/IP 客户端数据侦听

4.2.2.5 数据发送

参考 TCP/IP 客户端数据发送

ESTUN & Trio

4.2.2.6 数据接收

参考 TCP/IP 客户端数据接收

4.2.2.7 实例

参考 TCP/IP 客户端 实例 2

4.3 UDP

4.3.1描述

1. 控制器支持 UDP 通讯协议；
2. 最大可建立 10 个 UDP 通讯；
3. 通道 20-29 可被 UDP 通讯使用；

4.3.2连接建立

参考 OPEN 指令，关键字为 dgram;

```
OPEN#20 AS "dgram:192.168.0.250(1000)" FOR READ_WRITE
```

4.3.3连接断开

参考 CLOSE 指令

4.3.4连接状态检测

参考 TCP/IP 客户端连接状态检测

4.3.5数据发送

参考 TCP/IP 客户端数据发送

4.3.6数据侦听

参考 TCP/IP 客户端数据侦听

ESTUN & Trio

4.3.7数据接收

参考 TCP/IP 客户端数据接收

4.3.8实例

参考 TCP/IP 客户端 实例 2

4.4 EtherCAT

4.4.1 相关概念

4.4.1.1 伺服周期

参考 SERVO_PERIOD。

表示 PDO 数据的刷新周期。

4.4.1.2 控制模式

1. 相关对象

索引	含义	数据类型	读写权限
6060:00	控制模式	INT8	RW
6061:00	当前控制模式	INT8	RO

2. 值范围

值	说明
0	未设置控制模式
1	位置控制模式
3	速度控制模式
4	转矩控制模式
6	回零模式
8	位置插补控制模式（CSP）DRIVE_MODE=1
9	位置插补控制模式（CSV）DRIVE_MODE=2
10	位置插补控制模式（CST）DRIVE_MODE=3

ESTUN & Trio

4.4.1.3 PDO

1. 含义：过程数据；
2. 刷新周期由 SERVO_PERIOD 设置;
3. 部分产品，数据库内包含不同 PDO 配置，用户可以通过 DRIVE_MODE、DRIVE_PROFILE 进行选择；
4. 在数据库中可以查看系统内置 PDO 选项；
5. 轴部分 SDO 功能需要 PDO 配置才能实现；
6. 如果数据库内无伺服产品，可通过 EC_EXTEND 文件进行设置；
7. PDO 的映射过程如下：

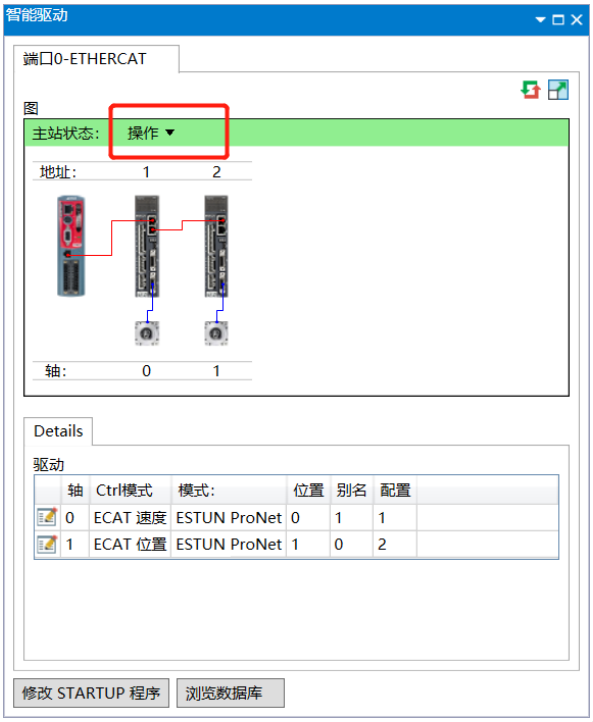
1 停止 PDO 分配功能（0x1C12 与 0x1C13 的子索引 0 设置为 0）。
2 停止 PDO 映射功能（0x1600～0x1603 和 0x1A00～0x1A03 的子索引 0 全部设为 0）。
3 设置 PDO 映射对象（0x1600～0x1603 和 0x1A00～0x1A03）的映射入口。
4 设置 PDO 映射对象（0x1600～0x1603 和 0x1A00～0x1A03）映射入口的数值。
5 设置 PDO 分配对象（设置 0x1C12 和 0x1C13 的子索引 1）。
6 重新打开 PDO 分配功能（设置 0x1C12 和 0x1C13 的子索引 0 为 1）。

4.4.1.4 SDO

1. 刷新周期非实时；
2. SDO 读通过指令 CO_READ_AXIS 实现，一般用于伺服状态的读取；
3. SDO 写通过指令 CO_WRITE_AXIS 实现，一般用于伺服参数的设置；

4.4.1.5 ETHERCAT 状态机

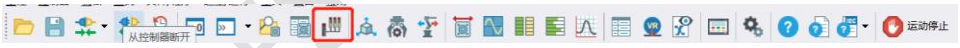
可在智能轴驱动查看总线的状态；也可以通过 ETHERCAT 指令查看总线状态；



4.4.2 智能轴驱动使用

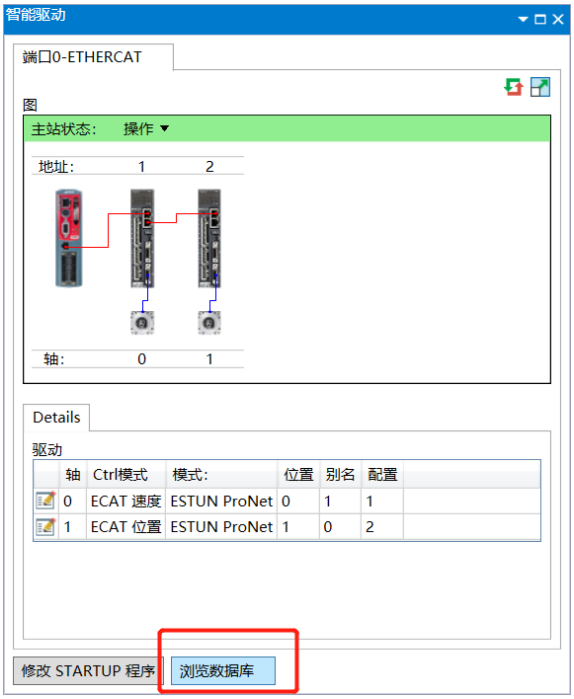
4.4.2.1 数据库查看

1. 连接实际控制器情况下，点击工具栏-智能轴驱动；

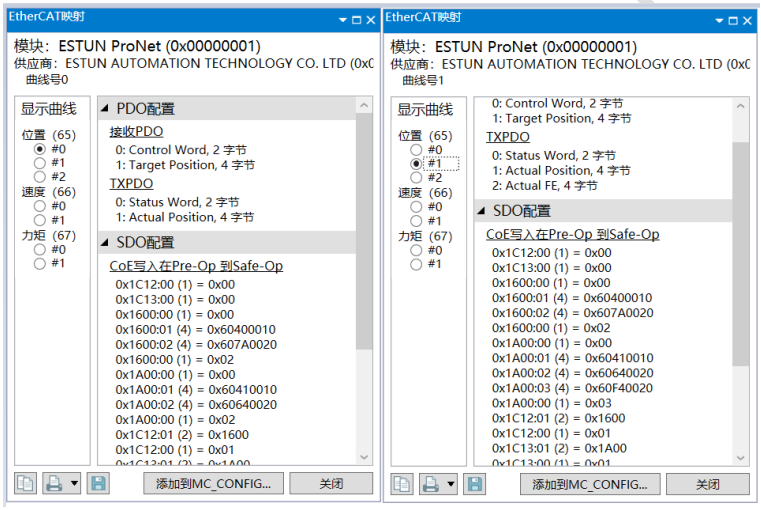


2. 选择浏览数据库

ESTUN & Trio

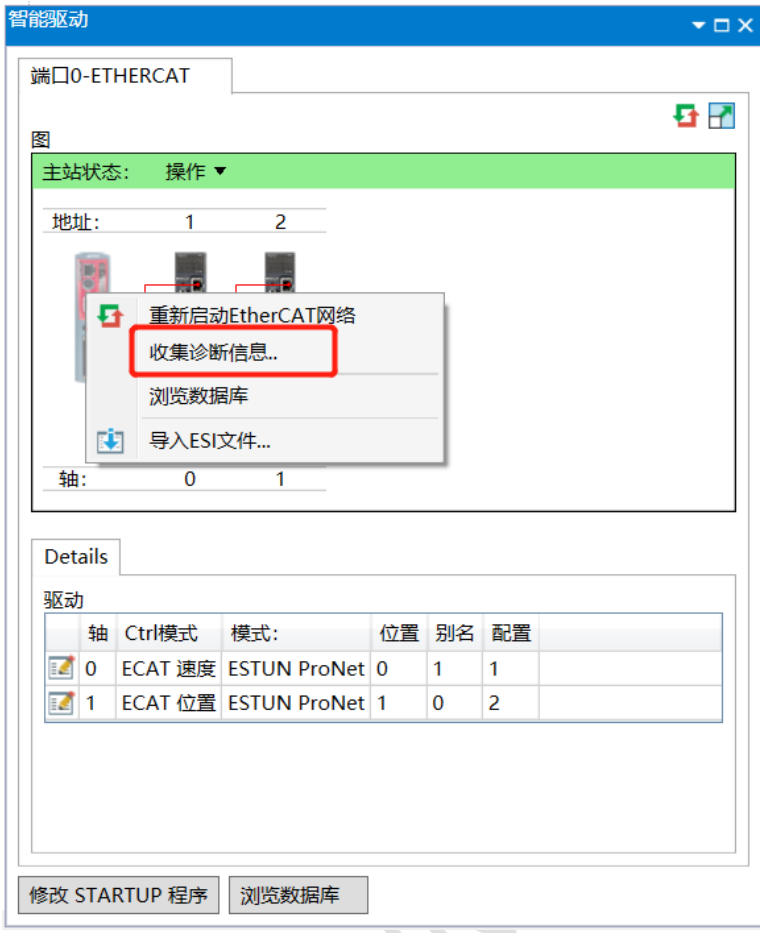


3. 查看对应型号内置 PDO 选项；



4.4.2.2 诊断报告收集

1. 点击工具栏-智能轴驱动；
2. 选中控制器图标，右键-收集诊断信息；



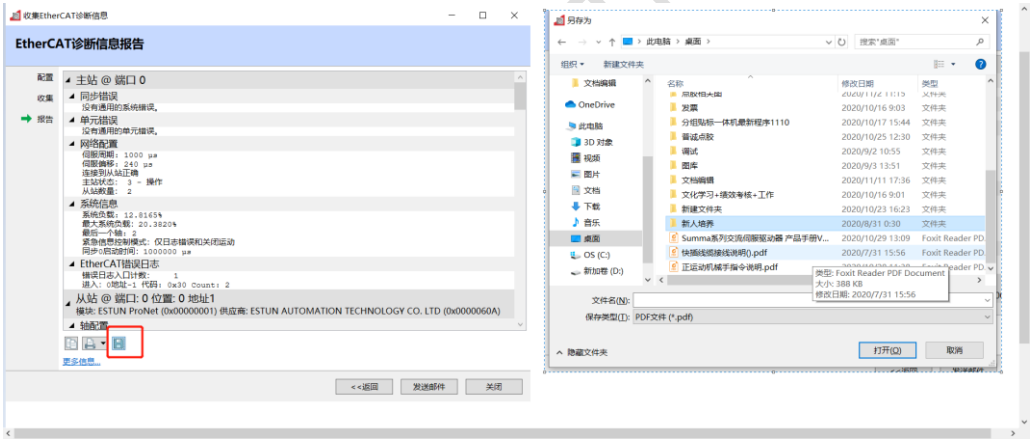
3. 勾选全部，点击下一步



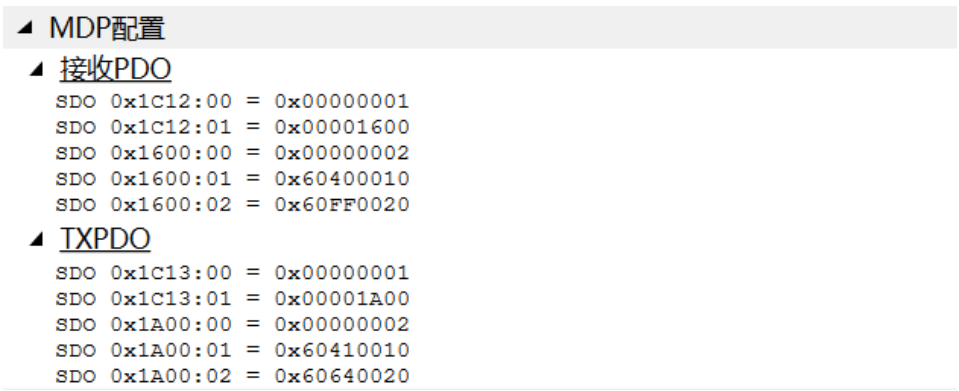
4. 生成诊断报告；



5. 另存为 PDF 文件；



6. 可在诊断报告中查询轴的 PDO 配置；



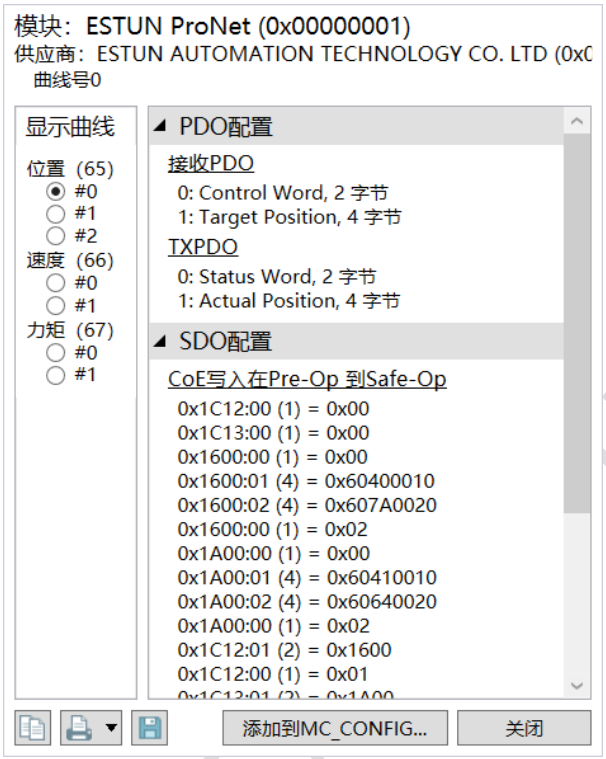
4.4.2.3 PDO 选择

A. 智能轴驱动中进行选择

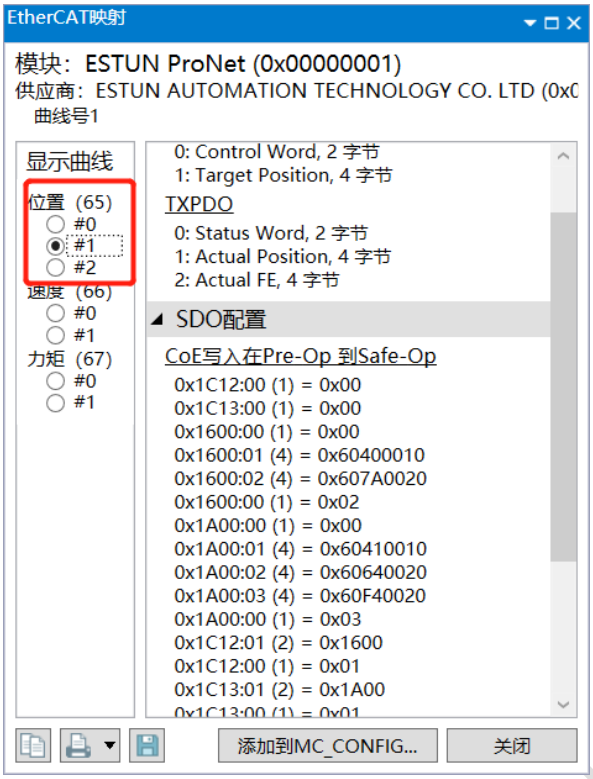
1. 点击工具栏-智能轴驱动；

2. 选择浏览数据库；

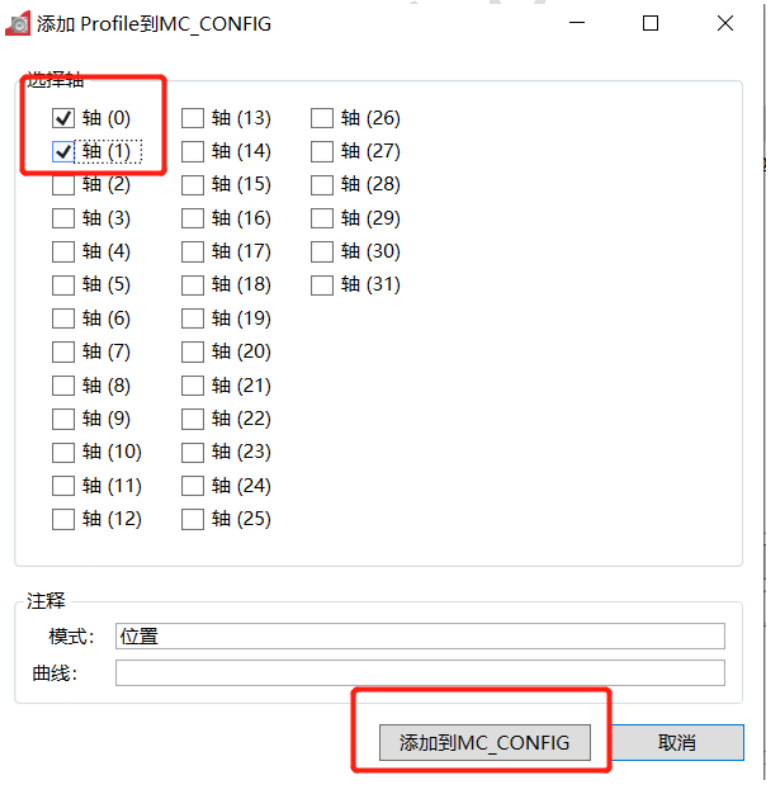
3. 查询当前伺服产品；



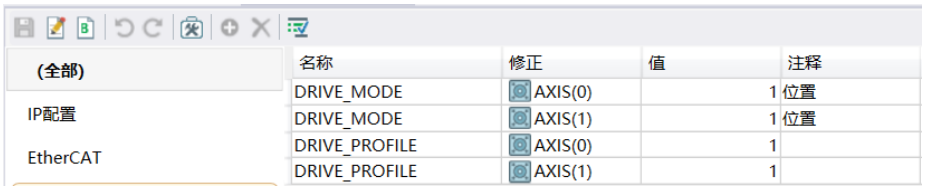
4. 选择模式，及内置 PDO 选项；



5. 点击添加到 MC_CONFIG 文件；勾选轴；

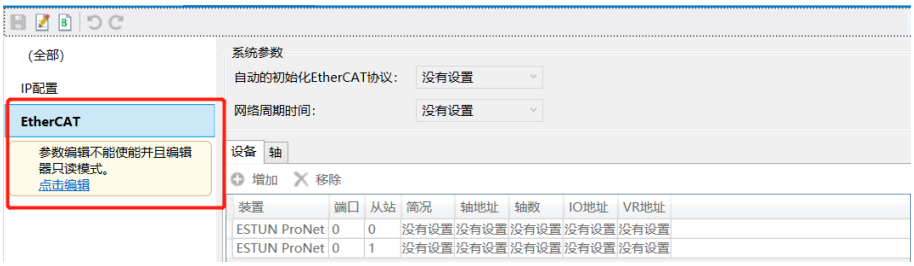


6. MC_CONFIG 文件中生成设置；

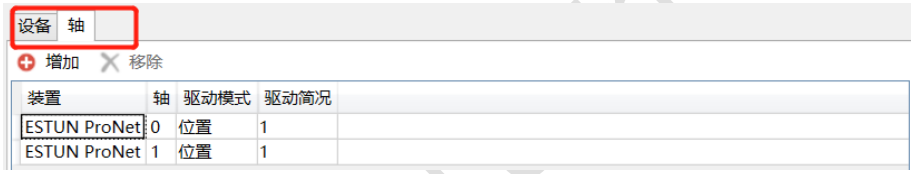


B. 在 MC_CONFIG 文件中编辑；

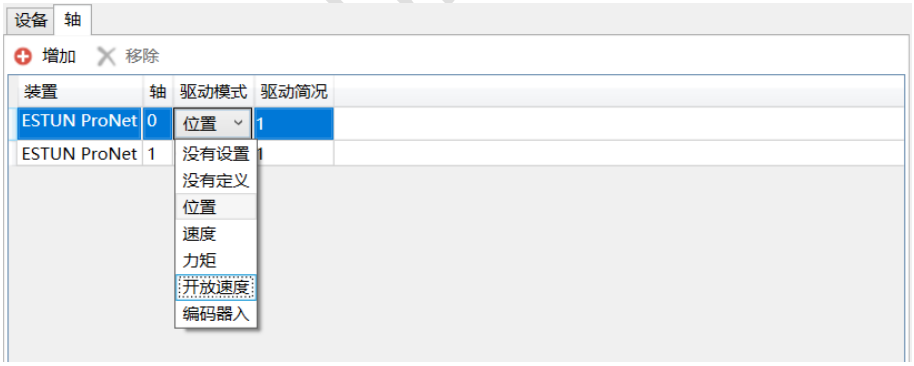
1. 打开 MC_CONFIG 文件，点击 ETHERCAT 选项，操作点击编辑按钮；



2. 切换至轴选项；



3. 修改驱动模式和驱动简况；



4. 生成设置。

4.4.3 PDO 配置

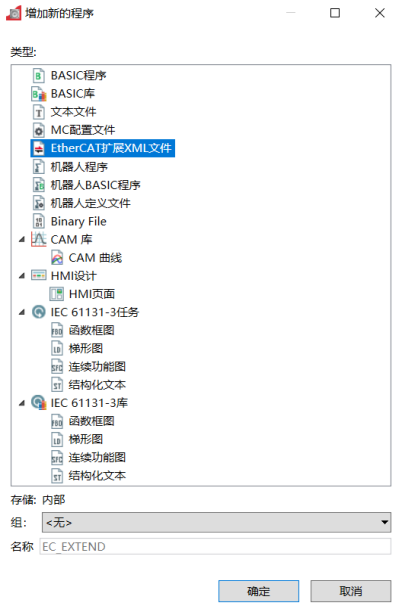
如果出现以下两种情况，需要用户自己编辑 PDO，

1. 当前数据库内查询不到驱动器型号，无内置 PDO 配置；
2. 内置的 PDO 无法满足应用需求；

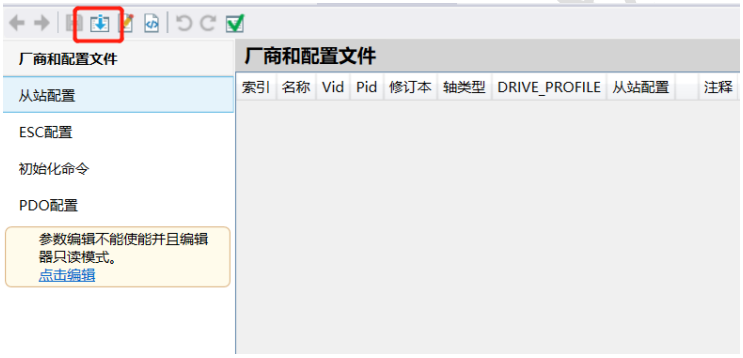
PDO 配置需要编辑 EC_EXTEND 文件。

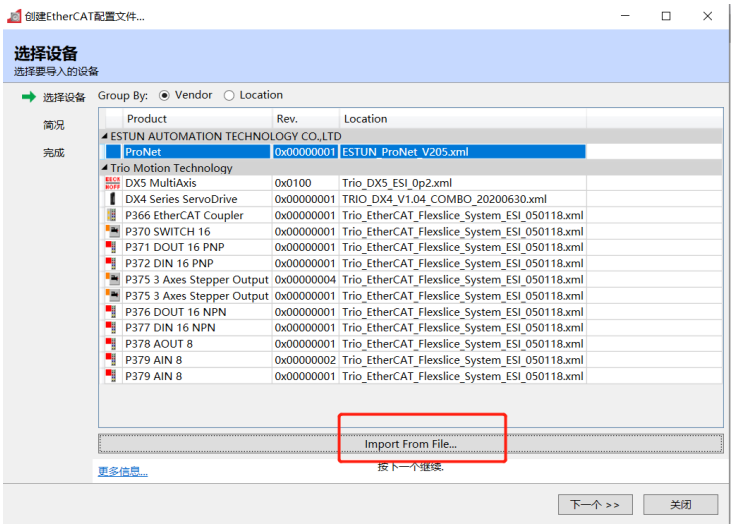
4.4.3.1 PDO 映射过程

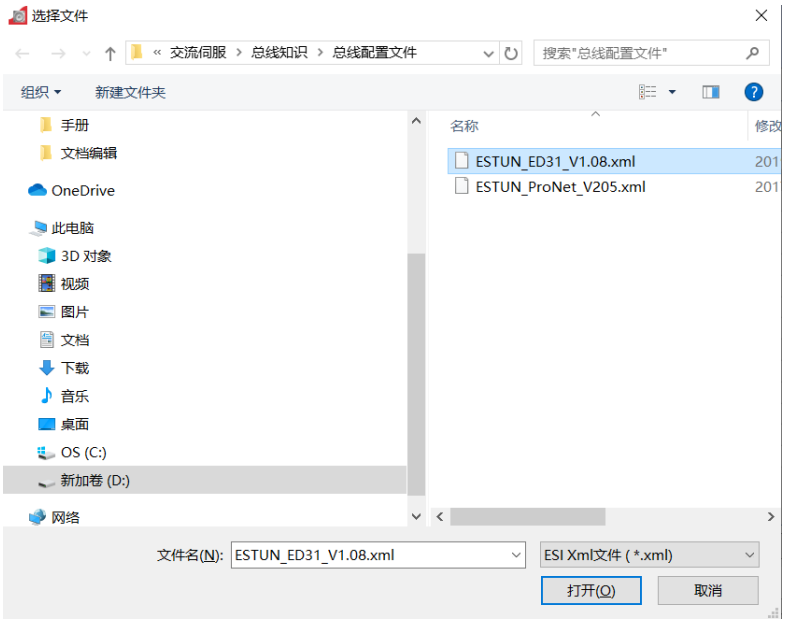
1. 新建 EC_EXTEND 文件；

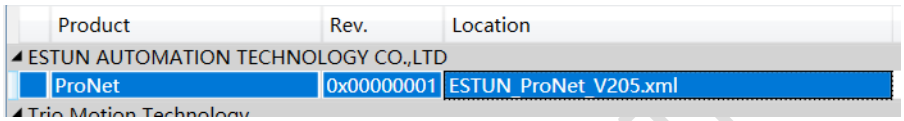


2. 加载驱动器的 xml 文件；

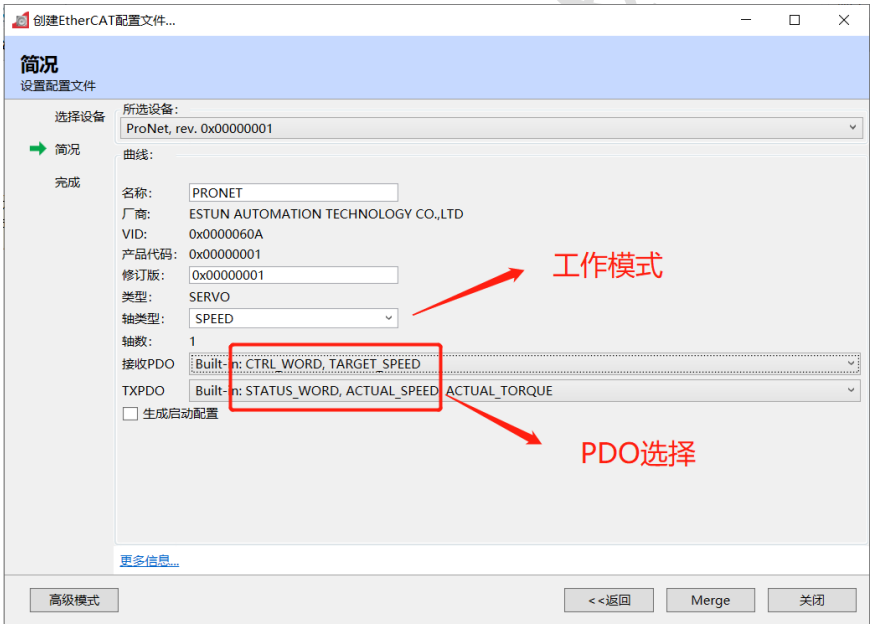








3. 选中产品 xml 文件，点击下一步进行编辑；

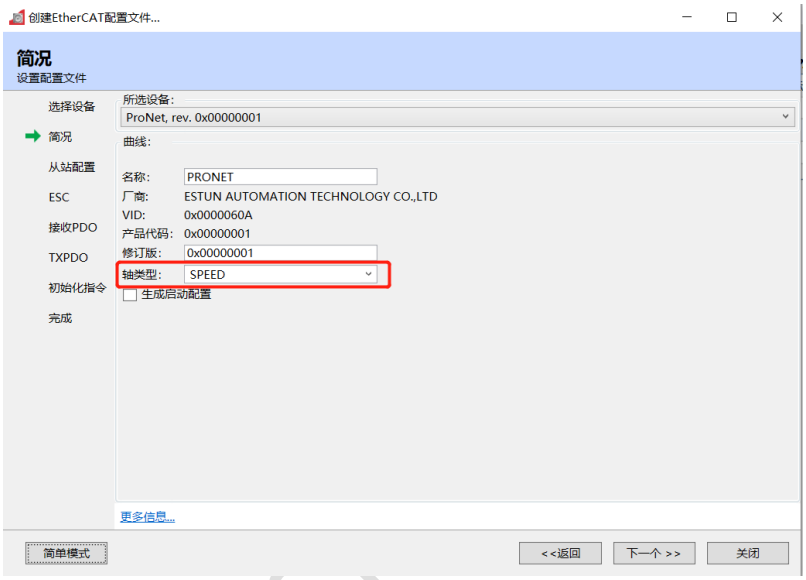


4. 点击 MERGE，可生产预设配置 PDO；



5. 点击高级模式，可以进一步编辑 PDO，添加其他的对象；

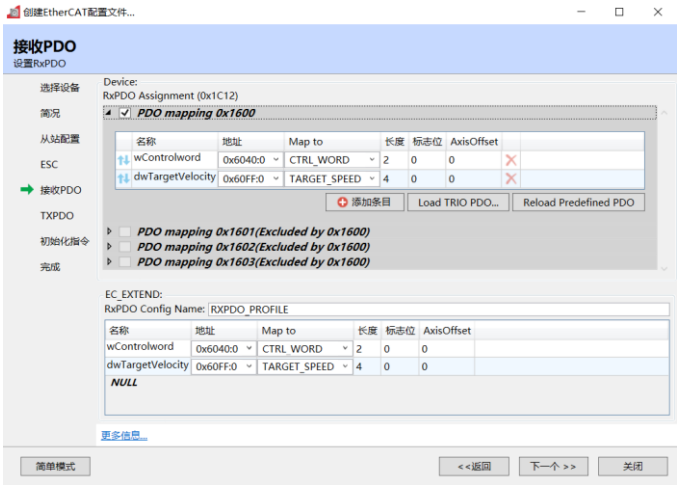
6. 设置轴类型



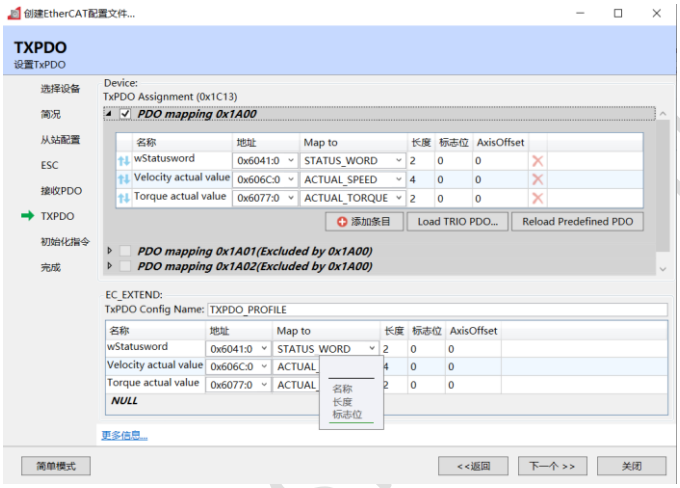
7. 设置从站参数；



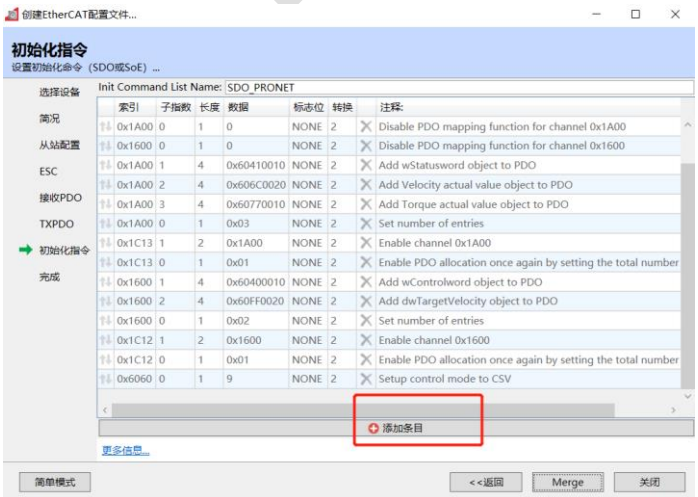
8. 接收 PDO 设置；



9. 写 PDO 设置；



10. 添加初始化 SDO 命令；



11. 查看生成文件；



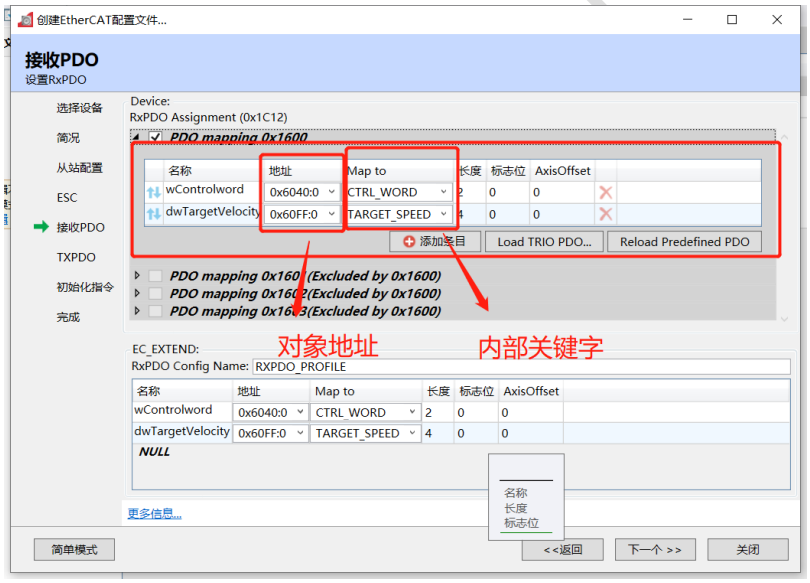
12. MC_CONFIG 文件设置 DRIVE_PROFILE 为《从站配置》值，即可选择该组 PDO 配置；

4.4.3.2 TxPDO RxPDO 与 BASIC 关键字的关联

EC_EXTEND 文件配置时，需要将对象地址与控制器的关键字进行关联，只有正确关联关键字，控制器指令方可正确执行。例如：

对象	关键字	运动指令
6040:00（控制字）	CTRL_WORD	WDOG/AXIS_ENABLE
6041:00（状态字）	STATUS_WORD	DRIVE_STATUSWORD

1. 在编辑 PDO 时，需要将对象与控制器的关键字进行关联；



2. 控制器内部将关键字与轴参数或者运动指令进行关联，参考下表；

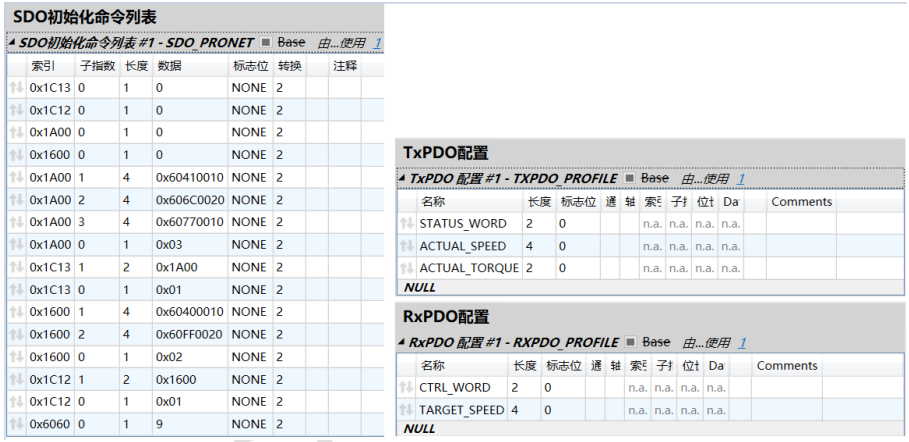
对象地址	关键字	BASIC 指令或参数
6061:00（控制模式）	ACTUAL_CTROL_MODE	内部参数
	ACTUAL_CURRENT	DRIVE_CURRENT（轴参数）
	ACTUAL_DRIVE_PARAM	DRIVE_PARAMETER（轴参数）

对象地址	关键字	BASIC 指令或参数
	ACTUAL_FE	DRIVE_FE（轴参数）
6064:00（电机位置）	ACTUAL_POS	ENCODER（轴参数）
	ACTULA_SLV_PARAM	内部参数
606C:00（电机转速）	ACTUAL_SPEED	DRIVE_VELOCITY（轴参数）
6077:00（电机转矩）	ACTUAL_TORQUE	DRIVE_TORQUE（轴参数）
	ACTUAL_VR_FLOAT32	VR(n)
	ACTUAL_VR_FLOAT64	VR(n)
	ACTUAL_VR_BIT	VR(n)
	ACTUAL_VR_INT	VR(n)
	ACTUAL_VR_MAPPED	VR(n)
	ACTUAL_VR_UINT	VR(n)
	AIN	AINn
	AIN_CH	AIN(channel)
	AOUT	AOUTn
	AOUT_CH	AOUT(channel)
6040:00(控制字)	CTRL_WORD	DRIVE_CONTROLWORD(轴参数)
60FD:00	DIN	IN
60FE:00	DOUT	OP
6041:00(状态字)	STATUS_WORD	DRIVE_STATUS（轴参数）
6060:00（控制模式）	TARGET_CTRL_MODE	内部参数
606C:00（目标位置）	TARGET_POS	DAC_OUT（只读，由运动指令生成）
60FF:00（目标速度）	TARGET_SPEED	DAC_OUT（只读，由 DAC 给出）
6071:00（目标转矩）	TARGET_TORQUE	DAC_OUT（只读，由 DAC 给出）
	TARGET _VR_FLOAT32	VR(n)
	TARGET _VR_FLOAT64	VR(n)
	TARGET _VR_BIT	VR(n)
	TARGET _VR_INT	VR(n)
	TARGET _VR_MAPPED	VR(n)
	TARGET _VR_UINT	VR(n)

ESTUN & Trio

对象地址	关键字	BASIC 指令或参数
60B8:00	TORCH_PROBE_FUNCTION	REGIST 和 DATUM
60B9:00	TORCH_PROBE_STATUS	REGIST 和 DATUM
60BA:00	TORCH_PROBE1_POS1	REGIST 和 DATUM
60BB:00	TORCH_PROBE1_POS2	REGIST 和 DATUM
60BC:00	TORCH_PROBE2_POS1	REGIST 和 DATUM
60BD:00	TORCH_PROBE2_POS2	REGIST 和 DATUM

3. 配置实例；



SDO初始化命令列表

SDO初始化命令列表 #1 - SDO_PRONET ■ Base 由...使用 1

索引	子指数	长度	数据	标志位	转换	注释
0x1C13	0	1	0	NONE	2	
0x1C12	0	1	0	NONE	2	
0x1A00	0	1	0	NONE	2	
0x1600	0	1	0	NONE	2	
0x1A00	1	4	0x60410010	NONE	2	
0x1A00	2	4	0x606C0020	NONE	2	
0x1A00	3	4	0x60770010	NONE	2	
0x1A00	0	1	0x03	NONE	2	
0x1C13	1	2	0x1A00	NONE	2	
0x1C13	0	1	0x01	NONE	2	
0x1600	1	4	0x60400010	NONE	2	
0x1600	2	4	0x60FF0020	NONE	2	
0x1600	0	1	0x02	NONE	2	
0x1C12	1	2	0x1600	NONE	2	
0x1C12	0	1	0x01	NONE	2	
0x6060	0	1	9	NONE	2	

TxPDO配置

TxPDO 配置 #1 - TXPDO_PROFILE ■ Base 由...使用 1

名称	长度	标志位	通	轴	索引	子指	位计	Da	Comments
STATUS_WORD	2	0			n.a.	n.a.	n.a.	n.a.	
ACTUAL_SPEED	4	0			n.a.	n.a.	n.a.	n.a.	
ACTUAL_TORQUE	2	0			n.a.	n.a.	n.a.	n.a.	

NULL

RxPDO配置

RxPDO 配置 #1 - RXPDO_PROFILE ■ Base 由...使用 1

名称	长度	标志位	通	轴	索引	子指	位计	Da	Comments
CTRL_WORD	2	0			n.a.	n.a.	n.a.	n.a.	
TARGET_SPEED	4	0			n.a.	n.a.	n.a.	n.a.	

NULL

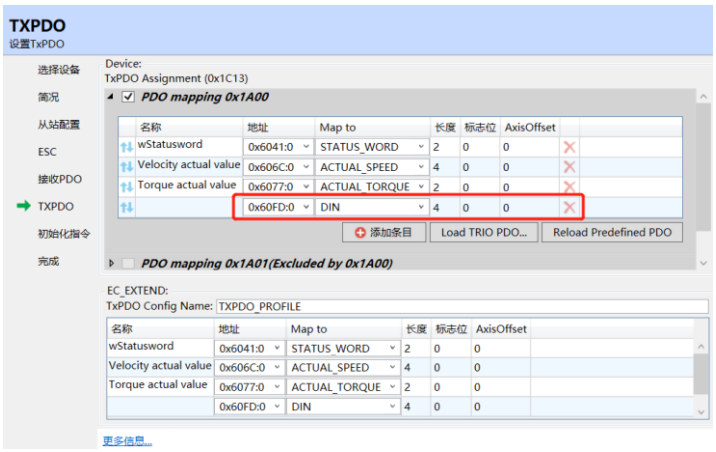
4.4.3.3 原点开关限位开关映射

很多应用中，将轴的原点开关及限位开关连接在驱动器 CN1 端子，节约输入模块个数。

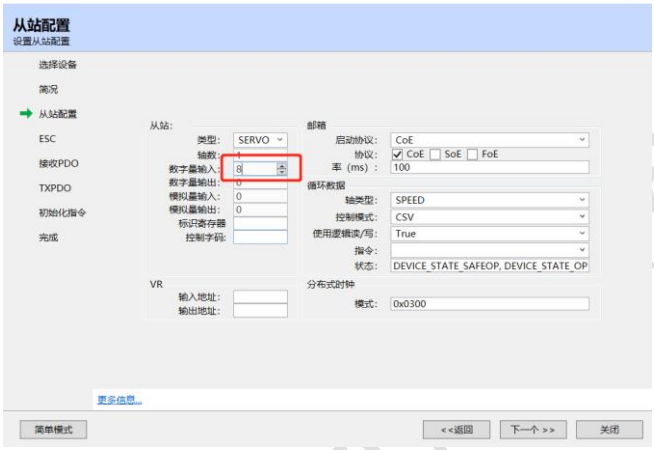
用户可以将原点相关的对象配置 PDO，控制器便可将 CN1 上的原点信号实时映射至控制器 DI 输入，使用与直接接入控制器的 DI 无异。

1. 读 PDO 将 60FD:00 与关键字 DIN 关联；

对象	关键字	运动指令
60FD:00（控制字）	DIN	映射至 IN
60FD 定义		
BIT0	负限位	
BIT1	正限位	
BIT2	原点	



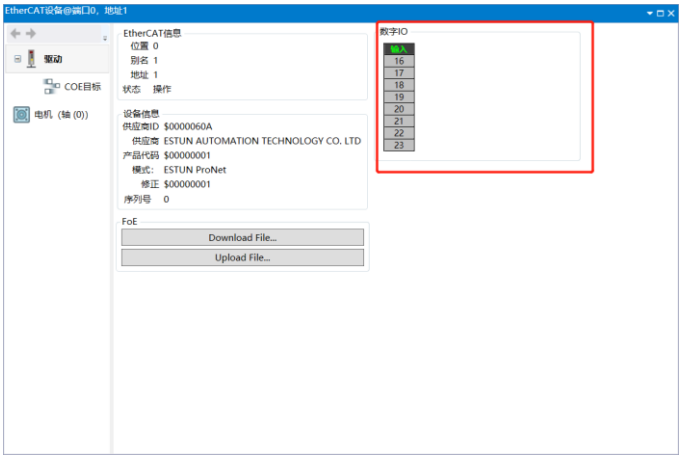
2. 从站配置中，将数字量输入设置为 8；



3. 生成配置文件，在 MC_CONFIG 文件中设置轴参数 DRIVE_PROFLE;

DRIVE_MODE	AXIS(0)	1	位置
DRIVE_MODE	AXIS(1)	1	位置
DRIVE_PROFILE	AXIS(0)	0	
DRIVE_PROFILE	AXIS(1)	0	

4. 重新初始化网络，在智能轴驱动中可看到轴上面关联输入；



5. 设置伺服参数，将输入信号设置为原点或者限位信号；

SUMMA 系列 PN509.0：为 CN1_14 分配信号；PN509.1：为 CN1_15 分配信号； PN509.2：为 CN1_16 分配信号；PN509.3：为 CN1_17 分配信号；	
1	P-OT（正限位）
2	N-OT（负限位）
6	HmRef（原点开关）
PRONET 系列 PN509.0：为 CN1_14 分配信号；PN509.1：为 CN1_15 分配信号； PN509.2：为 CN1_16 分配信号；PN509.3：为 CN1_17 分配信号；	
2	P-OT（正限位）
3	N-OT（负限位）
C	HmRef（原点开关）

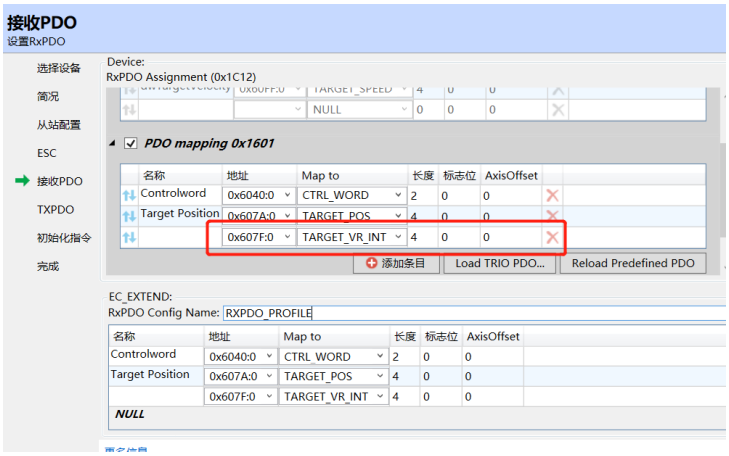
4.4.3.4 对象与 VR 的映射

部分对象，用户需要能够通过 PDO 方式对其进行读写，由于该对象没有关键字，无法通过操作轴参数的方式进行控制，那么用户可以将读对象与关键字 ACTUAL_VR_INT 关联，写对象与关键字 TARGET_VR_INT 关联，进而操作 VR 便可读写对象。

需求：

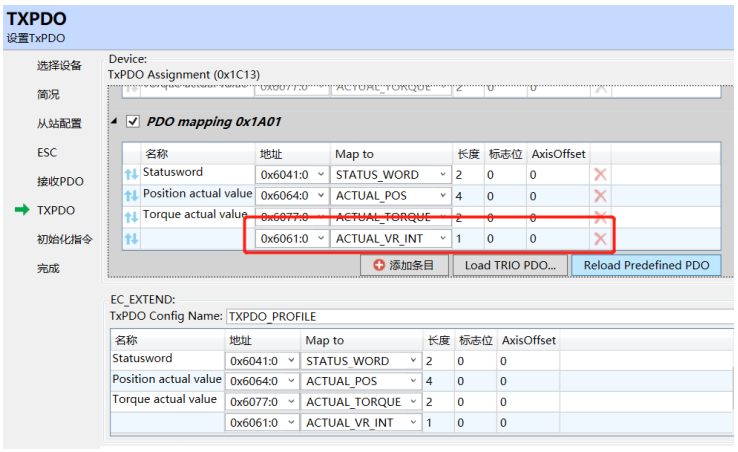
- 需要读取轴的控制模式(6061:00)，存储至 VR；
- 需要将轴的速度限幅（607F:00）关联至 VR，操作 VR，可以改变最大限幅；

1. 在写 PDO 中，将 607F:00 与关键字 TARGET_VR_IN 关联；



2. 在读 PDO 中，将 6061:00 与关键字 ACTUAL_VR_IN 关联；

ESTUN & Trio



3. 生成配置文件，查询 DRIVE_PROFILE 值；

4. 在 MC_CONFIG 文件中设置 VR 关联地址及 DRIVE_PROFLE;那么 0 号轴的最大速度限制可以通过 VR(2000)写，当前控制模式可以读取 VR(2001)。

装置	端口	从站	简况	轴地址	轴数	IO地址	VR地址
ESTUN ProNet	0	0	没有设置	没有设置	没有设置	没有设置	2000
ESTUN ProNet	0	1	没有设置	没有设置	没有设置	没有设置	没有设置

4.4.3.5 TOUCH_PROBE 功能

用户使用需求包含以下两种情况之 1 时，需要将 TOUCH_PROBE 相关对象配置 PDO，

1. DATUM 指令回零时，需要寻找 Z 脉冲；

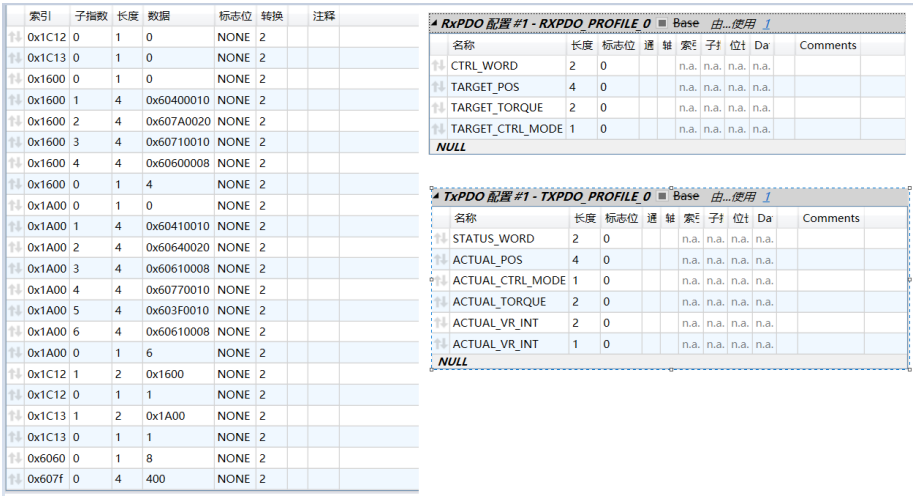
2. 使用 REGIST 指令做色标捕捉时，色标信号接入在伺服驱动器端子上；

TOUCH_PROBE 包含的对象地址和内部关联的关键字如下：

对象地址	关键字	BASIC 指令
60B8:00	TORCH_PROBE_FUNCTION	REGIST 和 DATUM
60B9:00	TORCH_PROBE_STATUS	REGIST 和 DATUM
60BA:00	TORCH_PROBE1_POS1	REGIST 和 DATUM
60BB:00	TORCH_PROBE1_POS2	REGIST 和 DATUM
60BC:00	TORCH_PROBE2_POS1	REGIST 和 DATUM
60BD:00	TORCH_PROBE2_POS2	REGIST 和 DATUM

4.4.3.6 控制模式切换

如果需要完成总线伺服位置模式和力矩模式的切换，需要将以下对象配置 PDO。



	索引	子指数	长度	数据	标志位	转换		注释
	0x1C12	0	1	0	NONE	2		
	0x1C13	0	1	0	NONE	2		
	0x1600	0	1	0	NONE	2		
	0x1600	1	4	0x60400010	NONE	2		
	0x1600	2	4	0x607A0020	NONE	2		
	0x1600	3	4	0x60710010	NONE	2		
	0x1600	4	4	0x60600008	NONE	2		
	0x1600	0	1	4	NONE	2		
	0x1A00	0	1	0	NONE	2		
	0x1A00	1	4	0x60410010	NONE	2		
	0x1A00	2	4	0x60640020	NONE	2		
	0x1A00	3	4	0x60610008	NONE	2		
	0x1A00	4	4	0x60770010	NONE	2		
	0x1A00	5	4	0x603F0010	NONE	2		
	0x1A00	6	4	0x60610008	NONE	2		
	0x1A00	0	1	6	NONE	2		
	0x1C12	1	2	0x1600	NONE	2		
	0x1C12	0	1	1	NONE	2		
	0x1C13	1	2	0x1A00	NONE	2		
	0x1C13	0	1	1	NONE	2		
	0x6060	0	1	8	NONE	2		
	0x607f	0	4	400	NONE	2		

RxPDO 配置 #1 - RXPDO_PROFILE_0 Base 由...使用 1

	名称	长度	标志位	通	轴	索引	子指	位计	Da	Comments
	CTRL_WORD	2	0			n.a.	n.a.	n.a.	n.a.	
	TARGET_POS	4	0			n.a.	n.a.	n.a.	n.a.	
	TARGET_TORQUE	2	0			n.a.	n.a.	n.a.	n.a.	
	TARGET_CTRL_MODE	1	0			n.a.	n.a.	n.a.	n.a.	

NULL

TxPDO 配置 #1 - TXPDO_PROFILE_0 Base 由...使用 1

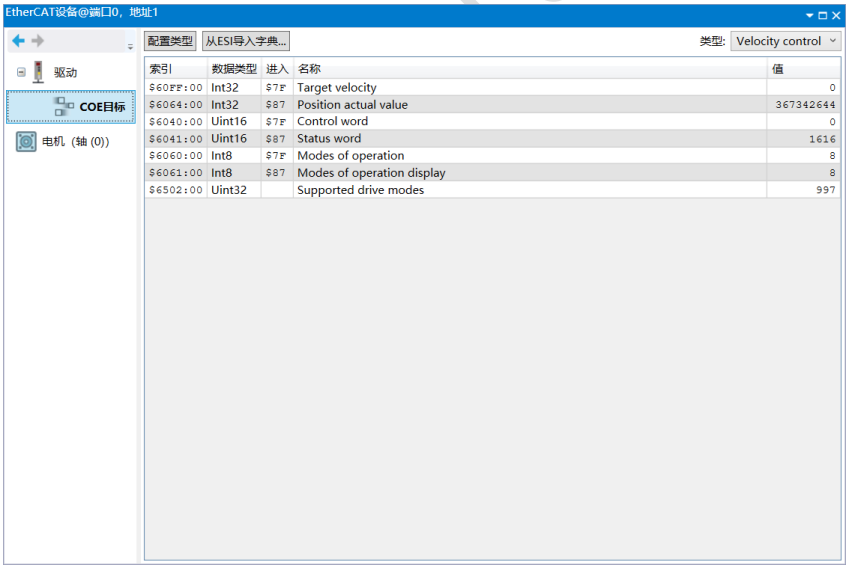
	名称	长度	标志位	通	轴	索引	子指	位计	Da	Comments
	STATUS_WORD	2	0			n.a.	n.a.	n.a.	n.a.	
	ACTUAL_POS	4	0			n.a.	n.a.	n.a.	n.a.	
	ACTUAL_CTRL_MODE	1	0			n.a.	n.a.	n.a.	n.a.	
	ACTUAL_TORQUE	2	0			n.a.	n.a.	n.a.	n.a.	
	ACTUAL_VR_INT	2	0			n.a.	n.a.	n.a.	n.a.	
	ACTUAL_VR_INT	1	0			n.a.	n.a.	n.a.	n.a.	

NULL

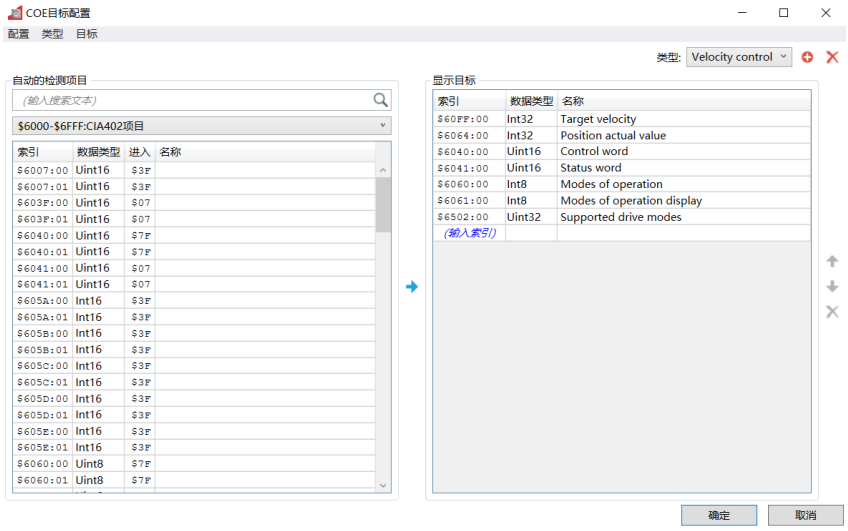
4.4.4 SDO 操作

4.4.4.1 对象数值查看

1. 打开智能轴驱动；
2. 双击需要操作的轴图标；
3. 选择《COE 目标》；



4. 点击《配置类型》，可以添加对象至监窗口；

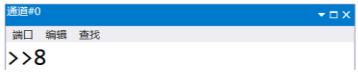
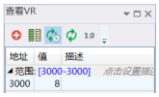


5. 可在窗口监控所有对象，或者写未配置进入 PDO 的对象。

4.4.4.2 CO_READ_AXIS

<table>
<tr><td>指令形式</td><td colspan="3">CO_READ_AXIS(axis_number, index, subindex ,type [,vr_number])</td></tr>
<tr><td>功能描述</td><td colspan="3">读取 ETHERCAT 对象，存储至 VR 或者打印在窗口 0；
<table>
<tr><td>Axis_num</td><td colspan="2">轴号</td></tr>
<tr><td>index</td><td colspan="2">对象索引</td></tr>
<tr><td>subindex</td><td colspan="2">对象子索引</td></tr>
<tr><td rowspan="10">type</td><td>1</td><td>Boolean</td></tr>
<tr><td>2</td><td>Int8</td></tr>
<tr><td>3</td><td>Int16</td></tr>
<tr><td>4</td><td>Int32</td></tr>
<tr><td>5</td><td>Uint8</td></tr>
<tr><td>6</td><td>Uint16</td></tr>
<tr><td>7</td><td>Uint32</td></tr>
<tr><td>8</td><td>Real32</td></tr>
<tr><td>9</td><td>String</td></tr>
<tr><td>Vr_number</td><td colspan="2">存储对象值的 VR 地址，如果该值为-1，则值打印在窗口 0；</td></tr>
</table></td></tr>
</table>

ESTUN&Inc

实例	读取 6060:00,打印在口 0；
	`CO_READ_AXIS( 0, $6061, 0, 2, -1 )`
	 
	
	读取 6060:00，存储在 VR(300)
	`CO_READ_AXIS( 0, $6061, 0, 2, 3000 )`
	 

备注	如果对非 ETHERCAT 总线轴执行该操作，运报错；
	`CO_READ_AXIS( 5, $6061, 0, 2, 3000 )`
	`Parameter 1 out of range`

4.4.4.3 CO_WRITE_AXIS

指令形式	`CO_WRITE_AXIS(axis_number, index, subindex, type, vr_number [,value])`
功能描述	写 ETHERCAT 对象，写 VR 存储数值给对象或者写固定数给对象；

Axis_num	轴号	
index	对象索引	
subindex	对象子索引	
type	1	Boolean
	2	Int8
	3	Int16
	4	Int32
	5	Uint8
	6	Uint16
	7	Uint32
	8	Real32
	9	String
Vr_number	存储写值的 VR 地址，如果该值为-1，则将 VALUE 中的值写入对象	
value	当 vr_number 设置为-1 有效	

实例	写 607F:00，给定值为 100； `CO_WRITE_AXIS( 0, $607F, 0, 7, -1, 1000 )` 写 607F:00，给定值存储在 VR(200)；

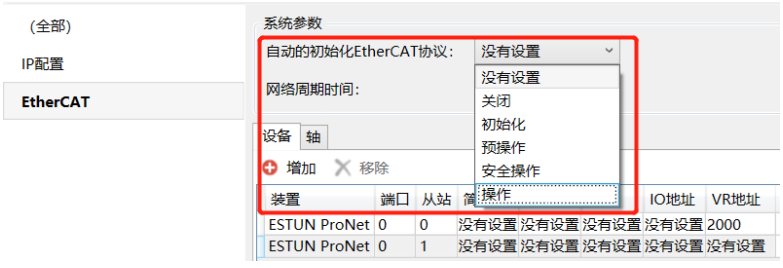
	CO_WRITE_AXIS(0, $607F, 0, 7, 2000)
备注	如果对非 ETHERCAT 总线轴执行该操作，运报错； CO_WRITE_AXIS(5, $607F, 0, 7, 2000) Parameter 1 out of range 可通过指令返回值判断是否写成功，如果写的值超过对象设定范围，写入不成功； 如果对象配置 PDO，写不成功；

4.4.5 总线初始化

4.4.5.1 AUTO_ETHERCAT

功能描述	设置 ETHERCAT 网络上电后的进入状态； 在 MC_CONFIG 文件中设置； 默认设定值是进入操作模式；

value	描述
0	上电后，ETHERCAT 网络不工作
1	上电后，ETHERCAT 网络自动工作，如果无误，将进入操作模式
$81	上电后，ETHERCAT 网络自动工作，如果无误，将进入操作模式
$41	上电后，ETHERCAT 网络自动工作，如果无误，将进入安全操作模式
$21	上电后，ETHERCAT 网络自动工作，如果无误，将进入预操作模式
$11	上电后，ETHERCAT 网络自动工作，如果无误，将进入初始化模式

实例	在 MC_CONFIG 文件行设置 
备注	控制器的启动时间比驱动器快，所以建议将该值设置为 0，通过程序控

ESTUN Trio

	制 ETHERCTA 网络进入操作模式，具体可参考 ETHERCAT 指令；

4.4.5.2 ETHERCAT

4.4.5.2.1 网络连接

指令形式	ETHERCAT(0, slot, [,MAC_retries])
功能描述	建立 ETHERCAT 网络连接，进入 OP 模式； slot：ETHERCAT 网口所在 SLOT 号，一般为 0
实例	建立网络连接，进入 OP 模式； ETHERCAT(0, 0)

4.4.5.2.2 网络断开

指令形式	ETHERCAT(1, slot)
功能描述	关闭 ETHERCAT 网络连接，进入初始化模式； slot：ETHERCAT 网口所在 SLOT 号，一般为 0
实例	关闭网络连接，进入初始化模式； ETHERCAT(1, 0)
注意	关闭网络，ETHERCAT 网络上的所有数字量输出状态将变为 0；

4.4.5.2.3 检测网络当前模式

指令形式	ETHERCAT($22, slot, vr_number)
功能描述	读取 ETHERCAT 网络当前状态； **slot**：ETHERCAT 网口所在 SLOT 号，一般为 0 **Vr_number**：存储 ETHERCAT 网络当前状态的 VR 地址 **0**：初始化模式 **1**：预操作模式 **2**：安全操作模式 **3**：操作模式
实例	读取网络状态，存储至 VR(100); ETHERCAT($22, 0, 100)
注意	关闭网络，ETHERCAT 网络上的所有数字量输出状态将变为 0；

ESTUN & Trio

4.4.5.2.4 驱动器故障清除

<table>
<tr><td>指令形式</td><td>ETHERCAT($64, axis_number[, mode[, timeout]])</td></tr>
<tr><td>功能描述</td><td>清除驱动器故障；
<table>
<tr><td>Axis_num</td><td colspan="2">轴号</td></tr>
<tr><td rowspan="2">Mode</td><td>0</td><td>控制字 BIT7 置 1，一段时间后清 0</td></tr>
<tr><td>1</td><td>控制字 BIT7 置 1，直到状态字 BIT3 清 0，控制字 BIT7 清 0；或者超时时间到，控制字 BIT7 清 0；</td></tr>
<tr><td>timeout</td><td colspan="2">Mode=1 时生效，超时时间，单位为 ms，范围为 1-10000ms;</td></tr>
</table></td></tr>
<tr><td>实例</td><td>通过轴 0 的 AXISSTATUS 检查驱动器是否故障，如果故障，操作 VR(0)，清除轴故障；
<pre>
IF VR(0) = 1 THEN
  IF AXISSTATUS AXIS(0).3 = 1 THEN
    IF ATYPE AXIS(0) >= 65 AND ATYPE AXIS(0) <= 67 THEN
      ETHERCAT($64, 0)
    ENDIF
    DATUM(7) AXIS(0)
  ENDIF
  VR(0) = 0
ENDIF
</pre></td></tr>
</table>

4.4.5.2.5 总线初始化实例

1) MC_CONFIG 文件中将 AUTO_ETHERCAT 设置为 0；

2) 初始化程序中执行代码:

ESTUN & Trio

```
DIM ec_times AS INTEGER
DIM result AS BOOLEAN
DIM ec_act_state AS INTEGER
IF CONTROL <> 400 THEN

    result = ETHERCAT( $22, 0, c_v_ec_state )

    VR(c_v_ec_error) = 0
    IF result THEN
        ec_act_state = VR(c_v_ec_state)
        VR(c_v_ec_state) = 0
    ENDIF

    IF ec_act_state <> 3 THEN
        ec_times = 0
        REPEAT
            ETHERCAT($1, 0)
            WA(1000)
            result = ETHERCAT($0, 0)
            ec_times = ec_times + 1
        UNTIL result OR ec_times > 3

        IF ec_times > 3 THEN
            VR(c_v_ec_error) = 1
        ENDIF
    ENDIF
    result = ETHERCAT($22, 0, c_v_ec_state)
ENDIF
```

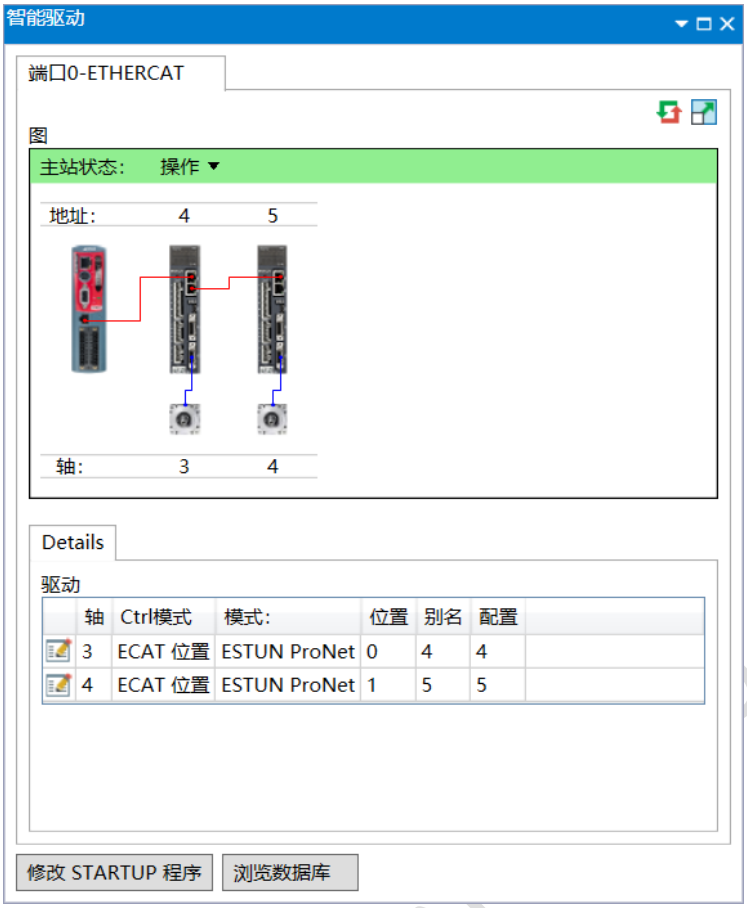
4.4.6 伺服节点设置

4.4.6.1 通过伺服设置指定节点

1. 设置驱动器参数 PN704，重新上电生效；不同驱动器 PN704 设定值不一致；

Pn704	3424	EtherCAT 通信节点设置	0～127	–	1	重启
	用于设置驱动器在 EtherCAT 通信网络中的节点号。					

2. 将网络上的两个伺服 PN704 设置为 4,5，总线初始化成功后，两个轴的轴号分别为 3,4；

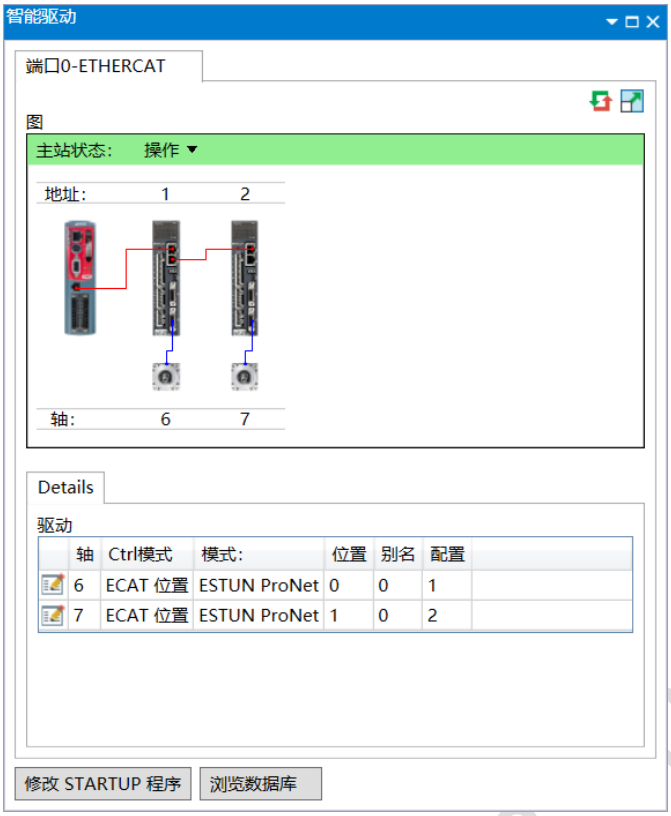


4.4.6.2 通过 NODE_AXIS 设置节点

1. 驱动器参数 PN704 均恢复为默认值 1；
2. 在 MC_CONFIG 文件中设置 NODE_AXIS 参数;



3. 复位控制器；



4. NOTE:如果 PN704 和 MC_CONFIG 文件均均设置了节点轴号，以 MC_CONFIG 文件中设置为准。

4.4.7常见总线错误

4.4.7.1 单元站丢失-0x40000

单元站丢失（0X40000），表示 ETHERCAT 网络拓扑上某个节点的连接出现问题。

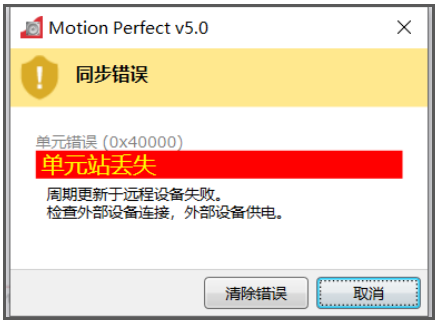


图 单元站丢失错误

排查步骤：

- 查看网线是否松动；
- 查看伺服 ETHERCAT 状态指示灯，从哪个节点开始故障指示灯闪烁，表示该节点的连接出现问题；参考故障示意图，可知 1#驱动器处的连接出现松动，需要排查该

处的硬件连接；



故障示意图

4.4.7.2 EC 初始化时无法指定轴号

EC 初始化时无法指定轴号，指定是 ETHERCAT 网络在进入 OP 模式时，无法给 ETHERCAT 网络上的轴分配轴号，具体错误体现在：

```
>>ethercat(0,0)
EtherCAT State: Init (0)
EtherCAT State: Init (0)
EtherCAT Warning: Ignored duplicated config station alias addr (of Slave Slot: 0,
Err: axis already assigned (Slave Pos: 0, Axis: 0).
EtherCAT State: Init (0)
EtherCAT Configuration (0):
  Estun ProNet:  0 : 1 : 1 () - ESC Status: 0x01, Code: 0x00
  Estun ProNet:  1 : 1 : 2 () - ESC Status: 0x01, Code: 0x00
  Estun ProNet:  2 : 1 : 3 () - ESC Status: 0x01, Code: 0x00
Error - unable to assign required axis number.
EtherCAT State: Init (0)
```

EC 初始化无法指定轴号错误

造成该故障的原因：

- MC4N/MC6N 等本体上包含编码器轴的控制器，未正确设置系统参数 AXIS_OFFSET；默认设置下，本体编码器轴号为 0，ETHERCAT 远程轴如果未做节点设置，也需将轴 0 分配给节点轴，导致产生该错误；
- 驱动器的节点设置超过控制器所能分配的最大轴号；（控制器固件版本过低时，当固件版本大于 2.303，则不会出现）；

4.4.7.3 伺服使能报错

伺服使能报错，该问题为：控制器与驱动器在未使能状态下通讯正常，在使能状态下，

则会出现通讯异常，使能断开的现象。

出现该问题的原因主要是：

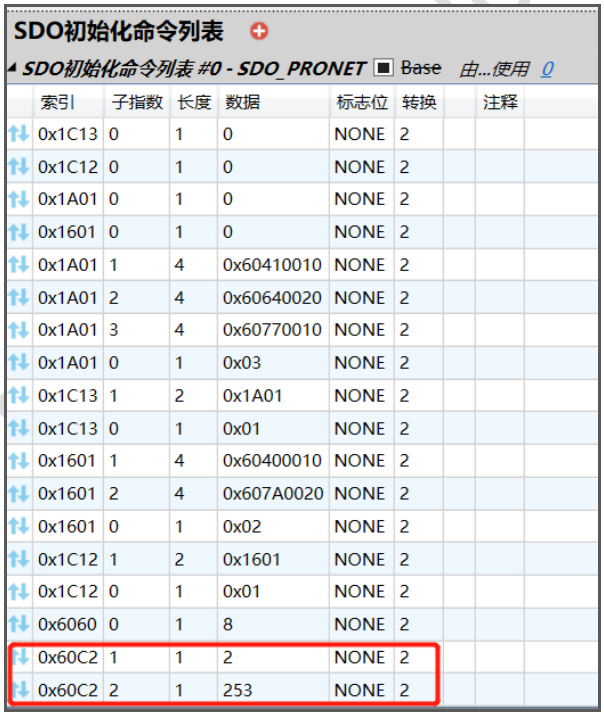
- 控制器的总线通讯周期与驱动器 CSP 控制的插补周期设置不一致（部分厂商伺服产品在 ETHERCAT 总线初始化时需要通过 SDO 命令设置插补周期）；

解决方法：

- 在 ETHERCAT 总线初始化时需要通过 SDO 命令设置插补周期，插补周期与控制器通讯周期一致；



MC_CONFIG 文件中设置 SERVO_PERIOD



索引	子指数	长度	数据	标志位	转换		注释
0x1C13	0	1	0	NONE	2		
0x1C12	0	1	0	NONE	2		
0x1A01	0	1	0	NONE	2		
0x1601	0	1	0	NONE	2		
0x1A01	1	4	0x60410010	NONE	2		
0x1A01	2	4	0x60640020	NONE	2		
0x1A01	3	4	0x60770010	NONE	2		
0x1A01	0	1	0x03	NONE	2		
0x1C13	1	2	0x1A01	NONE	2		
0x1C13	0	1	0x01	NONE	2		
0x1601	1	4	0x60400010	NONE	2		
0x1601	2	4	0x607A0020	NONE	2		
0x1601	0	1	0x02	NONE	2		
0x1C12	1	2	0x1601	NONE	2		
0x1C12	0	1	0x01	NONE	2		
0x6060	0	1	8	NONE	2		
0x60C2	1	1	2	NONE	2		
0x60C2	2	1	253	NONE	2		

EC_EXTEND 中写插补周期对象

NOTE:不是所有的伺服驱动器需要设置该参数，如遇到伺服使能出错，需要排查该因素。

4.5

5 机器人编程系统 RPS

机器人编程系统(RPS: Robotic Programming System)是一个开放的、易用性强的工具和软件包，提供了对机器人系统的完全控制和管理，降低编程的复杂度。在本章中，介绍了机器人语言扩展块。机器人语言扩展块是一种基于 Trio BASIC 的专门用于机器人技术的语言，它通过特定的机器人指令提高编程效率。它与标准的 TRIO BASIC 非常匹配，可以协同工作。为了在控制器中激活 RPS，需要一个配置数据来指定机械连杆的长度、操作模式和其它信息。

为了让系统工作，必须安装以下组件版本:

- 控制器固件版本 Firmware 2.03 以上
- Motion Perfect 5.0 以上

5.1 如何建立一个机器人工程

5.1.1 创建一个机器人模型

5.1.1.1 创建 ROBOT_DEFINITIONS 程序

为了让 RPS 运行，必须在一个名为 ROBOT_DEFINITIONS 的特殊程序中定义机器人。这个程序可以通过一个图形界面或一个文本编辑器来完成。

创建一个 ROBOT_DEFINITIONS 程序，在控制器项目程序树中右键单击“Programs”，点击“new…”添加一个新程序:

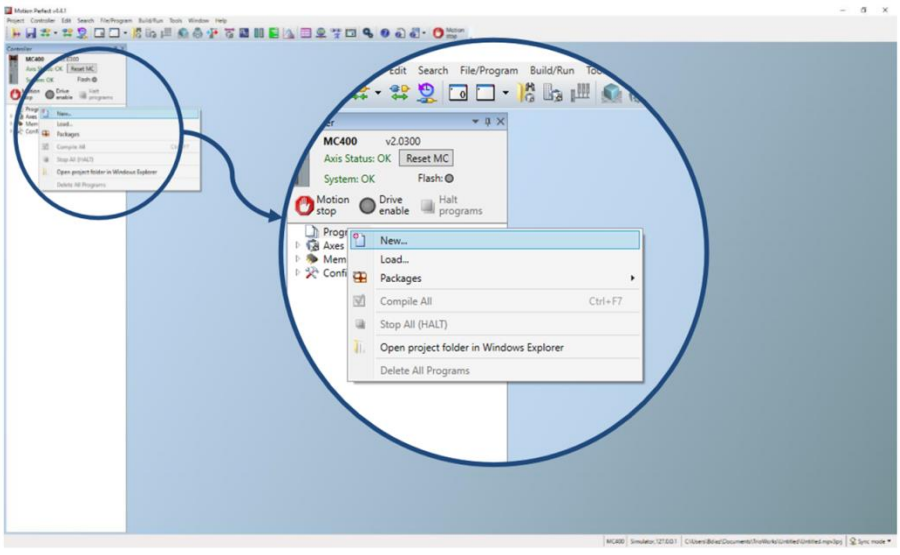


图 5-1 新建 ROBOT_DEFINITIONS 程序

ESTUN & Trio

选择 ROBOT_DEFINITIONS 文件并单击“OK”。

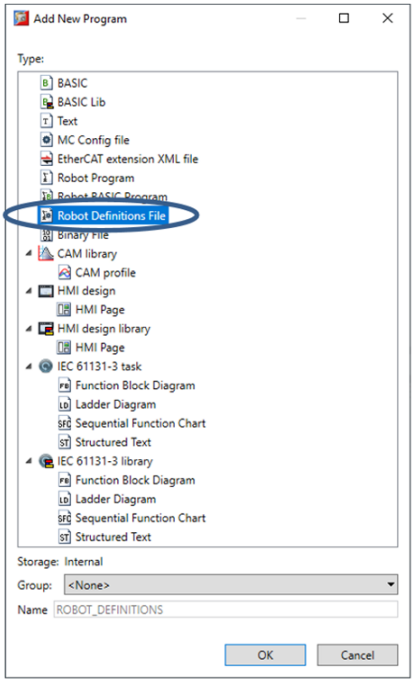


图 5-2 选择并创建 ROBOT_DEFINITIONS 程序

现在已经创建了 ROBOT_DEFINITONS 程序。

5.1.1.2 机器人模型参数编辑

要编辑 ROBOT_DEFINITIONS，用户可以直接编写/修改程序(非向导编辑器)，也可以使用 Robot Setup 向导图形界面设置机器人。

默认情况下，程序将以向导模式打开，操作方式如图 5-3:

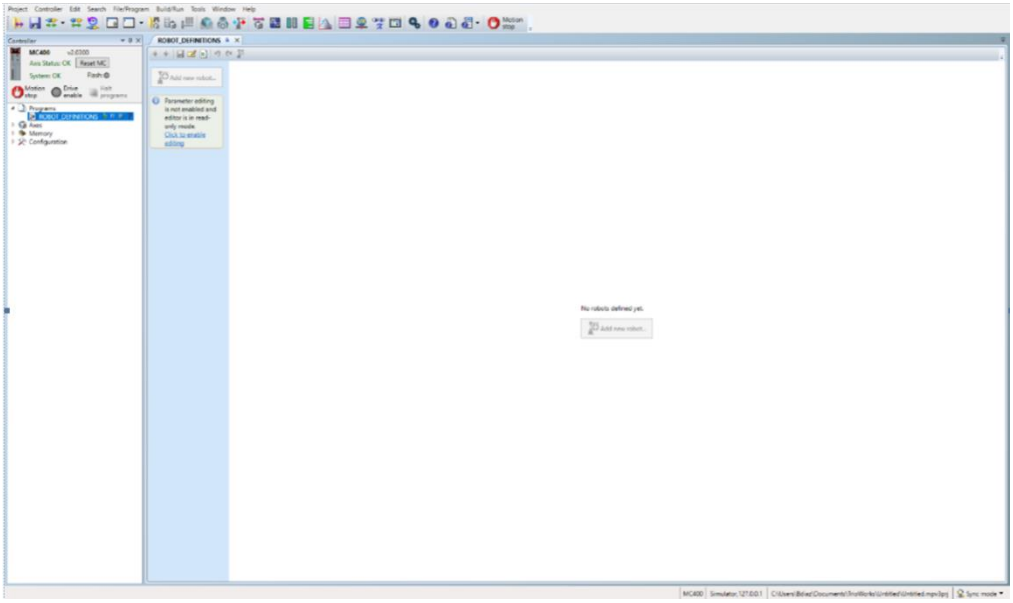


图 5-3 以向导模式打开的 ROBOT_DEFINITIONS 文件

如果 ROBOT_DEFINITIONS 被关闭，则通过双击控制器项目树中的 ROBOT_DEFINITIONS 程序重新打开它。

也可以通过图 5-4 中的图标在向导编辑器和文本编辑器之间切换。

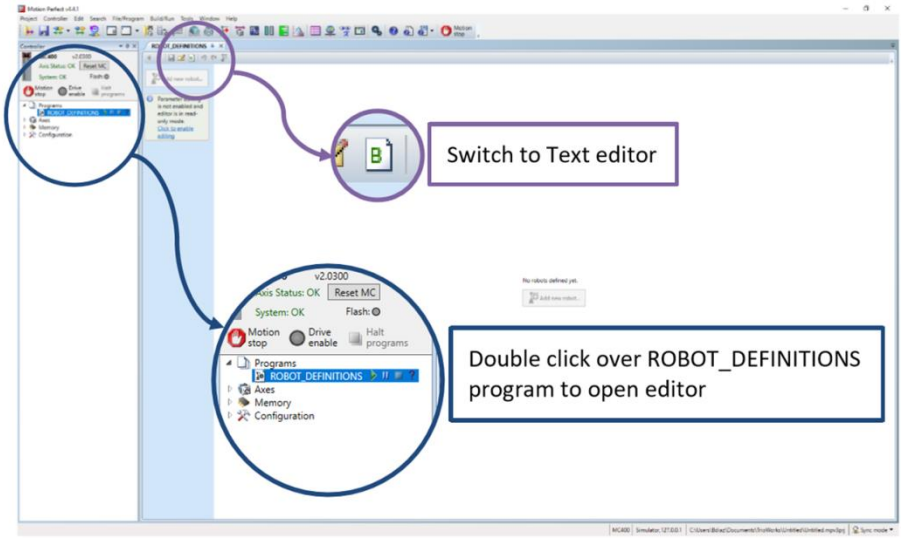


图 5-4 启动 ROBOT_DEFINITIONS 文件编辑

5.1.1.3 使用机器人向导设置机器人本体参数

单击“单击以启用编辑”。“添加新机器人”按钮将被启用。

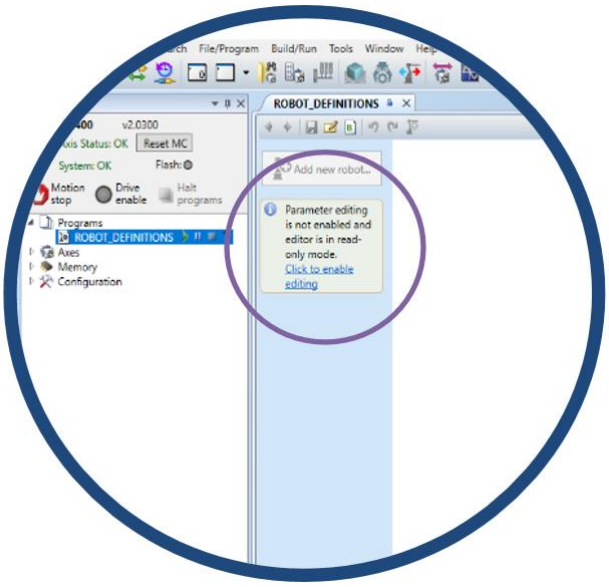


图 5-5 启动添加新机器人

点击“添加新机器人”按钮，选择要添加的机器人类型。有三种机器人模型供选择，分别是 6 轴多关节机器人，4 轴 SCARA 机器人和 4 轴码垛机器人。

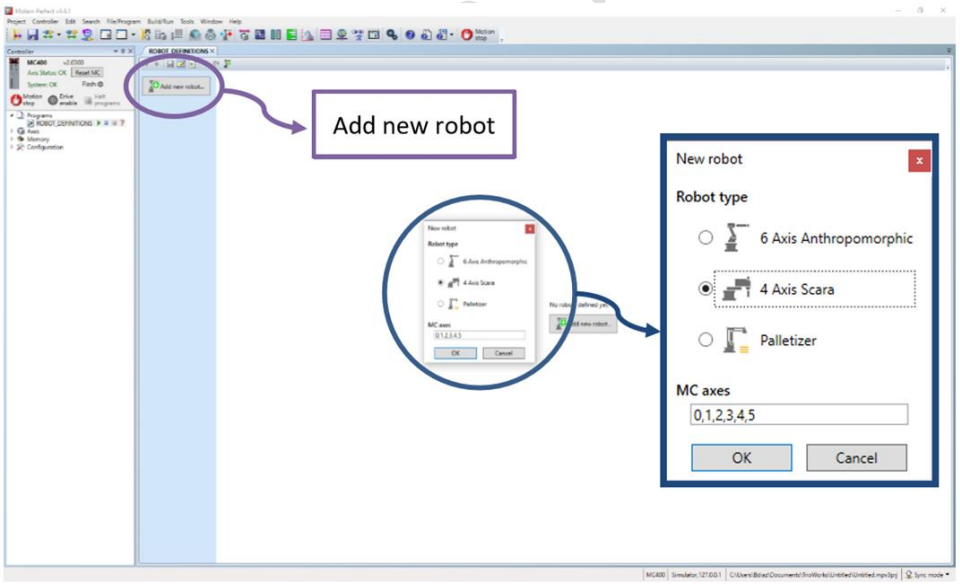


图 5-6 添加新机器人

“MC axes”文本框允许用户定义机器人轴号。机器人被选中，点击‘OK’。按确定键后将出现物理参数设置界面。

注意，这个界面会根据所选机器人的类型而有所不同。这里以 SCARA 机器人为例。

5.1.1.3.1 机器人本体参数设置

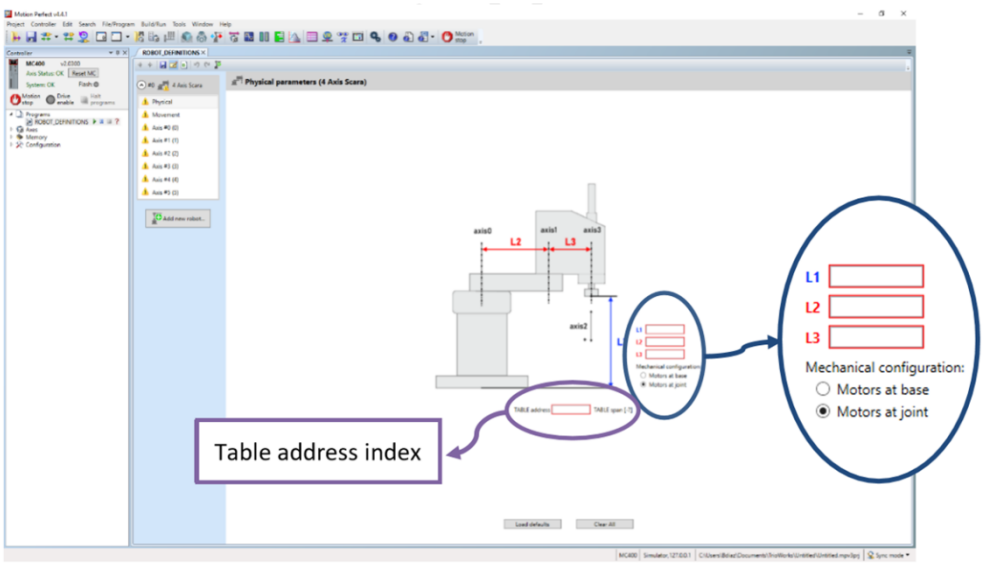


图 5-7 机器人本体参数设置

输入所需的连杆长度。

L1-Z 轴行程（mm）

L2-大臂长度（mm）

L3-小臂长度（mm）

SCARA 机器人电机都是安装在关节处，选择“Motors at joint”。

RPS 使用 TABLE 内存的一部分来存储机器人参数。“Table address”参数是存储机器人数据的起始地址。向导从“Table 地址”开始需要 10 个 Table 内存索引。这些索引将被 RPS 锁定，并且不能被任何程序写入。

以 ER6-600 SCARA 为例，参数设置如图 5-8：

ESTUN & Trio

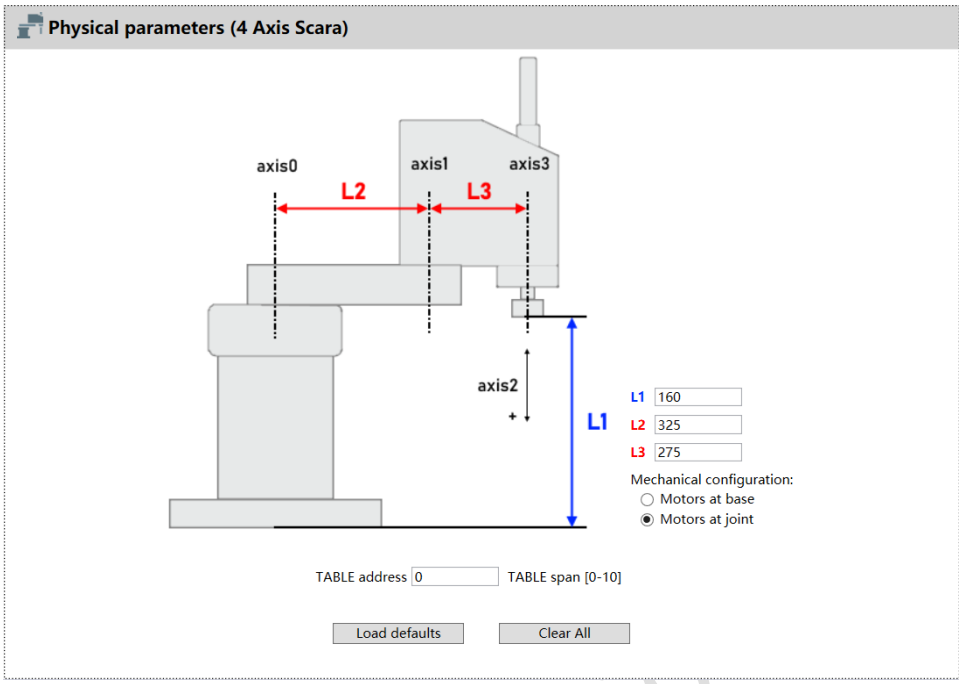


图 5-8 机器人本体参数设置示例

5.1.1.3.2 运动参数设置

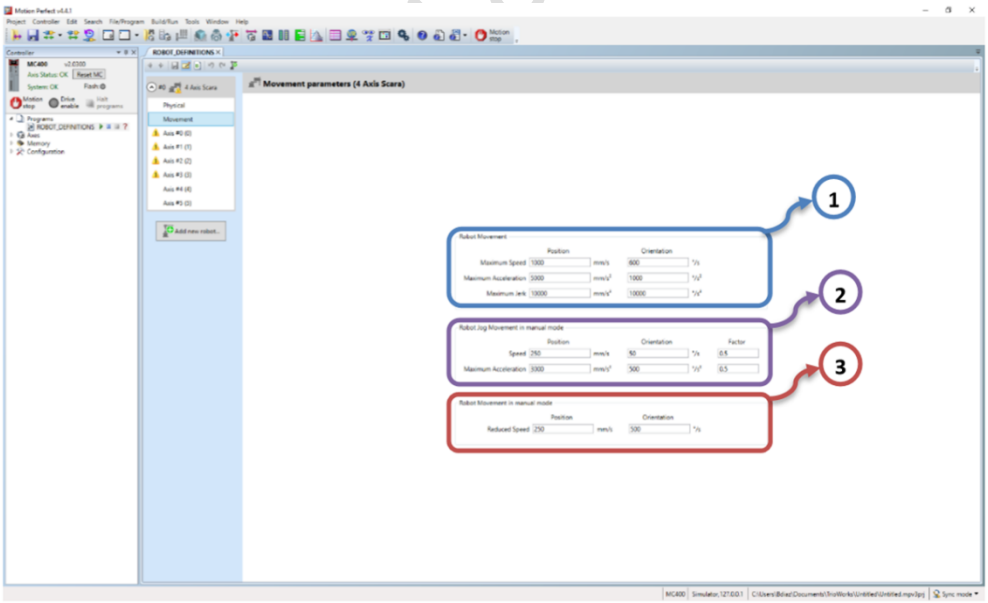


图 5-9 机器人运动参数设置

当页面上所有的参数都设置好了，点击下一页的“移动”。

‘Robot Movement’ ① 面板设置的是机器人末端执行器世界坐标系下直线运动的最大速度、加速度和加加速度。“位置”指的是笛卡尔 X、Y 和 Z 的运动。单位是毫米/秒。“方向”是指欧拉 U、V 和 W 运动。单位是度数/秒。程序设定的任何移动速度高于此速度将被限制

为最大速度。

例如，MOVEL GTA(0) S:=900。

最大速度设置 500 毫米/秒，在程序中，移动速度要求为 900 毫米/秒，程序执行时会按照 500 毫米/秒速度运行。

‘Robot jog movements in manual mode’ ② 面板设置的是机器人末端执行器的 JOG 参数。“Factor”指的是所输入的最大 JOG 速度和加速度的百分比。速度和加速因子在 JOG 时用来设定不同的速度。这个值在 0 和 1 之间。

‘Robot movement in manual mode’ ③ 是指手动模式下的速度。机器人在手动模式下运行时不会超过这个速度。手动模式下程序设定的任何移动速度高于此速度将被限制为手动速度。

例如，MOVEL GTA(0) S:=900。

手动速度设置 250 毫米/秒，在程序中，移动速度要求为 900 毫米/秒，但在手动模式下将以 250 毫米/秒的速度执行。

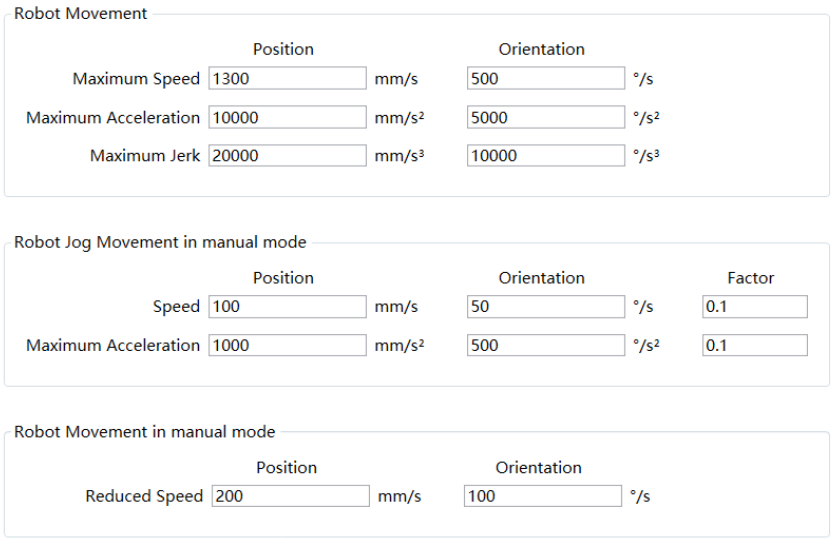


图 5-10 机器人设置参数示例

5.1.1.3.3 轴参数设置

机器人运动参数设置好后，点击“下一页”，轴(0):

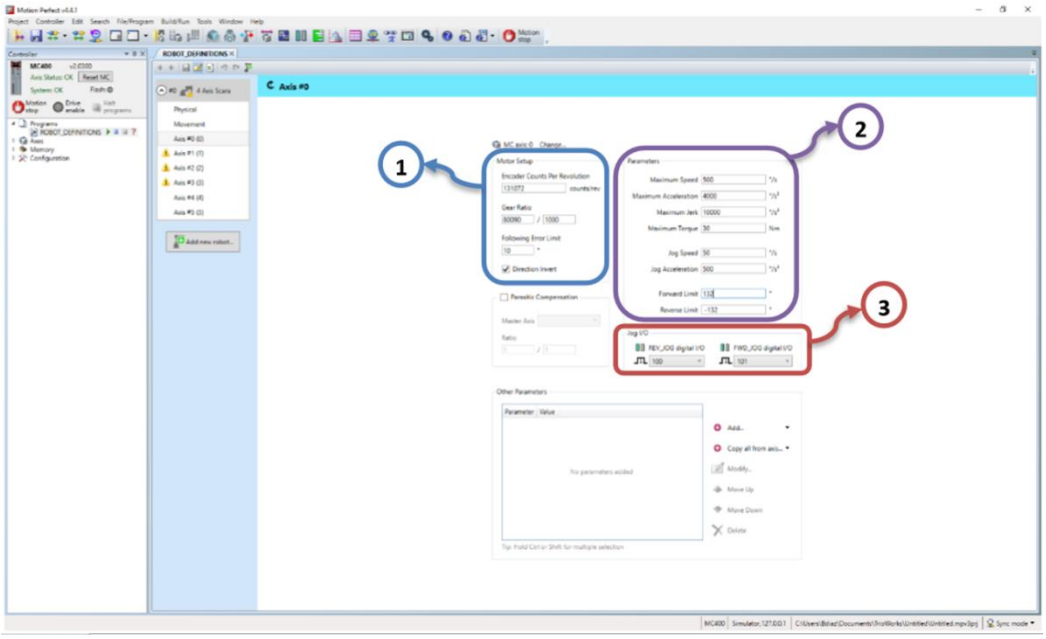


图 5-11 机器人轴参数设置

图标表示该轴为旋转轴。图标表示该轴为移动/线性轴。

轴的单位对应于轴所设的 AXIS_UNITS。通常，旋转轴单位为度数，移动/线性轴单位为毫米。

‘Motor setup’ ①用来设置每个轴的 AXIS_UNITS。需要填入编码器的分辨率，齿轮传动比和跟随误差。这些信息在机器人的出厂参数中获取。如果轴的旋转方向与物理设置中提到的旋转方向不一致，点击 “‘Direction Invert’”按钮可以改变该轴的旋转方向。

“Encoder Counts Per Revolution”是设置轴的编码器分辨率。

“Gear Ratio”是设置轴的减速比。

“Following Error Limit”是轴的跟随误差。

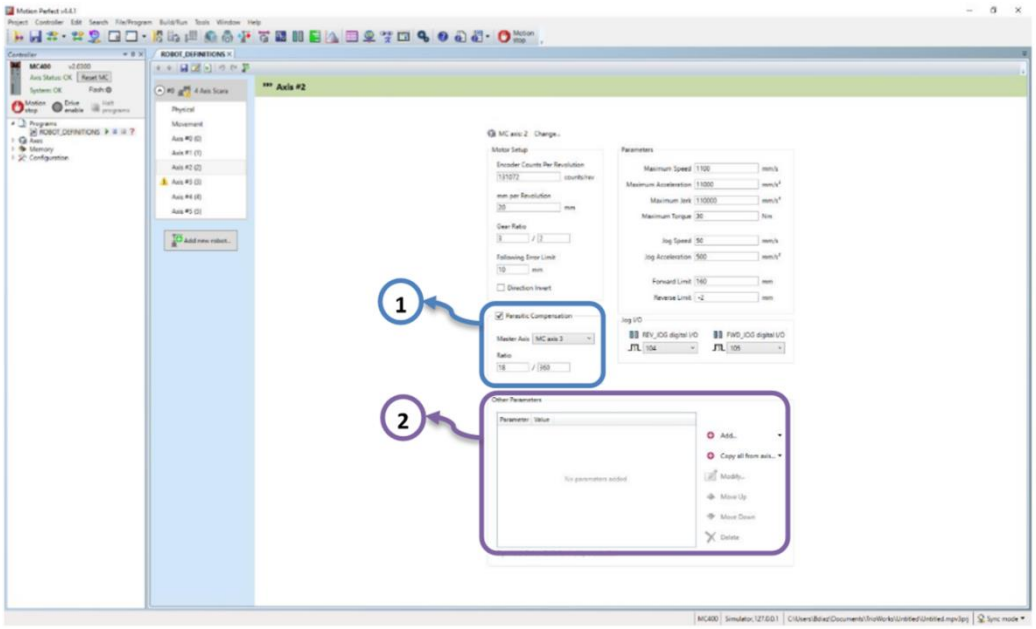
“Direction Invert”是设置轴的方向。

‘Parameters’ ②用来设置每个轴的最大的速度(以度数/秒计算)、加速度、关节速度和扭矩。该信息在机器人产品手册中可以找到。还需要为每个轴设置正向和反向的软件限制。

最大值参数用于自动模式下的移动指令。运行轴时使用 Jog 参数。

‘Jog IO’ ③可以选择 Jog 输入。这些输入必须是虚拟的，所以请确保选择正确。此设置区域还包含一个按钮，用于反转所选输入的有效信号电平。默认情况下，正向和反向 Jog 输入都是低电平激活。

ESTUN & Trio

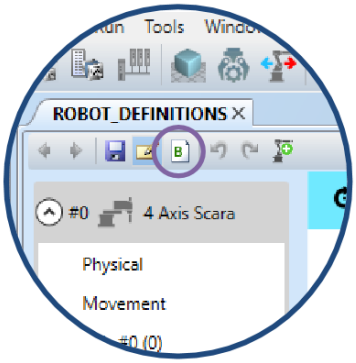


如果两轴之间存在耦合运动，需要补偿。

'Parasitic Compensation' ①选项来激活其设置。输入主轴/驱动轴和比率。例如，轴 3 与轴 2 是机械连接的。当轴 3 旋转 1 圈时，轴 2 移动 20mm。使用上述信息，MC 主轴为 3，比率为 20/360。

'Other Parameters' ②面板允许用户向设置文件添加额外的轴参数。

当所有的机器人轴都设置好了，点击  查看生成的 ROBOT_DEFINITIONS 文件。



在设置了机器人之后，必须执行 ROBOT_DEFINITIONS 文件，这样控制器就可以将选择的轴作为机器人分配，RPS 系统就可以启动并运行了。

例如 ER6-600 SCARA 各轴参数设置：

ESTUN & Trio

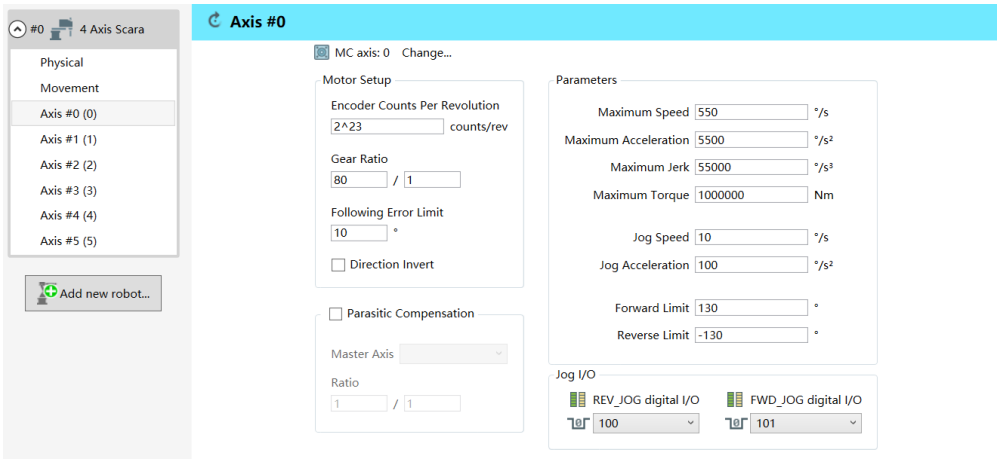


图 5-12 ER6-600 SCARA 轴#0 的设置示例

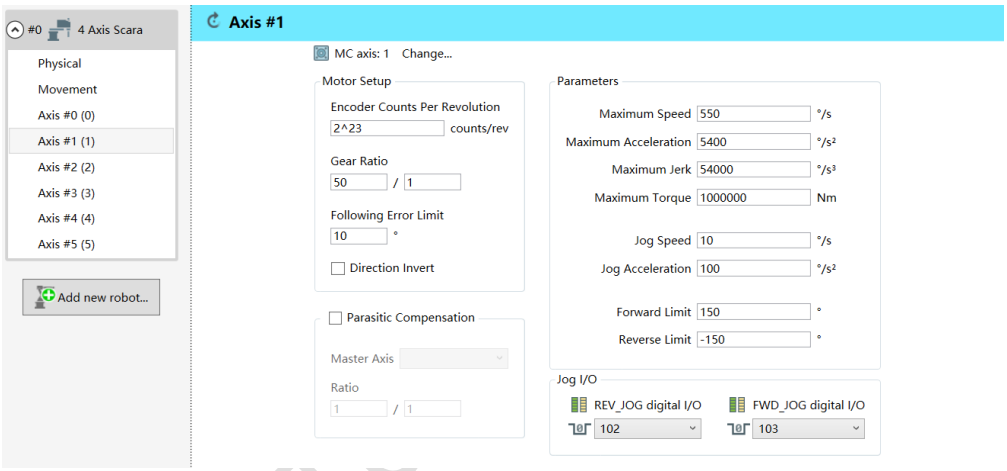


图 5-13 ER6-600 SCARA 轴#1 的设置示例

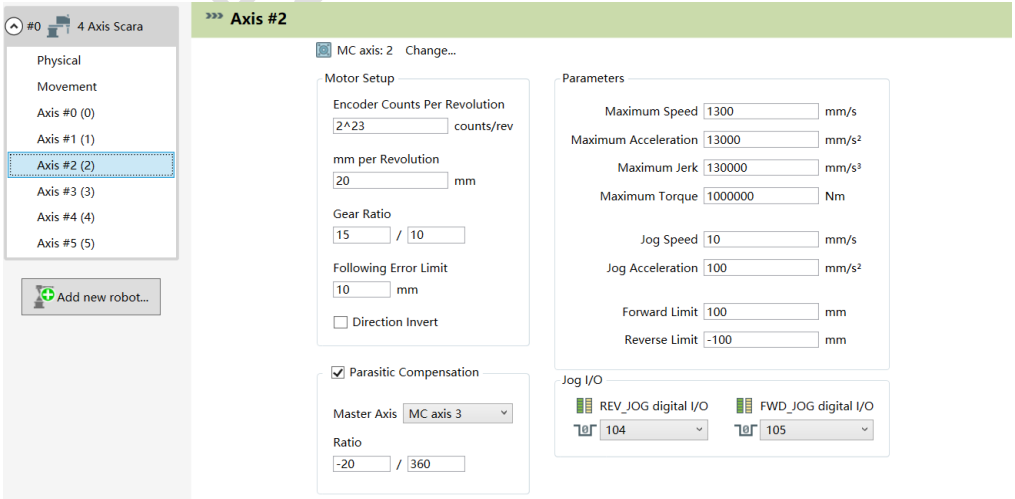


图 5-14 ER6-600 SCARA 轴#2 的设置示例

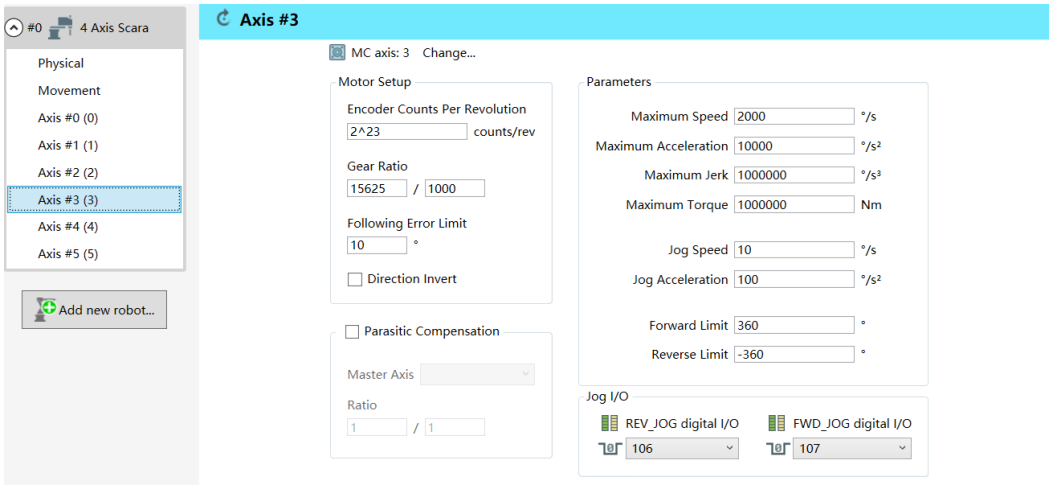


图 5-15 ER6-600 SCARA 轴#4 的设置示例

5.1.2 ROBOT_DEFINITIONS 文件

上述的 ROBOT_DEFINITIONS 文件所做的设置，包括如下参数，这些参数和意义说明如下：

TABLE_PROTECT：这个函数为选定的 TABLE 范围打开/关闭保护
KINEMATIC_GROUP(0,-1)：删除运动学组
TABLE(0,160,325,275,(2^23)*(80/10)/(2*PI),(2^23)*(50/1)/(2*PI))
TABLE(5,(2^23)*(1500/1000)/20,(2^23)*(15625/1000)/(2*PI),1000,1000,0,0)
：存储每弧度的机器人连杆长度和编码器分辨率
WORLD_POS_REDUCED_SPEED：手动模式下允许的最大线性速度
WORLD_POS_MAX_SPEED：最大允许直线速度
WORLD_POS_MAX_ACC：最大允许直线加速度
WORLD_POS_MAX_JERK：最大允许直线加加速
WORLD_ORI_REDUCED_SPEED :手动模式下允许的最大旋转速度
WORLD_ORI_MAX_SPEED:最大允许旋转速度
WORLD_ORI_MAX_ACC:最大允许旋转加速度
WORLD_ORI_MAX_JERK:最大允许旋转加加速度
WORLD_POS_JOG_SPEED:最大允许的线性 JOG 速度
WORLD_POS_JOG_ACC:最大允许的线性 JOG 加速度
WORLD_ORI_JOG_SPEED:最大允许的 JOG 旋转加速
WORLD_ORI_JOG_ACC :最大允许 JOG 旋转加速
JOG_SPEED_FACTOR:从 0 到最大控制 JOG 速度的因子
JOG_ACC_FACTOR:控制 JOG 加速从 0 到最大的因子

'开始轴 0 的配置
BASE(0)
AXIS_UNITS:所选轴的每度对应编码器值
AXIS_MAX_SPEED:允许的最大轴速度
AXIS_MAX_ACC:允许的最大轴加速度

ESTUN & Trio

AXIS_MAX_JERK:允许的最大轴加加速
AXIS_MAX_JOG_SPEED :允许的最大 JOG 速度
AXIS_MAX_JOG_ACC:允许的最大 JOG 加速度
AXIS_MAX_TORQUE:最大允许扭矩-未使用
AXIS_FS_LIMIT:正向软件限制
AXIS_RS_LIMIT:反向软件限制
AXIS_MODE.16:轴方向取反设置
FE_LIMIT:跟随误差限制
DRIVE_FE_LIMIT:驱动器跟随误差限制
VP_MODE:速度配置文件模式
INVERT_IN(111,ON)
OP(111,OFF)
FWD_JOG:正向 JOG 输入
INVERT_IN(110,ON)
OP(110,OFF)
REV_JOG:反向 JOG 输入
其他轴同轴 0

5.2 建立 3D 可视化

使机器人可视化，可点击图标打开 3D 可视化工具。将打开图 5-16 窗口。

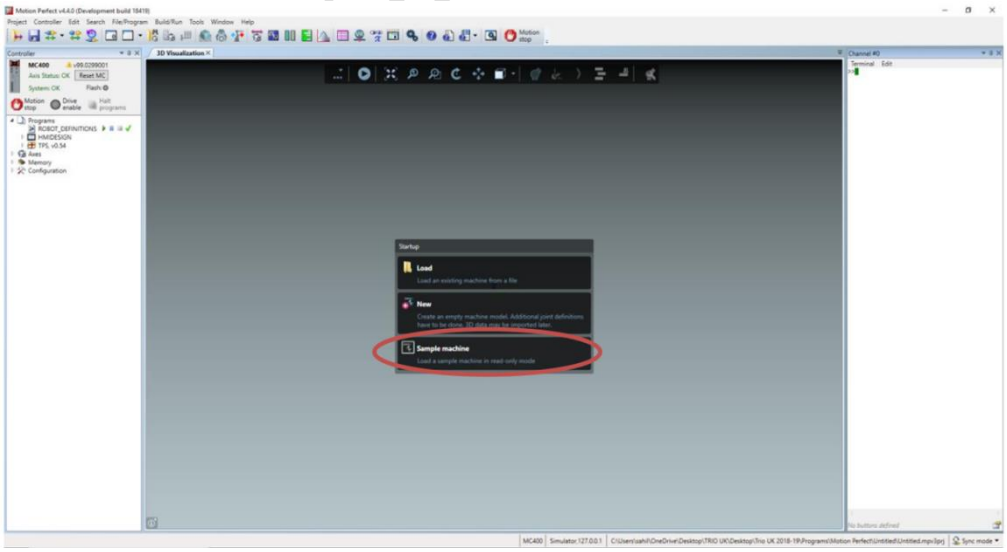
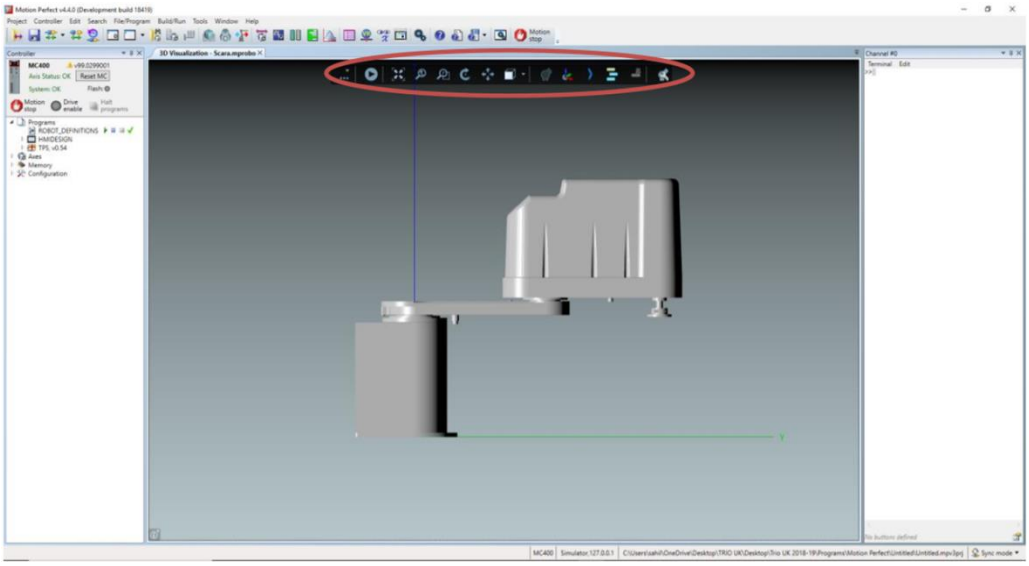
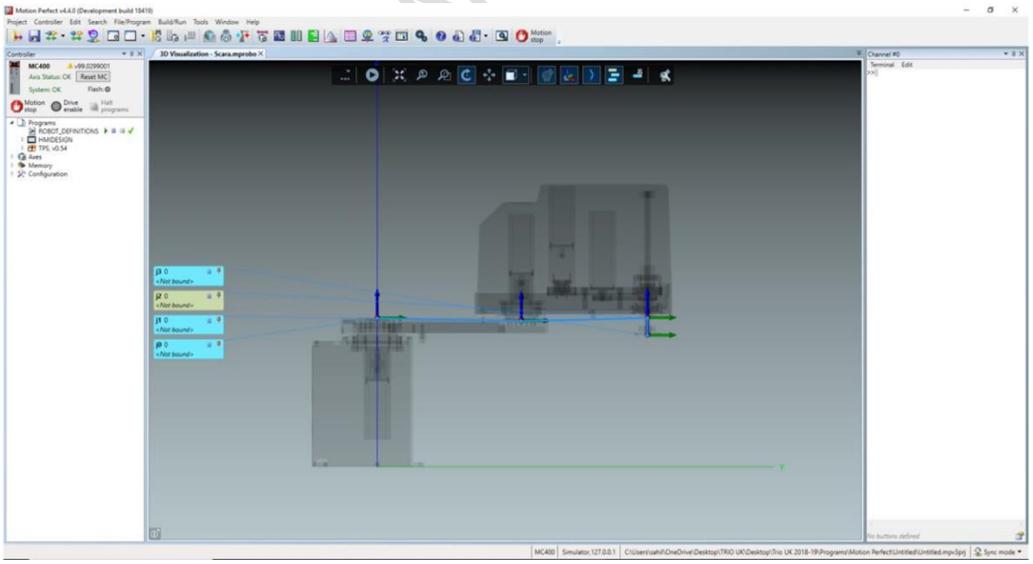


图 5-16 3D 可视化窗口

点击‘Sample Machine’，选择机器人。在本例中选择了一个 SCARA 机器人。



按一下  ，用户还可以点击按钮  对齐标签，用户也可以点击按钮  看关节的位置，用户还可以点击按钮  来查看模型的 x 光视图。使用鼠标导航 3D 模型，右键点击将平移模型，左键点击将旋转模型，滚动轮将放大/缩小模型。用户还可以点击按钮  查看关节层次结构和关节的标签分配。



要将关节指定为轴，单击关节标签区域。选择关节的轴。

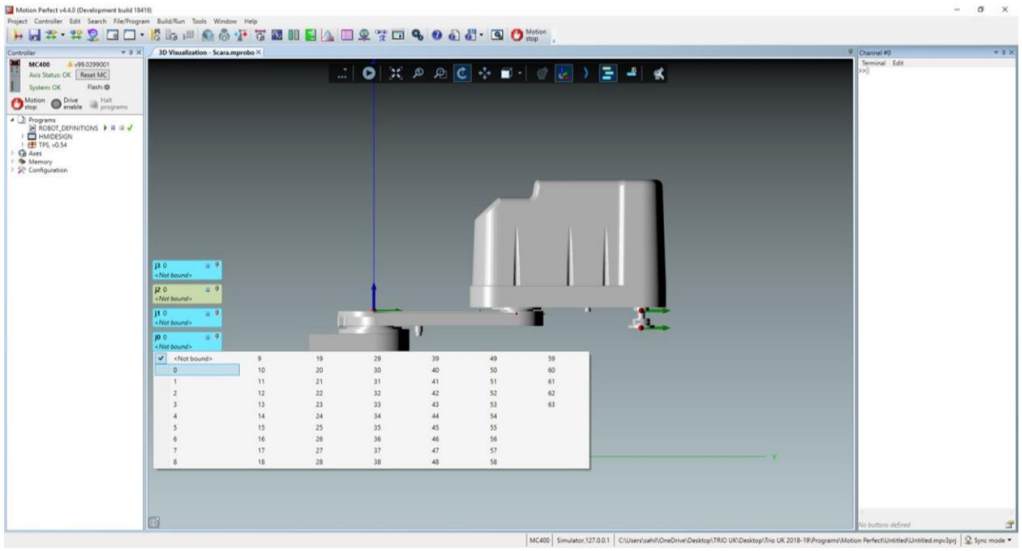


图 5-17 关联 3D 可视化中的机器人关节

3D 工具现在已经设置好了。点击“开始模拟”按钮。

1. 如何点动机器人

Jog 工具允许用户在不同模式下对机器人进行 Jog，以及选择不同的现有工具、对象框架和机器人框架。

要 Jog 机器人，点击打开“Jog 机器人”工具 图标。

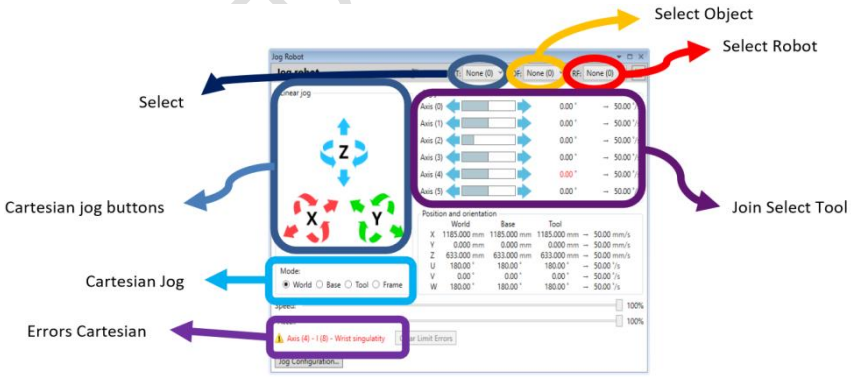


图 5-18 点动机器人页面

5.3 RPS 编程介绍

5.3.1 RPS 语言扩展

5.3.1.1 Kinematic group

KINEMATIC_GROUP 是激活 RPS 并将所需数据连接到指定机器人的命令，必须执行此命令才能设置 RPS，组中的轴数必须与运动学使用的轴数相匹配，这些轴必须是升序排列。

5.3.1.2 语法

KINEMATIC_GROUP(index, table offset, kinematic number, axis0 - axisn)

关键字	描述
Index	从 0 到 7，意味着在同一个控制器中最多可以设置 8 个机器人
Table offset	机器人参数设置首地址
Kinematic number	机器人类型
Axis0-axisN	机器人的轴，轴号递增

5.3.2 Behaviour

通过执行 KINEMATIC_GROUP(index)可以知道是否设置了 RPS 以及所有与 RPS 相关的信息。格式如下:

Group index [table_offset] [kinematic number]: axes

TO={index [name] : TOOL_OFFSET parameters}

RF={index [name] : ROBOT_FRAME parameters}

OB={index [name] : OBJECT_FRAME parameters}

CO{active collision objects}

激活的 COLLISION_OBJECTS 值是二进制的，显示为十进制格式。

例子:

KINEMATIC_GROUP(0)

Terminal: 0 [0] [19] : 10, 11, 12, 13, 14, 15 TO={0[TO_default]:0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000} RF={0[RF_default]:0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000} OF={0[OF_default]:0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000} CO{3}

使用 KINEMATIC_GROUP(index，-1)可以很容易地禁用 RPS，组中的所有轴将被释放。

ESTUN & Trio

5.3.3 机器人选择和程序类型

5.3.3.1 机器人选择

RPS 能够在同一系统中配置多达 8 个机器人。为了让程序知道什么机器人可以调用所有后续的运动命令和其他参数的读/写，首先必须选择一个机器人。使用 ROBOT(ROBOT ID)命令。一旦机器人被选中，它在这个过程就会执行动作，直到所产生的动作完成，程序停止。因此，如果一个程序停止(进程结束)，那么机器人将不会被释放，直到缓冲区空闲。通过读取 Robot (Robot ID)命令可以检查某个特定机器人所在的进程，如下所示:

```
my_robot = ROBOT(0)
```

使用-1 索引从过程中取消选择机器人，如下所示:

```
ROBOT(-1)
```

只有当 ROBOT(-1)与所选机器人执行同一进程时，才会使机器人从所选进程中脱离出来。

STOP_ROBOT(机器人 ID)将取消移动、清除缓冲区并停止属于所选机器人索引的进程。

这个命令可以从其他进程调用，使从主程序控制一个或一组机器人变得更容易。

DISABLE_ROBOT(boolean flag) :

此命令将禁止任何机器人程序运行。

如果任何程序已经在进程上运行，进程将被停止，缓冲区将被清除。

DISABLE_ROBOT(TRUE)将激活该命令，而 DISABLE_ROBOT(false)将禁用该命令。

5.3.3.2 程序类型

有两种不同的机器人程序类型: Robot Programs 和 Basic Robot Programs。所有的机器人程序类型都可以通过 Motion Perfect 完成编辑。

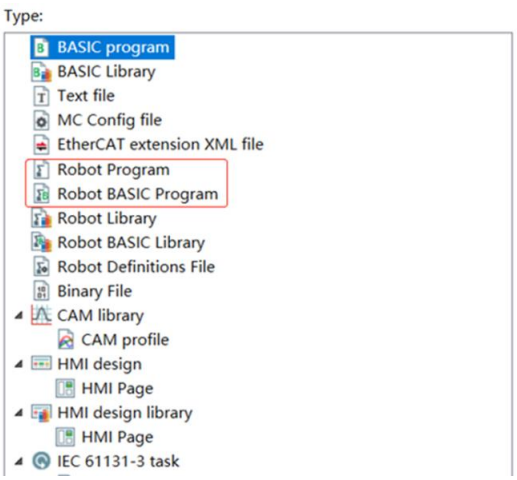


图 5-19 机器人程序类型

5.3.3.2.1 Robot Program

Robot program 只包含 TPS 能够处理的指令。所有的 MOVE 指令，结构(While, If, Repeat, For 结构)，函数调用，标签等机器人指令。

这种类型的程序将有一个‘.ROB’的文件扩展名。

5.3.3.2.2 Robot Basic Program

Robot Basic Programs 可以包含所有现有的 Trio 指令(与 BASIC 程序相同)。此外，TPS 可以编辑 Robot Basic Program，但仅对于用户级别 1。用户可以编写复杂的程序作为 Robot Basic Program，以更好地控制自己的应用程序。重要的是使用这种类型的程序而不是一个 BASIC program，因为控制器可以在紧急情况下关闭这种类型的程序。 这种类型的程序将有‘.RBS’ 的文件扩展名

5.3.3.3 Robot Library

Robot Library 包含机器人应用程序的用户功能。TPS 可以读取这个文件并列出示教器中的所有功能。该类型文件中允许的命令与基本库中允许的语法相同。 这种类型的库将有一个‘.ROBLIB’文件扩展名。

5.3.3.4 Program Pointer

一个 Robot Program 或者 Robot Basic Program 可以以两种方式运行。正常运行操作将正常运行所选程序。运行中的程序可以随时暂停或停止。此外，用户可以选择一行并从该行开始运行程序。如果有多个机器人，当前活动的机器人将被选中，所有的移动命令将在选中的机器人上运行。这可以通过从程序或 UI (MP 或 TPS)中选择另一个机器人来改变。这在测试时提供了更多的灵活性。 此功能只适用于 TPS，并且只能在示教器上使用。

5.3.4 Frames and Tools

Frames 和 Tools 是一个 4x4 正交矩阵，其位置和方向是完全确定的。

它们被定义为三轴平移和旋转。旋转应用欧拉 ZYX 约定。这意味着首先应用 Z 旋转，然后应用 Y 在新的坐标系上，最后应用 X。坐标系是用“右手规则”定义，原点的旋转是用“右手旋转”定义。

所选 Frames 和 Tools 将载入运动，这意味着您可以在不同的 Frames 和 Tools 上缓冲一系列的动作，激活的 Frames 和 Tools 与 MTYPE 中的移动相关联，如果 KINEMATIC_GROUP 是空闲的，那么激活的 Frames 或 Tools 就是选中的 Frames 或 Tools。

该系统由以下 Frame 组成:

- WORLD_FRAME: 表示唯一固定系统的世界坐标(0,0,0 位置和用于定位的单位矩阵)。它指定了移动观察者和被观察对象之间的关系。它不能被覆盖，但是 RPS 的坐标系统可以通过使用 OBJECT_FRAME 来修改。

- OBJECT_FRAME: 在一个对象上，它可以改变程序中坐标系统。一旦应用了

ESTUN & Trio

OBJECT_FRAME，机器人和所有的点都将与它相关。

• ROBOT_FRAME: 在机器人基础上，它可以改变与激活的坐标系统(默认为 WORLD_FRAME 或 OBJECT_FRAME)相关的机器人的位置和方向。无论机器人处于什么位置，TCP(工具中心点)总是与 WORLD_FRAME 或 OBJECT_FRAME 相关。

• TOOL_OFFSET: 机器人的 TCP 和末端执行器之间的偏移量。它设置了工具从末端执行器到 TCP 的距离和方向。如果没有应用 TOOL_OFFSET，则 TCP 位于机器人的末端执行器上。

• TOOL_COLLISION:构建一个围绕工具的碰撞区域。有关更多信息，请参阅碰撞检测部分。

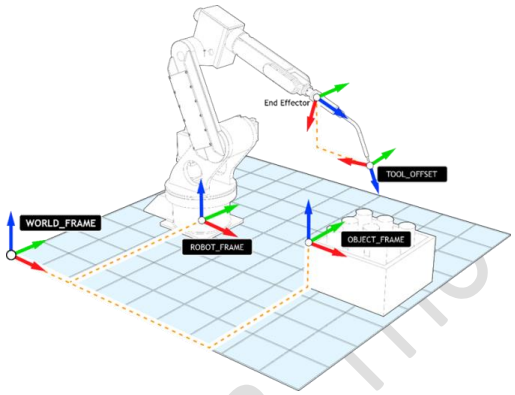


图 5-20 Frames 和 Tools

5.3.4.1 语法

OBJECT_FRAME (identity, name, x offset, y offset, z offset, x rotation, y rotation, z rotation)

X 偏移量，y 偏移量和 z 偏移量必须用 mm 来指定。

X 旋转，y 旋转和 z 旋转必须用角度来表示。

例子： OBJECT_FRAME(1,"camera",200,0,400,0,180,0)

TOOL_COLLISION(identity, name, x centre, y centre, z centre, half distance x, half distance y, half distance z)以刀具碰撞的尺寸作为刀具所能达到的最大尺寸。例如，如果工具是夹持器，使用夹持器打开的尺寸如下图所示。

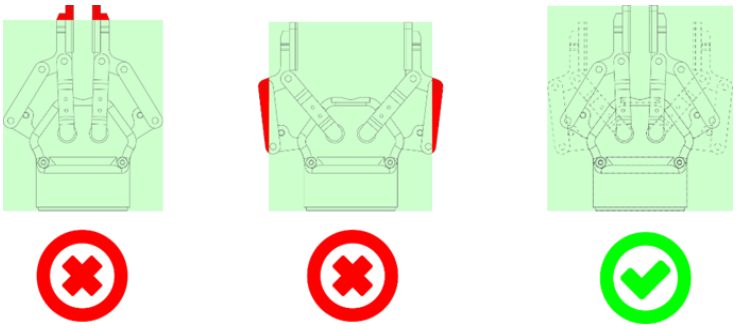


图 5-21 夹持器碰撞范围

5.3.4.2 Behaviour

从索引 1 到 31，每个 Frames 或 Tools 偏移最多有 32 个。索引 0 是值为“TO_default”的默认索引:0.00000、0.00000、0.00000、0.00000、0.00000、0.00000、0.00000 的默认索引，不能修改或删除。在使用它们之前，必须使用相应的命令和所需的值创建 Frames 和 Tools 偏移量。

例子： TOOL_OFFSET(1,"Gripper",100,50,0,0,0,0)

参数名称是可选的，如果不使用，系统会将其保留为空。如果一个条目与不同的索引创建一个现有条目的名称,系统会显示运行时错误“重复的标识符”。

可以在任何时候使用新值修改条目(激活的条目除外)。

例子：

Terminal: 0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 1 [Gripper] : 100.00000, 50.00000, 0.00000, 0.00000, 0.00000, 0.00000

TOOL_OFFSET(1,"Torch",200,0,50,0,45,0)

Terminal: 0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000

可以使用参数-2 和 index 删除条目。

Terminal: 0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000 2 [Gripper] : 150.00000, 50.00000, 0.00000, 0.00000, 0.00000, 0.00000

5 [Pencil] : 200.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 16 [Needle] : 300.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000

TOOL_OFFSET(-2,2)

Terminal: 0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000 5 [Pencil] : 200.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 16 [Needle] : 300.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000

仅使用参数-2 删除所有条目。如果一个条目被选中它不能被修改或删除,显示运行时错误”工具无法修改或删除而选定的。

可以使用参数-1 请求现有条目。它返回现有的值加上条目 0(默认值)。如果参数是(-1,index)，则只返回所请求的索引。

例子：

OBJECT_FRAME(-1)

Terminal: 0 [OF_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000 1 [of2] : 517.35967, 0.41128, 950.75573, 28.23244, -177.09862, -22.97674 2 [of3] : 633.29181, 188.67548,

ESTUN & Trio

799.24254, 0.00000, 0.00000, 0.00000 3 [of4] : 633.29181, 188.67548, 799.24254, 0.00000, 90.00000, 0.00000

OBJECT_FRAME(-1,2)

Terminal: 2 [of3] : 633.29181, 188.67548, 799.24254, 0.00000, 0.00000, 0.00000

如果您希望检查哪个 OBJECT_FRAME、ROBOT_FRAME 或 TOOL_OFFSET 是激活的，您可以使用 KINEMATIC_GROUP(group)将详细信息打印到终端。

为了使用一个 Frames 和 Tools，它必须被选择为一个特定机器人的独立命令或作为一个嵌入参数(在移动部分解释)。当它被选中，系统将使用它作为激活的。在选择一个 Frame 或 Tool 后执行的运动将与激活的相关，直到它完成。完成运动后，可以选择一个新的 Frame 或 Tool，但运动结束，它将是激活的。使用这种方法，可以对框架或工具进行“动态”选择，并给予程序员更大的灵活性。

要选择一个 Frame 或 Tool，只需用所需的索引或名称作为参数调用该命令。

例子：

OBJECT_FRAME(”of3”)

通过不带参数分配命令 TOOL_OFFSET、OBJECT_FRAME 或 ROBOT_FRAME 的返回值，可以请求哪个 Tool 或 Frame 处于激活状态。

例子：

i = OBJECT_FRAME

5.3.5 TARGET data type

Target 是一种数据类型，用来存储位置和方向，通常用于移动机器人的移动命令中。Target 可以是局部的，也可以是全局的，存储笛卡尔坐标或关节位置。

Targets 在使用时与激活的坐标系统相关。如果 Targets 存储的坐标系统与激活的坐标系统不同，机器人将不会移动到预期的目标。确保所选框架或工具偏移量与当前存储 Targets 的相同。

5.3.5.1 笛卡尔坐标系

Target 点可以被声明在笛卡尔的工作空间下， GTA 为全局使用和 Target 为局部使用。

5.3.5.1.1 TARGET

TARGET 是一个存储笛卡尔点的局部变量。本地变量在声明它们的程序停止运行时被删除。

```
DIM point1 AS TARGET'定义局部世界坐标点位 point1
point1=100,500,-20,180,0,0,70

MOVEL point1'SCARA 直线运动去 point1 点
```

ESTUN & Trio

```
MOVEJ point1'SCARA 关节运动去 point1 点
```

5.3.5.1.2 GTA

GTA 数组是全局数据类型，存储笛卡尔目标点(位置和方向)，可被所有程序访问。GTAs 保存在闪存中，在重新启动控制器时不会丢失(类似于 VRs)。

以下组合是正确的，并将按照描述的方式运行:

MOVEL GTA(0)

这条指令将使机器人的末端执行器沿直线移动到目标点。

MOVEJ GTA(0)

该指令将机器人的末端执行器非线性地移动到目标点。换句话说，机器人会在关节空间线性移动。

MOVEC GTA(0) GTA(1)

这条指令将使机器人的末端执行器作圆周运动。

所有其他组合都将触发一个错误。

5.3.5.2 关节坐标系

5.3.5.2.1 TARGETJ

TARGETJ 是一个存储关节坐标系点的局部变量。本地变量在声明它们的程序停止运行时被删除。

```
DIM point2 AS TARGETJ'定义局部关节坐标点位 point2
point2=50,100,-20,70,0,0,0
MOVEL point2'SCARA 直线去 point2 点
MOVEJ point2'SCARA 关节运动去 point2 点
```

5.3.5.2.2 GTAJ

GTAJ 数组是一个全局变量，存储关节点，可被所有程序访问。GTAJs 保存在闪存中，在重新启动控制器时不会丢失(类似于 VRs)。在关节模式下，GTAJs 的数据格式如下:

GTAJ('ID number') = θ1, θ2, θ3, θ4, θ5, θ6, "点名为字符串"

下列组合是正确的，并将按照所描述的方式运行：

MOVEL GTAJ(0)

控制器将使用关节坐标计算线性目标点。然后机器人的末端执行器将直线移动到目标点。

如果机器人处于不同的方向(与所教的关节位置相比)，它不会改变其配置。机器人将移动到当前配置中计算的笛卡尔坐标位置。

ESTUN & Trio

MOVEJ GTAJ(0)

机器人将以关节模式移动到关节目标点。它会在关节空间中线性移动。

所有其他组合都会产生一个错误。

GTA 和 GTAJ 共享相同的工作空间，这意味着最多只能有 1000 个全局目标(笛卡尔坐标和关节组合)。用户不能有单独的索引，例如:

GTA(1) = X1, Y1, Z1, U1, V1, W1, “pt1”

GTAJ(2) = t1, t2, t3, t4, t5, t6, “pt2”

在本例中，GTAJ 不能有索引 1，GTA 不能有索引 2。

5.3.5.2.3 语法

- Global target:

GTA(index number) = float x, float y, float z, float u, float v, float w, string name

GTAJ(index numb) = float θ1, float θ2, float θ3, float θ4, float θ5, float θ6, string name

例子：

GTA(10) = 100,0,50,180,0,180,"Cartesian_point1"

GTAJ(20) = -10,50,10,0,15,0,"Joint_point1"

名称是可选的。

- Local target:

name = float x, float y, float z, float u, float v, float w

例子: my_local_target = 250,0,150,180,0,180

5.3.5.2.4 Behaviour

局部 targets 必须在程序中声明，它们将被调用使用 DIM 声明。

例子: DIM point1 AS TARGET

通过在声明语句中定义局部目标的大小，可以拥有最多一维的局部目标数组。

示例: DIM point4 AS TARGET(10)

全局目标(GTA)是一个包含 1000 个值的固定数组，可以被所有程序中定义和调用。

默认情况下，在声明和执行之前，所有 GTAs 都是禁用的。即使它们已声明但未执行，也不会从任何命令中看到它们，从而产生运行时错误“目标点未激活”。

读和写访问位部分的 GTA 或本地目标"。在命令的末尾。它可以是 x y z u v w id 或激活。

关键字	描述

ESTUN & Trio

x, y, z, u, v, w	64 位浮点数
Id (GTA only)	字符串名
Active (GTA only)	布尔

示例： GTA(10).x = 250 my_variable = GTA(10).x

通过布尔属性“active”，它可以激活或停用和清除一个 GTA。如果激活属性被赋值为 false，那么 GTA 将被清除(所有值为 0 和空名称)并停用(从任何程序中都看不到)。

示例： GTA(10).active = FALSE

可以通过指定以逗号分隔的开始和结束索引来请求或分配 GTAs 的范围。

Example: GTA(10,13) = 250,0,150,180,0,180

PRINT GTA(10,13) Terminal:

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

GTA(10,12).y = 35 PRINT GTA(10,13) Terminal:

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

GTA(20,23) = GTA(10) PRINT GTA(20,23) Terminal:

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""

250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

如果指定了名称参数，则由于重复标识名称不兼容，无法进行范围分配。

分配之间的 GTA - GTA, GTA -本地目标和本地目标-本地目标是可能的。

示例： GTA(10).x = GTA(11).x my_target = GTA(10)

ESTUN & Trio

5.3.6 Robot Position

机器人的当前位置存储在两个系统变量 ROBOT_LPOS 和 ROBOT_JPOS 中。

5.3.6.1 ROBOT_LPOS

ROBOTL_POS 是数据型目标，存储末端执行器的笛卡尔坐标位置。可访问单个元素，例如 TARGET_LPOS。将返回末端执行器的当前 x 位置。这是一个只读系统变量。

可在程序中用于获取机器人的位置。

示例:

```
DIM my_variable AS TARGET DIM x_pos AS FLOAT
my_variable = ROBOT_LPOS
x_pos = my_variable.x
```

5.3.6.2 ROBOT_JPOS

ROBOTJ_POS 是一种数据型目标，用于存储末端执行器的关节位置。可以访问各个关节位置，例如 TARGET_JPOS。x 将返回机器人关节 1 的当前位置。这是一个只读系统变量。可以在程序中使用，获取机器人的轴位置。

示例:

```
DIM my_variable AS TARGETJ DIM j3_pos AS FLOAT
my_variable = ROBOT_JPOS
x_pos = my_variable.z。
```

5.3.7 机器人 JOG

Jog 是指手动将机器人的轴向正或负方向移动，或将 TCP 沿指定的笛卡尔方向移动的过程。

5.3.7.1 语法

JOG_OPERATION(mode)

默认 jog 模式为 0，表示禁用

5.3.7.2 Behaviour

若要 jog 机器人，需要为每个机器人轴设置 FWD_IN 和 REV_IN。

用于 FWD_IN 和 REV_IN 的输入是低电平激活

在设置了所有输入之后，必须使用 JOG_OPERATION 命令来设置 jog 模式。更改 jog 模式以激活进程，并使用 FWD_IN 和 REV_IN 输入来 jog 机器人。将 jog 模式设置回 0 以释放执行移动指令的 jog 进程。

ESTUN & TRIO

在轴处于空闲状态之前，不能执行 Jog。

JOG_OPERATION 命令必须在机器人的相同 base 组中执行。

Jog 机器人有五种模式:

- JOINT (mode = 1): 每个轴独立运动。

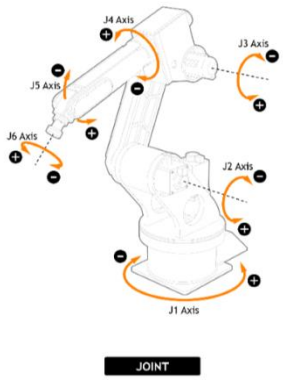


图 5-22 关节坐标系下 Jog 机器人的方向

- WORLD (mode = 2): 末端执行器沿着世界坐标系直线运动。方位使用外部旋转。
- BASE (mode = 3): 末端执行器沿基坐标系直线运动。方位使用外部旋转。

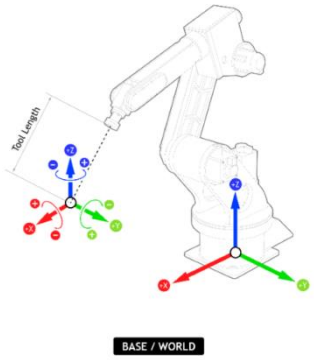


图 5-23 世界坐标系/基坐标系下 Jog 加入方向

- TOOL jog (mode = 4): 末端执行器沿工具坐标系直线运动。方向使用内在旋转。

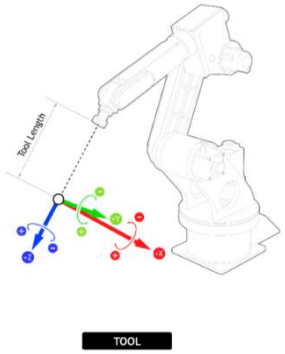


图 5-24 工具坐标系下 Jog 机器人

- OBJECT (mode = 5): 末端执行器沿激活对象框架直线运动。方位使用外部旋转。

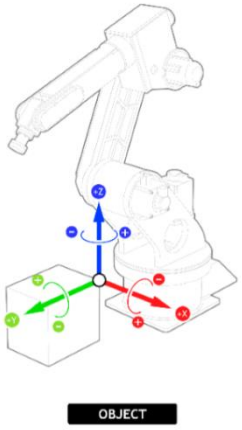


图 5-25 OBJECT 模式下 Jog 机器人

5.4 运动指令

5.4.1 总体概述

RPS 有一组运动命令，通过这些命令可以多种方式将机器人从一个位置移动到另一个位置。根据某些参数可以修改运动。这些参数可以嵌入到 move 指令中，也可以作为单独的命令进行设置。由于独立命令会覆盖默认命令并应用于所有后续的动作指令，因此该动作之后的所有动作都将使用新的参数值执行。由于内嵌参数只影响到所属的移动指令。如果一个移动指令没有参数或只有几个参数，它将使用任何缺失的参数默认值。内嵌参数的顺序不会影响移动指令

表 5-1 指令中的内嵌参数

	嵌入参数	参数范围
WORLD_POS_SPEED: linear speed mm/s	S	MOVEC:[0,WORLD_POS_MAX_SPEED] MOVEL:[0,WORLD_POS_MAX_SPEED]
WORLD_ORI_SPEED: joint speed deg/s	S	MOVEJ :[0, WORLD_ORI_MAX_SPEED]
WORLD_MAX_ACC	A	Percentage of WORLD_MAX_ACC [1 - 100]
WORLD_MAX_ACC	D	Percentage of WORLD_MAX_ACC [1 – 100]
TOOL_OFFSET	T	[0-31]
OBJECT_FRAME	O	[0-31]
ROBOT CONFIGURATION	C	SCARA : {0, 1} 6 Axis Anthropomorphic : {0,1,2,3,4,5,6,7} Palletizing : {0}

ESTUN e-Trio

	嵌入参数	参数范围
PRECISION_ZONE	ZF: 从过渡曲线开始到过渡曲线结束方向都是变化的。 ZT: 沿混合曲线方向变化。	[0 - Length of move]
按路径中断百分比	IP	[0 - 200]
输入中断	II	输入数字取决于控制器
中断时间	IT	[0, ∞) mS
程序输出	RO	输出数量取决于控制器

速度

速度可以指定为线性运动和方向运动的默认参数，使用 WORLD_POS_SPEED 表示线速度，使用 WORLD_ORI_SPEED 表示旋转速度，使用 deg/s 表示。它的内嵌参数是' S '，如果它是用于线性运动，如 MOVEL, MOVEC 或 MOVELREL。同样地，如果‘S’和关节运动一起使用，就表示关节速度，比如 MOVEJ 或 MOVEJREL。

加速度

在 ROBOT_DEFINITIONS 文件中，通过命令 WORLD_POS_MAX_ACC(用于线性轴)和 WORLD_ORI_MAX_ACC(用于方向轴)，加速度被指定为默认参数。它的内嵌参数是“A”，可以设置为 ROBOT_DEFINITIONS 中指定的加速度的%。这些值在 1(指定加速度的 1%)到 100(指定加速度的 100%)之间。

减速度

在 ROBOT_DEFINITIONS 文件中，通过命令 WORLD_POS_MAX_ACC(线性轴)和 WORLD_ORI_MAX_ACC(方向轴)将减速指定为默认参数。它的内嵌参数是' D '，可以设置为 ROBOT_DEFINITIONS 中指定的减速的%。值在 1(指定减速的 1%)到 100(指定减速的 100%)之间。

配置

机器人配置总是被当前配置覆盖，只能在 MOVEJ 中使用它。如果用与当前配置不同的配置执行 MOVEJ 命令，下一条 move 指令将使用新的机器人配置。这将防止奇异性，因为连续的线性运动不能有不同的机器人构型。

按路径中断百分比

当机器人到达指定的路径百分比时，将产生一个中断。中断将打开/关闭指定的输出。

示例: MOVEL GTA(10) IP:=50 RO:=9,ON

在这个例子中，当机器人完成到 gta10 的 50%的移动时，输出 9 将被打开。

这个参数的范围是从 0 到 200。即使在机器人混合两个动作时，也可以生成中断。可以

通过将 IP 指定为 100 到 200 来实现。

示例: MOVEL GTA(10) Z:=30 IP:=130 RO:=9,ON

在本例中，当机器人完成混合曲线的 30%时，将打开输出 9。

如果没有一个动作和一个 IP >指定为 100，则不执行任何操作。

输入数中断

当 II(输入中断)中指定的输入出现上升/下降沿时，指定的例程输出将被打开/关闭。

示例: MOVEL GTA(10) II:=3,ON RO:=9,ON

在本例中，当输入 3 给出上升沿(从低到高)时，在执行移动时将打开输出 9。

如果输入 3 在移动结束后打开，将不采取任何行动。

中断时间

指定的输出将在达到指定的时间(输入时间)后打开/关闭。

示例: MOVEL GTA(10) IT:=700 RO:=9,ON

在本例中，在移动开始后 700mS 之后将打开输出 9。

如果行动在 700 毫秒之前结束，将不采取任何行动

程序输出

当满足 IP(中断百分比)、II(中断输入)或 IT(中断时间)所规定的任何条件时，RO(输出)中规定的输出将被打开/关闭。如果输出已经处于指定的状态，则满足条件后将不采取进一步的操作。

5.4.2 过渡精度

过渡精度是机械手末端执行器对一个示教点的逼近程度。

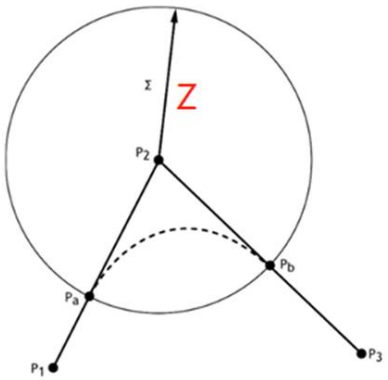


图 5-26 过渡精度

如果点到点的轨迹必须经过每一点，那么轨迹必须在路径方向改变的每一点停止。否则，

轨迹的速度就会出现不连续。这些停止可以通过允许路径稍微偏离点来避免，使用混合曲线来平滑靠近点的路径，同时改变方向但不停止。系统允许使用这种方法平滑过渡任意两条直线之间产生的 MOVEL 指令。无法控制过渡形状。然而，提供一个过渡参数，如何接近一个给定的点。有两个过渡参数:ZF 和 ZT。ZF 指示系统的取向变化将从直线的开始到结束的混合曲线,与此同时 ZT 型表明，取向变化将沿着混合曲线完全完成。

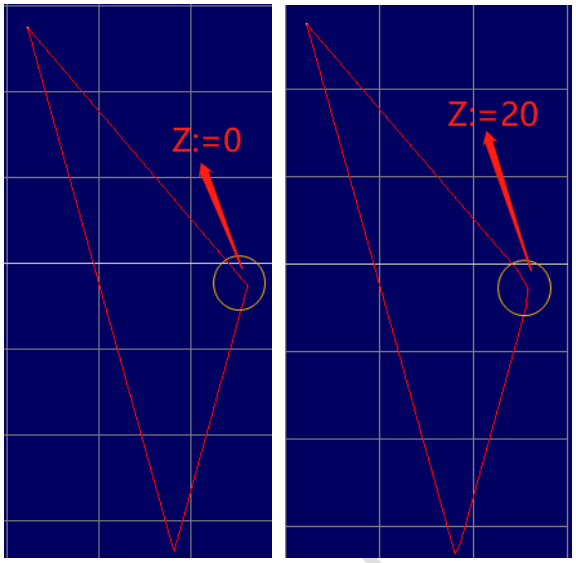


图 5-27 有过渡参数和没有过渡参数的运动轨迹对比

5.4.3 MOVEJ

MOVEJ 用于将机器人从一点移动到另一点，其移动过程不一定是线性的，机器人的所有轴同时到达目标位置。运动的过程不关注末端位置的中间点，可以获得较快的定位速度。

运动的速度是一个内插的速度，而不是任何单独的轴或末端执行器的速度。

例如:机器人必须从 10.5,0,20,0,0,0 的关节位置移动到 90.21,44.2,24.1,4.1,0.0,12.2，速度为 50。

速度 50 将应用于总位置变化，(90.21 - 10.5 + 44.22-0 + 24.1-20 + 4.1-0 + 0-0 + 12.2 -0)= 144.33

机器人将以 50 的速度完成 144.33 的距离。各个轴可以以高于或低于 50 的速度运动，但所有轴都将同时到达终点位置。

语法：

MOVEJ target [S:= T:= O:= C:= Z:= A:= D:=]

示例：

GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"

OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)
ROBOT(0)
MOVEJ GTA("p1")
MOVEJ GTA("p2")
MOVEJ GTA("p3")
STOP

这个例子命令机器人 0 线性移动到目标点 0。注意，在移动中没有其他嵌入的指令。此指令将使用以下参数将机器人移动到目标点 1。

表 5-2 示例中的参数说明 1

类型	参数
Object Frame	当前激活的对象框架
Tool Offset	当前激活的工具框架
Speed [$\boldsymbol{deg/s}$]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR
Acceleration [$\boldsymbol{deg/s2}$]	WORLD_ORI_MAX_ACC
Jerk [$\boldsymbol{deg/s3}$]	WORLD_ORI_MAX_JERK
Blend radius [$\boldsymbol{mm}$]	0
Configuration	0

MOVEJ GTA("p1") S:=100

表 5-3 示例中的参数说明 2

类型	参数
Speed[$\boldsymbol{deg/s}$]	100*LOOKAHEAD_FACTOR

MOVEJ GTA("p1") O:="of1"

表 5-4 示例中的参数说明 3

类型	参数
Object Frame	Object Frame 1
Speed[$\boldsymbol{deg/s}$]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR

框架 1 没有被设置为激活对象框架。这条指令只考虑了框架 1 的移动

MOVEJ GTA("p1") T:="tool1"

表 5-5 示例中的参数说明 4

类型	参数
Object Frame	当前激活的对象框架
Tool Offset	工具框架 1

框架 1 没有被设置为激活工具框架。这条指令只考虑了框架 1 的移动

ESTUN & TRO

MOVEJ GTA("p1") Z:=67

MOVEJ GTA("p2")

表 5-6 示例中的参数说明 5

类型	参数
Tool Offset	当前激活的工具坐标系
Blend radius[$\boldsymbol{mm}$]	67mm

机器人将向目标点 0 和 67 前进，在到达目标点之前，它将混合下一步的移动并向目标点 1 前进。

MOVEJ GTA("p1") S:=150 O:="of1" T:="tool1" Z:=42 C:=0

MOVEJ GTA("p2") S:= 100 C:=1

表 5-7 示例中的参数说明 6

类型	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)
Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset
Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ ACC_FA CTOR	WORLD_POS_MAX_ACC*JOG_ACC _FACT OR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0
Configuration	0	1

GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)
ROBOT(0) MOVEJ GTA("p1") S:=80 O:=1 T:=1 Z:=159 C:=1
MOVEJ GTA("p2") S:= 60 C:=0
MOVEJ GTA("p3")
STOP

ESTUN & TIP

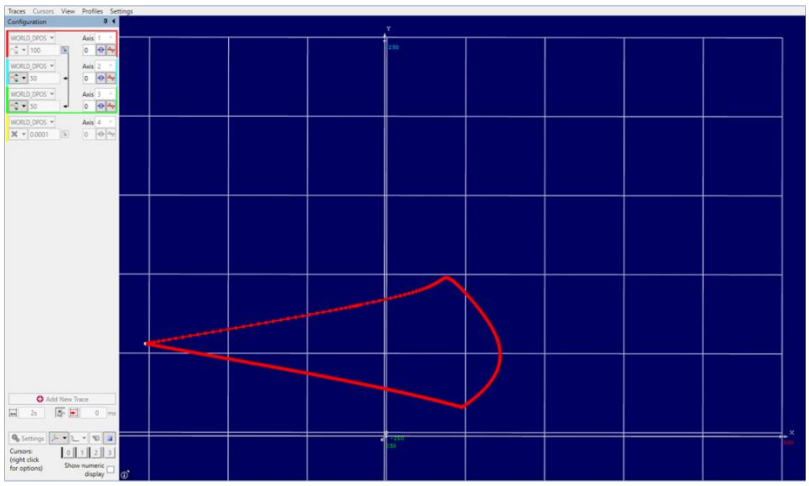


图 5-28 示例中机器人运行轨迹

5.4.4 MOVEL

MOVEL 是用来将机器人沿线性路径从一点移动到另一点。在笛卡尔空间中，末端执行器沿当前点和目标点之间的直线运动，所有轴同时到达目标位置。目标点角度相对当前点角度超过 180 度，建议使用 MOVEJ 指令。

语法

MOVEL target [S:= T:= O:= ZT:= ZF:= A:= D:=]

示例：

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)
ROBOT(0)
MOVEL GTA("p1")
MOVEL GTA("p2")
MOVEL GTA("p3")
STOP
```

这个例子命令机器人 0 线性移动到目标点 0。注意，在移动中没有其他嵌入的指令。此指令将使用以下参数将机器人移动到目标点 1。

表 5-8 MOVEL 示例中的参数说明 1

类型	参数
Object Frame	Currently active object frame
Tool Offset	Currently active tool offset
Speed [mm/s]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR

ESTUN & Trio

Acceleration [$mm/s2$]	WORLD_ORI_MAX_ACC*JOG_ACC_FACTOR
Jerk [$mm/s3$]	WORLD_ORI_MAX_JERK
Blend radius [mm]	0

MOVEL GTA("p1") S:=100

表 5-9 MOVEL 示例中的参数说明 2

类型	参数
Speed[mm/s]	100*LOOKAHEAD_FACTOR

MOVEL GTA("p1") O:="of1"

表 5-10 MOVEL 示例中的参数说明 3

类型	参数
Object Frame	Object Frame “of1”
Speed[mm/s]	WORLD_ORI_SPEED *LOOKAHEAD_FACTOR

框架 1 没有被设置为激活的对象框架。这条指令只考虑框架 1 的移动

MOVEL GTA("p1") T:="tool1"

表 5-11 MOVEL 示例中的参数说明 4

类型	参数
Object Frame	Currently active object frame
Tool Offset	Tool Frame “tool1”

MOVEL GTA("p1") Z:=67

MOVEL GTA("p1")

表 5-12 MOVEL 示例中的参数说明 5

类型	参数
Tool Offset	Currently active tool offset
Blend radius[mm]	67

机器人将向目标点 0 和 67 前进，在到达目标点之前，它将混合下一步的移动并向目标点 1 前进。

MOVEL GTA("p1") S:=150 O:="of1" T:="tool1" Z:=42

MOVEL GTA("p1") S:= 100

表 5-13 MOVEL 示例中的参数说明 6

类型	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)
Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset

ESTUN & Inc

Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ ACC_FA CTOR	WORLD_POS_MAX_ACC*JOG_ACC_ FACT OR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1, -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1, 0, 0, 50, 0, 0, 0)
ROBOT(0)
MOVEL GTA("p1") S:=280 O:=1 T:=1 Z:=80
MOVEL GTA("p2") S:=360
MOVEL GTA("p3")
STOP
```

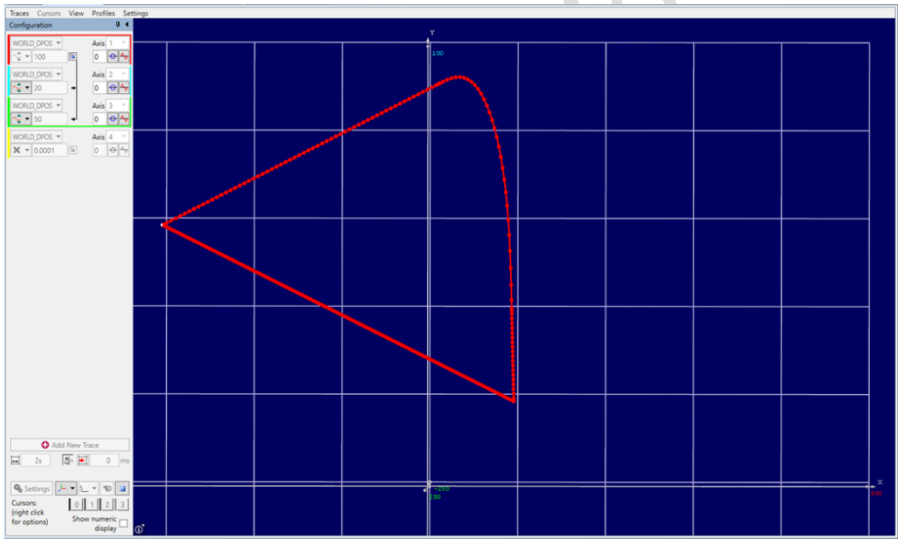


图 5-29 示例中机器人的运动轨迹

5.4.5 MOVEC

MOVEC 是用来将机器人沿圆周路径从一点移动到另一点。所有轴同时到达目标位置。

语法

MOVEC target (middle point) target (end point) [S:= T:= O:= A:= D:=]

示例：

```
GTA(0) = 350, 0, 50, 180, 0, 0
GTA(1) = 400, 100, 20, 180, 0, 0
GTA(2) = 450, 0, 60, 180, 0, 0
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
```

TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)
ROBOT(0)
MOVEC GTA("p1") GTA("p2")
MOVEL GTA("p3")
STOP

这个例子命令机器人线性移动到目标点 0。注意，在移动中没有其他嵌入的指令。此指令将使用以下参数将机器人移动到目标点 1。

表 5-14 MOVEC 示例中的参数说明 1

类型	参数
Object Frame	Currently active object frame
Tool Offset	Currently active tool offset
Speed [$\boldsymbol{mm/s}$]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR
Acceleration [$\boldsymbol{mm/s2}$]	WORLD_ORI_MAX_ACC*JOG_ACC_FACTOR
Jerk [$\boldsymbol{mm/s3}$]	WORLD_ORI_MAX_JERK
Blend radius [$\boldsymbol{mm}$]	0

MOVEC GTA("p1") GTA("p2") S:=100

表 5-15 MOVEC 示例中的参数说明 2

类型	参数
Speed[$\boldsymbol{mm/s}$]	100*LOOKAHEAD_FACTOR

MOVEC GTA("p1") GTA("p2") O:="of1"

表 5-16 MOVEC 示例中的参数说明 3

类型	参数
Object Frame	Object Frame 1
Speed[$\boldsymbol{mm/s}$]	WORLD_ORI_SPEED *LOOKAHEAD_FACTOR

MOVEC GTA("p1") GTA("p2") T:="tool1

表 5-17 MOVEC 示例中的参数说明 4

类型	参数
Object Frame	Currently active object frame
Tool Offset	Tool Frame 1

MOVEC GTA("p1") GTA("p2") Z:=67

MOVEL GTA("p3")

表 5-18 MOVEC 示例中的参数说明 5

类型	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)

Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset
Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ ACC_FA CTOR	WORLD_POS_MAX_ACC*JOG_ACC_ FACT OR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEC GTA("p1") GTA("p2") S:=150 O:="of1" T:="tool1"
MOVEL GTA("p3") S:= 100
STOP
```

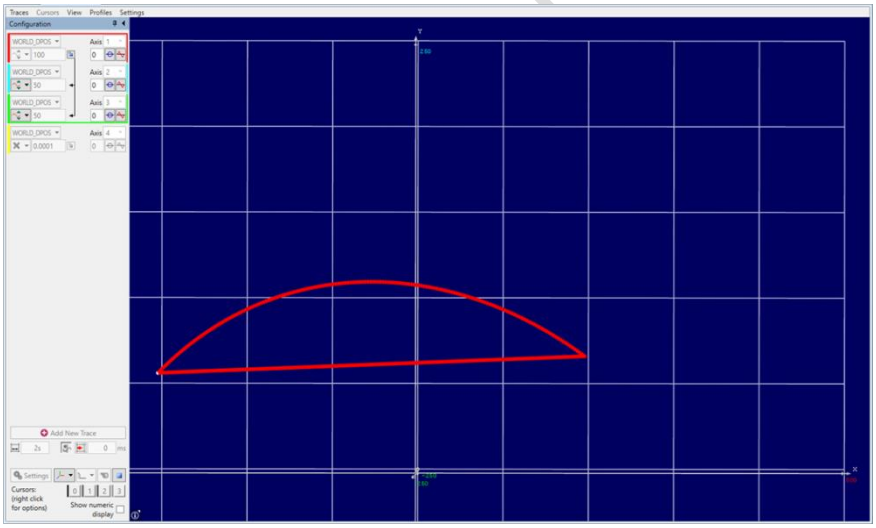


图 5-30 MOVEC 示例中机器人运动轨迹

5.4.6 MOVEJREL

MOVEJREL 用于相对于起始位置沿非线性路径将机器人从一点移动到另一点。所有轴同时到达目标位置。

运动的速度是一个内插的速度，而不是任何单独的轴或末端执行器的速度。只需将目标的所有值设置为 0，就可以移动单个关节，但需要移动的关节除外。

语法

MOVEJREL target [S:= T:= O:= C:= Z:= A:= D:=]

示例：

```
GTA(0) = 0,-20,0,0,0,0
ROBOT(0)
MOVELREL GTA(0) '机器人将从当前位置向负方向移动第二个关节 20 度
STOP
```

5.4.7 MOVELREL

MOVELREL 是用来将机器人相对于起始位置沿线性路径从一点移动到另一点。在笛卡尔空间中，末端执行器沿当前点和目标点之间的直线运动。所有轴同时到达目标位置。只需将目标的所有值设置为 0，就可以移动单个向量。当目标点旋转角度相对当前点大于等于 180 度时，建议使用 MOVJREL。

语法

MOVELREL target [S:= T:= O:= ZT:= ZF:= A:= D:=]

示例：

```
GTA(0) = 10,0,0,0,0,0
ROBOT(0)
MOVELREL GTA(0) ' “机器人会从当前位置 X 方向移动 10mm
STOP
```

5.5 碰撞区域

5.5.1 定义

碰撞检测算法可以检测和控制真实物体周围的定向区域(COLLISION_OBJECT 指定的 OBB)与 TOOL_COLLISION 指定的 OBB 之间的碰撞。

每个伺服周期检查 TOOL_COLLISION 与激活的 COLLISION_OBJECT(s) 之间的距离，当碰撞距离小于减速距离时，机器人将以 FAST_DEC 减速。

建议指定 COLLISION_OBJECT 和 TOOL_COLLISION 比实际对象稍大一点，以便工具能够在未碰之前停止。

ESTUN & Trio

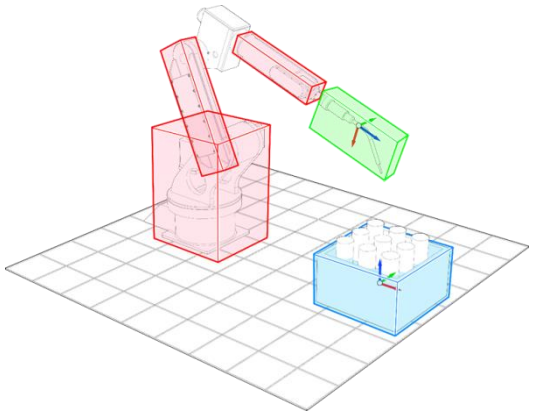


图 5-31 机器人碰撞区域定义

Tool collision object 用绿色表示。Collision object 用蓝色表示，OBBs 绑定机器人用红色表示。碰撞算法测量 tool collision object 与机器人连杆的 collision objects 和 OBBs 之间的距离。如果该距离小于快速减速距离，机器人将安全停止，避免碰撞。注意算法使用 tool collision 其他物体，机器人连杆的 OBBs 会与碰撞物体发生碰撞。

5.5.2 语法

COLLISION_OBJECT(identity, name, x centre, y centre, z centre, x rotation, y rotation, z rotation, x half distance, y half distance, z half distance)

X 中心，y 中心和 z 中心必须用毫米表示。

X 旋转，y 旋转和 z 旋转必须用角度来表示。

X 半距离，y 半距离和 z 半距离必须用 mm 表示。

关键字	描述
x, y and z centres	OBB 在物体上的中心位置
x, y and z rotations	OBB 在物体上的方向
x, y and z half distances	在 OBB 的每个向量中测量一半的距离。

这幅图描述了碰撞对象的参数。框架箭头位于对象的中心表示物体的位置和方向。

在不同向量中测量的一半距离用相同颜色的向量表示。

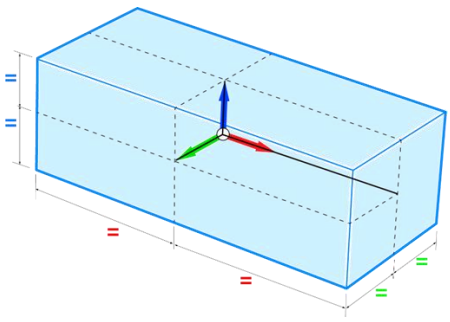


图 5-32 碰撞区域的尺寸定义

碰撞对象可以按如下方式激活或停用:

•激活: COLLISION_OBJECT(identity, ON)

•停用: COLLISION_OBJECT(identity, OFF)

相同的 COLLISION_OBJECT 可以对一些运动学组激活，对其他一些组禁用。COLLISION_OBJECTs 活动的最大数量是 10。要知道哪些对象是活动的，只需简单地打印 KINEMATIC_GROUP。

如果它在任何运动组中都是活动的，它就不能被编辑。

删除一个条目可以使用参数-2 和索引，或者只使用一个参数(-2)来删除所有条目。

例子:

COLLISION_OBJECT(-2)

对现有条目的请求可以通过参数-1 实现。如果参数是(-1,index)，则只返回请求的索引。

例子:

COLLISION_OBJECT(-1,2)

Terminal:

2 [object2] : 350.00000, 0.00000, 50.00000, 0.00000, 0.00000, 150.00000,50.00000, 50.00000, 50.00000

ESTUN & Trio

5.6 动态抓取

5.6.1 语法

有可能使机器人 TCP 与一个移动的对象或与另一个机器或机器人同步。这表示机器人 TCP 将与被同步对象相同的方向移动或旋转。有两种方法来同步机器人：TCP:与 object frame 同步和与 GTA 同步。为了与一个 object frame 同步，它必须先被附加到某个轴上。

因为同步不会加载到移动缓冲区中，所以它们不会通过 CANCEL 或 RAPIDSTOP 取消，必须执行去同步或分离。

5.6.2 Attach Object Frame

附加 Object Frame 是一种方法，将一个 Object Frame 的矢量与最多六个轴连接起来并放置它在一个特定的位置。物体坐标系会根据这些轴移动。有两种不同的模式附加一个对象框架;一种是专门设计来配合传送带工作的，另一种是配合多用途机器工作的。

1.传送带附加 Object Frame

由于传送带可以在不同的方向，Object Frame 可以设置在多个方向匹配传送带。选定的

Object Frame 矢量将沿着输送机矢量方向移动。

ATTACH_OBJECT_FRAME(control, object frame, syncVector, syncPos, syncAxis)

表 5-19 传送带附加 ATTACH_OBJECT_FRAME 参数说明

参数	说明
Control	选择 ATTACH_OBJECT_FRAME 模式。值= 1。
Object Frame	附加 Object Frame。填索引或名称。
syncVector	Object Frame 向量，将附加到 syncAxis。 有 3 种可能值:1:X, 2: Y, 3: Z
syncPos	syncAxis 上的捕获位置
syncAxis	与之同步的轴

例子:

一个常见的行业例子是从托盘中挑选对象，并将其放置到另一个工作站或 A 盘中。

利用附加 object Frame 技术可以简化同步过程。一个 object Frame 将被放置在托盘的特定位置。所有的对象将与 Object Frame“托盘”相关。Object Frame“托盘”将被附加到传送带上，使用三种选项:向量 X，向量 Y 或向量 z。输送带不可能完全对齐，因此，可以利用 Object Frame 的方位数据对机器人进行调整。

ATTACH_OBJECT_FRAME(1,"pallet",1,registed_position,11)

命令执行后，Object Frame “托盘”将精确地跟随传送带。在这种状态下,机器人应与 Object Frame “托盘”同步。

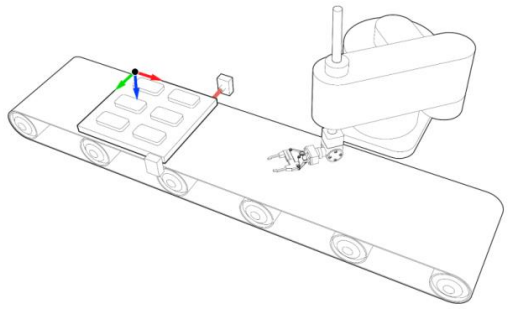


图 5-33 ATTACH_OBJECT_FRAME 应用示例

2.多机附加 Object Frame

对于具有多个轴的机器，可以附加的位置和方向矢量多达 6 个轴的控制参数。

例如，如果一个物体坐标系附加在 6 自由度的机器人手臂上，Object Frame 的位置和方向将遵循机器人 TCP。

ATTACH_OBJECT_FRAME(control, object frame, axisX, axisY, axisZ, axisU, axisV, axisW, xOffset, yOffset,zOffset, uOffset, vOffset, wOffset)

表 5-20 多机附加 ATTACH_OBJECT_FRAME 参数说明

参数	说明
Control	选择 ATTACH_OBJECT_FRAME 模式。 值= 2。
Object Frame	附加 Object Frame。填索引或名称。
axisX - axisW	与将要的 Object Frame 向量相对应的机器轴数被附加。设 -1 为非附加向量。
xPos - wPos	Object Frame 的位置和方向与机器人的原点有关。

例子：

ATTACH_OBJECT_FRAME(2,3,-1,11,-1,-1,-1,-1,400,0,400,0,90,0)

3.分离 Object Frame

ATTACH_OBJECT_FRAME(control, object frame)

表 5-21 分离 ATTACH_OBJECT_FRAME 参数说明

参数	说明
Control	选择 ATTACH_OBJECT_FRAME 模式。值= 4。
Object Frame	将被附加的 Object Frame。 填索引或名称。

例子：

ATTACH_OBJECT_FRAME(4,"of2")

可以通过在命令中使用 index 作为参数来请求现有条目。它返回 TRUE 或 FALSE，取决于条目是否已经附加。

例子：

ATTACH_OBJECT_FRAME("of2")

5.6.3 Sync to object frame

当附加了一个 object frame，可以用命令 SYNC_TO_OBJECT_FRAME。

SYNC_TO_OBJECT_FRAME(control, time, object frame, GTA)

表 5-22 SYNC_TO_OBJECT_FRAME 的参数说明

参数	说明
Control	选择 SYNC_TO_OBJECT_FRAME 工作模式。 1 =同步，4 =停止同步，10 =重新同步到另一个附加的 object frame
time	以毫秒为单位完成同步的时间。

ESTUN & Fanuc

参数	说明
object frame	object frame 已经附加。 它可以是索引或名称。
GTA	GTA 相关 object frame 同步。 它可以是索引或名称。 设为-1 表示没有 GTA

在“停止同步”的情况下，唯一可用的参数是 control, time

SYNC_TO_OBJECT_FRAME(control, time)

通过询问 SYNC_CONTROL 命令来检查机器人是否已经同步。

表 5-23 机器人是否同步的查询参数

参数	说明
SYNC_CONTROL	Desync = 0 Synced = 3

例子:

```
WAIT UNTIL SYNC_CONTROL axis(2) = 3
```

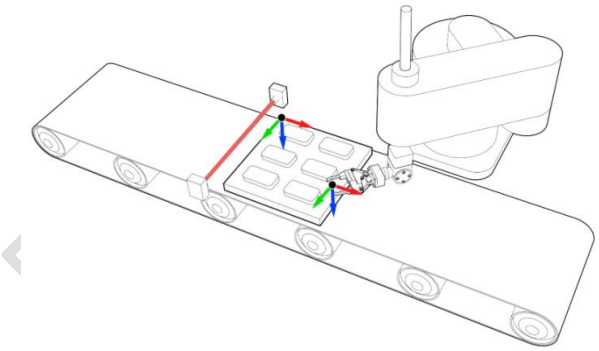


图 5-34 机器人同步应用示例

例子:

在将 object frame 附加到托盘上之后，机器人必须与物体同步，在这个特殊的情况下，放置在一个特定的对象上(GTA 与对象框架“托盘”相关)

SYNC_TO_OBJECT_FRAME(1,1000,"pallet","object_1")

Sync to GTA

Sync to GTA 使同步更简单。它与机器人 TCP 进行同步 X 方向上的一个轴。如果机器人没有完全与 X 方向对齐，可以预先设置和选择 object fram。

SYNC_TO_GTA(control, time, syncPos, syncAxis, object frame, GTA)

表 5-24 SYNC_TO_GTA 工作模式的参数说明

参数	说明
control	选择 SYNC_TO_GTA 工作模式。 1 =同步，4 =停止同步，10 =重新同步到另一个 GTA。
time	以毫秒为单位完成同步的时间。
syncPos	syncAxis 上的捕获位置
syncAxis	同步轴
object frame	与选定的 GTA 相关的 Object frame。如果没有 Object frame，则设置为 0(默认)。它可以是索引或名称。
GTA	与 GTA 同步。它可以是索引或名称。 设为-1 表示没有 GTA

在“停止同步”的情况下，唯一可用的参数是 control 和 time

SYNC_TO_OBJECT_FRAME(control、time)

通过 SYNC_CONTROL 命令来检查机器人是否已经同步。

在这种情况下，因为它与 X 轴同步，所以没有必要指定机器人轴。

表 5-25 机器人同步的查询参数

参数	说明
SYNC_CONTROL	Desync = 0 Synced = 3

例子:

```
WAIT UNTIL SYNC_CONTROL = 3
```

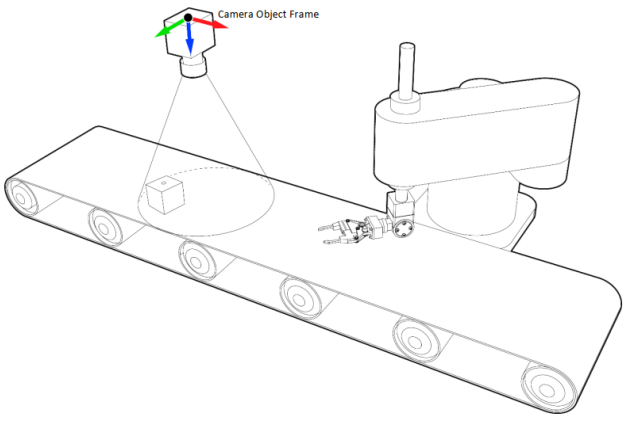


图 5-35 机器人同步的应用示例

例子:

产品被随机放置在传送带上，被机器人抓取并放置到特定的位置。视觉系统识别产品的

ESTUN & Trio

位置和方向，并将这些信息发送给控制器。产品的位置和方向信息可以通过以太网通讯或使用 Trio ActiveX 发送。视觉系统测量的偏移量与坐标的相机原点有关，所以我们需要通过使用 OBJECT_FRAME 告诉控制系统原点在哪里。机器人与传送带 X 方向同步，因此 OBJECT_FRAME 的 X 向量应该与输送带的方向对齐。在相机原点中设置 OBJECT_FRAME 之后，GTA 应该被设置为检测的偏移量。在本例中，相机将向 TABLE 数据区域发送 5 个参数:table 0:X 偏移量，table 1:Y 偏移量，table 2:方位，table 3:输送机位置，表 table 4:检测到的位置。

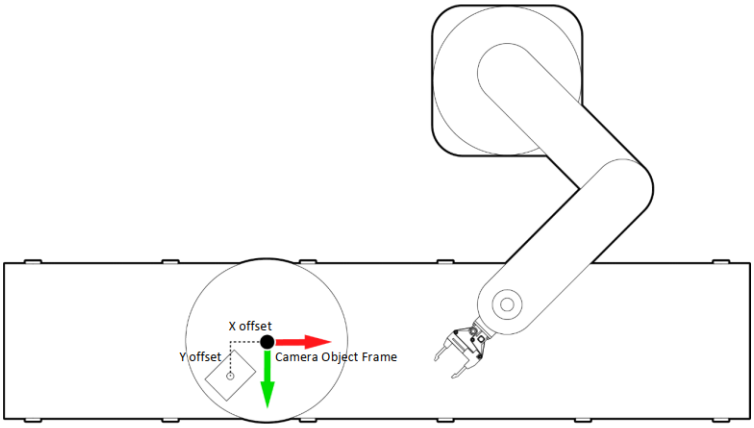


图 5-36 机器人同步应用示例

代码:

```
fixed_z = 50 '由于输送带与地面平行，Z 值总是相等的
OP(gripper,OFF) ' 松开夹爪
SYNC_TO_GTA(4,500) '同步 500ms 后结束同步
WAIT UNTIL SYNC_CONTROL = 0 ' 判断同步结束
WAIT UNTIL TABLE(4) = 1 '等待产品被检测出来
TABLE(4) = 0 '重置产品检测
'设置 GTA 值为相机测量偏移量
GTA(0) = TABLE(0), TABLE(1),fixed_z, 180,0, TABLE(2)
'当摄像机检测到产品时，定位输送机位置
registed_position = TABLE(3)
'同步轴 10, GTA(0) in Object Frame “camera” (1) in 2000ms
SYNC_TO_GTA(1,2000,registed_position,10,1,0)
WAIT UNTIL SYNC_CONTROL = 3 ' 等待完全同步
OP(gripper,ON) ' 关闭夹爪
WA(200) ' 夹爪关闭延时 200ms
SYNC_TO_GTA(4,500) '同步 500ms 后结束同步
WAIT UNTIL SYNC_CONTROL = 0 '判断同步结束
STOP
```

IEC61131-3 篇

6 概述

6.1 IEC61131-3 标准

IEC 61131 是国际电工委员会(IEC) 制定的可编程逻辑控制器标准。在 IEC 编号系统更改以前，称为 IEC 1131。

1993 年国际电工委员会（IEC）正式颁布了可编程控制器的国际标准 IEC 1131（以后改称 IEC 61131），其中的第三部分关于编程语言的标准，规范了可编程控制器的编程语言及其基本元素。这一标准为可编程控制器软件技术的发展，乃至整个工业控制软件技术的发展，起了举足轻重的推动作用。它是全世界控制工业第一次制定的有关数字控制软件技术的编程语言标准。此前，国际上没有出现过有实际意义的，为制定通用的控制语言而开展的标准化活动,这个标准的推出正是注意到了以数字技术为基础的控制装置在发展进程中，过于专有化，给用户带来的大量不便，为今天 PLC 走向开放式系统的打下了坚实基础。
国际标准 IEC 61131 组成：

1992 年以后，IEC 陆续颁布施行可编程序控制器国际标准 IEC 61131 的各个部分。已正式颁布的有：

IEC 61131-1 通用信息(1992，2003 年 5 月出版第二版）

IEC 61131-2 装置要求与测试（1992，2003 年 2 月出版第二版）

IEC 61131-3 编程语言（1993，2003 年 1 月出版第二版）

IEC 61131-4 用户导则（1995）

IEC 61131-5 通讯服务规范（2000）

IEC 61131-6 功能安全(2012)

IEC 61131-7 模糊控制编程软件工具实施（2000）

IEC 61131-8 语言的应用和实现导则（2001）

IEC 61131-3 是可编程控制器编程语言的标准，它极大地推动了工业控制系统软件设计的发展，对现场总线设备的软件设计也产生了很大的影响。符合 IEC 61131-3 标准的软件系统是一个结构完美、可重复使用、可维护的工业控制系统软件，它不仅能应用于可编程逻辑控制，而且能应用于过程控制。

6.2 PLCopen 组织

PLCOpen 是一个国际组织，它是独立于生产商和产品的全球性机构，成立于 1992 年，

总部设在荷兰。其宗旨是致力于提高控制软件编程方法、效率、规范等相关方面问题，从而支持使用该领域国际标准。随着中国经济迅猛增长，国内工业控制领域不断发展，市场主体对技术的依托要求越来越高，为满足业界需求，进一步在中国推广 PLCopen 相关技术。2005 年 9 月，PLCopen 国际组织与中国机电一体化协会合作成立 PLCopen 国际组织中国分支机构：PLCopen China。该组织是 PLCopen 国际组织继美国、日本后，在世界范围内设立的第三个分支机构。PLCopen 国际组织下设技术和推广委员会以开展技术指导、推介等工作。

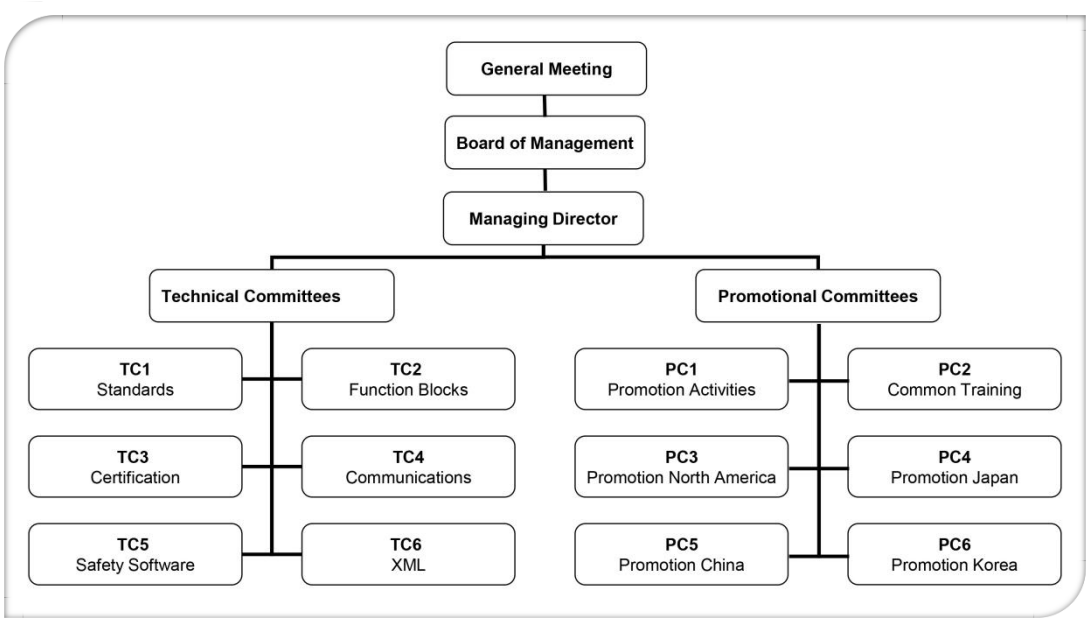


图 6-1 PLCopen 组织架构图

技术委员会包含 6 个分支机构：TC1 致力于标准制定、TC2 致力于功能块库的定义和协调功能块的调用、TC3 致力于认证工作、TC4 致力于制定编程语言的通信模型、TC5 致力于功能安全、TC6 致力于制定 IEC61131-3 各种编程语言和工程文件的 XML 数据交换格式。

推广委员会：PC1~PC5，分管公共培训、各地区的推介事务。PC5 委员会主要管理与中国有关的推广和介绍等事务。

6.3 认证等级

制造商可编程控制器产品的认证等级就是证明该产品符合 IEC61131-3 标准的程度，分为 3 个等级：

基本级（BL）：表示由该产品开发的程序基本结构与 IEC61131-3 的兼容，能够提供编程语言的基本语言元素。

符合级（CL）：表示产品符合 IEC61131-3 的全部性能，所支持的数据类型与它的服务适配。

可重复试用级（RL）：表示工程单元功能和功能块可在不同的可编程控制器系统重复使用，它们可以通过结构化文本语言实现转换。



图 6-2 IEC61131-3 认证等级标识

6.4 公用元素

IEC61131-3 标准可以被分成两个部分，第一个部分是公共元素、第二个部分是编程语言图 6-3。



图 6-3 IEC61131-3 标准的构成

公共元素包含：数据类型、变量、程序组织单元（函数、功能块、程序）、软件模型（配置、资源和任务）、顺序功能图。

6.4.1 数据类型

在 IEC61131-3 标准中通过数据类型严格定义了文字和变量可能的值，以及可以做的操作和存储值得方法，从而有效防止因对变量定义了不同数据类型而造成的错误，使程序的可靠性、可维护性和可读性大大提高。例如时间变量除以整型变量这种错误可以避免。

标准数据类型包括：

BOOL（布尔）、BYTE（8 位长度的位串）；

- INTEGER （整型）: SINT，INT，DINT，LINT，USINT，UINT，UDINT，ULINT；
- REAL（实数）、LREAL（常实数）
- DATE（日期）, TIME（时间）
- STRING（字符串）

在此基础上我们还可以定义用户数据类型（也称导出数据类型）。

1) 直接衍生数据类型：

缩写 PI 表示实数类型 REAL，初始值是 3.1415927，以后应用中可以直接用 PI 表示圆周率。

```
TYPE
    PI:REAL := 3.1415927;
END_TYPE
```

枚举数据类型：

```
TYPE
    AI_signal：(Single_Ended,Differential);
END_TYPE
```

2) 子范围数据类型：

可以约束数据的实际允许范围

```
TYPE
    Analog：INT(-10.0 .. 10.0);
END_TYPE
```

3) 数组数据类型：

多个数据类型相同的数据元素组成的集合

```
TYPE
    AI：ARRAY[1..10,1..10]  OF   Analog := ( 10(0), 10(0) );
END_TYPE
```

4) 结构化数据类型：

多个不同数据类型的数据组成的集合

```
TYPE
    PRESSURE_SENSOR：
        STRUCT
            SingleType:AI_signal
            Status:BOOL;
            Input:Analog;
            Min:Analog;
            Max:Analog;
```

ESTUN & Trio

```
        END_STRUCT
END_TYPE
```

5) 位域数据类型（Trio 编程环境特色）：

从整型数据类型中派生出位域数据类型，对某个位的访问直接使用名称，如 VarName.BitName，但是目前不支持从 LINT 类型派生。

6.4.2 变量

变量提供能够改变其内容的数据对象的识别方法。例如，可改变与控制器输入和输出或存储器有关的数据。

变量按照表示方法分为：单元素变量和多元素变量

单元素变量：可以直接表示基本数据类型元素、枚举类型元素或子范围数据类型。

多元素变量：可以表示数组或结构体类型变量。

变量按照是否规定存储器或输入输出的可寻址物理或逻辑地址分：直接变量和符号变量。

直接变量：使用百分号“%”开始，随后是位置前缀和大小前缀，如果有分级，则使用整数表示分级，并用小数点符号“.”分隔。无符号整数表示变量，如%IX0.0，%QW0。 直接变量类似传统可编程控制的操作数，它对应于某一可寻址的存储单元。

表 6-1 存储位置标识

位置前缀	含义	示例	示例说明
I	输入单元位置	%IW215	输入位置是第 215 字
Q	输出单元位置	%QB7	输出位置是第 7 字节
M	存储器单元位置	%MD48	存储器位置是第 48 双字开始

表 6-2 变量的数据类型

大小前缀	含义	示例	约定数据类型
X	单个位	%IX1	BOOL
None	单个位	%I1	BOOL
B	字节位（8 位）	%IB2	BYTE
W	字位（16 位）	%IW2	WORD
D	双字位（32 位）	%ID2	DWORD
L	长字位（64 位）	%IL2	LWORD

表 6-3 变量的寻址

分级寻址	示例	说明
简单寻址	%IX1	1 级
使用“.”分层寻址	%QX7.5	2 级

符号变量：用符号表示的变量，其地址对不同的可编程控制器可以不同，从而为程序的移植创造条件。例如，VAR_INPUT SW_1 AT % IX1.1:BOOL，符号变量 SW_1 表示从%IX1.1 地址读取布尔量，当实际控制器更改时，只需修改地址定义，在程序的其它部分仍然可以使用该符号，便于程序移植。

变量属性：通过变量属性的关键词进行定义。

表 4-1 变量属性关键字

变量属性关键字	变量属性说明
VAR	内部变量，程序组织单元内部的变量
VAR_INPUT	输入变量，由外部提供，在程序组织单元内部不能修改
VAR_OUTPUT	输出变量，由程序组织单元提供给外部实体使用
VAR_IN_OUT	输入-输出变量，由外部实体提供，能在程序组织单元内部修改
VAR_EXTERNAL	被定义在外部的全局变量，能在程序组织单元内部修改，
VAR_GLOBAL	全局变量，能在对应的配置、资源内使用
VAR_ACCESS	存取变量，用于与外部设备的不同程序间变量的传递
VAR_TEMP	暂存变量，在程序或功能块中暂时存储的变量
VAR_CONFIG	配置变量，实例规定的初始化和地址分配
RETAIN	变量的附加属性，即电源掉电时能够保持该变量值
CONSTANT	常数，程序执行时不能被修改
AT	变量存取的地址

初始化：

变量在系统启动时会进行初始化，初始化数值有三种来源。

- 当系统停止时，被定义为掉电保持属性变量的数值会被系统存储。如果系统重新启动，该变量将恢复之前的数值。
- 用户在声明变量时定义的初始值。
- 由系统根据不同的变量类型，提供的约定数值，默认为零。

变量初始取值的优先级： 掉电保持优先级最高，其次是用户初始值，最后是系统约定值。

系统重新上电过程中，如果包含掉电保持变量恢复过程，那么这个启动过程称为热启动，

ESTUN RTD

否则称为冷启动。具有 VAR_EXTERNAL 和 VAR_INPUT 属性的变量不能赋予初始值。

变量声明定义为建立变量与它的数据类型之间的关系，同时设置变量的用户初始值，变量声明格式如下：

VAR

声明格式：变量名：变量数据类型（:= 用户定义初始值）

END_VAR

表 6-4 变量申明示例

描述	示例	说明
基本数据类型变量	VAR START : BOOL; MYSTR : STRING[10]; VALUE AT %QW25 : INT; END_VAR	申明 START 是布尔类型 声明 MYSTR 是 10 个字符的字符串 声明 VALUE 是直接表示变量，整型数，地址为%QW25
用户定义数据类型变量	VAR MyVAR : myTYPE; END_VAR	用户数据类型 myTYPE 的变量申明
数组变量	VAR BITS : ARRAY[0..7] OF BOOL; CNTA : ARRAY[1..2,1..3] OF INT; OUTA AT %QW6 : ARRAY[0..9] OF INT; END_VAR	BITS 是 8 个布尔变量组成的数组 CNTA 是 2*3 整型变量维数组 OUTA 是 10 个整型变量的数组，开始地址是%QW6

ESTUN & TRIO

6.4.3 软件模型（配置、资源、任务）

IEC61131-3 的软件模型采用分层结构表示，外层包含内层的特性，从而构成有别于传统 PLC 的软件结构，图 6-4 可以帮助较好的理解这个软件模型。

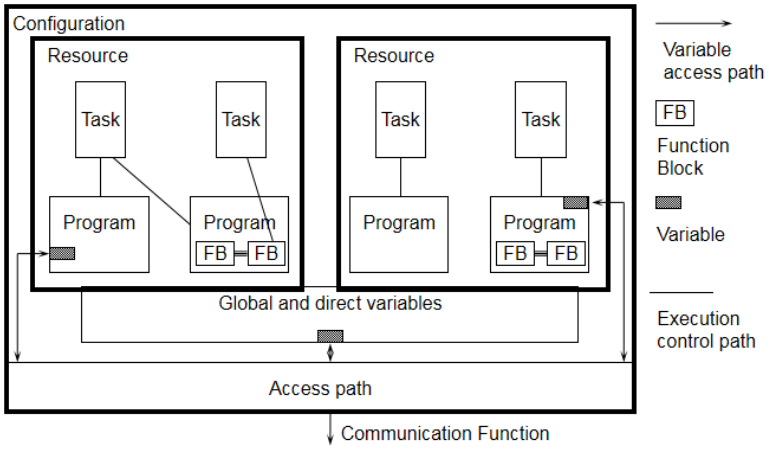


图 6-4 IEC61131-3 编程软件模型框图

模型的最外层是配置，对应一个控制系统，用于解决特定的控制问题。配置的内容包括，硬件装置、处理资源、I/O 通道的存储地址和系统能力。在资源内部包含多个资源，每个资源可以看做一个 CUP 单元，它具有独立执行 IEC 程序的运算能力。每个资源中可以定义若干个任务，每个任务中可以运行一个或者多个程序和功能块，多个任务可以被设置为不同优先级并行运行，在同一个时间只有最高优先级的任务在资源里运行，任务可以周期运行也可以被外部事件触发运行（如变量的变化）。

传统的 PLC 只包含一个资源，只运行一个任务，这个任务只控制一个程序循环执行，IEC61131-3 对这个简单系统进行了扩充，构建了一个实时多任务环境，更能满足现代控制系统开发的要求。

IEC 标准中允许变量在不同层级中定义，变量可访问的范围由它在哪个层级里定义决定。全局变量可以在整个系统中被访问，任务中里定义的变量可以被本任务中的所有程序访问，其他任务中的程序不可访问。程序中定义的变量只可以在本程序内部被访问，其他程序不可访问。

存取路径可以用于将全局变量、直接表示变量、功能块的输入和输出，程序内部变量联系起来，实现信息存取。它可以提供了在不同配置之间交换数据和信息的方法。

6.4.4 程序组织单元

在 IEC61131-3 标准中，程序、功能块和功能的集合称作程序组织单元 POU。IEC 的功能或者功能块分为两部分，标准部分（例如加减乘除、开方、三角函数）由 PLC 厂家提供，用户定义部分可以由用户复用标模块外加自定义部分实现。功能块等价于集成电路，具有特定的控制功能。

功能和功能块使用上具有差别，功能块既包含数据又算法逻辑，它能够记忆过去的行为。功能只包含算法逻辑，不包含记忆功能。功能和功能块都有清晰的外部接口便于隐藏内部的执行情况，它就像一个集成块或者软件黑盒，有了这种特性就可以把软件开发工作根据不同的编程水平进行区分。温度闭环控制的 PID 功能块就是一个好例子，它可以被不同程序重复使用，无需从零开发，除非有特殊要求。

ESTUN & TRIO

标准功能 FU：

ADD, SQRT, SIN, COS, GT, MIN, MAX, AND, OR, etc.

标准功能块 FBD：

SR,RS, SIN, CTU, PID, etc.

自定义功能 USER_FU：

```
FUNCTION Max2Int : REAL
        VAR_INPUT
            Var01     : REAL;
            Var02     : REAL;
        END_VAR
        VAR_OUTPUT
            Max2Int   : INT ;
        END_VAR
    END FUNCTION
```

END FUNCTION

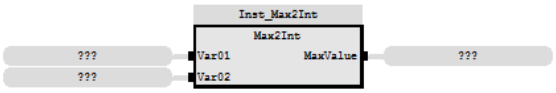


图 6-5 自定义功能示例

自定义功能块 USER_FBD：

```
FUNCTION_BLOCK    UFB_R_Trig

  VAR
       Inst_R_TRIG : r_trig ;
      enCheck : BOOL :=    FALSE ;    //Avoid rebooting checking when bIn has been TRUE.
  END_VAR
  VAR_OUTPUT
     Q : BOOL :=    FALSE ; //Output TRUE in the cycle period of bIn rising edge*=)
  END_VAR
  VAR_INPUT
       bIn : BOOL :=    TRUE ;    //Signal for checking
  END_VAR

END_FUNCTION_BLOCK
```

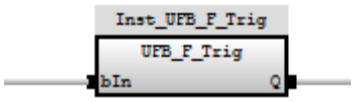


图 6-6 功能块的示例

程序定义为由一系列的功能或者功能块使用 IEC 编程语言构建的网络，图 6-7 是使用功能块图（FBD）编程语言编写的程序。

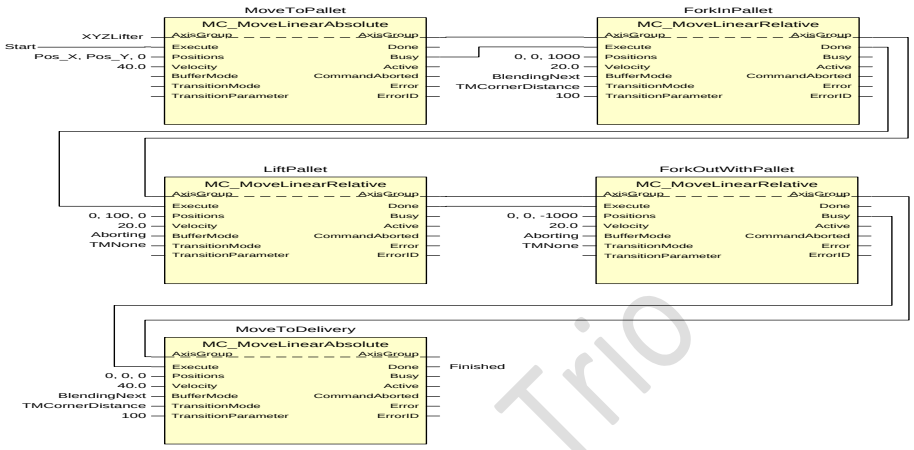


图 6-7 FBD 的程序示例

通过 POU，使用功能、功能块和程序，可以将复杂问题分解，采用模块化思想试下应用。这是 IEC61131-3 标准的目的之一，为软件实现模块化、多人协作、技术积累等软件的设计带来便利。

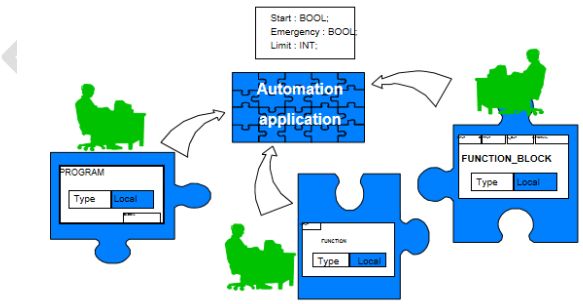


图 6-8 POU 的模块化设计思想

IEC61131-3 标准支持结构化软件开发，可以把复杂的控制分解为若干简单的自定义或者标准功能块，采用自顶层向下层的方式开发，也可以通过复用或继承已有的功能块，采用自底层向上层构建的方式开发。

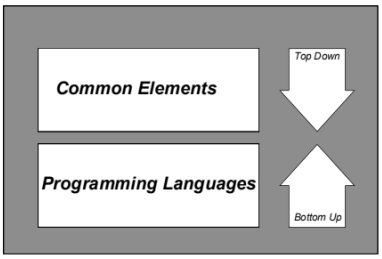


图 6-9 自顶向下或自底向上的软件设计

6.4.5 顺序功能图 SFC

在 IEC61131-3 标准中 SFC 即没有被归入文本编程语言也没有归入图形编程语言，它被作为公共元素加以定义。SFC 是一个问题分解工具，它通过图形的方式把控制程序描述为多个顺序行为。SFC 因为具有表达能力强、支持初步设计、详细设计和支持软件的分段执行等特点，所以 IEC 标准推荐采用 SFC 作为设计控制系统的主要工具，其他编程语言则用在控制软件代码开发阶段。

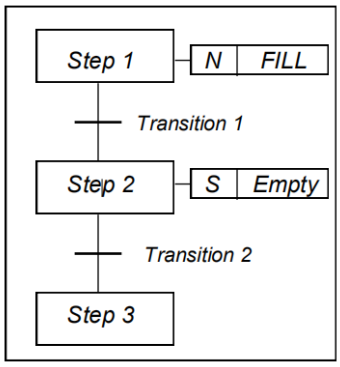


图 6-10 SFC 的程序构成

SFC 由步（Step）、与步连接的动作块（Action）和转换条件（Transtion）组成，如图 6-10。每一步代表系统的一个特定状态，动作与步进行连接，实行一定的控制功能。每个转换与一定的条件绑定，当条件满足时，将引起前一步转为非活动态，下一步转换为活动态。每个动作块或者转换条件可以使用 IEC 支持的任意编程语言开发，梯形图、功能块图、结构化文本甚至 SFC 本身。SFC 可支持单序列、选择分支、并行序列多种程序结构，如图 6-11。

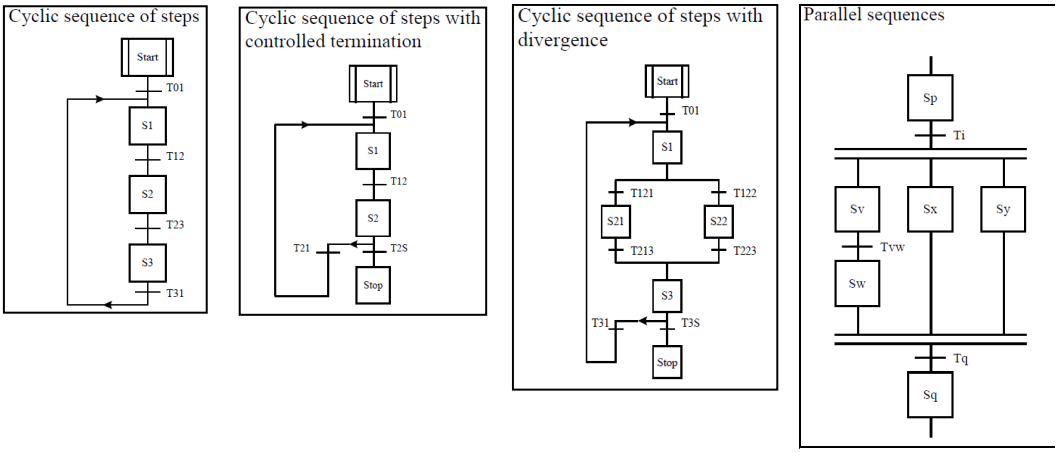


图 6-11 SFC 支持的分支结构

6.5 编程语言

IEC61131-3 标准中定义了四种语言，分为文本类语言和图形化语言两大类。文本类语言包括 IL 指令表（Motion Perfect 不支持）、ST 结构化文本（一种高级语言）；图形类语言包括 LD 梯形图和 FBD 功能块图。

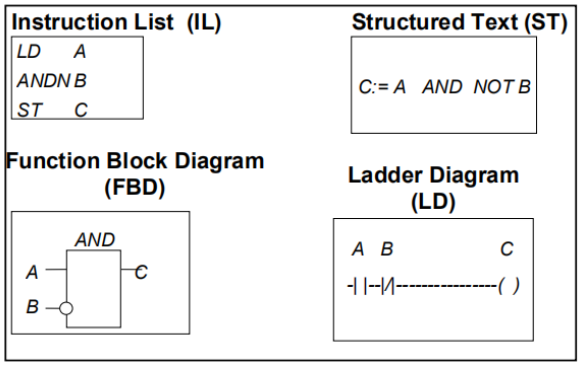


图 6-12 四种不同编程语言描述相同的功能(A 与 B 非得到 C)

Trio 支持的四种编程语言详细使用说明请参考 Motion Perfect V5.0 中的 IEC 61131-3 帮助文件。

不同的编程语言有各自的特点，梯形图（LD）编程在基于电气原理图的逻辑设计方面，其直观、易于理解被广泛使用。结构化文本（ST）语言由于其更接近高级语言的编程风格和思想，便于复杂逻辑和算法的实现，也被广泛使用。功能块图（FBD）采用图形化的编程语言，其直观、易于使用，被用户所接受。顺序功能图（SFC）可以实现程序流程的设计，可清晰地展示整个控制过程，常被用于软件的顶层设计。

IEC61131-3 提供了多种编程语言，在应用过程中选择何种语言，往往考虑如下因素：

- 项目合作开发人员和项目维护人员的知识背景。
- 程序开发者的知识背景，擅长使用的编程语言。
- 项目的特点、问题描述的复杂度和控制系统的结构。

7 Motion Perfect 中的 IEC61131-3 的使用

7.1 MotionPerfect 编程主界面

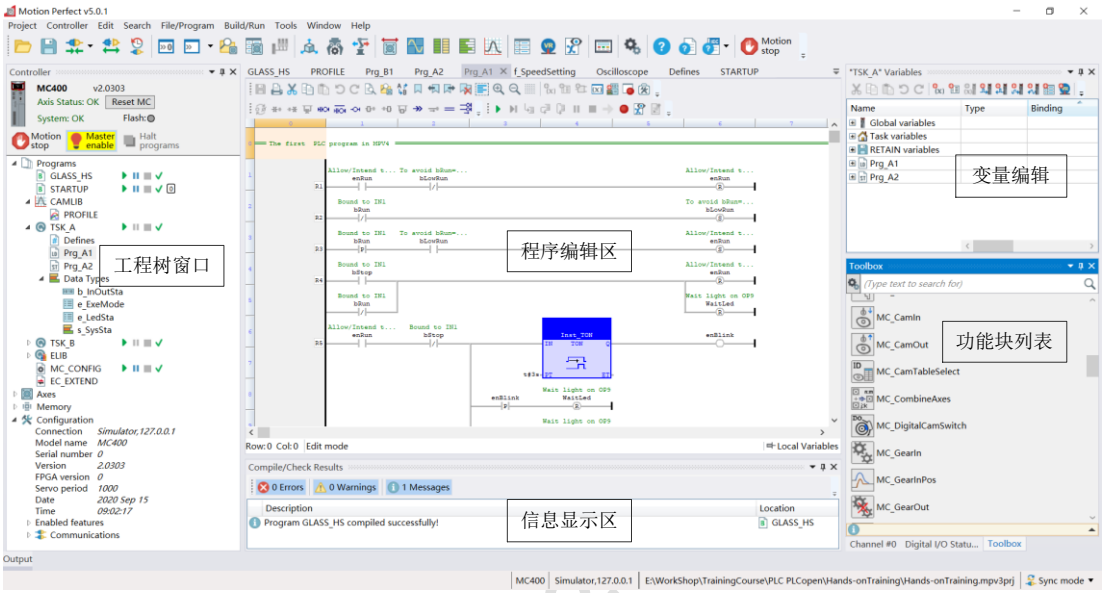


图 7-1 Motion Perfect V5 中的 IEC61131-3 主界面

Motion Perfect V5.0 中集成了 IEC61131-3 编程环境，在此环境中可以创建 IEC 工程文件，使用 4 种编程语言编写程序。

工程树窗口：可以添加浏览 IEC 任务和程序。

程序编辑区：可以适用特定编程语言编写、查看程序。

信息显示区：输出编译信息和错误信息。

变量编辑窗口：可以添加浏览变量。

功能块列表窗口：可以浏览选择功能或者功能块，用于程序编写。

7.2 创建 IEC61131-3 任务

IEC61131-3 标准支持任务控制，IEC61131-3 的程序通过任务进行管理。每个任务中可以包含多个 IEC61131-3 的程序。在 Trio 运动控制器中，同一工程中，IEC61131-3 的任务可以和 TrioBasic 程序协同工作。两者的数量总和受限控制性能约束，可以在工程树窗口中的 Memory->Programers 中查找到当前控制器允许的最大数量限制。

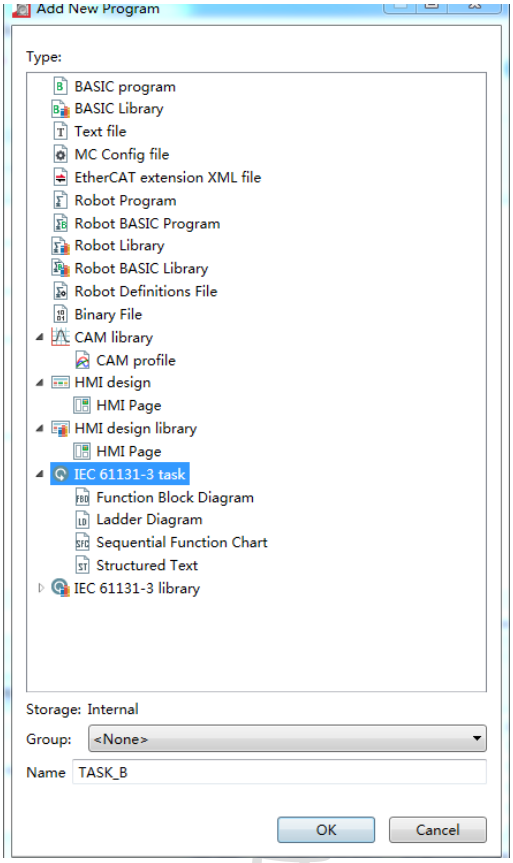


图 7-2 IEC61131-3 的任务创建

每个 IEC61131-3 任务下可以创建多个程序，理论上最大程序数量为 32767。Motion PerfectC V5 提供四种语言编程（SFC 顺序功能图、LD 梯形图、FBD 功能块图、ST 结构化文本语言）进行程序的设计。创建程序时，需要选择执行类型为程序（Main Program）。

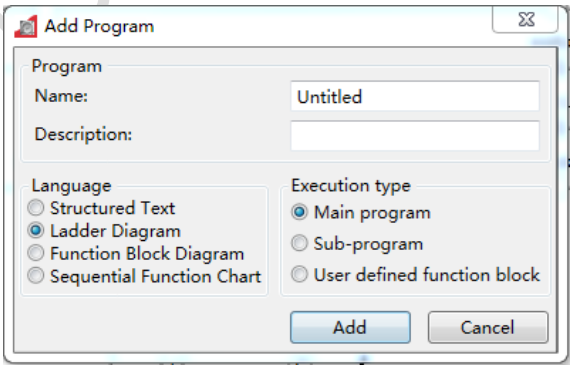


图 7-3 IEC61131-3 程序的创建

任务中的程序默认以创建先后被顺序执行，用户可以对任务下的程序执行顺序进行管理、调整（任务的右键菜单“程序执行顺序”的窗口中设置）。任务在运行时，会根据设置的程序执行顺序逐个程序运行。

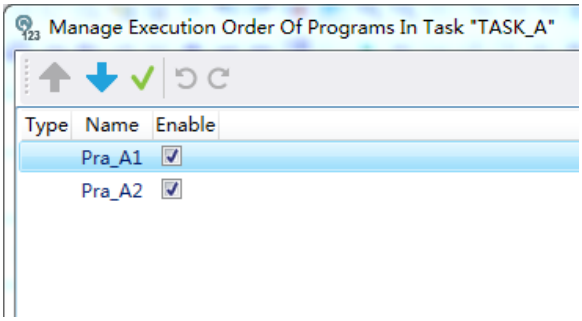


图 7-4 任务中的程序执行顺序设置示例

在 Motion Perfect V5.0 中 IEC61131-3 任务需要设置两个属性：任务优先级和任务的执行模式。任务优先级根据符合 Trio 运动控制器特点，具有快速任务（两个最大值的进程号）和慢速任务之分。

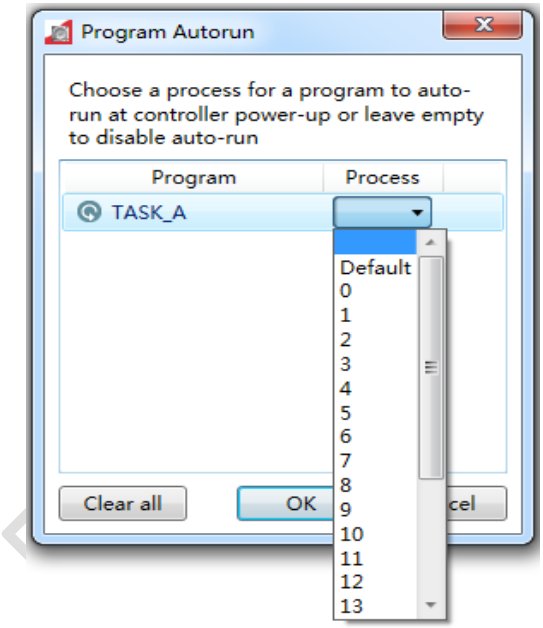


图 7-5 任务优先级/进程号的设置

任务的执行模式，目前 Trio IEC 环境可以支持任务周期的设置。任务的周期设置最小值为 1ms。任务的执行还支持“Run as fast as possiable”充分利用控制器的资源，完成一次任务的执行后，立即重新开始另一个周期的执行。

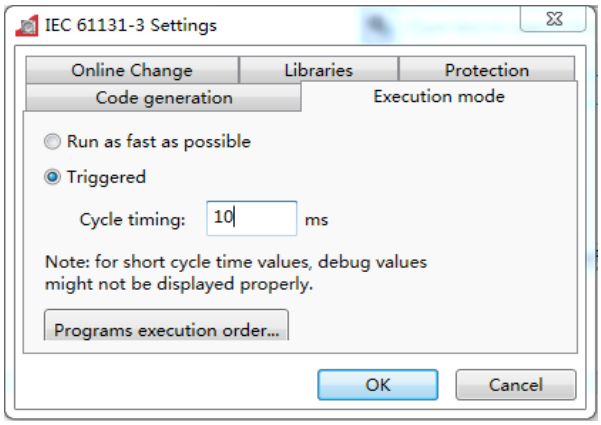


图 7-6 任务的执行模式设置

7.3 数据类型

Motion Perfect V5 的 IEC61131-3 支持的基础数据类型如表 7-1。

表 7-1 IEC61131-3 支持的基础数据类型

数据类型	描述
BOOL	布尔型，（值为 FALSE 或 TRUE），使用一个位存储。
SINT	8 bits 有符号整型 (- 128 ~ + 127)
USINT	8 bits 有符号整型(0~ + 255)
BYTE	单字节型，同 USINT
INT	16 bits 有符号整型(- 32768 ~ + 32767).
UINT	16 bits 无符号整型(0 to + 65535).
WORD	双字节型，同 UINT.
DINT	32 bits 有符号整型 (- 2147483648 ~ + 2147483647).
UDINT	32 bits 无符号整型 (0 ~ + 4294967295).
DWORD	双字类型，同 UDINT.
LINT	64 bits 有符号长整型.
REAL	32 bits 单精度浮点类型
LREAL	64 bits 双精度浮点类型
TIME	一天的时间（24 小时），精度为 1ms
DATA	日期，包含年、月、日。
STRING	在声明的最大长度范围内可变长度的字符串
	最大字符长度为 255

Motion Perfect V5 的 IEC61131-3 标准的开发环境支持复杂数据类型的定义和使用。且支持这些用户自定义数据类型的导入、导出，实现复用。

用户自定数据类型的编辑窗口，可以在工程树窗口中选择 Program->IEC_Task->Open IEC61131-3 Types 或程序编辑区的顶端快捷图标区打开用户数据类型定义窗口。

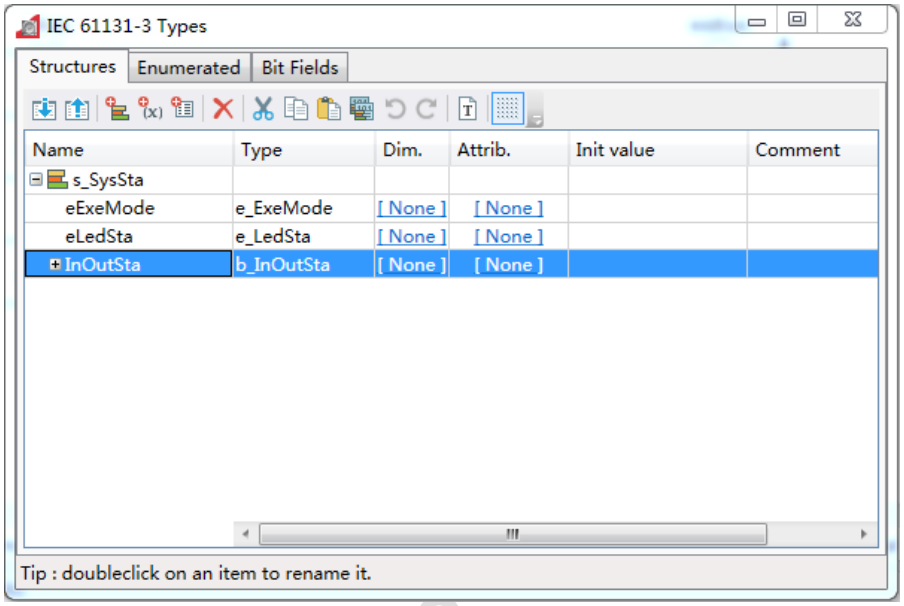


图 7-7 用户数据类型定义窗口

Motion Perfect 编程环境里可以创建三种用户数据类型，Structures 结构体、Enumerasted 枚举类型，Bit Field 位域类型，每种数据类型含义参见 IEC61131-3 标准定义。

7.4 创建变量

变量的创建，可以支持多种方式进行创建：1）在工程树窗口选择某个 IEC_XX 任务，打开 Open IEC 61131-3 Variables 变量编辑窗口； 2）通过程序编辑区的顶端的快捷图标启动变量的创建； 3） 在程序编辑时，直接输入变量名并回车即可进行新的变量创建。

变量分为局部变量、任务变量和全局变量。三种变量在创建页面中的作用域来设定变量的类型。局部变量选择其所属的程序/功能块/Sub-program 的名称。变量定义时，会同时设置变量的各个属性。包括 Name 变量名、Types 变量的数据类型、Where 变量的作用域选择、Binding 变量的绑定对象、InitValue 变量的初始值设置、Dim 变量的是否为数组（最多支持 3 维数组），Attrib 变量的读写属性以及 Comment 变量的注释。

ESTUN &

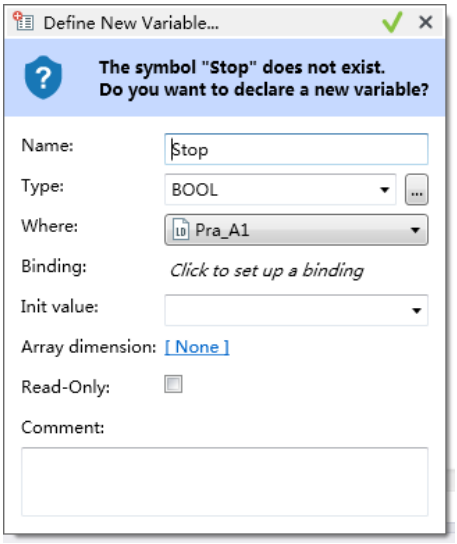


图 7-8 变量及属性定义页面

变量的绑定，可以将变量与控制器配置的特定资源关联起来。可支持如图 7-9 所示的资源绑定。

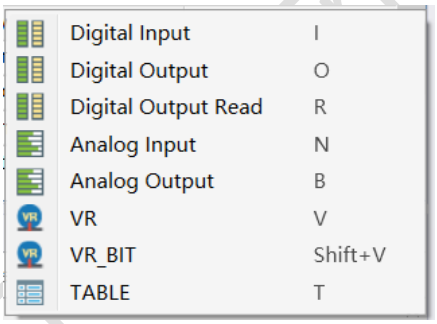


图 7-9 变量可绑定的资源对象

三种类型变量的作用域各不相同，局部变量只能在本程序中被访问，任务变量只能被任务里的程序访问，全局变量可以被所有任务里的程序访问。变量创建后，将显示在变量表(如图 7-10)里。所有定义的属性可以通过变量表进行查看和修改，实现对变量的管理。

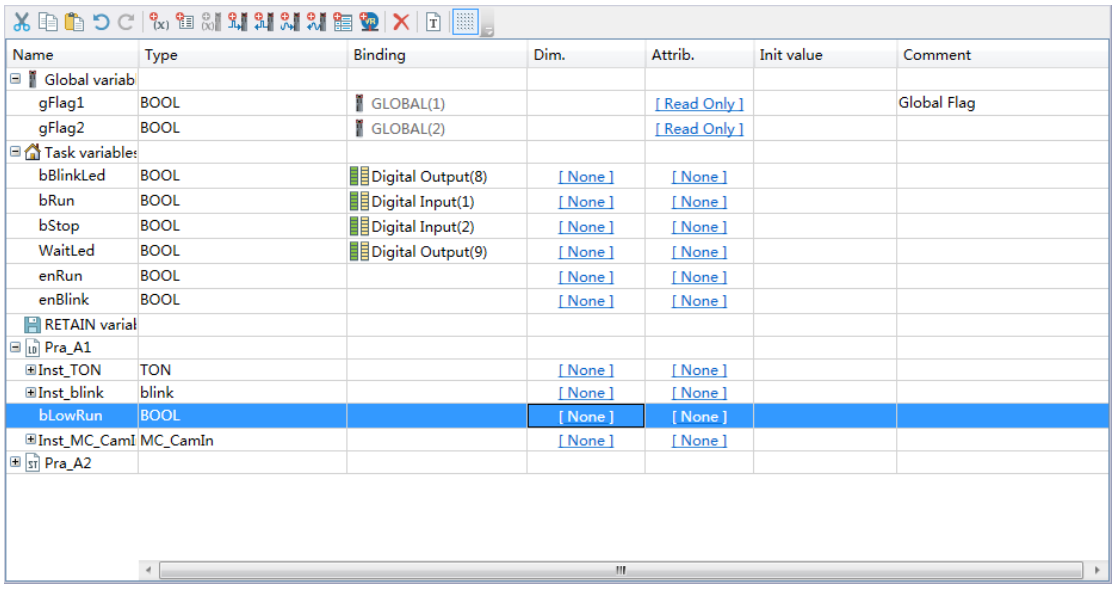


图 7-10 IEC61131-3 变量管理表

变量表可被切换为文本方式。可以使用文本方式进行变量的查看与编辑。文本方式支持选定的程序、任务变量或在特定程序的 Local 变量区域的切换。

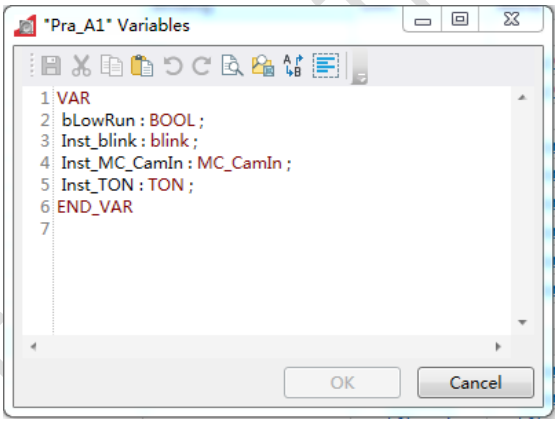


图 7-11 文本方式的变量管理

7.5 创建用户功能库

IEC61131-3 标准为实现模块化的设计支持，Motion Perfectv V5 通过了基本的函数库、功能块库，还支持用户创建自定义的功能、功能块和对应的 IEC61131-3 的库。实现模块化的设计和技术的积累和复用。

库的创建，可通过 Program->New->IEC 61131 Libarary 创建自己的 IEC 库，接着在 IEC 库中就可以创建 User Define Function Block 用户定义功能块或者 Sub-Program 子程序。创建的库可以通过导入导出，实现共享、复用。

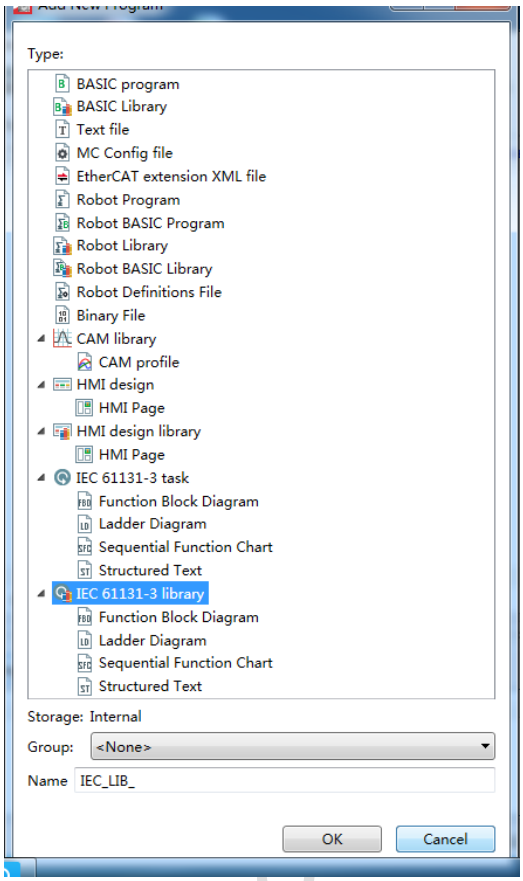


图 7-12 IEC61131-3 库的创建

在 IEC61131-3 的库中添加 IEC61131-3 的程序，选择程序的开发语言与执行类型（Sub-program：功能， User defined function block:用户自定义功能块），实现功能和功能块的创建。

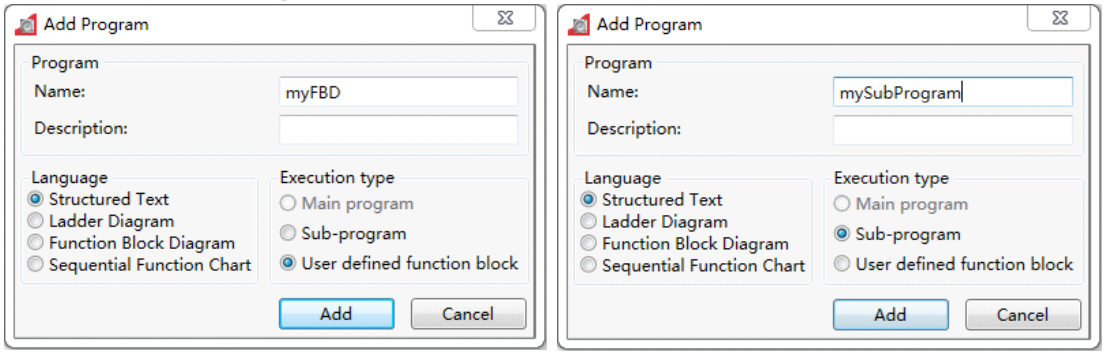


图 7-13 功能、功能块的创建

功能（函数）和功能块的主要区别在于，功能（函数）的给定的输入，输出是确定的，与实现顺序无关。自定义功能与功能块的过程和方法一样，在使用时，功能（函数）不需要被实例化，而功能块是需要被实例化的。

定义好的用户功能库可以保存为文件提供给他人使用，同时可以添加密码保护。使用功能库时首先要把功能库加载到工程文件中，然后在需要使用功能库的任务中添加后方可使用。

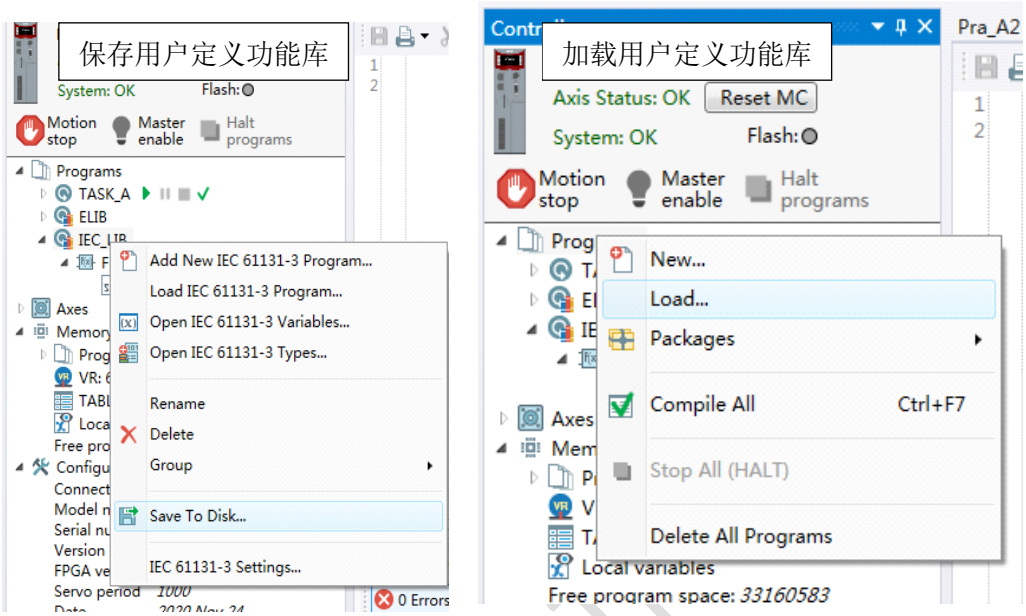


图 7-14 功能块库的导出与加载

功能块库被添加到工程后，若在任务中需要使用，需要在对应的任务中，添加所需的自定义功能块库。被添加的自定义功能块库可以再 Toolbox 工具盒中显示，同已有的标准功能/功能块的使用。自定义的功能/功能块显示为灰色。

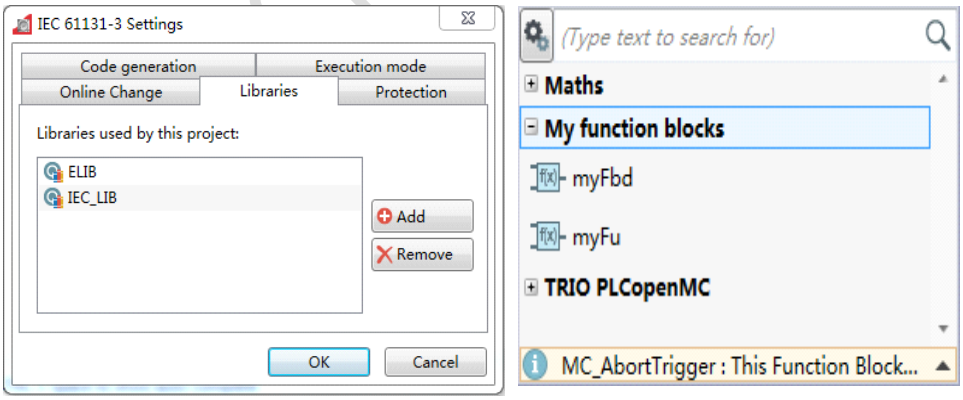


图 7-15 向 IEC61131-3 的任务中添加自定义功能块库

7.6 选择编程语言

Motion Perfect V5.0 支持 IEC 61131-3 定义的四种编程语言：

- 梯形图(LD)

- 结构化文本 (ST)
- 功能块图 (SFD)
- 顺序功能图 (SFC)

每种编程语言有自己的语法规则，编写程序时根据项目需要或者程序开发者的知识背景选择。Motion Perfect V5 的 IEC61131-3 的任务可以有多个程序组成，不同程序可以采用不同的编程语言实现。功能和功能块也可以使用不同的语言实现。



图 7-16 不同编程语言的程序编辑页面

在创建 POU（功能、功能块、程序）时，选择该 POU 的编辑语言。对于已经创建的 POU，可以选择 POU 的右键菜单，实现不同编程语言的切换。

不同语言的语法可以参考 IEC61131-3 标准相关资料，也可以参考 Trio 的 IEC61131-3 的帮助文件。对于使用到的标准库文件，对于功能/功能块的具体功能和使用，请查阅 IEC61131-3 的帮助文件。

7.7 Defines 的应用

在每创建一个 IEC61131-3 的任务时，都会在每个新创建的任务中添加一个名为 Defines 的文件。该文件只支持结构化文本 ST 进行编写。该文件在任务运行时，被首先执行一次。可以用作任务中程序的初始化和一些宏的定义。

7.7.1 预处理指令

Motion Perfect V5.0 支持 Defines 预处理指令，可以用它定义一个记号替代一个常数或字符串。

Defines 预定义指令作用域分为两级：

全局 Defines，作用于同一个任务下的所有程序

局部 Defines，只作用于一个程序

全局 Defines:在每个任务程序列表的顶端都有一个 Defines 文件，全局 Defines 可以在这个文件中声明，它作用于该任务下的所有程序，如下图。

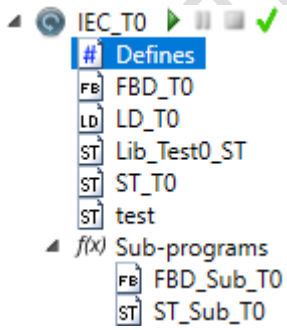


图 7-17 Defines 文件自动创建在任务中

局部 Defines:在程序内部定义只作用于本程序。

Defines 语法：

#define Identifier Equivalence ,举例如下：

#define OFF FALSE (* redefinition of FALSE constant *)

#define PI 3.14 (* numerical constant *)

#define ALARM (bLevel > 100) (* complex expression *)

注意 ：任务中的用户库将使用全局 Defines 进行编译。

7.7.2 条件编译指令

Motion Perfect V5.0 支持在 ST、 LD、和 FBD 编程语言中使用条件编译指令。

语法如下：

```
#ifdef CONDITION
    statementsYES...
#else
    statementsNO...
#endif
```

如果 CONDITION 被#define 定义，那么 statementsYES 部分的代码将被编译否则编译 statementsNO 部分代码。

在 ST 语言中，#define 语句独占一行。

在 FBD 语言中，#define 使用 break 网络分隔线进行声明

在 LD 语言中，#define 使用 comment lines 网络注释线进行申明。

7.7.3 声明异常处理程序

编译器允许你编写异常处理程序处理系统异常事件，异常事件包含：

- Startup (首次循环开始前)
- Shutdown (末次循环结束后)
- Division by zero（除零异常）
- Array index out of bounds（数组访问越界）

7.7.3.1.1 Startup

你可以编写一个异常程序在任务首次循环前执行，且只执行一次，使用方法如下：

- 在任务里创建一个主程序，用于异常处理，不可以使用 SFC 语言编辑。
- 在任务全局 Defines 文件中使用如下语句，声明调用哪个程序。

```
#OnStartup ProgramName
```

注意：因为任务的首次循环运行由于需要执行 Startup 异常程序，所以首次循环执行的时间将大于任务正常循环执行时间。Startup 声明的异常程序无法添加断点进行调试。

7.7.3.1.2 Shutdown

你可以编写一个异常程序在任务末次循环后执行，且只执行一次，使用规则如 Startup，这里不再详述。

```
#OnShutdown ProgramName
```

ESTUN & Trio

7.7.3.1.3 除零异常

编写处理除零异常事件的程序方法如下：

- 创建一个没有任何参数的子程序处理异常。
- 在任务 Defines 文件中添加 #OnDivZero SubProgramName

子程序中可以进行如下操作：

- 不做任何操作，这时系统将发出错误信息，将使用系统最大值代替零进行除运算，任务继续运行。
- 调用 FatalStop 功能，实时系统将立刻停止进入 Fatal Error 模式。
- 调用 CycleStop 功能，实时系统完成当前程序后进入 cycle setting 模式。

除零异常处理功能也可以在 DEBUG 模式下使用，用来定位错误，可以在异常中设置断点，定位错误源。

7.7.3.1.4 数组访问越界

编写处理数组访问越界的异常程序，方法如 Division by zero

#OnBadArrayIndex SubProgramName

提示: 如果在任务设定中"Check array bounds" 编译条件被选中，那么实时系统在调用异常处理后将进入 "fatal error"模式。

ESTUN & Trio

8 PLCopen MC 库

8.1 PLCopen 规范概述

PLCopen 组织应用户对运动控制软件标准化的强烈需求，于 1996 年建立运动控制规范工作组。工作组在 IEC61131-3 为基础的编程环境下，在运动控制软件的开发、安装和维护等各个阶段，协调不同的编程开发平台，建立标准的运动控制功能块库。其目的：

- 让运动控制软件的开发平台独立于运动控制的硬件
- 让运动控制的软件具有良好的可复用性
- 让运动控制软件在开发、安装和维护等阶段，都能满足运动控制功能块的标准化要求。

由于 IEC61131-3 具有良好的结构化编程支持，其软件模型、结构化文本语言和顺序功能图语言为结构化奠定基础。结构化编程技术为软件的可靠性、可维护性和广泛的实用性提供了本质保障。因此 PLCopen 标准在 IEC61131-3 的基础上，为机械部件的运动控制提供一种良好的架构。

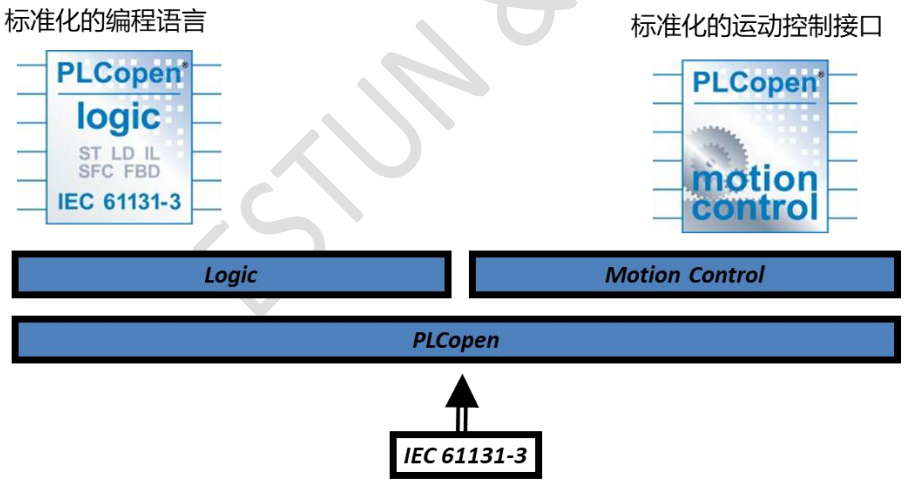


图 8-1 PLCopen 运动控制规范与 IEC61131-3 的关系

近年来，运动控制功能与 PLC 功能的融合日趋明显。IEC61131-3 与 PLCopen 的运动控制规范的紧密结合，提供了理想的机电一体化的解决方案。

PLCopen 运动控制规范的组成

- 第一部分： 基本运动控制功能库
- 第二部分： 扩展指令集，提供一系列附加功能块
- 第三部分： 用户指导，便于用户对于 PLCopen 的理解和应用
- 第四部分： 多轴协调和插补运动，增加了轴组状态机

- 第五部分： 回原点功能，描述了多种回原点过程的模型
- 第六部分： 液压扩展部分

运动控制器厂商也越来越多的支持 PLCopen 运动控制规范，现已成为国际公认的事实上的运动控制标准。

PLCopen 运动控制规范主要关注不同供应商提供的不同系统间的软件可复用性，同时还具备以下特性：

- 易用性。编写应用程序方便，安装和维护容易。
- 效率性。编程效率高，表现为具有足够数量且经验证的功能块，很大程度上提高了编程的效率；
- 一致性。符合 IEC61131-3 标准的结构化和模块化。
- 通用性。软件独立于硬件。
- 完全性。非强制性，但很充分。

PLCopen 运动控制功能块仅定义到接口和数据结构的层面，功能块的具体实现有各自厂商自行定义。用户可以再 PLCopen 运动控制规范规定的基本功能块组合的基础上，以任务为导向建立具有知识产权的独特功能块/功能块库，可极大地提高软件的可复用性，尤其能体现自身的特色和竞争差异性。采用不同功能块/功能块库的组合，可以快速形成差异化的产品序列，满足众多应用场合的需求，同时这些软件还独立于硬件系统，在更换硬件系统的时候，可以将软件的移植成本压缩到最小。因此，以这种标准化的方式实现的运动控制解决方案，还可以减少维护和培训成本。

目前，PLCopen 运动控制规范定义的功能块主要可分为用于单轴或多轴简单系统运动的第一/第二部分，以及多轴协调运动控制的第四部分。

ESTUN

8.2 PLCopen 运动控制功能块

PLCopen 运动控制功能块的定义分为两个部分：激活部分和状态相关部分。

激活功能块有两个选项：Execute/Done 和 Enabel/Valid。“Execute”输入激活的功能块又称为边沿触发（Edge Triggered）功能块，边沿（上升沿）触发的功能块需要利用 PLC 的当前周期和前一周期判断处理。“Enable”输入激活的功能块又称为电平控制（Level Controlled）功能块，实现在一个扫描周期更新一次功能块的输出状态。

8.2.1 边沿触发功能块

Execute/Done：执行一个输入，触发功能块的执行；用相关的输出 Done（或 InVelocity、InGear、 InTorque 或 InSync）表示功能块执行完成。

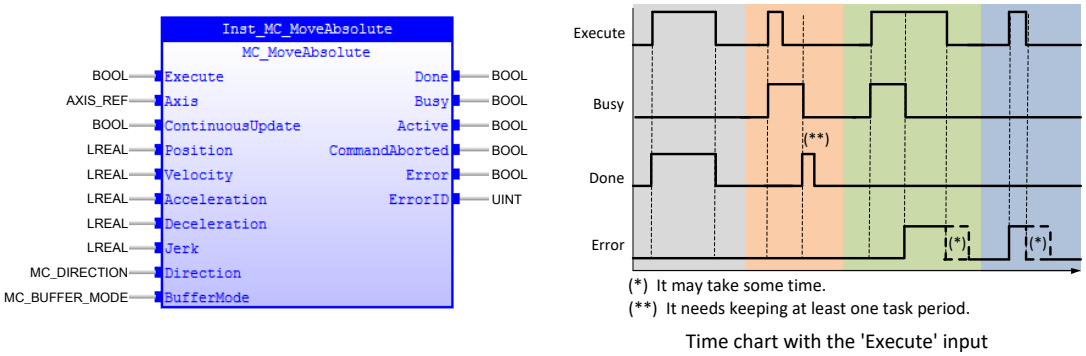


图 8-2 Execute/Done 激活功能

若使用 Execute/Done，输入“Execute”使用上升沿触发功能块。“Execute”的下降沿将复位状态输出信号“Done”（或 InVelocity, InGear, InTorque, InSync）、“Error”、“ErrorID”和“CommandAborted”。但“Execute”并不会停止或影响已经激活的功能块的执行。即使在功能块完成其功能之前复位已经发生，也必须保证对应的输出能够至少一个周期内保持其设定的状态。如果某个功能块的实例在它完成之前有收到一个新的执行触发，功能块不必进行任何反应。

有关边沿触发功能块的一些规则定义如下:

1）使用输入变量“Execute”作为功能块的激活信号，该输入是边沿触发功能块的特征。

2）在输入变量“Execute”的上升沿被检测的时刻，将激活功能块的执行（并不一定立即开始执行）。

3）在输入变量“Execute”的上升沿被检测的时刻，所有输入（除“Execute”和中止信号,如“xAbort” 外）被存储，这表示这些输入的后期变化将不会影响功能块的操作执行（未使能 ContinuousUpdate 功能为前提）。

4）在输出变量“Busy”为 TRUE 时，输入变量“Execute”切换为 FASLSE，不会使所定义的操作失效。所激活的操作将正常运行。

5）对于有中止功能块运行输入变量（为了叙述方便，将其命名为“Abort”）的功能块，当 “Abort”为 TRUE 时，所定义的功能块的操作被取消。即使“Execute”和“Abort”同时为 TRUE，功能块会立即进入中止状态。

6）功能块的输出状态信号具有互斥特性。在同一时刻输出变量“Done”、“Busy”、“Error”、“CommandAborted”中，只能有一个输出为 TRUE。

7）在设置输出变量“Busy”为 FALSE 后，如果检测到中止信号输入，输出变量“CommandAborted”将被置为 TRUE.

8）在一次主动的复位请求后，且输出变量“Done”、 “Error”、“CommandAborted”中有任一个其状态为 TRUE 之后，输入变量“Execute”再次设置为 TRUE,功能块将再次启动器定义的操作执行。

ESTUN & Trio

8.2.2 电平控制功能块

Enable/Valid：赋予一个对电平敏感的输入有效，同时 Valid 作为有效执行的输出状态指示。

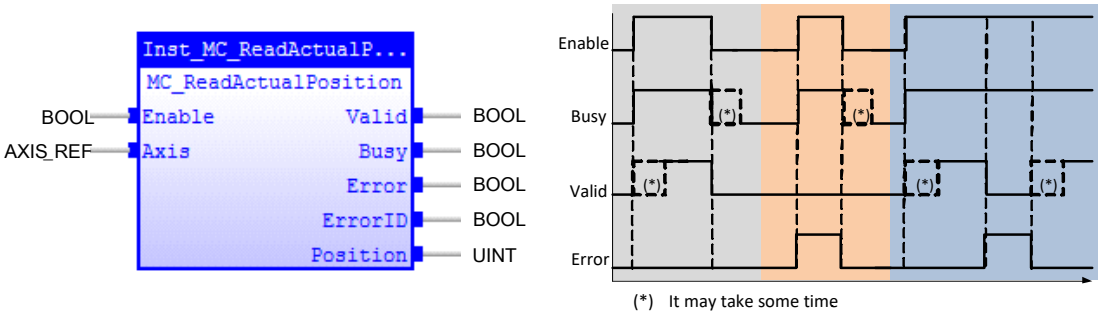


图 8-3 Enable/Valid 激活功能

若使用 Enable/Valid 时，在输入“Enable”的下降沿，输出“Valid”、“Enabled”、“Busy”、“Error”和“ErrorID”，会尽快地复位。

有关电平控制功能块的一些规则定义如下:

1）使用输入变量“Enable”作为功能块的激活信号，该输入是电平控制功能块的特征。

2）当检测到输入变量“Enable”为 TRUE，激活功能块的操作执行。在输入变量“Enable”为 TRUE 期间，所定义的操作执行器功能。当检测到输入变量“Enable”为 FALSE，功能块的操作执行将被中止。

3）功能块的输出状态信号具有互斥特性。在同一时刻输出变量“Valid”、“Busy”、“Error”、中，只能有一个输出为 TRUE。

4）只要输出变量“Valid”或“Done”为 TRUE，其输出变量的状态均有效。

5）在一次有效的复位请求和输出变量“Done”或“Error”，其中有一个状态为 TRUE，输入变量“Enable”再次被设置为 TRUE，功能块将再次启动器操作执行。

8.2.3 共性特性

功能块的状态在激活和执行过程中，具有如下共性：

1） 若功能块的一次调用检测到一次启动条件（满足启动条件），输出变量“Busy”立即设置为 TRUE。

2）功能块的运行过程中，未完成操作前，输出变量“Busy”将一直保持 TRUE。

3）若功能块的操作完成，输出变量“Done”设置为 TRUE，同时输出变量“Busy”设置为 FALSE。

4）若功能块检测出错误，在“Enable”为 TRUE 期间，输出变量“Busy”被置位 FALSE，“Error”置为 TRUE，输出变量“ErrorID”输出 UNIT 类型的出错代码。

ESTUN & Trio

5）若功能块的操作可以再一次调用中全部处理，则完成状态或出错状态立即成立，那么输出状态变量“Busy”为 TRUE 的状态将不再显示出来。

6）在输出状态变量“Done”的上升沿，所有的输出变量的状态将被冻结。

7）边沿触发型模块即使在 Execute 为低时，功能也可能在运行，如果 Done、 Error 信号有输出，至少保证输出一个周期。

功能块的输出信号，反映功能块的工作状态，下面给出功能块的输出信号的定义：

- Busy：功能块尚未执行完，新的输出尚在行形成中。在输入信号“Execute”的上升沿，“Busy”置 TRUE。若输出“Done” （或 InVelocity, InGear, InTorque, InSync）、“Aborted”和“Error”中的某一个置 TRUE 时，“Busy”复位。
- Active：表示功能块已经开始执行输出数据有效。
- Done （或 InVelocity, InGear, InTorque, InSync）：表示功能块已经执行完了命令，达到目标。
- CommandAborted：当前命令已经被打断，被另一个命令替代。
- Error：功能块内已经发生了错误。
- ErrorID：UINT16 类型整数，给出 Error 对应的错误码。0 表示没有错误。错误码可以查询厂商提供的错误码列表，获取错误原因，排查错误。
- Valid：反映输出的状态是否有效，为 TRUE 时，表示输出有效。

8.2.4 ContinuousUpdate

对于 PLCopen 运动控制规范的第一部分的 V2.0 中，增加了一个输入“ContinuousUpdate”。当功能块被触发时（“Execute”上升沿），如果“ContinuousUpdate”为 TRUE，且只要它一直保持 TRUE，则功能块使用输入变量的当前值作为正在进行的运动的参数。如果功能块在被触发时（“Execute”上升沿），“ContinuousUpdate”为 FALSE，在整个运动中输入参数的变化将被忽略，使用功能块触发时的参数。

“ContinuousUpdate”的功能不会影响到功能块的功能，也不会影响到状态机（后续章节会介绍）。它仅作用在正在执行的功能块上，如果“Busy”为 FALSE 或“ContinuousUpdate”未使能时，输入参数的变化将不会作用到功能块上。

“ContinuousUpdate”不支持使用“Execute”输入进行重复触发。它只可以在功能块被中止或完成当前指令后，才能被再次使能。

ESTUN & Trio

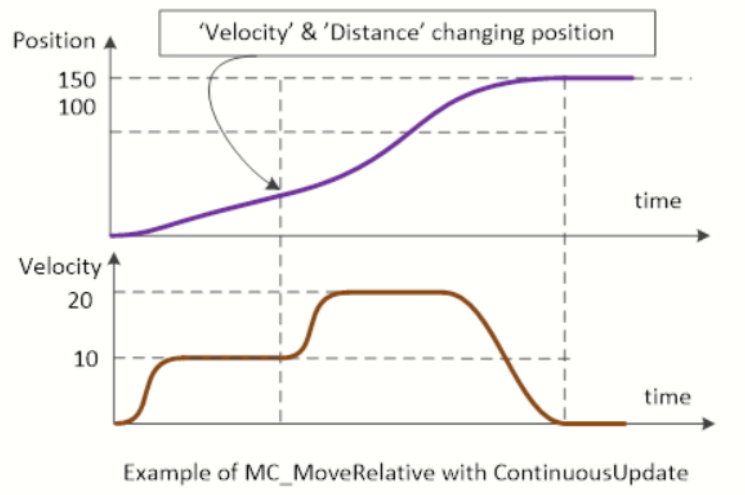


图 8-4 ContinuousUpdate 功能示例

以 MC_MoveRelative 功能块的 “ContinuousUpdate”的功能为例。在启动了功能块 MC_MoveRelative 的实例，以速度 10 运行到目标位置 100，在功能块的运行过程中，改变了目标速度至 20，目标位置至 150。其运行曲线如图 8-4 所示。从示例可以看出，在相对运动的参考位置并没有改变，仍以功能块开始运行时的位置为参考位置。

8.2.5 Re-execution

重复触发是为了将改变的输入变量作用到正在执行的功能块实例上。在 Trio PLCopen 运动控制功能块库中，对于具有输入变量“Execute”且没有输入变量“ContinuousUpdate”的运动控制功能块。

对于没有输入变量“ContinuousUpdate”的实例，其输入变量的值将会在下一个“Execute”上升沿触发，输入到功能块实例中。

对于具有输入变量“ContinuousUpdate”的实例，若“ContinuousUpdate”未使能，在实例执行过程中，输入的改变都将被忽略。

8.2.6 功能块的命名

功能块的标准化是实现运动控制功能块在单轴上额可移植性，并且为用户提供一个相似的功能的文档，包含规定的用户接口，输入输出变量和使用的参数和单位。

一个遵循 PLCopen 规范实现的运动控制功能块集合，需要包含最基本的输入输出变量，在规范文件中被标识为“B”，对于更高级别系统和未来输入输出变量自己的扩展需要，这些输出输出变量被标识为“E”。供应商增加的输入输出变量标识为“V”。这些标识标识是英文单词的首字母，意义如下：

- Basic：强制基本输入输出变量，标识为“B”

- Extended：可选扩展输入/输出变量，标识为“E”
- Vendor Specific：供应商自定义的输入输出变量，标识为“V”

PLCopen 运动控制规范定义的接口名称是明确规定的，但实际应用中可能应为特定的系统（软/硬件）的限制，对功能块名称的长度有显示，需要对命名进行缩写。且这些缩写需要在相应的文档中说明。所有的缩写名称都需要在标准的认证文档中明确说明。

如表 8-1 的一些缩写规则，是被认为符合规范的。

表 8-1 常见名称缩写

完整名称	缩写名称
Command	Cmd
Position	Pos
Velocity	Vel
Acceleration	Acc
Deceleration	Decel
Absolute	Abs
Relative	Rel
Actual	Act
Superimposed	SupImp
Additive	Add
Parameter	Par
Continuous	Cont
GearRatioDenominatorM1	RatioDenM1
GearRatioNumeratorM1	RatioNumM1

ESTUN & Trio

按上述简写规则，缩写示例：

表 8-2 名称缩写示例

完整名称	缩写名称
CommandAborted	CmdAborted
MC_MoveContinuousRelative	MC_MoveContRel
MC_ReadParameter	MC_ReadPar

8.3 Trio PLCopen MC 介绍

Trio PLCopenMC 是在 Motion Perfect 的 IEC61131-3 开发环境中添加的运动控制功能块库。该运动控制功能块库遵循 PLCopen Part1 V2.0 规范，与 IEC61131-3 的环境融合，且可以实现与 Trio BASIC 程序系统工作和通信。用户可以根据自身的知识背景和喜好，选择倾向的编程环境和语言实现应用。

8.3.1 状态图

状态图是轴的真实状态的抽象层，它着重于单轴。轴的状态将随其执行的运动指令而变化。

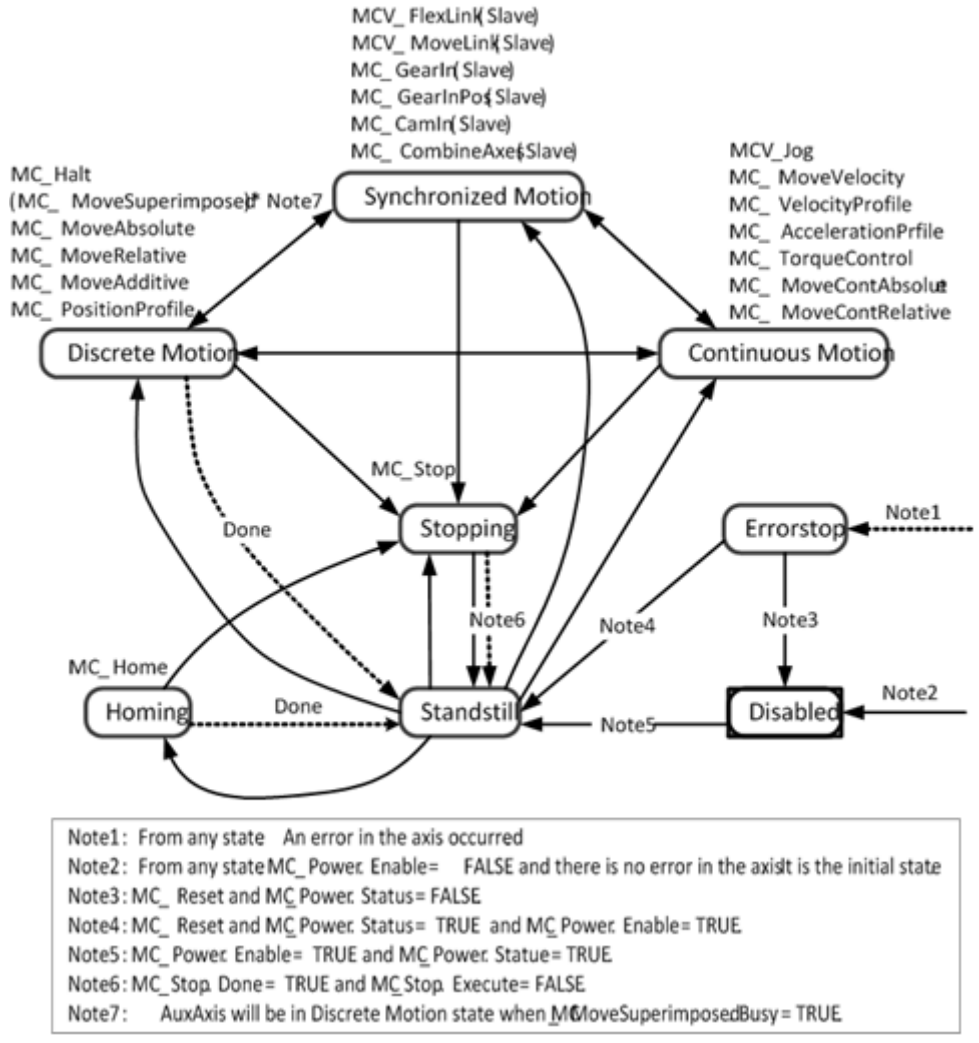


图 8-5 Trio PLCopen MC 状态图

从状态图的角度看，多轴功能块 MC_CamIn，MC_GearIn 和 MC_PhasingXX 可以看作处于特定装填的多个单轴。例如，CAM-主轴可以处于状态“持续运动（ContinuousMotion）”中，相应的从轴属于“同步运动（SynchronizedMotion）”。连接从轴对主轴的状态没有影响。

状态途中的箭头表示状态之间的可能的状态转换。状态的转换取决于图中实现箭头发出

的指令。虚线箭头用于在轴的命令已经终止或系统相关转化（如与错误相关）时发生的状态转换。在该状态图上面列出了将轴转化到相应的运动状态的运动指令。当轴已经处于相应的运动状态时，也可以发出这些运动指令。

状态图中各个状态的说明：

关闭（Disabled）状态：“关闭”状态为轴的初始状态，在该状态中，轴的运动是不受功能块的影响，轴处于“关闭”状态，轴中没有错误。如果轴在“关闭”状态下调用“Enable : = TRUE”的 MC_Power 功能块时，则状态改变为“保持静止（Standstill）”状态，在进入“保持静止”状态前，轴反馈的是可操作状态。除了轴处于“故障停止（ErrotStop）”状态外的任何状态下，使用“Enable : = FALSE”的 MC_Power 功能块，可以将轴的状态转换为“关闭”状态。轴上的任何正在运行指令都将被中止（CommandAborted）。

故障停止（ErrorStop）状态：具有最高优先级，出现轴的错误时进入该状态。只要轴的错误处于挂起状态，轴的状态将保持“故障停止”，直到错误被复位。“故障停止”的意图是使轴在可能的情况下，将轴停止。在“故障停止”状态下，轴不能接收进一步的运动指令，直至错误状态被复位。导致轴进入“故障停止”状态的原因是来自轴和轴控制的错误，而不是功能块实例自身的错误。

保持静止（StandStill）状态: 轴中没有错误，轴被使能，具备执行运行指令的条件。当使用 MC_Stop 停止轴时，只要 MC_Stop 实例的输入变量“Execute”保持为 TRUE，轴就保持在停止状态（Stopping），即使轴已经停止且输出信号“Done”为 TRUE。 “静止状态”可以通过调用指令进入运动状态。在进入“离散运动（DiscreteMotion）”和“回零（Homing）”状态后，完成指令后，即自动返回“静止”状态。对于“同步运动（SynchronizedMotion）”和“持续运动（ContinuousMotion）”状态，需要通过调用 MC_Stop 才能返回“静止”状态。

回零状态（Homing）状态: 调用 MC_Home 时，进入回零操作是的轴状态。完成回零操作后，轴自动返回“静止”状态。

停止状态（Stopping）状态: 在轴调用 MC_Stop 时，轴为停止或 MC_Stop 的输入变量“Execute”为 TRUE。

离散运动（DiscreteMotion）状态: 在完成指令运动后，轴将进入“保持静止”状态。

连续运动（ContinuousMotion）状态：通过调用特定指令，进入该状态。在此状态的轴将不会停止。除非出现使用 MC_Stop 或轴出现故障。

同步运动（SynchrionizedMotion）状态：进行主从轴同步运动的从轴将进入该状态。MC_GearOut 和 MC_CamOut 将从轴从轴“同步运动”改变到“连续运动”状态。

ESTUN eTrio

8.3.2 库的构成

Trio PLCopen MC 功能块库包含 63 个功能块，这些功能块分为两类：单轴管理类运动控制功能块和多轴管理类运动控制功能块，以及单轴运动类运动控制功能块和多轴运动类控制功能块。

表 8-3 Trio PLCopen MC 运动控制功能块分类

管理类指令		运动类指令	
单轴	多轴	单轴	多轴
MC_Power	MC_CamTableSelect	MC_Home	MC_CamIn
MC_ReadStatus	MCV_CamTableDelete	MC_Stop	MC_CamOut
MC_ReadAxisError		MC_Halt	MC_GearIn
MC_ReadAxisInfo		MC_MoveAbsolute	MC_GearOut
MC_ReadMotionState		MC_MoveRelative	MC_GearInPos
MC_ReadParameter		MC_MoveAdditive	MC_PhasingAbsolute
MC_ReadBoolParameter		MC_MoveSuperimposed	MC_PhasingRelative
MC_WriteParameter		MC_MoveVelocity	MC_CombineAxes
MC_WriteBoolParameter		MC_MoveContAbsolute	MCV_MoveLink
MC_ReadDigitalInput		MC_MoveContRelative	MCV_FlexLink
MC_ReadDigitalOutput		MC_PositionProfile	MCV_MoveLine2D
MC_WriteDigitalOutput		MC_VelocityProfile	MCV_MoveLine3D
MC_ReadActualPosition		MC_AccelerationProfile	MCV_MoveCirc
MC_ReadActualVelocity		MC_TorqueControl*	MCV_Spherical
MC_ReadActualTorque		MCV_Jog	MCV_Helical
MC_SetPosition		MCV_AddDac	
MC_SetOverride		MCV_AddAx	
MC_TouchProbe		MCV_Backlash	
MC_DigitalCamSwitch			
MC_Reset			
MC_AbortTrigger			
MC_HaltSuperimposed			
MCV_EncRatio			

* It has not implemented by now.

各个功能块的具体作用，不在此处重述。请参考 Trio PLCopen 功能块帮助文档。

8.3.3 定义的数据类型

Motion Perfect V5 支持基本的数据类型和用户自定义数据类型。在 Trio PLCopen MC 功能块图中，有如表 8-4 下一些新建的数据类型（枚举类型和自定义结构体数据类型）。

表 8-4 Trio PLCopen MC 定义的枚举数据类型

枚举类型	说明
MC_BUFFER_MODE	该枚举类型支持如下的枚举值: **mcAborting**:默认模式，无缓冲。该模式将中止当前运行运动，并立即作用到轴上，缓冲区将被清空。 **mcBuffered**: 当前一个 FB 完成时，该 FB 立即执行操作。该模式没有运动的混成。 **mcBlendingLow**: 在前一个功能块完成运动时，但轴并不停止，而是以两个功能块的目标速度较低的速度为目标速度结束。 **mcBlendingPrevious**: 实例的指令完成时的速度以其自身的目标速度为结束目标速度。 **mcBlendingNext**: 实例的指令完成时的速度以下一个功能块的目标速度为结束目标速度。 **mcBlendingHigh**: 在前一个功能块完成运动时，但轴并不停止，而是以两个功能块的目标速度较高的速度为目标速度结束。
MC_AXIS_DIRECTION	轴运动的方向选择： **mcBoth** : 正负方向 mcPositive: 正方向 mcNegative: 负方向
MC_CAMSWITCH_MODE	数字凸轮开关的工作模式: mcPosition: 位置模式 **mcTime**: 时间模式
MC_COMBINE_MODE	轴叠加的方式 : **mcAddAxes** : 将两个主轴移动位置相加后作用到从轴上 **mcSubAxes** : 将两个主轴移动位置相减后作用到从轴上
MC_DIRECTION	It supports the following enumerated value: mcPositiveDirection: 轴的正方向/增量方向

ESTUN & TRIO

枚举类型	说明
	mcShortestWay: 最短路径（旋转轴有效）. mcNegativeDirection: 轴的负方向/减量方向 mcCurrentDirection: 轴当前的运动方向
MC_HOME_MODE	回零模式： **mcForwardZ**: 编码器正向零脉冲回零 **mcReverseZ**: 编码器反向零脉冲回零 **mcForwardS**: 正向参考点开关回零 **mcReverseS**: 反向参考点开关回零 **mcForwardZS**: 编码器正向参考点+零脉冲回零 **mcReverseZS**: 编码器反向参考点+零脉冲回零
MC_SOURCE	功能块使用的控制源的值选择 (例如：位置): **mcCommandedValue**: 是基于功能块输入的值，能被用作轮廓产生器的输入 **mcSetValue**: 处于“较低”水平，靠近执行机构，是最新的值（由轮廓产生器产生），即送到伺服回路，执行机构下个要用到的值。 **mcActualValue**: 从反馈系统获取的最新的值。
MC_START_MODE	定位方式选择： mcAbsolute: 绝对定位 mcRelative: 相对定位
MC_SYNC_MODE	同步路径/方式选择: **mcShortest**: 从轴使用最短同步位置 **mcCatchUp**: 使用输入的速度追赶主轴进行同步 **mcSlowDown**: 使用节能的方式追赶主轴
MC_TRIGGER_EDGE	位置锁存的触发方式: **mcRisingEdge**: 输入信号的上升沿触发位置锁存 **mcFallingEdge**: 输入信号的下降沿触发位置锁存
MC_TRIGGER_MODE	位置锁存触发模式: **mcHardware**: 同 Trio BASIC REGIST()m 命令的 Mode:21 **mcTime**: 同 Trio BASIC REGIST()m 命令的 Mode:20

ESTUN & Trio

Trio PLCopen MC 运动控制功能块库使用到的结构体数据类型如表 8-5 所示：

表 8-5 Trio PLCopen MC 定义的结构体数据类型

结构体类型	说明
AXIS_REF	轴参考类型 STRUCT **AxisNo** : UINT; (* 从 0 开始 *) END_STRUCT
MC_CAM_REF	电子凸轮数据在 TABLE 中的位置定义 STRUCT TableStart : UDINT; TableFinish : UDINT; END_STRUCT
MC_CAMSWITCH_REF	数字凸轮开关结构体定义 STRUCT **TrackNumber** : ARRAY[0..15] of UINT; (*数字凸轮的编号*) **OutputNumber** : ARRAY[0..15] of UINT; (*数字量输出端口号，端口号的值取自控制器 MC_OUTPUT_REF 类型的定义*) **FirstOnPosition** : ARRAY[0..15] of LREAL; (*开启位置*) **LastOnPosition** : ARRAY[0..15] of LREAL; (*关闭位置*) **AxisDirection** : ARRAY[0..15] of MC_AXIS_DIRECTION; (* 位置比较输出的方向选择，支持 mcBoth **:** Both directions, mcPositive **:** Positive direction, mcNegative: Negative direction 可选*) **CamSwitchMode** : ARRAY[0..15] of MC_CAMSWITCH_MODE; (* 数字凸轮输出模式，默认位置模式。支持 mcPosition **:** Position based 和 mcTime **:** Time based *) **Duration** : ARRAY[0..15] of TIME; END_STRUCT
MC_HOME_REF	回零参数设置 STRUCT **HomeMode**: MC_HOME_MODE; (* 回零模式选择 *) **HomeSpeed**: LREAL; (*寻找参考点开关的目标速度*) **CreepSpeed**: LREAL; (* 找到参考点开关后，寻找 Z 脉冲的速

ESTUN & Trio

结构体类型	说明
	度。*) **Acceleration**: LREAL; (*加速度值*) **Deceleration**: LREAL; (*减速度值*) **Jerk**: LREAL; (*加加速度值*) **HomeSwitch**: UINT; (* 参考点开关端口号*) END_STRUCT
MC_INPUT_REF	数字量输出范围 STRUCT **FirstInput**: UINT;(* 输入端口号起始值 *) **LastInput** : UINT;(* 输入端口号结束值 *) END_STRUCT
MC_OUTPUT_REF	数字量输出范围 STRUCT **FirstOutput**: UINT; (*输出端口号起始值 *) **LastOutput** : UINT;(* 输出端口号结束值 *) END_STRUCT
MC_TP_REF	位置轮廓参数 STRUCT **TableStart** : UDINT;(* 位置轮廓值在 TABLE 中的起始值 *) **NumberOfPairs** : UDINT; (*轮廓的点数 *) END_STRUCT
MC_TV_REF	速度轮廓参数 STRUCT **TableStart** : UDINT; (* 速度轮廓值在 TABLE 中的起始值 *) **NumberOfPairs** : UINT; (*轮廓的点数 *) END_STRUCT
MC_TA_REF	加速度轮廓参数 STRUCT **TableStart** : UDINT; (* 加速度轮廓值在 TABLE 中的起始值 *) **NumberOfPairs** : UDINT; (*轮廓的点数 *) END_STRUCT
MC_TRACK_REF	位置跟踪比较输出参数 STRUCT

ESTUN & Trio

结构体类型	说明
	OnCompensation : ARRAY[0..15] of TIME; (* 跟踪通道对应的输出打开的延时补偿时间*) **OffCompensation**: ARRAY[0..15] of TIME; (*跟踪通道对应的输出关闭的延时补偿时间*) **Hysteresis** : ARRAY[0..15] of LREAL; (*为了防止输出的开关在临界位置的频繁切换输出状态，而增加的位置波动范围限制*) END_STRUCT
MC_TRIGGER_REF	位置捕获触发参数 STRUCT **Mode** : MC_TRIGGER_MODE; (* 位置捕获触发模式选择 *) **Channel**: UINT; (* 触发信号选择 *) **Source**: UINT; (*触发信号源选择 *) **Edge**: MC_TRIGGER_EDGE;; (*触发信号的边沿选择，上升沿和下降沿*) END_STRUCT

8.3.4 相关的规则说明

8.3.4.1 一般规则

1） 输入参数

“Execute”（无“ContinuousUpdate”）：参数在输入“Execute”的上升沿时被激活，修改任何参数都必须通过“Execute”的上升沿激活。

“Execute”（有“ContinuousUpdate”）：参数在输入“Execute”的上升沿时被激活，只要“ContinuousUpdate”使能期间（如何使能，参见 ContinuousUpdate 的说明），输入参数可以不断地修改。

“Enable”：参数在输入的“Enable”的高电平时，参数可以被不断修改（MC_Power 除外）。

2） 输入参数超出应用限制

模块将输出错误，输入参数不会被接受。

3） 输入参数缺失

在 Trio IEC 环境中，所有的输入参数不支持缺省，否则功能块将报错。

ESTUN & Trio

4） “Acceleration”,“Deceleration”和“Jerk”的输入

对于加速度、减速度、加加速度类型参数，如果输入值为零，功能块将使用轴配置时的最大值替代，不会输出错误。

5） 输出互斥

“Execute”输入的功能块：输出信号“Busy”、“Done”、“Error”和“CommandAborted”是互斥的。在一个功能块中，同一时刻他们中只能有一个为 TRUE。

“Enable”输入的功能块：“Valid”和“Error”是互斥的，一个 FB 中，同一时刻它们中只能有一个为 TRUE。

6） 输出状态

“Execute”输入的功能块：输出信号“Done”、“Error”、“ErrorID”和“CommandAborted”随着“Execute”的下降沿被复位。但“Execute”的下降沿不会停止和影响功能块的执行。如果这种情况发生，必须保证对应的输出至少保持一个周期。

7） “Done”输出的行为

当指令动作完成时，“Done”输出置位。

在同一个轴有多个功能块依次工作时，当在一个轴上的运动被同一个轴的另一个运动中断，且第一个功能块没有完成指令动作，第一个功能块的“Done”不会被置位。

8） “Busy”输出的行为

“Execute”输入的功能块：每一个功能块有一个输出“Busy”，反映功能块未完成预期的输出值/功能。“Busy”在“Execute”的上升沿置位，当输出“Done”、“Aborted”或“Error”置位时，“Busy”复位。

“Enable”输入的功能块：每个功能块可以输出“Busy”，以表明这个功能块正在工作，并期望输出新的值。“Busy”在“Enable”的上升沿置位，只要该功能块在执行操作，“Busy”维持置位状态。

9） “InVelocity”,“InGear”,“InTorque”和“InSync”的行为

它们的置位，表示功能块已经达到其控制目标，如达到目标速度（InVelocity），达到设定的齿轮比（InGear）。

10） “Active”输出的行为

具有缓冲的功能块（有 BufferMode 输入变量）上需要“Active”输出，以表明该功能块控制正在控制轴的运动。对于非缓冲的功能块，“Active”和“Busy”具有相同的值。

对于单轴的运动，几个功能块可能都处于“Busy”置位状态，但每次只能有一个功能块的“Active”置位。对于主从轴的多轴同步运动来说，其中与一个轴相关的多个功能块可以输出“Active”。

11） “CommandAborted”输出的行为

ESTUN &Trio

当功能块的运动指令执行中被另一个运动指令中断时，“CommandAborted”被置位。

“CommandAborted”的复位类似于“Done”。当“CommandAborted”发送时，功能块上的其他输出信号将被复位。

12） “Enable”与“Valid”

对于有“Enable”和“Valid”输入输出信号的功能块，“Valid”输出置位，表明功能块的输出信号是有效的。

只要“Eable”为 TRUE，功能块的相关输出就可以刷新。

如果功能块出现错误（“Error”置位），则输出无效（“Valid”复位）。当功能块的错误消失时，如果“Enable”仍保持为 TRUE，功能块的输出将被刷新，输出有效值。

13） “Position”与“Distance”

“Position”是坐标系内定义的值。“Distance”是相对值，是两个“Position”之间的差值。

14） 符号规则

“Acceleration”、“Deceleration”和“Jerk”总是正值。“Velocity”、“Position”和“Distance”可以是正值和负值。

15） 错误处理行为

所有的功能块都有两个输出：“Error”和“ErrorID”。

“Error”的上升沿告知在功能块执行期间发生错误。

“ErrorID”为错误标识，提供 UINT 类型的错误码（错误码参考 Trio PLCopen MC 的帮助文件），帮助用户了解错误原因，进行错误排查。

16） FB 的命名

在 Trio PLCopen MC 功能块库中，命名严格遵守 PLCopen Part1 标准的规定，包括缩写的要求。Trio PLCopen MC 功能块库的功能块分为 MC_xx 和 MCV_xx。MC_xx 是 PLCopen 运动控制规范中规定的功能块名称，而 MCV_xx 为 Trio 提供的厂商自定义的功能块。

8.3.4.2 中止与缓冲模式

在 Trio PLCopen MC 功能块库中，有些功能块具有“BufferMode”的输入，通过这个输入变量，可以设置功能块工作在“缓冲模式”和“非缓冲模式”。这些模式的区别主要在功能块在何时开始和以什么方式结束动作。

在非缓冲模式下（对应于 BufferMode := mcAborting），命令将立即作用，使之前的功能块中断，当前功能块接管轴的控制。缓冲区被清除。

在缓冲模式下，缓冲区的指令等待，之道其前面的功能块完成（“Done”或“InVelocity”或 “In Gear”…）。缓冲模式有如下几种：

ESTUN & Trio

mcBuffered: 前一个功能块指令操作完成，后一个功能块开始工作。若轴在“DescreteMotion”状态下，轴将停止后开始后一个功能块的动作。该模式没有运动的混成。

mcBlendingLow：后一个功能块在前一个功能块完成后控制轴的运动（等效于“mcBuffered”），但轴不会在运动之间停止。第一个功能块完成时的目标速度为两个功能块中目标速度较低的值。

mcBlendingHigh：后一个功能块在前一个功能块完成后控制轴的运动（等效于“mcBuffered”），但轴不会在运动之间停止。第一个功能块完成时的目标速度为两个功能块中目标速度较高的值

mcBlendingPrevious：后一个功能块在前一个功能块完成后控制轴的运动（等效于“mcBuffered”），但轴不会在运动之间停止。第一个功能块完成时的目标速度为前一个功能块的目标速度值。

mcBlendingNext：后一个功能块在前一个功能块完成后控制轴的运动（等效于“mcBuffered”），但轴不会在运动之间停止。第一个功能块完成时的目标速度为后一个功能块的目标速度值。

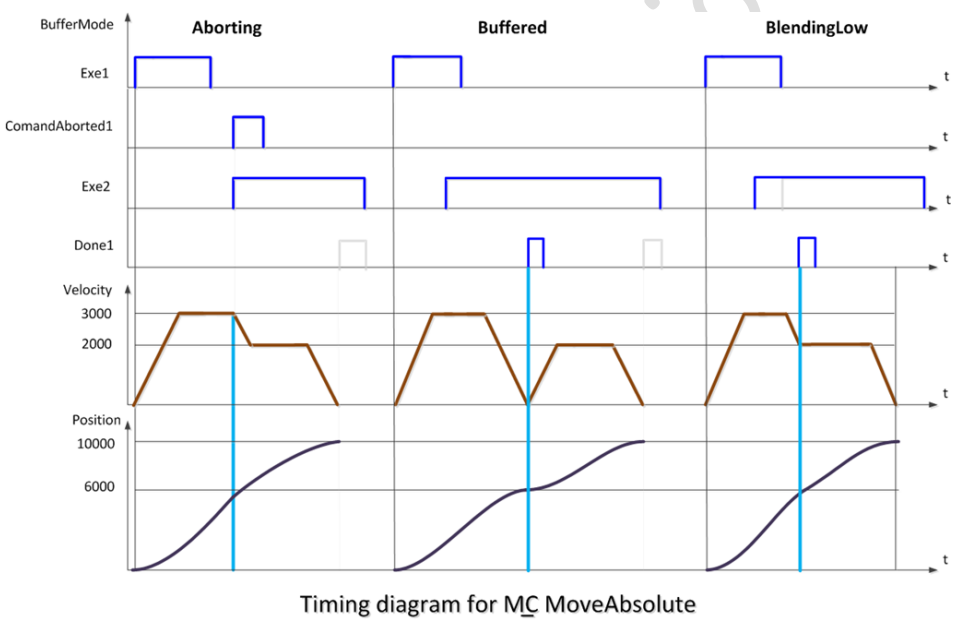


图 8-6 MC_MoveAbsolute 的不同 BufferMode 时序图

图 8-6 所示为两个 MC_MoveAbsolute 功能块实例的不同 BufferMode 的运动状态。可见，缓冲模式可以提升运行的效率和平滑性。Trio PLCopen MC 功能块支持 6 种缓冲模式，可实现运动的混成。

8.3.5 轴参数

Trio PLCopen MC 功能块的轴参数复用了 Trio BASIC 的参数。用户可以使用 Trio BASIC 的方式对轴的参数进行设置和查看，也可通过 PLCopen MC 功能块库中提供的功能块来对轴的参数进行读写操作。

参数的读写功能块：

- MC_ReadParameter：对非布尔类型的参数。
- MC_ReadBoolParameter：读布尔类型的参数。
- MC_WriteParameter：写非布尔类型的参数。
- MC_WriteBoolParameter：写布尔类型的参数。

轴的参数请参考 Trio PLCopenMC 的帮助文件。在帮助文件中，每个轴参数都被附以对应的参数号，每个参数都有相关属性的描述，包括参数的数据类型、参数的读写属性、参数的数值范围和备注信息。

参数的数据类型包括如下：

- BOOL: 布尔类型
- DINT: 32-bit 有符号整型
- REAL: 32-bit 浮点型
- INT: 16-bit 有符号整型
- LINT: 64-bit 有符号整型
- LREAL: 64-bit 浮点型
- UINT: 16-bit 无符号整型
- UDINT: 32-bit 无符号整型
- ULINT: 64-bit 无符号整型

参数的读写属性：

- R: 只读
- W: 只写
- R/W: 可读写

参数额数值范围说明了参数的设置范围，超出该范围，功能块将报错。

备注信息简要地介绍了参数的意义。

ESTUN & Trio

8.3.6 错误处理

功能块在被激活时和运行过程中都会检测输入和轴的状态是否满足指令执行的条件，有没有操作上的错误。当检测到错误时，功能块将在输出变量“Error”输出 TRUE。“ErrorID”将输出对应错误码，供用户确定错误的原因，进行错误排查。

若错误为轴的错误，轴的状态机将会切换到故障停止状态“ErrorStop”。若功能块正在

执行运动，“CommandAborted”输出置位，用户可以通过功能块 MC_ReadAxisError / MC_ReadStatus / MC_ReadAxisInfo 获取轴的具体状态和出错的原因。“AxisErrorID’”的输出反映了轴错误的原因。

帮助文件中，通过错误编号进行检索，查询错误码的错误原因。错误码以 WORD 类型的整数表示。错误的来源与功能块和轴错误。因此帮助文件中，把错误分为轴错误和非轴错误两类。

8.3.6.1 轴错误

轴错误的“ErrorID”> 0x8000，轴错误的出现会是轴的状态切换到故障状态“ErrorStop”。轴错误出现后，需要调用 MC_Reset 实例来清除错误。

8.3.6.2 非轴错误

非轴错误的“ErrorID”< 0x8000，非轴错误通常由功能块自身产生，如检测到输入不正确或用户的异常操作。

对于“Enable”输入的功能块，非轴错误的原因清除后，若“Enable”为 TRUE 的情况先，“Error”输出会自动复位。对于有“Execute”输入的功能块出现非轴错误后，若在使能了“ContinuousUpdate”功能后，出现错误，将会影响轴的运行，并清除轴的运动缓存，停止轴，进入保持静止 “Standstill” 状态。对于未使能“ContinuousUpdate”的功能块，在“Execute”的上升沿会检测输入的参数和轴的状态，若发现错误，清除轴运动缓存，停止轴。在“Execute”的上升沿的异常输入功能块予以忽略，因此不会对轴的运动产生影响。

8.4 使用 Trio PLCopen MC 功能块

Trio PLCopen MC 功能块遵循 PLCopen Part1 V2.0 规范进行开发的。其基于 Motion Perfect V5 的 IEC61131-3 的集成开发环境，为用户提供标准化的运动控制功能块，方便用户进行应用开发。其开发思想和编程方法，同 IEC61131-3 的程序开发。区别在于功能块的功能，以实现运动控制为主要任务。所以要了解 PLCopen 运动控制功能块的一些规则。

接下来，通过介绍一些运动控制应用中的内容，为用户提供一些参考。

8.4.1 轴参数的设置

轴参数的设置，是结合实际的机械部件的配置，进行设置。包括电机轴的单位、方向、安全相关参数限制、速度、加速度等内容。这些参数的设置具有多种方式可选。

由于 Trio PLCopen MC 中复用了 Trio BASIC 环境中的轴参数，因此，对于熟悉 Trio BASIC 编程的人员，可以使用 Trio BASIC 的方式对轴的参数进行初始化。

当然，也可以仅使用 IEC61131-3 的编程语言实现轴参数的设置。Trio PLCopen MC 提供了 MC_ReadParameter, MC_ReadBoolParatmeter, MC_WriteParameter 和 MC_WriteBoolParameter 功能块，供用户设置轴的参数。用户可以利用 IEC61131-3 开发环境进行编程实现。

轴参数的监控参考 2.7.2 轴参数。

8.4.2 轴状态查询

在使用 PLCopen 运动控制功能块时，对轴进行运动相关的操作，需要了解轴的当前状态。可以通过 MC_ReadStatus 功能块获取。

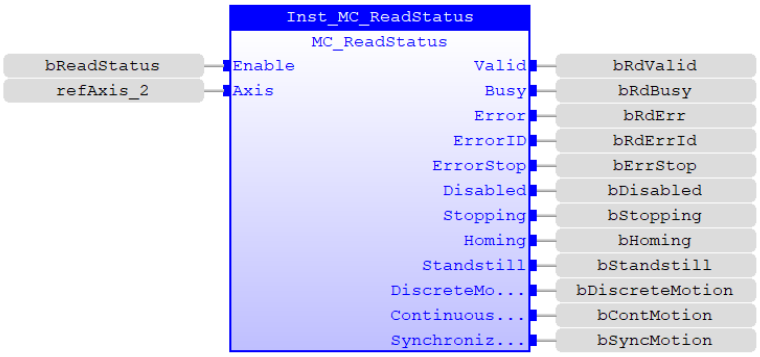


图 8-7 MC_ReadStatus 实例

在一些运动指令控制时，了解轴的状态是必要的。如使用 MC_MoveAbsolute 进行轴的定位时，对于停止的轴，需要确定轴的状态是否为保持静止“StandStill”状态，如果不在这个状态，那么就可能需要对轴进行使能（轴处于关闭 “Disabled” 状态）或做复位操作（轴处于故障停止“ErrorStop”状态）。

8.4.3 轴使能与禁能

轴的使能是启动所有运动前对轴的操作，以使其能够接受外部的指令。轴的使能/禁能通过 MC_Power 功能块进行控制。该功能块在使用时需要注意：一个轴使用一个 MC_Power 的实例来控制，不可使用一个功能块实例控制多个轴的使能/禁能。

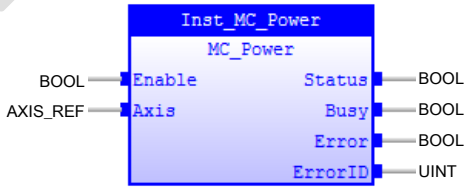


图 8-8 轴使能/禁能 MC_Power 实例

MC_Power 功能块为电平触发功能块，使能轴通常需要一些时间才能使输出变量 Status 输出。

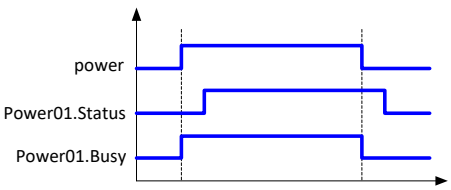


图 8-9 轴使能的时序图

8.4.4 回零

回零操作在一些应用中需要寻找运动的参考位置。通过 MC_Home 的实例，为用户提供了 6 中回零模式。

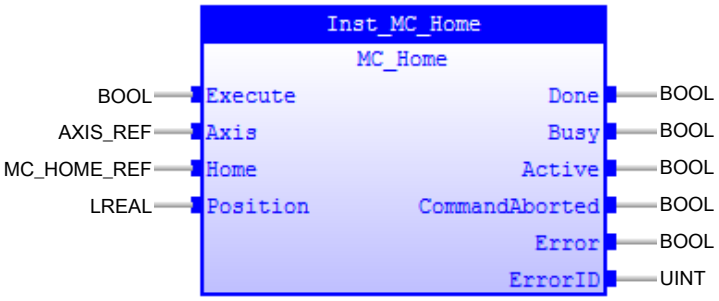


图 8-10 回零功能块 MC_Home

MC_Home 提供 6 中回零模式，各个模式动作如图 8-11 所示。

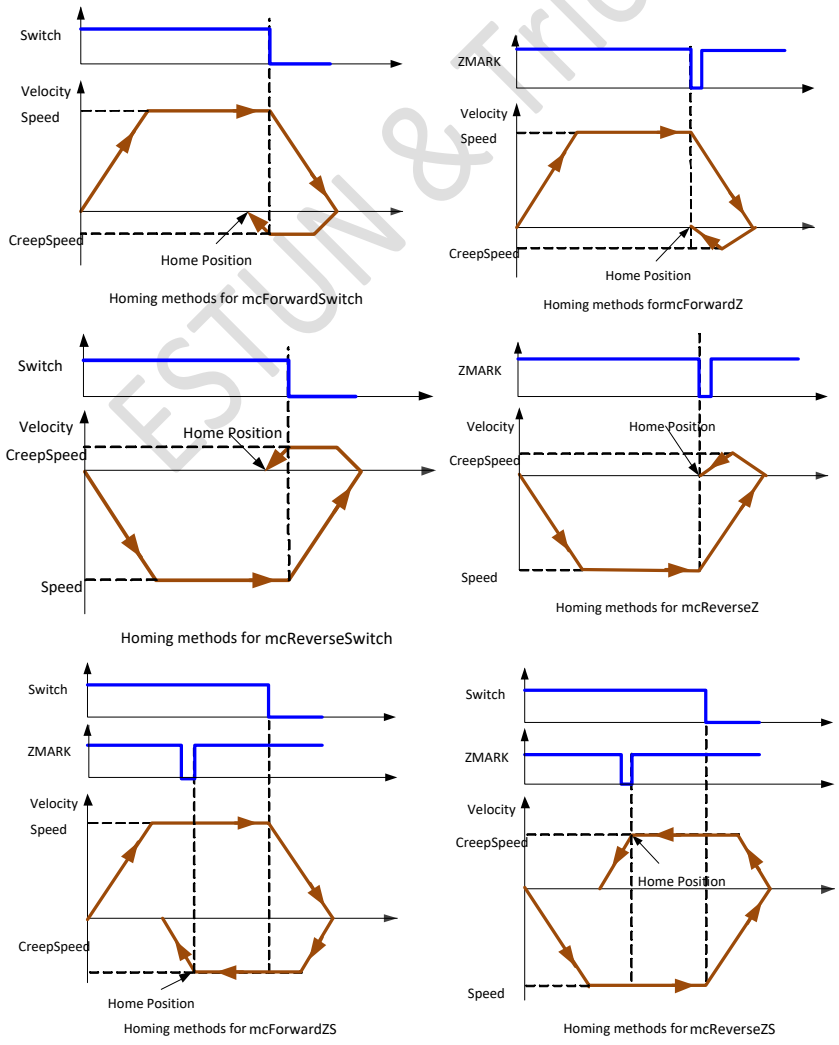


图 8-11 不同回零模式的动作

ESTUN & Trio

8.4.5 运行轴

轴的运动，对于未运动的轴，需要将轴的状态进入到保持静止状态“StandStill”，对于运动的轴需要，要保证其不在故障停止“ErrorStop”和回零“Homing”状态。以MC_MoveAbsolute为例，在完成轴参数配置，满足轴状态要求后，给实例以有效的输入，通过输入变量“Execute”的上升沿激活运动指令。

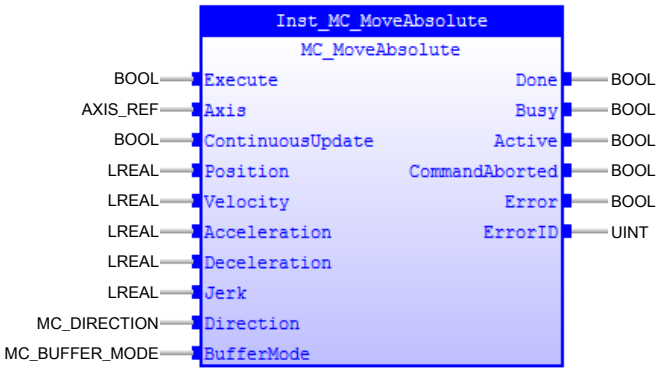


图 8-12 MC_MoveAbsolute 实例

MC_MoveAbsolute的输入变量具有速度、加速度和加加速度的设置，可以实现S型速度曲线。

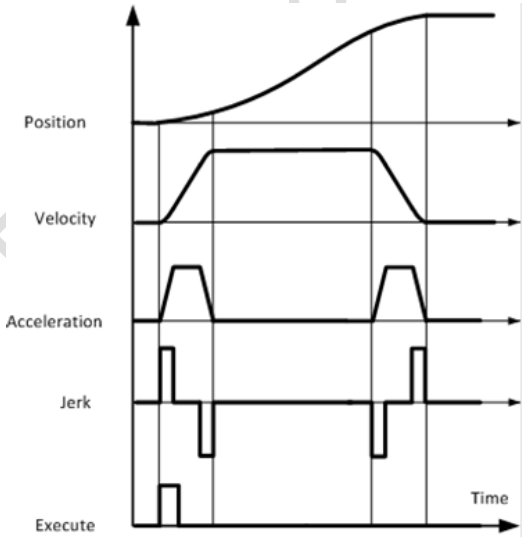


图 8-13 S 型速度曲线的定位运动

8.4.6 停止轴

停止轴的命令有MC_Stop功能块实现（MC_Halt可以实现轴的停止，但其对状态机会进入离散运动“DescreteMotion”状态，直至完成进入保持静止“StandStill”状态，详见功能块的帮助文件）。

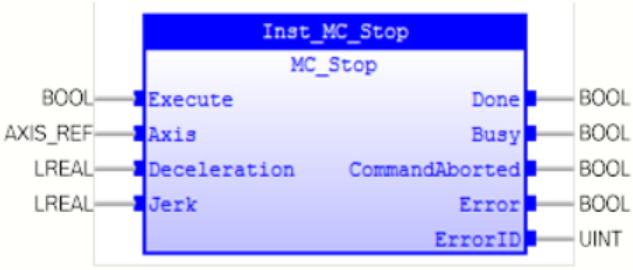


图 8-14 MC_Stop 功能块实例

在使用 MC_Stop 停止轴的运动时，会使轴的运动缓存中的指令清除。在轴停止过程中，轴处于停止“Stopping”状态。在轴停止后，若“Execute”保持为 TRUE，轴的状态将继续保持为停止“Stopping”状态。在轴停止后，若“Execute”为 FALSE，轴切换为保持静止“StandStill”状态。

8.4.7 程序示例

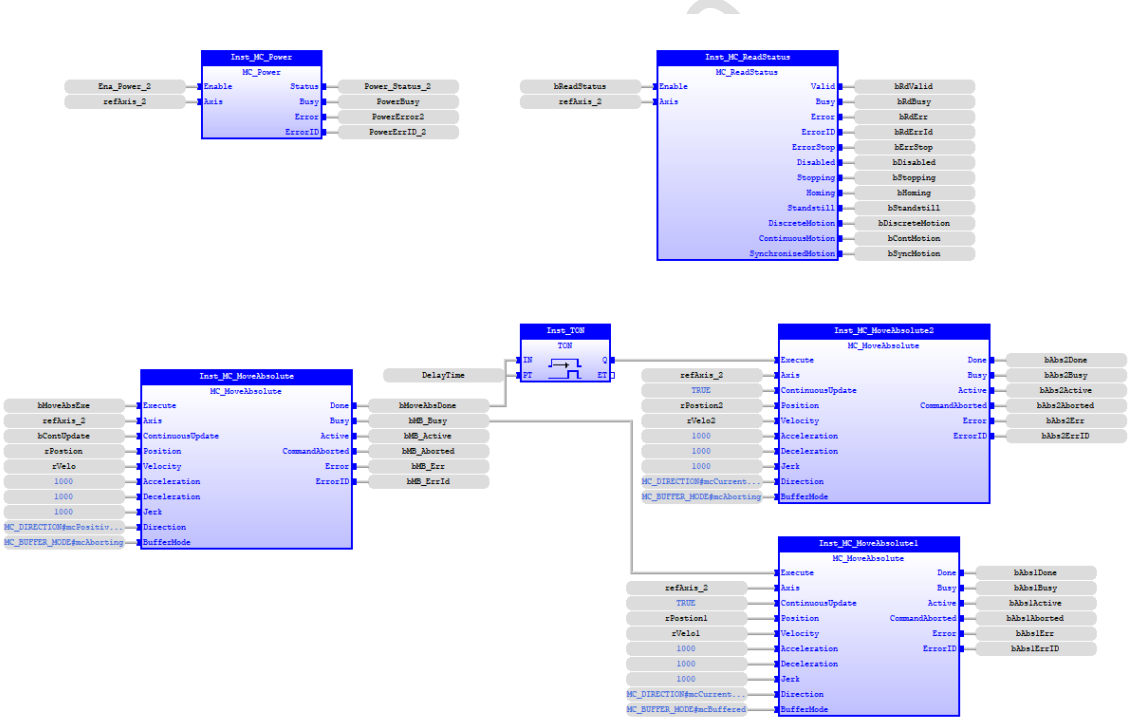


图 8-15 FBD 程序示例

程序示例说明：

refAxis_2 为轴参考变量，设置 refAxis_2.AxisNo（示例中 refAxis_2.AxisNo：=2）的操作轴号。使用 Exe_Power_2 变量操作 MC_Power 功能块实例 Inst_MC_Power，使能 refAxis_2.AxisNo 选定的轴。

bReadStatus 启动功能块实例 Inst_MC_ReadStatus，读取轴的状态机。

在轴被使能后，进入“Standstill”状态后，使用 bMoveAbsolute 启动功能块实例

Inst_MC_MoveAbsolute 的运行。Inst_MC_MoveAbsolute1 为 mcBuffered 模式的运动，Inst_MC_MoveAbsolute2 为 mcAborting 模式的绝对定位运动。

示例演示轴的使能、禁能；轴状态的查看；以及不同 BufferMode 的运行情况。

变量申明：

文本方式的变量申明如下：

```
VAR
    Inst_MC_Power : MC_Power ;
    bMB_Active : BOOL ;
    bMB_Aborted : BOOL ;
    bMB_Err : BOOL ;
    bMB_ErrId : UINT ;
    Inst_MC_MoveAbsolute1 : MC_MoveAbsolute ;
    Inst_MC_MoveAbsolute2 : MC_MoveAbsolute ;
    bAbs1Done : BOOL ;
    bAbs1Busy : BOOL ;
    bAbs1Active : BOOL ;
    bAbs1Aborted : BOOL ;
    bAbs1Err : BOOL ;
    bAbs1ErrID : UINT ;
    Inst_TON : TON ;
    bAbs2Done : BOOL ;
    bAbs2Busy : BOOL ;
    bAbs2Active : BOOL ;
    bAbs2Aborted : BOOL ;
    bAbs2Err : BOOL ;
    bAbs2ErrID : UINT ;
    bMB_Busy : BOOL ;
    bContUpdate : BOOL ;
    bMoveAbsDone : BOOL ;
    bErrStop : BOOL ;
    bRdValid : BOOL ;
    bRdBusy : BOOL ;
    bRdErr : BOOL ;
    bRdErrId : UINT ;
    bMoveAbsExe : BOOL ;
    bDisabled : BOOL ;
    PowerError2 : BOOL ;
    PowerErrID_2 : UINT ;
    PowerBusy : BOOL ;
    bHoming : BOOL ;
    bStandstill : BOOL ;
    bDiscreteMotion : BOOL ;
    bContMotion : BOOL ;
    bSyncMotion : BOOL ;
    Inst_MC_MoveAbsolute : MC_MoveAbsolute ;
    Power_Status_2 : BOOL ;
    bStopping : BOOL ;
    Inst_MC_ReadStatus : MC_ReadStatus ;
    Ena_Power_2 : BOOL := FALSE ;
    bReadStatus : BOOL := FALSE ;
    rPostion : LREAL := LREAL#1000 ;
    rVelo : LREAL := LREAL#20 ;
    rPostion1 : LREAL := LREAL#2000 ;
    rVelo2 : LREAL := LREAL#30 ;
    rVelo1 : LREAL := LREAL#50 ;
    rPostion2 : LREAL := LREAL#5000 ;
    DelayTime :  :TIME := time#10s ;
    refAxis : lib:AXIS_REF := (AxisNo := 2)  ;
END_VAR
```

变量可以通过变量表格进行维护，这里以文本方式展示。

PC Motion 篇

近年来，由于工业生产的控制系统随着自动化水平的提高而变得越发复杂，很多时候需要借助 PC 端优秀的图形界面、强大的业务逻辑处理能力，以及高级编程语言的通用性、重用性、灵活性等优势，完成人机交互、运动控制逻辑设计等任务。这也是目前在国内的运动控制系统中普遍使用高级编程语言进行项目开发，以及 PC-based 运动控制器比较受欢迎的主要原因。

针对客户需要选择使用高级编程语言或者其他平台，并基于 PC 进行项目的开发的需求，Trio 公司推出了 PC Motion。

9 PC Motion

PC Motion 是一种可支持多种平台、多种编程语言的插件。用户可以通过调用该插件提供的 API 接口，在 PC 端实现应用程序与 Trio 控制器的交互，从而完成信息采集、运动控制等任务。PC Motion 包含两种形式的插件：ActiveX(.ocx 控件)和 DLL(动态链接库)。

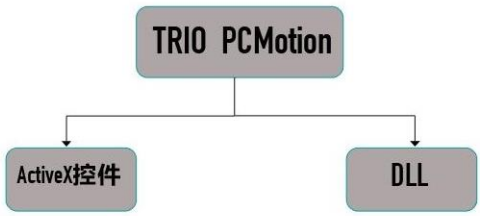


图 9-1 PC Motion 提供两种形式的插件

PC Motion 中所提供的 API 函数是基于 Trio BASIC 所提供的功能开发而来，因此，PC Motion 延续了 Trio BASIC 中功能函数以及函数的名称。在 PC 端（上位机）程序中，通过调用相应的 API 可进行读/写 IO、轴参数的设置或读取，以及实现单轴运动、多轴直线插补、圆弧插补、电子齿轮等复杂运动控制任务。PC + Trio PC Motion 的典型应用方式如图 9-2 所示。

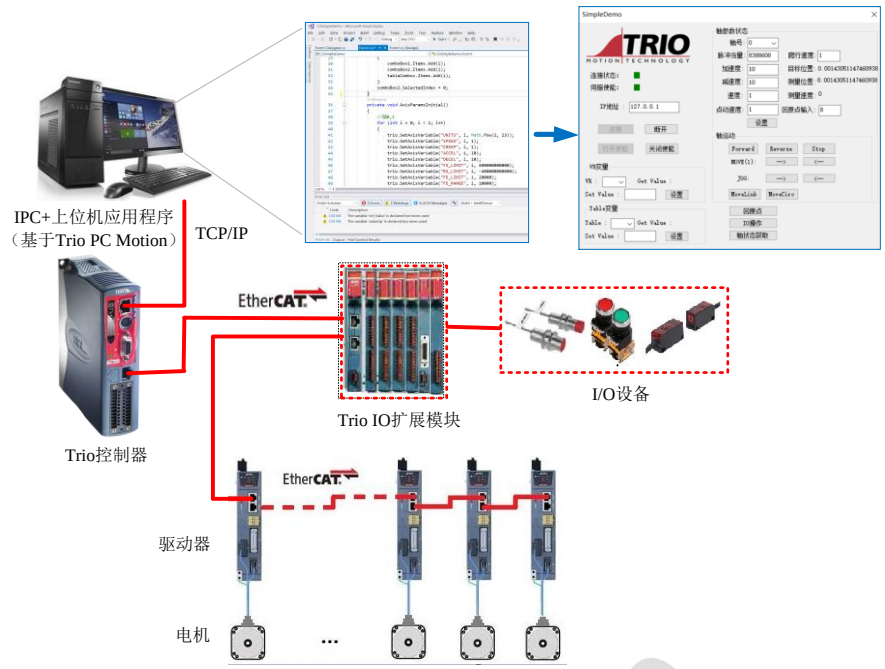


图 9-2 PC+Trio PC Motion 典型应用

9.1 PC Motion 的特点

（1）支持多种平台及编程语言

a）ActiveX 支持 ActiveX(OCX)组件的任何语言，如微软的可视化语言（BASIC, C, C++, VB，C#等）、LabView、Delphi 等。

b）DLL 目前支持.NET Framework 下的编程语言（如 C#，VB，C++等语言）以及 MFC 开发方式。

（2）使用方便

a) 安装：只需要安装 Trio PCMotion 插件，便可得到两种形式的插件。

下载链接：

http://www.triomotion.uk/public/software/softwareSupport.php?tabno=5

b) 使用：安装插件后，DLL 的方式只需要在项目中添加对相关 DLL 的引用，即可使用 DLL 中的封装的方法；ActiveX 则只需要将控件添加至项目中，并拖动至界面，即可使用控件中封装的方法。

（3）功能全面

PC Motion 提供的两种形式的插件都提供功能全面 API，它们都支持读/写轴和系统参数，读/写 VR 和 TABLE 变量、执行运动控制指令、IO 操作、文件加载等功能。另外，PC Motion 还提供了可直接调用 Trio BASIC 中的指令的 API，所以 PC Motion 基本可以实现 Trio BASIC 的功能。

PC Motion 提供的功能接口，会根据应用的需要，不断丰富。

ESTUN & Trio



图 9-3 PC Motion 提供全面的功能

（4）具有普适性

a) Trio PC Motion 目前适用于 Trio 大多数型号的控制器（包括常用的 MC403 / MC6N / MC664 / Flex-6-Nano 等）。

b) 应用程序与 Trio 控制器之间支持 Ethernet、USB、串口等多种通讯方式。

（5）灵活性强（PCMotion+MotionPerfect）

a) 为用户提供多种程序开发方式：可以选择功能强大的 MotionPerfect 软件，或者选择 PC Motion 进行开发项目。若选择 PC Motion，还可以选择 ActiveX 或者 DLL 进行项目开发。

b) PCMotion+MotionPerfect 软件，即使用上、下位程序共同提供完整解决方案。

c) 在基于 PCMotion 进行项目开发时，可使用 MotionPerfect 软件强大的监视功能辅助开发。

9.2 适用场景

PC Motion 通过提供 PC 端的组件，与运动控制器的交互，实现运动控制的功能。可以较好地与 PC 端软件融合、集成，让应用可以利用 PC 端资源优势。PC Motion 在如下应用场景中发挥作用。

（1）借助 PC 端的软件资源。在一些系统较复杂的行业，往往需要借助 PC 端专业软件(如 CAD/CAM，机器视觉)完成一些信息的采集及处理，从而完成控制任务，比如点胶机、纺织设备等。

ESTUN & Trio



图 9-4 立式点胶机

（2）用户软件与运动控制的紧密集成的专业化系统。用户需要将与控制器进行交互的程序集成到 PC 端专用应用程序中，以方便软件的管理，并可以充分利用 PC 端的软件优势进行文件、数据库的管理，比如数控机床、AOI(自动光学检测)、木工机械、纺织行业、锂电池卷绕等；



图 9-5 锂电池卷绕



图 9-6 纺织行业

（3）技术管理和技术背景限制。用户需要考虑现有技术的延续性和管理，如习惯于使用 PCI 卡的用户，选择 PC Motion 可完成无缝对接。另外，会高级编程语言的用户，只需要了解 API 中提供的相应的功能函数，即可快速上手。

9.3 ActiveX

9.3.1 ActivXe 控件

ActiveX 控件是一个组件或者自包含的软件包，可以在应用程序和网络中重复使用，从而加快程序开发速度和质量，创建它的主要技术是组件对象模型（Component Object Model, COM）。ActiveX 控件能以小程序下载装入网页，也能用在一般的 Windows 和 Macingosh 应用程序环境中。ActiveX 的一大优点是可以被大多数应用程序再使用，且可由不同语言的开发工具开发包括 C++、Visual Basic、PowerBuilder 等，甚至一些技术性语言，如 VBScript。

PCMotion 中 ActiveX 控件最大的特点是，它使用简单、快捷，并且用户不需要了解组件是如何开发的，有些情况下，甚至不需要自己编程便可以完成应用程序的设计。它能以控件的形式添加到窗体中，然后在后台程序中基于此控件对象调用封装的 API 来实现相应的功能，这种方式在可视化编程中显得非常方便。且 ActiveX 支持任何可支持 ActiveX（OCX）组件的平台或编程语言进行开发，例如，微软的可视化编程语言（BASIC，C#， VB，C++等）、LabView、Delphi 等，所以 ActiveX 更具有通用性。

9.3.2 ActiveX 的使用

下面以使用 C#编程语言在 VS2017 进行项目开发为例，对 ActiveX 的使用进行说明，若选择使用 VB 编程语言，其使用过程与此类似：

（1）安装固件

根据自己的系统选择安装 32 位还是 64 位 Trio PC Motion（如果要 x64 版本，建议先安装 x86 版本，避免因为 Visual Studio 的版本而在使用的过程中产生错误），如图 9-7 所示。



图 9-7 Trio PC Motion 安装包

（2）新建项目

打开 VS2017 新建一个 WinForm 项目，如图 9-8 所示。

ESTUN & Trio

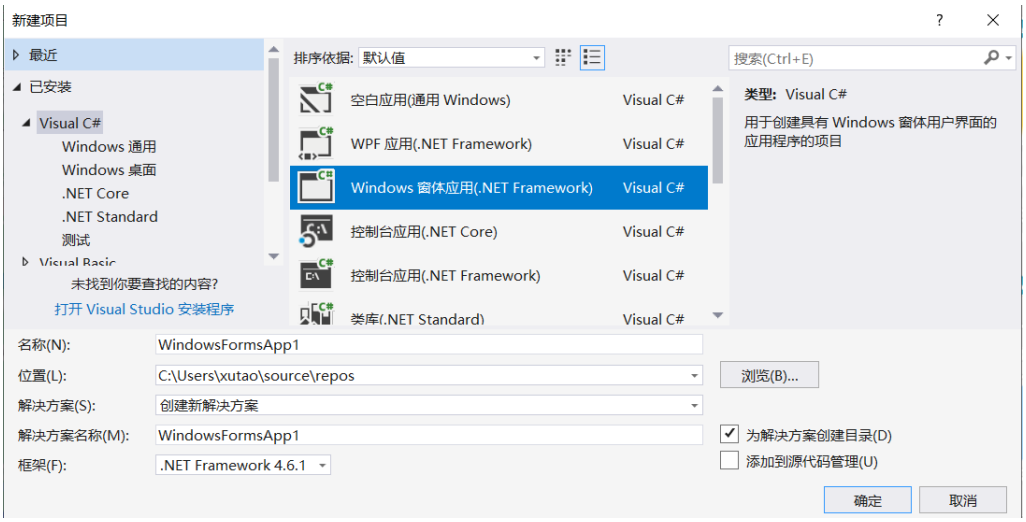


图 9-8 新建一个 WinForm 窗体

（3）添加控件。如图 9-9 所示，在工具栏的搜索框中输入 “TRIO”，找到 TrioPC Control，并将 TrioPC Control 添加到窗体中。若没有搜索到该控件，则参见步骤（4），否则进入步骤（5）。

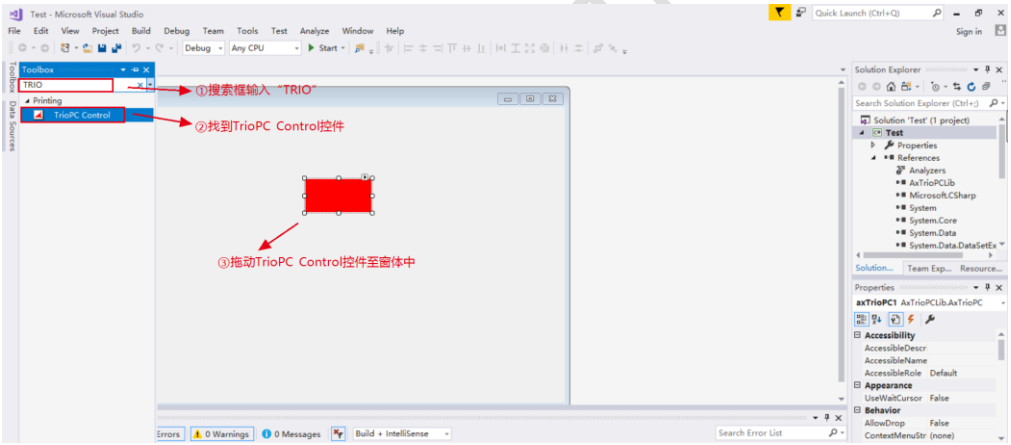


图 9-9 向窗体中添加 TrioPC Control 控件

（4）如图 9-10 所示，打开 “工具”，选择“选择工具箱项”，在弹出的对话框中，如图 9-11 所示，切换到“COM 组件”选项卡，找到名称为“TrioPC Control”的控件，并勾选此控件，点击“确认”按钮，进入步骤（3）。若在“COM 组件”中没有找到此控件，如图 9-12 图 9-10 所示，点击“浏览”按钮，在安装目录（C:\Program Files (x86)\TrioMotion\TrioPCMotion）中，找到 TrioPC.ocx 文件，并选中，然后点击“打开”按钮，将此文件添加至 COM 组件中，然后在 COM 组件选项卡中勾选“TrioPC Control”，再点击“确认”按钮，进入步骤（3）。

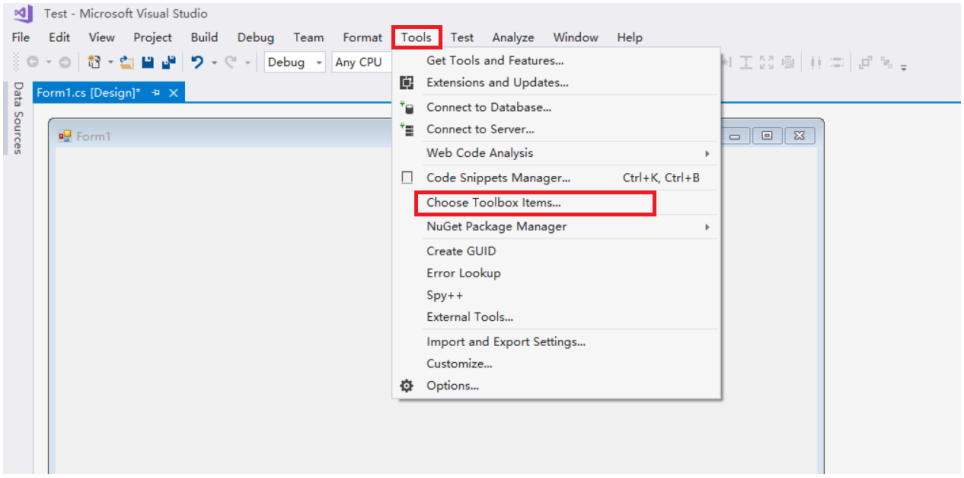


图 9-10 打开“选择工具箱项”

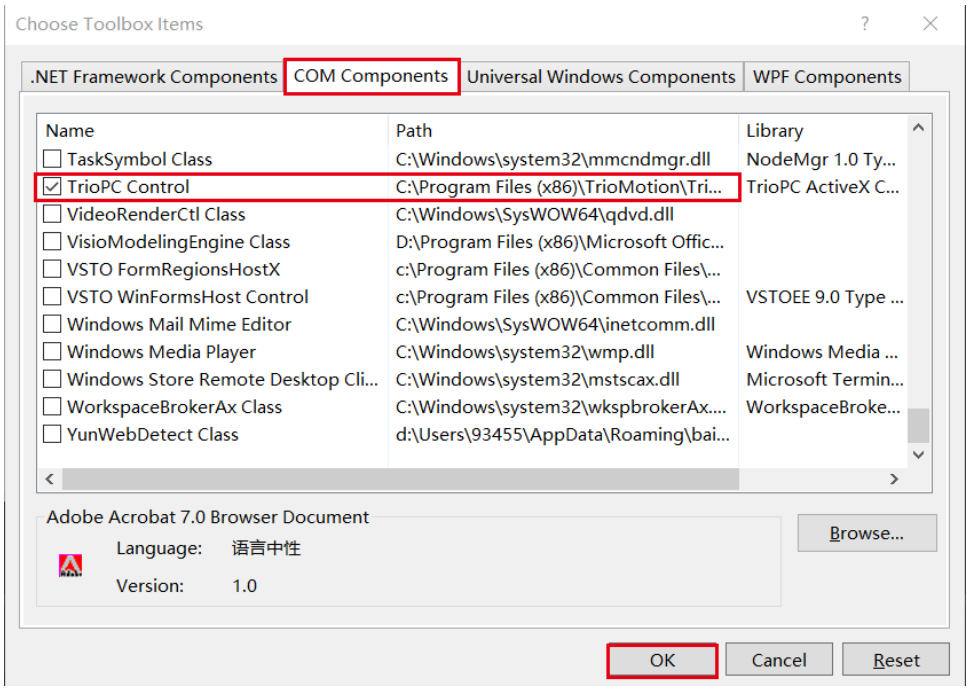


图 9-11 勾选 TrioPC Control

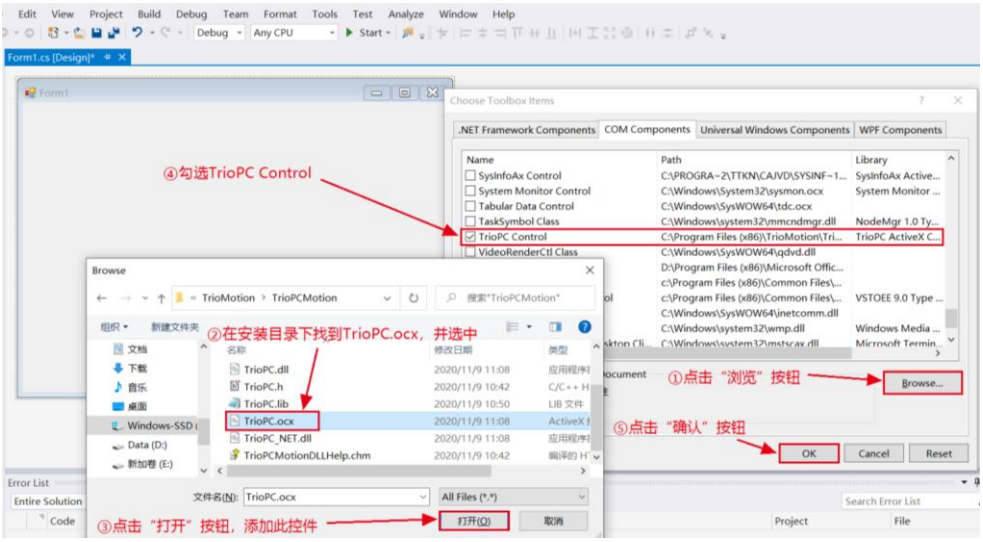


图 9-12 向 COM 组件中添加 ocx 控件

（5）连接控制器

①在窗体中添加一个 Button，用于建立应用程序与 Trio 运动控制器的连接，并修改其 Name 属性为“BtnConn”，修改 Text 属性为“连接”，如图 9-13 所示。

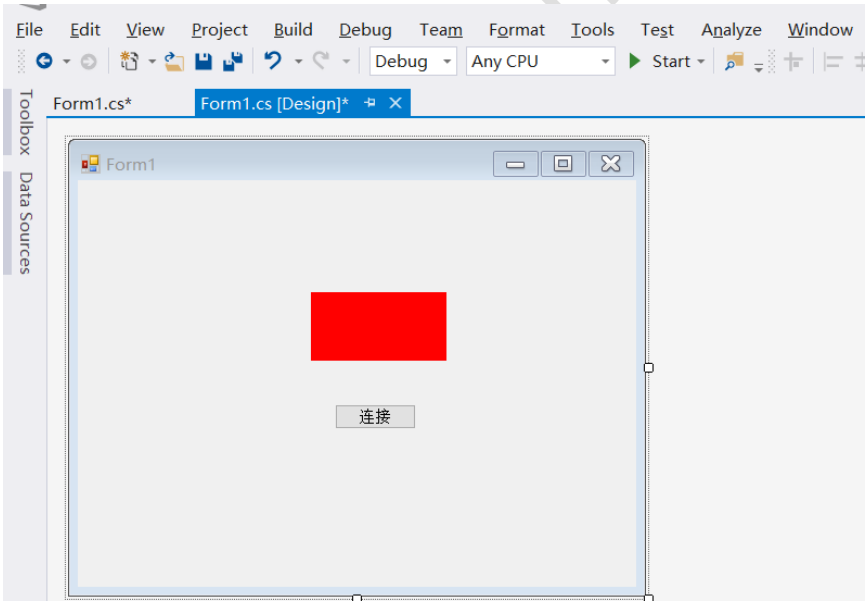


图 9-13 添加一个连接按钮

②双击按钮，进入到后台按钮的单击事件，通过 axTrioPC1 对象（通过添加控件至窗体中自动生成）调用控件中封装的方法来完成程序与控制器的连接，如图 9-14 **错误!未找到引用源。**所示。

```
private void BtnConn_Click(object sender, EventArgs e)
{
    axTrioPC1.HostAddress = "127.0.0.1"; // 设置IP地址（运动控制器）
    bool b = axTrioPC1.Open(2, 0); // 建立程序与控制器之间的连接
}
```

图 9-14 连接控制器程序

运行程序，连接成功后控件颜色由红色变成绿色，如图 9-15 所示。

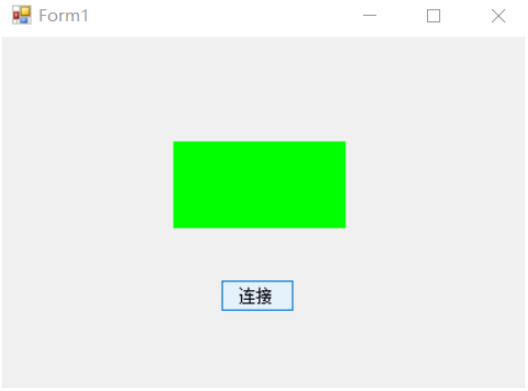


图 9-15 应用程序与控制器连接成功

9.4 DLL

9.4.1 DLL 简介

DLL（Dynamic Link Library，动态链接库）是一个包含函数和数据的模块集合（通常为一个.dll 或.exe 可执行文件），可以被其他应用程序重新共享。DLL 中虽然包含了可执行代码，但是不能够单独执行，只能由其他应用程序直接或间接调用。一个 DLL 中封装了一个或者多个已被编译、链接的函数，但可以同时在多个线程中被调用。

在安装 PC Motion（x86 版本）后可以得到 TrioPC.h、TrioPC.lib、TrioPC.dll（C++DLL 文件，若安装 x64 版本的 PC Motion 则会得到对应的 TrioPC64.h、TrioPC64.lib、TrioPC64.dll）以及 TrioPC_NET.dll（.NET Framework）。其中 TrioPC.dll 支持 MFC 开发，TrioPC_NET.dll 支持.NET Framework 开发方式（使用 C#，VB，C++等语言）。

PC Motion 中的 DLL 最大的优势在于使用上的灵活，它更便于用户根据自己的需求对 DLL 中的类和 API 进行自由封装，提高代码的重用性以及在实际项目中的开发效率，此形式的插件在国内更加通用。

9.4.2 DLL 的简单使用

下面以使用 C#编程语言在 VS2017 进行项目开发为例，对 C# DLL 的使用进行简单说明，若选择使用 VB 编程语言，其使用过程与此类似：

（1）安装固件

根据自己的系统选择安装 32 位还是 64 位 Trio PC Motion，如图 9-16 所示。如果已经安装过，无须重复安装，直接进入步骤（2）。

ESTUN & Trio

PCMotion Setup x64-3.2.1.msi	2020/9/15 13:12	Windows Install...	3,644 KB
PCMotion Setup x86-3.2.1.msi	2020/9/15 13:12	Windows Install...	3,152 KB

图 9-16 Trio PC Motion 安装包

（2）新建项目

打开 VS2017 新建一个 WinForm 项目，如图 9-17 所示。

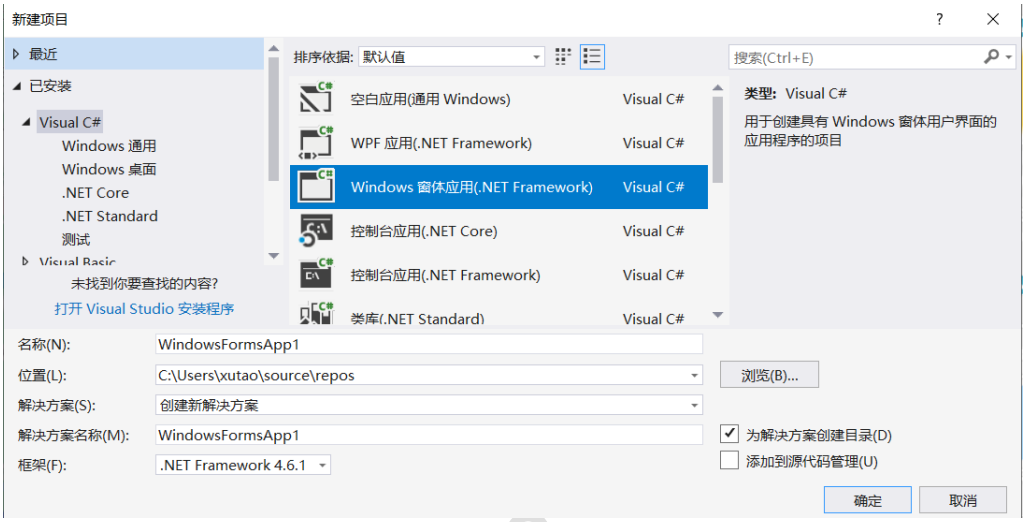


图 9-17 新建一个 WinForm 项目

（3）添加引用

将 TrioPC_NET.dll（在固件默认安装路径 C:\Program Files\TrioMotion\TrioPCMotion 中）添加到项目引用中，如图 9-18 所示。

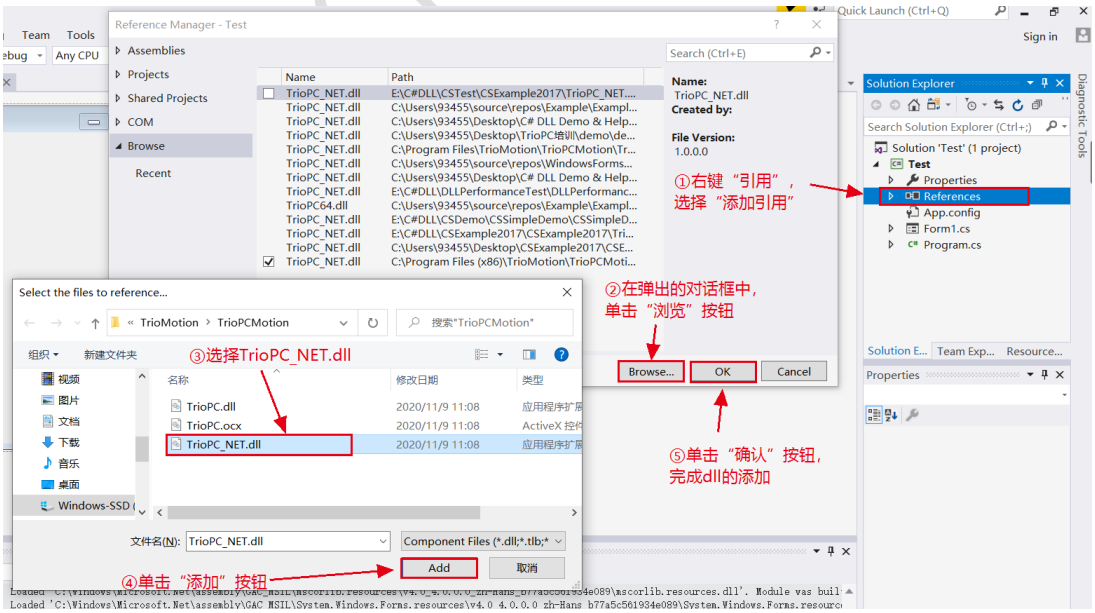


图 9-18 项目中添加对 DLL 的引用

（4）连接控制器

①为了使用方便，在使用 DLL 中的方法前，先添加对 DLL 中命名空间的引用，如图 9-19 所示；

```
using TrioMotion.TrioPC_NET;
```

图 9-19 添加引用命名空间

②窗体中添加一个用于连接控制器的按钮，DLL 中方法的使用与 ActiveX 不同的是：在使用前需要手动新建一个 TrioPC 的对象，然后通过此对象调用 DLL 中的方法。建立与控制器的连接的过程，如图 9-20 所示；

```
private void BtnConn_Click(object sender, EventArgs e)
{
    TrioPC trio = new TrioPC(); // 新建一个TrioPC对象
    trio.HostAddress = "127.0.0.1"; // 设置IP
    bool b = trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); // 建立程序与控制器的连接
    if (b)
    {
        MessageBox.Show("Connected!");
    }
}
```

图 9-20 连接控制器的过程

③运行程序，连接成功的画面如图 9-21 所示。

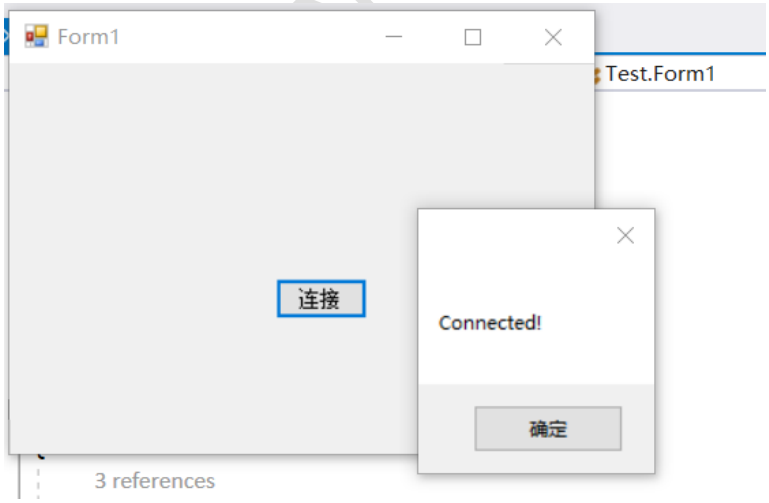


图 9-21 应用程序与控制器连接成功

9.5 应用介绍

PC Motion 在使用过程中主要涉及以下几个部分：

- 控制器初始化：建立程序与控制器的连接；
- 系统及轴参数初始化：设置伺服使能开关、轴参数（如脉冲当量、速度、加减速度、轴使能）等；

- 轴运动：包括单轴直线运动、多轴直线和圆弧插补、回零运动、电子凸轮和追剪运动等。
- I/O 操作：包括数字量与模拟量的输入、输出。
- 全局数据操作：VR 和 TABLE 变量的读写。

下面以使用 C#建立一个 WinForm 项目为例，通过使用 DLL 的方式对 PC Motion 的基本使用方式以及设计思路进行讲解，使用 ActiveX 进行项目开发与此类似。

在窗体中添加相关控件，设计如图 9-22 所示的界面。



图 9-22 项目示例

9.5.1 控制器初始化

操作控制器之前，需要调用 SetHost 函数设置控制器 IP 地址，并调用 Open 函数连接控制器。同样，当程序结束对运动控制器的操作时，必须调用 Close 函数释放运动控制器所占用的 PC 系统资源，使得所占资源可被其它设备使用。

表 9-1 示例使用的控制器连接函数接口

名称	功能	参考
SetHost	设置控制器 IP 地址	相关指令请参考 DLL 使用帮助文档
HostAddress	读取或者设置以太网连接访问地址	
Open	控制器连接	
Close	控制器断开	
IsOpen	控制器连接状态	

⚠注意：可以通过 PCI、串口、USB 和 Ethernet 等方式连接控制器，并且可以设置同步或异步模式运行。

例 1：通过 Ethernet 方式在同步模式下连接 TRIO 模拟仿真器

```
TrioPC trio = new TrioPC(); //实例化一个 TrioPC 对象
private void CntBtn_Click(object sender, EventArgs e) //点击连接控制器的 button 按钮，执行
{
string  ipAddress = IpTBox.Text.Trim(); //在 IP 地址输入框中输入”127.0.0.1”。
trio.SetHost(ipAddress); //或者采用 trio. HostAddress = ipAddress;
trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接 TRIO 模拟仿真器
 }
```

例 2：通过 Ethernet 方式在同步模式下连接 TRIO 控制器

```
TrioPC trio = new TrioPC(); //实例化一个 TrioPC 对象
private void CntBtn_Click(object sender, EventArgs e) //点击连接控制器的 button 按钮，执行
{
    string  ipAddress = IpTBox.Text.Trim(); //假设 TRIO 控制 IP 为” 192.168. 0.250”
    trio.SetHost(ipAddress);
    //连接控制器
    trio.Open(PortType.Ethernet, PortId.EthernetREMOTE);
    if (trio.Is Open(PortId.EthernetREMOTE)) //判断控制器是否连接成功
    {
        IsCnt.BackColor = Color.Green;//连接状态指示灯变为绿色
        markA = 1;//连接状态标识
    }
    else
    {
        IsCnt.BackColor = Color.Red;//连接状态指示灯变为红色
        MessageBox.Show("无法连接控制器！ ");
    }
}
```

9.5.2 系统及轴参数初始化

电机运动前需要先对系统参数如 WDOG（总使能）、LIMIT_BUFFERED（运动缓存区大小）等进行设置。其次需要对轴参数如 UNITS（轴单位当量）、SPEED（轴速）、ACCEL（加速度）、

DECEL（减速度）等进行设置。

表 9-2 示例使用的系统及轴参数操作函数接口

名称	功能	参考
SetVariable	设置系统参数	相关指令请参考 DLL 使用帮助文档
GetVariable	读取系统参数	
SetAxisVariable	设置轴参数	
GetAxisVariable	读取轴参数	

⚠注意：Trio 控制器中系统参数 LIMIT_BUFFERED 默认值为 1，最大可设为 64。表示控制器运动缓存区最大可储存 64 条运动指令，超出该范围时系统暂停执行该段程序中的后续指令（进入休眠状态），直至其余运动指令加载至缓存区中。

例 1：打开总使能开关

```
//连接控制器后打开使能开关
private void EnableBtn_Click(object sender, EventArgs e) //点击连接控制器的 button 按钮，执行
{
    if (markA = 1)  //判断控制器连接是否成功
    {
        bool b = trio.SetVariable("WDOG", 1);
        //使能状态判断
        if (b)
        {
            IsEnable.BackColor = Color.Green;  //若使能打开则指示灯亮绿色
            markB = 1; //使能状态标识
        }
        else
        {
            IsEnable.BackColor = Color.Red;  //若使能打开失败则指示灯亮红色
        }
    }
    else //若控制器连接失败，则提示
    {
        MessageBox.Show("请先连接控制器并选择相应轴号...");
    }
}
```

例 2：设置轴相关参数

```
if (markA == 1 && AxisTBox.Text != string.Empty) //判断控制器是否连接成功且选择相应的轴号
{
    trio.SetAxisVariable("UNITS", numAxis, Math.Pow(2, 23));  //单位当量
```

ESTUN & Trio

```
        trio.SetAxisVariable("ACCEL", numAxis, 10); //加速度
        trio.SetAxisVariable("DECEL", numAxis, 10); //减速度
        trio.SetAxisVariable("SPEED", numAxis, 1); //速度
        trio.SetAxisVariable("FS_LIMIT", numAxis, 40000000); //右极限
        trio.SetAxisVariable("RS_LIMIT", numAxis, -40000000); //左极限
        trio.SetAxisVariable("FE_LIMIT", numAxis, 2000); //跟随误差极限值
        trio.SetAxisVariable("FE_RANGE", numAxis, 1000); //跟随误差范围
        trio.SetAxisVariable("SERVO", numAxis, 1); //闭环控制
        trio.SetAxisVariable("AXIS_ENABLE", numAxis, 1); //单轴使能
        MessageBox.Show("轴基本参数设置成功");
    }
    else //若控制器未连接或未选择轴号，则提示
    {
        MessageBox.Show("请先连接控制器并选择相应轴号...");
    }
```

9.5.3 轴运动

9.5.3.1 单轴连续运动

控制器初始化、系统及轴参数初始化成功后，便可以使用运动控制指令控制轴运转。这里，我们以伺服电机单轴正向和反向运动为例，分别调用 Forward 和 Reverse 函数对轴运动的使用方法进行分析。

表 9-3 使用的单轴连续运动指令

名称	功能	参考
Forward	电机正转函数	相关指令请参考 DLL 使用帮助文档
Reverse	电机反转函数	

例 1：电机正转

```
/****************连接控制器**************/
trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/****************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节
trio.SetVariable("LIMIT_BUFFERED", 64); //运动缓存区

/****************设置轴参数**************/
//轴参数，详见1.5.2节例2

/******************轴运动****************/
if (markA == 1 && markB == 1) //判断控制器连接、使能是否成功
{
    trio.Cancel(2, numAxis); //先停止当前轴运动
    trio.Forward(numAxis); //电机正传
```

```
}
else //若控制器未连接或使能未成功，则提示
{
    MessageBox.Show("请先连接控制器...");
}
```

例 2：电机反转

```
/****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/***************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节

trio.SetVariable("LIMIT_BUFFERED", 64); //运动缓存区

/****************设置轴参数**************/
//轴参数，详见1.5.2节例2

/*****************轴运动*****************/
if (markA == 1 && markB == 1) //判断控制器连接、使能是否成功
{
    trio.Cancel(2, numAxis); //先停止当前轴运动
    trio. Reverse (numAxis); //电机正传
}
else //若控制器未连接或使能未成功，则提示
{
    MessageBox.Show("请先连接控制器并打开使能...");
}
```

例 3：当电机出现运动异常时，按下急停按钮

```
private void RapidStop_Test_Click(object sender, EventArgs e)
{
    if (numAxis >= 0)
    {
        trio.Cancel(2, numAxis); //取消轴运动
    }
}
```

9.5.3.2 直线和圆弧插补运动

插补运动即根据给定的位置及速度等相关信息进行数字计算，不断计算出参与运动的各坐标轴的进给指令，然后分别驱动各自相应的执行部件产生协调运动，以使被控机械部件按理想的路线与速度移动。TRIO 中多轴插补运动一般包括直线插补（MoveRel、MoveAbs）和

ESTUN & Trio

圆弧插补（MoveCirc），以下将结合 DLL 插补函数的用法进行介绍。

表 9-4 使用到的插补指令

名称	功能	参考
MoveRel	相对位置的直线插补函数	相关指令请参考 DLL 使用帮助文档
MoveAbs	绝对位置的直线插补函数	
MoveCirc	圆弧插补函数	

例 1：两轴直线插补

```
/****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器, 详见1.5.1节

/***************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节

trio.SetVariable("LIMIT_BUFFERED", 64);  //运动缓存区

/****************设置轴参数**************/
//轴参数，详见1.5.2节例2

/***************函数执行直线插补***********/
//判断控制器连接、使能是否成功
if (markA == 1 && markB == 1) //若成功，则执行
{
    trio.Cancel(2, numAxis);  //先停止当前轴运动
    double[] axes = new double[] { 0,1 }; //设置插补轴号为 0 轴和 1 轴
    trio.Base(2, axes); //设置插补轴
    double[] Distance = new double[] { -20, 10 };//相对插补距离
    trio.MoveRel(Distance, 0, 2, numAxis); //启动插补
}
else//若不成功，则提示
{
    MessageBox.Show("请先连接控制器并打开使能...");
}
```

例 2：两轴圆弧插补：使用机器将化学药品滴入试管中，喷嘴可以沿其轨道上下移动，最有效的运动是使喷嘴在试管之间成弧形运动。过程如图 9-23 所示。

ESTUN & Trio

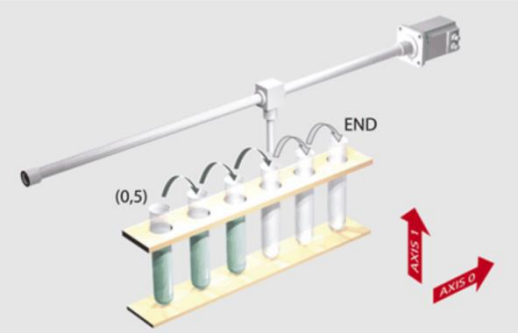


图 9-23 设备运动示例图

```
/*****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/***************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节

trio.SetVariable("LIMIT_BUFFERED", 64);  //运动缓存区

/*****************设置轴参数**************/
//轴参数设置，详见1.5.2节例2

/**************函数执行圆弧插补***********/
//先判断控制器连接、使能是否成功
if (markA == 1 && markB == 1) //若成功，则执行
{
      trio.Cancel(2, numAxis);  //先停止当前轴运动
      double[] axes = new double[] { 0,1 }; //设置插补轴号为 0 轴和 1 轴
      trio.Base(2, axes); //设置插补轴
      double[] Distance = new double[] { 0, 0 }; //设置插补位置
      trio.MoveAbs(Distance); //从当前位置直线插补到零点位置
      //圆弧插补
      for (int i = 0; i < 6; i++)
       {
          trio.MoveCirc(5, 0, 2.5, 0, 1); //顺时针圆弧插补
          trio.Execute("WAIT IDLE"); //等待插补结束
          trio.Op(15, 15, 1); //打开控制喷嘴的IO
          trio.Execute("WA(20)"); //等待20ms
          trio.Op(15, 15, 0);//关闭控制喷嘴的IO
       }
      trio.MoveAbs(Distance);//最后直线插补返回到零点位置
}
else   //若控制器连接或伺服使能不成功，则提示
```

ESTUN & Trio

```
{
    MessageBox.Show("请先连接控制器并打开使能...");
}
```

9.5.4 轴跟随运动

9.5.4.1 追剪运动

该运动包括一个线性运动，该线性运动分别具有可变的加速和减速阶段，从轴按照指令中设定的参数与主轴进行耦合运动。从而可实现与传送带同步、追剪、追线（攻丝）、线圈绕制等功能。需要调用的 DLL 函数为 MoveLink，以下将结合 MoveLink 在追剪机上的应用进行详细介绍。

表 9-5 追剪应用的指令

名称	功能	参考
MoveLink	追剪运动函数	相关指令请参考 DLL 使用帮助文档

例 1：追剪运动：追剪机每 16 m 就会将长纸切成卡片，同时以材料的速度移动。剪子可以走长达 1.2 米，在此示例中使用 1m。 纸张距离由编码器测量，单位转换因子设置为在两个轴上均以米为单位：（请注意，轴 7 是链接轴），过程如图 9-24 所示。

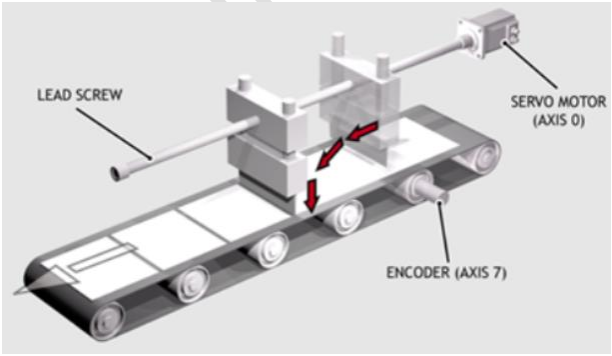


图 9-24 追剪运动的示例图

```
/*****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/****************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节

trio.SetVariable("LIMIT_BUFFERED", 64);  //运动缓存区

/*****************设置轴参数**************/
//轴参数设置，详见1.5.2节例2
```

```
/************函数执行追剪运动************/
//先判断控制器连接、使能是否成功
if (markA == 1 && markB == 1) //若成功，则执行
{
    int vaule =1;
    while (vaule == 1)
    {
        trioPC.Execute("RAPIDSTOP(2)"); //所有轴先停止运动
        trioPC.Forward(7); //传送带轴正转
        trioPC.MoveLink(0, 10, 0, 0, 7, 0, 0, 0); //假设驱动轴从0位置开始，先运行10m
        trioPC.MoveLink(0.3, 0.6, 0.6, 0, 7, 0, 10, 0); //基准轴加速至驱动轴速度
        trioPC.MoveLink(0.7, 1, 0, 0.6, 7, 0, 10.6, 0); //跟随连接轴运动后减速
        trioPC.Execute("WAIT IDLE AXIS(0)"); //等待基准轴运动缓存加载完后
        trioPC.Op(9, 9, 1); //启用追剪机
        trioPC.MoveLink(-1.0, 4.4, 0.5, 0.5, 7, 0, 11.6, 0); //跟踪连接轴并返回到初始位置
        trioPC.Execute("WAIT IDLE AXIS(0)");
        trioPC.Op(9, 9, 0); //关闭追剪机
        trioPC.In(8, 8, out vaule); //读取追剪I/O开关状态
    }
}
else   //若控制器连接或伺服使能不成功，则提示
{
     MessageBox.Show("请先连接控制器并打开使能...");
}
```

ESTUN & Trio

9.5.4.2 电子齿轮运动

若要实现多轴速度之间的链接，并产生电子变速，可调用 Connect 函数。

表 9-6 电子齿轮运动使用的接口函数

名称	功能	参考
Connect	速度跟随函数	相关指令请参考 DLL 使用帮助文档

例 1：简单地通过调用 Connect 易引起速度的阶跃变化。通过 CLUTCH_RATE 与零的初始和最终连接比率一起使用，以获取所需的运动，实现如图 9-25 所示的速度变化曲线的运动。

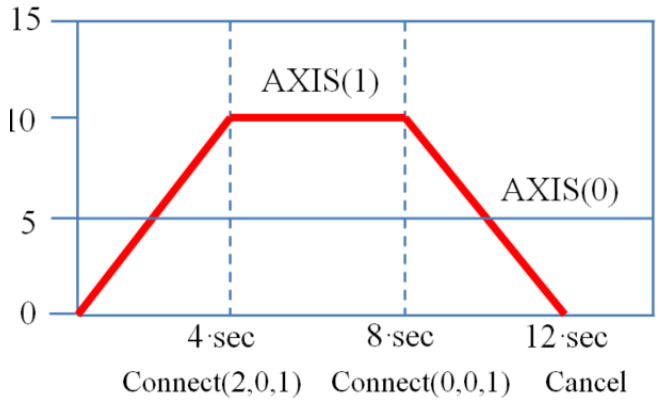


图 9-25 速度曲线

```
/****************连接控制器**************/
trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/***************设置系统参数*************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节
trio.SetVariable("LIMIT_BUFFERED", 64);  //运动缓存区

/****************设置轴参数**************/
//轴参数设置，详见1.5.2节例2

/***************速度连接运动*************/
private void Connect_Test_Click(object sender, EventArgs e)  //点击button按钮后执行
{
    double value;
    trio.GetAxisVariable("MTYPE", numAxis, out value);//读取基准轴是否在运动中
    if (markA == 1 && markB == 1 && value == 0)//控制器连接、使能判断
    {
        trio.Connect(0, 0, 1);//  将初始比率设置为零
        trio.SetAxisVariable("CLUTCH_RATE", 1, 0.5); // CLUTCH_RATE表示以速率/秒的单位更改连接比率，这里需要花费4秒从0变为2加速链接轴，故CLUTCH_RATE设为0.5。
        trio.Connect(2, 0, 1);//  应用所需的连接比率
        trio.Execute("WA(8000)");//  运行8s后
        trio.Connect(0, 0, 1);//  按比率断开连接
        trio.Execute("WA(4000)"); //等待减速完成
        trio.Cancel(2, 1);//取消连接运动
    }
    else if (markA == 1 && markB == 1 && value != 0) //若基准轴未开启，则提示
    {
```

ESTUN & Trio

```
            MessageBox.Show("请先开启基准轴...");
        }
        else //若控制器连接或伺服使能不成功，则提示
        {
            MessageBox.Show("请先连接控制器并打开使能...");
        }
    }
```

9.5.4.3 位置跟随运动

若要实现轴与轴之间的链接，可调用 ConnPath 函数。

表 9-7 示例使用到的接口函数

名称	功能	参考
ConnPath	位置跟随函数	相关指令请参考 DLL 使用帮助文档

例 1：要求在轴 0 上移动 156mm，在轴 1 和 2 上移动 100mm 的插补路径距离。这可以通过使用虚拟轴 3 作为插补组的路径距离并从轴 0 对其应用 MOVELINK 来实现。SPEED 最初设置为零，以便 MOVE and MOVELINK 同时启动。

```
/****************连接控制器**************/
trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/****************设置系统参数**************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节
trio.SetVariable("LIMIT_BUFFERED", 64);  //运动缓存区

/****************设置轴参数**************/
//轴参数设置，详见1.5.2节例2

/****************位置跟随运动**************/
if (markA == 1 && markB == 1)//判断控制器连接、使能是否成功
{
    trio.ConnPath(1,1,3); //将虚拟轴3与轴1连接
    double vaule;
    double[] axes = new double[] { 1, 2 }; //轴号为1轴和2轴
    trio.Base(2, axes);  //设置基准轴
    for (int i = 1; i < 3; i++)
```

ESTUN & Trio

```
        {
            trio.SetAxisVariable("SPEED", i, 0); //初始速度设为0
            trio.SetAxisVariable("MERGE", i, 1); //速度合并
        }
        double[] Distance = new double[] { 100, 100 }; //设置插补位置
        trio.MoveRel(Distance, 0, 2, 1); //直线插补
        trio.Execute("WA(1)"); //等待1ms
        trio.GetAxisVariable("REMAIN", 1, out vaule); //读取剩余插补距离
        trio.MoveLink(156, vaule, 0, 0, 3, 0, 0, 0); //位置跟随运动
        for (int i = 1; i < 3; i++)
        {
            trio.SetAxisVariable("SPEED", i, 10); //将速度设为10，开始运动
        }
    }
    else //若控制器连接或伺服使能不成功，则提示
    {
        MessageBox.Show("请先连接控制器并打开使能...");
    }
```

9.5.4.4 回零运动

在对电机进行精确的运动控制之前，需要先设定运动坐标系的原点（零点），伺服电机的运动都是基于原点坐标。目前，TRIO 控制器支持 6 种回零方式，调用的函数为 DATUM。

表 9-8 示例使用的回零函数接口

名称	功能	参考
DATUM	回零运动函数	相关指令请参考 DLL 使用帮助文档

例 1：采用不同方式回零，建立如图 9-26 所示的工程。

ESTUN & Trio

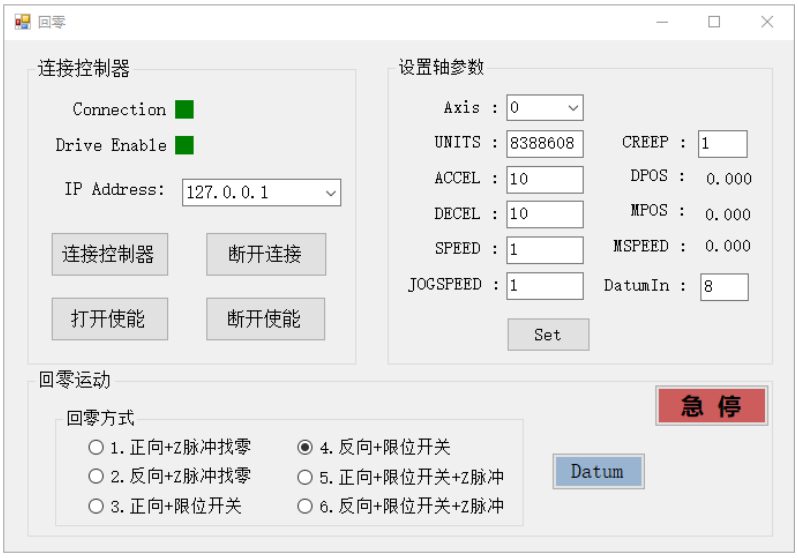


图 9-26 示例的交互界面

```
/****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/****************设置系统参数**************/
trio.SetVariable("WDOG", 1); //伺服使能，详见1.5.2节

trio.SetVariable("LIMIT_BUFFERED", 64);  //设置运动缓存区

/****************设置轴参数**************/
trio.SetAxisVariable("CREEP", numAxis, 1); //回零蠕动速度
trio.SetAxisVariable("DATUM_IN", numAxis, 8); //原点开关
trio.SetAxisVariable("****", numAxis, *****);  //其余轴参数设置，详见1.5.2节例2

/****************选择回零模式**************/
private int GetHomeMode() //选择回零模式
{
    int mode = 0; //  初始化回零模式，将其设置为0
    if (radioButton1.Checked) //选择模式1
    {
        mode = 1;
    }
    else if (radioButton2.Checked) //选择模式2
    {
        mode = 2;
    }
    else if (radioButton3.Checked) //选择模式3
    {
```

ESTUN & Trio

```
            mode = 3;
        }
        else if (radioButton4.Checked) //选择模式4
        {
            mode = 4;
        }
        else if (radioButton5.Checked) //选择模式5
        {
            mode = 5;
        }
        else if (radioButton6.Checked) //选择模式6
        {
            mode = 6;
        }
        return mode; //返回值
    }

    /**************函数执行回零运动***************/
    private void Datum_Test_Click(object sender, EventArgs e) //按下button按钮后执行
    {
        trio.Execute("RAPIDSTOP(2)"); //先停止轴运动
        trio.Datum(GetHomeMode(), numAxis); //根据选择的模式进行回零操作
    }
```

9.5.5 输入/输出

调用 Op/ In 函数可以对数字量 I/O 进行设置和读取，调用 Aout/ Ain 函数可以对模拟量 I/O 进行设置和读取。

表 9-9 示例用到的输入输出函数接口

名称	功能	参考
In	读取数字量输入的状态	相关指令请参考 DLL 使用帮助文档
Op	设置数字量输出的状态	
Ain	读取模拟量输入的状态	
Aout	设置模拟量输出的状态	

⚠注意：I/O 状态中 0 表示断开，1 表示开启。Op/ In 函数在使用时需要将 I/O 状态（二进制数）转换为十进制数，例如：设置 I/O 0-2 的通断状态为 100，则需将 Op 函数的值设为 1（状态位顺序是从右到左）。

以下将结合如图 9-27 所示的工程对相关 DLL 函数进行详细分析。

ESTUN & Trio



图 9-27 输入输出示例的人机交互界面

例 1：设置数字量 I/O 输出的状态

```
/*****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/*************设置数字量I/O输出***********/
private void Op_Click(object sender, EventArgs e) //按下button按钮后执行
{
    if (IO1_Num.Text != string.Empty) //判断是否设置首个I/O的位置
    {
        if (IO2_Num.Text != string.Empty && OpBox.Text != string.Empty) //判断是否设置末位I/O的位置和I/O控制状态
        {
            int numIO1 = Convert.ToInt32(IO1_Num.SelectedIndex.ToString());//选择首位I/O号
            int numIO2 = Convert.ToInt32(IO2_Num.SelectedIndex.ToString()); //选择末位I/O号
            int IOnum = numIO2 - numIO1 + 1;     //要设置的IO数量
            string value = OpBox.Text.Trim();       //读取位状态操作数（二进制）
            if (IOnum == value.Length)              //判断输入的位状态操作数是否在有效范围内
            {
                string REvalue = string.Empty;
                for (int i = 1; i <= value.Length; i++)
                {
                    int ASCvalue = Convert.ToInt32(value[IOnum - i]);//分别读取输入的位状态操作值，字符串0、1分别对应着ASCII码十进位中的48、49
                    if (47 < ASCvalue && ASCvalue < 50) //位状态数是否为有效值，即0或1
                    {
```

ESTUN & Trio

```
                    REvalue = REvalue + value[IOnum - i];//反转是因为trio中操作位是从右往左的
                }
                else //若为无效值时，则显示
                {
                    MessageBox.Show("状态位操作值为0或1");
                    return;//跳出程序
                }
            }
            int IOValue = Convert.ToInt32(REvalue, 2); //将二进制转成十进制
            trio.Op(numIO1, numIO2, IOValue); //设置IO输出
            for (int j = 0; j <= numIO2 - numIO1; j++)
            {
                int condition = value[j] - 48; //ASCII转换
                int numIO = numIO1 + j;
                this.IO1_Num.Items[numIO] = numIO + " -" + condition; //显示I/O设置状态
                this.IO2_Num.Items[numIO] = numIO + " -" + condition; //显示I/O设置状态
            }
        }
        else          //输入的位状态无效则显示
        {
            MessageBox.Show("状态位操作值无效，请重新输入...");
        }
    }
    else if (IO2_Num.Text == string.Empty && OpBox.Text != string.Empty) //若仅设置IO首状态位，则只对一个I/O进行操作
    {
        int numIO1 = Convert.ToInt32(IO1_Num.Text.Trim()); //读取I/O号
        int IOValue;
        string value = OpBox.Text.Trim(); //读取I/O号状态
        bool b = int.TryParse(value, out IOValue); //类型转换
        if (b && IOValue == 0 || b && IOValue == 1) //若类型转换成功，且位状态设置正确
        {
            trio.Op(numIO1, numIO1, IOValue); //设置IO输出
        }
        else //若状态位设置不正确，则显示
        {
            MessageBox.Show("状态位操作值无效，请重新输入...");
        }
    }
    else //若状态位设置不正确，则显示
    {
```

ESTUN & Trio

```
                MessageBox.Show("请输入有效的操作位值...");
            }
        }
        else //若I/O号未选择，则提示
        {
            MessageBox.Show("请选择有效的I/O号...");
        }
}
```

例 2：设置模拟量 I/O 输出的状态

```
/****************连接控制器**************/
trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器, 详见1. 5. 1节

/**************设置模拟量I/O输出************/
private void Aout _Click(object sender, EventArgs e)
{
    if (Aout_Box.Text != String.Empty) //判断是否设置模拟量I/O输出值
    {
        double value = Convert.ToInt32(Aout_Box.Text.Trim());//读取模拟量I/O输出值
        trio.Aout(0, out value); //模拟量I/O输出操作
    }
    else    //若未设置模拟量I/O输出值，则提示
    {
        MessageBox.Show("请输入有效值...");
    }
}
```

ESTUN & Trio

9.5.6 VR/TABLE 变量操作

Trio 控制器中提供两种数据存储区，分别为 Vr 和 Table。其中 Vr 可储存单个变量值，掉电保存；Table 可储存数组类型数值，掉电不保存。变量在程序之间是全局共享的。

表 9-10 VR/TABLE 变量操作接口函数

名称	功能	参考
GetTable	获取控制器中 Table 值函数	相关指令请参考 DLL 使用帮助文档
SetTable	设置控制器中 Table 值函数	

GetVr	获取控制器中 Vr 值函数	
SetVr	设置控制器中 Vr 值函数	

以下将结合如图 9-28 所示的工程对相关 DLL 函数进行详细分析。



图 9-28 VR/TABLE 示例人机交互界面

例 1：读取 Table 值

```
/****************连接控制器*************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/****************读取Table值***************/
private void TableRead_Click(object sender, EventArgs e)
{
    TableBox.Text = string.Empty;
    if (TableNum.Text != string.Empty) //判断是否选择Table位置
    {
        if (Tab_Length.Text != string.Empty) //判断是否写入Table长度
        {
            int tableValue = Convert.ToInt32(Tab_Length.Text.Trim()); //读取Table长度
            double[] value = new double[tableValue]; //长度为 tableValue 的数组
            int numTable = Convert.ToInt32(TableNum.Text.Trim()); //读取Table位置
            trio.GetTable(numTable, ref value); //获取从numTable开始， 长度为 tableValue 的
Table值
            for (int i = 0; i < tableValue - 1; i++)
            {
                TableBox.Text = this.TableBox.Text + value[i] + ","; //将获取的Table值分别用
```

“，”隔开

```
                }
                TableBox.Text = this.TableBox.Text + value[tableValue - 1]; //Table最后一位值
            }
            else  //若未写入Table长度，则提示
            {
                MessageBox.Show("请输入有效的Table值长度...");
            }
        }
        else //若未选择Table位置，则提示
        {
            MessageBox.Show("请选择Table位置...");
        }
    }
```

例 2：存储 Vr 值

```
/****************连接控制器**************/

trio.Open(PortType.Ethernet, PortId.EthernetREMOTE); //连接控制器，详见1.5.1节

/*****************存储Vr值**************/
private void VRWrite_Click(object sender, EventArgs e) //点击存储Vr的button按钮
{
    if (VRBox.Text != string.Empty) //判断是否写入Vr值
    {
        string value = VRBox.Text.Trim(); //获取Vr值
        int numVR = Convert.ToInt32(VRNum.Text.Trim()); //获取Vr位置
        double VRValue;
        bool b = double.TryParse(value, out VRValue); //转换类型
        if (b) //判断类型转换是否成功
        {
            trio.SetVr(numVR, VRValue); //将VRValue存入到numVR索引对应的Vr中
        }
        else //若不成功，则提示
        {
            MessageBox.Show("请输入有效的Vr值...");
        }
    else //若未写入Vr值，则提示
    {
        MessageBox.Show("请输入Vr值...");
```

ESTUN & Trio

```
        }
    }
```

9.5.7 Execute 指令

尽管 PC Motion 中 ActiveX 和 DLL 分别都提供较全面的 API 函数（大概 100 多个），但是在实际的使用中可能需要使用到其他 Trio BASIC 指令，这时需要用到 Execute 函数。Execute 可用来执行 Trio BASIC 中的语句或者命令，该语句或者命令是以一个字符串的形式，做为 Execute 函数的参数来被执行的。

表 9-11 Execute 指令

名称	功能	参考
Execute	远程命令执行函数	相关指令请参考 DLL 使用帮助文档

⚠注意：在使用 Execute 函数时，需注意不能在引号内加入空格。例如，停止所有轴运动 Execute("RAPIDSTOP(2)")，不能写成 Execute(" RAPIDSTOP(2) ")。

例 1：读取 EtherCAT 网络初始化

```
trio.SetVr(0, -1); // 初始化VR(0)，将其设置为-1
trio.Execute("ETHERCAT($22,0,0)"); // 将EtherCAT的状态返回到VR(0)中。EtherCAT指令详见Trio BASIC
trio.GetVr(0, out vr0_Value); // 读取VR(0)
int i = 0;
while (vr0_Value != 3 && (i < 3))// 控制器未连接驱动器，则重新初始化EC（若重启3次还未成功，则可能是其他原因）
        {
          trio.Execute("ETHERCAT(0,0)"); // 重新初始化EC
          Thread.Sleep(3000);
          trio.Execute("ETHERCAT($22,0,0)");
          i++;
        }
if (vr0_Value == 3)
        {
          AxisParamsInitial(); // 轴参数设置
        }
else
        {
          MessageBox.Show("控制器与伺服未建立连接！");
        }
```

ESTUN & Trio

参考文献

[1] Trio Motion Technology. Motion Perfect V5 Help.chm

[2] Trio Motion Technology. Trio BASIC Help.chm

[3] Trio Motion Technology. IEC61131-3 Help.chm

[4] Trio Motion Technology. Trio PLCopen Function Block Help.chm

[5] 蔡建华，温秀兰主编；孟飞，李辉，李洪海，邢绍邦副主编．计算机测控技术：东南大学出版社，2016.12：第 170 页

[6] 彭渝 何衍庆. IEC61131-3 编程语言及应用基础[M]. 北京：机械工业出版社，2009.1

[7] 彭渝 何衍庆. 运动控制系统软件原理及其标准功能块应用[M]. 北京：机械工业出版社，2019.4

[8] 崔金玉，唐红霞，郝利丽著．电路中的理论计算及应用设计：黑龙江大学出版社，2014.06

[9] 陈建明，王亭岭，孙标．电气控制与 PLC 应用．北京：电子工业出版社，2010：83-84

[10] 潘新民，王燕芳编著．微型计算机控制技术：电子工业出版社，2011.01

[11] 徐立芳主编．工业自动化系统与技术．黑龙江：哈尔滨工程大学出版社，2014：163-175

[12] 郁有文等编著.传感器原理与工程应用.西安：西安电子科技大学出版社，2003.7

[13] 王舒润，基于 PLCopen 的开放式运动控制器功能块的设计与实现.合肥.合肥工业大学，2019.5

[14] 董慧敏主编；刘勇军，姚志英副主编；王德俊主审．电力电子技术：哈尔滨工业大学出版社，2012.07：第 150 页

[15] 杨啸. 基于 RTX64 的开放式运动控制器关键技术研究[D].华南理工大学,2019.

[16] 赵春云. 基于 PCI 运动控制卡的上位机控制系统设计与实现[D].兰州理工大学,2019.

[17] 秦磊. 多轴运动控制系统的 EtherCAT 通信及其实现[D]. 苏州大学, 2019

[18] 廖华. 基于 IEC 61131-3 的伺服驱动器开发研究[D]. 华中科技大学, 2017

[19] 唐小会,吉方,王亚军,雷艳华,张连新,徐刚,王宝瑞 基于 PC+开放式运动控制器的通用控制软件系统研究. 中国工程物理研究院机械制造工艺研究所, 西安：26-28, 2013

[20] 王舒润. 基于 PLCopen 的开放式运动控制器功能块的设计与实现[D], 合肥工业大学，2019.5

[21] 高朝阳 基于 PLCOpen 及 OPC UA 的标准化机器人控制软件研究[D], 华中科技大学, 2019.5

[22]陈剑遥 运动控制系统的速度规划算法研究与实现[D], 浙江工业大学，2017.3

[23] 尹国梁 运动控制系统中 NURBS 曲线插补与速度前瞻算法研究[D], 哈尔滨工业大学，2019.6

[24] 肖腾腾 遵循 IEC61131-3 与 PLCopen 标准的嵌入式软 PLC 应用开发的研究[D], 华中科技大学，2016.5

[25] 魏冬冬 高精度编码器数字化接口的研究与实现[D], 合肥工业大学，2017.4

[26] 张圣 一种嵌入式多轴控制器的设计[D], 东南大学，2017.11

[27] 孙振兴 交流伺服系统先进控制理论及应用研究[D], 东南大学，2017.12

[28] 王晓峰 步进电机的精准控制[D], 内蒙古科技大学，2020.6

[29] 伍艳雄 步进电机细分驱动控制研究[D], 湖北民族大学，2019.9

[30] 高洪兵 工业以太网 EtherCAT 的 IO 从站设计与实现[D], 广东工业大学, 2019.5

[31] 曹润泽 基于 TCP/IP 的数据采集与控制系统设计[D], 杭州电子科技大学，2015.3

[32] 庆鹏展 面向运动控制的 EtherCAT 主站研究[D], 广东工业大学，2019.6

ESTUN & Trio