运动控制器编程手册

CNC篇

上海安浦鸣志自动化设备有限公司

目录

前言

本手册是基于 eXtreme、Agile系列控制器“运动控制器编程手册-动控制篇”的续篇，本手册重点讲解eXtreme、Agile控制器的CNC插补控制功能原理及应用方法。

■ 面向用户

本手册阅读对象：有一定的编程基础、有相关CNC基础知识、熟悉运动控制的工程技术人员！读者在完整阅读本手册后，再配合在实际应用系统上的编程调试实践，就可以掌握 CNC 运控原理和编程方法。

■ 主要内容

第一章 eXtreme、Agile控制器CNC 功能介绍

第二章 控制器支持的G代码指令集介绍

第三章 G-code编程指令及语法详解

第四章 G代码程序轨迹的输入与编辑

第五章 CNC轨迹曲线的描述形式与编译格式

第六章 典型的轨迹运动控制程序的构成

第七章 典型的CNC轨迹插补POU程序

第八章 功能更灵活的轨迹插补功能块

附录内容

■ 术语和缩写

术语	说明
McEngine Pro	eXtreme、Agile 系列 PLC 的编程软件
GateWay	eXtreme、Agile 系列 PLC 的专用通讯服务
PLC	Programmable Logic Controller (可编程控制器的简称)

■ 更多资料

资料名称	版本号	内容简介
《运动控制器编程手册 基础篇》	V1.0	运动控制相关基础编程内容的介绍
《运动控制器编程手册 运动控制篇》	V1.4	运动控制相关编程内容的介绍
《运动控制器编程手册 Robotic篇》	V1.2	轴组相关编程内容的介绍

■ 版本记录

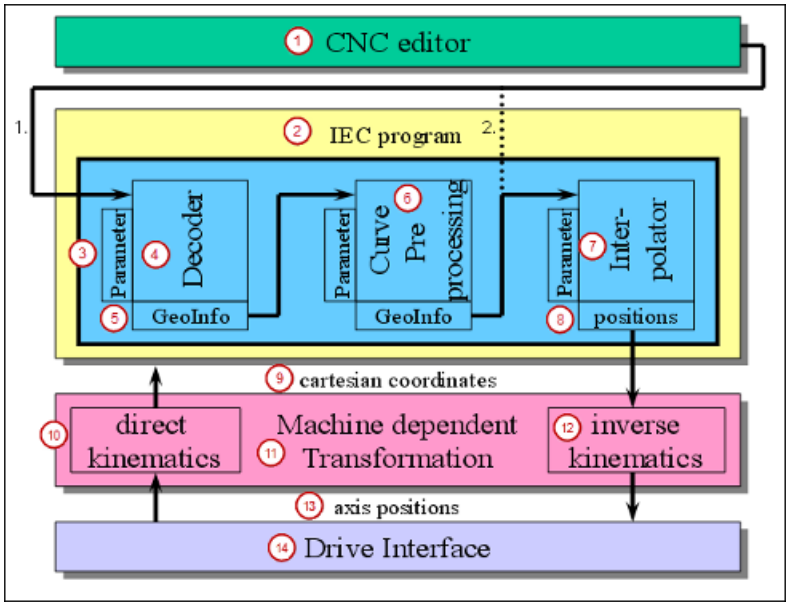
变更时间	版本号	变更说明
2023年7月	V1.0	第一版发行
2024年8月	V1.1	更新文档名称
2025年1月	V1.2	增加对Agile中型控制器的支持

第一章 PLCopen规范简介

eXtreme、Agile配套的McEngine Pro编程环境中，带有SM3_CNC.library库，无需另外的附件，即具有CNC轨迹插补控制功能。能将用户以 G-code 代码形式描述的轨迹，通过软件的位置插补和坐标系变换后，控制多个轴同时运动，合成得到所需的运动轨迹。

eXtreme、Agile控制器的CNC功能可以胜任激光切割、激光焊接、玻璃切割、木工机械、点胶控制等多轴轨迹运动控制的应用领域。

CNC的SoftMotion体系结构如下所示：



第二章 控制器支持的G代码指令集介绍

eXtreme、Agile的系统软件所使用的G 代码语法和执行规则，符合DIN66025标准，与常见的CNC数控系统的编程语言相同，支持运动控制类、运行中等待、维度模式切换、执行跳转、坐标转换、轨迹圆滑与刀具补偿等大类，同时支持M、H等辅助指令。eXtreme、Agile的控制逻辑可由用户程序设计，对G代码命令集有完整的支持，因此能实现更灵活的功能配合。

2.1 插补运动类G代码

G代码	执行功能
G0	线性定位，只是定位到指定坐标，无需同步运行
G1	线性插补运动，XYZ轴同步运行到指定坐标
G2/G3	顺时针/逆时针圆弧插补，同步运动到圆弧终点
G6	抛物线插补
G8/G9	椭圆形插补

2.2 执行等待类G代码

G代码	执行功能
G4	等待
G75	暂停插补（清除插补缓存）

2.3 模式切换类G代码

G代码	执行功能
G10	切换到2.5轴模式
G16	切换到3轴模式，
G17	切换到3轴模式，选择XY平面
G18	切换到3轴模式，选择ZX平面
G19	切换到3轴模式，选择YZ平面
G90/G91	切换为绝对/相对坐标模式
G98/G99	切换为绝对/相对极坐标模式

2.4 变量修改与条件跳转G代码

G代码	执行功能
G36	给变量写一个值
G37	向变量加一个值
G25	根据变量值跳转到G代码行

2.5 系统坐标转换类G代码

G代码	执行功能
G53	取消坐标偏移
G54	位置跟踪坐标的偏移量定义
G55	附加偏移量定义
G56	根据参数指定位置的偏移量修改

2.6 运行预处理类G代码

G代码	执行功能
G40	结束刀具半径补偿
G41	开始左侧刀具半径补偿
G42	开始右侧刀具半径补偿
G50	结束平滑倒角
G51	开始平滑处理
G52	开始边缘倒角处理
G60	切结束无效区域环路处理
G60	开始无效区域环路处理

第三章 G-code编程指令及语法详解

3.1 eXtreme、Agile支持的轴类型及G-code标识符

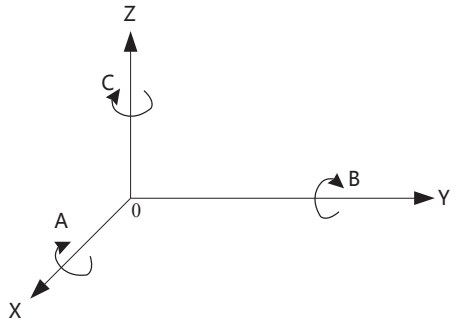
eXtreme、Agile 的 CNC 运动控制部分除了支持XYZ 3 轴插补运动外，另外还支持A/B/C/P/Q/U/V/W 等 8 个辅助轴的定位控制，以便在机械设备的插补运动轴在运动过程中，让辅助轴可以配合运行，这些辅助轴的运行起止时间可以由用户 G-Code 代码程序决定， 但与 XYZ 轴运动并没有完全同步关系。

控制器支持的插补轴与辅助轴特性定义如下：

轴名称	特性说明
XYZ	插补轴，该 3 个轴严格按G-Code定义的轨迹运动作插补同步运动
ABC	3 阶多项式样条辅助轴，在G-Code定义的一个轨迹对象期间，会以S曲线加速到设定速度后运动，以S曲线减速停止在轨迹终点。ABC轴的中间点与插补轴的中间点位置、时间对齐。 若目标点距离过大，可能出现速度超限；或限定速度过小，会出现不能按G-Code的要求及时到达目标点的情况。
PQUVW	线性辅助轴，在G-Code定义的一个轨迹对象期间，会以直线加速、减速，与插补轴到达终点时间对齐。 若目标点距离过大，可能出现速度超限；若限定速度过小，会出现不能按G-Code的要求及时到达目标点的情况。

例如简易上下料机构的手臂心在按指定轨迹运动过程中，但其手掌需要转动一定的角度，工艺上并不要求其转动角度与轨迹严格对应，就可以将手掌的转动用辅助轴来驱动；有些更复杂的设备中，可能需要控制传送带或工件转动，来配合机构的操作，这些都可以辅助轴来完成。

为了方便记忆，有的加工机床中将辅助轴定义如下：X轴伺服运动的结果是沿X轴方向的前进或倒退，辅助A轴的运动结构则是以 X 轴为圆心的转动，辅助 B/C 轴的运动结果则分别是以Y/Z轴为圆心的转动。同样的道理，P/Q/U/V/W 轴也都是可以使用的辅助轴，



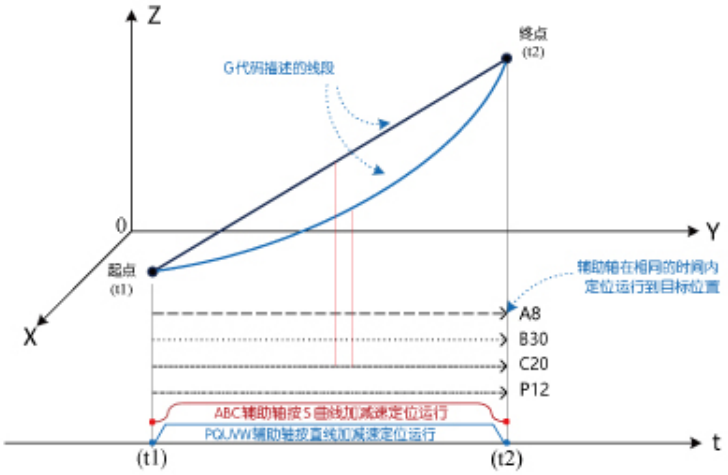
注意：在eXtreme、Agile的运控应用编程中，辅助轴名ABCPQWUW 并不要求与特定XYZ插补轴名关联，方便用户的理解和记忆更重要。

用户程序中，辅助轴运动的编程写法与 XYZ 轴写法相同：

```
N020 G01 X941 Y475 Z804 A8 B30 C20 P12
```

其中跟随A/B/C/P/Q/U/V/W等变量后边的数值，就是对应辅助轴的目标位置。

轨迹插补运行时，各辅助轴按定位运行方式运行，其执行完成时间则与 XYZ 完成时间相同，如下图：



G代码支持的语法

为了方便辨别，上述语句中只列出了轴和辅助命令的G-Code标识符，并没有表各轴坐标和指令操作数：

G5 X Y Z A B C P Q U V W F E H L/O D S

G10 X Y Z A B C P Q U V W F E H L/O D S

M K L

各标识符的定义如下：

G代码	执行功能
X,Y,Z	3 维笛卡尔轴坐标的目标位置，彼此可有同步关系
A,B,C,P,Q,U,V,W	附加轴坐标的目标位置，最多可有 8 个辅助轴，无同步关系
F,E	轨迹速度，轨迹加速度/轨迹加速度
H,L/O	H 指令
D	刀具半径
S	S 曲线
M,K,L	M 指令及传递参数

3.2 线性插补G01指令

```
N010 G01 X1000 Y2000 Z500 F10 E150 E-150
①    ②   ③      ④     ⑤   ⑥    ⑦     ⑧
```

其中：

N***为G代码行的行号，***代表0,1,2…的编号，

G**为G指令编号，例如G01代表直线插补，G02代表正向圆弧插补…

X***为X轴的目标位置，其中***为 Lreal 型坐标值；

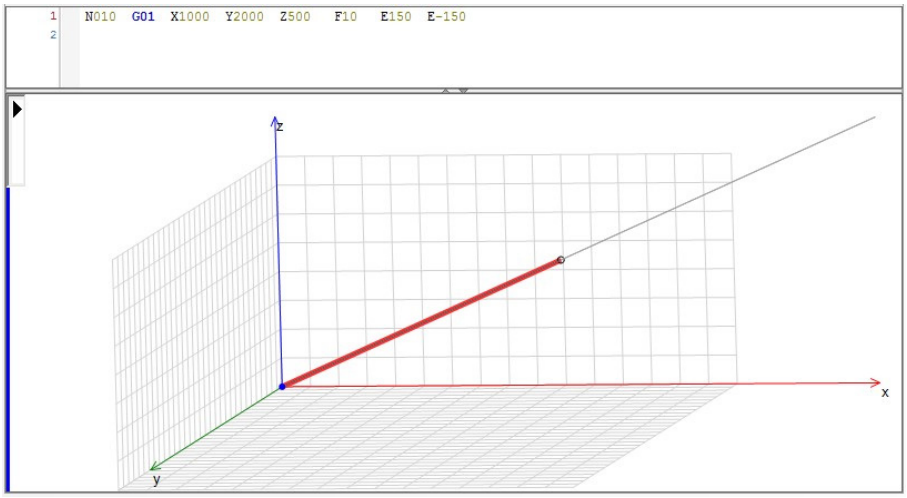
Y***为Y轴的目标位置，其中***为 Lreal 型坐标值；

Z***为Z轴的目标位置，其中***为 Lreal 型坐标值；

F***为插补轨迹运行速度；

E***为插补轨迹运行加速度；E-***为插补轨迹运行减速度；

在 McEngine Pro 编程界面，输入该指令行，可以看到如下的曲线图形：



用户所描述的CNC轨迹往往由多段曲线段组成，因此需要多 G 代码行组成，N***就是用于标识代码行的顺序，或行跳转时的标识，其编号可以不连续，但不得重复。

G代码指令行编写格式比较灵活，上述指令行中，除①②两项必需放在指令行的最前面以外，其他几项的顺序可以颠倒；当下一目标点的某项参数值没有变化时，对应项可以缺省。

3.3 圆弧插补G02/G03指令

```
G2  X500  Y400  Z20  R100  F8  E130  E-130
G2  X600  Y500  Z30  I280  J270  K20
G2  I300  J290  K25  T120
```

表示以当前位置为起点，以半径100的顺时针圆弧，到达（500,400,20）的目标点，圆弧的中心点坐标、弧度等由控制器自动计算得到。

其中：

N***为G代码行的行号，***代表0,1,2…的编号，

G**为G指令编号，例如G02代表顺时针圆弧插补，G03 代表逆时针圆弧插补…

X***为X轴的目标位置，其中***为 Lreal 型坐标值；

Y***为Y轴的目标位置，其中***为 Lreal 型坐标值；

Z***为Z轴的目标位置，其中***为 Lreal 型坐标值；

R***为圆弧半径；

I***为圆弧的圆心在x轴上的坐标

J***为圆弧的圆心在y插上的坐标

K***为圆弧的圆心在z插上的坐标

T***为圆弧的弧度（扇形角），以度° 为单位

F***为插补轨迹运行速度；

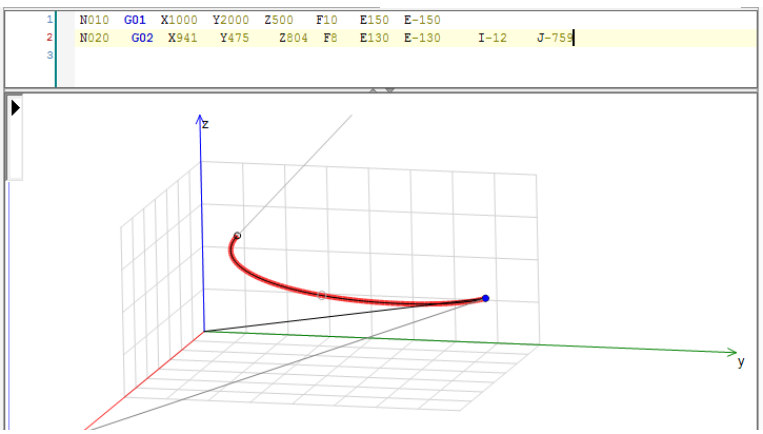
E***为插补轨迹运行加速度；E-***为插补轨迹运行减速度；描述圆弧有三种方式给定：

1. 可以半径R方式：给出目标点X/Y/Z+半径R；

2. 可以圆心坐标（I/J/K）方式：给出目标点X/Y/Z+圆心坐标I/J/K，这种给出方式容易出现矛盾的条件，圆心点的偏差必需小于10%，系统才能自动修正；

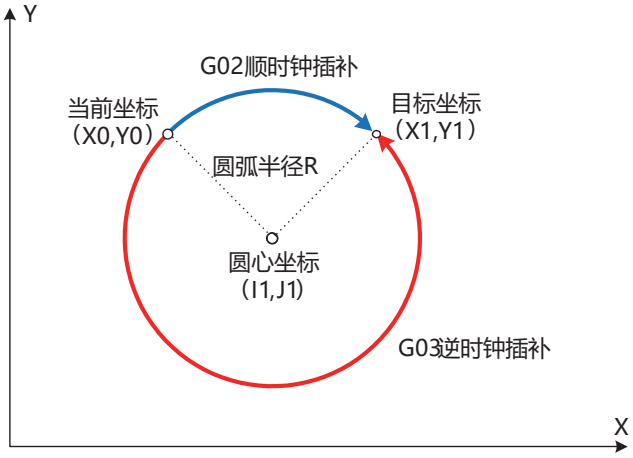
3. 弧度方式：在I/J/K中，以扇形角 T 和圆心点坐标I/J/K，自动计算目标位置。

命令中注意不要出现矛盾的条件，如半径方式中的半径值，要注意不得小于当前点与目 标点距离的一半；而圆心坐标方式中，需要注意当前点、目标点、圆心等三个点距离有约束， 容易出现条件矛盾，需特别注意，一旦有错，控制器可能会直线方式运动，而不会按预期的 圆弧轨迹要求运动。



当指令中 Z 轴方向坐标值有变化时，所描述是即为螺旋线

G02为顺时针方向圆弧插补，G03为逆时针方向圆弧插补，如果是在 X、Y 平面上观察，当目标点位置相同的情况下，两者的轨迹效果如下图

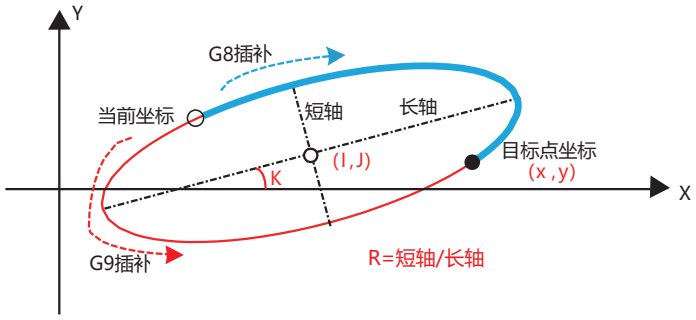


3.4 椭圆插补G08/G09指令

椭圆插补指令只能用于2.5D的轨迹，在XY平面上插补，其他平面不支持；

G8 指令用于顺时针椭圆弧的插补；G9 指令用于逆时针椭圆弧的插补；

该命令指定通过目标坐标X/Y、椭圆中点I/J、椭圆主轴方向 K 和主轴与次轴之间的长度比R的椭圆弧。



例如：

```
N030  G98
N040  G8  X200  Y100  I150  J100  K25  R0.5
```

表示以当前位置为起点，以半径100的顺时针圆弧，到达（200,100）的目标点，圆弧的中心点坐标、弧度等由控制器自动计算得到。

其中：

N***为G代码行的行号，***代表 0,1,2…的编号，

G**为G指令编号，例如 G8 代表顺时针椭圆弧插补

X***为X轴的目标位置，其中***为 Lreal 型坐标值；

Y***为Y轴的目标位置，其中***为 Lreal 型坐标值；

I***为椭圆的圆心在 x 轴上的坐标

J***为椭圆的圆心在 y 插上的坐标

K***为椭圆长轴的方向，以度为单位，0=按X轴方向；90=按Y轴方向；-90=按Y轴反方向；

R***为椭圆的短轴与长轴之比，取值为 0<r<=1；

r在(0，1]的范围内。只有当椭圆弧不是由主轴的端点、中点和斜率唯一定义时，才使用r。当两个端点与主轴的距离相同时，情况就是如此。然后，两个端点必须有相同的距离， 到第二轴。否则，在这些点上没有椭圆，系统用一条线代替椭圆。

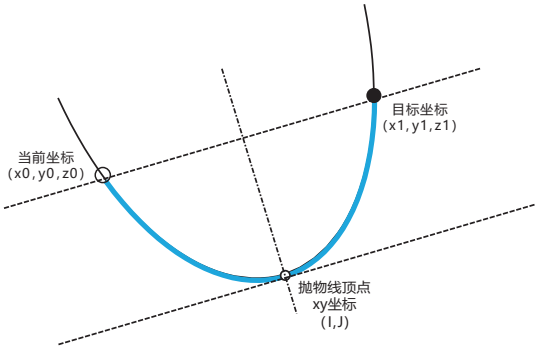
3.5 抛物线插补G06指令

抛物线指令只能用于 2.5D 的轨迹，在 XY 平面上插补，其他平面不支持。

N020	G06	X500	Y400	Z20	I100	J80
①	②	③	④	⑤	⑥	⑦

表示以当前位置为起点，沿顶点在XY平面（100，80）的抛物线，到达（500,400,20）的目标点，具体轨迹由控制器自动计算得到。

G6 命令为抛物线插补，该抛物线由端点 X/Y/Z 和顶点 I/J 的坐标定义，抛物线顶点是轨迹元素的切线，与起点和终点连接线平行的点



其中：

N***为G代码行的行号，***代表0,1,2…的编号，

G**为G指令编号，G06 代表抛物线插补…

X***为X轴的目标位置，其中***为Lreal型坐标值；

Y***为Y轴的目标位置，其中***为Lreal型坐标值；

Z***为Z轴的目标位置，其中***为Lreal型坐标值；

I***为抛物线顶点的x轴坐标

J***为抛物线顶点的y轴坐标

在本句语句中EFABCPQUVWMH等辅助指令都可以同时指定。

3.6 样条插补的G05/G10指令

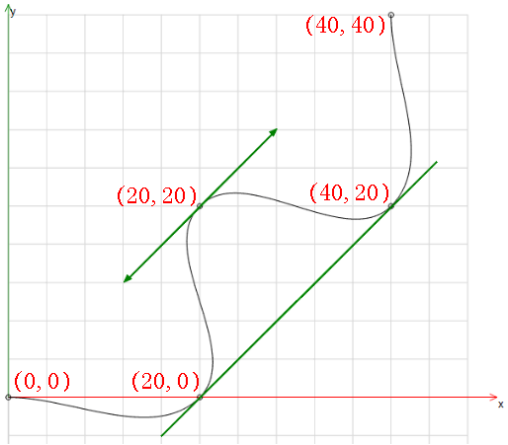
G5 是 2D 平面的样条曲线插补；G10 则是 3D 空间的样条曲线插补指令。

样条插值是在前后两个轨迹线段 PATH 中，插入一段多次曲线，从而使从前一个 PATH 元素到下一个PATH 元素的转换不间断地联接，包括位置、速度、加速度、加加速度的连续。即使得前一路径单元的端切线与样条的起始切线相一致，同样，样条的末端切线与后续路径元素一致，这样可使得轨迹运行更平顺。

例如对于如下的阶梯状间断点：

```
N0  G0  X0  Y0  Z0  F100  (起始点)
N10  G5  X20  Y0
N20  G5  X20  Y20
N30  G5  X40  Y20
N40  G5  X40  Y40
```

采用样条曲线连接，其轨迹如下：



正是因为样条曲线是为了保证与其前面一段轨迹的多阶连续，其起始段的切线会有如下情形： 起始点：

1. 如果前端是刀具运动的路径语句(例如：G1、G2、G3、G8、G9 等指令)，则使用路径元素的末端切线作为样条的开始切线；

2. 如果前端没有可用的刀具操作路径语句(而是如：G0、G92、M 等指令)，则使用起始点与第一样条点之间的连接线作为起始切线；

中间点：

3. 若有多个样条轨迹点相邻相连，中间点（如坐标点（20,20））的切线与连接线(绿线)平行；

结束点：

4. 如果后续有工具操作的路径语句(例如：G1、G2、G3、G8、G9 等指令)，则使用路径元素的端切线作为样条的结束切线；

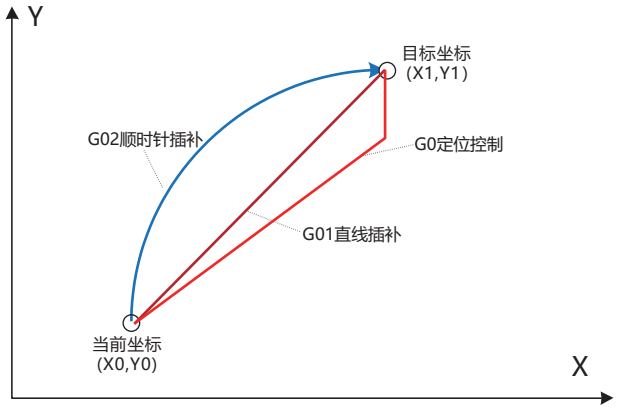
5. 如果没有有效的刀具操作(如只有 G0、G92、M 等指令)，则使用终点与第一样条点之间的连接线作为端切线。

因此，样条曲线的形状与前后线段的特征有关，多用于PP运动轨迹连续性优化处理。

3.7 定位G0、G92指令

G0指令与G1插补指令不同，G0指令只能实现定位控制，即使指令同时控制3个轴的运动，此时该3个按直线运动，彼此之间没有同步特性，而是因每个轴的运行距离差异呈现不同的运动完成时间。该指令执行效果也不会受轨迹圆滑、速度限制、刀具补偿等预处理影响。

G0指令常用于设备加工前的刀具的快速移动和位置调整，G0指令可以设置与插补指令不同的运行速度、加速度、减速度。



由上图可知，G0运动到目标点的轨迹，可能并不是直线，采用独立的快速移动速率来决定每一个轴的位置，依据每个轴需要的移动的距离先后停止。

G92指令用于移动下一个轨迹插补运行的起始坐标，

3.8 G-Code 程序的M辅助指令

eXtreme、Agile支持G-Code程序的M辅助功能，可以在插补运行中，按M指令要求暂停插补轨迹的运行，待接到对应的确认信号，才继续执行。

运动设备往往需要与其他设备配合运行，如冲床上下料运动机构（或简易机械手），往往要等待冲头冲压完毕并回到顶点后，才开始抓取加工工件动作；而另外的机构要等待该工件被取走后，才去开始放入新料的动作，这些暂停和等待、信号确认后才继续运动的控制特性，就需要用到G-Code程序的M辅助功能。

用户编程时，在需要暂停等待的位置，设置M辅助指令，当插补器执行到该指令行后，就会停下来，语句例如：

```
N050  M10  K87.6  L120.5
```

该语句中的M后可以跟随 1~65535 范围的值（Word类型）；

该语句中的K、L为可选参数，使用 K、L 传递常数值（Lreal类型），可以为逻辑程序提供更多的参数，该参数需要用另外的功能块 SMC_GetMParameters 来读取。

插补器执行到具有M指令的N050行语句时，就会停下来，并在插补器（wM端口）输出变量参数值10，用户程序接到该参数值后，可以处理该参数值对应的逻辑，直到满足了设定的条件后，在插补器输入端（bAcknM）变量置位1，插补器继续轨迹插补，再加速运行；

在用户G 代码轨迹的不同点，设置不同 M 参数值；执行轨迹程序时，用户程序根据得到参数值，就可以进行对应的逻辑响应。

但在许多应用中，希望在M条件满足已经满足的情况下就不要出现停顿，这样可以提高生产效率，用户程序中需使用预判断功能块SMC_PreAcknowledgeM对M参数进行预先判断，若满足，在执行到M指令语句时，就不会出现减速停顿，而是连贯地执行后续的 G 代码语句。

注意：

有些品牌的数控系统中，对M指令后面的参数值取值有限制，有些甚至把部分M参数值定义为特定的程序结束、特定轴和阀开关的控制，等等，在 EXTREME 中没有这些特别定义。

3.9 G-Code程序的H辅助指令

eXtreme支持G-Code程序的H辅助功能，可以在连续轨迹的插补运行中，经过H指令指定的轨迹点，输出指定的逻辑控制信号，用于逻辑控制。

H指令与M指令不同，经过监控点位置时，输出逻辑信号，并不出现轨迹运行的减速或暂停。借助这个功能，可以让系统知悉轨迹运行已到达特定位置，或用于触发相应的控制逻辑，如点胶机喷嘴的控制等。

H开关功能基本语句如下，其中L表示开关动作的绝对位置：

```
N90  G1  X20
N100  G1 X100  H2  L20
                ① ②
```

轨迹插补器执行N100行的 H 指令语句，在离线段起点 X20+20的坐标处，就会在插补器的（iLastSwitch端口）输出 H 行指定的“2”开关号闭合，按16位有符号数的解析方法:

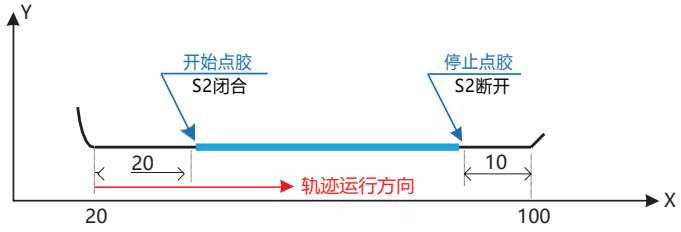
H 参数值	执行功能
开关号（bit0~bit14）	0=无开关动作命令； 1~32767=开关编号
符号位（bit15）	0=（正数）将开关闭合；1=（负数）将开关断开
L距离数值	大于 0：表示线段上离起点的距离时动作； 小于 0：表示线段上离终点的距离时动作；
O相对位置	表示在线段起点——终点之间的相对位置处动作

插补器还有一个DW型输出变量（dwSwitches），该变量的 32bit状态分别指示在插补器中已经被操作过，且编号为1~32的开关的输出状态。

有时，希望H 输出指令在线段进行开关的闭合与断开动作，可以采用如下语句：

```
N90  G1  X20
N100  G1  X100  H2  L20  H-2  L-10
                ① ②        ③     ④
```

就表示2号开关（S2）在X20+20处闭合，在 X100-10 处断开，可以用于如下的控制应用：



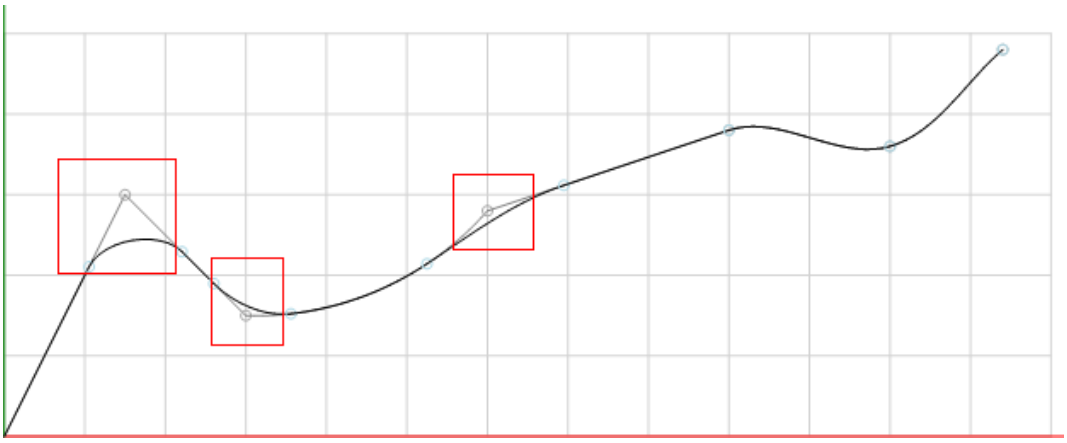
若对H开关动作的轨迹绝对距离不太关注，只是对所在线段的相对位置比较关心，可以采用H开关功能的相对O参数选项，如下语句：

```
N90  G1  X20  Y20
N100  G1 X100  H-2  O0.25
                ①      ②
```

就表示在线段起点——终点之间的 25%的相对位置（X40，Y25），2号开关断开。

3.10 轨迹的圆滑与倒角处理指令G51/G52

在实际应用中，对于轨迹转折处可能需要有倒角和圆滑处理，用G51/G5指令就可以启动圆滑路径处理，将拐角处位置改为连续即可实现连续插补。如下图：



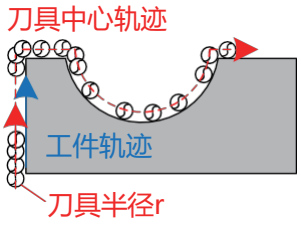
编程举例如下：

```
N000 G51 D1 //开始圆滑处理，D 值=拐角点到圆滑切点距离
N010 G01 X1.5 Y3 F1 E0.5 E-0.5
N020 G01 X3 Y1.5
N030 G03 X6 Y2.8 I0.06 J3.8
N040 G01 X9 Y3.8
N050 G05 X11 Y3.6
N060 G05 X12.4 Y4.8
N070 G50    //取消圆滑处理
```

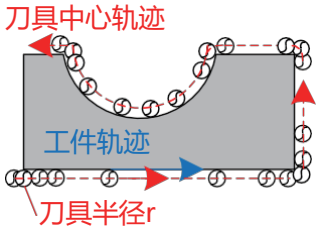
注意：在使用了轨迹圆滑或倒角功能的情况下，在编写的插补处理功能块中，需要有轨迹圆滑处理的功能块 SMC_SmoothPath。

3.11 轨迹的刀具补偿处理指令G40/G41/G42

用户一般设计所需工件外形或希望加工的图案形状，而设备控制的却是加工工具的运动轨迹，加工工具一般都会有其外形尺寸，如镗床、铣床、磨床的工具，因此加工工具的行走轨迹需要在工具轨迹的基础上，考虑工具的尺寸，预留一个刀具半径的偏移量，这就是刀具补偿。G-Code 指令集中的 G41、G42、G40 就分别是刀具半径左补偿、右补偿和取消半径补偿。如下图：



左补偿指令G41
刀具在工件前行轨迹左侧



右补偿指令G42
刀具在工件前行轨迹右侧

图中灰色区域为G-Code描述的工件轮廓轨迹，用半径为R的刀具切削工件时，刀轨必须始终与切削轮廓有一个距离为R的偏置，刀具中心（Tool Center Point,TCP）需按图中虚线所示的轨迹运动。在手工编程中进行这种偏置计算往往十分麻烦，如果采用G41、G42指令，刀具中心（TCP）运动轨迹可自动偏移一个R距离，而用户 G-Code编程只要按工件轮

廓考虑即可。在G41、G42指令中，刀具半径是用其后的D指令指定，可由加工时选用的刀具尺寸来决定。

加工类型的不同，要求的补偿方法也不同，如镗床在加工内圆、磨床加工外表面，插 补轨迹方向的就不同，参考了工件外形所描述的轨迹，可将补偿分为左补偿和右补偿。是指 沿着刀具前进的方向，刀轨向左侧或右侧偏置一个刀半径的距离。

若是将R变量与刀具的磨损情况关联，即根据刀具磨损系数与已加工的轨迹长度，计算出当前的实际的刀具半径R，可以提高加工精度。

编程指令用法如下：

```
N000 G41 D1          // 开始刀具左补偿，D 值表示刀具补偿半径，这里为 1
N010 G01 X1.5 Y3 F1 E0.5 E-0.5
N020 G01  X3 Y1.5
N030 G03 X6 Y2.8 I0.06 J3.8
N040 G01 X9 Y3.8
N050 G05 X11 Y3.6
N060 G05 X12.4 Y4.8
N070 G40 // 取消刀具补偿
```

注意：在使用了刀具补偿功能的情况下，在编写的插补处理功能块中，需要有刀具补偿 处理的功能块 SMC_ToolCorr。

在 CNC 机床应用中，还有刀具长度补偿，主要是针对 Z 轴的坐标补偿，防止刀头与工件或机床部件的碰撞，这种补偿比较容易计算，在此不作讲解。

3.12 代码指令的模式特性

从G代码指令的有效范围来分，可分为“模态代码”和“一般代码”。

“一般代码”在收到该命令时执行，指令执行完毕后，对后续的指令执行模式没有影 响。定义移动的代码通常是“一般代码”，像直线、圆弧和循环代码；

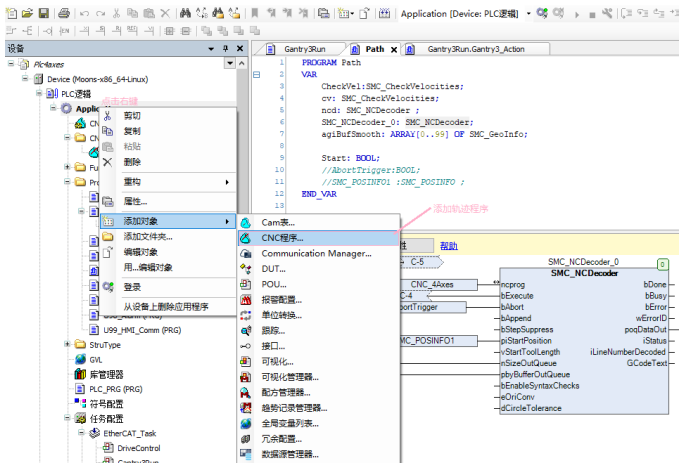
“模态代码”在它被执行后，其作用效果会继续维持，对后续的指令执行模式影响。使用模式特性的 G-Code 指令，设定了插补模式（如改变位置模式、设定的平面、圆滑倒角等等），或对某个标识符赋值后，对后续指令行持续有效，直到有新的模式命令或标识符值输入，才会按新的设定进行插补运行。例如：

```
N000 G51 D1          // 开始刀具左补偿，D 值表示拐角点到圆滑切点距离，这里为1
N010 G01 X1.5 Y3 F1 E0.5 E-0.5
N020 G01  X3  Y1.5       //此处的 F、E 值按上一行的设定值执行
N030 G03 X6 Y2.8 I0.06 J3.8
N040 G01 X9          //此处的 y 值按上一行的设定值执行
N050 G05 X11 Y3.6
N060 G05 X12.4 Y4.8
N070 G50    //取消圆滑处理
```

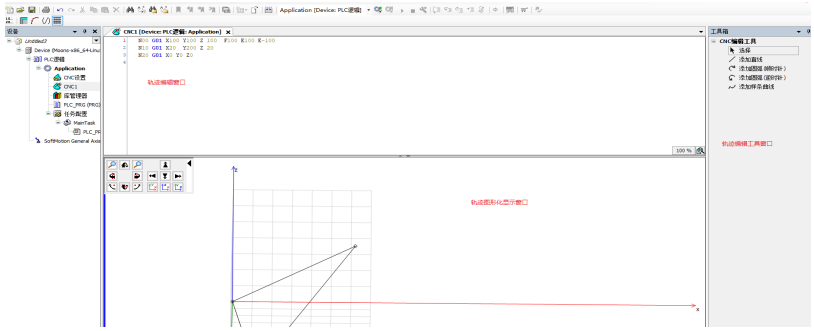
第四章 G代码程序轨迹的输入与编辑

4.1 在McEngine Pro中编写G-Code程序

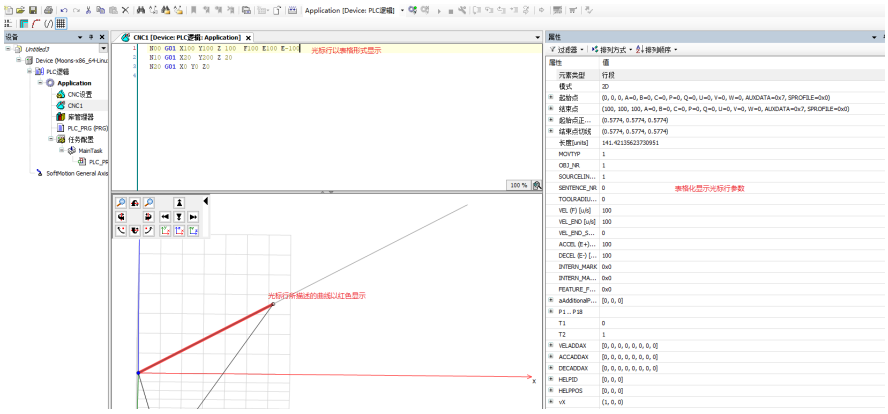
McEngine Pro中提供了G-code轨迹程序编辑器，在用户工程中添加“CNC轨迹程序”的方法如下图， 在一个用户工程中，可以添加若干个不同名称的CNC轨迹程序：



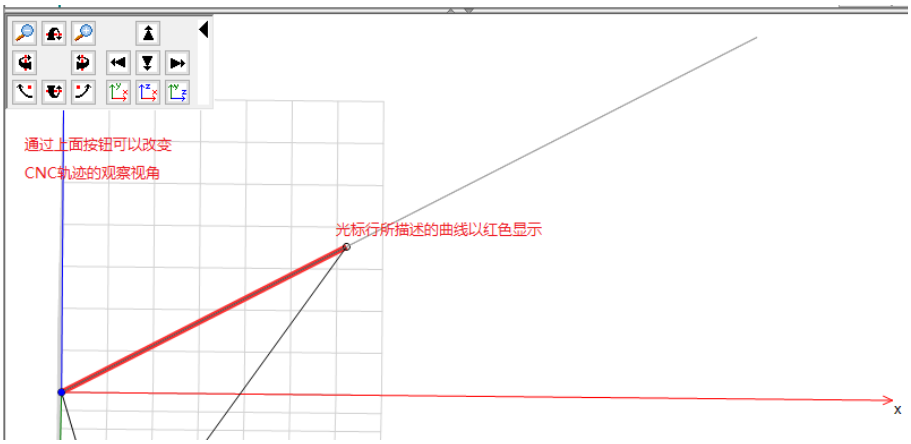
这样就添加了CNC1轨迹，轨迹名可以自由命名。双击CNC1，即可进入CNC1的编辑画面，中间的窗口，按照G-code的格式要求，以文本行方式输入G-code，中间下方的图形窗口，能即时看到轨迹形状，便于检查确认；也可以反过来，在图形窗口直接进行添减或拖拽图形，文本窗口的G-code 代码程序会自动变化为图形的描述指令，这样可提高编程效率， 直到将所需的所有轨迹线段输入完成。



为了更精确的查看G-code代码指令编程参数，可在右侧的“属性”窗口，以表格形式直观看到当前G代码语句行的意义及各项参数，表格中可看到更丰富的轨迹分析信息，并可以在表格中进行参数的修改和输入，如下图：



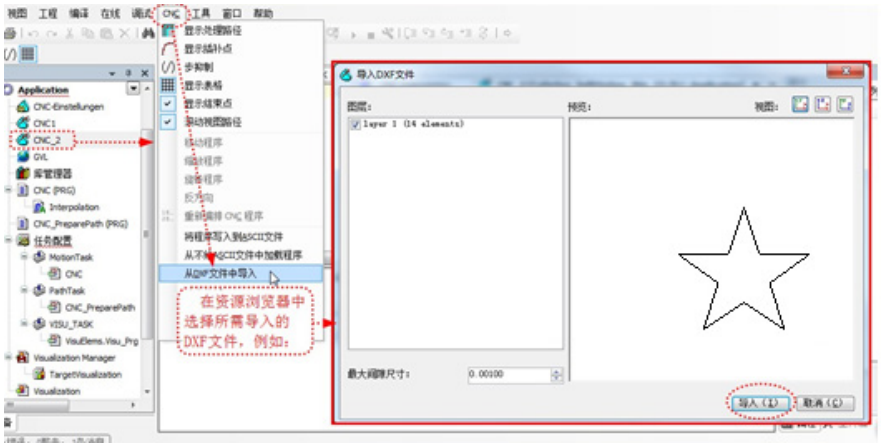
McEngine Pro编程软件中，还提供了方便查看轨迹曲线的按钮，以便从不同角度和平面来进行缩放观察。



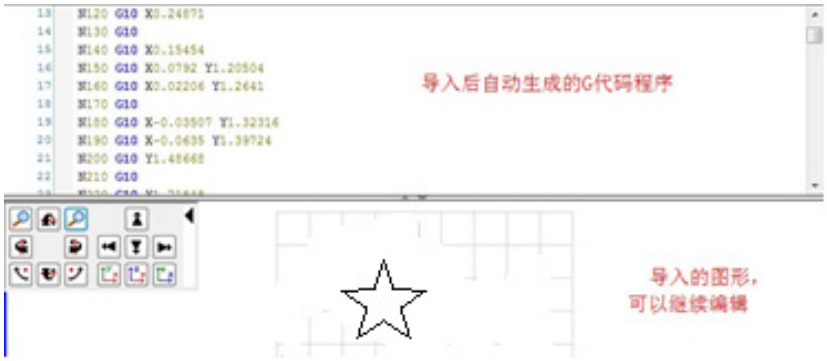
在McEngine Pro中采用了与 CAM 凸轮曲线相似的方式来管理CNC轨迹程序，当用户声明一个CNC程序时，系统内部同时为之声明了一个同名的轨迹数据结构体，用户程序中可以采用数据结构指针变量访问该结构体数据。

4.2 导入DXF格式文件的设计图形

许多加工轨迹往往采用CAD设计图导出，或需要借助CAD软件来设计比较复杂的图案轨迹，这类设计方式已越来越多地被采用。McEngine Pro编程软件就提供了DXF格式的CAD文件导入，使用方法如下图：



导入后，系统会自动生成G代码程序，以及与之对应的轨迹图形，两种轨迹形式都可以继续编辑。轨迹图上的小圆圈表示轨迹分段拐点，点击其中的线段，在文本窗口的对应G代码行就会高亮显示，反之，点击文本窗口的G代码行，图形窗口对应的线段就会高亮显示，方便了轨迹程序的编辑修订。

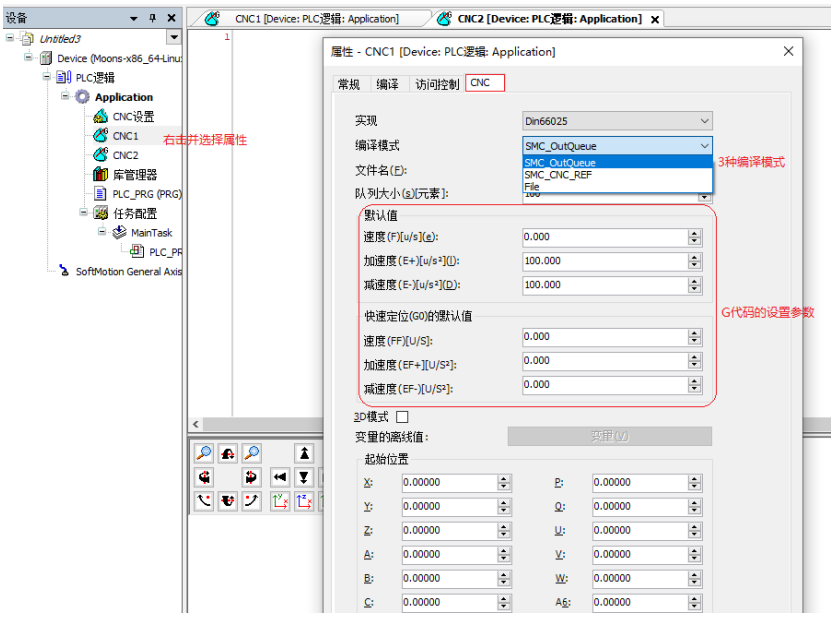


由于G代码的图形采用直线、圆弧、样条曲线组成，CAD设计图保存时，若采用的样条阶数比较少，会使得导入的部分曲线失真，改善的方法是在CAD设计中，要5阶样条来保存曲线信息，就可以避免失真。

第五章. G-Code程序的编译处理

5.1 CNC轨迹曲线的描述形式与编译格式

我们编写的G-Code轨迹程序，是以文本字符串的形式存在的，其中有些轨迹信息是需要进行上下文分析后才能准确还原用户所需的控制特性，为了提高轨迹插补的运算效率，实现运行轨迹的平稳控制，需要将之编译成适合处理器执行的代码序列，典型格式如OutQueue格式，例如下图的 CNC_2 轨迹程序：



我们可以这样来理解OutQueue格式，它是将G-Code源文件编译成方便插补器执行的数据表格，如轨迹的起止坐标与线型、加减速参数、模式指令参数、辅助轴运动参数、M与H指令参数，以及其他辅助信息等等，解析成有序的、无需前后文分析的正则表格数组。插补器在执行时，按表格数据与线型直接插补输出即可；根据这个表格数组，可以实现轨迹的双向运动、多个连续线段的速度连续缓冲（Buffer）处理等。因此，OutQueue格式可以理解为正则的G代码文件。

正是因为OutQueue格式具有更完整的轨迹线段坐标与线型信息，方便作轨迹的圆滑倒角处理（Smooth），运行速度分析调整（CheckVelocity）、甚至是CNC加工轨迹控制所需的刀具补尝（ToolCorrect）的处理，因而轨迹控制的预处理，往往都是基于轨迹的 OutQueue格式进行的。这里提到的几种预处理，在EXTREME、AGILE系统中已提供对应的FB功能块，可以根据控制需要进行实例化调用。

在McEngine Pro编程时，若轨迹是一个明确的G-Code，就可以直接编译成OutQueue格式，下载到控制器中，有插补器执行。

实际运动控制应用中，常将CAD/CAM软件设计的轨迹以文件形式（File）进行保存和发布，PC电脑与控制器之间的通信交互等，采用这个格式，可方便控制器按需要选择不同的轨迹程序。

还有些轨迹控制的应用中，目标点的位置是根据运行中临时指定的，如手动示教、机械 视觉的引导，这些“运行时决定”的轨迹控制，就需要采用另外的轨迹格式（SMC_CNC_REF）来描述，执行前进行变量关联，形成控制轨迹。

综上所述三种轨迹描述模式，对应编译选项中有3种编译格式选项，每种编译格式的特性分别如下表：

编译格式	执行功能
SMC_OutQueue	这是插补功能块 MC_Interpolator 能接受的唯一数据流格式；该格式可以被其他速度和轨迹圆滑等功能块进行处理； 数据流中每一行指令的插补模式与操作数均为确定的数据；
SMC_CNC_REF	该格式是将用户G-Code程序读入内存后的文本数据串，其中可以能确定，需要解码后才能成为OutQueue格式；
File	以文本文件方式存在的轨迹代码，用户可以对文件进行读写操作。 File方式尤其适合终端用户需临时更换控制轨迹、或线段数量巨大的轨迹控制应用，File 可由控制器读取，或由上位机写入

从上表可以看出，SMC_OutQueue是最基本的轨迹格式，而File、SMC_CNC_REF格式则需要经过控制器内部的相关功能块处理，最后得到统一的SMC_OutQueue格式，才由轨迹插补器（InterPolater）进行插补运算，产生轴控数据。

5.2 控制器轨迹数据的来源与处理功能块

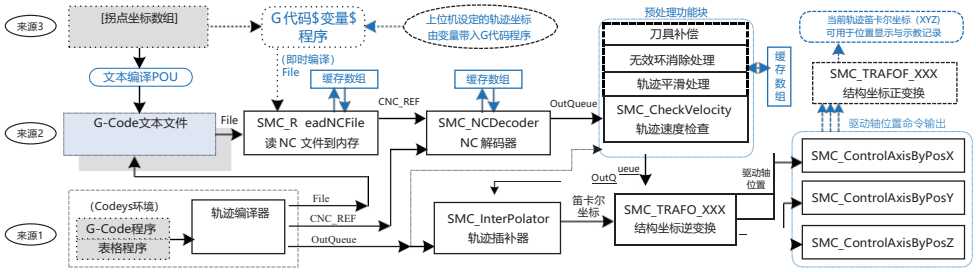
eXtreme、Agile的用户轨迹数据可以有多种输入方式，对应的编译处理环节也稍有不同。分别有如下三种类型：

1. 由PC运行McEngine Pro对轨迹程序编译后，与用户程序一起下载到控制器；

2. 由PC或HMI将G-Code轨迹程序，以文本文件形式单独下载到控制器内部的虚拟盘；

3. 由示教器、HMI或机械视觉控制器，将轨迹拐点或运动目标点，以点坐标数组的形式，写入控制器数据内存中。

三种轨迹输入方式各有适合的应用设备类型，不同轨迹数据类型，在编译解码时需要使 用对应的功能块来处理。下图为CNC轨迹处理的综合流程图，其中最后的插补器、坐标变换和驱动轴控制等功能块将在后面讲解，本节主要说明轨迹编译、解码、轨迹预处理等功能块的功能特点：



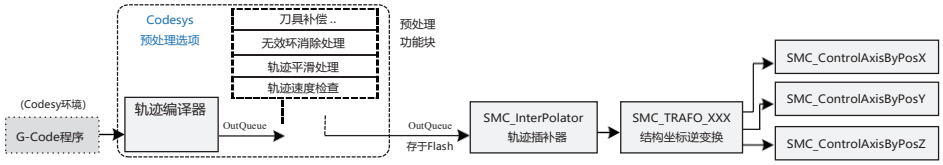
针对用户要求的轨迹曲线输入方式，可以利用其对应的便利性，轨迹曲线的编译会稍有 不同，下面以轨迹曲线编译格式分类说明：

5.2.1 在编程环境中生成的轨迹数据

McEngine Pro编程软件默认的编译输出是OutQueue类型的数据结构，是一种有序的轨迹数据队列，插补功能块SMC_Interpolator（插补器）接受轨迹数据也是OutQueue格式是轨迹数据对象，这是最常用的数据类型，该数据结构记录的是一连串有序的拐点坐标、线型描述、运行速度、加速度、附加轴动作描述、M条件、H动作点描述等完整信息。根据这些信息， 插补器可以按操作的命令临时插补暂停、甚至可作双方向插补（即可以倒退）的运行。因此，OutQueue轨迹数据看作是以系统定义码表示G-Code指令程序，将其中需前后文关联的模式指令、省略表述均以明确完整代码的用户轨迹指令来表示。

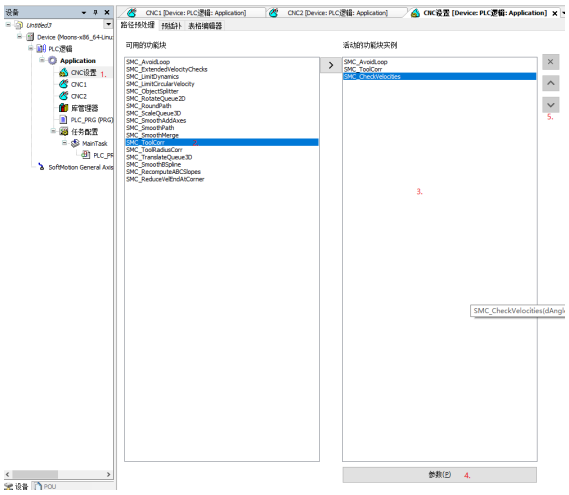
由McEngine Pro编程软件的编译得到的OutQueue轨迹数据，是与机器码格式的用户程序一道，被下载到控制器Flash存储区的，在开始轨迹插补前，控制器会自动将一批OutQueue数据读取到DDR内存缓存后解析执行，以便插补器在插补运算时可对轨迹运行速度连续的预判断、M/H动作的预判断等处理，以提高执行效率。

对OutQueue轨迹数据的执行流程如下图：



McEngine Pro 的可以对OutQueue轨迹数据还可以根据编程选项，对OutQueue作进一步的优化处理， 就如上图中的“预处理功能块”，例如可按用户设定的进行轨迹平滑、倒角等要求进行插补速度优化，检查轨迹拐点处的轴速度是否存在突变，减小轨迹运行中的机械冲击，这些预处理后的轨迹数据仍然是OutQueue数据格式。

由于在McEngine Pro中的G-Code轨迹往往是确定的，对其轨迹的预处理也可在编程时进行处理，并仿真观察预处理的效果。预处理的设置方法和步骤如下图所示：



①在工程的结构树中双击“CNC设置”，进入CNC设置窗口，选择“参数预处理”页面；

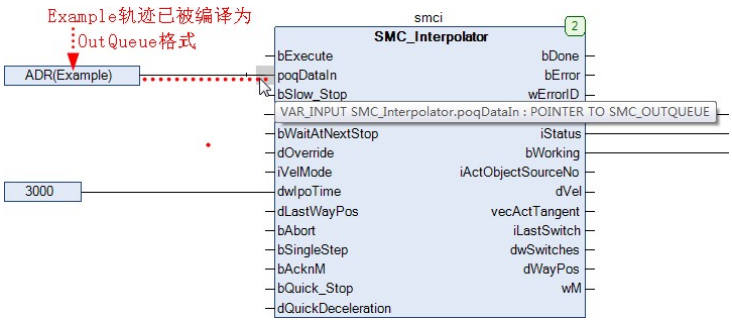
②点击选中需要加入的预处理功能块，

③点击按钮“>”添加到激活窗口，

④点击按钮“参数P”，可以在弹出的窗口中，对所选功能块实例设置参数。

⑤注意功能块实例的顺序，可以通过^或 v 按钮，可以将光标选中的功能块执行顺序前移或后移；

关于预处理功能块的功能特点，合理的执行顺序，可参考本章后续的预处理功能块说明。在eXtreme、Agile控制器中，OutQueue 轨迹数据结构被定义为 SMC_OutQueue 型变量，相关功能块通过指针（Pointer to SMC_OutQueue）或地址来引用。例如用户编写的文件名Example的G-Code轨迹程序，Codesys 编译为OutQueue格式， 输入插补器时，可以用如下的变量名方式指定，ADR()表示取地址操作：



在有些应用中，根据需要从几个OutQueue格式的轨迹中，挑选其中一个轨迹来执行，就可以声明一个指针变量，再根据需要赋值来实现，例如：

```
poqPath:Pointer to SMC_OutQueue;        //声明指针变量 poqPath
….
if router=1 then        //根据变量router 的取值来选择轨迹
poqPath:=ADR(Example1); //条件成立，指针指向轨迹 Example1 else
poqPath:=ADR(Example2); //否则指针指向轨迹 Example2
end_if
…
Interpolater1(, ,poqDataIn:=poqPath, ,,);    //插补器执行poqPath 指定的轨迹
…..
```

从技术原理来讲，功能块SMC_Interpolator默认使用了OutQueue轨迹数据队列缓冲区，这样才能做到相连线段的速度预判，这个缓冲区无需编程者声明。

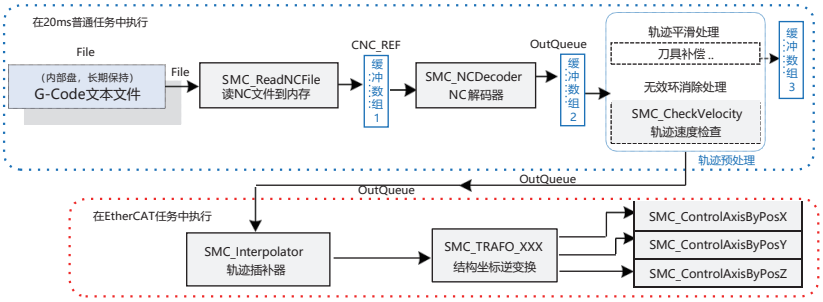
举例来说，若上例代码中的变量router与条形码扫描器读取的读数相关联，就不难实现根据扫描得到的条码，自动选择控制器内存中的轨迹文件，实现对应的轨迹控制工艺。

5.2.2 不同优先级任务POU之间的数据交互

eXtreme、Agile控制器支持文本格式G-Code文件的写入，用户可以在PC上用文本编辑器编写所需的G-Code轨迹程序；或由 CAD/CAM 工具软件设生成的G-Code，以文本文件保存，通过上位机（PC 或 HMI）通过以太网向控制器内部虚拟驱动盘，写入该轨迹文件。在McEngine Pro中可以直观地看到 eXtreme、Agile 的虚拟驱动盘，如下图：

这种File轨迹文件，其中的字符都是以ASCII字符来表示，可用文本编辑器阅读观察或修改，因此称为文本文件，这种文件使用了模式指令后，往往需要上下文关联阅读，才能准 确理解指令要求。

eXtreme、Agile需要先用功能块SMC_ReadNCFile读取到控制器内存区（可以 NC_REF 进行索引或路径指向），然后用功能块 SMC_SMC_NCDECODER_1ecoder对该数据（也称为NC_REF所向的对象）进行解码，得到无歧义的OutQueue格式轨迹数据结构，供后续的插补器进行插补运算。



File轨迹文件方式尤其适合需经常更改轨迹的设备应用，例如木工、雕刻等设备的加工轨迹控制，用户可在PC上用CAD/CAM工具软件设计好加工轨迹，编译生成G-Code文本文件，通过U盘发布，操作者可由PC/HMI 将之写入EXTREME内部虚拟盘；

有的G-Code文件描述比较复杂的轨迹很多，导致代码量很大，读取文件、解码文件需要很长时间，因此无法在EtherCAT任务中一次执行完毕，这样将插补之前的轨迹处理，都放在普通任务中执行，而将插补、坐标变换、轴控等需要实时处理的工作才放在EtherCAT任务中执行，这两个任务之间所需交互的OutQueue数据流，插补器通过内置的FIFO数据队列来接受数据，让该区域堆放几十个待插补的连续轨迹段，方便插补器作前瞻判断，每完成一段轨迹的插补，就会清除已执行的轨迹；普通任务中的预处理功能块会检查缓冲数组 2） 和 1）的轨迹数据剩余数，一旦数据余量不足，就会增加读取、解码、预处理等功能块的执行量，以分别补充缓冲数组 1）、2）的数据量至合适的程度。这种处理机制，使轨迹运行启动延迟时间缩短，还可减少轨迹处理的所需内存量。注意缓冲数组 1）、2）是需要声明的， 其 sizeof 一般设为 50~100。

采用 File 文件方式输入轨迹，还有一个重要的功能特点，就是用户编写的 G-Code 轨迹文件中，可以包含有变量定义的轨迹坐标，在执行时，用户程序对这些变量赋值，决定其运

行轨迹，这在现场示教法设定运行轨迹，或机械视觉引导的运动机构控制应用中，非常灵活。 利用变量定义的 G 代码语句举例如下：

```
N080  G01  X$x_pos$  Y$y_pos$  F8  E130 E-130
N090  G01 Y$z_pos$
```

上面G代码语句中两个$符号中间的字符串，就是用户定义的变量，如X$x_pos$表示由变量x_pos代表的X目标地址，同理，y_pos、z_pos也是用户定义的变量。

该G代码文件首先由功能块SMC_ReadNCFile读取，并转换为SMC_NC_REF类型指向的数据结构块存于内存；

然后由解码功能块SMC_SMC_NCDECODER_1ecoder对SMC_NC_REF指向的数据块执行解码时，会自动将其中变量以其当前值代入，解码为OutQueue格式的数据，供后续的插补器插补。File轨迹文件所支持的变量特点，为用户现场数据的加入，提供了灵活性。

请注意File文件的执行特性：

◆ 采用变量输入的方式，每执行一遍File，就需要重新解码装载一次变量参数，因此， 逻辑程序需要在每次解码之前，确定和更新变量参数；

◆ 若File只有一行G代码的轨迹文件执行时，运行到目标点后会停下来，待下一个轨迹命令解码完成后，再启动运行；由于插补器无法做到轨迹预判，这样就会有运行 的停顿现象；

◆ 若File有多行G代码，在执行这几行G代码之前，插补器可以预判轨迹，运行速度才可能做到连续；

◆ 以 File 方式编写的用户程序，轨迹坐标可以改变，但线型不能改变，若需要改变线型，可以通过写有不同线型的 File，根据需要进行挑选的编程方法来处理。

文件读取、解码、速度检查操作的编程思想与 C 语言的编写相似，采用ST语言编写更方便，举例：

```
//程序变量声明---------->
PROGRAM CNC_PreparePath VAR_INPUT
xStart: BOOL;
sFileName: STRING := ‘_cnc/CNC1.cnc’;        //被读的文件路径和文件名
END_VAR
VAR_OUTPUT
xStartIpo: BOOL;
poqPath: Pointer to SMC_Outqueue; END_VAR
VAR
iState: INT;
SMC_ReadNCFile_1 : SMC_ReadNCFile; SMC_NCDecoder_1: SMC_SMC_NCDECODER_1ecoder; SMC_CheckVelocities_1: SMC_CheckVelocities;
agcwBufReadNCFile: ARRAY[0..99] OF SMC_GCODE_WORD; //声明读缓冲区，为100 个记录
agiBufDecoder: ARRAY[0..99] OF SMC_GeoInfo; //声明解码缓冲区，为100 个记录
END_VAR.
```

```
//轨迹读取、解码、速度检测处理----------> poqPath:=SMC_CheckVelocities_1.poqDataOut;
CASE iState OF      //控制器上电时，状态指针iState=0 0:          // idle，等待状态
IF xStart THEN      //当 xStart=1 时，本程序执行一次初始化，将所有功能块触发标志清 0 iState:=10;
xStartIpo := FALSE;
SMC_ReadNCFile_1(bExecute:=FALSE); SMC_NCDecoder_1(bExecute:=FALSE, ncprog:=SMC_ReadNCFile_1.ncprog); SMC_CheckVelocities_1(bExecute:=FALSE);
xStart:=FALSE; END_IF
10: //读 G-code 文件、解码、速度检查处理状态
SMC_ReadNCFile_1(
bExecute:=TRUE ,  //启动读文件功能块
sFileName:=sFilename ,    //读指定路径和文件名的文件pvl:= ,
pBuffer:=ADR(agcwBufReadNCFile) , dwBufferSize:=SIZEOF(agcwBufReadNCFile) ,
bExecuteDecoder=> ,       //一旦缓冲区满，置位，可用于启动解码操作
```

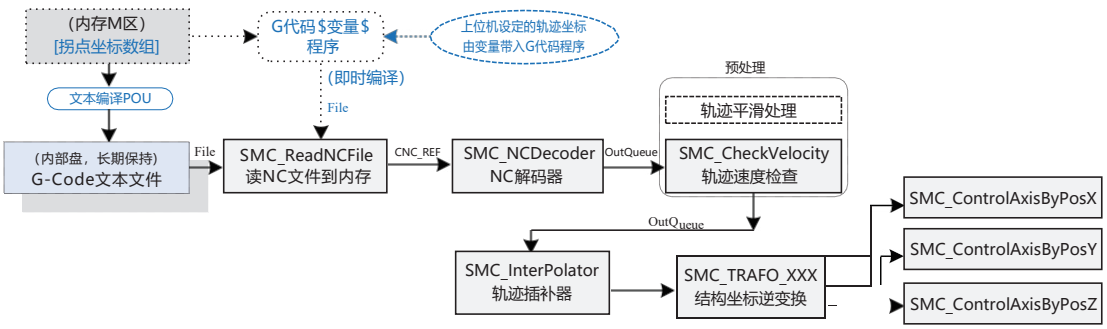
```
ncprog=> );
SMC_NCDecoder_1(
bExecute:=SMC_ReadNCFile_1.bExecuteDecoder , //读缓冲区满，启动解码操作nSizeOutQueue:=SIZEOF(agiBufDecoder) ,
pbyBufferOutQueue:=ADR(agiBufDecoder) ,
ncprog:=SMC_ReadNCFile_1.ncprog ,
GCodeText=> );
SMC_CheckVelocities_1(
bExecute:=SMC_ReadNCFile_1.bExecuteDecoder , //读缓冲区满，启动速度检查操作
poqDataIn:=SMC_NCDecoder_1.poqDataOut ,
poqDataOut=> );

xStartIpo := SMC_ReadNCFile_1.bExecuteDecoder; //读缓冲区满，启动 EtherCAT 任务中的插补运算
IF NOT SMC_CheckVelocities_1.bBusy THEN
iState:=0;    //当所有的轨迹的速度检查出来执行完毕，返回iState=0 等待状态
END_IF END_CASE
```

5.2.3 拐点坐标定义的轨迹数据

有些CNC轨迹运动控制的应用，如简易搬运机械手、上下料机构的控制，用户要求对轨迹的起点和终点的要求位置准确，对中间的轨迹线型和精度要求则并不高，可通过现场示教 所需的折线轨迹拐点，并按顺序记录折线的所有拐点坐标，以数组或数据结构存储，或以File文件方式进行保存，以便数据发布。

对于以拐点坐标数组定义的轨迹曲线，设法将坐标数组转换为G-Code文件格式，就可以沿用File文件轨迹处理流程了，如下图中蓝色字体和虚线框部分：

有两种方法将拐点数组变换为轨迹文件：带变量 G 轨迹程序赋值法、文本编译法。现分别介绍如下。



5.2.4 带变量G轨迹程序赋值法

这在轨迹示教的单步运动控制时，需要使用这个方法，让机械部分运行指定位置或姿态，再按键记录当前的位置坐标，最后形成轨迹数组。

这首先要求用户程序中具有带变量参数的G-Code用户程序，例如如下语句：

```
n20  g10  X$P_X$  Y$P_Y$  Z$P_Z$  M$M_cmd$  H$H_cmd
```

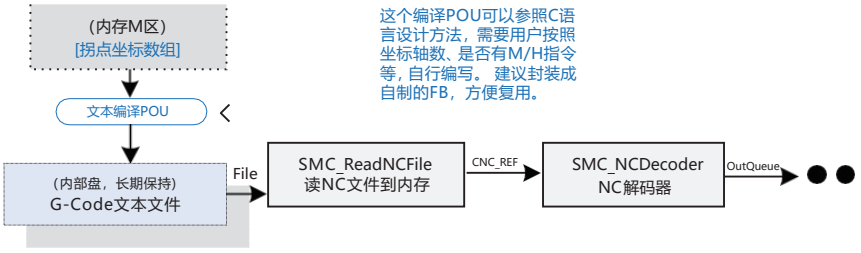
其中XYZ的坐标、M/H指令参数是分别以变量表示，每个变量以两个$分隔符区隔，上例中XYZ的目标坐标就分别为全局变量P_X，P_Y，P_Z。M、H指令的参数是M_cmd、H_cmd，是否需要这些变量，编程人员可根据控制对象酌情决定。

若上位机（手操器、示教器或HMI等）给定了一组X/Y/Z坐标，令控制器定位到该坐标，只要将这些坐标带入具有上述代码行的G-Code程序，再ReadNCFile读取,SMC_NCDECODER_1ecoder解码后，由 InterPolater 插补执行就可以了。

在运动轨迹示教完成后，需要自动连续运行时，也可以采用这个方式控制，但会出现每 段轨迹都有一次加速和减速，影响效率，这是因为插补器无法预测下一线段或圆弧的起止坐 标和速度要求，没法做到目标位置缓冲。要改善这种现象，需要采用下面的文本编译法，先将多个拐点的数组，编译成无变量的G-Code文件，让控制器进行处理，运行速度就顺畅了。

5.2.5 文本编译法

利用eXtreme、Agile系统支持的文字串处理功能库，用户自行编写一个“编译程序 POU”，在eXtreme、Agile的虚拟盘中的创建一个G-Code文件（例如.\CNC\demo1.cnc），将每个拐点的XYZ坐标，编译为一行G-Code程序，添加到该文件中，直到将所有拐点的坐标，都添加到该文件后，保存文件。在采用经典的轨迹文件读取、解码、插补等功能程序来处理就可以了。



由上图可知，文本编译POU是轨迹数组处理的核心，现举例如下，编写POU功能如下：由示教器记录的数组翻译成文本文件格式的G-code程序，存到硬盘上，数组的格式如下，示教数组行宽度为固定的24word宽度，每行代表一个轨迹目标拐点坐标、线型的说明，第0行按数组说明进行解析，编译时可忽略该行的内容：

0	总步数1W	文件名长度2W	文件名字符串，以 Unicode 编码表示，最多20Word

从第1行开始，数组按如下数据结构进行解析：

步数序号 1W	线型 1W	目标坐标（REAL 类型，各占 2Word）						速度 2W	加速度 2W	减速度 2W	M 1W	H 1W	备用 2W
		X	Y	Z	A	B	C						

定义说明如下：

1. 行编号需大于0，按1…65535 从小到大顺序编写，可以不连续，以“N010xx”，这样便于后续的G-code代码执行时的位置行显示；

2. 坐标/速度/加速度/减速度等，在数组中为REAL类型数据，翻译时转换为 LREAL 类型；

3. 当M、H为0时，表示没有M、H功能，编译时对其在G-code行不予编译；

3. 当线型类型字定义：

1=记录无效，跨过

10=定位

11=直线

12=圆弧

0=记录无效，表示用户轨迹程序完毕，编译结束。

根据上面的定义，翻译成文本格式的 G-Code 轨迹程序 POU 程序可以设计如下：

```
PROGRAM Trancefer          //------- POU 的变量声明开始
VAR
    i: DINT;
    bExcute: BOOL; number: DINT; istate: DINT:=0;
    bExcuteold: BOOL; ErrorID: STRING(256);
    bError:BOOL;
    {attribute ‘instance-path’}
    {attribute ‘noinit’} StrinDest:STRING:=’’;
    sFileName: STRING := ‘_cnc/CNC3.txt’;
    hFile:                    CAA.HANDLE
    filop:                     FILE.Open;
    filwr:                     FILE.Write;
    filrd:                     FILE.Read;
    filcl:                     FILE.Close;
```

```
    filsp:          FILE.SetPos;
    filgep:         FILE.GetPos;
    filges:         FILE.GetSize;
    hangnumber: STRING;
    G_CODE: STRING;
    bDone: BOOL;
    szFileSize1:          CAA.SIZE := 0;
    szFileSize2:          CAA.SIZE := 0;
    TargetPos1: LREAL;
    TargetPos2: LREAL;
    TargetPos3: LREAL;
    Velocity: LREAL;
    ACC: LREAL;
    DCC: LREAL;
END_VAR
    //-------变量声明结束
```

```
//轨迹编译程序---------：
//执行上升沿切换到步骤1
IF bExcute AND NOT bExcuteold THEN
        istate:=1;
END_IF
bExcuteold:=bExcute;
CASE istate OF
        1:
        (*GVL.Save_Word  数据存放示教点位数据，格式规范参考示教拐点数据说明数组的格式如下， 示教数组宽度为固定的 24word 宽度，每 24word 代表 CNC 一行 G 代码*)
//0 行：GVL.Save_Word[0]~GVL.Save_Word[23]
IF GVL.Save_Word[0]=0 THEN
        IF GVL.Save_Word[1]>0 THEN
                number:=GVL.Save_Word[1];//获得总步数
                istate:=2;
                ELSE bError:=TRUE ;
                        ErrorID:=’无解析数据’;
        END_IF
END_IF
    filop( xExecute:=FALSE);
    filgep( xExecute:=FALSE);
```

```
    filsp(xExecute:=FALSE);
    filwr(xExecute:=FALSE);
    szFileSize1:=0;
2:
  (* 打开文件为数据存成 G 代码文件做准备*)
    filop.sFileName:=sFileName;//存储文件名以及路径
    filopeFileMode:=FILE.MODE.MWRITE;//写模式
    filop.xExclusive:=TRUE;
    filop( xExecute:=TRUE);
    IF filop.xDone THEN
        hFile:=filop.hFile;
          //StrinDest:=' ';
          istate:=3;
          i:=1;//0行为头数据,1行开始为点位数据
    END_IF
    IF filop.xError THEN
        bError:=TRUE ;
          ErrorID:=存储路径失败';
    END_IF

3:
    hangnumber:=DINT_TO_STRING(GVL.Save_Word[i*24]*10);//步数序号
    StrinDest:=CONCAT(STR1:=StrinDest,STR2:='N');
    StrinDest:=CONCAT(STR1:=StrinDest,STR2:=DINT_TO_STRING(GVL.Save_Word[i*24]*10));
    StrinDest:=CONCAT(STR1:=StrinDest,STR2:='');
    IF GVL.Save_Word[i*24+1]=10 THEN
        G_CODE:='G0';
    END_IF
    IF GVL.Save_Word[i*24+1]=11THEN
        G_CODE:='G1';
    END_IF
    IF GVL.Save_Word[i*24+1]=12 THEN
        //G_CODE:='G2';
        //G_CODE:='G3';
    END_IF
    TargetPos1:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+3],L:=GVL.Save_Word[i*24+2]);
    TargetPos2:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+5],L:=GVL.Save_Word[i*24+4]);
    TargetPos3:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+7],L:=GVL.Save_Word[i*24+6]);
```

```
Velocity:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+9],L:=GVL.Save_Word[i*24+8]);
ACC:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+11],L:=GVL.Save_Word[i*24+10]);
DCC:=0;//HEX_TO_REAL(H:=GVL.Save_Word[i*24+13],L:=GVL.Save_Word[i*24+12]);
StrinDest := CONCAT(STR1:=StrinDest,STR2:=G_CODE);
StrinDest := CONCAT(STR1 :=StrinDest, STR2 :=' X');
StrinDest := CONCAT(STR1:=StrinDest,STR2:=LREAL_TO_STRING(TargetPos1));
StrinDest := CONCAT(STR1:=StrinDest,STR2:=' Y');
StrinDest := CONCAT(STR1:=StrinDest,STR2:=LREAL_TO_STRING(TargetPos2));
StrinDest  := CONCAT(STR1:=StrinDest,STR2:='Z');
StrinDest := CONCAT(STR1:=StrinDest,STR2=LREAL_TO_STRING(TargetPos3));
StrinDest := CONCAT(STR1:=StrinDest, STR2 :='F');
StrinDest := CONCAT(STR1:=StrinDest, STR2 :=LREAL_TO_STRING(Velocity));
StrinDest := CONCAT(STR1:=StrinDest,STR2 :=' E');
StrinDest := CONCAT(STR1 :=StrinDest, STR2 :=LREAL_TO_STRING(ACC));
StrinDest := CONCAT(STR1 :=StrinDest, STR2 :=' E');
StrinDest := CONCAT(STR1 :=StrinDest, STR2 :=LREAL_TO_STRING(DCC));
StrinDest:=CONCAT(STR1 :=StrinDest, STR2 :='$r$n');
istate:=4;


   4:
filsp(hFile:=hFile,udiPos:=szFileSize1,xExecute:=TRUE);
szFileSize2:=SIZEOF(StrinDest);
filwr(hFile:=hFile,pBuffer:=ADR(StrinDest),szSize:=szFileSize2,udi TimeOut:=100000,
xExecute:=TRUE);//写入文件
IF filwr.xDone THEN//完成一行后跳转执行下一行数据
        i:=i+1;
        IF i>number THEN
            istate:=5;
            ELSE istate:=3;
        END_IF
        szFileSize1:=szFileSize2+szFileSize1;
        StrinDest:='';
        filwr(xExecute:=FALSE);
        filsp(xExecute:=FALSE);
END_IF
IF filwr.xError THEN
    (* error handling*)
    ;
```

```
        END_IF
5:
        filcl.hFile:=hFile;
        filcl( xExecute:=TRUE);
        IF filcl.xDone THEN
                bDone:=TRUE;
        END_IF
        IF filcl.xError THEN
                (* error handling*)
                 ;
        END_IF
END_CASE
```

若将上面的POU编写为FB，就可以形成示教型机械手专有的FB，方便在类似的应用中沿用。将表格式的轨迹转换为G-Code代码，可使得运动机构的速度不至于反复停顿，保持连续速度运行。

第六章.典型的轨迹运动控制程序的构成

本章讲解的轨迹控制的核心部分，如何将 OutQueue 格式数据，经过轨迹插补、轴坐标变换、轴控处理等，由此可以实现从 3 轴垂直坐标 CNC、SCARA、DELTA、多关节 ROBOT 等设备控制应用。

运动轨迹控制类的设备结构可能千变万化，用户轨迹曲线描述方式也有多种方式，为了 保证这些控制运算的易用与灵活性，PLCopen 编程体系中，将运动控制的运算与控制分成几种功能块，用户编程时，根据应用系统的结构类型，选择对应的功能块组合，就可以简单地 实现满足所需特性的控制程序了。

每个功能环节简介如下：

6.1 插补运算

功能块SMC_Interpolator将OutQueue格式的轨迹数据进行插补，每次运算得到用户轨迹上的一个点位置，由功能块的piSetPosition端口输出各轴的下一周期的目标位置指令，其中包含了XYZ插补轴的坐标、ABCUVWPQ辅助轴的坐标等数据，准确地说插补功能块输出的是一个数据结构。

6.2 坐标变换

坐标变换功能块与选用的运动机构的结构类型有关，其功能是由插补计算得到轨迹点XYZ垂直坐标，推算出对应驱动轴所需的运行位置。

上例用于控制3轴龙门架结构，因此选用了SMC_Trafo_gantry3坐标变换功能块，由于变换前后都是垂直坐标系，且目标点坐标与伺服轴的位置成线性关系，无需坐标系的旋转变 换，故其计算最为简单，只需对XYZ轴的坐标作适当偏移就可以了。若设备操作者有临时性的参考坐标修改，如机床的原点偏移，只需要修改该功能块输入侧变量的偏移值即可

若用户坐标系与运动机构轴机构并不平行，而是存在偏转角度，就是在此使用坐标偏转 变换运算功能块；

对于2或3轴SCARA 机械手结构，目标点坐标与伺服轴的位置成非线性关系，需要SCARA极坐标变换，McEngine Pro 中提供了2或3轴Scara框架库功能块；

对于3轴Delta机械手结构，就需要按Delta变换；依此类推，若用与6关节机械手的控制，就选用6关节机械手的坐标变换功能块。McEngine Pro中提供了对应框架库的坐标变换功能块；

总而言之，用户编程时，需要按应用系统的结构类型、结构尺寸等，选用对应的功能块 并进行参数设置，以便推算出要到达目标点，各驱动轴必需运行到达的轴目标位置。

6.3 轴位置控制

将每个轴的位置位置命令，写入对应轴控制数据结构，由EXTREME、AGILE系统自动发送给相应的伺服驱动器，使轴运动。该功能块还可根据本轴设定的最大速度、最大加速度、最大减速度设定值，进行分析检查，若有超出设定值，或接到伺服轴的告警反馈（bError/bStopIpo），将及时反馈给插补器，以降低轨迹插补的运行速度。

功能块SMC_ControlAxisByPos看似没有参数输出，但输入给该功能块的SetPosition端口的参数，就会发送给相应的伺服驱动器，作为Axis.SetPositon位置命令，使轴运动。

6.4 异常状态的减速与停机处理

检查各轴当前的负载状况，若某轴位置指令的运行速度、加减速度超过设定值、伺服载荷超过其过载保护点，将此状态信号反馈至插补器，令插补速度降低，以降低轴的负载，保证设备能持续运行。既要设备高速率运行、又要伺服轴不过载、速度调整时不出现时快时慢的抖动，这需要根据轴的载荷、目标点要求精度，对设定插补速度、插补降速时间等可控量之间，有合理的 PID 的调节策略，调试到敏稳准的程度。

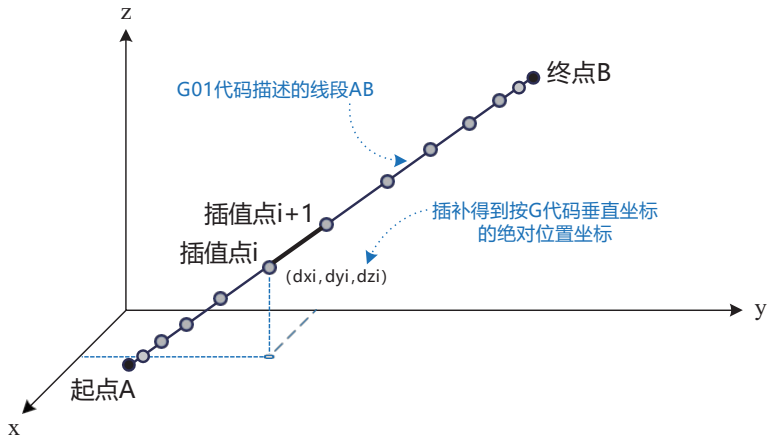
检查各轴的当前异常状态，若任一轴出现故障停机，应命令插补器停止插补，及时停机；实际控制中还应加入更多的监视处理，比如伺服轴位置到达位置滞后时，也需降低轨迹插补器的运行速度，避免出现运行轨迹与设计轨迹有差异的情况，影响控制精度。编程者可以举一反三，将希望避免的运行误差、错误进行检查，及时应对处理。

6.5 插补功能块SMC_Interpolator详解

对于 CNC 轨迹插补控制来讲，SMC_Interpolator 功能块是一个重要的插补功能块，其功能强大，但编程者使用简单。相比于以往的脉冲控制方式的插值原理，基于 EtherCAT 控制方式的插值要“简单”许多，因为控制器只需要将指定轨迹曲线中，插补得到有限个离散的中间点，用数学方法计算得到这些中间点的坐标（dx，dy，dz），让合成轨迹能以设定的加速度、减速度、设定速度等运行，插补器支持 DIN66025 标准中的所有 G、M、H 指令的解析执行。

我们可以将这些中间点坐标经过必要的坐标变换处理后，以位置命令方式发送给驱动XYZ 轴的伺服，令三个轴同时运动，这三个轴运动的合成轨迹即为所需的运动轨迹。

如下图所示为一个直线线段 CNC 插值的原理图，因为在线段的轨迹运动中，存在加速和减速的过程，中间的插入的间距会有加速时间距逐渐加大、减速时间距逐渐减小的情况。

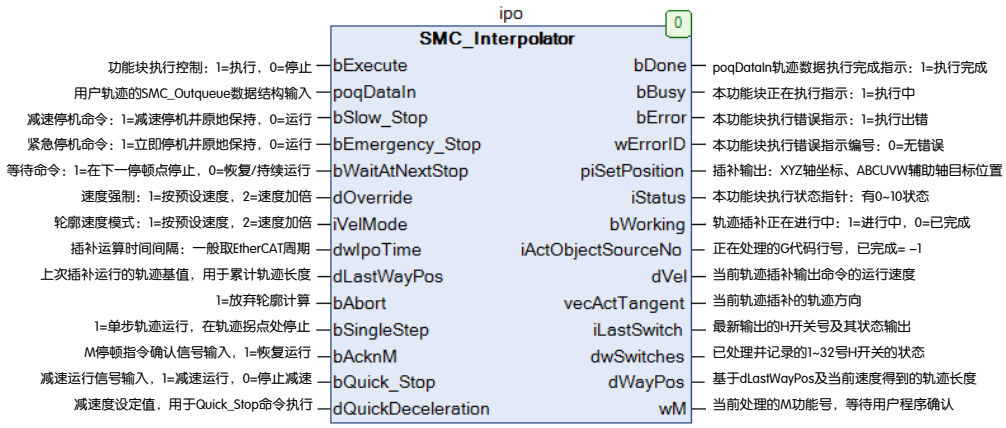


因EtherCAT中断任务是按固定的周期执行，如周期一般设为1ms、2ms或4ms等，插补周期与EtherCAT周期相同，一旦开始插值运行后，每次 EtherCAT 中断时执行一次SMC_Interpolator功能块，该功能块就以一个 EtherCAT 周期内，按照用户设定的轨迹速度、允许的加减速时间，计算每个轴在接下来的这个周期内，需要运行的距离，以 G 代码的坐标系为参考，按照垂直坐标系目标绝对位置坐标（dx,dy,dz），作为令 XYZ 轴伺服运行的位置指令。在插补处理的同时，对 ABCPQUVW 等辅助轴同时进行对应的定位控制处理。

SMC_Interpolator插补功能块还能处理用户G-Code程序中使用到的刀具补偿、倒角、平滑处理等等，都由该功能块引发对插补轴的处理。

SMC_Interpolator插补功能块不仅具有轨迹插补，还提供了单步执行、外部异常时暂停与恢复运行、中途减速停机、紧急停机等控制功能；还提供了用户M、H指令的输入输出控制、当前运动速度的监控、插补运行轨迹长度的累计等功能，借助这些功能变量，用户程序可以 实现灵活的 CNC 插补应用功能。

SMC_Interpolator 功能块的输入输出参数及其简介如下图



1.SMC_Interpolator 功能块的执行时，必需使用到的几个输入输出变量

poqDataIn：用户需要执行的轨迹数据结构指针变量，指向所需进行插补执行的用户轨迹 数据结构；

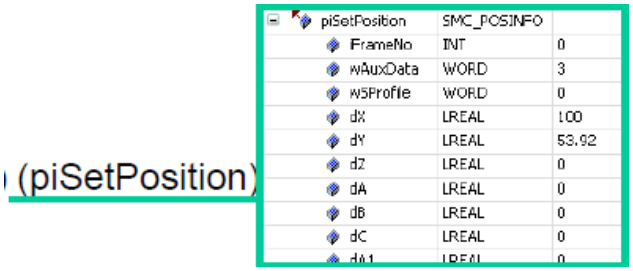
bExecute：功能块运行控制信号，

bEmergency_Stop：紧急停机命令，当外部结构出现位置超限、伺服告警等异常，或操作人员需要紧急停机时，就给该输入参数端口置位“1”，就可以令插补器停止运行，各伺服也就立即停机；当该信号端复位清零“0”，且 bExecute 端口为“1”状态，即可恢复运行；

dwIpoTime：插补运算周期间隔，一般设定为与 EtherCAT 任务周期相同；

bDone：轨迹插补完成信号，用户程序可以根据这个状态标志，进行后续的逻辑处理；

piSetPosition：轨迹插补得到的目标位置数据结构，包含 XYZ 轴及辅助轴在接下来的 EtherCAT 周期内目标坐标，是基于 G 代码坐标系的绝对位置坐标，用于后续的轴坐标变换、位置控制，该数据结构提供的详细数据信息可在 Codesys 中监视观察到，如下图：



iStatus：用于指示本插补功能块执行状态，用于后续轴跟随运动的状态控制，该输出变 量为枚举类型，其状态值的意义如下：

iStatus	插补器状态指示
0	未知状态。一旦完成一次插补后，就不会有该状态
1	模块正在初始化，poqDataIn 数据尚没装载完毕
2	模块正在加速运行
3	模块正在恒速运行
4	模块正在减速运行
5	已完成对 GeoInfo 列表的处理，不再对后续 poqDataIn 数据进行处理

iStatus	插补器状态指示
6	模块就会处于等待状态，可能由如下任一原因导致： bEmergency_Stop=1； bSlow_Stop = 1 以及 dVel=0； bWait_At_Next_Stop = 1 以及 dVel = 0；
7	模块正在加加速度运行
8	模块正在减加速度运行
9	模块正在加减速度运行
10	模块正在减减速度运行

iWorking：轨迹插补正在进行中，该状态信号作为后续轴控功能块的使能信号。

bBusy：表示插补功能块正在执行，

bError/wErrorID：表示插补功能块执行出错/出错原因；

2.使用 SMC_Interpolator 功能块的降速暂停功能

在需要按轨迹运行的设备中，常常需要有手动暂停功能，暂停时希望减速停止，避免机 械冲击，当暂停信号取消后，系统自动再次加速运行起来，这就需要用到插补功能块的降速 暂停功能；

在有些执行搬运操作的运动机构中，轻载的时候可以运行的比较快，当抓取了比较重的 工件后，需要降速运行，避免伺服驱动器的过载，此时利用插补器的自动降速功能，让插补 后的合成轨迹速度降低，可避免驱动过载的情况发生；当过载状况消失后，系统可自动加速 运行。配合合适的用户逻辑程序判断处理，可以让应用系统得到比较好的动力学表现。

使用该功能时，用户程序可以根据相关轴负载率，或应用系统的物理量超限情况，经用 户逻辑程序判断处理后，反馈给插补器的 bQuick_Stop 输入端。

若要使用SMC_Interpolator功能块的自动限速功能时，需要如下输入输出变量：

bQuick_Stop：当该状态信号输入为1时，插补输出的轨迹速度会按 wQuickDecelaration的设定值自动减速，直到停机；当 bQuick_Stop=0 停止减速，自动加速到设定速度运行；

wQuickDecelaration：自动降速的减速度设定输入端。

3.使用 G 代码的 M 功能

用户的应用中，若需要轨迹运行中，有暂停动作（轨迹速度=0），等待某条件满足后（用户程序给出对应的确认信号），再继续轨迹运行，就需要使用到 M 功能。典型的应用如上下料机构的控制中，需要在一个特定的工位等待工件的加工完成，其等待时间并不确定，当检 测到工件加工完成后，才继续运动到工件位进行取件操作。

使用 M 功能的原理和方法是：

用户在 G 代码轨迹程序中，在需要暂停等待点，编写一行M指令，指定有M功能号；

SMC_Interpolator 功能块在执行该 G 代码轨迹程序中，执行到 M 指令行时，就会停下来（轨迹速度降为0），并在SMC_Interpolator功能块的 wM 端口输出M功能号；

用户逻辑程序中，需要对 M 功能号与相关 IO 信号进行判断，若条件满足，用户程序就将SMC_Interpolator 功能块的 M 功能反馈端状态 bAcknM 置 1；

当SMC_Interpolator功能块检测到 bAcknM=1，就继续执行用户轨迹程序中该 M 代码行后续的轨迹程序，轨迹速度由 0 开始加速。

因此本质上，M功能提供了“轨迹插补”与“逻辑控制”进行逻辑条件交互的一种机制。用户程序中若用到了M功能，需要使用SMC_Interpolator功能块的输入输出变量： wM：输出的 M 功能号。当 M 号输出有效（大于0）时，轨迹速度已降为0；bAcknM：M 功能确认状态输入，一旦由M功能暂停后，只有该信号置为1，才能恢复轨迹运行。

M指令被执行时，轨迹运行先减速停机，确认后再加速运行的动作，为了提高设备的运行效率，有些应用中，用户往往希望 M 功能对应的条件信号满足时，不要有停顿的情况， 而是直接“跨过”该停顿点。解决方法是在用户程序中，增加SMC_PreAcknowledgeM 功能块，对 M 变量进行预先判断，当条件满足，插补器在该点就不会有停顿了。

4.使用G代码的H功能

用户的应用中，若需要轨迹运行中，经过特定轨迹点时有开关信号的输出操作，以便应用系统的逻辑控制与轨迹控制能精确配合，就需要使用到H功能。与M功能不同，H功能在进行开关量输出的时，不会有轨迹运行的停顿，也无需用户的确认。典型的应用如点胶机控制应用中，当轨迹经过某设定点后，打开注胶开关，开始点胶，当经过设定的点胶终点后，关闭注胶控制开关，因开关注胶开关时，轨迹速度为匀速，可保证注胶量的均匀。

使用H功能的原理和方法是：

用户在G代码轨迹程序中，在需要进行开关量输出操作点，编写一行 H 指令，指定H开关的变化和输出状态；

SMC_Interpolator功能块在执行该G代码轨迹程序中，执行到 H 指令行时，就会在iLastSwitch端口输出H行指定的开关号和状态状态，插补器并不减速停顿，而是继续执行后续的用户轨迹程序。

H功能定义比较灵活，可以定义在线段的起点、终点，也可以定义为（当前起点+L）的距离，也可定义为离（终点-L）的距离，参见 G 代码指令章节说明。

用户程序中若用到了H功能，需要使用 SMC_Interpolator 功能块的输入输出变量：

iLastSwitch：当插补器按照用户轨迹程序执行，满足 H 指令所要求的开关动作条件后，最新输出的有效的H开关编号及其输出状态，为有符号整形数：

iLastSwitch	输出开关号及输出状态
0	无开关输出操作
1,…,32767	数字的绝对值为输出开关编号，符号为正，表示要将该开关置为 1
-1,…,-32767	数字的绝对值为输出开关编号，符号为负，表示要将该开关清为 0

当用户程序中有多个 H 开关指令，这个端口总是只指示最后一个开关的输出状态。

dwSwitches：该变量是一个 32bit 的无符号 DW 型变量，按 bit 位状态来指示在插补器中已经被操作过，且编号为 1~32 的开关的输出状态，

bit0=0/1，代表#1 开关为 0/1 状态；bit1=0/1，代表#2 开关为 0/1 状态，……，

bit31=0/1，代表#32 开关为 0/1 状态

dOverride：速度强制信号变量输入，lreal型变量，若将该变量与手摇轮信号关联，就可以通过手摇轮对轨迹插补的运行方向（对双向插补功能块有效）和速度进行实时修改了。使用该功能块时，还需注意：

1) 若bStartAtEnd=1，轨迹数据必需将编译后的 poqDataIn 结构数据全部装载到缓存；

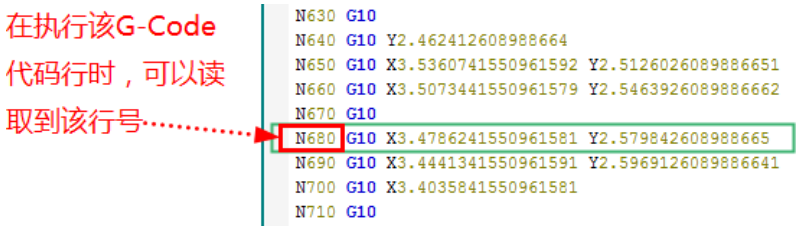
2) 若轨迹数据不能全部装载，应该定义一个 SMC_QueueSetReservedEntries 功能块实例，声明允许倒退插补运行的轨迹队列中的线段数量，例如希望能倒退 10 个线段，就需设置10+3个元件数，其中的 3 个线段用于安全缓存，但是缓存只需适当放大，因为这个缓存的数据并不能用于插补的预先判断。

3) SMC_Interpolator2Dir 功能块需要与 SMC_Interpolator2Dir_SlowTask 功能块配合使用。

5.插补运行的调试与监视功能

bWaitAtNextStop插补器正常连续运行时，该标志应默认清0；若将该标志置1，当插补运行到轨迹的下一个速度为0的点，例如轨迹的拐点时，就会停下来，直到该标志清0，才恢复插补运行，该功能在设备调试时可实现单步运行功能；

iActObjectSourceNo插补器在执行OutQueue轨迹数据时，会将其中的源程序代码行号数值，通过该输出变量指示，上位机可以根据这个变量值，对应指示正在执行G-Code代码行，这在CNC控制应用中常常用到。



6.6 根据运动机构类型选择坐标变换功能块

在实际运动控制应用中，按运动机械结构类型，可分为垂直坐标型、极坐标型、SCARA、Delta、多关节，或者是这些类型的混合型。

人们习惯使用垂直坐标系描述运动的起止点坐标和轨迹线段，G-Code用户程序就是以垂直坐标系来描述用户轨迹。对于控制器而言，控制的是运动机构驱动轴的旋转角度，将机械 的空间坐标，转化为驱动轴的角度位置，就是“坐标变换”。当设备的运动轴机构不呈垂直 坐标架构，或由多个轴运动合成所需的运动时，如6轴机械手的控制，就需要进行几何学的坐标变换，才能得到各运动轴的“轴位置”，该“轴位置”就是对应的伺服马达的位置参数。

由垂直坐标系XYZ坐标（点位置）变换为非垂直坐标系的轴位置，称为运动学逆变换理论上逆变换有可能得到多个数学解，我们需要按轴位置约束、运动连贯性、运动节能等条件选择最优解，这些处理过程需在逆变功能块中判断选择；

由运动机构的轴结构、相关轴位置变换为垂直坐标位置，称为正变换，因此只有唯一XYZ坐标值。这在非垂直坐标机构的应用中，需要能显示当前轨迹XYZ位置、用示教法记录轨迹XYZ位置等，往往需要用到正变换得到XYZ坐标值。

eXtreme、Agile控制器提供了常用运行机构的坐标变换功能块，编程者根据运动驱动轴几何结构选择对应的变化功能块，设置其中的结构尺寸，就可以实现坐标变换了。

坐标变换功能块的执行，是在EtherCAT任务中，紧随轨迹插补之后执行，将每次EtherCAT任务中插补得到的目标位置，都作一次坐标变换为驱动轴的位置数据，供后续的轴控功能块 使用，实现机构轨迹的控制。

在许多运动控制的应用，需要以手操器示教和记录的方法进行编程，因控制器只能读取各伺服轴的位置，而需要记录的是轨迹上的“点位置坐标”，因此需要将轴位置变换为轨迹点坐标（也称拐点坐标），这种由“轴位置--轨迹点坐标”的坐标变换被称为正向坐标变换。正向坐标变换只会得到唯一的空间坐标值。

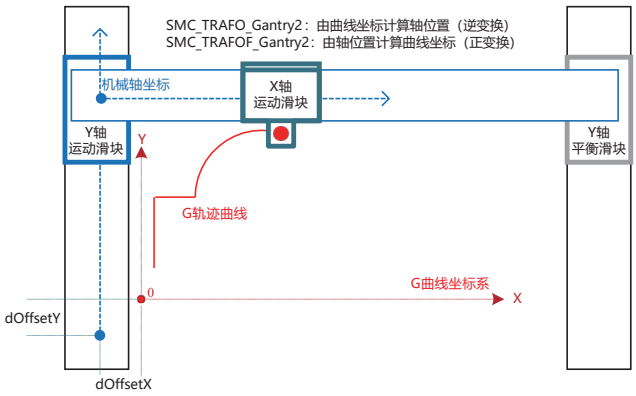
McEngine Pro 中提供了几种典型的结构框架的坐标正向、反向变换功能块，编程时只需按应用系统结构调用相应的功能块即可。

6.6.1 龙门架型运动系的坐标变换

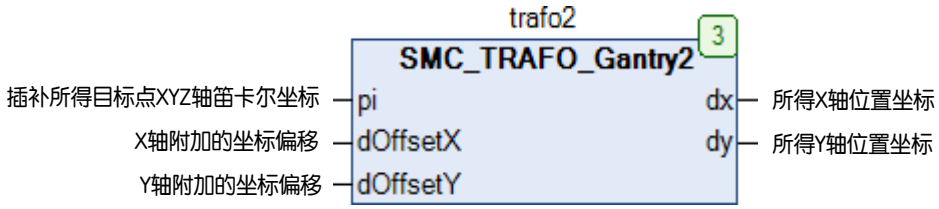
龙门架型运动机构是典型的笛卡尔坐标系，用户轨迹程序、轨迹插补后的输出中间点坐标，也是按笛卡尔坐标系描述，当用户应用系统的坐标轴也与轨迹坐标轴彼此平行，因此坐 标变换的计算最为简单，只需要对每个轴分别进行适当的坐标偏移处理即可，McEngine Pro 中提供了标准龙门架坐标变换功能块。

坐标变换功能块	功能说明
SMC_TRAFO_Gantry2	用于 2 轴垂直坐标运动机构。将 X、Y 两个轴位置指令分别加上 dOffsetX、dOffsetY 偏值后，作为轴位置输出。
SMC_TRAFOF_Gantry2	用于 2 轴垂直坐标运动机构。将 X、Y 两个轴位置指令分别加上 dOffsetX、dOffsetY 偏值后输出，并对 X、Y 轴有最小、最大值限制处理
SMC_TRAFO_Gantry3	用于 3 轴垂直坐标运动机构。将 X、Y、Z 三个轴位置指令分别加上 dOffsetX、dOffsetY、dOffsetZ 偏值后，作为轴位置输出。
SMC_TRAFOF_Gantry3	用于 3 轴垂直坐标运动机构。将 X、Y、Z 三个轴位置指令分别加上 dOffsetX、dOffsetY、dOffsetZ 偏值后输出，只对其中 X、Y 轴 有最小、最大值限制处理

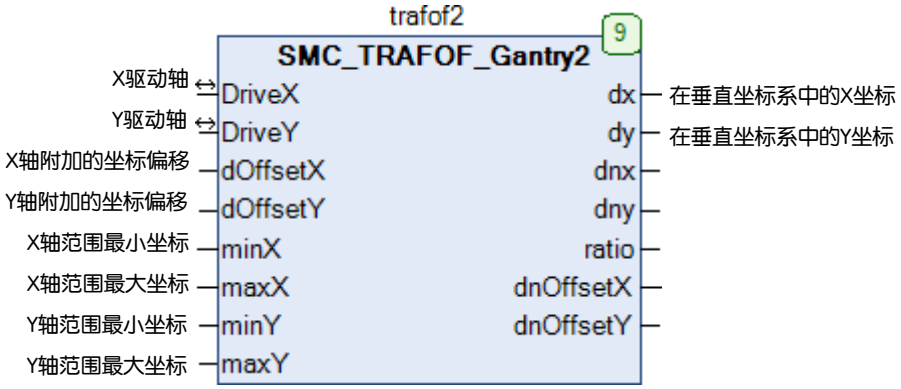
两轴坐标系的坐标变换，只需进行轴偏移量的计算：



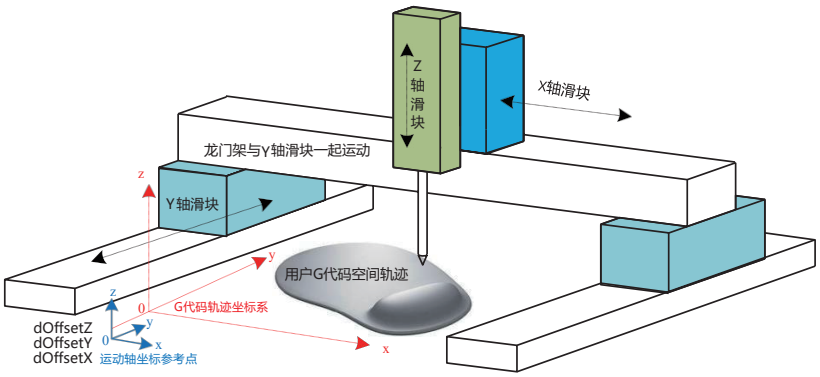
其中的偏移量dOffsetX、dOffsetY作为变量，应考虑机械结构坐标系固有的参考点、用户希望的轨迹参考点在工件上参考点两个方面的偏移量。两轴垂直坐标系统的坐标变换功能 块指令如下图：



垂直坐标系统的坐标正向变换功能块指令如下，由于正向变换后，输出变量将用于轴运动控制，需要按应用系统的机械允许范围进行限制，本功能块对XY轴的运动范围限制处理：

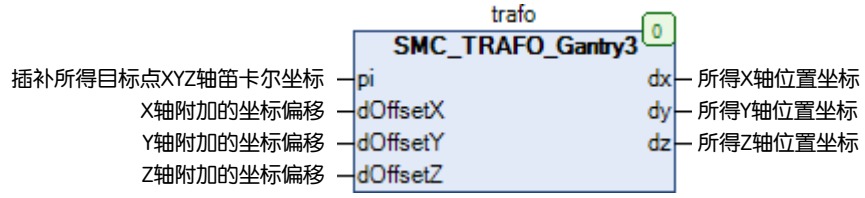


对于三轴垂直坐标机构的坐标变换，同样只需进行轴偏移量的计算：

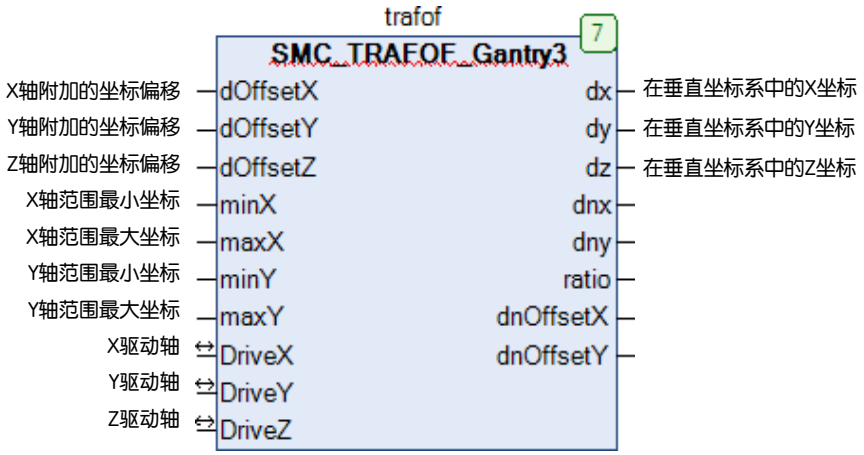


SMC_TRAFO_Gantry3：由曲线坐标计算轴位置（逆变换）
SMC_TRAFOF_Gantry3：由轴位置计算曲线坐标（正变换）

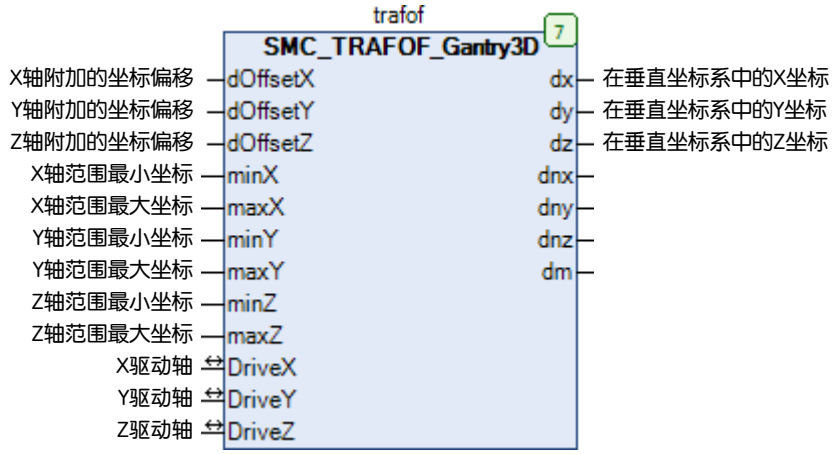
其中的偏移量dOffsetX、dOffsetY、dOffsetZ作为变量，应考虑机械结构坐标系固有的参考点、用户希望的轨迹参考点在工件上参考点两个方面的偏移量。垂直坐标系统的坐标逆变换功能块指令如下，如果只需要对其中的 2 个轴进行变换，也可以使用该功能块，把不需要的轴输入输出端口空缺即可：



垂直坐标系统的坐标正向变换功能块指令如下，由于正向变换后，输出变量将用于轴运 动控制，需要按应用系统的机械允许范围进行限制，本功能块只对其中的XY轴进行了运动范围限制处理：



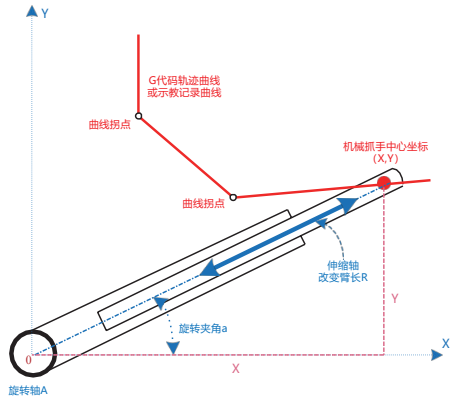
如果需要对XYZ的3个轴运动范围进行限制处理，可以使用如下该功能块：



6.6.2 极坐标型运动机构的坐标变换

平面极坐标型运动机构，是一种旋转+变臂长的两轴运动机构，一个轴使臂旋转，另一个轴驱动臂伸缩，改变臂长，使手掌能在XY平面区域移动，还有第三个轴往往为Z轴方向移动，典型应用如简易上下料机构。为了提高设备的运行效率，希望抓手（工具中心）能按要求的轨迹的两个拐点之间作最短直线的运动。

运动控制中，需要将（X，Y）平面坐标，变换为极坐标（R，a），继而算出变臂长轴位置、旋转轴位置，如下图：



用户轨迹插补后的输出中间点的坐标数据结构也是按笛卡尔坐标系描述，而我们要求控制的是旋转轴、伸缩轴，虽然没有直接的坐标变换功能模块可以直接使用，但根据几何学分析，不难直接编程变换：

因考虑到旋转臂R=已知基本长度 R0+可变长度r，得可变长度r=R-R0；XY平面坐标与极坐标之间有如下关系：

X2+Y2=R2；

Y/X=tg(a)；于是：

r=sqrt(X2+Y2)-R0;----------（1）即为伸缩轴的位置

a=arctg(Y/X); ----------（2）即为旋转轴的角度位置

（1）、（2）式就是将直角坐标 XY 轴逆变换为极坐标A轴、R轴控制的目标位置。而Z轴仍为垂直方向运动，无需变换，可将 Z 坐标分量直接用于相应轴的控制。

许多简易机械手需要有手动示教记录轨迹的功能，即先通过示教器令伺服轴运动，使机械手运动到合适位置，然后记录该坐标点（x，y，z），以此方法记录多个有序的坐标点，形成运动轨迹数组。注意，记录的是垂直坐标系的（x，y，z）坐标。对于极坐标平面的坐标 记录，我们需要根据 R 轴、a 轴的位置，推算出（x，y）坐标，计算方法是：

X=(R0+r)cos(a); ----------（3）即为轨迹的 X 轴坐标

Y=(R0+r)sin(a); ----------（4）即为轨迹的 Y 轴坐标

（3）、（4）式就是将极坐标 A 轴、R 轴的位置变换为直角坐标 XY 平面的坐标。同样，Z轴也无

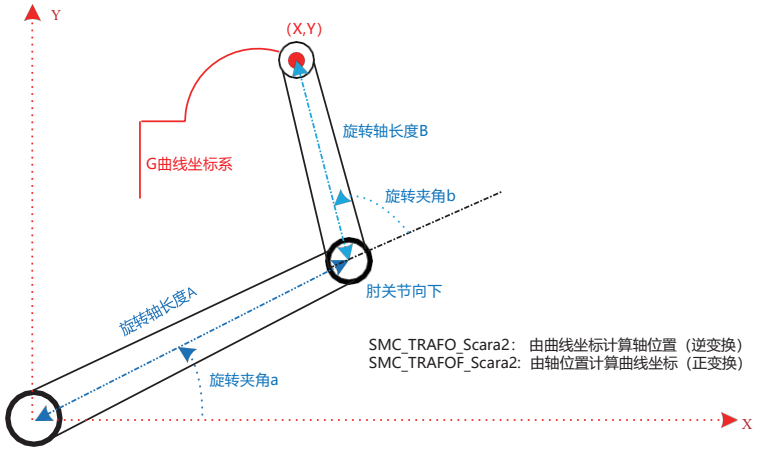
需特别的变换，将Z坐标加上实际坐标偏值，即得其轴坐标值。

> 注意：
>
> 对于极坐标机械手的示教记录法编程的应用中，只有记录轨迹（X，Y，Z）的 点坐标，而不是记录在每个拐点处的各轴的位置，这样在机械手运行时，采用轨 迹点插补+坐标变换，才能得到所示教的线段轨迹。

6.6.3 SCARA运动机构的坐标变换

用户轨迹程序都是按笛卡尔垂直坐标描述，轨迹插补的输出中间点的坐标也是按笛卡尔坐标系描述，SCARA型机械手手掌的运动，是由2关节或3关节平面极坐标旋转，加上Z轴的上下运动方式来完成，因此需要对其中的平面极坐标变换的计算。

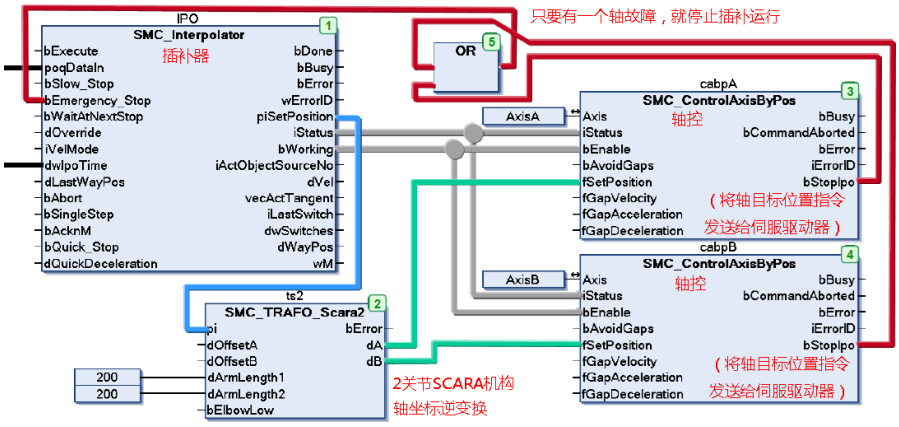
两关节SCARA 机械手结构的几何图如下：



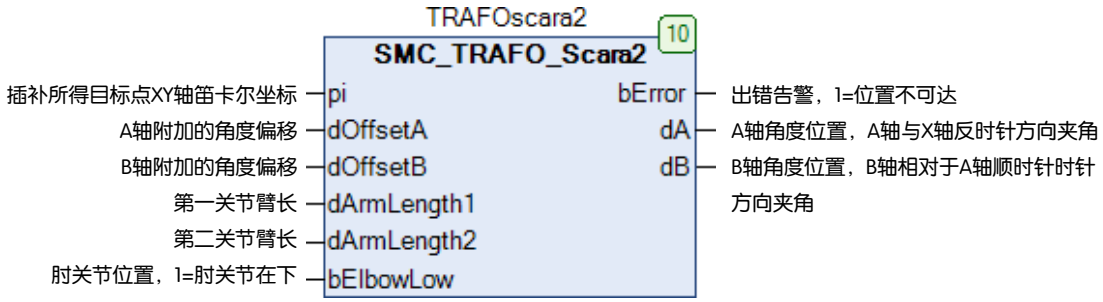
McEngine Pro中提供了笛卡尔坐标对2关节SCARA 的坐标变换功能块：

SMC_TRAFO_Scara2	由曲线坐标计算轴位置（逆变换）
SMC_TRAFOF_Scara2	由轴位置计算曲线坐标（正变换）

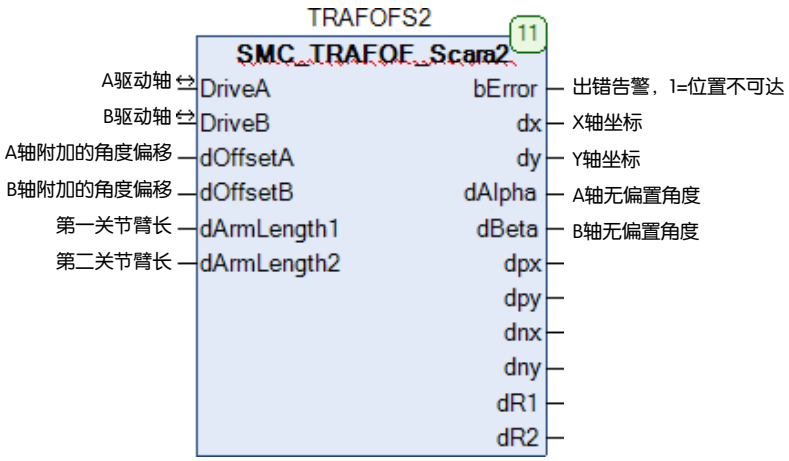
典型的2关节SCARA机械手控制程序如下图：



逆变换功能块的输入输出变量定义如下：

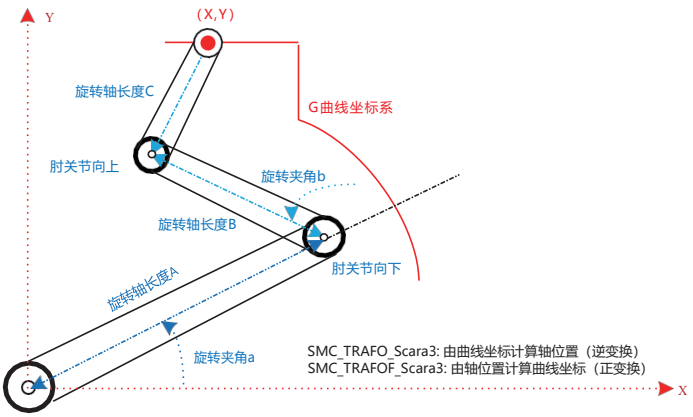


二关节SCARA系统的坐标正向变换功能块指令如下，由于两个旋转臂的长度直接影响SCARA 机械手手掌的运动范围，因此该功能块相比其他功能块多了一个位置是否可达的bError布尔型信号输出，后续的轴控功能块对该告警信号应有判断处理，防止有误动作。正向变换功能块的输入输出变量如下：



注意：
若设备操作者需要临时的参考坐标修改，只需要修改相关的偏移值即可。

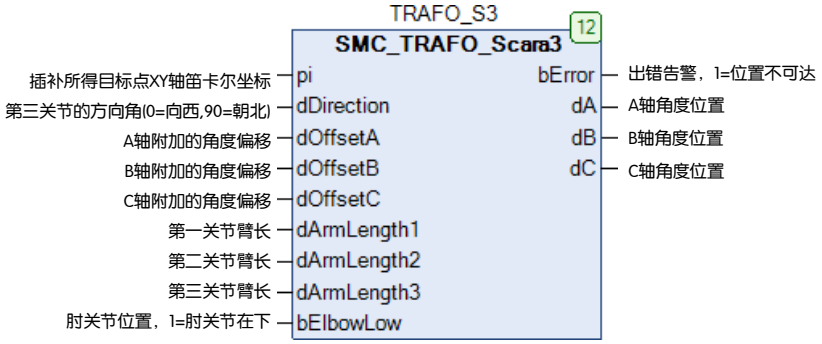
三关节SCARA机械手结构的几何图如下，该结构的优点是最后一个关节可以按指定方向接近目标点，实际运动应用中，方便避开障碍，活动范围更大：



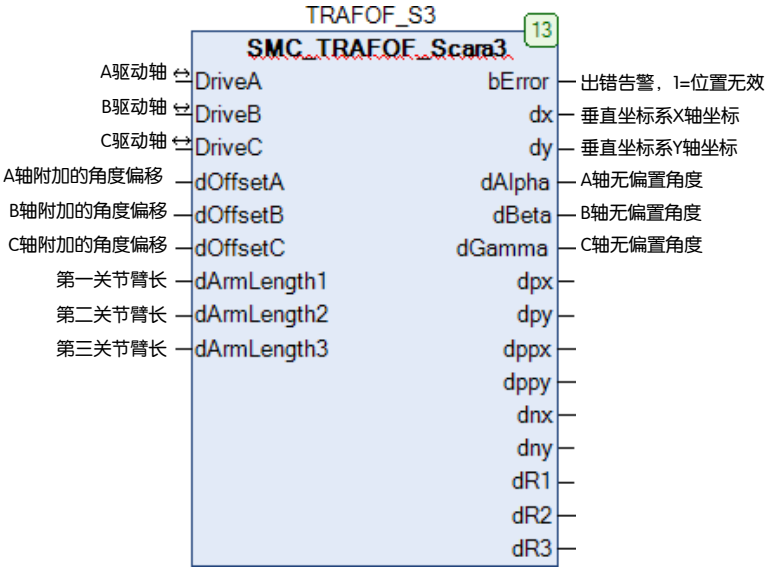
McEngine Pro中提供了笛卡尔坐标对3关节 SCARA 的坐标变换功能块：

SMC_TRAFO_Scara3	由曲线坐标计算轴位置（逆变换）
SMC_TRAFOF_Scara3	由轴位置计算曲线坐标（正变换）

逆变换功能块的输入输出变量如下：



正向变换功能块的输入输出变量如下：

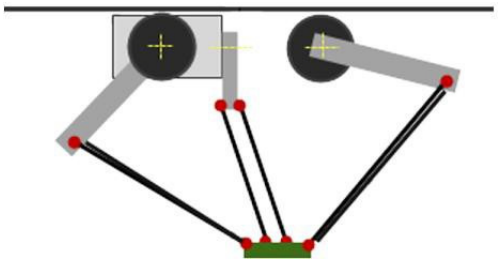


6.6.4 DELTA运动机构的坐标变换

Delta型结构的机械手，按旋转轴的数量可以有2轴型和3轴型结构，因其重量轻，动作速度快，分别适宜在码垛、装箱、分拣类设备中应用。

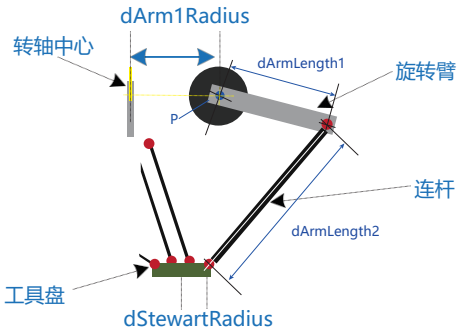
如下为3轴型Delta机构，其三轴机械结构对称分布，即三个手臂的长度相等、三个连接杆的长度相等、相对位置为均匀分布，三个轴的下臂都采用的是两根平行四边形结构，因此下方的小平板无论运行到什么位置，总是呈水平姿态，固定于该小平板的抓手也就总是向下或某固定方向。3轴型Delta机构的上肢有旋转和线型移动两种，通过3个上肢的不同运动位置组合，下方工具小板在一定空间区域内自由快速运动。

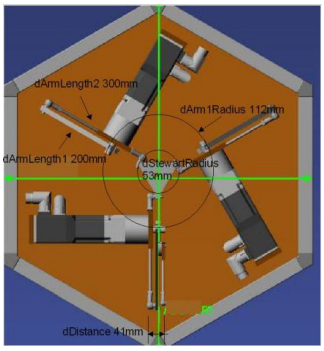
3 轴旋转DELTA机构的结构如下：



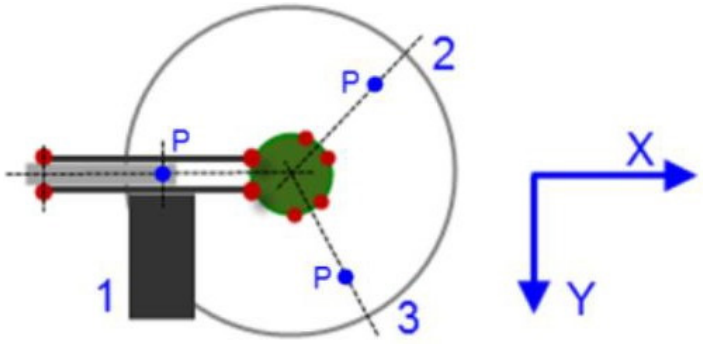
3轴旋转DELTA

对于旋转轴 DELTA 机构，需要根据结构，确定如下几个长度尺寸参数，右侧图为一种结构尺寸举例：





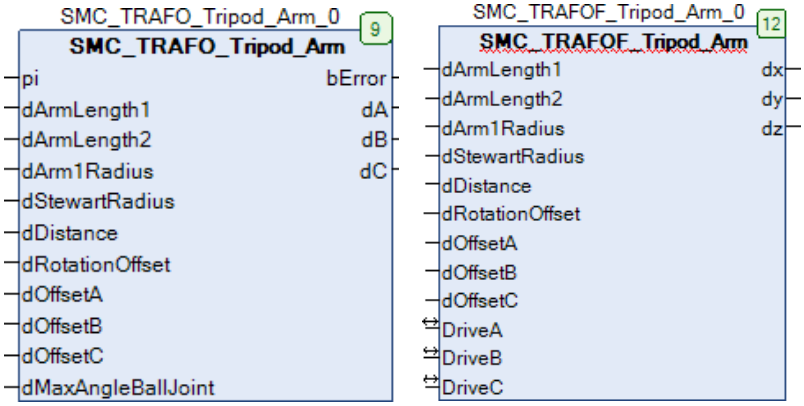
下图为工具盘中心的XY坐标o位置定义：



采用变换的功能块为：

SMC_TRAFO_Tripod_Arm	3 轴旋转 Delta 机构，由下盘中心坐标计算轴位置（逆变换）
SMC_TRAFOF_ Tripod_Arm	3轴旋转Delta机构，由轴位置计算下盘中心坐标（正变换）

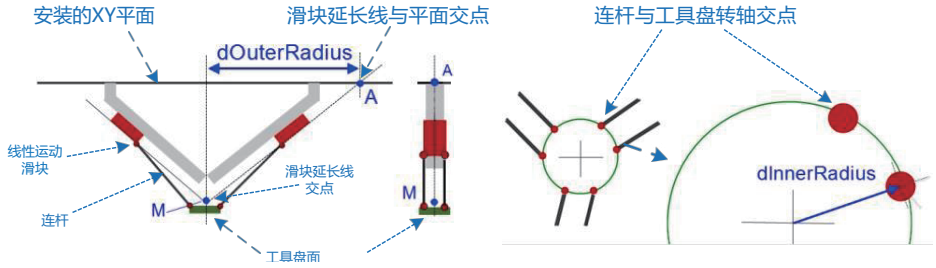
逆变换及正变换功能块如下：



3 轴线型 DELTA 机构的结构如下：

3轴线性DELTA

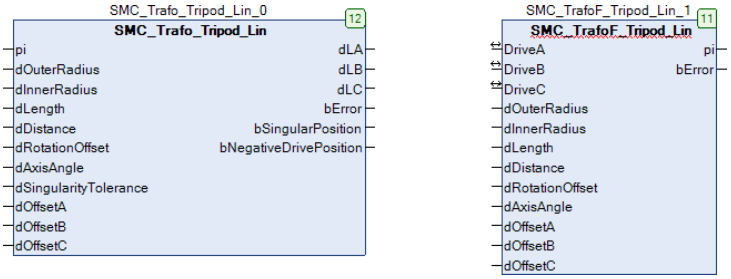
根据其结构，确定如下几个长度尺寸参数：



采用变换的功能块为：

SMC_TRAFO_Tripod_Lin	3轴线型 Delta 机构，由下盘中心坐标计算轴位置（逆变换）
SMC_TRAFOF_ Tripod_Lin	3轴线型Delta机构，由轴位置计算下盘中心坐标（正变换）

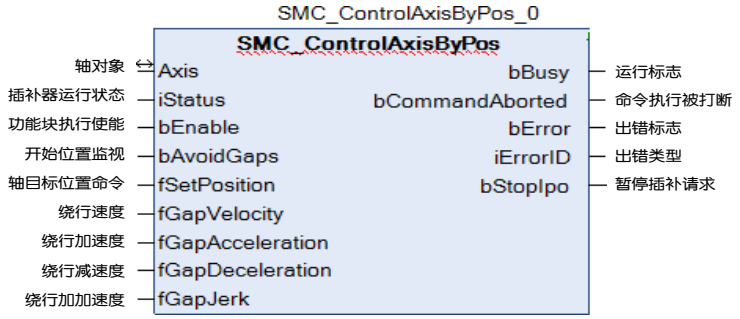
逆变换及正变换功能块如下：



当主轴为相对位置模式，在进入CAM时，凸轮运算模块会以当前位置作为主轴的起点 X=0，进行Codesys 编程软件中，提供了几种典型的CNC插补控制的例程，可供借鉴引用，用户应用编程时，主要的工作量是输入所要求的运动轨迹曲线，即CNC轨迹，即可进行离线仿真实验。

6.7 驱动轴控制功能块

当经过坐标逆变换后，就得到了每个参与轨迹运动各轴的目标位置，需要使用轴控功能块SMC_ControlAxisByPos，将位置指令（fSetPosition）写入到对应轴的数据结构中，系统自动通过的EtherCAT总线PDO命令数据发送缓存区，在接下来的 EtherCAT 周期，发送给对应的伺服从站，实现轴运动的控制。功能块如下图：



也就是说，虽然看不到该功能块的输出数据，但它是将轴的“目标位置”命令数据写入 总线命令发送缓存，由总线命令去控制对应轴的运动，完成了命令输出。

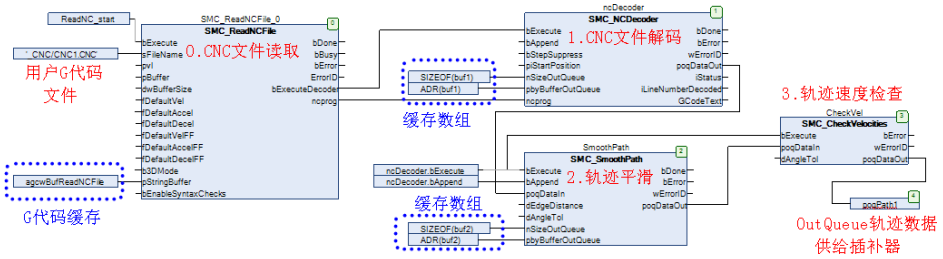
当bAvoidGaps=1时，该功能块会检查轴的位置数据命令，判断运行速度是否会超过设定的限制值，若有超出，会置位 bStopIpo=1输出标志变量，该变量应该被送给前方的插补器暂停插补命令 bEmergency_Stop 输入端，令插补器暂停一段时间，驱动轴按照设定的最大速度赶上插补器的命令位置后，复位 bStopIpo=0，让插补器恢复插补运行。

编程时需注意：

fGapDeceleration和fGapJerk输入应该始终设置为合理的值，即使在 bAvoidGaps=0 时也是如此。当bEnable输入设置为 0 时，也会使用这两个输入变量的值。

6.8 CNC轨迹文件的解码编译POU程序

对于按 File 文本文件 G-Code 轨迹进行控制的应用，控制器需对 G-Code 程序进行读取、解码及预处理，其典型程序如下图，下图程序的执行可能花比较长的时间，不能在 EtherCAT 任务中执行，而需放在普通任务中执行：



以便在执行大容量的G-Code轨迹程序时，让普通任务中的解码、预处理与EtherCAT任务的轨迹插补等环节可以流水线方式“同时”进行，而不是待前一环节全部处理完成后，才开始后一功能块环节的处理，这样可以即时开始一个轨迹的执行，省去了长时间的等待；另外也方便插补器的轨迹预判断，防止OutQueue数据队列中途被用光而导致轨迹运行停顿的情况。对于以点到点（PP）轨迹控制为主的应用，缓存区容量（sizeof（buf））可以设置为50~100；对于高密度轨迹数据的应用，如金属加工的轨迹控制，缓存区容量（sizeof（buf））可以设置为 500~1000。

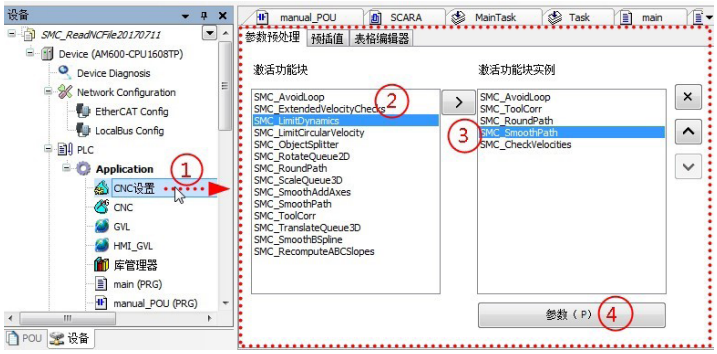
有些功能块需要有其专用缓存数据区，编程时需要注意声明。编程时，需注意每个功能块bExecute命令触发源及其时序，可使得运行响应更顺畅。

上图所示的File读取、NC 文件的解码、轨迹预处理、轨迹速度检查处理程序，是典型的POU，类似应用的编程时我们可以直接借用。

控制器系统提供了一些预处理功能块，可能使得控制器功能变得强大，如刀具补偿，也 可以让系统运行运行更稳定，如轨迹圆滑、速度的检查与限制等。

6.9 轨迹预处理功能块

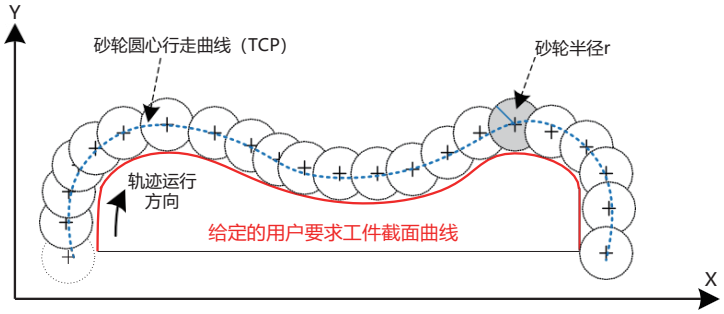
在McEngine Pro编程环境中编写CNC程序时，往往需要在没有完善的插补程序的情况下，观察轨迹执行的仿真效果，于是可以采用如下图所示的简单配置来选择预处理功能块、修改其设定参数，当调试满意后，再将这些设置固化为用户程序，下载到控制器中：图5-9非周期模式运行上图红色虚线框②中列出的是控制器可以选用的各种预处理功能块，光标选中所需的功能块后，点击③按钮，在右侧窗口会增加该项目，系统就等效声明了该预处理功能块的实例，高亮选择右侧窗口内的项目后，点击④按钮就可以在弹出的窗口内设置功能块所需的参数。这种配置方法，方便参数的修改，直到调试满意为止，可大为简化调试过程。



毕竟多数的终端用户应用中，是不需使用McEngine Pro编程的，因此需通过插补程序中使用预处理功能块来进行处理。本节讲解几个常用的预处理功能块，未提到的参考 McEngine Pr中的Help帮助不难理解。

6.9.1 刀具补偿功能及功能块SMC_ToolCorr

eXtreme、Agile控制器的刀具补偿功能，是一种实用的轨迹补偿处理，在磨床、镗床设备的轨迹控制中，可以大大简化用户的轨迹编程，如下图所示为磨床加工异形表面的情况，红色的轨迹由CAD软件设计，为用户希望得到的工件轮廓外形。考虑砂轮的半径后，砂轮轴的行走轨迹就应该如虚线所示了，这个偏移处理就称为刀具补偿。



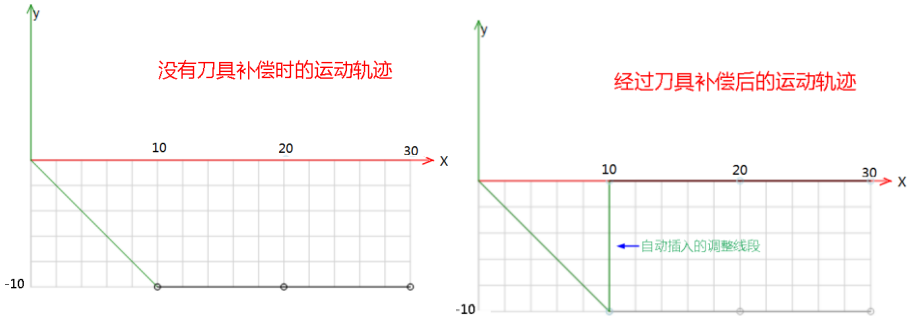
使用控制器的刀具补偿功能，用户只需输入所需的工件表面形状曲线，圆形工具的半径r，按照轨迹运行的方向，选择“右刀补”，控制器就会自动按图中补偿后的 TCP 曲线行走了， 而且砂轮与工件表面接触处的行走速度仍按指定恒速速度运动，这个功能在磨床、镗床等需 要考虑刀具尺寸的应用控制中非常有用。

1、采用G代码使用刀具补偿功能的应用

例一：用户编写 G-Code 程序，可以使用刀补功能、设置刀补参数，控制器在执行 G-Code轨迹程序时，自动启用刀补功能，编程语句如下：

```
N00 G0 X10 Y-10 F10 E100 E-100
N10 G41 D10          //开启刀具左补偿功能，刀具半径为10 N20
G1 X20 Y-10
N30 G1 X30 Y-10
```

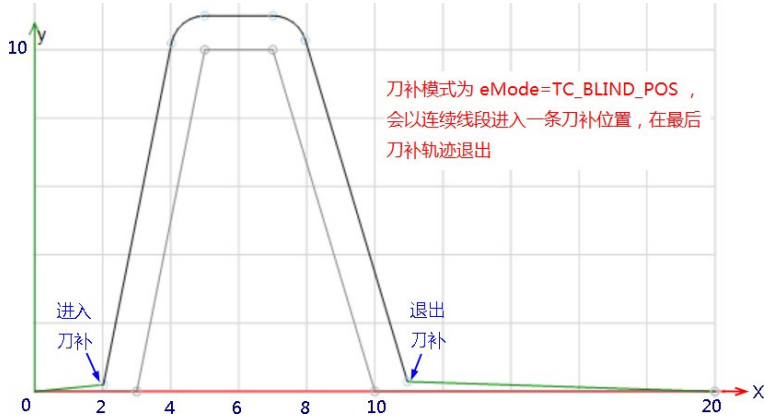
具有刀具补偿的运行轨迹比较如下，开始按刀具补偿时，会自动插入一个定位调整的线段，让刀具先到达补偿偏移位置，以便得到希望的刀补行走轨迹：



例二：在刀具补偿模式常采用 eMode=TC_BLIND_POS，（通过 SMC_TOOLCORR功能块的输入变量 eMode 设为 0 来选择），插入的是定位线段，在待插补的轨迹前端和后端，分别有G00 定位指令的情况下，补偿起止的定位调整插入的是一个连续调整的线段，如下程序举例：

```
N000 G41 D1          //开始左刀补偿功能，刀具半径为1
N010 G00  X3 F100 E1000 E-1000          //原有的定位语句
N020 G01 X5 Y10
N030 G01 X7 Y10
N040 G01 X10 Y0
N050 G00 X20          //原有的定位语句
N060 G40          //关闭刀补功能
```

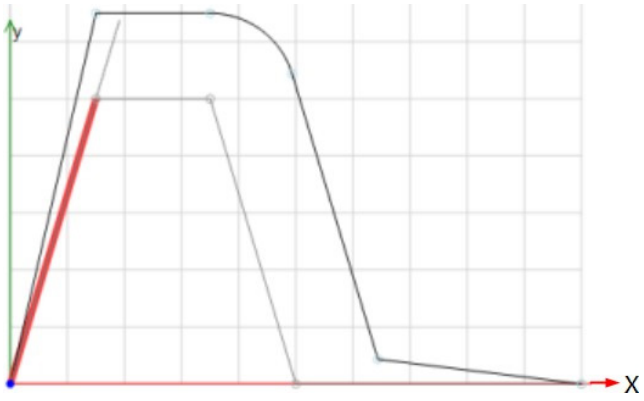
运行的刀具中心（TCP）轨迹如下：



在刀具补偿模式若采用 eMode=TC_BLIND_IN，（通过 SMC_TOOLCORR 功能块的输入变量 eMode设为1选择），起始点将没有刀补偏移，而是在第一个轨迹对象的后端才调整到刀补位，如下程序举例：

```
N000 G41 D3        //开始左刀补偿功能，刀具半径为3
N010 G01 X3 Y10 F100 E1000 E-1000
N020 G01 X7 Y10 N030 G01 X10 Y0
N040 G40           //关闭刀补功能
N050 G01 X20
```

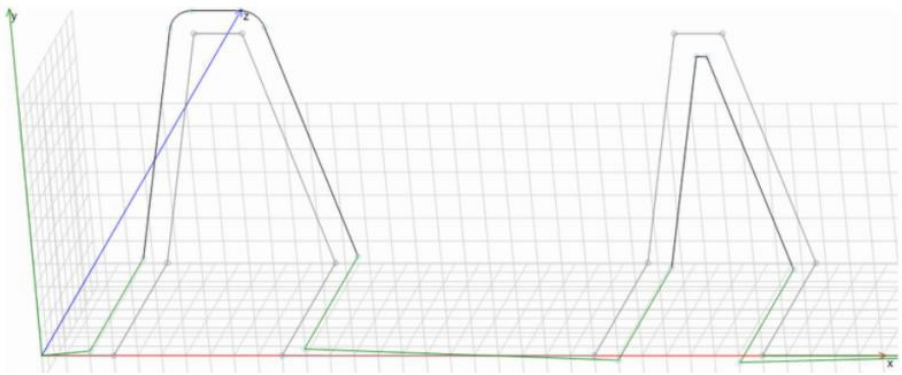
其执行的轨迹如下图，从中可以看到第一条轨迹的起点没有补偿，直到其终点刀补偏移 才调整完成，编程时要注意差别：



例三：一个钻床/锣床加工的控制程序举例，其中对 XYZ 轴的都有补偿效果：

```
//第一部分
N000 G41 D1 F100 E1000 E-1000      //开始左刀补偿功能，刀具半径为 1
N010 G00 X3
N020 G01 Z10
N030 G01 X5 Y10                    //锣工件外形开始
N040 G01 X7 Y10
N050 G01  X10 Y0                   //锣工件外形结束
N060 G01 Z0
N070 G40                           //左刀补偿结束
N080 G00 X20
    //第二部分
N080 G42 D1                        //开始右刀补偿功能，刀具半径为1
N090 G00 X23
N100 G01 Z10
N110 G01 X25 Y10
N120 G01 X27 Y10
N130 G01 X30 Y0
N140 G01 Z0
N150 G40                           //右刀补偿结束
N160 G00 X40
```

执行的轨迹如下图：



上图的例子显示包含了 Z 方向上定位的刀具补偿的应用，补偿模式设置为 BLIND_POS。在（G41-G40）/（G42-G40）代码区间的第一个语句是定位于轮廓的起点，接着就是在Z轴定位就可以看到刀具补偿的偏移，然后轮廓轨迹开始；当Z轴定位离开工件后，刀具半径补偿关闭。示例的第二部分显示了另一个方向的工具半径补偿效果。

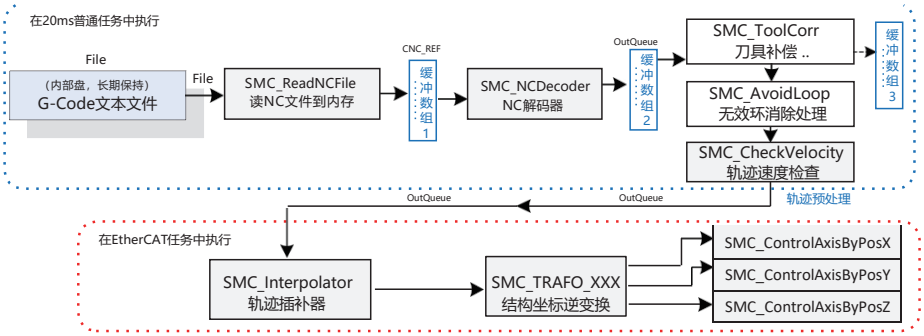
编程提示：

1）对于需要精确刀具补偿的加工轨迹，最好在其前端、后端分别有 G00 定位指令，这样加工不会出现意外TCP轨迹的情况；

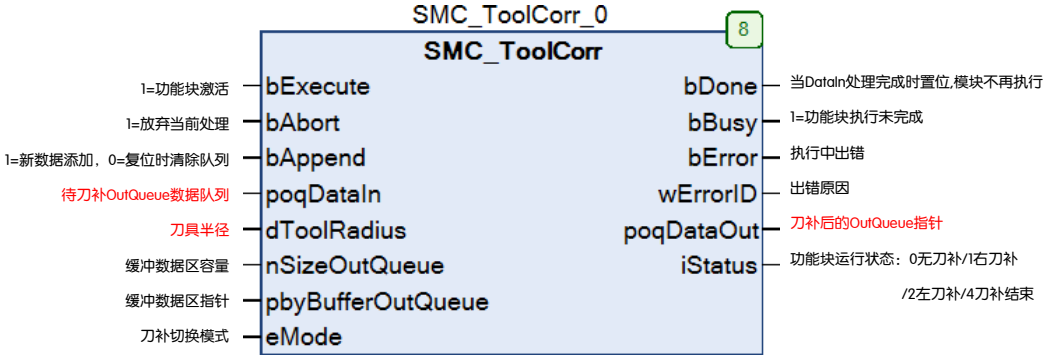
2）刀具补偿起止指令（G41-G40）或（G42-G40）要配对使用；在 G-Code 轨迹中可以出现多次，但不要有相互交叉、嵌套使用的情况；

2、采用功能块 SMC_ToolCorr 进行刀具补偿预处理

前面介绍的是通过G-Code程序调用刀补功能的方法，实际应用中，人们使用更多的采用CAD设计文件描述所需加工得到的工件外形，并由此生成控制轨迹，生产操作人员需根据加工机床所使用的钻头或磨头尺寸，现场确定刀具补偿尺寸，这样就不便在 G-Code 文件中插入刀补指令了，解决这个问题的方法就是在解码 G-Code轨迹程序，调用功能块SMC_ToolCorr，最后生成插补用的OutQueue，如下图：



其中刀补功能块定义如下：

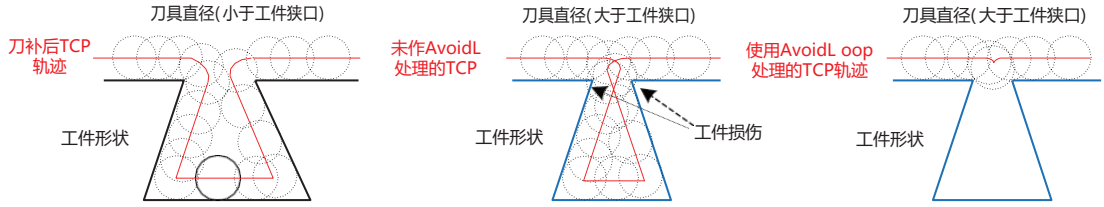


从SMC_ToolCorr功能块的变量中可以看出：

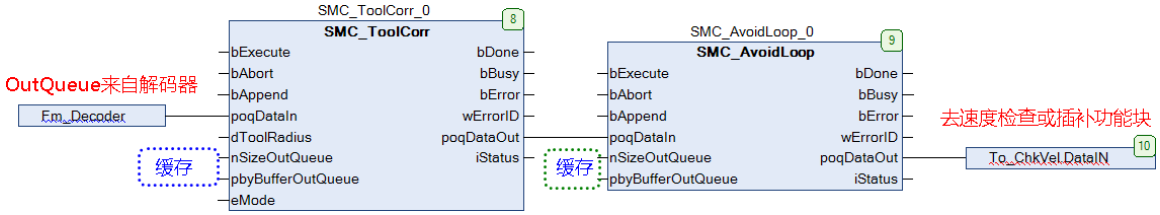
bExuecute：即使程序中使用了刀补功能块，在运行时可以使之运行生效，或禁止其运行；当禁止其运行时，其输出的OutQueue为输入侧的相同轨迹；poqDataIn：指向待插补的OutQueue队列，一般与前一级功能块的OutQueue输出端连接；dToolRadius：设定的刀具半径 ，可以在线修改；pbyBufferOutQueue：该功能块运行时需要有一个专用的数据缓存区，便于进行TCP轨迹计算时，插入必要的调整线段、拐角圆弧段等，一般定义【 50~100 】的数组。

6.9.2 无效环检查功能块SMC_AvoidLoop

该功能块一般跟随刀补功能块 SMC_ToolCorr 使用，SMC_AvoidLoop 用于检查刀补后的轨迹是否会出现工件损伤，若刀补后的轨迹有环路交叉，意味着会出现加工工件的损伤，因此需消除刀具中心轨迹（TCP）中的环路，以避免因刀具尺寸超大导致出现加工废料的情况，如下图所示：

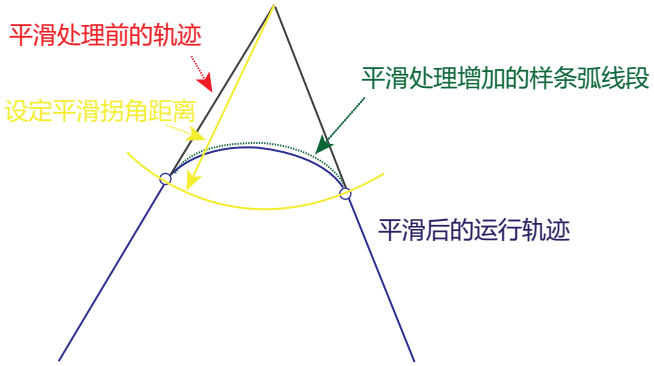


SMC_AvoidLoop功能块的使用与刀补功能块相似，也需要定义一个专用的数据缓冲区，连接关系如下：



6.9.3 轨迹平滑功能块SMC_SmoothPath

在一些点到点的运动控制应用中，对起点、终点的定位精度要求高，希望运动完成的时 间短，但对轨迹的精确度要求并不高，对于这样的应用需求，通过轨迹平滑处理，将轨迹运行速度可能降为零的拐角进行圆滑处理，预处理功能块 SMC_SmoothPath就可实现该功能。平滑处理时，按照编程者设定的拐角条件，插入一段样条弧线，运行时就不会出现速度突变 的情况了，如下图：



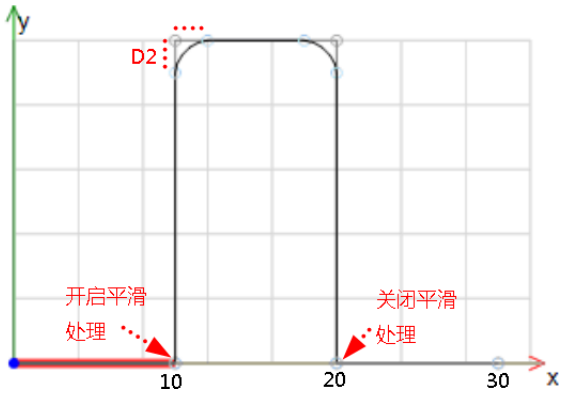
图中插入的样条曲线的起止位置由用户设定的平滑拐角距离决定，以兼顾平滑效果与轨 迹偏差。

插入样条曲线，控制器系统会根据前端、后端的轨迹情况，自动优化计算插入的样条轨 迹，使得合成轨迹的结合点位置连续、速度连续、加加速度连续，合理利用这个特点，可使 得运动机构运行平稳。

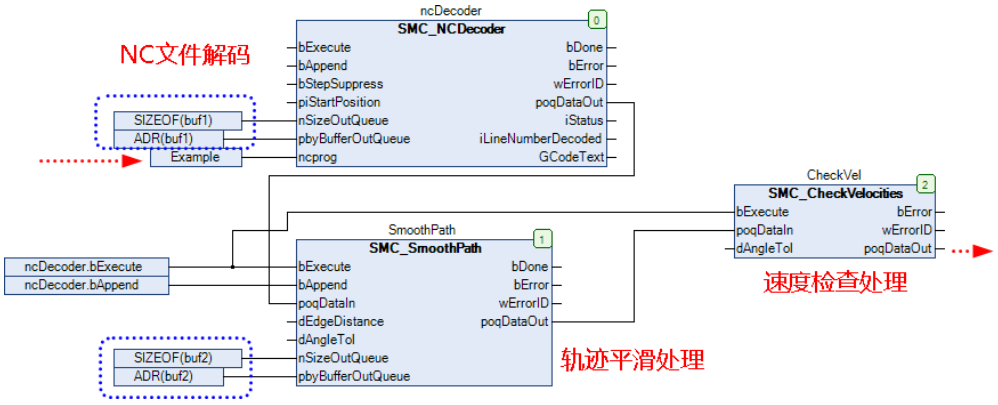
平滑处理功能可以在 G-Code 程序中启用（G51）和关闭（G50），并设定平滑参数，举例如下：

```
N000 G01 X10 Y0 F100
N010 G51 D2        //开启平滑处理，设角距为2
N020 G01 X10 Y20 N030 G01 X20 Y20 N040 G01 X20 Y0
N050 G50    //关闭平滑处理
N060 G01 X30
```

运行的轨迹如下：



与其他预处理功能的使用相似，平滑处理功能也可以在插补程序中，通过调用SMC_SmoothPath功能块来实现，典型的平滑处理程序如下：

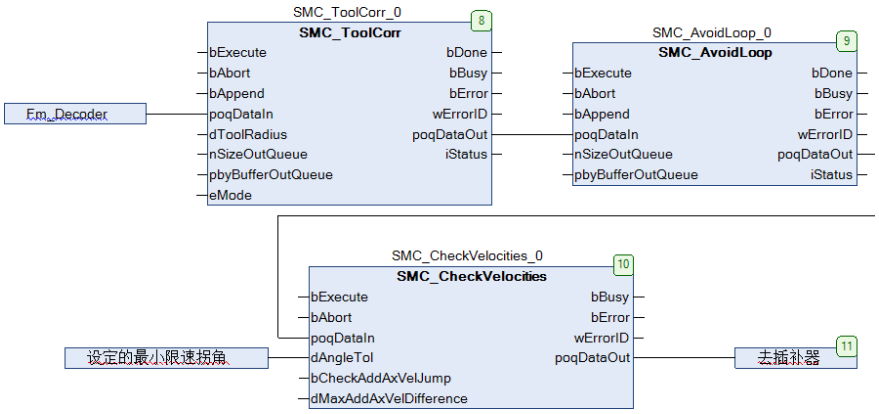


用户程序中，通过对功能块的 bExecute 变量的控制，可以开启或关闭平滑功能。

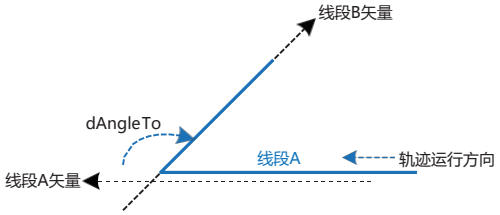
使用时需要特别注意，若用户程序中有刀具补偿、无效环消除、轨迹平滑等功能块，那么，轨迹平滑功能块应该放在刀具补偿、无效环消除等功能块的后面，否则若顺序颠倒，会 导致补偿功能失效。

6.9.4 轨迹速度检查功能块SMC_CheckVelocities

这是最常用的预处理功能块，用于检查特定路径段的轨道速度是否超限，并进行速度连续性的优化调整，以提高轨迹运行的效率。如果用户的轨迹程序不是由McEngine Pro编辑器创建，而是由IEC程序(例如由轨迹示教的拐点数组或G-Code文件+SMC_NCDecoder)创建，则需调用该功能块。



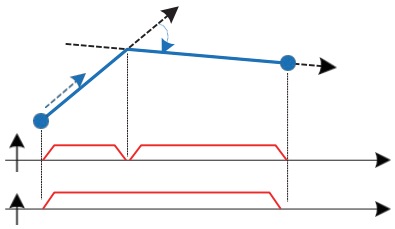
该功能块往往是在插补器之前的最后一个预处理功能块本功能块的一个设置参数dAngleTol是圆滑允许角度，可以设定速度检测功能在两线段矢量夹角的最小生效范围，即当两线段矢量的夹角大于该设定值时才生效，如下图



注意，dAngleTol 设定角不是轨迹的夹角，而是矢量的夹角，单位为度（Lreal型，允许小数表示），而线段的矢量方向是由 G-Code 程序中线段的起始终点的坐标定义来确定的。

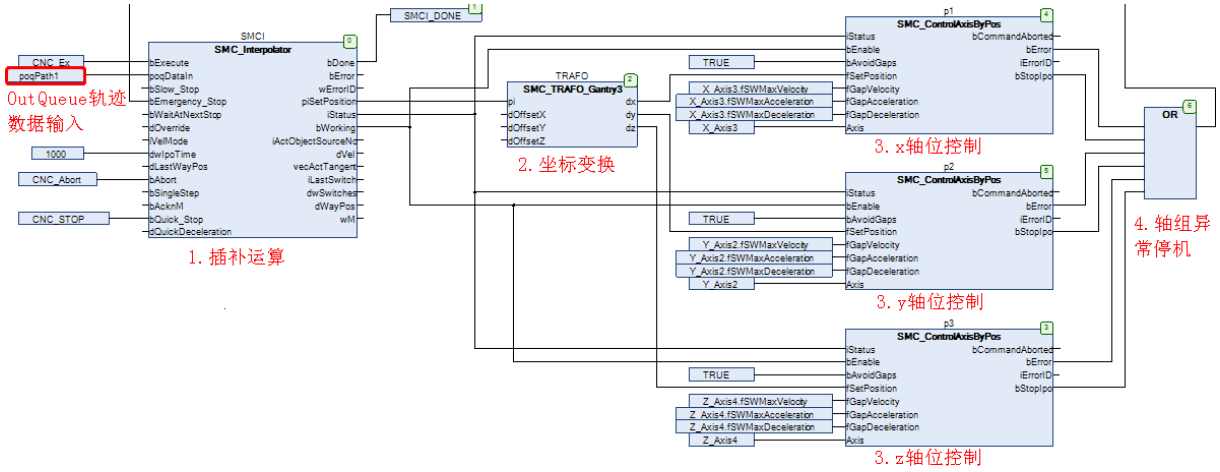
当轨迹线段的矢量夹角小于dAngleTol的设定值时，如上图轨迹有锐角，水平方向的驱动轴会有折返运动，势必会有速度的突变，这时SMC_CheckVelocities处理后，会在该轨迹线段A锐角处有减速到0，然后再沿线段 B 反向加速运动。

当轨迹线段的矢量夹角大于dAngleTol的设定值时，如下图轨迹所示，轨迹在拐角处以连续速度运行，到达终点的时间更短：



第七章.典型的CNC轨迹插补POU程序

正如前面所述，G-Code是以三维垂直坐标系来描述空间位置和轨迹，下图是简单的 3轴垂直坐标系统（也称龙门架结构）的插补执行程序，以 CFC 图形化语言给出，用户的轨迹数据已编译成 SMC_OutQueue 格式，则最简化的插补控制程序如下图：



该程序必须将在EtherCAT任务中执行，以确保每个EtherCAT周期中，该程序能被扫描执行一次。

第一次执行该程序，将OutQueue的第一条轨迹起点开始插补，得到线段中的起点位置点坐标（PiSetPosition），该坐标值作为XYZ插补轴的下一个目标点；该目标点坐标经过坐标变换后，即可送给轴位控制功能块，送给EtherCAT总线控制器发送缓存，下发给伺服轴从站。以后每执行一次，插补得到的目标点依次向OutQueue轨迹线段的终点移动，依次类推，直到OutQueue定义的所有线段都被插补、运动完成。

在Codesys安装目录下，随机附带多个CNC例程，用户可以参照例程，学习CNC的编程，进一步扩充CNC的编程应用。

C:\Program Files\CODESYS\CODESYS\Projects\SoftMotion\4.10.0.0\Examples\Tutorial

CNC Example 1: Generating OutQueue Directly

CNC Example 2: Online Decoding with Variables

CNC Example 3: Performing Path Preprocessing Online

CNC Example 4: Programming CNC using the Table Editor

CNC Example 5: Creating CNC from a File

CNC Example 6: Using Path3D with SoftMotion CNC

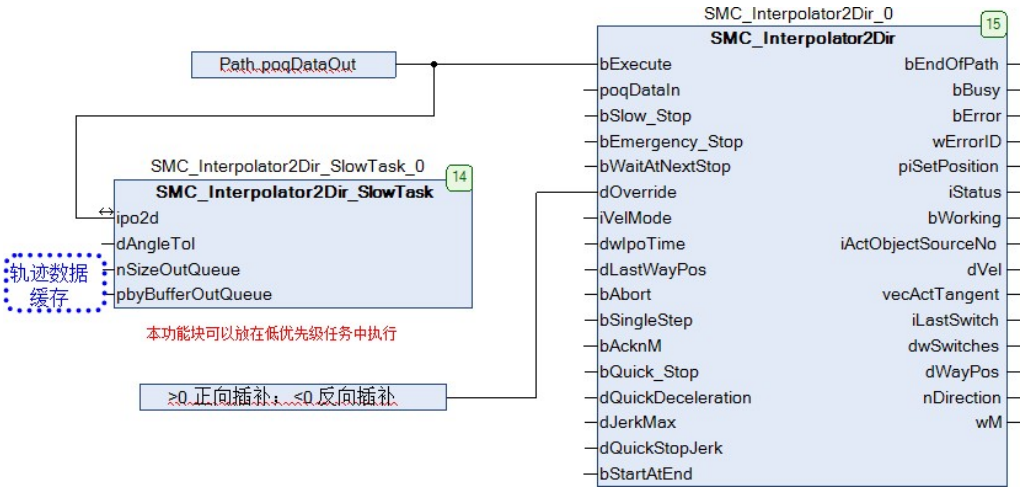
CNC Example 7: Using Expressions and Subprograms

第八章.功能更灵活的轨迹插补功能块

eXtreme、Agile除了前面介绍的 SMC_Interpolater 功能块以外，还有可双向轨迹插补、可以其中X轴的位置作为插补参考的功能块，在一些运动控制设备中，可以实现更灵活的控制。

8.1 具有双向插补功能的SMC_Interpolator2Dir功能块

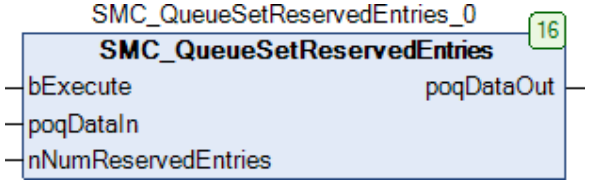
本功能块与 SMC_Interpolator 的功能和用法相同，在控制程序中可替换 SMC_Interpolator， 但具有正向和反向轨迹插补的功能，将输入变量端口 dOverride 设为负值，就可以让插补器进行反向轨迹的插补动作：



因为要支持随时可能的反向运动，需要将需要插补用的 OutQueue 数据存放在一个比较合适的缓冲区中，可以通过调用与之配套使用的 SMC_Interpolator2Dir_SlowTask 功能块，因为这个功能块功能简单，可以放在普通任务中执行。

还有一个配套使用的功能块 SMC_QueueSetReservedEntries 用于设定和限制 OutQueue

缓存的大小，当倒退到允许的数值后，系统就会报“缓存空”的报警。



使用时注意：

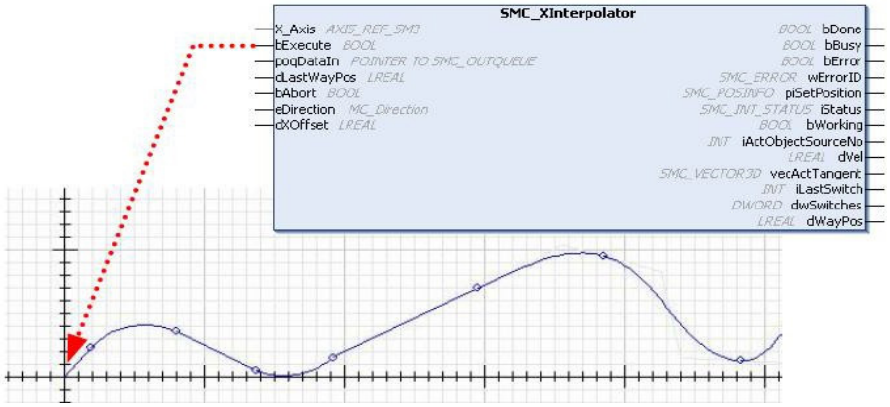
1.为SMC_Interpolator2Dir_SlowTask功能块声明专用的缓存区的大小，如果只是倒退 n 个线段或弧段，缓存的大小可以声明为（n+3），宜声明得更大一些；若希望全轨迹都可以倒退的话，其缓存将是很大的。

2.本功能块不能用于SMC_CheckVelocities或SMC_Checkforlimits 功能块的后面，因为这些功能块生成的插入数据，不会被复制到缓存区中。

8.2 将X轴为插补参考的SMC_XInterpolator功能块

标准功能块SMC_Interpolator是基于时间参考的插补器，即插补器按照用户设定的轨迹运行速度、加速度及S曲线加速减速过程后，对G-Code轨迹进行插补后，输出XYZ插补轴的目标位置。

而功能块SMC_XInterpolator则是基于参考X轴的当前位置，按照G-Code轨迹进行插补后，输出YZ插补轴的目标位置，实现CAM和CNC的混合，而且是1主2从的CAM，可以实现更灵活的轨迹控制应用。例如要从工件中切割出一个指定的形状(G代码描述)，通过参考外部轴的运动(例如沿X轴)，来控制其他轴(Y，Z)移动工件，利用SMC_XInterpolator插补器按工件(X)的当前位置和路径轮廓所给出的目标轨迹进行控制。



在希望开始插补运行的X轴坐标点处，触发SMC_XInterpolator功能块的bExecute输入命令端口的上升沿并保持，即可令多个轴开始插补运行，就如凸轮运行一样，其他几个轴的位置以X的位置作为参考、以G-Code的轨迹作为解算依据，直到所有的轨迹数据执行完毕。本模块的触发执行特点与常用的MC运控功能块相同。

8.3 提高轨迹控制精度的方法

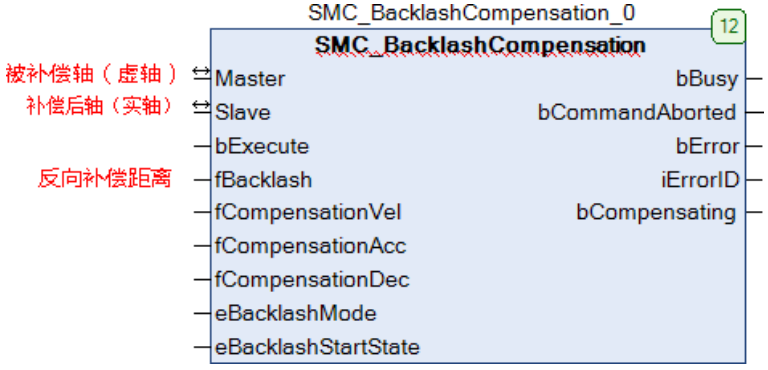
在有些轨迹控制中，运行的轨迹精度直接决定了产品的质量，如金属切割、磨、雕刻等 加工设备中，对行走的轨迹，与用户轨迹程序的描述，误差需要小于指标要求。

产生误差的原因有机械结构、驱动执行、传感检测、控制软件、系统离散延迟等多方面 的原因，分析归纳为机构误差、驱动误差、系统固有延迟等三方面。对于一个应用系统的误 差，我们需分析其主要原因，针对性的采取消除方法，才能达到预期的效果。

8.3.1 机构误差的改善消除

机械结构产生的误差，例如丝杆的导程不均匀、传动齿轮的不均匀，皮带受载变形等， 理论上难以用软件的方式消除，只能通过选用合适硬件材料选择来改善。

但有些误差是可以改善的，如齿轮传动、皮带传动的正反向旷动间隙，若能准确测量， 则可以通过使用“反向间隙补偿”功能块 SMC_BacklashCompensation 来改善，功能块如下图：



使用方法是将待补偿轴声明为虚轴（作为补偿的主轴），补偿处理后的轴为实轴（作为补偿从轴），还需设定一旦主轴反向运行时，需要补偿的位置、补偿期间的速度、加速度、减速度、补偿模式等，运行中，模块会自动根据主轴的速度方向，开始补偿，用于位置控制 命令的输出之前即可。

8.3.2 驱动误差的改善消除

运动控制机构往往有一定的惯量，虽然是伺服驱动，但机构也会有其加减速时间，在负 载比较大的情况下，其加减速时间更大，当插补器设定的轨迹速度比较大时，就会出现轴跟 不上命令的情况。消除这种误差的方法是，比较轴的当前位置 fActPosition 与控制器前 2 个周期发送的位置指令 fSetPosition 进行比较，若偏差大于允许的设定值，置暂停插补运行请求，将 SSMC_Interpolator 的 bEmergency_Stop 输入变量置 1，直到偏差小于设定值后清除。这样就可以对整个轨迹运行过程中，都实现位置闭环，避免了轨迹误差。

这样处理后，可能出现轨迹运行速度降低的情况，轨迹运行速度与误差是互为因果的关 系，这可以在设置允许偏差时综合考虑。

8.3.3 系统离散误差的改善消除

基于EtherCAT总线的运动控制，都有一个因为EtherCAT时针周期离散化时产生的一个误差，例如对于轨迹速度为每秒钟运行2米的应用，若EtherCAT 周期设为2ms，那么每个EtherCAT周期的数值化间隔是 4mm，若只是根据伺服的当前位置判断，其对齐误差最大可能是 4mm，这在许多应用中不满足精度要求，因此，对于起始对齐点，应检测初始偏差（Offset），在其轨迹定位的目标点时，要将初始偏差计算在内，避免离散误差。

对于色标检测的高速应用中，通过色标信号触发伺服的探针功能，控制器读取探针记录 数据，是一个准确检测初始偏差的好方法，许多定位控制功能块中，都有 Offset 变量输入端。

客户咨询电话

400-820-9661

更多安浦鸣志资讯，请扫码关注！

公众号　　微官网

鸣志总部

上海市闵行区闵北路88弄7号楼
邮编：201107

鸣志电器（太仓）有限公司

江苏省太仓市港区银港路16、18号
邮编：215434

国内办事处

北京
北京市朝阳区东三环中路16号京粮大厦1206室
邮编：100022

青岛
山东省青岛市市北区山东路171号科技创新大厦1号楼19楼1913室
邮编：266033

西安
陕西省西安市唐延路1号旺座国际城D座1006室
邮编：710065

武汉
湖北省武汉市江汉区解放大道686号世贸大厦3001室
邮编：430022

合肥
安徽省合肥市蜀山区井岗路CBC拓基广场B座1521室
邮编：230088

南京
江苏省南京市江宁区天元中路126号新城发展中心2号楼11楼1101/1102室
邮编：211106

苏州
江苏省苏州市姑苏区南环东路758号汇邻广场4号北楼1103-1105室
邮编：215007

宁波
浙江省宁波市江东区惊驾路565号泰富广场B座309室
邮编：315040

成都
四川省成都市锦江区东御街19号茂业天地3907室
邮编：610066

重庆
重庆市江北区福泉路18号源著南区20栋2108室
邮编：400000

广州
广东省广州市天河区林和西路9号耀中广场B座40层06室
邮编：510610

东莞
广东省东莞市松山湖研发五路1号林润智谷5号楼1206-1207室
邮编：523000

深圳
广东省深圳市南山区留仙大道4168号众冠时代广场A座3901室
邮编：518000

北美地区

美国

MOONS' INDUSTRIES (AMERICA), INC. (Chicago)
1113 North Prospect Avenue, Itasca, IL 60143, USA

MOONS' INDUSTRIES (AMERICA), INC. (Boston)
36 Cordage Park Circle, Suite 310 Plymouth, MA 02360, USA

APPLIED MOTION PRODUCTS, INC. (Morgan Hill)
18645 Madrone Parkway. Morgan Hill, CA 95037, USA

LIN ENGINEERING, INC. (Morgan Hill)
16245 Vineyard Blvd., Morgan Hill, CA 95037, USA

欧洲地区

德国

AMP & MOONS' AUTOMATION(GERMANY)GMBH
Kaiserhofstr. 15
60313 Frankfurt am Main Germany

意大利

MOONS' INDUSTRIES (EUROPE) HEAD QUARTER S.R.L.
Via Torri Bianche n.1 20871 Vimercate(MB) Italy

瑞士

TECHNOSOFT (SUISSE) SA
Avenue des Alpes 20 CH 2000 Neuchâtel Switzerland

英国

MOONS' INDUSTRIES (UK), LIMITED
Rooms 4&5, 1st Floor, Greenbank, London Road, Reading, UK. RG1 5AQ

亚洲地区

新加坡

MOONS' INDUSTRIES (SOUTH-EAST ASIA) PTE. LTD.
33 Ubi Avenue 3 #08-23 Vertex Singapore 408868

日本

MOONS' INDUSTRIES JAPAN CO., LTD. (Yokohama)
Room 602, 6F, Shin Yokohama Koushin Building,
2-12-1, Shin-Yokohama, Kohoku-ku, Yokohama, Kanagawa
Japan 222-0033

印度

MOONS' INTELLIGENT MOTION SYSTEM INDIA PVT. LTD.
Room. 908, 9th Floor, Amar Business Park,
Tal. Haveli, Baner, Pune India 411045

越南

MOONS' INDUSTRIES (VIETNAM) COMPANY LIMITED.
Factory C1&D1, Lot IN3-11*A, VSIP Hai Phong Industrial Park in Dinh Vu – Cat Hai Economic Zone,Lap Le Commune, Thuy Nguyen District, Hai Phong City, Vietnam
Vietnam 04359

ISO 14001 TÜV SÜD ISO 9001 OHSAS 18001

http://www.moons.com.cn
E-mail:ama-info@moons.com.cn

MOONS' 安浦鸣志
moving in better ways