Softlink

Powered By Googoltech

iDEABOX 3

智能控制器编程手册
高级功能

V1.4

2018.04

www.softlink.cn

版权申明

运动中的机器有危险！使用者有责任在机器中设计有效的出错处理和安全保护机制，固高欧辰没有义务或责任对由此造成的附带的或相应产生的损失负责。

联系我们

上海固高欧辰智能科技有限公司

地　　址：上海闵行区东川路 555 号 4 号楼 1 层

客户服务：4006 300 321

电　　话：021-54708386　　54708786

传　　真：021-54708386

电子邮件：info@softlink.cn

网　　址：http://www.softlink.cn

文档版本

版本号	修订日期
1.0	2016 年 02 月 12 日
1.1	2017 年 03 月 07 日
1.2	2017 年 08 月 21 日
1.3	2018 年 01 月 31 日
1.4	2018 年 04 月 16 日

上海固高欧辰智能科技有限公司 www.softlink.cn 021-54708786 021-54708386 上海市闵行区东川路555号4号楼一层

前言

感谢选用固高欧辰控制器

为回报客户，我们将以品质一流的运动控制器、完善的售后服务、高效的技术支持，帮助您建立自己的控制系统。

固高欧辰产品的更多信息

固高欧辰的网址是 http://www.softlink.cn 。在我们的网页上可以得到更多关于公司和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等等。

您也可以通过电话（4006 300 321）咨询关于公司和产品的更多信息。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

电子邮件： info@softlink.cn
电　　话： 4006 300 321
发 函 至：上海闵行区东川路 555 号 4 号楼 1 层
　　上海固高欧辰智能科技有限公司
邮　　编： 200241

编程手册的用途

用户通过阅读本手册，能够了解 iDEABOX 3 智能控制器的控制功能，掌握函数的用法，熟悉特定控制功能的编程实现。最终，用户可以根据自己特定的控制系统，编制用户应用程序，实现控制要求。

编程手册的使用对象

本编程手册适用于具有 C 语言编程基础或 Windows 环境下使用动态链接库的基础，同时具有一定运动控制工作经验，对伺服或步进控制的基本结构有一定了解的工程开发人员。

编程手册的主要内容

本手册由六章内容组成，详细介绍了 iDEABOX 3 智能控制器的**高级运动控制功能**及编程实现。

相关文件

关于 iDEABOX 3 智能控制器调试和安装，请参见随产品配套的《iDEABOX 3 智能控制器用户手册》。

关于 iDEABOX 3 智能控制器配置文件及配置工具，请参见随产品配套的《EtherCAT 配置文件&配置工具 EthercatConfig 使用说明》。

目录

上海固高欧辰智能科技有限公司 www.softlink.cn 021-54708786 021-54708386 上海市闵行区东川路555号4号楼一层

第1章 指令列表

本章表格中右侧的数字为“页码”，其中指令右侧的为“**第 5 章 指令详细说明**”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

本手册中所有字体为蓝色的指令（如 GT_ArcXYC）均带有超级链接，点击可跳转至指令详细说明；不带超链接的指令详细信息请查阅《iDEABOX 3 智能控制器编程手册之基本功能》。

表 1-1 指令列表

上海固高欧辰智能科技有限公司 www.softlink.cn 021-54708786 021-54708386 上海市闵行区东川路555号4号楼一层

第2章 OtoStudio 中运动函数库的使用

2.1 OtoStudio 软件库的使用

使用 CPAC 系列运动控制器，首先需要安装 OtoStudio 开发环境，运动控制器指令函数库将存放在默认路径下。iDEABOX 3 控制器的高级运动控制库文件名为 IDB3_Advance.lib。由于运动控制器要使用 EtherCAT 总线，因此还需要调用 EtherCAT 专用库，库文件名为 IDB3_ECAT.lib，指令详细说明请参考《iDEABOX 3 智能控制器编程手册之基本功能》。

2.1.1 OtoStudio 平台中库的使用

(1) 启动OtoStudio.exe，新建一个工程；

(2) 选择目标平台：iDeabox3 PAC Controller。如果目标系统中没有该选项，请从官网上下载IDB3 Target，并参阅《ReadMe》安装ideabox3.trg；

(3) 选择正确的目标系统后，库文件管理器自动添加IDB3_Standard.lib和IDB3_ECAT.lib；

(4) 手动添加库文件IDB3_Advance.lib；

至此，用户就可以在 OtoStudio 中调用函数库中的任何函数，开始编写应用程序。

第3章 指令返回值及其意义

3.1 本章简介

本章主要介绍了运动控制器指令的所有返回值及其意义。在'第 5 章 指令详细说明'，每一条指令介绍的'指令返回值'一项中，都有更加详细的关于返回值意义和操作的介绍。

3.2 指令返回值

CPAC 控制器按照主机发送的指令工作，相关指令封装在动态链接库中。用户在编写高级运动控制应用程序时，通过调用高级运动控制库 IDB3_Advance.lib 指令来操纵 CPAC 控制器的运动控制。

CPAC 控制器在接收到主机发送的指令时，将执行结果反馈到主机，指示当前指令是否正确执行。指令返回值的定义如下。

表 3-1 运动控制器指令返回值定义

返回值	意义	处理方法
0	指令执行成功	
1	指令执行错误	1. 检查当前指令的执行条件是否满足
7	指令参数错误	1. 检查当前指令输入参数的取值
-1	主机和运动控制器通讯失败	1. 是否正确安装运动控制器驱动程序 2. 检查运动控制器是否接插牢靠 3. 更换主机 4. 更换控制器
-6	打开控制器失败	1. 是否正确安装运动控制器驱动程序 2. 是否调用了 2 次 GT_Open 指令 3. 其他程序是否已经打开运动控制器
-7	运动控制器没有响应	1. 更换运动控制器

建议在用户程序中，检测每条指令的返回值，以判断指令的执行状态。并建立必要的错误处理机制，保证程序安全可靠地运行。

第4章 运动模式

4.1 本章简介

运动模式是指规划一个或多个轴运动的方式。控制器支持的高级运动模式有插补运动模式和 PVT 运动模式。

本章表格中右侧的数字为“页码”，其中指令右侧的为“**第 5 章指令详细说明**”中的对应页码，其他为章节页码，均可以使用“超级链接”进行索引。

本手册中所有字体为蓝色的指令（如 GT_ArcXYC）均带有超级链接，点击可跳转至指令详细说明；不带超链接的指令详细信息请查阅《运动控制器编程手册之基本功能》。

用户需注意，每一个轴在任一时刻只能处在一种运动模式下。

4.2 插补运动模式

插补运动模式可以实现多轴的协调运动，从而完成一定的运动轨迹。该插补运动模式具有以下一些功能，可以实现直线插补和圆弧插补；可以同时有两个坐标系进行插补运动；每个坐标系含有两个缓存区，可以实现缓存区暂停、恢复等功能；具有缓存区延时和缓存区数字量输出的功能；具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

4.2.1 指令列表

表 4-1 设置插补运动指令列表

指令	说明	页码
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	73
GT_GetCrdPrm	查询坐标系参数	63
GT_CrdData	向插补缓存区增加插补数据	60
GT_LnXY	缓存区指令，二维直线插补	65
GT_LnXYZ	缓存区指令，三维直线插补	66
GT_LnXYZA	缓存区指令，四维直线插补	67
GT_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)	66
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	68
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	67
GT_ArcXYR	缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)	54
GT_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)	54
GT_ArcYZR	缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)	56

GT_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)	55
GT_ArcZXR	缓存区指令，ZX 平面圆弧插补(以终点位置和半径为输入参数)	57
GT_ArcZXC	缓存区指令，ZX 平面圆弧插补(以终点位置和圆心位置为输入参数)	56
GT_BufDelay	缓存区指令，缓存区内延时设置指令	58
GT_BufLmtsOn	缓存区指令，缓存区内有效限位开关	59
GT_BufLmtsOff	缓存区指令，缓存区内无效限位开关	58
GT_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动	60
GT_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动	58
GT_CrdSpace	查询插补缓存区剩余空间	61
GT_CrdClear	清除插补缓存区内的插补数据	60
GT_CrdStart	启动插补运动	61
GT_CrdStatus	查询插补运动坐标系状态	62
GT_SetUserSegNum	缓存区指令，设置自定义插补段段号	75
GT_GetUserSegNum	读取自定义插补段段号	64
GT_GetRemainderSegNum	读取未完成的插补段段数	64
GT_SetOverride	设置插补运动目标合成速度倍率	74
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	74
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	63
GT_GetCrdPos	查询该坐标系的当前坐标位置值	63
GT_GetCrdVel	查询该坐标系的合成速度值	63
GT_InitLookAhead	初始化插补前瞻缓存区	65

4.2.2 重点说明

1. 直线插补与圆弧插补

插补运动在数控机床，切削加工工艺等数控装置中应用广泛。它可以实现多轴的协调运动，将数据段所描述的曲线的起点、终点之间的空间进行数据密化，从而形成要求的轮廓轨迹，根据密化后的数据向各个坐标发出进给脉冲，对应每个脉冲，机床在相应的坐标方向上移动一个脉冲当量的距离，从而将工件加工出所需要的轮廓形状。

插补最常见的两种方式是直线插补和圆弧插补。

直线插补方式中，两点间的插补沿着直线的点群来逼近。首先假设在实际轮廓起始点处沿 x 方向走一小段（如一个脉冲当量），发现终点在实际轮廓的下方，则下一条线段沿 y 方向走一小段，此时如果线段终点还在实际轮廓下方，则继续沿 y 方向走一小段，直到在实际轮廓上方以后，再向 x 方向走一小段。依次循环类推。直到到达轮廓终点为止。这样实际轮廓是由一段段的折线拼接而成，虽然是折线，如果我们每一段走刀线段都在精度允许范围内，那么此段折线还是可以近似看做一条直线段。这就是直线插补。假设某数控机床刀具在 xy 平面上从点(x_0, y_0)运动到点(x_1, y_1)，其直线插补的加工过程如图 4-1 所示。

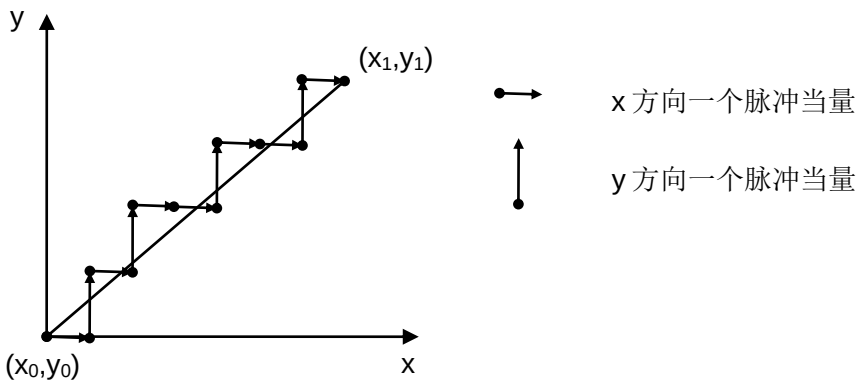


图 4-1 直线插补示意图

圆弧插补是给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制刀具沿这些点运动，加工出圆弧曲线。圆弧插补只能在某一平面进行。假设某数控机床刀具在 xy 平面第一象限走一段逆圆弧，圆心为原点，半径为 5，起点 A(5, 0)，终点 B(0, 5)，其圆弧插补的加工过程如图 4-2 所示。

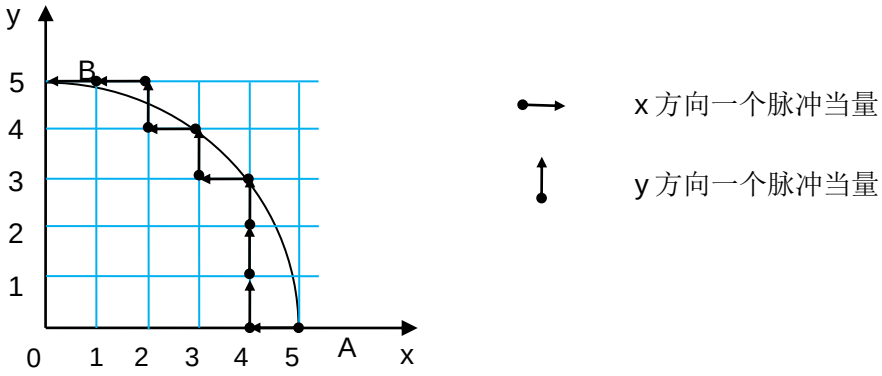


图 4-2 圆弧插补示意图

2. 运动控制器的插补模式

运动控制器的插补运动模式具有以下功能：

(1) 可以实现直线插补和圆弧插补；

(2) 可以同时有两个坐标系进行插补运动；

(3) 每个坐标系含有两个缓存区，可以实现缓存区暂停、恢复等功能；

(4) 具有缓存区延时和缓存区数字量输出的功能；

(5) 具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

3. 使用插补模式的步骤

使用插补模式需要至少两步操作：建立坐标系和向缓存区存入数据。下面分别就这两个步骤分别进行详细讲解。

(1) 建立坐标系

在运动控制器的初始状态下，复位之后或者还未使用过插补运动状态下，所有的规划轴都处于单轴运动模式下，两个坐标系也是无效的。所以，进行插补运动时，首先需要建立坐标系，将规划轴映射到相应

的坐标系中。每个坐标系最多支持四维(X-Y-Z-A)，用户根据自己的需求，也可以利用二维(X-Y)、三维(X-Y-Z)坐标系描述运动轨迹。

用户通过调用指令 GT_SetCrdPrm 指令将在坐标系内描述的运动通过映射关系映射到相应的规划轴上。运动控制器根据坐标映射关系，控制各轴运动，实现要求的运动轨迹。调用指令 GT_SetCrdPrm 时，所映射的各规划轴必须处于静止状态。

例程 4-1 建立坐标系

建立了一个二维坐标系，规划轴 1 对应为 x 轴，规划轴 2 对应为 y 轴，坐标系原点的规划位置是(100, 100)，单位：pulse，在此坐标系内运动的最大合成速度为 500pulse/ms，最大合成加速度为 1pulse/ms^2，最小匀速时间为 50ms。如图 4-3 所示。

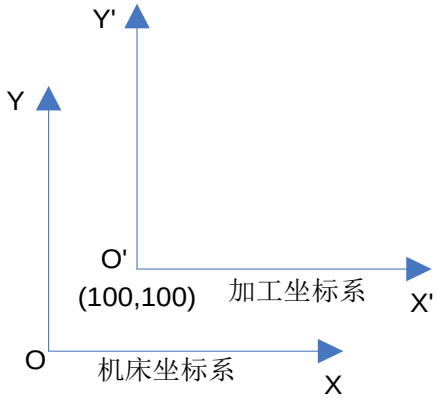


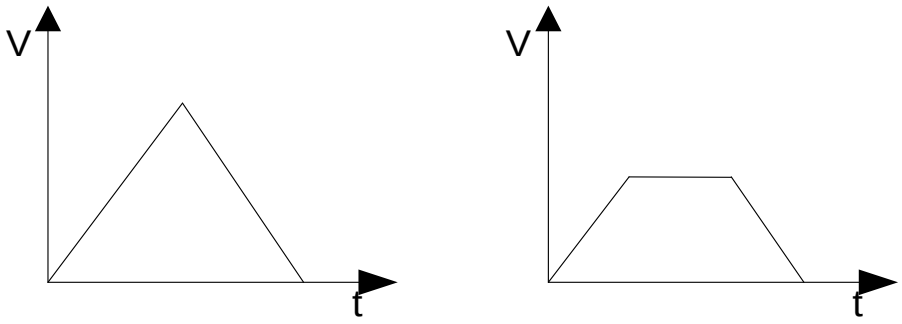
图 4-3 加工坐标系偏移量示意图

```
…… …… ……
(*指令返回值变量*)
rtn: INT;
(*TCrdPrm结构体变量，该结构体定义了坐标系*)
crdPrm:TCrdPrm;
(* @END_DECLARATION := '0' *)
(*将结构体变量初始化为0*)
SysMemSet(ADR(crdPrm), 0, SIZEOF(crdPrm));
(*为结构体赋值*)
crdPrm.dimension:= 2;          (*坐标系为二维坐标系*)
crdPrm.synVelMax:= 500;        (*最大合成速度：500pulse/ms*)
crdPrm.synAccMax:= 1;          (*最大加速度：1pulse/ms^2*)
crdPrm.evenTime:= 50;          (*最小匀速时间：50ms*)
crdPrm.profile[0]:= 1;         (*规划器1对应到X轴*)
crdPrm.profile[1]:= 2;         (*规划器2对应到Y轴*)
crdPrm.setOriginFlag:= 1;      (*表示需要指定坐标系的原点坐标的规划位置*)
crdPrm.originPos[0]:= 100;     (*坐标系的原点坐标的规划位置为（100, 100）*)
crdPrm.originPos[1]:= 100;

(*建立1号坐标系，设置坐标系参数*)
rtn:= GT_SetCrdPrm(1, ADR(crdPrm));
…… …… ……
```

例程说明：

- dimension：表示所建立的坐标系的维数，取值范围：[1, 4]，该例程中所建立的坐标系是二维，即 X-Y 坐标系。
- synVelMax：表示该坐标系所能承受的最大合成速度，如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。
- synAccMax：表示该坐标系所能承受的最大合成加速度，如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。
- evenTime：每个插补段的最小匀速时间。当用户设置的插补段比较短时，而该插补段的目标速度又设置的比较大，则会造成合成速度的曲线如图 4-4 (a)所示，只有加速段和减速段，形成一个速度尖角，加速度在尖角处瞬间由正值变为了负值，造成较大的冲击；设置了 evenTime 之后，可以减小目标速度，使速度曲线如图 4-4 (b)所示，减小了加速度突变的冲击。



(a) evenTime=0　　(b) evenTime>0

图 4-4 不同 evenTime 下的速度曲线

(2) 向缓存区存入数据

运动控制器插补运动模式采用缓存区运动方式，即用户需要向插补缓存区中传递插补数据，然后，启动插补运动，运动控制器则会依次执行用户所传递的插补数据，直到所有的插补数据全部运动完成。

向缓存区存入数据的指令分直线插补（以 GT_Ln 开头）和平面圆弧插补指令（以 GT_Arc 开头）两种。

例程 4-2 直线插补例程

假设某数控机床刀具在 xy 平面从原点出发，走一段如图 4-5 所示的正六边形轨迹。一共需要走七段轨迹，图中标号已标出。每走完一段轨迹会输出一次 IO 信号，并且暂停 400ms，其直线插补的例程如下，直线插补例程运动轨迹如图 4-5 所示。

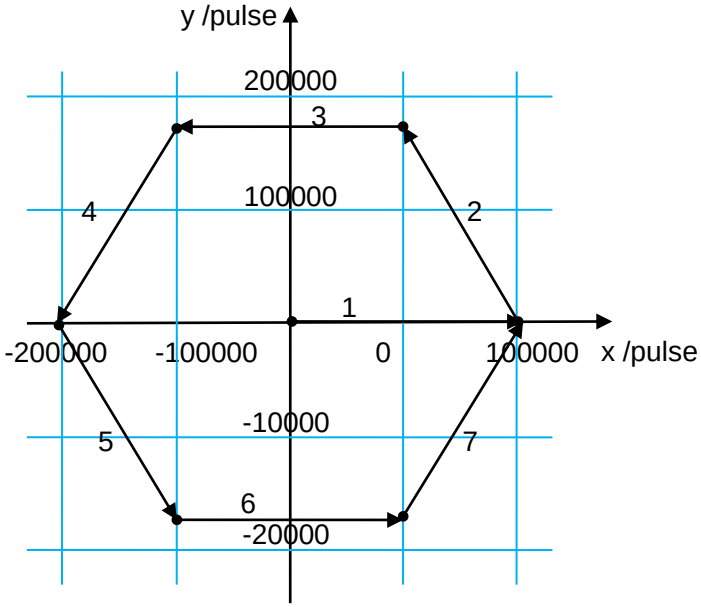


图 4-5 直线插补例程运动轨迹

```
…… …… ……
(*指令返回值变量*)
rtn: INT;
(*坐标系运动状态查询变量*)
run: INT;
(*坐标系运动完成段查询变量*)
segment: DINT;
(*坐标系的缓存区剩余空间查询变量*)
space: DINT;
(* @END_DECLARATION := '0' *)
(*即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据*)
rtn:= GT_CrdClear(1, 0);
(*向缓存区写入第一段插补数据*)
rtn:= GT_LnXY(
    1,              (*该插补段的坐标系是坐标系1*)
    200000, 0,      (*该插补段的终点坐标(200000, 0)*)
    100,            (*该插补段的目标速度：100pulse/ms*)
    0.1,            (*插补段的加速度：0.1pulse/ms^2*)
    0,              (*终点速度为0*)
    0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

(*向缓存区写入第二段插补数据*)
rtn:= GT_LnXY(1, 100000, 173205, 100, 0.1, 0, 0);
(*第三段插补数据*)
rtn:= GT_LnXY(1, -100000, 173205, 100, 0.1, 0, 0);
(*第四段插补数据*)
rtn:= GT_LnXY(1, -200000, 0, 100, 0.1, 0, 0);
(*缓存区延时指令*)
```

```
rtn:= GT_BufDelay(
    1,              (*坐标系是坐标系1*)
    400,            (*延时400ms*)
    0);             (*向坐标系1的FIFO0缓存区传递该延时*)
(*第五段插补数据*)
rtn:= GT_LnXY(1, -100000, -173205, 100, 0.1, 0, 0);
(*缓存区延时指令*)
rtn:= GT_BufDelay(1, 100, 0);
(*第六段插补数据*)
rtn:= GT_LnXY(1, 100000, -173205, 100, 0.1, 0, 0);
(*第七段插补数据*)
rtn:= GT_LnXY(1, 200000, 0, 100, 0.1, 0, 0);
(*查询坐标系1的FIFO0所剩余的空间*)
rtn:= GT_CrdSpace(1, ADR(space), 0);
(*启动坐标系1的FIFO0的插补运动*)
rtn:= GT_CrdStart(1, 0);

(*查询坐标系1的FIFO的插补运动状态*)
rtn:= GT_CrdStatus(
    1,              (*坐标系是坐标系1*)
    ADR(run),       (*读取插补运动状态*)
    ADR(segment),   (*读取当前已经完成的插补段数*)
    0);             (*查询坐标系1的FIFO0缓存区*)
```

…… …… ……

(3) 圆弧插补

运动控制器的插补模式支持在 XY 平面、YZ 平面和 ZX 平面的圆弧插补。其中圆弧插补的旋转方向按照右手螺旋定则定义为：从坐标平面的“上方”(即垂直于坐标平面的第三个轴的正方向)看，来确定逆时针方向和顺时针方向。可以这样简单记忆：将右手拇指前伸，其余四指握拳，拇指指向第三个轴的正方向，其余四指的方向即为逆时针方向。映射坐标系为二维坐标系(X-Y)时，XOY 坐标平面内的圆弧插补逆时针方向同样定义，如图 4-6 示。

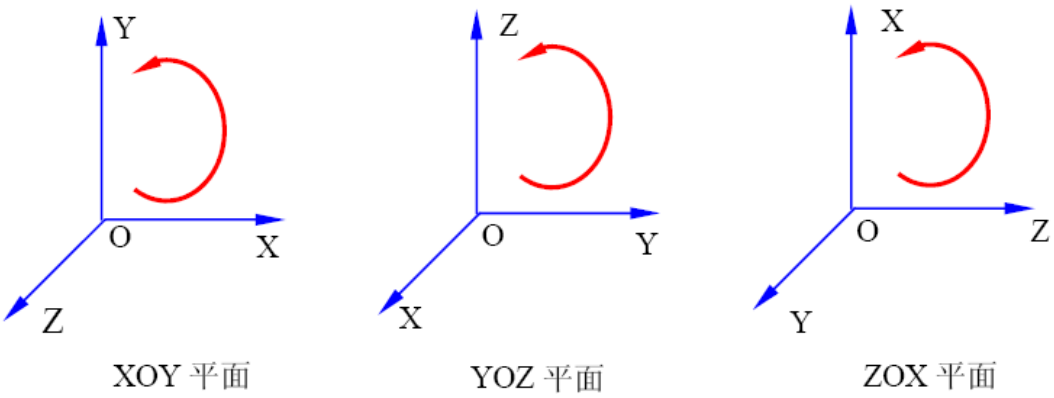


图 4-6 圆弧插补逆时针方向

圆弧插补有两种描述方法：半径描述方法和圆心坐标描述方法，用户可以根据加工数据选择合适的描述方法来编程。所使用的描述方法遵循 G 代码的编程标准。

a) 半径描述方法

调用指令 GT_ArcXYR、GT_ArcYZR、GT_ArcZXR 是使用半径描述方法对圆弧进行描述。使用半径描述方法，用户需要输入圆弧终点坐标、圆弧半径、圆弧的旋转方向、速度和加速度等。其中参数半径可为正值，也可为负值，其绝对值为圆弧的半径，正值表示圆弧的旋转角度≤180°，负值表示圆弧的旋转角度＞180°，如图 4-7 所示，半径描述方法无法描述 360°的整圆。

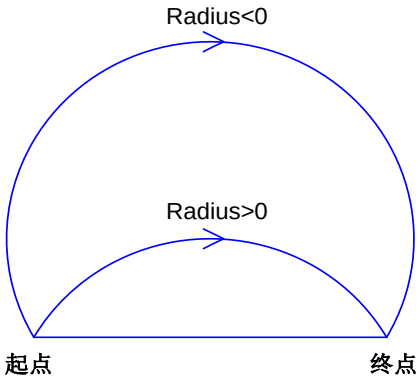


图 4-7 半径取正值/负值圆弧插补示意图

b) 圆心坐标描述方法

调用指令 GT_ArcXYC、GT_ArcYZC、GT_ArcZXC 是使用圆心坐标描述方法对圆弧进行描述。使用圆心描述方法，用户需要输入圆弧终点坐标、圆心相对于起点坐标的相对位置值、圆弧的旋转方向、速度和加速度等。其中，圆心位置值参数的定义如图 4-8 所示。

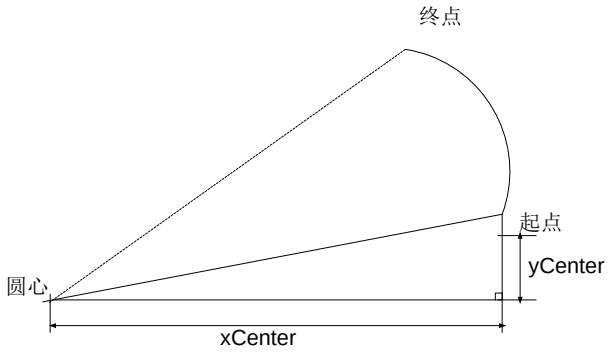


图 4-8 圆心坐标描述方法示意图

圆心参数为相对于起点位置的增量值，带正负号，如果起点的坐标为(xStart, yStart)，用户设置的圆心参数为(xCenter, yCenter)，则圆心坐标值为(xStart+xCenter, yStart+yCenter)。用户设置的起点坐标和终点坐标重合时，则表示将要进行一个整圆的运动。

注意	用户应该保证圆弧描述指令可以正确描述一段圆弧，如果用户所设置的参数不能生成一段正确的圆弧，指令会返回 7(参数错误)。

例程 4-3 圆弧插补例程

假设某数控机床刀具在 xy 平面走一段如图 4-9 所示的圆弧。该例程使用圆心坐标描述方法描述一个整圆，使用半径描述方法描述一个 1/4 圆弧，最后回到原点位置。一共需要走三段轨迹，图中用标号标出。整圆的圆心坐标为(100000, 0)，半径 100000pulse。圆弧的圆心坐标为原点(0, 0)，半径 200000pulse。

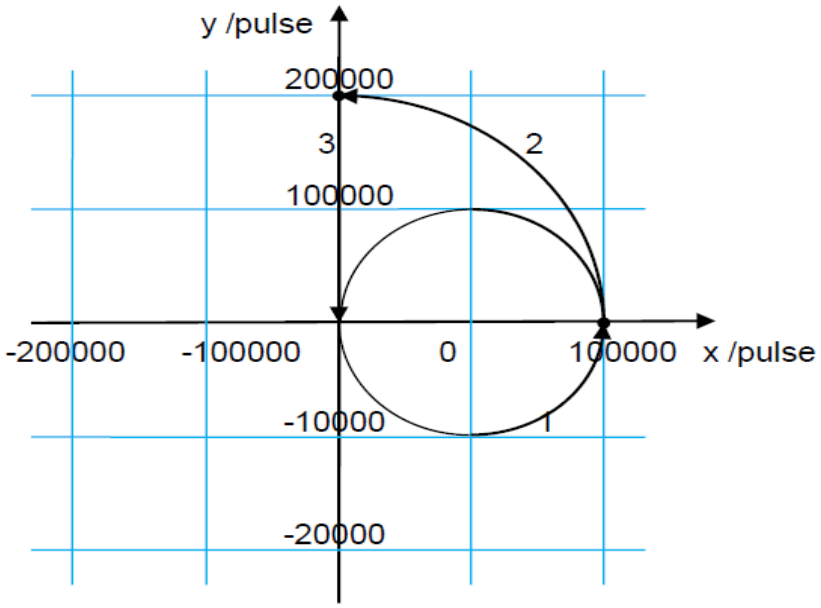


图 4-9 圆弧插补例程运动轨迹

```
…… …… ……
(*指令返回值变量*)
rtn: INT;
(*坐标系运动状态查询变量*)
run: INT;
(*坐标系运动完成段查询变量*)
segment: DINT;
(*坐标系的缓存区剩余空间查询变量*)
space: DINT;
(* @END_DECLARATION := '0' *)
(*即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据*)
rtn:= GT_CrdClear(1, 0);
(*向缓存区写入第一段插补数据*)
rtn:= GT_LnXY(
    1,                  (*该插补段的坐标系是坐标系1*)
    200000, 0,          (*该插补段的终点坐标(200000, 0)*)
    100,                (*该插补段的目标速度：100pulse/ms*)
    0.1,                (*插补段的加速度：0.1pulse/ms^2*)
    0,                  (*终点速度为0*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

(*向缓存区写入第二段插补数据，该段数据是以圆心描述方法描述了一个整圆*)
rtn:= GT_ArcXYC(
    1,                  (*坐标系是坐标系1*)
    200000, 0,          (*该圆弧的终点坐标(200000, 0)*)
    -100000, 0,         (*圆弧插补的圆心相对于起点位置的偏移量(-100000, 0)*)
    0,                  (*该圆弧是顺时针圆弧*)
    100,                (*该插补段的目标速度：100pulse/ms*)
    0.1,                (*该插补段的加速度：0.1pulse/ms^2*)
```

```
        0,                  (*终点速度为0*)
        0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

    (*向缓存区写入第三段插补数据，该段数据是以半径描述方法描述了一个1/4圆弧*)
    rtn:= GT_ArcXYR(
        1,                  (*坐标系是坐标系1*)
        0, 200000,          (*该圆弧的终点坐标(0, 200000)*)
        200000,             (*半径：200000pulse*)
        1,                  (*该圆弧是逆时针圆弧*)
        100,                (*该插补段的目标速度：100pulse/ms*)
        0.1,                (*该插补段的加速度：0.1pulse/ms^2*)
        0,                  (*终点速度为0*)
        0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

    (*向缓存区写入第四段插补数据，回到原点位置*)
    rtn:= GT_LnXY(1, 0, 0, 100, 0.1, 0, 0);
    (*查询坐标系1的FIFO0所剩余的空间*)
    rtn:= GT_CrdSpace(1, ADR(space), 0);
    (*启动坐标系1的FIFO0的插补运动*)
    rtn:= GT_CrdStart(1, 0);

    (*查询坐标系1的FIFO的插补运动状态*)
    rtn:= GT_CrdStatus(
        1,                  (*坐标系是坐标系1*)
        ADR(run),           (*读取插补运动状态*)
        ADR(segment),       (*读取当前已经完成的插补段数*)
        0);                 (*查询坐标系1的FIFO0缓存区*)
    …… …… ……
```

4. 以 GT_Ln 开头 G0 结尾的指令

这一类指令包括 GT_LnXYG0、GT_LnXYZG0 和 GT_LnXYZAG0。这类运动指令将会完成一个完整的加减速过程，即每段运动的合成速度都是从 0 开始，结束的时候也是 0。

这类指令与其他插补运动指令(包括直线插补指令和圆弧插补指令)的区别在于：如果使用了前瞻预处理功能，调用其他插补运动指令，则用户设置的终点速度将会无效，实际的终点速度是前瞻预处理模块根据用户设置的前瞻预处理参数和运动轨迹计算出来的一个合理的终点速度；但如果调用 GT_LnXYG0 系列指令，终点速度仍然是 0，控制器不会优化或改变这类指令的终点速度。

5. 缓存区 FIFO 的管理（运动暂停与恢复）

每个坐标系包含两个缓存区(FIFO)：FIFO0 和 FIFO1，其中 FIFO0 为主要运动 FIFO，FIFO1 为辅助运动 FIFO，每个 FIFO 都含有 4096 段插补数据的空间。

缓存区的释放：用户如果不使用插补模式，直接切换到其他运动模式，插补模式的缓存区就自动释放了。如调用指令 GT_PrfPvt 即可。

在运动控制器的插补模式下，不能随意切换到其他运动模式，否则会导致插补坐标系破坏，并且原来

压入插补缓存区的数据会丢失。但是在实际应用中，经常会有类似下面例子描述的情况。假如机床在走一段轨迹的途中需要暂停下来，更换路径换刀或者移到安全的地方以查看加工效果，然后再回到暂停时的坐标继续完成剩余的轨迹。为了实现上述操作，应利用每个坐标系提供的两个缓存区 FIFO0 和 FIFO1。

FIFO0 是主运动 FIFO，用户的主体插补运动的插补数据应该放在 FIFO0 中。FIFO0 的插补运动可以被中断（通过调用 GT_Stop 指令），中断后可以进行辅助 FIFO1 的插补运动，辅助 FIFO1 的插补运动完成后，需要将坐标系位置恢复到 FIFO0 主运动被打断的位置，之后 FIFO0 可从断点处继续恢复原来的运动（恢复运动调用 GT_CrdStart 指令）。

FIFO1 是辅助运动 FIFO，用户的辅助插补运动的插补数据可以放在 FIFO1 中，FIFO1 的插补数据必须在 FIFO0 的运动停止的情况下才能输入，如果 FIFO0 在运动，向 FIFO1 中传递插补数据，将会提示错误。FIFO1 的插补运动也可以暂停、恢复，但是，在 FIFO1 暂停时不可以进行 FIFO0 的插补运动，否则，FIFO1 缓存区将会被清空，不可以再恢复 FIFO1 的运动，插补主运动与辅助运动流程如图 4-10 所示。

注意	在主运动 FIFO0 的运动暂停之后，又进行了 FIFO1 的运动，如果用户希望在 FIFO1 运动结束之后，继续进行 FIFO0 的运动，则用户必须保证，FIFO1 运动结束后，坐标位置值与 FIFO0 停止时的坐标位置值(断点位置)相同，否则，FIFO0 将不接受启动运动指令，调用 GT_CrdStart 指令恢复启动 FIFO0 的运动，将会提示错误。

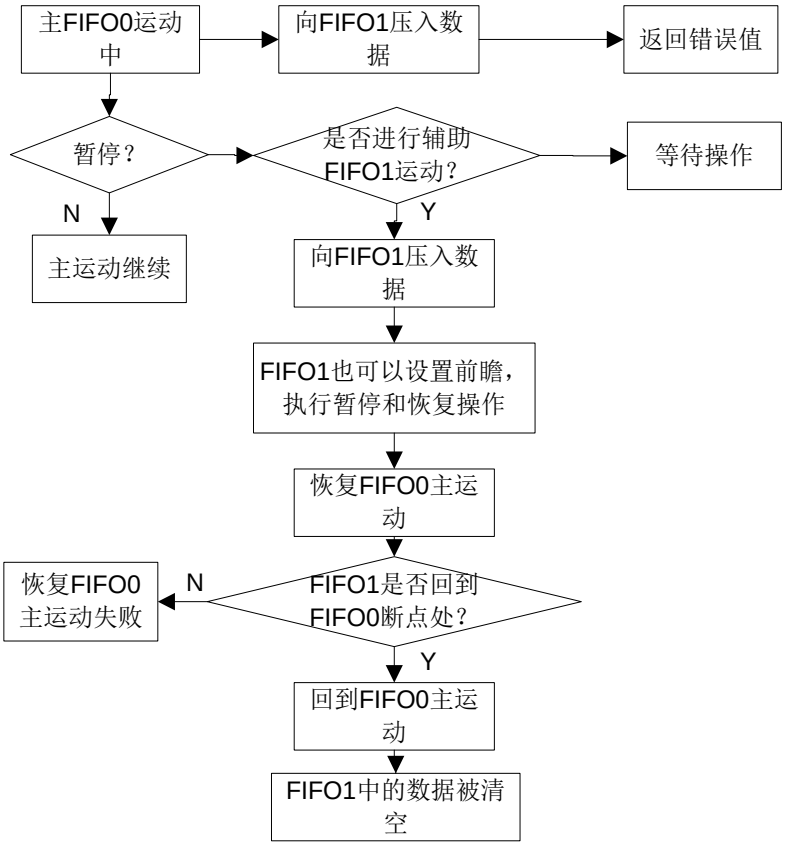


图 4-10 插补主运动与辅助运动流程

例程 4-4 插补 FIFO 管理

假设机床需要做运动：主加工运动需要从(0, 0)运动到(100000, 100000)位置，中途在(50000, 50000)附近出现了断刀，需要暂停运动去换刀。由于主加工运动已经将预定的轨迹数据压入了缓存区，若重新压

入新的数据则会破坏原来的运动，因此需要启动辅助运动来协助完成。

假设换刀轨迹是先走到(70000, 30000)，然后走到(110000, 50000)位置完成换刀动作。但为了能继续主加工运动，需要回到暂停点，如图 4-11 所示。

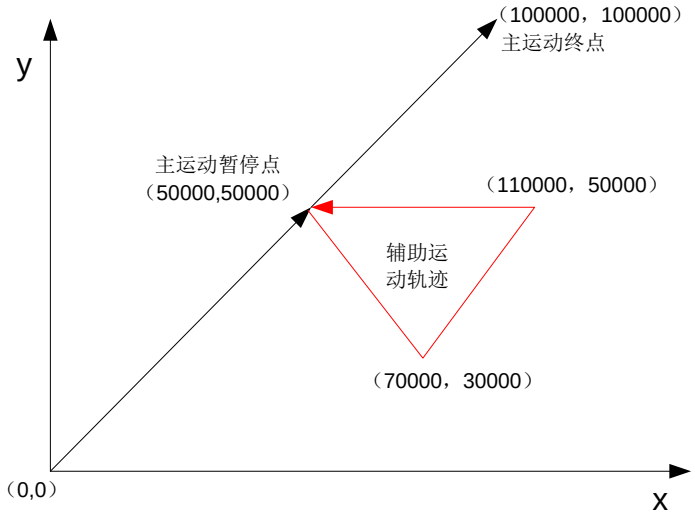


图 4-11 插补 FIFO 管理例程之换刀轨迹

```
…… …… ……
rtn: INT;                    (*函数返回值*)
i:INT;                       (*循环变量*)
run: INT;                    (*查询坐标系运行标志*)
seg: DINT;                   (*查询坐标系运行的段数*)
flag:INT:= 0;                    (*本程序局部运行标志*)
crdPos:ARRAY [0..1] OF LREAL;        (*坐标系规划位置*)
crdstoppos:ARRAY [0..1] OF LREAL;    (*坐标系暂停位置*)
crdPrm:TCrdPrm;              (*坐标系结构体*)
(* @END_DECLARATION := '0' *)
CASE flag OF
0:
    (*清除fifo数据*)
    rtn:= GT_CrdClear(1, 0);
    rtn:= GT_CrdClear(1, 1);

    (*向FIFO0缓存区写入主运动插补数据*)
    rtn:= GT_LnXY(
        1,                       (*该插补段的坐标系是坐标系1*)
        100000, 100000,          (*该插补段的终点坐标(100000, 100000)*)
        10,                      (*该插补段的目标速度：10pulse/ms*)
        1,                       (*插补段的加速度：1pulse/ms^2*)
        0,                       (*终点速度为0*)
        0);                      (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
    (*启动主运动，并将标志flag置1*)
    rtn:= GT_CrdStart(1, 0);
    flag:= 1;
```

```
1:
    (*查询插补坐标位置*)
    rtn:= GT_GetCrdPos(1, ADR(crdPos));
    (*当X轴的坐标位置大于50000时，暂停*)
    IF (crdPos[0] > 50000.0) THEN
        (*停止坐标系1的运动*)
        rtn:= GT_Stop(SHL(WORD#1,12), SHL(WORD#1,12));
        flag:= 2;
    END_IF
2:
    (*注意:要等坐标系真正停止下来去获取当前的位置*)
    (*读取坐标系的状态，查看是否停止*)
    rtn:= GT_CrdStatus(1, ADR(run), ADR(seg), 0);
    IF (run == 0) THEN
        rtn:= GT_GetCrdPos(1, ADR(crdstoppos));
        flag:= 3;
    END_IF
3:
    (*向FIFO1缓存区写入辅助运动插补数据*)
    rtn:= GT_LnXY(1, 70000, 30000, 10, 1, 0, 1);
    rtn:= GT_LnXY(1, 110000, 50000, 10, 1, 0, 1);
    (*启动坐标系1的FIFO1中运动*)
    rtn:= GT_CrdStart(1, 1);
    flag:= 4;
4:
    (*查询插补规划位置*)
    rtn:= GT_GetCrdPos(1, ADR(crdPos));
    (*等待辅助运动完毕*)
    rtn:= GT_CrdStatus(1, ADR(run), ADR(seg), 1);
    IF (run == 0) THEN
        (*向FIFO1压入暂停后的位置点*)
        rtn:= GT_LnXY(
                1,
                LREAL_TO_DINT(crdstoppos[0]),
                LREAL_TO_DINT(crdstoppos[1]),
                10,
                1,
                0,
                1);
        (*启动回暂停点运动*)
        rtn:= GT_CrdStart(1, 1);
        flag:= 5;
    END_IF
5:
    (*查询插补规划位置*)
```


上海固高欧辰智能科技有限公司 www.softlink.cn 021-54708786 021-54708386 上海市闵行区东川路555号4号楼一层

```
        rtn:= GT_GetCrdPos(1, ADR(crdPos));
        (*等待回到暂停点位置运动完毕
        rtn:= GT_CrdStatus(1, ADR(run), ADR(seg), 1);
        IF (run == 0) THEN
            (*恢复主运动*)
            rtn:= GT_CrdStart(1, 0);
            flag:= 6;
        END_IF
    6:
        (*获取当前的位置*)
        rtn:= GT_GetCrdPos(1, ADR(crdPos));
        (*等待主运动完毕*)
        rtn:= GT_CrdStatus(1, ADR(run), ADR(seg), 1);
    END_CASE
    …… …… ……
```

6. 前瞻预处理

在数控加工等应用中，要求数控系统对机床进行平滑的控制，以防止较大的冲击影响零件的加工质量。运动控制器的前瞻预处理功能可以根据用户的运动路径计算出平滑的速度规划，减少机床的冲击，从而提高加工精度。

下面用一个实例来说明前瞻预处理的机制优势。假设机床要加工一个长方形的零件，刀具所走的轨迹如图 4-12 (a)所示。假设 m 点到 n 点距离 3000 个单位长度，有 30 段规划。n 点到 p 点距离 2000 个单位长度，有 20 段规划。每段规划 100 个单位长度。

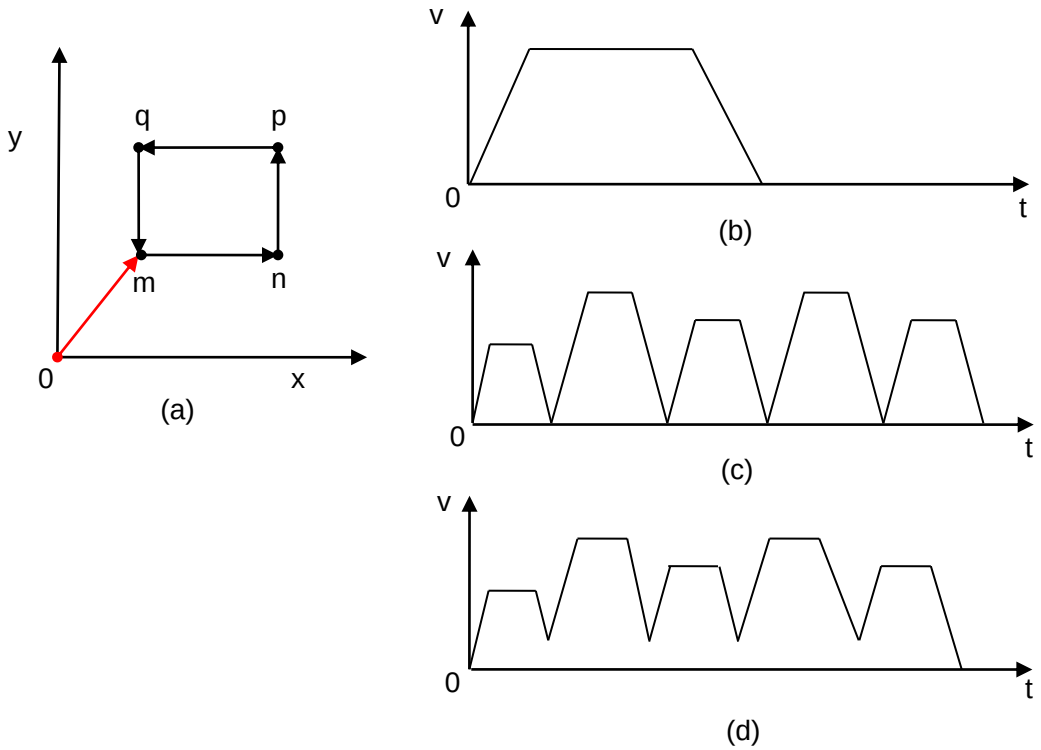


图 4-12 使用前瞻与不使用前瞻的速度规划区别

如果按照图 4-12 (b)所示的速度规划，即在拐角处不减速，则加工精度一定会较低，而且可能在拐弯

时对刀具和零件造成较大冲击。如果按照图 4-12 (c)所示的速度规划，即在拐角处减速为 0，可以最大限度保证加工精度，但加工速度就会慢下来。如果按照图 4-12(d)所示的速度规划，在拐角处将速度减到一个合理值，既可以满足加工精度又能提高加工速度，就是一个好的速度规划。

为了实现类似图 4-12 (d)所示的好的速度规划，前瞻预处理模块不仅要知道当前运动的位置参数，还要提前知道后面若干段运动的位置参数，这就是所谓的前瞻。例如在对图 4-12 (a)中的轨迹做前瞻预处理时，我们设定控制器预先读取 50 段运动轨迹到缓存区中，则它会自动分析出在第 30 段将会出现拐点，并依据用户设定的拐弯时间计算在拐弯处的终点速度。前瞻预处理模块也会依照用户设定的最大加速度值计算速度规划，使任何加减速过程都不会超过这个值，防止对机械部分产生破坏性冲击力。

从下图 4-13 可以直观地了解，使用前瞻预处理功能模块来规划速度，在小线段加工过程中的对速度的显著提升。前瞻预处理流程图如图 4-14 所示。

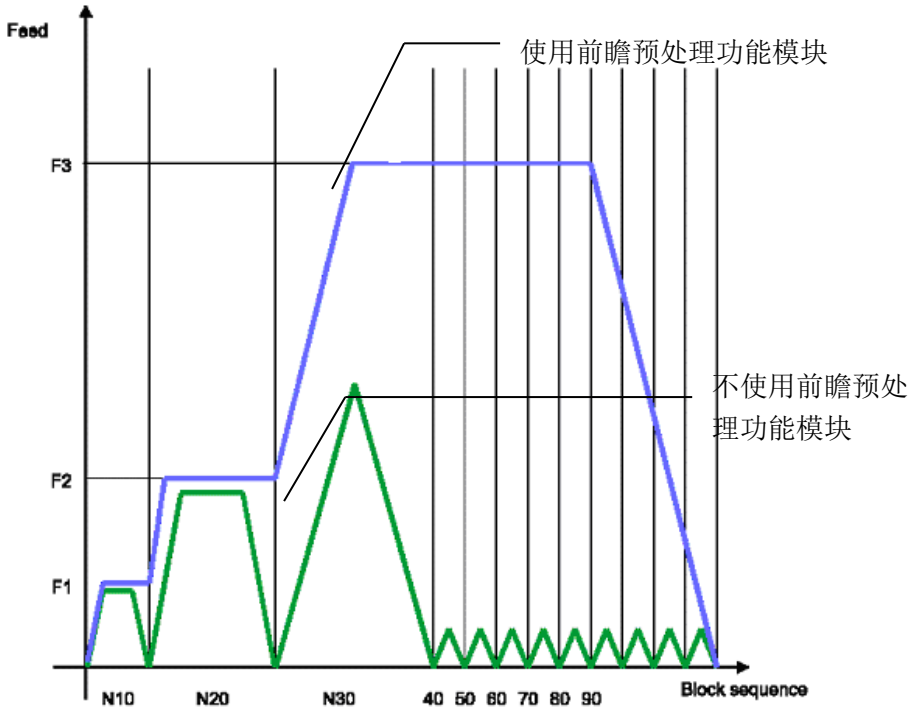


图 4-13 使用和不使用前瞻预处理功能模块的速度曲线对比图

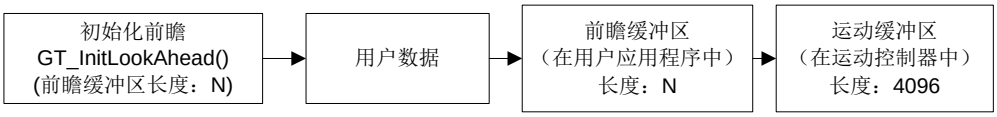


图 4-14 前瞻预处理流程图

(1) 用户在应用程序中定义一个结构体 TCrdData 的数组作为前瞻缓存区，数组大小用户自己设定，比如数组大小为 200，那么前瞻缓存区长度 N=200 段；然后调用指令 GT_InitLookAhead 作初始化前瞻。

(2) 用户调用如 GT_LnXY 等直线插补指令和圆弧插补指令将数据段输入缓存区。这时，插补数据先流入开辟的前瞻缓存区，当流入前瞻缓存区的数据段数大于了 N 之后，才能逐段流入运动缓存区。运动缓存区是控制器内部资源，大小 4096 段，每一段可以存放一条指令。控制器只能执行压入运动缓存区中的数据，所以用户一定要确保前瞻缓存区的数据进入运动缓存区。分不同情况分析：

1) 假设用户数据只有 190 段，则当用户调用完后，数据会一直停留在前瞻缓存区，因此，用户需要调用 GT_CrdData(1, 0, 0)来将前瞻缓存区数据压入运动缓存区。

2) 假设用户数据只有 300 段，则当用户调用完后，有 100 段数据已经流入运动缓存区，但还有 200 段留在前瞻缓存区，同样，用户需要调用 GT_CrdData(1, 0, 0)来将前瞻缓存区数据压入运动缓存区。

3) 假设用户有数据 5000 段，则在用户调用插补指令过程中，会出现前瞻缓存区和运动缓存区都被压满的情况，因此，需要注意，若一直压数据，没有启动插补运动，当压入第 4297 段时（前瞻缓存区和运动缓存区一共 4296 段大小），由于两缓存区都满了，所以调用指令会返回 1 。此时需要调用 GT_CrdSpace 查询运动缓存区的空间，只有当查询到当前运动缓存区有空间时，才能继续调用插补指令压入剩下的数据。同样，最后用户需要调用 GT_CrdData(1, 0, 0)来将前瞻缓存区数据压入运动缓存区。

前瞻预处理功能只支持 3 轴或者 3 轴以下的插补运动，如果建立的坐标系大于 3 轴，则在使用前瞻预处理功能时，会返回错误值，调用缓存区指令时，会返回 7(参数错误)。

例程 4-5 前瞻预处理例程

假设机床加工过程中，需要走一长直线，该直线由 300 条小直线段组成，现对这段路径进行前瞻预处理。其轨迹如图 4-15 所示。红色线段为起始轨迹。

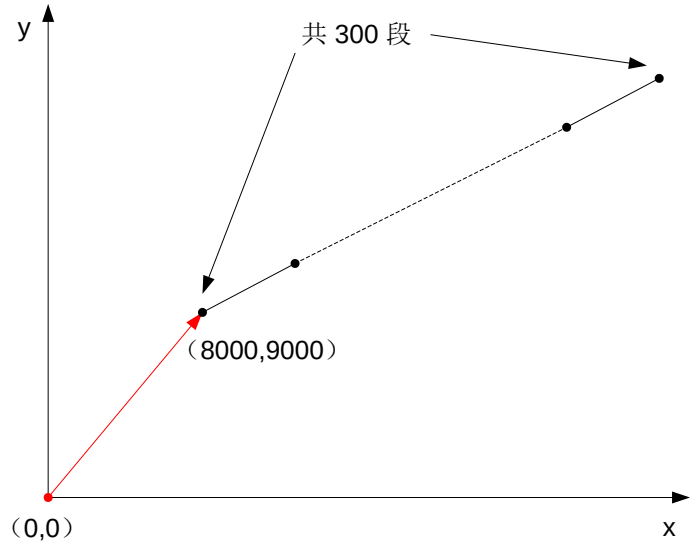


图 4-15 前瞻预处理例程之运动轨迹图

```
…… …… ……
rtn: INT;                    (*函数返回值*)
i:INT;                       (*循环变量*)
crdData: ARRAY [0..209] OF TCrdData; (*定义前瞻缓存区内存区*)
posTest:ARRAY [0..1] OF DINT;
space:DINT;
(* @END_DECLARATION := '0' *)
(*初始化坐标系1的FIFO0的前瞻模块*)
rtn:= GT_InitLookAhead(1, 0, 5, 1, 200, ADR(crdData));
(*压插补数据：小线段加工*)
posTest[0]:= 0;
posTest[1]:= 0;
```

```
FOR i:=0 TO 299 DY 1 DO
    rtn:= GT_LnXY(1, 8000+posTest[0], 9000+posTest[1], 100, 0.8, 0, 0);
    posTest[0]:= posTest[0]+1600;
    posTest[1]:= posTest[1]+1852;
END_FOR
(*将前瞻缓存区中的数据压入控制器*)
rtn:= GT_CrdData(1, 0, 0);
(*启动运动*)
rtn:= GT_CrdStart(1, 0);
…… …… ……
```

例程说明：

- 拐弯时间(T)：GT_InitLookAhead 指令的第三个参数，单位：ms。T 的经验范围是：1ms~10ms，T 越大，计算出来的终点速度越大，但却降低了加工精度；反之，提高了加工的精度，但计算出的终点速度偏低。因此要合理选择 T 值。
- 最大加速度(accMax)：GT_InitLookAhead 指令的第四个参数，单位：pulse/ms2。系统能承受的最大加速度，根据不同的机械系统和电机驱动器取值不同。
- 前瞻缓存区(pLookAheadBuf)：前瞻缓存区是用户在应用程序中自己定义的，用于存放描述运动轨迹的数组。用户应根据自己的需要以及计算机的条件定义合适的缓存区大小，并且要在 GT_InitLookAhead 指令的第五个参数中说明数组的大小。

调用 GT_InitLookAhead 指令之后，在进行前瞻预处理的过程中，用户不能再对前瞻缓存区进行任何操作，否则将会破坏前瞻缓存区中的数据，造成数据的错误。

- 运动缓存区：插补缓存区是运动控制器内部专门用于插补运动的缓存区资源，大小 4096 段，每一段可以放一条指令。

  当前瞻缓存区的段数不为 0 时，用户调用缓存区指令传递的插补数据先进入前瞻缓存区，当前瞻缓存区放满之后，如果再有新的数据传入，最先进入前瞻缓存区的数据，则会进入插补缓存区。

  如果用户所有的插补数据已经输入完毕，前瞻缓存区中还有数据没有进入插补缓存区，这时，需要调用 GT_CrdData(1, 0, 0)，运动控制器会将前瞻缓存区的数据依次传递给插补缓存区，直到前瞻缓存区被清空为止。

  在数据量比较大的时候，用户需要配合 GT_CrdSpace 指令查询插补缓存区的剩余空间，在有空间的时候再调用缓存区指令传递数据，如果插补缓存区已满，调用缓存区指令将会返回错误，说明该段插补数据没有输入成功，需要再次输入该段插补数据。

- 没有前瞻预处理和经过前瞻预处理的区别如图 4-16 和图 4-17 所示。

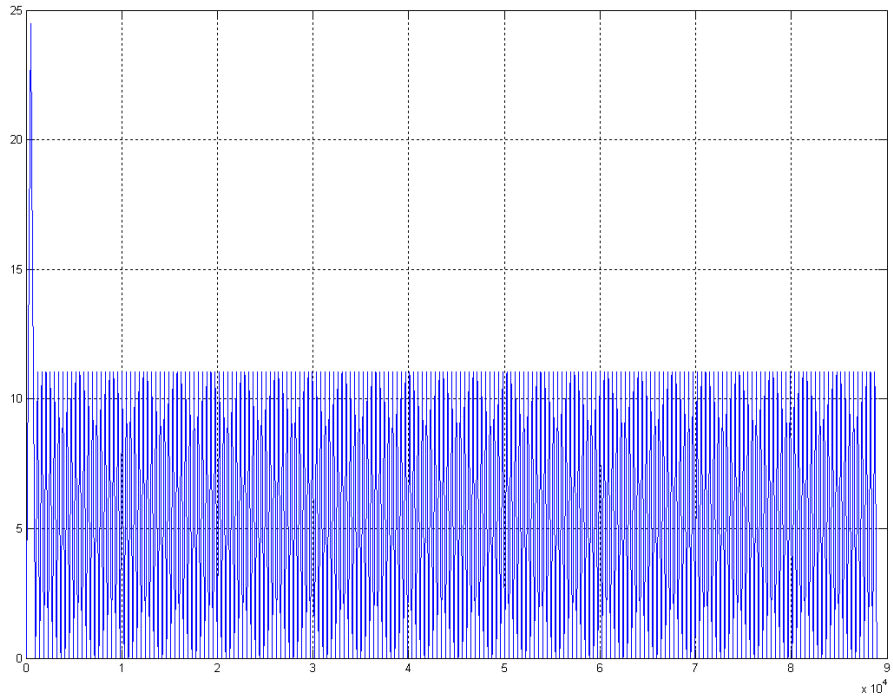


图 4-16 没有进行前瞻预处理的合成速度曲线

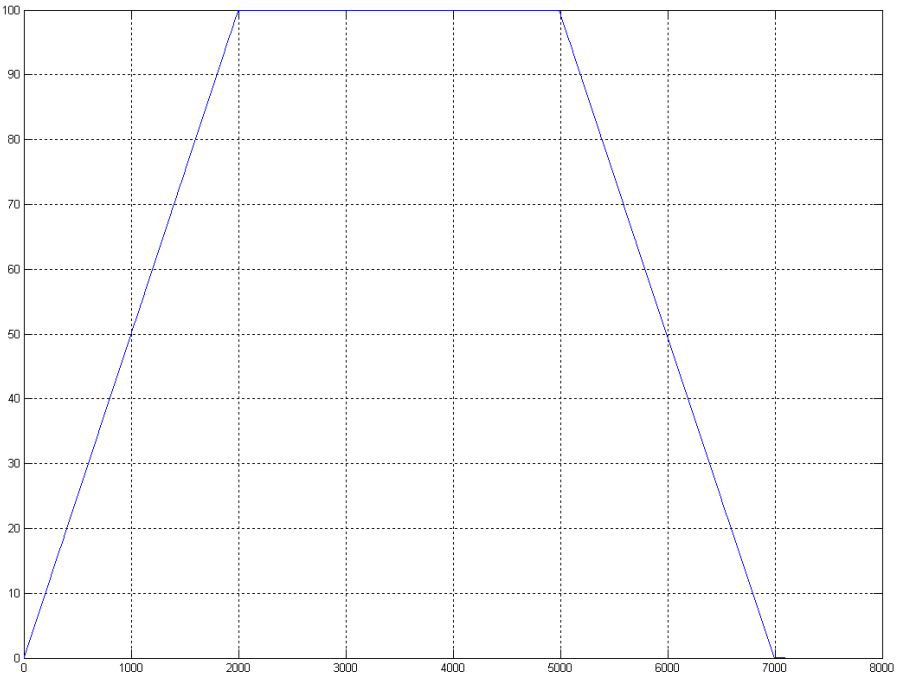


图 4-17 进行了前瞻预处理后的合成速度曲线

7. 刀向跟随功能

刀向跟随，就是在插补运动的过程中，使非插补轴随着插补运动的合成位移的变化而变化，从而实现在加工过程中，刀具始终处于合适的加工方向和位置的工艺。在本控制器的插补模块中有两条指令来实现该工艺：GT_BufMoveGT_BufMove 和 GT_BufGear。

(1) GT_BufMove

在插补运动过程中，如果需要坐标系以外的轴进行点位运动，则可以通过在缓存区中压入 GT_BufMove 指令来实现。

GT_BufMove 可以在插补运动的过程中插入模态和非模态的点位运动。**模态指令**的意义是，在进行该点位运动时，后续的插补缓存区中的指令将会被暂停执行，直到该指令执行完毕后，才执行下一条指令；**非模态指令**的意义是，该指令启动了一个轴的点位运动后，立即取下一条缓存区中的指令执行，不会等待点位运动的结束。

该指令的第二个参数是需要进行点位运动的轴号。

注意	需要进行点位运动的轴必须是坐标系外的轴，该轴的运动模式必须是点位运动，且该轴必须处于静止状态，否则该指令不能正常执行。

该指令的第三个参数是点位运动的目标位置，该位置值是相对于机床原点的绝对位置。该指令的第四个参数是点位运动的目标速度，该值必须为正值。该指令的第五个参数是点位运动的加速度，该值必须为正值。该指令的第六个参数表示该点位运动是模态的还是非模态的。

例程 4-6 刀向跟随功能 GT_BufMove

假设一个机床加工环境，有一个 XY 的二维坐标系，和一个不在坐标系内的轴 A。在 XY 坐标系内，刀具先从(0, 0)运动到(200000, 200000)（第 1 段轨迹）。然后，在 XY 方向从(200000, 200000)运动到(200000, 0)的同时，在 A 轴方向以 30 pulse/ms 的速度运动到 50000pulse 的位置（第 2 段轨迹）。这里用到 GT_BufMove 的非模态方式。然后，在 A 轴方向以 30 pulse/ms 的速度运动 100000 的位置。等到 A 轴方向运动完成，在 XY 坐标内画一个圆心为原点，半径 200000，位于三四象限的半圆弧（第 3 段轨迹）。这里用到 GT_BufMove 的模态方式，刀向跟随功能 GT_BufMove 的运动轨迹如图 4-18 所示。

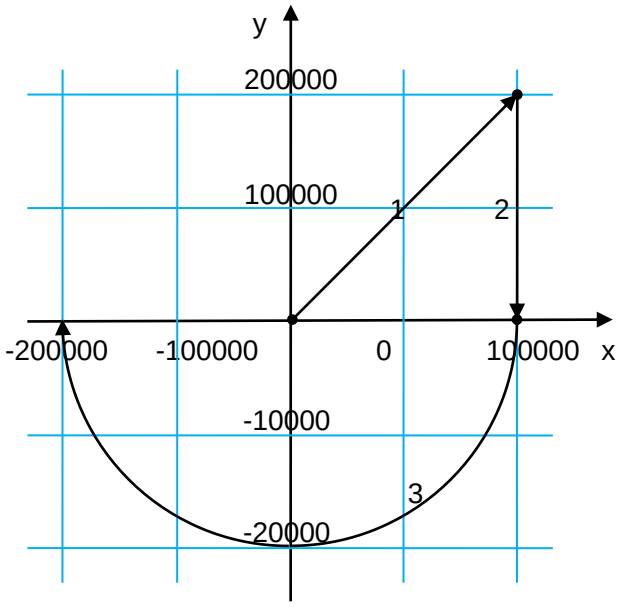


图 4-18 刀向跟随功能 GT_BufMove 的运动轨迹

```
…… …… ……
rtn: INT;
run:INT;
segment:DINT;
(* @END_DECLARATION := '0' *)
```

```
(*清除坐标系1的FIFO0中的数据*)
rtn:= GT_CrdClear(1, 0);
(*向FIFO0缓存区写入一段直线插补数据*)
rtn:= GT_LnXY(
    1,                  (*该插补段的坐标系是坐标系1*)
    200000, 200000,     (*该插补段的终点坐标(200000, 200000)*)
    100,                (*该插补段的目标速度：100pulse/ms*)
    0.1,                (*插补段的加速度：0.1pulse/ms^2*)
    0,                  (*终点速度为0*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向FIFO0缓存区写入一段非模态点位运动数据*)
rtn:= GT_BufMove(
    1,                  (*该插补段的坐标系是坐标系1*)
    4,                  (*点位运动的轴号：第4轴*)
    50000,              (*点位运动的目标位置：50000 pulse*)
    30,                 (*点位运动的目标速度：30 pulse/ms*)
    0.1,                (*点位运动的目标加速度：0.1 pulse/(ms*ms)*)
    0,                  (*该点位运动是非模态指令*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向FIFO0缓存区写入一段直线插补数据*)
rtn:= GT_LnXY(1, 200000, 0, 100, 0.1, 0, 0);    (*直线插补指令*)
(*向FIFO0缓存区写入一段模态点位运动数据*)
rtn:= GT_BufMove(
    1,                  (*该插补段的坐标系是坐标系1*)
    4,                  (*点位运动的轴号：第4轴*)
    100000,             (*点位运动的目标位置：100000 pulse*)
    30,                 (*点位运动的目标速度：30 pulse/ms*)
    0.1,                (*点位运动的目标加速度：0.1 pulse/(ms*ms)*)
    1,                  (*该点位运动是模态指令*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向缓存区写入一段圆弧插补数据，该段数据是以圆心描述方法描述了一个半圆*)
rtn:= GT_ArcXYC(
    1,                  (*坐标系是坐标系 1*)
    -200000, 0,         (*该圆弧的终点坐标(-200000, 0)*)
    -200000, 0,         (*圆弧插补的圆心相对于起点位置的偏移量(-200000, 0)*)
    0,                  (*该圆弧是顺时针圆弧*)
    100,                (*该插补段的目标速度：100pulse/ms*)
    0.1,                (*该插补段的加速度：0.1pulse/ms^2*)
    0,                  (*终点速度为0*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

(*查询坐标系运动状态*)
rtn:= GT_CrdStatus(1, ADR(run), ADR(segment), 0);
…… …… ……
```

例程插补合成速度和点位速度的运行结果如图 4-19 所示。

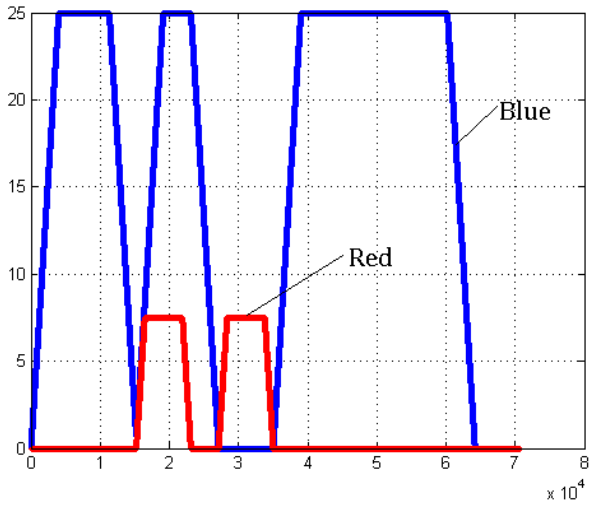


图 4-19 插补缓存区内的点位运动速度图

其中蓝色为插补运动的合成速度，红色为点位运动轴的速度值，可以看出第一个点位运动是非模态指令，与插补运动同时运动，而第二个点位运动是模态指令，会阻塞插补运动，只有点位运动结束之后，才能进行插补运动。

注意	在使用插补缓存区中的点位运动功能时，需要注意以下内容： 1. 点位运动的目标位置是相对于机床原点的绝对位置。 2. 如果在上一次的缓存区点位运动没有运动完成，又发送了新的点位运动，则会按照新的点位运动指令进行规划，即可以在插补缓存区中修改点位运动的目标位置和目标速度。 3. 如果在运动过程中停止插补缓存区的运动，则点位运动将不会停止，如果需要停止点位运动，则需要调用 GT_Stop 指令停止响应轴的运动。恢复缓存区运动时，用户需自行保证之前点位运动的轴在合适的位置上。 4. 当在模态点位运动的过程中，进行点位运动的轴由于触发限位等异常原因停止时，插补缓存区将不会再继续运行，此时用户需排查异常情况，重新设置相应参数，使系统正常后才可以工作。

(2) GT_BufGear

在插补过程中，需要坐标系外某一轴跟随坐标系插补运动的时候，可以通过在缓存区中压入 GT_BufGear 指令来实现，该指令的第二个参数是需要进行跟随运动的轴号。

注意	需要进行跟随运动的轴不能是坐标系中的轴；如果在发送跟随指令 GT_BufGear 时该轴正在运动，该指令将不能正常执行。

这里需要注意的是，该指令的第三个参数是跟随运动的位移量，该位移量是相对值，即下一段插补段运动过程中，跟随轴需要运动的位移量。使用的具体例程如下：

注意	GT_BufMove 中的位置参数是相对于机床原点的绝对位置，而 GT_BufGear 中的位置参数是相对位置。

例程 4-7 刀向跟随功能 GT_BufGear

假设一个机床加工环境，有一个 XY 的二维坐标系，和一个不在坐标系内的轴 A。在 XY 坐标系内，刀具先从(0, 0)运动到(200000, 200000)（第 1 段轨迹）。然后，在 XY 方向从(200000, 200000)运动到

(200000, 0)的同时，在 A 轴方向运动到 50000pulse 的位置，两者将同时到达各自指定的规划位置（第 2 段轨迹）。然后，在 XY 坐标内画一个圆心为原点，半径 200000，位于三四象限的半圆弧，同时在 A 轴方向运动 100000 的位置，两者将同时到达各自指定的规划位置（第 3 段轨迹），刀向跟随功能 GT_BufGear 的运动轨迹如图 4-20 所示。

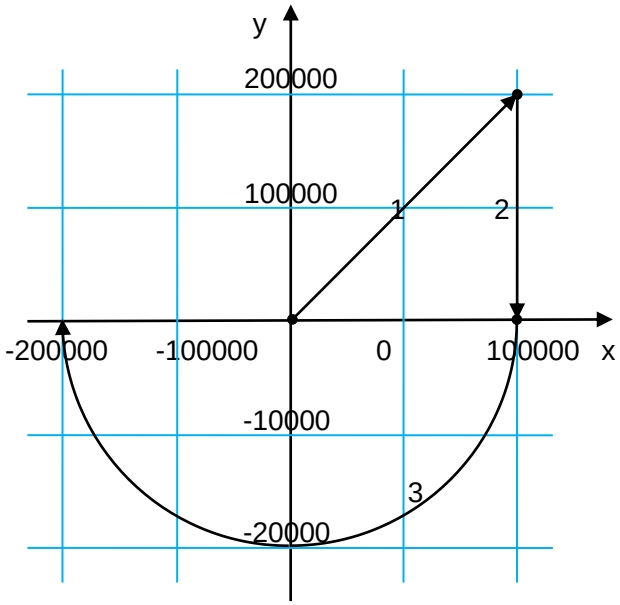


图 4-20 刀向跟随功能 GT_BufGear 的运动轨迹

```
…… …… ……
rtn: INT;
run:INT;
segment:DINT;
(* @END_DECLARATION := '0' *)
(*清除坐标系1的FIFO0中的数据*)
rtn:= GT_CrdClear(1, 0);
(*向FIFO0缓存区写入一段直线插补数据*)
rtn:= GT_LnXY(
    1,                  (*该插补段的坐标系是坐标系1*)
    200000, 200000,     (*该插补段的终点坐标(200000, 200000)*)
    100,                (*该插补段的目标速度：100pulse/ms**)
    0.1,                (*插补段的加速度：0.1pulse/ms^2*)
    0,                  (*终点速度为0*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向FIFO0缓存区写入一段跟随运动数据，GT_BufGear()指令需要在所要跟随的插补段前*)
rtn:= GT_BufGear(
    1,                  (*该插补段的坐标系是坐标系1*)
    4,                  (*跟随运动的轴号：第4轴*)
    50000,              (*跟随运动的位移量：50000 pulse。这里的位置参数是相对位置*)
    0);                 (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向FIFO0缓存区写入一段直线插补数据*)
rtn:= GT_LnXY(1, 200000, 0, 100, 0.1, 0, 0);
```

```
(*向FIFO0缓存区写入一段跟随运动数据*)
rtn:= GT_BufGear(
    1,              (*该插补段的坐标系是坐标系1*)
    4,              (*跟随运动的轴号：第4轴*)
    50000,          (*跟随运动的位移量：50000 pulse。这里的位置参数是相对位置*)
    0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*向缓存区写入一段圆弧插补数据，该段数据是以圆心描述方法描述了一个半圆*)
rtn:= GT_ArcXYC(
    1,              (*坐标系是坐标系 1*)
    -200000, 0,     (*该圆弧的终点坐标(-200000, 0)*)
    -200000, 0,     (*圆弧插补的圆心相对于起点位置的偏移量(-200000, 0)*)
    0,              (*该圆弧是顺时针圆弧*)
    100,            (*该插补段的目标速度：100pulse/ms*)
    0.1,            (*该插补段的加速度：0.1pulse/ms^2*)
    0,              (*终点速度为0*)
    0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)

(*查询坐标系运动状态*)
rtn:= GT_CrdStatus(1, ADR(run), ADR(segment), 0);
…… …… ……
```

例程的运行结果如图 4-21 所示。

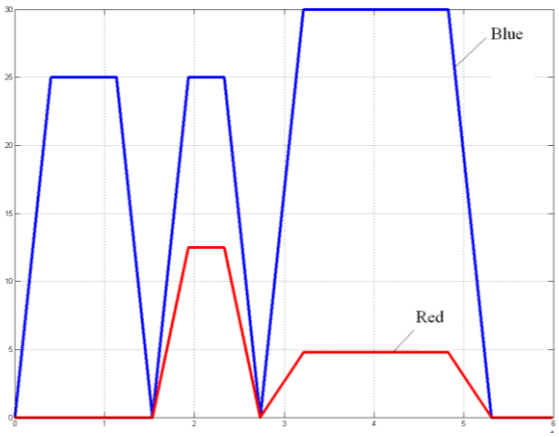


图 4-21 插补缓存区内的跟随运动速度图

其中蓝色为插补运动的合成速度，红色为跟随运动轴的速度值，跟随轴的速度跟随插补运动的合成速度的变化而变化。

在使用插补缓存区中的跟随运动功能时，需要注意以下内容：

1) GT_BufGear 指令需要在所要跟随的插补段前，不要间隔其他种类的指令，可以同时调用多个 GT_BufGear 指令来实现多个轴跟随插补运动。

2) 当暂停坐标系运动时，插补的合成速度减速到 0，跟随轴的速度也会为 0，如果希望重新恢复坐标系运动时，跟随轴仍能够实现跟随，不要调用 GT_Stop 指令停止跟随轴的运动，否则重新启动插补缓存区的运动时，跟随轴将无法完成正在进行的跟随运动。

例程 4-8 刀向跟随功能——实际工件加工

假设机床加工过程中，机床需要加工一个工件，XY 平面的尺寸如图 4-22 所示，工件的厚度为 500（单位均为：pulse），采用的工艺是用砂轮磨削工件外轮廓，加工时不但要调整刀具的 Z 方向高度，同时需要调整砂轮的 C 向角度（假设砂轮旋转一周的脉冲总数是 10000）。在加工工艺中，首先为避免撞到工件，需要将砂轮抬到一定的高度:2000（假设安全高度为 2000），从原点空走到 A(6000, 6000)位置，之后调整砂轮逆时针转 45°（对应为 1250 pulse），使之与工件的加工切向方向一致，然后降低砂轮高度到加工位置:-100。接着，沿直线方向加工到 B(6000, 12000)位置，从 B 到 C 为一圆弧，需要实时更新砂轮的方向，使之保持与工件的加工切向方向一致，所以需要调用 GT_BufGear 指令，跟随的位移量是 2500（90°）。随之后面的加工与之前类似，直至到加工点 F(36000, 6000)位置，此时需要抬刀使砂轮改变 90°，使之与 FA 段的加工方向一致，之后再放下砂轮至加工位置，直至加工完工件。

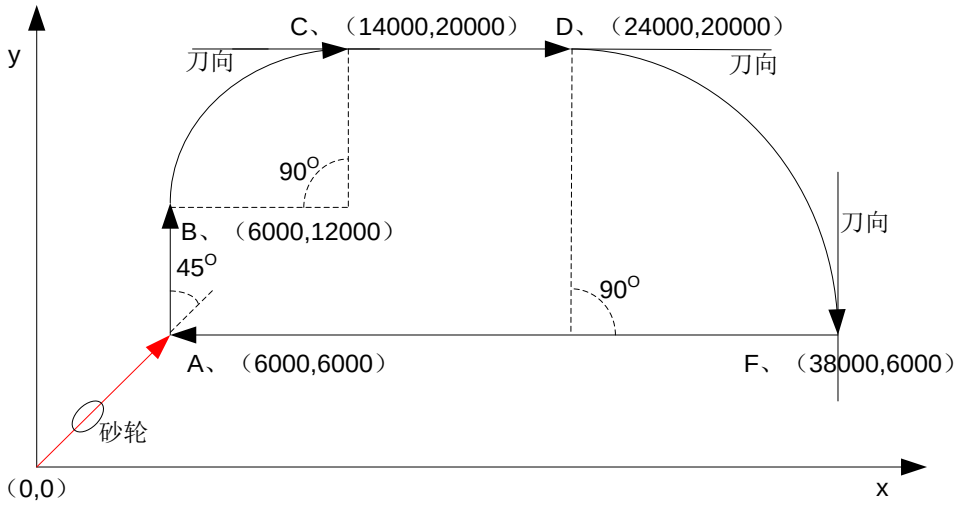


图 4-22 刀向跟随功能之工件尺寸和刀运动轨迹

假设 1 和 2 轴为 XY 轴，3 轴为 Z 轴，4 轴为 C 轴，并且 C 向初始角度为 0°，加工程序代码实现如下：

```
…… …… ……
rtn: INT;
run:INT;
segment:DINT;
(* @END_DECLARATION := '0' *)
(*清除坐标系1的FIFO0中的数据*)
rtn:= GT_CrdClear(1, 0);
(*向FIFO0缓存区写入一段直线插补数据*)
rtn:= GT_BufMove(
    1,          (*该插补段的坐标系是坐标系1*)
    3,          (*点位运动的轴号：第3轴*)
    500,        (*点位运动的目标位置：500 pulse*)
    10,         (*点位运动的目标速度：10 pulse/ms*)
    0.1,        (*点位运动的目标加速度：0.1 pulse/(ms*ms)*)
    1,          (*该点位运动是模态指令，等砂轮抬高后才执行下面的指令*)
    0);         (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
(*直线插补指令，到达A点*)
rtn:= GT_LnXY(1, 6000, 6000, 50, 0.1, 0, 0);
rtn:= GT_BufMove(
    1,          (*该插补段的坐标系是坐标系1*)
```

```
        4,              (*点位运动的轴号：第4轴*)
        -1250,          (*使其逆时针旋转45°与BA方向一致*)
        10,             (*点位运动的目标速度：10 pulse/ms*)
        0.1,            (*点位运动的目标加速度：0.1 pulse/(ms*ms)*)
        1,              (*该点位运动是模态指令，等角度到位才后才执行下面的指令*)
        0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
    (*模态指令，将砂轮放下到加工高度位置*)
    rtn:= GT_BufMove(1, 3, -100, 10, 0.1, 1, 0);
    (*直线插补指令，到达B点*)
    rtn:= GT_LnXY(1, 6000, 12000, 50, 0.1, 0, 0);
    rtn:= GT_BufGear(
        1,              (*该插补段的坐标系是坐标系1*)
        4,              (*跟随运动的轴号：第4轴*)
        2500,           (*跟随运动的位移量：2500 pulse（相应为90°）*)
        0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
    (*到达C点位置*)
    rtn:= GT_ArcXYR(
        1,              (*坐标系是坐标系1*)
        0, 14000,       (*该圆弧的终点坐标(0, 200000)*)
        8000,           (*半径：8000pulse*)
        0,              (*该圆弧是顺时针圆弧*)
        50,             (*该插补段的目标速度：50pulse/ms*)
        0.1,            (*该插补段的加速度：0.1pulse/ms^2*)
        0,              (*终点速度为0*)
        0);             (*向坐标系1的FIFO0缓存区传递该直线插补数据*)
    (*直线插补指令，到达D点*)
    rtn:= GT_LnXY(1, 24000, 20000, 50, 0.1, 0, 0);
    (*跟随运动的位移量：2500 pulse（相应为90°）*)
    rtn:= GT_BufGear(1, 4, 2500, 0);
    (*到达F点*)
    rtn:= GT_ArcXYR(1, 38000, 6000, 14000, 0, 50, 0.1, 0, 0);
    (*模态指令，将砂轮抬高到安全高度位置*)
    rtn:= GT_BufMove(1, 3, 500, 10, 0.1, 1, 0);
    (*模态指令，将砂轮抬旋转至与FA方向一致*)
    (*该位置相对初始位置为225°，即目标位置6250*)
    rtn:= GT_BufMove(1, 4, 6250, 10, 0.1, 1, 0);
    (*模态指令，将砂轮放下到加工高度位置*)
    rtn:= GT_BufMove(1, 3, -100, 10, 0.1, 1, 0);
    (*直线插补指令，到达A点*)
    rtn:= GT_LnXY(1, 6000, 6000, 50, 0.1, 0, 0);
    (*启动运动*)
    rtn:= GT_CrdStart(1, 0);

    (*查询坐标系运动状态*)
    rtn:= GT_CrdStatus(1, ADR(run), ADR(segment), 0);
```

…… …… ……

4.3 PVT 模式

4.3.1 指令列表

表 4-2 PVT 指令列表

指令	说明	页码
GT_PrfPvt	设置指定轴为 PVT 模式	68
GT_SetPvtLoop	设置循环次数	75
GT_GetPvtLoop	查询循环次数	64
GT_PvtTable	向指定数据表传送数据，采用 PVT 描述方式	70
GT_PvtTableComplete	向指定数据表传送数据，采用 Complete 描述方式	71
GT_PvtTablePercent	向指定数据表传送数据，采用 Percent 描述方式	72
GT_PvtPercentCalculate	计算 Percent 描述方式下各数据点的速度	69
GT_PvtTableContinuous	向指定数据表传送数据，采用 Continuous 描述方式	71
GT_PvtContinuousCalculate	计算 Continuous 描述方式下各数据点的时间	68
GT_PvtTableSelect	选择数据表	72
GT_PvtStart	启动运动	69
GT_PvtStatus	读取状态	70

4.3.2 重点说明

PVT 模式使用一系列数据点的“位置、速度、时间”参数来描述运动规律。位置、速度和时间满足如下函数关系：

$$p = at^3 + bt^2 + ct + d$$

$$v = \frac{dp}{dt} = 3at^2 + 2bt + c$$

如果给定相邻 2 个数据点的“位置、速度、时间”参数，可以得到如下方程组：

$$\begin{cases} at_1^3 + bt_1^2 + ct_1 + d = p_1 \\ 3at_1^2 + 2bt_1 + c = v_1 \\ at_2^3 + bt_2^2 + ct_2 + d = p_2 \\ 3at_2^2 + 2bt_2 + c = v_2 \end{cases}$$

求解该方程组，可以得到 a、b、c、d，因此相邻 2 个数据点的运动规律就可以确定下来。

运动控制器提供 32 个数据表存储数据点。每个数据表具有 1024 个存储空间。数据表和轴之间相互独

立，一个数据表可以供多个轴使用。

调用 GT_PvtTable、GT_PvtTableComplete、GT_PvtTablePercent 或 GT_PvtTableContinuous 指令向数据表中传递数据。这些指令会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

调用 GT_PvtTableSelect 指令选择数据表。可以在运动状态下切换数据表，但是不会立即切换。只有当前数据表执行完毕以后，才会切换到新的数据表。

调用 GT_PvtStart 启动运动。启动以后，各轴时间清 0。如果第一个数据点的时间为 0 则立即启动，否则会延时启动，延时时间等于第一个数据点的时间。

数据表可以循环执行，调用 GT_SetPvtLoop 设置循环次数，循环次数为 0 表示无限循环。当遍历完数据表以后，时间初始化为第一个数据点的时间，而不是 0。

假设有如下 4 个数据点，采用 PVT 方式进行描述。

表 4-3 PVT 方式描述的数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	1,000	0	0
P2	2,000	5,000	10
P3	3,000	15,000	10
P4	4,000	20,000	0

(1) 调用 GT_PrfPvt 将轴切换到 PVT 模式；

(2) 调用 GT_PvtTable 将 4 个数据点传递到数据表；

(3) 调用 GT_SetPvtLoop 设置为循环执行；

(4) 调用 GT_PvtStart 启动运动。

由于 P1 的时间为 1000 毫秒，因此调用 GT_PvtStart 以后延时 1000 毫秒启动。由于是循环执行，到达 P4 以后返回到 P1，速度曲线如图 4-23 所示。

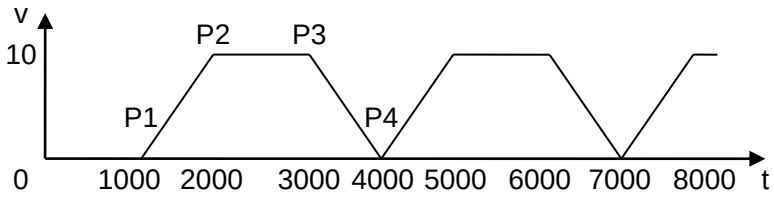


图 4-23 循环执行数据表

PVT 模式有 4 种方式描述运动规律，PVT、Complete、Percent 和 Continuous，下面对此进行详细说明。

1. PVT 描述方式

PVT 描述方式直接定义各数据点的“位置、速度、时间”。相邻 2 个数据点之间，运动控制器使用 3 次多项式对位置进行插值，使用 2 次多项式对速度进行插值。因此当给出各数据点“位置、速度、时间”参数以后，相应的运动规律也就确定下来。例如表 4-4 所示的 4 组数据点，采用 PVT 描述方式。

表 4-4 PVT 描述方式下的四组数据点

数据组	数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
1	P1	0	0	0
	P2	1, 000	5, 000	10
	P3	2, 000	15, 000	10
	P4	3, 000	20, 000	0
2	P1	0	0	0
	P2	1, 000	5, 000	9
	P3	2, 000	15, 000	9
	P4	3, 000	20, 000	0
3	P1	0	0	0
	P2	1, 000	5, 000	7.5
	P3	2, 333	15, 000	7.5
	P4	3, 333	20, 000	0
4	P1	0	0	0
	P2	750	1, 667	6.6669
	P3	2, 250	18, 333	6.6669
	P4	3, 000	20, 000	0

这 4 组数据点对应的运动规律如图 4-24 所示。

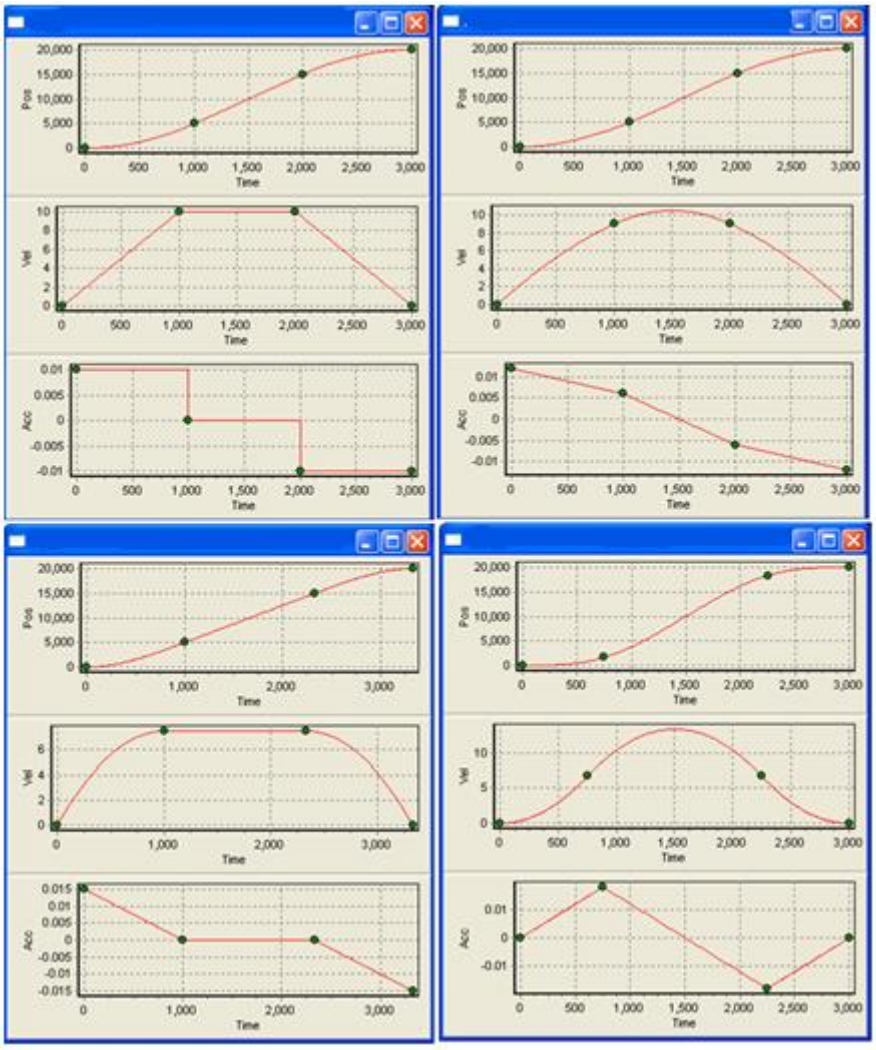


图 4-24 合理的 PVT 描述方式运动规律

可以看出，PVT 描述方式非常灵活。给定数据点的“位置、速度、时间”参数，就能够得到相应的运动规律。需要注意的是，数据点参数需要仔细设计，否则难以得到理想的运动规律。例如表 4-5 所示的 2 组数据点参数不合理，得到的速度曲线不够平滑。

表 4-5 两组不合理的 PVT 描述方式下的数据点

数据组	数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
1	P1	0	0	0
	P2	1, 000	5, 000	15
	P3	2, 000	15, 000	15
	P4	3, 000	20, 000	0
2	P1	0	0	0
	P2	1, 000	5, 000	5
	P3	2, 000	15, 000	5
	P4	3, 000	20, 000	0

这 2 组数据点对应的运动规律如图 4-25 所示。

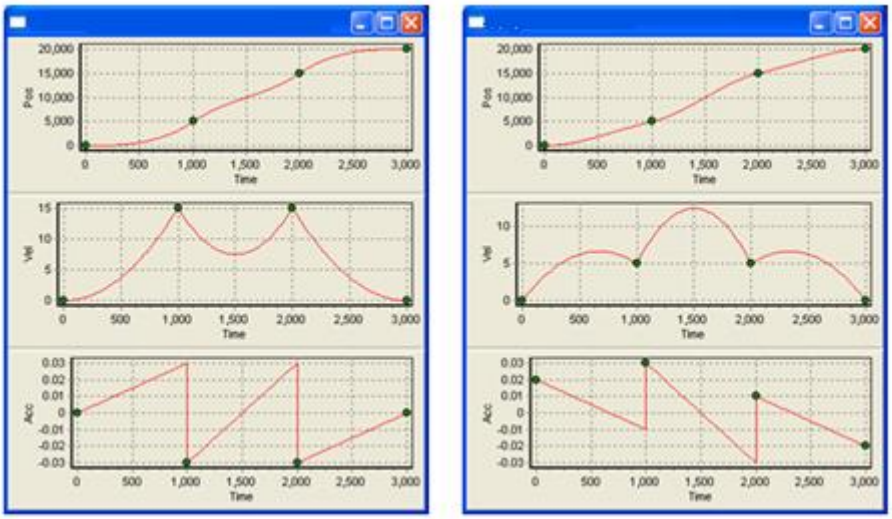


图 4-25 不合理的 PV 描述方式运动规律

2. Complete 描述方式

Complete 描述方式定义各数据点的“位置、时间”，以及起点速度和终点速度。Complete 方式只定义了起点速度和终点速度。运动控制器根据各数据点的“位置、时间”参数计算中间各点的速度，确保各数据点速度连续和加速度连续。例如表 4-6 所示的这组数据点，采用 Complete 描述方式，可以轻松得到光滑的速度曲线。

表 4-6 Complete 描述方式的一组数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	不指定
P3	2,000	15,000	不指定
P4	3,000	20,000	0

这组数据点对应的运动规律如图 4-26 所示。

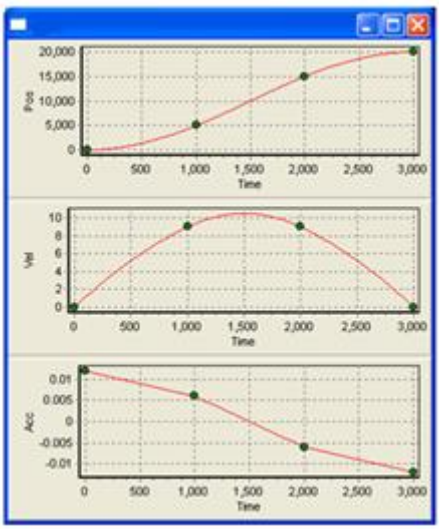


图 4-26 Complete 描述方式运动规律

Complete 适合描述光滑的速度曲线，例如三角函数等。假设位置和时间之间的关系由函数 $P=50000\sin^2(\pi/2000*t)$确定。在一个函数周期[0,2000]内取 5 个时间点计算相应的位置，如表 4-7 所示。

表 4-7 Complete 方式描述三角函数的数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	500	25,000	不指定
P3	1,000	50,000	不指定
P4	1,500	25,000	不指定
P5	2,000	0	0

这组数据点对应的运动规律如图 4-27 所示。

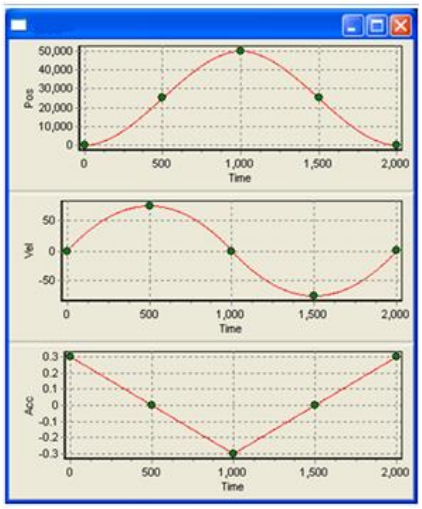


图 4-27 Complete 描述方式运动规律

增加数据点可以减小与函数 $P=50000\sin^2(\pi/2000*t)$的逼近误差。图 4-28 给出了数据点数为 5、10、50 时的位置误差。

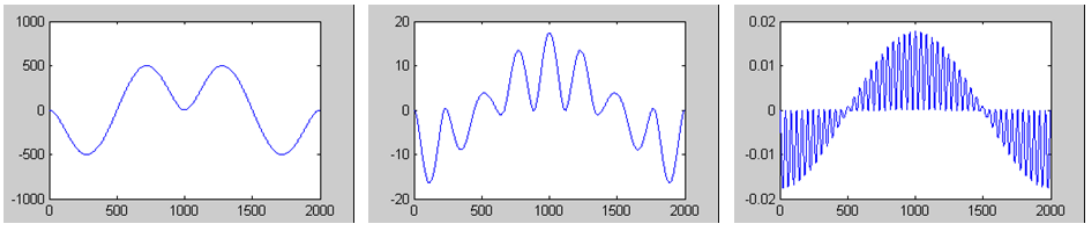


图 4-28 Complete 方式下数据点数分别为 5、10、50 时的位置误差

3. Percent 描述方式

Percent 描述方式定义各数据点的“位置、时间、百分比”，以及起点速度。Percent 描述方式能够精确定义加速段、匀速段、减速段的位移、速度和时间。Percent 描述方式假设相邻 2 个数据点之间速度为线性变化，利用起点速度以及各数据点的“位置、时间”参数，通过如下递推公式可以计算出各数据点的速度。

$$v_{i+1}=\frac{2(p_{i+1}-p_i)}{t_{i+1}-t_i}-v_i$$

因此指定了各数据点的“位置、时间”参数以后，各数据点的速度实际上也就已经确定下来。通过“百分比”参数可以调整速度曲线的光滑性。数据点的百分比参数是指“相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比”。以图 4-29 为例来进行说明。

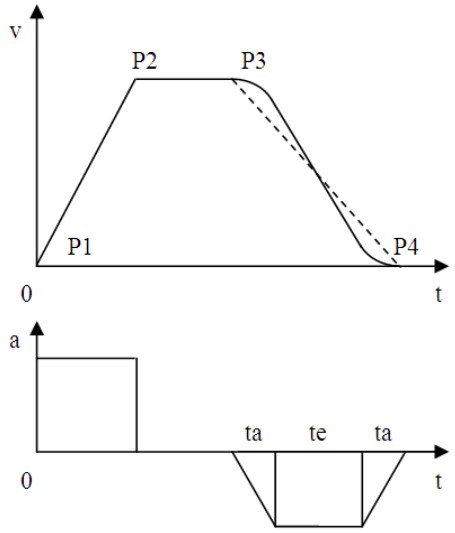


图 4-29 Percent 描述方式下的百分比定义

数据点 P1 和 P2 之间加速度不变，因此数据点 P1 的百分比为 0。数据点 P2 和 P3 之间加速度不变，因此数据点 P2 的百分比为 0。数据点 P3 和 P4 之间加速度变化时间为 2ta，运动时间为 2ta+te，因此数据点 P3 的百分比为 2ta/(2ta+te)*100%。

调整百分比参数，不会影响数据点的“位置、时间参数”。以上图为例，当数据点 P3 的百分比为 0 时，数据点 P3 和 P4 之间的速度曲线为虚线；当数据点 P3 的百分比不为 0 时，数据点 P3 和 P4 之间的速度曲线为实线。

例如表 4-8 所示的这组数据点，采用 Percent 描述方式。

表 4-8 Percent 描述方式下的数据点

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	100	不指定
P4	3,000	20,000	0	不指定

这组数据点对应的运动规律如图 4-30 所示。

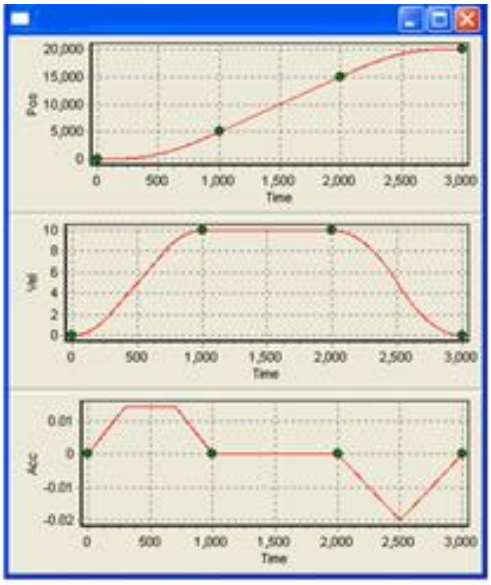


图 4-30 Percent 描述方式下的运动规律

4. Continuous 描述方式

Continuous 描述方式定义各数据点的“位置、速度、最大速度、加速度、减速度、百分比”。不用指定数据点的时间。运动控制器根据数据点参数，自动将相邻 2 个数据点之间拆分为加速段、匀速段和减速段。

数据点 P_i 的最大速度是指从数据点 P_i 到数据点 P_{i+1} 之间的速度上限。数据点 P_i 的加速度是指从数据点 P_i 到数据点 P_{i+1} 之间的加速段所使用的加速度。数据点 P_i 的减速度是指从数据点 P_i 到数据点 P_{i+1} 之间的减速段所使用的减速度。数据点 P_i 的百分比是指从数据点 P_i 到数据点 P_{i+1} 之间的加减速段中，加速度变化时间占速度变化时间的百分比。

相邻 2 个数据点之间能够拆分出来的段数和这 2 个数据点的参数有关，图 4-31 示例了一些可能的分段情况。

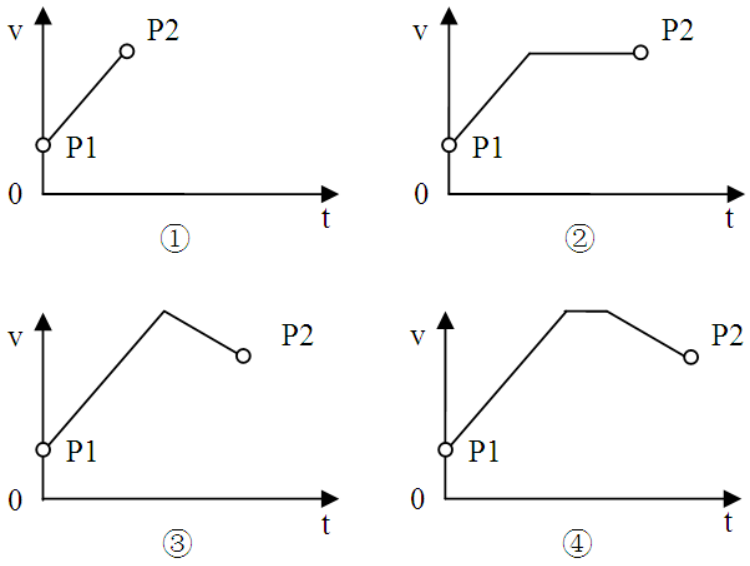


图 4-31 Continuous 描述方式

例如表 4-9 所示的这两组数据点，采用 Continuous 描述方式。

表 4-9 Continuous 描述方式下的数据点

数据组	数据点	位置	速度	最大速度	加速度	减速度	百分比
1	P1	0	0	10	0.01	0.01	60
	P2	20,000	0	10	0.01	0.01	0
2	P1	0	0	10	0.01	0.01	60
	P2	19,800	2	2	0.02	0.02	0
	P3	21,800	0	2	0.02	0.02	0

这 2 组数据点对应的运动规律如图 4-32 所示。

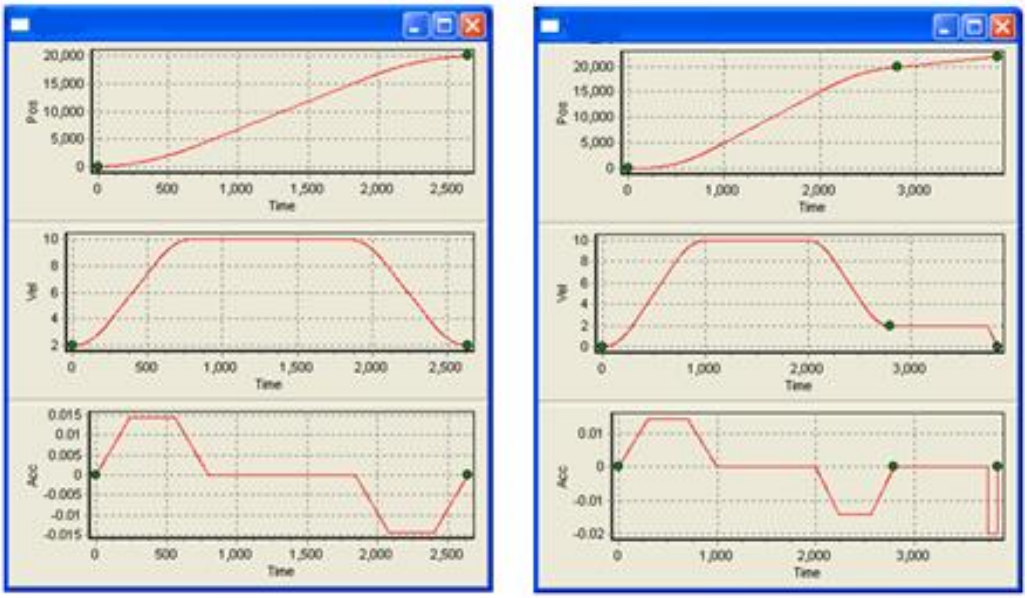


图 4-32 Continuous 描述方式下的运动规律

4.3.3 例程

1. PVT 描述方式

如图 4-33 所示，整个速度曲线由 5 段组成，并且带有起跳速度。第一段速度增大，加速度保持不变；第二段速度增大，加速度减小；第三段速度不变，加速度为 0；第四段速度减小，加速度增大；第五段速度减小，加速度不变。

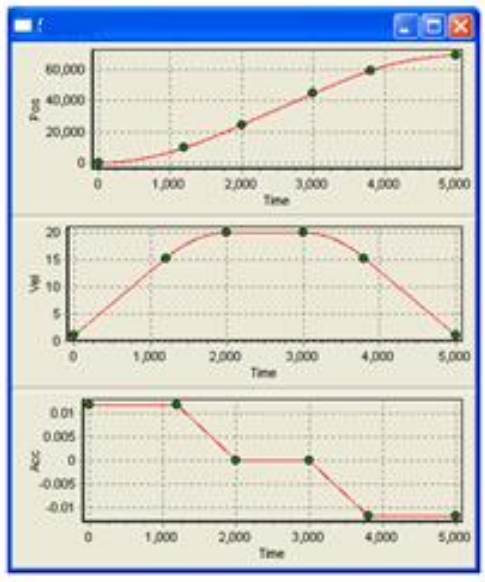


图 4-33 PVT 例程描述方式下的运动规律

可以满足上述要求的一组数据表如表 4-10 所示。

表 4-10 PVT 例程描述方式下的数据点

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	1
P2	1,200	9,750	15.25
P3	2,000	24,483	20
P4	3,000	44,483	20
P5	3,800	59,216	15.25
P6	5,000	68,966	1

例程 4-9 PVT 描述方式

```
PROGRAM PLC_PRG
VAR
    xInitDone:BOOL:= FALSE;
    rtn:INT;
    mask:DINT;
    xStart: BOOL:= FALSE;
    (*X轴的数据点参数*)
    ttime:ARRAY [0..5] OF LREAL:= 0,1200,2000,3000,3800,5000;
    pos: ARRAY [0..5] OF LREAL:= 0,9750,24483,44483,59216,68966;
    vel: ARRAY [0..5] OF LREAL:= 1,15.25,20,20,15.25,1;
    prfVel, prfPos, t:LREAL;
    tableId:INT;
END_VAR
VAR CONSTANT
    AXIS:INT:= 1;
    TABLE:INT:= 1;
END_VAR
```

```
(* @END_DECLARATION := '0' *)
IF NOT xInitDone THEN
    (*配置运动控制器*)
    (*注意：配置文件取消了各轴的报警和限位*)
    rtn:= GT_LoadConfig('test.cfg);
    (*清除各轴的报警和限位*)
    rtn:= GT_ClrSts(1,8);
    (*伺服使能*)
    rtn:= GT_AxisOn(AXIS);
    xInitDone:= TRUE;
END_IF
IF xStart THEN
    (*设置为PVT模式*)
    rtn:= GT_PrfPvt(AXIS);
    (*发送数据*)
    rtn:= GT_PvtTable(TABLE, 6, ADR(ttime), ADR(pos), ADR(vel));
    (*选择数据表*)
    rtn:= GT_PvtTableSelect(AXIS, TABLE);
    mask:= SHL(WORD#1,AXIS-1);
    rtn:= GT_PvtStart(mask);
    xStart:= FALSE;
END_IF
(*读取数据表和运动时间*)
rtn:= GT_PvtStatus(AXIS, ADR(tableId), ADR(t),1);
(*读取规划速度*)
rtn:= GT_GetPrfVel(AXIS, ADR(prfVel),1,0);
(*读取规划位置*)
rtn:= GT_GetPrfPos(AXIS, ADR(prfPos),1,0);
END_PROGRAM
```

2. Complete 描述方式

假设位置和时间之间的关系由函数 $P=40000\sin^2(\pi/2000*t)$确定。要求启动以后能够循环运动，Complete 描述方式下的速度曲线如图 4-34 所示。

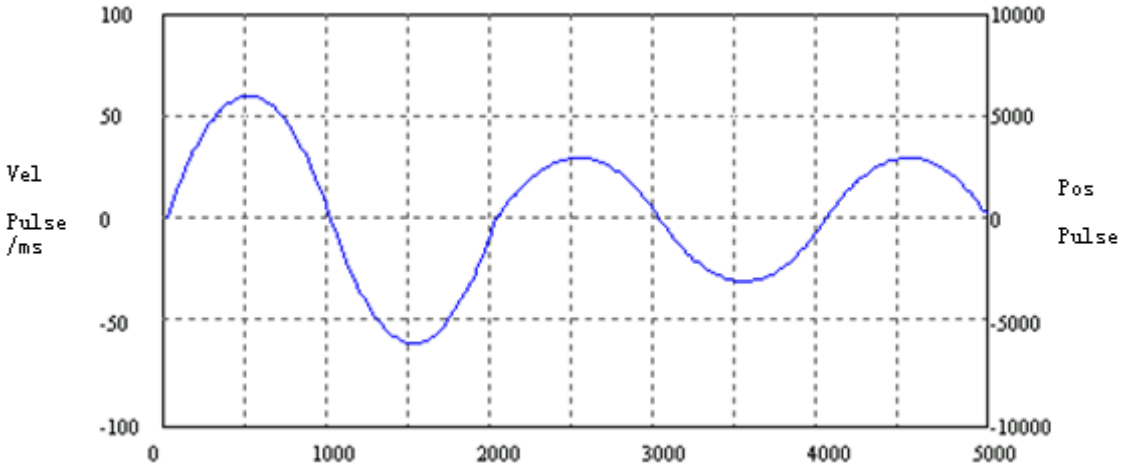


图 4-34 Complete 描述方式下的速度曲线

例程 4-10 Complete 描述方式

```
FUNCTION Calculate : INT
VAR_INPUT
    amplitude:LREAL;
    n:INT;
    pTime:POINTER TO LREAL;
    pPos:POINTER TO LREAL;
END_VAR
VAR
    i:DWORD;
    p1,p2: POINTER TO LREAL;
END_VAR
VAR CONSTANT
    PI:LREAL:= 3.1415926;
END_VAR
(* @END_DECLARATION := '0' *)
FOR i:= 0 TO (n-1) DO
    p1:= pTime + i;
    p2:= pPos + i;
    p2^:= amplitude*SIN(PI/2000*p1^)*SIN(PI/2000*p1^);
END_FOR
END_FUNCTION

PROGRAM PLC_PRG
VAR
    xInitDone:BOOL:= FALSE;
    rtn:INT;
    mask:DINT;
    xStart:BOOL:= FALSE;
    (*X轴的数据点参数*)
    ttime:ARRAY [0..4] OF LREAL:= 0,500,1000,1500,2000;
```

```
        pos:ARRAY [0..4] OF LREAL;
        a,b,c:ARRAY [0..4] OF LREAL;
        prfVel, prfPos, t:LREAL;
        tableId:INT;
        amplitude:LREAL:= 40000;
    END_VAR
    VAR CONSTANT
        AXIS:INT:= 1;
        TABLE:INT:= 1;
    END_VAR
    (* @END_DECLARATION := '0' *)
    IF NOT xInitDone THEN
        (*配置运动控制器*)
        (*注意：配置文件取消了各轴的报警和限位*)
        rtn:= GT_LoadConfig('test.cfg);
        (*清除各轴的报警和限位*)
        rtn:= GT_ClrSts(1,8);
        (*伺服使能*)
        rtn:= GT_AxisOn(AXIS);
        xInitDone:= TRUE;
    END_IF
    IF xStart THEN
        (*设置为PVT模式*)
        rtn:= GT_PrfPvt(AXIS);
        Calculate(amplitude, 5, ADR(ttime), ADR(pos));
        (*发送数据*)
        rtn:= GT_PvtTableComplete(TABLE, 5, ADR(ttime), ADR(pos), ADR(a), ADR(b),
ADR(c),0,0);
        (*选择数据表*)
        rtn:= GT_PvtTableSelect(AXIS, TABLE);
        (*设置为循环执行*)
        rtn:= GT_SetPvtLoop(AXIS, 0);
        mask:= SHL(WORD#1,AXIS-1);
        rtn:= GT_PvtStart(mask);
        xStart:= FALSE;
    END_IF
    (*读取数据表和运动时间*)
    rtn:= GT_PvtStatus(AXIS, ADR(tableId), ADR(t),1);
    (*读取规划速度*)
    rtn:= GT_GetPrfVel(AXIS, ADR(prfVel),1,0);
    (*读取规划位置*)
    rtn:= GT_GetPrfPos(AXIS, ADR(prfPos),1,0);
    END_PROGRAM
```

3. Percent 描述方式

X 轴往复运动，Y 轴正向进给。X 轴加减速时 Y 轴开始进给，X 轴匀速运动时，Y 轴保持静止。X 轴和 Y 轴的运动规律如图 4-35 所示。

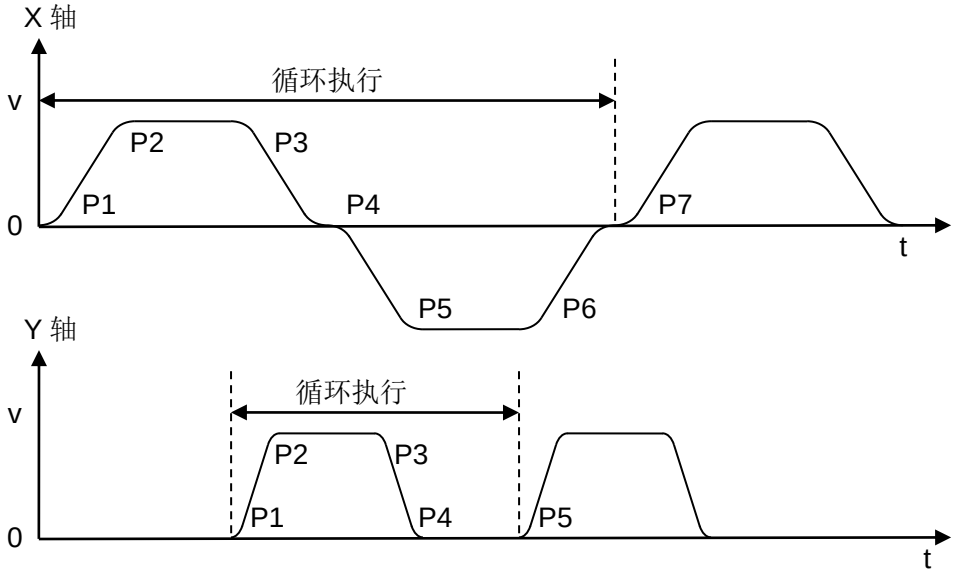


图 4-35 Percent 描述方式下 X 轴和 Y 轴的运动规律

X 轴取 7 个数据点，设置为循环模式。数据点参数如表 4-11 和表 4-12 所示。

表 4-11 Percent 描述方式下的数据点 1

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	60	不指定
P4	3,000	20,000	60	不指定
P5	4,000	15,000	0	不指定
P6	5,000	5,000	60	不指定
P7	6,000	0	0	不指定

根据 X 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

表 4-12 Percent 描述方式下的数据点 2

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	2(5000-0)/(1000-0)-0=10
P3	2,000	15,000	2(15000-5000)/(2000-1000)-10=10
P4	3,000	20,000	2(20000-15000)/(3000-2000)-10=0
P5	4,000	15,000	2(15000-20000)/(4000-3000)-0=-10
P6	5,000	5,000	2(5000-15000)/(5000-4000)-(-10)=-10
P7	6,000	0	2(0-5000)/(6000-5000)-(-10)=0

Y 轴取 5 个数据点，设置为循环模式。X 轴启动以后到达数据点 P3 时 Y 轴才启动，因此第 1 个数据点的时间设置为 2000 毫秒。当 Y 轴到达 P5 以后，返回到 P1 循环执行。数据点参数如表 4-13 和表 4-14 所示。

表 4-13 Percent 描述方式下的数据点 3

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	2,000	0	60	0
P2	2,500	2,500	0	不指定
P3	3,500	12,500	60	不指定
P4	4,000	15,000	0	不指定
P5	5,000	15,000	0	不指定

根据 Y 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

表 4-14 Percent 描述方式下的数据点 4

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	2,000	0	0
P2	2,500	2,500	2(2500-0)/(2500-2000)-0=10
P3	3,500	12,500	2(12500-2500)/(3500-2500)-10=10
P4	4,000	15,000	2(15000-12500)/(4000-3500)-10=0
P5	5,000	15,000	2(15000-15000)/(5000-4000)-0=0

X 轴循环 n 次，Y 轴需要循环 2n-1 次。图 4-36 是当 X 轴的循环次数为 2，Y 轴循环次数为 3 时的 XY 位置图。横轴是 X 轴的位置，纵轴是 Y 轴的位置。蓝线是实际的运动轨迹。

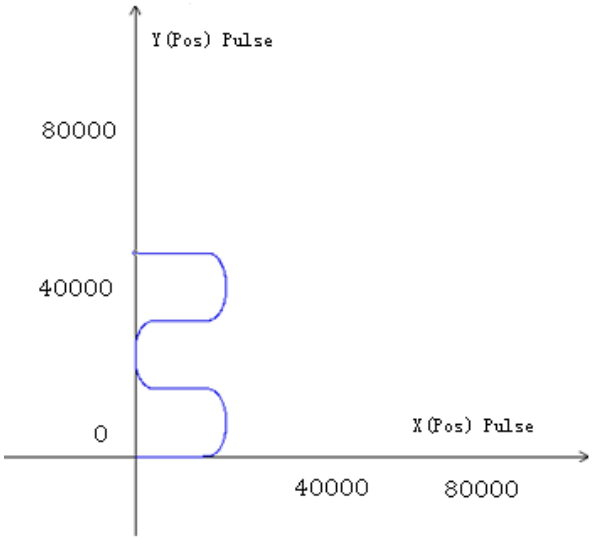


图 4-36 Percent 描述方式下的 X-Y 位置图

例程 4-11 Percent 描述方式

```
PROGRAM PLC_PRG
VAR
    xInitDone:BOOL:= FALSE;
    rtn:INT;
    mask:DINT;
    xStart:BOOL:= FALSE;
    (*X轴的数据点参数*)
    time_x:ARRAY [0..6] OF LREAL:= 0,1000,2000,3000,4000,5000,6000;
```

```
        pos_x:ARRAY [0..6] OF LREAL:= 0, 5000, 15000, 20000, 15000, 5000, 0;
        percent_x:ARRAY [0..6] OF LREAL:= 60, 0, 60, 60, 0, 60, 0;
        (*Y轴的数据点参数*)
        time_y:ARRAY [0..4] OF LREAL:= 2000,2500,3500,4000,5000;
        pos_y:ARRAY [0..4] OF LREAL:= 0, 2500, 12500, 15000, 15000;
        percent_y:ARRAY [0..4] OF LREAL:= 60, 0, 60, 0, 0;
        prfVel, prfPos, t: ARRAY [0..1] OF LREAL;
        tableId:ARRAY [0..1] OF INT;
    END_VAR
    VAR CONSTANT
        AXIS_X:INT:= 1;
        AXIS_Y:INT:= 2;
        TABLE_X:INT:= 1;
        TABLE_Y:INT:= 2;
        LOOP_COUNT:INT:= 2;
    END_VAR
    (* @END_DECLARATION := '0' *)
    IF NOT xInitDone THEN
        (*配置运动控制器*)
        (*注意：配置文件取消了各轴的报警和限位*)
        rtn:= GT_LoadConfig('test.cfg);
        (*清除各轴的报警和限位*)
        rtn:= GT_ClrSts(1, 8);
        (*伺服使能*)
        rtn:= GT_AxisOn(AXIS_X);
        rtn:= GT_AxisOn(AXIS_Y);
        xInitDone:= TRUE;
    END_IF
    IF xStart THEN
        (*设置为PVT模式*)
        rtn:= GT_PrfPvt(AXIS_X);
        rtn:= GT_PrfPvt(AXIS_Y);
        (*向数据表发送数据*)
        rtn:= GT_PvtTablePercent(TABLE_X, 7, ADR(time_x), ADR(pos_x), ADR(percent_x),0);
        rtn:= GT_PvtTablePercent(TABLE_Y, 5, ADR(time_y), ADR(pos_y), ADR(percent_y),0);
        (*选择数据表*)
        rtn:= GT_PvtTableSelect(AXIS_X, TABLE_X);
        rtn:= GT_PvtTableSelect(AXIS_Y, TABLE_Y);
        (*设置循环次数*)
        rtn:= GT_SetPvtLoop(AXIS_X, LOOP_COUNT);
        rtn:= GT_SetPvtLoop(AXIS_Y, 2*LOOP_COUNT-1);
        (*同时启动X轴和Y轴，由于Y轴的第1个数据点时间为2000ms，因此X轴启动2000ms以后，
    Y轴才开始运动*)
        mask:= SHL(WORD#1,AXIS_X-1) OR SHL(WORD#1,AXIS_Y-1);
        rtn:= GT_PvtStart(mask);
```

```
        xStart:= FALSE;
    END_IF
    (*读取数据表和运动时间*)
    rtn:= GT_PvtStatus(AXIS_X, ADR(tableId[0]), ADR(t[0]),1);
    rtn:= GT_PvtStatus(AXIS_Y, ADR(tableId[1]), ADR(t[1]),1);
    (*读取规划速度*)
    rtn:= GT_GetPrfVel(AXIS_X, ADR(prfVel[0]),1,0);
    rtn:= GT_GetPrfVel(AXIS_Y, ADR(prfVel[1]),1,0);
    (*读取规划位置*)
    rtn:= GT_GetPrfPos(AXIS_X, ADR(prfPos[0]),1,0);
    rtn:= GT_GetPrfPos(AXIS_Y, ADR(prfPos[1]),1,0);
    END_PROGRAM
```

4. Continuous 描述方式

X 轴从 A 点运动到 B 点，Y 轴从 C 点运动到 D 点。要求 X 轴到达 B 点时，Y 轴同时到达 D 点。

首先调用 GT_PvtContinuousCalculate 指令计算 X 轴的运动时间 t_x 和 Y 轴的运动时间 t_y。该指令不会把数据点发送到运动控制器。如果 $t_x>t_y$，Y 轴延时 t_x-t_y 启动；如果 $t_x<t_y$，X 轴延时 t_y-t_x 启动。X 轴和 Y 轴速度曲线如图 4-37 所示。

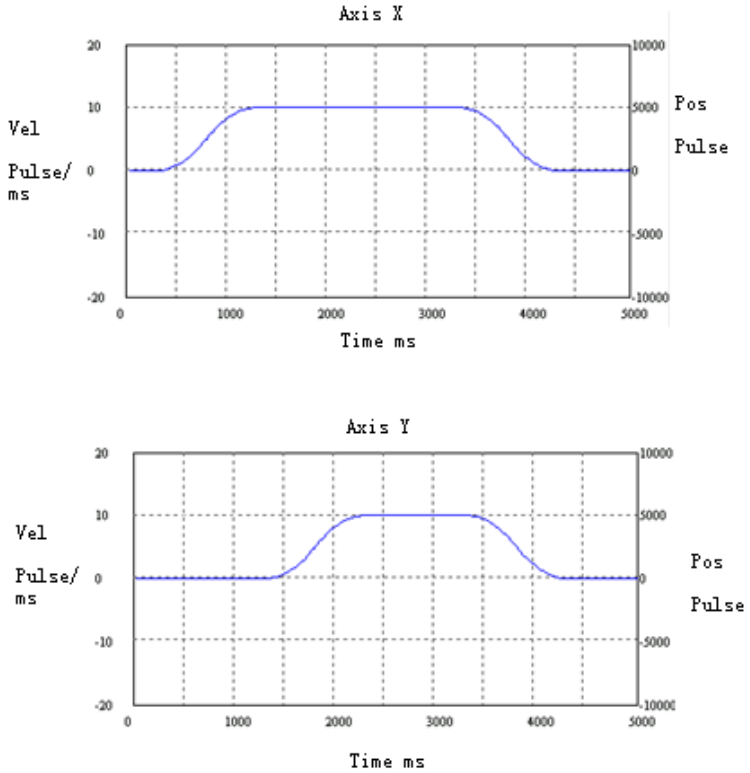


图 4-37 Continuous 描述方式下的 X 轴和 Y 轴速度曲线

例程 4-12 Continuous 描述方式

```
PROGRAM PLC_PRG
VAR
    xInitDone:BOOL:= FALSE;
    rtn:INT;
```

```
        mask:DINT;
        xStart:BOOL:= FALSE;
        (*X轴的数据点参数*)
        pos_x:ARRAY [0..1] OF LREAL:= 0,30000;
        vel_x:ARRAY [0..1] OF LREAL:= 0,0;
        percent_x:ARRAY [0..1] OF LREAL:= 100,100;
        velMax_x:ARRAY [0..1] OF LREAL:= 10,10;
        acc_x:ARRAY [0..1] OF LREAL:= 0.01,0.01;
        dec_x:ARRAY [0..1] OF LREAL:= 0.01,0.01;
        time_x:ARRAY [0..1] OF LREAL;
        timeBegin_x:LREAL;
        (*Y轴的数据点参数*)
        pos_y:ARRAY [0..1] OF LREAL:= 0,20000;
        vel_y:ARRAY [0..1] OF LREAL:= 0,0;
        percent_y:ARRAY [0..1] OF LREAL:= 100,100;
        velMax_y:ARRAY [0..1] OF LREAL:= 10,10;
        acc_y:ARRAY [0..1] OF LREAL:= 0.01,0.01;
        dec_y:ARRAY [0..1] OF LREAL:= 0.01,0.01;
        time_y:ARRAY [0..1] OF LREAL;
        timeBegin_y:LREAL;
        prfVel, prfPos, t: ARRAY [0..1] OF LREAL;
        tableId:ARRAY [0..1] OF INT;
    END_VAR
    VAR CONSTANT
        AXIS_X:INT:= 1;
        AXIS_Y:INT:= 2;
        TABLE_X:INT:= 1;
        TABLE_Y:INT:= 2;
    END_VAR
    (* @END_DECLARATION := '0' *)
    IF NOT xInitDone THEN
        (*配置运动控制器*)
        (*注意：配置文件取消了各轴的报警和限位*)
        rtn:= GT_LoadConfig('test.cfg);
        (*清除各轴的报警和限位*)
        rtn:= GT_ClrSts(1, 8);
        (*伺服使能*)
        rtn:= GT_AxisOn(AXIS_X);
        rtn:= GT_AxisOn(AXIS_Y);
        xInitDone:= TRUE;
    END_IF
    IF xStart THEN
        (*设置为PVT模式*)
        rtn:= GT_PrfPvt(AXIS_X);
        rtn:= GT_PrfPvt(AXIS_Y);
```

```
        (*计算X轴运动时间*)
        rtn:= GT_PvtContinuousCalculate(2, ADR(pos_x), ADR(vel_x), ADR(percent_x),
                ADR(velMax_x), ADR(acc_x), ADR(dec_x), ADR(time_x));
        (*计算Y轴运动时间*)
        rtn:= GT_PvtContinuousCalculate(2, ADR(pos_y), ADR(vel_y), ADR(percent_y),
                ADR(velMax_y), ADR(acc_y), ADR(dec_y), ADR(time_y));
        (*计算启动延时*)
        IF time_x[1] < time_y[1] THEN
            timeBegin_x:= time_y[1] - time_x[1];
            timeBegin_y:= 0;
        ELSE
            timeBegin_x:= 0;
            timeBegin_y:= time_x[1] - time_y[1];
        END_IF
        (*向数据表发送数据*)
        rtn:= GT_PvtTableContinuous(TABLE_X, 2, ADR(pos_x), ADR(vel_x),
                ADR(percent_x), ADR(velMax_x), ADR(acc_x), ADR(dec_x),timeBegin_x);
        rtn:= GT_PvtTableContinuous(TABLE_Y, 2, ADR(pos_y), ADR(vel_y),
                ADR(percent_y), ADR(velMax_y), ADR(acc_y), ADR(dec_y),timeBegin_y);
        (*选择数据表*)
        rtn:= GT_PvtTableSelect(AXIS_X, TABLE_X);
        rtn:= GT_PvtTableSelect(AXIS_Y, TABLE_Y);
        (*设置循环次数*)
        rtn:= GT_SetPvtLoop(AXIS_X, LOOP_COUNT);
        rtn:= GT_SetPvtLoop(AXIS_Y, 2*LOOP_COUNT-1);
        (*同时启动X轴和Y轴，由于Y轴的第1个数据点时间为2000ms，因此X轴启动2000ms以后，
    Y轴才开始运动*)
        mask:= SHL(WORD#1,AXIS_X-1) OR SHL(WORD#1,AXIS_Y-1);
        rtn:= GT_PvtStart(mask);
        xStart:= FALSE;
    END_IF
    (*读取数据表和运动时间*)
    rtn:= GT_PvtStatus(AXIS_X, ADR(tableId[0]), ADR(t[0]),1);
    rtn:= GT_PvtStatus(AXIS_Y, ADR(tableId[1]), ADR(t[1]),1);
    (*读取规划速度*)
    rtn:= GT_GetPrfVel(AXIS_X, ADR(prfVel[0]),1,0);
    rtn:= GT_GetPrfVel(AXIS_Y, ADR(prfVel[1]),1,0);
    (*读取规划位置*)
    rtn:= GT_GetPrfPos(AXIS_X, ADR(prfPos[0]),1,0);
    rtn:= GT_GetPrfPos(AXIS_Y, ADR(prfPos[1]),1,0);
    END_PROGRAM
```

4.4 速度滤波

4.4.1 指令列表

表 4-15 速度滤波指令列表

指令	说明	页码
GT_SetAxisPrfVelFilter	设置轴的规划速度滤波参数	73
GT_GetAxisPrfVelFilter	获取轴的规划速度滤波参数	62
GT_SetAxisEncVelFilter	设置轴的编码器速度滤波参数	73
GT_GetAxisEncVelFilter	获取轴的编码器速度滤波参数	62

4.4.2 重点说明

上述指令在当用户觉得运动不平滑时或者编码器有波动时使用，用于优化运动控制中的速度，指令的用法参考指令详细说明。

上海固高欧辰智能科技有限公司 www.softlink.cn 021-54708786 021-54708386 上海市闵行区东川路555号4号楼一层

第5章 指令详细说明

提示	以下表格中的“章节页码”即为此指令在章节中的位置。可以使用“超级链接”进行索引。“指令示例”即为与此指令相关的例程，可以使用“超级链接”进行索引。

指令 1 GT_ArcXYC

指令原型	short GT_ArcXYC(short crd, long x, long y, double xCenter, double yCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。使用圆心描述方法描述圆弧。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-3 圆弧插补例程**		

指令 2 GT_ArcXYR

指令原型	short GT_ArcXYR(short crd, long x, long y, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		

x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
radius	圆弧插补的圆弧半径值。取值范围：[-1073741824, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-3 圆弧插补例程**

指令 3 GT_ArcYZC

指令原型	short GT_ArcYZC(short crd, long y, long z, double yCenter, double zCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。		

	(2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-3 圆弧插补例程**

指令 4 GT_ArcYZR

指令原型	short GT_ArcYZR(short crd, long y, long z, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-1073741824, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-3 圆弧插补例程**		

指令 5 GT_ArcZXC

指令原型	short GT_ArcZXC(short crd, long z, long x, double zCenter, double xCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	ZX 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 10 个参数，参数的详细信息如下。		

crd	坐标系号。正整数，取值范围：[1, 2]。
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-3 圆弧插补例程**

指令 6 GT_ArcZXR

指令原型	short GT_ArcZXR(short crd, long z, long x, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	ZX 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-1073741824, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：		

	(1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-3 圆弧插补例程**

指令 7 GT_BufDelay

指令原型	short GT_BufDelay(short crd, unsigned short delayTime, short fifo=0)		
指令说明	缓存区内延时设置指令。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
delayTime	延时时间。取值范围：[0, 16383]，单位：ms。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-2 直线插补例程**		

指令 8 GT_BufGear

指令原型	short GT_BufGear(short crd, short gearAxis, long pos, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴跟随运动。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
gearAxis	需要进行跟随运动的轴号，取值范围：[1, 8]。该轴不能处于坐标系中。		
pos	跟随运动的位移量，单位：pulse。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-7 刀向跟随功能**		

指令 9 GT_BufLmtsOff

指令原型	short GT_BufLmtsOff(short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内无效限位开关。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
axis	需要将限位无效的轴的编号。 当core=1时，取值范围为[1,12] 当 core=2 时，取值范围为[1,16]		
limitType	需要无效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：需要将该轴的正限位无效。 MC_LIMIT_NEGATIVE(该宏定义为 1)：需要将该轴的负限位无效。 -1：需要将该轴的正限位和负限位都无效，默认为该值。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GT_BufLmtsOn		
指令示例	无		

指令 10 GT_BufLmtsOn

指令原型	short GT_BufLmtsOn(short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内有效限位开关。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
axis	需要将限位有效的轴的编号 当core=1时，取值范围为[1,12] 当 core=2 时，取值范围为[1,16]		
limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：需要将该轴的正限位设置为有效。 MC_LIMIT_NEGATIVE(该宏定义为 1)：需要将该轴的负限位设置为有效。 -1：需要将该轴的正限位和负限位都设置为有效，默认为该值。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GT_BufLmtsOff		
指令示例	无		

指令 11 GT_BufMove

指令原型	short GT_BufMove(short crd, short moveAxis, long pos, double vel, double acc, short modal, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴点位运动。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
moveAxis	需要进行点位运动的轴号， 当core=1时，取值范围为[1,12] 当 core=2 时，取值范围为[1,16] 该轴不能处于坐标系中。		
pos	点位运动的目标位置，单位：pulse。		
vel	点位运动的目标速度，单位：pulse/ms。		
acc	点位运动的加速度，单位：pulse/ms^2。		
modal	点位运动的模式。 0：该指令为非模态指令，即不阻塞后续的插补缓存区指令的执行。 1：该指令为模态指令，将会阻塞后续的插补缓存区指令的执行。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-6 刀向跟随功能**		

指令 12 GT_CrdClear

指令原型	short GT_CrdClear(short crd, short fifo)		
指令说明	清除插补缓存区内的插补数据。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
fifo	所要清除的插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-2 直线插补例程**		

指令 13 GT_CrdData

指令原型	short GT_CrdData(short crd, TCrdData *pCrdData, short fifo=0)		
指令说明	用于在使用前瞻时。调用该指令表示后续没有新的数据，将会一次性把前瞻缓存区的数据压入运动缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pCrdData	只能设置为：NULL。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]。默认值为：0。		
指令返回值	若返回值为非零值，说明前瞻缓存区还有数据没有被压入运动缓存区，而运动缓存区没有空间了。此时需要检查运动缓存区的空间（调用 GT_CrdSpace 检查）。当检查运动缓存区有空间时，再次调用 GT_CrdData 指令，直至返回值为 0 时，前瞻缓存区的数据才被完全送入运动缓存区。		
相关指令	无		
指令示例	**例程 4-5 前瞻预处理例程**		

指令 14 GT_CrdSpace

指令原型	short GT_CrdSpace(short crd, long *pSpace, short fifo=0)		
指令说明	查询插补缓存区剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSpace	读取插补缓存区中的剩余空间。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-5 前瞻预处理例程**		

指令 15 GT_CrdStart

指令原型	short GT_CrdStart(short mask, short option)
指令说明	启动插补运动。
指令类型	立即指令，调用后立即生效。 章节页码 9
指令参数	该指令共有 2 个参数，参数的详细信息如下。
mask	从 bit0~bit1 按位表示需要启动的坐标系。 bit0 对应坐标系 1，bit1 对应坐标系 2。 0：不启动该坐标系，1：启动该坐标系。
option	从 bit0~bit1 按位表示坐标系需要启动的缓存区的编号。 bit0 对应坐标系 1，bit1 对应坐标系 2。 0：启动坐标系中 FIFO0 的运动，1：启动坐标系中 FIFO1 的运动。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 若使用了辅助 fifo1 运动，检查当前坐标系位置没有恢复到 fifo0 断点坐标系位置。 (3) 检查参数设置是否启动了坐标系。

	(4) 检查坐标系是否在运动。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-2 直线插补例程**

指令 16 GT_CrdStatus

指令原型	short GT_CrdStatus(short crd, short *pRun, long *pSegment, short fifo=0)		
指令说明	查询插补运动坐标系状态。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pRun	读取插补运动状态。0：该坐标系的该 FIFO 没有在运动；1：该坐标系的该 FIFO 正在进行插补运动。		
pSegment	读取当前已经完成的插补段数。当重新建立坐标系或者调用 GT_CrdClear 指令后，该值会被清零。		
fifo	所要查询运动状态的插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-2 直线插补例程**		

指令 17 GT_GetAxisEncVelFilter

指令原型	short GT_GetAxisEncVelFilter(short axis, short *pFilterNumExp)		
指令说明	获取轴的编码器速度滤波参数。		
指令类型	立即指令，调用后立即生效。	章节页码	53
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	轴号，取值范围：[1, 12]。		
pFilterNumExp	获取滤波窗宽度指数参数，取值范围：0-7 表示窗宽 2^n		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetAxisEncVelFilter		
指令示例	无		

指令 18 GT_GetAxisPrfVelFilter

指令原型	short GT_GetAxisPrfVelFilter(short axis, short *pFilterNumExp)		
指令说明	获取轴的规划速度滤波参数。		
指令类型	立即指令，调用后立即生效。	章节页码	53
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	轴号，取值范围：[1, 12]。		
pFilterNumExp	获取滤波窗宽度指数参数，取值范围：0-7 表示窗宽 2^n		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetAxisPrfVelFilter		
指令示例	无		

指令 19 GT_GetCrdPos

指令原型	short GT_GetCrdPos(short crd, double *pPos)		
指令说明	查询该坐标系的当前坐标位置值。获取的坐标值可能和规划位置不一致，取决于建立坐标系的原点是否为零。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pPos	读取的坐标系的坐标值。单位：pulse。该参数应该为一个数组首元素的指针，数组的元素个数取决于该坐标系的维数。		
指令返回值	请参照指令返回值列表		
相关指令	无		
指令示例	**例程 4-4 插补 FIFO 管理**		

指令 20 GT_GetCrdPrm

指令原型	short GT_GetCrdPrm(short crd, TCrdPrm *pCrdPrm)		
指令说明	查询坐标系参数。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pCrdPrm	读取坐标系的相关参数 结构体的成员含义参照 GT_SetCrdPrm 指令说明。		
指令返回值	请参照指令返回值列表.		
相关指令	GT_SetCrdPrm		
指令示例	无		

指令 21 GT_GetCrdStopDec

指令原型	short GT_GetCrdStopDec(short crd, double *pDecSmoothStop, double *pDecAbruptStop)		
指令说明	查询插补运动平滑停止、急停合成加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pDecSmoothStop	查询坐标系合成平滑停止加速度，单位：pulse/ms^2。		
pDecAbruptStop	查询坐标系合成急停加速度，单位：pulse/ms^2。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetCrdStopDec		
指令示例	无		

指令 22 GT_GetCrdVel

指令原型	short GT_GetCrdVel(short crd, double *pSynVel)
指令说明	查询该坐标系的当前坐标速度值。

指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSynVel	读取的坐标系的合成速度值，单位：pulse/ms。		
指令返回值	请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 23 GT_GetPvtLoop

指令原型	short GT_GetPvtLoop(short profile, long *pLoopCount, long *pLoop)		
指令说明	查询 PVT 运动模式循环次数。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划轴号。		
pLoopCount	查询已经循环的次数。		
pLoop	查询循环执行的总次数。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetPvtLoop		
指令示例	无		

指令 24 GT_GetRemainderSegNum

指令原型	short GT_GetRemainderSegNum(short crd, long *pSegment, short fifo=0)		
指令说明	读取未完成的插补段段数。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSegment	读取的剩余插补段的段数。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 25 GT_GetUserSegNum

指令原型	short GT_GetUserSegNum(short crd, long *pSegment, short fifo=0)		
指令说明	读取自定义插补段段号。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSegment	读取的用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		

指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。
相关指令	GT_SetUserSegNum
指令示例	无

指令 26 GT_InitLookAhead

指令原型	short GT_InitLookAhead(short crd, short fifo, double T, double accMax, short n, TCrdData *pLookAheadBuf)		
指令说明	初始化插补前瞻缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
T	拐弯时间。单位：ms。		
accMax	最大加速度。单位：pulse/ms^2。		
n	前瞻缓存区大小。取值范围：[0, 32767)。		
pLookAheadBuf	前瞻缓存区内存区指针。		
指令返回值	请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-5 前瞻预处理例程**		

指令 27 GT_LnXY

指令原型	short GT_LnXY(short crd, long x, long y, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面二维直线插补。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-2 直线插补例程**		

指令 28 GT_LnXYG0

指令原型	short GT_LnXYG0(short crd, long x, long y, double synVel, double synAcc, short fifo=0)		
指令说明	二维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 29 GT_LnXYZ

指令原型	short GT_LnXYZ(short crd, long x, long y, long z, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	三维直线插补。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 30 GT_LnXYZA

指令原型	short GT_LnXYZA(short crd, long x, long y, long z, long a, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	四维直线插补。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 31 GT_LnXYZAG0

指令原型	short GT_LnXYZAG0(short crd, long x, long y, long z, long a, double synVel, double synAcc, short fifo=0)		
指令说明	四维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返		

	回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	无

指令 32 GT_LnXYZG0

指令原型	short GT_LnXYZG0(short crd, long x, long y, long z, double synVel, double synAcc, short fifo=0)		
指令说明	三维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 33 GT_PrfPvt

指令原型	short GT_PrfPvt(short profile)		
指令说明	设置指定轴为 PVT 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
profile	规划轴号。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-9 PVT 描述方式**		

指令 34 GT_PvtContinuousCalculate

指令原型	short GT_PvtContinuousCalculate(long count, double *pPos, double *pVel, double *pPercent, double *pVelMax, double *pAcc, double *pDec, double *pTime)
指令说明	计算 PVT 运动模式 Continuous 描述方式下各数据点的时间。

指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
count	数据点个数。该指令用来计算各数据点时间，不会将数据点下载到运动控制器。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pVel	数据点速度数组。单位：pulse/ms，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
pVelMax	数据点最大速度数组。单位：pulse/ms，数组长度为 count。		
pAcc	数据点加速度数组。单位是“pulse/ms²”，数组长度为 count。		
pDec	数据点减速度数组。单位是“pulse/ms²”，数组长度为 count。		
pTime	返回各数据点的时间。单位：ms。		
指令返回值	请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-12 Continuous 描述方式**		

指令 35 GT_PvtPercentCalculate

指令原型	short GT_PvtPercentCalculate(long count, double *pTime, double *pPos, double *pPercent, double velBegin, double *pVel)		
指令说明	计算 PVT 运动模式 Percent 描述方式下各数据点的速度。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
count	数据点个数。该指令用来计算各数据点的速度，不会将数据点下载到运动控制器。		
pTime	数据点时间数组。单位：ms，数组长度为 count。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
velBegin	起点速度。单位：pulse/ms。		
pVel	返回各数据点的速度。单位：pulse/ms。		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 (2) 若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 (3) 请检查传递的数据点是否大于 1024 个。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-11 Percent 描述方式**		

指令 36 GT_PvtStart

指令原型	short GT_PvtStart(long mask)		
指令说明	启动 PVT 运动。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
mask	按位指示需要启动的轴号。当 bit 位为 1 时表示启动对应的轴。		

	在启动运动之前，可以调用 GT_PvtTableSelect 选择数据表。如果没有选择数据表，默认使用数据表 1。如果数据表为空，则启动失败。
指令返回值	若返回值为 1： (1) 目标数据表不应为空。请检查目标数据表是否空。 (2) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-9 PVT 描述方式**

指令 37 GT_PvtStatus

指令原型	short GT_PvtStatus(short profile, short *pTableId, double *pTime, short count)		
指令说明	读取 PVT 运动状态。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。		
pTableId	当前正在使用的数据表 ID。		
pTime	当前轴已经运动的时间，单位：ms。		
count	读取的轴数。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-9 PVT 描述方式**		

指令 38 GT_PvtTable

指令原型	short GT_PvtTable(short tableId, long count, double *pTime, double *pPos, double *pVel)		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 PVT 描述方式。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1 个存储空间		
pTime	数据点时间数组，单位：ms，数组长度为 count。		
pPos	数据点位置数组，单位：pulse，数组长度为 count。		
pVel	数据点速度数组，单位：pulse/ms，数组长度为 count。		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 (2) 若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 (3) 请检查传递的数据点是否大于 1024 个。 其他返回值：请参照指令返回值列表。		

相关指令	无
指令示例	**例程 4-9 PVT 描述方式**

指令 39 GT_PvtTableComplete

指令原型	short GT_PvtTableComplete(short tableId, long count, double *pTime, double *pPos, double *pA, double *pB, double *pC, double velBegin, double velEnd)		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 Complete 描述方式。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1 个存储空间。		
pTime	数据点时间数组。单位：ms，数组长度为 count。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pA、pB、pC	工作数组，内部使用，数组长度为 count。 该数组用户不必赋值。		
velBegin	起点速度。单位：pulse/ms。		
velEnd	终点速度。单位：pulse/ms。		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 (2) 若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 (3) 请检查传递的数据点是否大于 1024 个。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-10 Complete 描述方式**		

指令 40 GT_PvtTableContinuous

指令原型	short GT_PvtTableContinuous(short tableId, long count, double *pPos, double *pVel, double *pPercent, double *pVelMax, double *pAcc, double *pDec, double timeBegin)		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 Continuous 描述方式。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1~8 个存储空间。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pVel	数据点速度数组。单位：pulse/ms，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
pVelMax	数据点最大速度数组。单位：pulse/ms。数组长度为 count。		
pAcc	数据点加速度数组。单位是“pulse/ms²”。数组长度为 count。		
pDec	数据点减速度数组。单位是“pulse/ms²”。数组长度为 count。		

timeBegin	起点时间。单位：ms。
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 (2) 若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 (3) 请检查传递的数据点是否大于 1024 个。 其他返回值：请参照指令返回值列表。
相关指令	无
指令示例	**例程 4-12 Continuous 描述方式**

指令 41 GT_PvtTablePercent

指令原型	short GT_PvtTablePercent(short tableId, long count, double *pTime, double *pPos, double *pPercent, double velBegin)		
指令说明	向 PVT 运动模式指定数据表传送数据，采用 Percent 描述方式。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
tableId	指定数据表。取值范围：[1, 32]。		
count	数据点个数。每个数据表具有 1024 个存储空间。 每个数据点占用 1~3 个存储空间。		
pTime	数据点时间数组。单位：ms，数组长度为 count。		
pPos	数据点位置数组。单位：pulse，数组长度为 count。		
pPercent	数据点百分比数组。数组长度为 count。 百分比的取值范围：[0, 100]。		
velBegin	起点速度。单位：pulse/ms。		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 (2) 若当前轴在规划运动，请调用 GT_Stop()停止运动再调用该指令。 (3) 请检查传递的数据点是否大于 1024 个。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 4-11 Percent 描述方式**		

指令 42 GT_PvtTableSelect

指令原型	short GT_PvtTableSelect(short profile, short tableId)		
指令说明	选择 PVT 运动模式数据表。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。		
tableId	指定数据表。 PVT 模式提供 32 个数据表，取值范围：[1, 32]。		
指令返回值	若返回值为 1：目标数据表不应为空。请检查目标数据表是否空。 其他返回值：请参照指令返回值列表。		

相关指令	无
指令示例	**例程 4-10 Complete 描述方式**

指令 43 GT_SetAxisEncVelFilter

指令原型	short GT_SetAxisEncVelFilter(short axis, short filterNumExp)		
指令说明	设置轴的编码器速度滤波参数。		
指令类型	立即指令。	章节页码	53
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	轴号，取值范围：[1, 12]。		
filterNumExp	设置滤波窗宽度指数参数，取值范围：0-7 表示窗宽 2^n		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetAxisEncVelFilter		
指令示例	无		

指令 44 GT_SetAxisPrfVelFilter

指令原型	short GT_SetAxisPrfVelFilter(short axis, short filterNumExp)		
指令说明	设置轴的规划速度滤波参数。		
指令类型	立即指令。	章节页码	53
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	轴号，取值范围：[1, 12]。		
filterNumExp	设置滤波窗宽度指数参数，取值范围：0-7 表示窗宽 2^n		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetAxisPrfVelFilter		
指令示例	无		

指令 45 GT_SetCrdPrm

指令原型	short GT_SetCrdPrm(short crd, TCrdPrm *pCrdPrm)		
指令说明	设置坐标系参数，确立坐标系映射，建立坐标系。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号，取值范围：[1, 2]。		
pCrdPrm	设置坐标系的相关参数。 typedef struct CrdPrm { short dimension; short profile[8]; double synVelMax; double synAccMax; short evenTime; short setOriginFlag; long originPos[8]; }TCrdPrm; **dimension**：坐标系的维数。取值范围：[1, 4]。		

	Profile[8]：坐标系与规划器的映射关系。Profile[0..7]对应规划轴 1~8，如果规划轴没有对应到该坐标系，则 profile[x]的值为 0；如果对应到了 X 轴，则 profile[x]为 1，Y 轴对应为 2，Z 轴对应为 3，A 轴对应为 4。不允许多个规划轴映射到相同坐标系的相同坐标轴，也不允许把相同规划轴对应到不同的坐标系，否则该指令将会返回错误值。每个元素的取值范围：[0, 4]。 **synVelMax**：该坐标系的最大合成速度。如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。取值范围：(0, 32767)。单位：pulse/ms。 **synAccMax**：该坐标系的最大合成加速度。如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。取值范围：(0, 32767)。单位：pulse/ms^2。 **evenTime**：每个插补段的最小匀速段时间。取值范围：[0, 32767)。单位：ms。 **setOriginFlag**：表示是否需要指定坐标系的原点坐标的规划位置，该参数可以方便用户建立区别于机床坐标系的加工坐标系。0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上。1：需要指定原点坐标值，坐标系的原点在 originPos 指定的规划位置上。 **originPos[8]**：指定的坐标系原点的规划位置值。
指令返回值	若返回值为 1： (1) 若坐标系下各轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2) 请检查映射到 Profile 有没被激活，若无，则返回错误。 (3) 请见检查相应轴是否在坐标系下。 其他返回值：请参照指令返回值列表。
相关指令	GT_GetCrdPrm
指令示例	**例程 4-1 建立坐标系**

指令 46 GT_SetCrdStopDec

指令原型	short GT_SetCrdStopDec(short crd, double decSmoothStop, double decAbruptStop)		
指令说明	设置插补运动平滑停止、急停合成加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
decSmoothStop	设置的坐标系合成平滑停止加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
decAbruptStop	设置的坐标系合成急停加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetCrdStopDec		
指令示例	无		

指令 47 GT_SetOverride

指令原型	short GT_SetOverride(short crd, double synVelRatio)		
指令说明	设置插补运动目标合成速度倍率。		
指令类型	立即指令，调用后立即生效。	章节页码	9
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
synVelRatio	设置的插补目标速度倍率，取值范围：(0, 1]，系统默认该值为：1。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		

相关指令	无
指令示例	无

指令 48 GT_SetPvtLoop

指令原型	short GT_SetPvtLoop(short profile, long loop)		
指令说明	设置 PVT 运动模式循环次数。		
指令类型	立即指令，调用后立即生效。	章节页码	34
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。		
loop	指定循环执行的次数。 0 表示无限循环。		
指令返回值	若返回值为 1：请检查当前轴是否为 PVT 模式，若不是，请先调用 GT_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetPvtLoop		
指令示例	**例程 4-10 Complete 描述方式**		

指令 49 GT_SetUserSegNum

指令原型	short GT_SetUserSegNum(short crd, long segNum, short fifo=0)		
指令说明	设置自定义插补段段号。		
指令类型	缓存区指令。	章节页码	9
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
segNum	设置用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetUserSegNum		
指令示例	无		

第6章 索引

6.1 指令索引

6.2 例程索引

6.3 表格索引

6.4 图片索引