固高科技
GOOGOL TECHNOLOGY

GSN 系列运动控制器编程手册

高级功能

R1.1

2019 年 04 月

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，固高科技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

固高科技（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二楼 W211 室

电　　话：0755-26970817　26737236　26970824

传　　真：0755-26970821

电子邮件：support@googoltech.com

网　　址：http://www.googoltech.com.cn

固高科技（香港）有限公司

地　　址：香港九龍觀塘偉業街 108 號絲寶國際大廈 10 樓 1008-09 室

電　　話：+(852) 2358-1033

傳　　真：+(852) 2719-8399

電子郵件：info@googoltech.com

網　　址：http://www.googoltech.com

臺灣固高科技股份有限公司

地　　址：台中室西屯區台中港路三段 97 號 7 樓之 3

電　　話：+886-4-23588245

傳　　真：+886-4-23586495

電子郵件：googoltw@googoltech.com

文档版本

版本号	修订日期
1.0	2018 年 10 月 17 日
1.1	2019 年 04 月 30 日

前言

感谢选用固高运动控制器

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。

固高产品的更多信息

固高科技的网址是 http://www.googoltech.com.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（0755－26970817）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：support@googoltech.com；

电话：0755－26970843

发函至：深圳市高新技术产业园南区园深港产学研基地西座二楼 W211 室
固高科技（深圳）有限公司

邮编： 518057

编程手册的用途

用户通过阅读本手册，能够了解固高运动控制器的控制功能，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，且对固高运动控制器基本功能比较了解，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，详细介绍了运动控制器的高级运动模式及编程实现。

相关文件

关于控制器的调试和安装，请参见随产品配套的《GTN 运动控制器用户手册》。
关于控制器基本功能使用，请参见随产品配套的《GTN 系列运动控制器编程手册之基本功能》
关于扩展模块的使用，请参见随产品配套的《GTN 扩展功能-扩展模块功能》。

相关手册及控制器适用文档列表见于光盘的 manual 目录下。
亦可通过固高科技公司网站下载如驱动程序、dll 文件、例程、Demo 等相关文件，网址为：www.googoltech.com.cn/pro_view-53.html

目录

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 GTN_FollowData）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《GTN 系列运动控制器编程手册之基本功能》。

表 1-1　指令列表

第2章　PT 运动模式

提示	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 GTN_FollowData）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《GTN 系列运动控制器编程手册之基本功能》。

2.1 指令列表

表 2-1　PT 运动模式指令列表

指令	说明	页码
GTN_PrfPt	设置指定轴为 PT 运动模式	61
GTN_PtSpace	查询 PT 运动模式指定 FIFO 的剩余空间	63
GTN_PtData	向 PT 运动模式指定 FIFO 增加数据	62
GTN_PtClear	清除 PT 运动模式指定 FIFO 中的数据 运动状态下该指令无效 动态模式下该指令无效	62
GTN_SetPtLoop	设置 PT 运动模式循环执行的次数 动态模式下该指令无效	71
GTN_GetPtLoop	查询 PT 运动模式循环执行的次数 动态模式下该指令无效	59
GTN_PtStart	启动 PT 运动	65
GTN_SetPtMemory	设置 PT 运动模式的缓存区大小	71
GTN_GetPtMemory	读取 PT 运动模式的缓存区大小	60

2.2 重点说明

PT 模式使用一系列“位置、时间”数据点描述速度规划，用户需要将速度曲线分割成若干段，如图 2-1 所示。

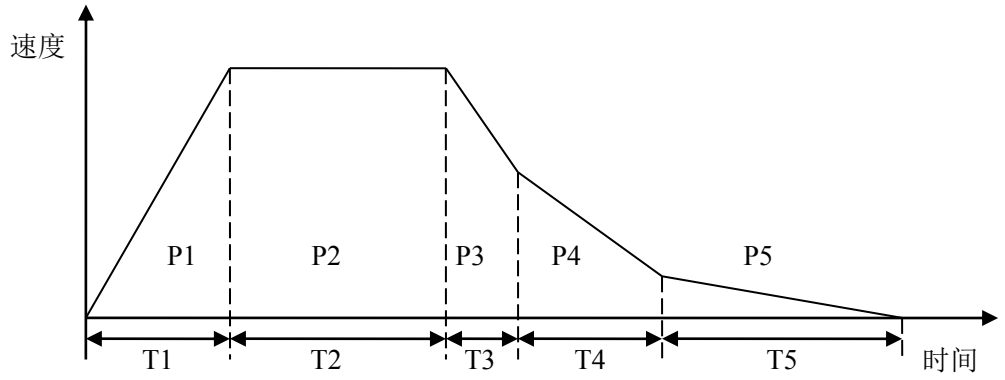

图 2-1　PT 运动速度曲线

整个速度曲线被分割成 5 段，第 1 段起点速度为 0，经过时间 T1 运动位移 P1，因此第 1 段的终

点速度为 $v1=\frac{2P1}{T1}$；第 2 段起点速度为 v1，经过时间 T2 运动位移 P2，因此第 2 段的终点速度为 $v2=\frac{2P2}{T2}-v1$；第 3、4、5 段依此类推。PT 模式的数据段要求用户输入每段所需时间和位置点。

在描述一次完整的 PT 运动时，第 1 段的起点位置和时间被假定为 0。压入控制器的数据为位置点，即相对于第 1 段的起点的绝对值，而不是每段位移长度。位置的单位是脉冲（pulse），时间单位是毫秒（ms）。

PT 模式在实现任意速度规划方面非常具有优势。用户将任意的速度规划分割为足够密的小段，用户只需要给出每段所需时间和位置点，运动控制器会计算段内各点的速度，生成一条连续的速度曲线。为了得到光滑的速度曲线，可以增加速度曲线的分割段数。

(1)　如何切换到 PT 模式？

用户必须要调用 GTN_PrfPt，才能将指定轴设定为 PT 模式。

(2)　认识 PT 模式的数据段类型。如何向 PT 模式的 FIFO 中写入数据段？

- PT 模式的数据段有 3 种类型。
- PT_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。
- PT_SEGMENT_EVEN 表示匀速段，FIFO 中各段的段内速度保持不变，段内速度=段内位移/段内时间。如图 2-2 所示。

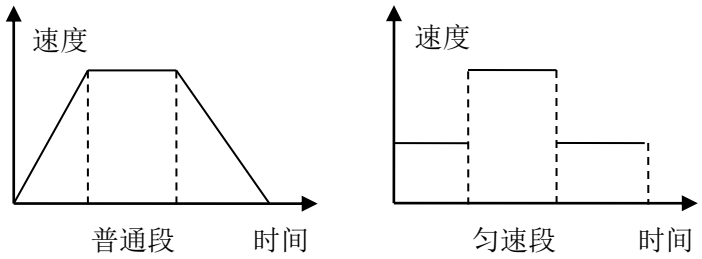

图 2-2　PT 模式匀速段类型

- PT_SEGMENT_STOP 表示停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。如图 2-3 所示。

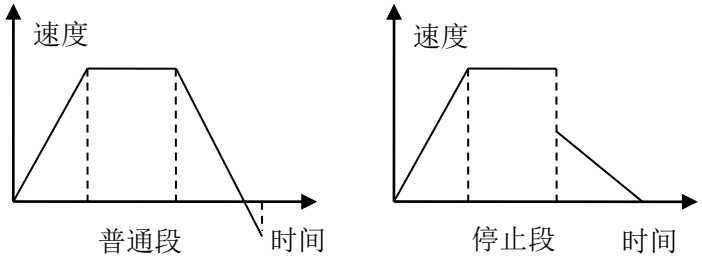

图 2-3　PT 模式停止段类型

- 调用 GTN_PtData(short profile, double pos, long time, short type, short fifo)，将数据段写入指定 FIFO 中。profile 指轴号，pos 和 time 分别为一段曲线的位置点和时间，type 指数据段类型，fifo 是指定的 FIFO 编号。

(3)　如何设置 PT 循环次数？

如果轴的运动是周期性的，用户可以只写入一个周期的运动规划数据段到 FIFO 中，然后设定循环次数，即可实现周期性的 PT 运动。调用指令 GTN_SetPtLoop 即可。

(4)　PT 模型下，如何使用 FIFO？

- PT 模式下，有 2 个 FIFO 用来存放数据，分别为 FIFO1 和 FIFO2。PT 具有 2 种 FIFO 使用模式：静态模式和动态模式。
- 静态模式下，可以选择启动其中一个 FIFO，运动完成以后规划停止。控制器不会清除 FIFO 中的数据，用户可以重复使用 FIFO 中的数据。静止状态下调用 GTN_PtClear 指令可以清空指定 FIFO。在运动状态下不能清空正在使用的 FIFO，但可以清除没有在使用的 FIFO。
- 动态模式下，不可以选择启动哪一个 FIFO，控制器会启用两个 FIFO（动态模式下，PT 指令中选择 FIFO 的参数都是无效的）。当一个 FIFO 中的数据用完以后会自动清空，同时切换到另一个 FIFO，此时可以向控制器发送新的 PT 数据。当 2 个 FIFO 中的数据都用完以后规划停止。为了避免异常停止，必须在 2 个 FIFO 中的数据都用完之前及时发送新的数据。调用 GTN_PtSpace 指令可以查询剩余多少数据空间。
- 调用 GTN_SetPtMemory 可以设置 FIFO 的大小，有 32 段和 1024 段两种选择。默认为 32 段。

在切换到 PT 模式时（调用指令 GTN_PrfPt），应设置 FIFO 为“静态模式”或“动态模式”。运动时不能修改 FIFO 的使用模式。

2.3 例程

1. PT 静态 FIFO

例程 2-1　PT 静态 FIFO

该例程生成一段梯形曲线速度规划，一共三段数据段，如表 2-2 所示。

表 2-2　PT 静态 FIFO 例程数据段

	第一段	第二段	第三段
位置点(pos)	1024	2048	1024
时间(time)	1024	1024	1024
数据段类型	普通段	普通段	普通段

PT 模式梯形曲线速度规划如图 2-4 所示。

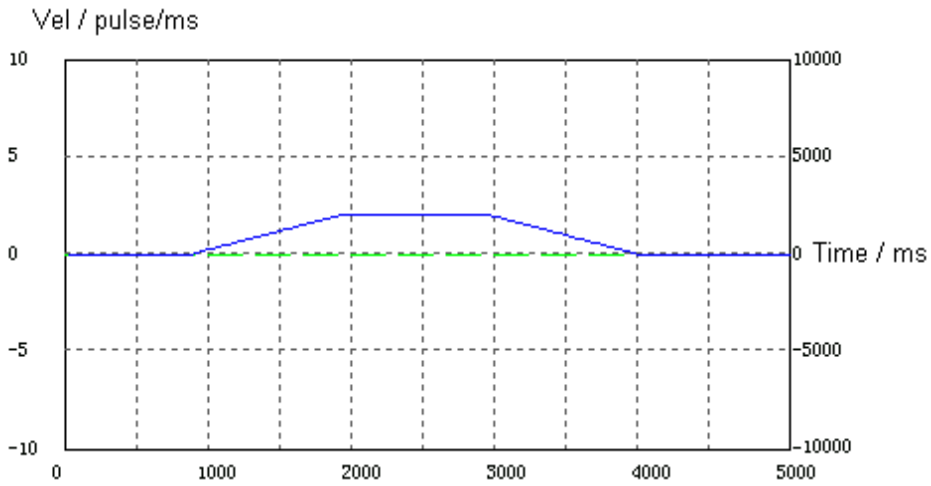

图 2-4　PT 模式梯形曲线速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

#define AXIS        1

int main(int argc, char* argv[])
{
    short sRtn, space;
    double pos;
    long time;
    long sts;
    double prfPos, prfVel;

    // 打开运动控制器
    sRtn = GTN_Open();
    // 指令返回值检测，请查阅例3-1
    commandhandler("GTN_Open", sRtn);
    // 复位控制器
    sRtn = GTN_Reset(1);
    commandhandler("GTN_Reset", sRtn);
    // 配置运动控制器
    // 注意：配置文件取消了各轴的报警和限位
    sRtn = GTN_LoadConfig(1,"test.cfg");
    commandhandler("GTN_LoadConfig", sRtn);
    // 清除各轴的报警和限位
    sRtn = GTN_ClrSts(1,1, 8);
    commandhandler("GTN_ClrSts", sRtn);
    // 伺服使能
    sRtn = GTN_AxisOn(1,AXIS);
    commandhandler("GTN_AxisOn", sRtn);
    // 位置清零
    sRtn = GTN_ZeroPos (1,AXIS);
```

```
commandhandler("GTN_ZeroPos", sRtn);
// 将AXIS轴设为PT模式
sRtn = GTN_PrfPt(1,AXIS);
commandhandler("GTN_PrfPt", sRtn);
// 清除AXIS轴的FIFO
sRtn = GTN_PtClear(1,AXIS);
commandhandler("GTN_PtClear", sRtn);
// 查询PT模式FIFO的剩余空间
sRtn = GTN_PtSpace(1,AXIS, &space);
printf("GTN_PtSpace()=%d\tspace=%d\n", sRtn, space);
// 向FIFO中增加运动数据
pos = 1024;
time = 1024;
sRtn = GTN_PtData(1,AXIS, pos, time);
commandhandler("GTN_PtData", sRtn);
// 查询PT模式FIFO的剩余空间
sRtn = GTN_PtSpace(1,AXIS, &space);
printf("GTN_PtSpace()=%d\tspace=%d\n", sRtn, space);
// 向FIFO中增加运动数据
pos += 2048;
time += 1024;
sRtn = GTN_PtData(1,AXIS, pos, time);
commandhandler("GTN_PtData", sRtn);
// 查询PT模式FIFO的剩余空间
sRtn = GTN_PtSpace(1,AXIS, &space);
printf("GTN_PtSpace()=%d\tspace=%d\n", sRtn, space);
// 向FIFO中增加运动数据
pos += 1024;
time += 1024;
sRtn = GTN_PtData(1,AXIS, pos, time);
commandhandler("GTN_PtData", sRtn);
// 启动PT运动
sRtn = GTN_PtStart(1,1<<(AXIS-1));
commandhandler("GTN_PtStart", sRtn);

while(!kbhit())
{
    // 读取AXIS轴的状态
    sRtn = GTN_GetSts (1,AXIS, &sts);
    // 读取AXIS轴的规划位置
    sRtn = GTN_GetPrfPos (1,AXIS, &prfPos);
    // 读取AXIS轴的规划速度
    sRtn = GTN_GetPrfVel (1,AXIS, &prfVel);
    printf("sts=0x%-10lxprfVel=%-10.2lfprfPos=%-10.1lf\r", sts, prfVel, prfPos);
}
```

```
    // 伺服关闭
    sRtn = GTN_AxisOff(1,AXIS);
    printf("\nGTN_AxisOff()=%d\n", sRtn);
    getch();
    return 0;
}
```

2. PT 动态 FIFO

例程 2-2　PT 动态 FIFO

该例程生成一段正弦曲线速度规划，周期为 1s，循环次数为 2。将该正弦曲线分割成若干小线段，每段时间间隔为 0.016s。计算每个小线段的时间和位置点，传入 FIFO 中。由于数据段较多，使用 PT 的动态 FIFO 模式。在存入数据段的同时不断检查 FIFO 是否已满；当两个 FIFO 都满了时，启动运动；当其中一个 FIFO 中的数据遍历完了，控制器将清空这个 FIFO，此时查询到有空间，就继续存入后面的数据段；依次下来，直到整条描述正弦曲线的小线段全部遍历完。PT 模式正弦曲线速度规划如图 2-5 所示。

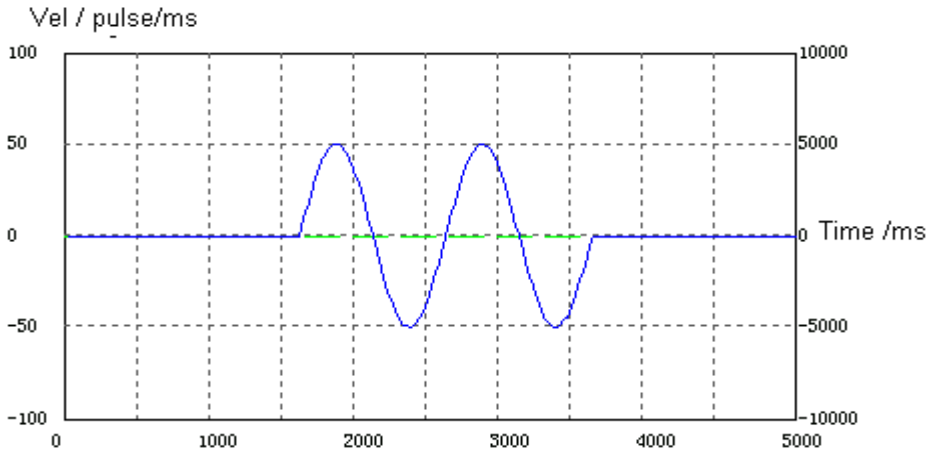

图 2-5　PT 模式正弦曲线速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "math.h"
#include "gts.h"

#define AXIS        1
#define A           50                  // 幅值
#define T           1                   // 周期
#define DELTA       0.016               // 时间分段
#define LOOP        2                   // 循环次数
#define PI          3.1415926

int main(int argc, char* argv[])
{
    short sRtn, space;
    double pos, vel, velPre, time;
    long sts;
```

```
double prfPos, prfVel;
short start, loop;

// 打开运动控制器
sRtn = GTN_Open ();
// 指令返回值检测，请查阅例3-1
commandhandler("GTN_Open ", sRtn);
// 复位控制器
sRtn = GTN_Reset (1);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn = GTN_LoadConfig (1,"test.cfg");
commandhandler("GTN_LoadConfig ", sRtn);
// 清除各轴的报警和限位
sRtn = GTN_ClrSts (1,1, 8);
commandhandler("GTN_ClrSts", sRtn);
// 伺服使能
sRtn = GTN_AxisOn (1,AXIS);
commandhandler("GTN_AxisOn", sRtn);
// 位置清零
sRtn = GTN_ZeroPos (1,AXIS);
commandhandler("GTN_ZeroPos", sRtn);
// 将AXIS轴设为PT模式
sRtn = GTN_PrfPt(1,AXIS, PT_MODE_DYNAMIC);
commandhandler("GTN_PrfPt", sRtn);
// 清空AXIS轴的FIFO
sRtn = GTN_PtClear(1,AXIS);
commandhandler("GTN_PtClear", sRtn);
pos = 0;
vel = velPre = 0;
time = 0;
start = 0;
loop = 1;
while(1)
{
    // 查询PT模式FIFO的剩余空间
    sRtn = GTN_PtSpace(1,AXIS, &space);
    if(space> 0 )
    {
        time += DELTA;
        // 计算段末速度
        vel = A*sin((2*PI)/T*time);
        // 计算段内位移
        pos += 1000*(vel+velPre)*DELTA/2;
        velPre = vel;
        if(time<loop*T)
```

```
            {
                // 发送新数据
                sRtn = GTN_PtData(1,AXIS, pos, (long)(time*1000));
                if( 0 != sRtn )
                {
                    printf("\nGTN_PtData()=%d\n", sRtn);
                    getch();
                    return 1;
                }
            }
            else
            {
                // 发送终点数据
                sRtn = GTN_PtData(1,AXIS, 0, loop*T*1000, PT_SEGMENT_STOP);
                if( 0 != sRtn )
                {
                    printf("\nGTN_PtData()=%d\n", sRtn);
                    getch();
                    return 1;
                }
                pos = 0;
                time = loop*T;
                velPre = 0;
                ++loop;
                if(loop>LOOP )
                {
                    break;
                }
            }
        }
        elseif( 0 == start )
        {
            // 启动PT运动
            sRtn = GTN_PtStart(1,1<<(AXIS-1));
            commandhandler("GTN_PtStart", sRtn);
            start = 1;
        }
        // 读取AXIS轴的状态
        sRtn =GTN_GetSts (1,AXIS, &sts);
        // 读取AXIS轴的规划位置
        sRtn = GTN_GetPrfPos (1,AXIS, &prfPos);
        // 读取AXIS轴的规划速度
        sRtn = GTN_GetPrfVel (1,AXIS, &prfVel);
        printf("sts=0x%-10lxprfVel=%-10.2lfprfPos=%-10.1lf\r", sts, prfVel, prfPos);
    }
```

```
    while(!kbhit())
    {
        // 读取AXIS轴的状态
        sRtn = GTN_GetSts (1,AXIS, &sts);
        // 读取AXIS轴的规划位置
        sRtn = GTN_GetPrfPos (1,AXIS, &prfPos);
        // 读取AXIS轴的规划速度
        sRtn = GTN_GetPrfVel (1,AXIS, &prfVel);
        printf("sts=0x%-10lxprfVel=%-10.2lfprfPos=%-10.1lf\r", sts, prfVel, prfPos);
    }
    // 伺服关闭
    sRtn =GTN_AxisOff(1,AXIS);
    printf("\nGTN_AxisOff()=%d\n", sRtn);
    getch();
    return 0;
}
```

第3章　Follow 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 GTN_FollowData）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《GTN 系列运动控制器编程手册之基本功能》。

3.1　指令列表

表 3-1　Follow 运动模式指令列表

指令	说明	页码
GTN_PrfFollow	设置指定轴为 Follow 运动模式	60
GTN_SetFollowMaster	设置 Follow 运动模式跟随主轴	69
GTN_GetFollowMaster	读取 Follow 运动模式跟随主轴	58
GTN_SetFollowLoop	设置 Follow 运动模式循环次数	69
GTN_GetFollowLoop	读取 Follow 运动模式循环次数	58
GTN_SetFollowEvent	设置 Follow 运动模式启动跟随条件	68
GTN_GetFollowEvent	读取 Follow 运动模式启动跟随条件	57
GTN_FollowSpace	查询 Follow 运动模式指定 FIFO 的剩余空间	56
GTN_FollowData	向 Follow 运动模式指定 FIFO 增加数据	55
GTN_FollowClear	清除 Follow 运动模式指定 FIFO 中的数据 运动状态下该指令无效	55
GTN_FollowStart	启动 Follow 运动	56
GTN_FollowSwitch	切换 Follow 运动模式所使用的 FIFO	57
GTN_SetFollowMemory	设置 Follow 运动模式的缓存区大小	70
GTN_GetFollowMemory	读取 Follow 运动模式的缓存区大小	59

3.2　重点说明

在很多应用中，两轴或多轴之间需要保证位置同步和速度同步。我们把被跟随的轴叫主轴，把跟随的轴叫从轴。一对典型的主轴和从轴的规划如图 3-1 所示。

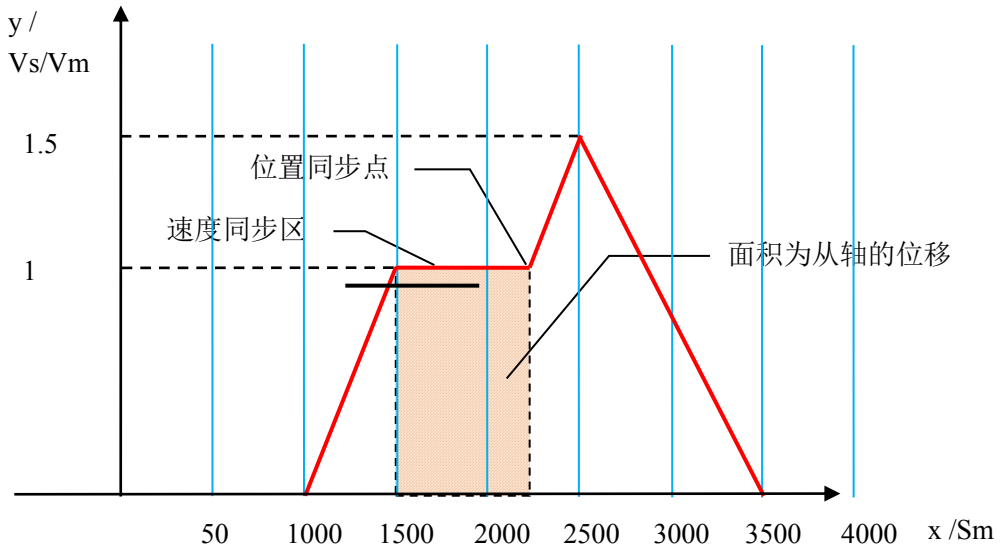

图 3-1　Follow 模式主从轴规划

图中画了一个周期的 Follow 运动，其中横坐标 x 表示主轴的位移，纵轴 y 表示从轴速度 Vs 与主轴速度 Vm 的比率。根据 Sm*(Vs/Vm)可得:Vs*(Sm/Vm)=Vs*t，即图 3-1 中的面积为从轴的位移。整个跟随模式如下：

(1)　在主轴运动到设定位置（1000pulse）时，从轴启动跟随。

(2)　在主轴运动到 1500pulse 时，从轴运动 250 ((1500-1000)*1/2)pulse 到达速度同步区，即图中阴影部分所示的速度同步区。

(3)　在主轴运动到 2250pulse 时，从轴与主轴的分别同时到达各自的位置点，即图中标注的位置同步点。

位置同步点表示主轴和从轴必须同时到达各自指定位置。

速度同步区表示主轴和从轴之间必须保持准确的速度比。

Follow 模式就是针对这种应用，给用户提供了主轴和从轴的位置和速度规划方式。用户只需要学习如何设置主轴，从轴，从轴启动跟随的条件，如何设置 Follow 循环次数，如何利用 Follow 模式中的数据段类型实现应用中所需要的规划以及如何管理 FIFO，就可以轻松实现 Follow 运动。

(1)　如何切换到 Follow 模式？

用户必须要调用 GTN_PrfFollow，才能将指定轴设定为 Follow 模式。一般应将从轴设定为 Follow 模式。

(2)　如何设置主轴，从轴？

调用 GTN_SetFollowMaster。profile 为从轴轴号，masterIndex 为主轴轴号。

为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

(3)　如何设定从轴启动跟随条件？

所谓从轴启动跟随条件，是描述什么情况下从轴开始启动运动。有两种情况：第一，调用

指令 GTN_FollowStart 以后从轴立即启动；第二，调用指令 GTN_FollowStart 以后，从轴还要等待主轴穿越了设定位置以后才启动跟随运动。用户需调用指令 GTN_SetFollowEvent 来选择使用哪种跟随条件。

(4)　如何设置 Follow 循环次数？

如果从轴的跟随运动是周期性的，用户可以只写入一个周期的运动规划到 FIFO 中，然后设定循环次数，即可实现周期性的 Follow 运动。调用指令 GTN_SetFollowLoop 即可。

(5)　认识 Follow 模式的数据段类型。如何向 Follow 模式的 FIFO 中写入数据段？

调用指令 GTN_FollowData(short core,short profile, long masterSegment, double slaveSegment, short type= FOLLOW_SEGMENT_NORMAL, short fifo)，将数据段写入指定 FIFO 中。profile 指从轴轴号，masterSegment 和 slaveSegment 分别指主轴和从轴应同时走过的位移，type 指从轴的数据段类型，fifo 是指定的 FIFO 编号。

用户压入的数据段都是对位置点的描述，控制器会根据用户选择的类型来自行计算速度。

- Follow 模式的数据段有 4 种类型。注意，都是针对从轴的规划。
- FOLLOW_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度比率为 0，从第 2 段起每段的起点速度比率等于上一段的终点速度比率。
- FOLLOW_SEGMENT_EVEN 表示恒速比率段，FIFO 中各段的段内速度比率保持不变。
- FOLLOW_SEGMENT_STOP 表示停止段，该段的终点速度比率为 0，起点速度比率根据段内位移和段内时间计算得到，和上一段的终点速度比率无关。
- FOLLOW_SEGMENT_CONTINUE 表示连续段，FIFO 中第一段的起点速度比率等于上个 FIFO 的终点速度比率，从第 2 段起每段的起点速度比率等于上一段的终点速度比率。

用户如何根据自己主轴和从轴的速度比率及位置规划，来设计相应的数据段类型？可以参考飞剪案例。

(6)　如何切换 FIFO？

Follow 模式下有 2 个独立的 FIFO 用来保存数据。2 个 FIFO 之间可以在运动状态下进行切换。下面描述一个切换 FIFO 的典型案例，如图 3-2 所示。

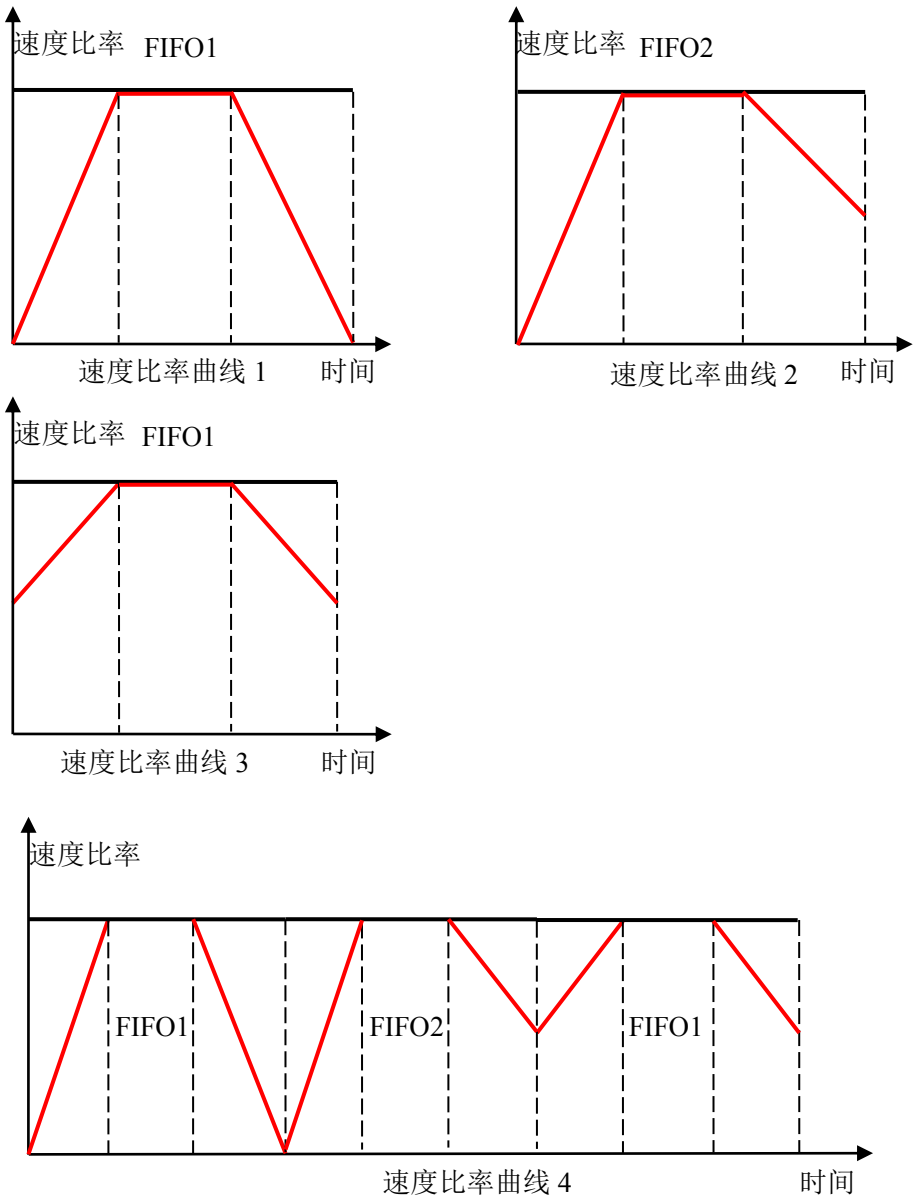

图 3-2　Follow 模式切换 FIFO

如图所示，黑色粗线为主轴规划，红色粗线为从轴规划。从轴的运动规律需要从“速度比率曲线 1”变化到“速度比率曲线 3”，为了实现从轴速度的平滑过渡，增加了一个“速度比率曲线 2”的过渡状态。“速度比率曲线 2”的起始速度比率和“速度比率曲线 1”相等，“速度比率曲线 2”的终点速度比率和“速度比率曲线 3”相等。“速度比率曲线 4”是“速度比率曲线 1”、“速度比率曲线 2”和“速度比率曲线 3”的合成。

具体的操作步骤是：

1）“速度比率曲线 1”放入 FIFO1 中，把“速度比率曲线 2”放入 FIFO2 中。假设当前正在运行的数据来自 FIFO1，调用指令 GTN_FollowSwitch，控制器会在 FIFO1 中的数据全部运行完后，自动切换去运行 FIFO2 中的数据，并且将 FIFO1 全部清空。

注意	为了实现 2 个 FIFO 之间的速度连续，存入 FIFO2 的第一段数据的时候，调用 GTN_FollowData 指令时应当将数据类型设置为 FOLLOW_SEGMENT_CONTINUE。

2）在控制器运行 FIFO2 的数据的时候，调用指令 GTN_FollowSpace 查询 FIFO1 是否被清

空。如果已被清空，就将“速度比率曲线 3”的数据存入 FIFO1 中。然后调用指令 GTN_FollowSwitch，控制器会在 FIFO2 中的数据全部运行完后，自动切换去运行 FIFO1 中的数据，并且将 FIFO2 全部清空。

用户欲知道怎么用代码具体实现这个例子，可以查看“例程 3-3　Follow 双 FIFO 切换”。

3.3 例程

1. 飞剪案例

例程 3-1　飞剪中的 follow 模式应用

飞剪应用背景介绍：

简化的飞剪设备结构是主传送带（主轴）匀速拉着待剪物品定向移动，同时，在传送带上方装有一带切刀的转子，当转子旋转一周（逆时针为正向）时，刚好和待剪物体接触，使之剪断，剪切长度为：10000pulse，转子旋转一周为：8000pulse。如图 3-3 所示。

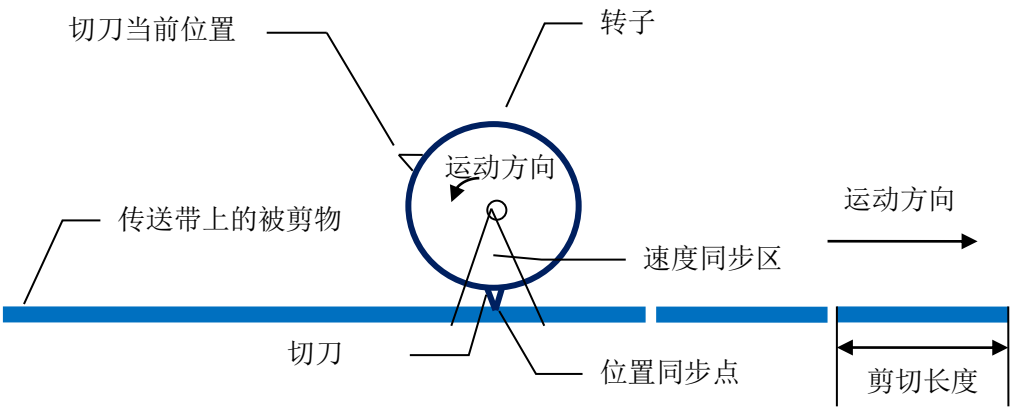

图 3-3　飞剪模型

下面我们分析如何为这样一个同步机构设计 Follow 模式下的速度规划。

首先要找出同步机构的**位置同步点**。位置同步点表示主轴和从轴必须同时到达各自指定位置（比如被剪物体每走完上图中的“剪切长度”，转轮就要刚好走完一圈，两者在各自最终位置点上必须同时到达）。该例中，待剪物被切断时主轴和从轴的位置即位置同步点。要求主轴走完 10000pulse 时，从轴必须走完 8000pulse。

我们假设以切刀当前的位置来看，转轮还要正向运动 2500 个脉冲切刀才可达到位置同步点。

其次，查看该同步机构是否需要**速度同步区**。速度同步区表示在这段区域内主轴和从轴之间必须保持准确的速度比。该设备在待剪物被切断（位置同步点）前后一段距离内，需要有一速度同步区。在此速度同步区内，要求主轴和从轴速度相等。

因此，我们可以画出飞剪的主从轴速度曲线图，如图 3-4 所示。

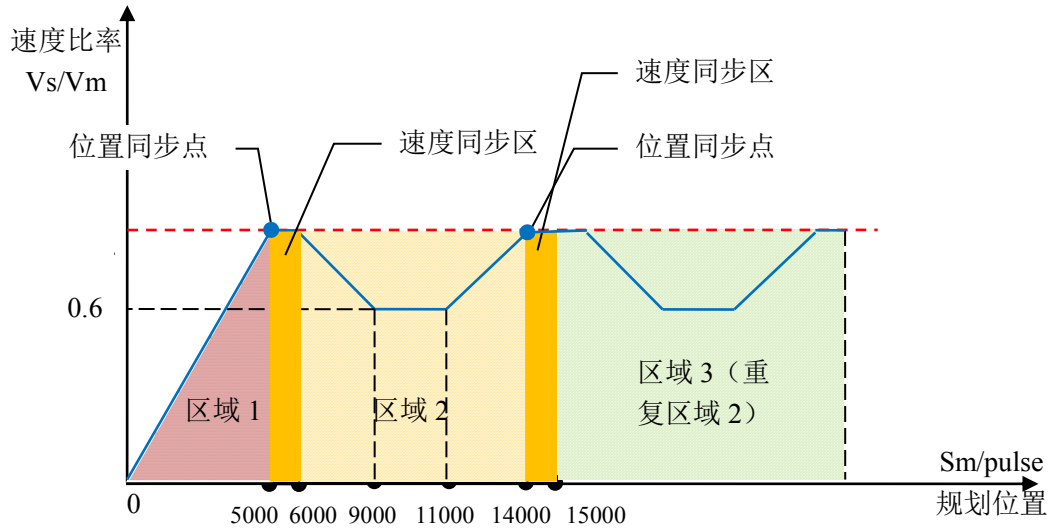

图 3-4　飞剪案例之 Follow 模式规划曲线

从上图来看，假设主轴（即传送带）是以 Jog 模式在运动，而从轴（即转轮）是 Follow 模式运动。区域 1（红色阴影部分）是从轴启动跟随，表示从轴追赶主轴到达位置同步点的位移。区域 2（黄色阴影部分）表示从轴旋转完整一周，回到起始点的位移，区域 3（绿色阴影部分）与区域 2 一样，表示从轴循环旋转以达到等长切断主轴传送带上的被剪物体。

蓝色实心点即位置同步点，橙色方框区域为速度同步区。区域 2 曲线表示要求从轴上的切刀在与主轴上的被剪物体接触后保持一定时间的同速运行，以便转轮上的切刀切断主轴传送带上的被剪物。之后以较低速度和主轴物体分离；当切刀再次运动到临接近被切物体时，又要与主轴的速度同步，以此类推，循环运行。

如何实现以上规划曲线，现以上图的具体数字说明。

我们注意到，区域 1 和区域 2 是功能完全不同的数据段。区域 1 的数据段只是过渡段，当速度和位置到达预定值后便不再执行了，区域 2 则是需要循环执行的段，因此需要将区域 1 的数据放在一个 FIFO，区域 2 的数据放在另外一个 FIFO。

区域 1：从轴追赶主轴的位移段，当主轴走完 5000pulse 时，从轴需要走 2500pulse，如表 3-2 所示。以主轴规划位置为参考，该数据段的起点为规划 0 位置。

表 3-2　飞剪案例区域 1 的数据段

	第一段(pulse)
主轴位置	5000
从轴位置	2500

区域 2：可以分成 5 个数据段：第一段为切刀从位置同步点离开的速度同步区段；第二段为切刀减速脱离速度同步区段；第三段为从轴恒速段；第四段为从轴往主轴速度变化的加速段；最后一段是切刀接近被剪物体的速度同步区段。计算可得如表 3-3 所示。以主轴规划位置为参考，该数据段的起点为规划位置 5000pulse。

表 3-3　飞剪案例区域 2 的数据段

	第一段	第二段	第三段	第四段	第五段
主轴位置	1000	4000	6000	9000	10000
从轴位置	1000	3400	4600	7000	8000
主轴位移长度	1000	3000	2000	3000	1000
从轴位移长度	1000	2400	1200	2400	1000

1. 压入控制器的数据为位置点（相对于数据段起点位置的位移），而不是位移长度。
2. 位置点的起点都是以启动 Follow 运动（调用指令 GTN_FollowStart 之后）那一刻的位置点为零点（如设置的启动条件为 FOLLOW_EVENT_PASS，则以穿越点为零点基准）。
3. 若切换了 FIFO，位置点又是以换 FIFO 后的位置为零点。

2. Follow 单 FIFO

例程 3-2　Follow 单 FIFO 模式

该例程主轴为 Jog 模式，速度为 50pulse/ms。从轴为 Follow 模式，跟随主轴的规划位置。从轴启动的跟随条件是：主轴走过 50000pulse 后，从轴启动跟随。从轴的运动规律由 3 段组成，如表 3-4 所示，加速段跟随，匀速跟随，减速跟随，类似一个梯形曲线。并且无限次循环此数据段。主轴速度规划如图 3-5Follow 单 FIFO 模式主轴速度规划所示，从轴速度规划如图 3-6 所示。

表 3-4　Follow 单 FIFO 数据段

	第一段	第二段	第三段
主轴位置	20000	20000	20000
从轴位置	10000	20000	10000

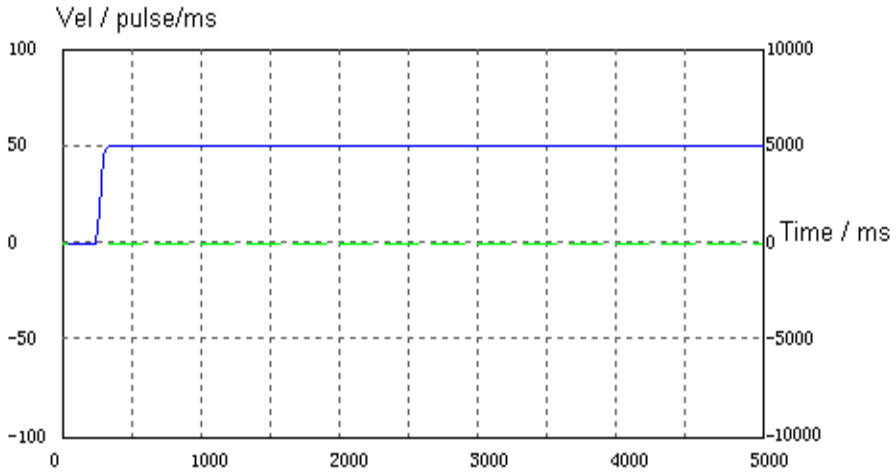

图 3-5　Follow 单 FIFO 模式主轴速度规划

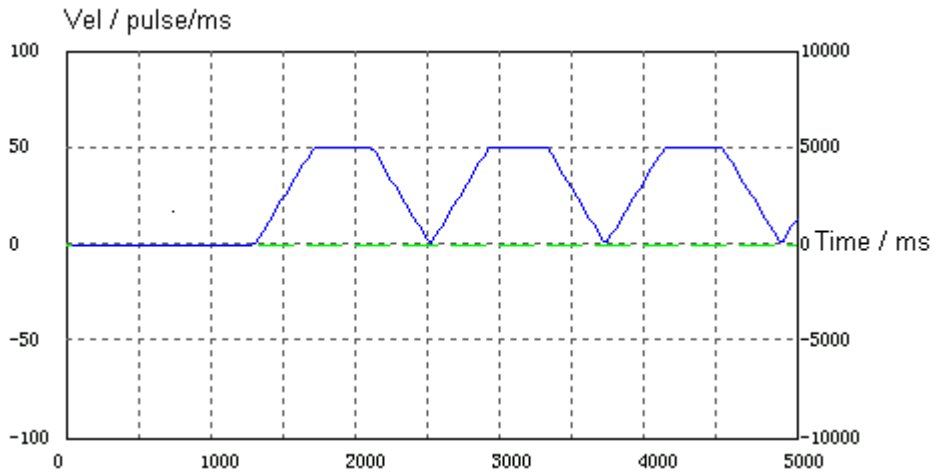

图 3-6　Follow 单 FIFO 模式从轴速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

#define MASTER          1
#define SLAVE           2

int main(int argc, char* argv[])
{
    short sRtn;
    double prfVel[8];
    TJogPrm jog;
    short space;
    long masterPos;
    double slavePos;
    long loop;

    // 打开运动控制器
    sRtn = GTN_Open();
    // 指令返回值检测，请查阅例3-1
    commandhandler("GTN_Open", sRtn);
    // 复位运动控制器
    sRtn = GTN_Reset(1);
    commandhandler("GTN_Reset", sRtn);
    // 配置运动控制器
    // 注意：配置文件 test.cfg 取消了各轴的报警和限位
    sRtn = GTN_LoadConfig(1,"test.cfg");
    commandhandler("GTN_LoadConfig", sRtn);
    // 清除各轴的报警和限位
    sRtn = GTN_ClrSts (1,1, 8);
    commandhandler("GTN_ClrSts", sRtn);
    // 伺服使能
    sRtn = GTN_AxisOn (1,MASTER);
```

```
commandhandler("GTN_AxisOn", sRtn);
sRtn = GTN_AxisOn (1,SLAVE);
commandhandler("GTN_AxisOn", sRtn);
// 位置清零
sRtn = GTN_ZeroPos (1,MASTER);
commandhandler("GTN_ZeroPos", sRtn);
sRtn = GTN_ZeroPos (1,SLAVE);
commandhandler("GTN_ZeroPos", sRtn);
// 将主轴设为 Jog 模式
sRtn = GTN_PrfJog (1,MASTER);
commandhandler("GTN_PrfJog", sRtn);
// 设置主轴运动参数
sRtn = GTN_GetJogPrm (1,MASTER, &jog);
commandhandler("GTN_GetJogPrm", sRtn);
jog.acc = 1;
sRtn = GTN_SetJogPrm (1,MASTER, &jog);
commandhandler("GTN_SetJogPrm", sRtn);
sRtn = GTN_SetVel (1,MASTER, 50);
commandhandler("GTN_SetVel", sRtn);
// 启动主轴
sRtn = GTN_Update (1,1<<(MASTER-1));
commandhandler("GTN_Update", sRtn);
// 将从轴设为 Follow 模式
sRtn = GTN_PrfFollow(1,SLAVE);
commandhandler("GTN_PrfFollow", sRtn);
// 清空从轴 FIFO
sRtn = GTN_FollowClear(1,SLAVE);
commandhandler("GTN_FollowClear", sRtn);
// 设置主轴，默认跟随主轴规划位置
sRtn = GTN_SetFollowMaster(1,SLAVE, MASTER);
commandhandler("GTN_SetFollowMaster", sRtn);
// 查询 Follow 模式的剩余空间
sRtn = GTN_FollowSpace(1,SLAVE, &space);
printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
// 向 FIFO 中增加运动数据
masterPos = 20000;
slavePos = 10000;
sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos);
commandhandler("GTN_FollowData", sRtn);
// 查询 Follow 模式的剩余空间
sRtn = GTN_FollowSpace(1,SLAVE, &space);
printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
// 向 FIFO 中增加运动数据
masterPos+= 20000;
slavePos += 20000;
sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos);
```

```
        commandhandler("GTN_FollowData",sRtn);
        // 查询 Follow 模式的剩余空间
        sRtn = GTN_FollowSpace(1,SLAVE, &space);
        printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
        // 向 FIFO 中增加运动数据
        masterPos += 20000;
        slavePos += 10000;
        sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos);
        commandhandler("GTN_FollowData", sRtn);
        // 设置循环次数为无限循环
        sRtn = GTN_SetFollowLoop(1,SLAVE, 0);
        commandhandler("GTN_SetFollowLoop", sRtn);
        // 设置启动跟随条件
        sRtn = GTN_SetFollowEvent(1,SLAVE, FOLLOW_EVENT_PASS, 1, 50000);
        commandhandler("GTN_SetFollowEvent", sRtn);
        // 启动从轴 Follow 运动
        sRtn = GTN_FollowStart(1,1<<(SLAVE-1));
        commandhandler("GTN_FollowStart", sRtn);

        while(!kbhit())
        {
            // 查询各轴的规划速度
            sRtn = GTN_GetPrfVel (1,1, prfVel, 8);
            // 查询循环次数
            sRtn = GTN_GetFollowLoop(SLAVE, &loop);
            printf("master=%-10.2lf\tslave=%-10.2lf\tloop=%d\r",
                prfVel[MASTER-1], prfVel[SLAVE-1], loop);
        }

        // 伺服关闭
        sRtn = GTN_AxisOff(1,MASTER);
        printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, MASTER);
        sRtn = GTN_AxisOff(1,SLAVE);
        printf("\nGTN_AxisOff()=%d\n", sRtn, SLAVE);
        getch();
        return 0;
}
```

3. Follow 双 FIFO 切换

例程 3-3　Follow 双 FIFO 切换

该例程主轴为 Jog 模式，速度为 50pulse/ms。从轴为 Follow 模式，跟随主轴的规划位置。从轴启动的跟随条件是：从轴在调用指令 GTN_FollowStart 后立即启动跟随。从轴在运动时更换跟随策略，其速度规划经过一个过渡的数据段，然后变成一个新的梯形曲线，并且无限次循环。如下面三个表所示。我们把表 3-5 中的数据写入 FIFO1 中，把表 3-6 中的数据写入 FIFO2 中，在运动过程中切换到 FIFO2，同时在检查并确认 FIFO1 被控制器自动清空之后，将表 3-7 中的数据写入 FIFO1 中，

并切换运动 FIFO1 的数据。这样即可利用双 FIFO 切换完成跟随策略的更换。主轴速度规划如图 3-7 所示，从轴速度规划如图 3-8 所示。

表 3-5　Follow 双 FIFO 切换之原来的跟随策略

	第一段	第二段	第三段
主轴位置	20000	20000	20000
从轴位置	10000	20000	10000

表 3-6　Follow 双 FIFO 切换之更换跟随策略时的过渡段

	第一段	第二段	第三段
主轴位置	20000	20000	20000
从轴位置	10000	20000	16000

表 3-7　Follow 双 FIFO 切换之更换后的跟随策略

	第一段	第二段	第三段
主轴位置	20000	20000	20000
从轴位置	16000	20000	16000

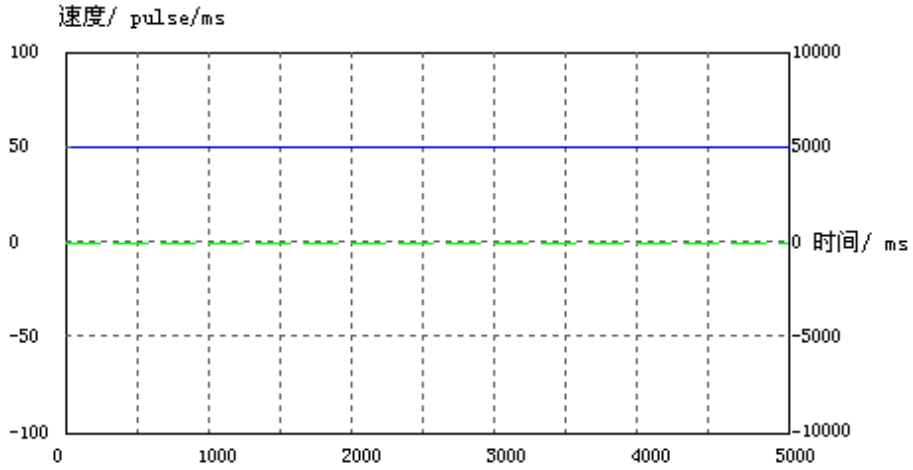

图 3-7　Follow 双 FIFO 切换主轴速度规划

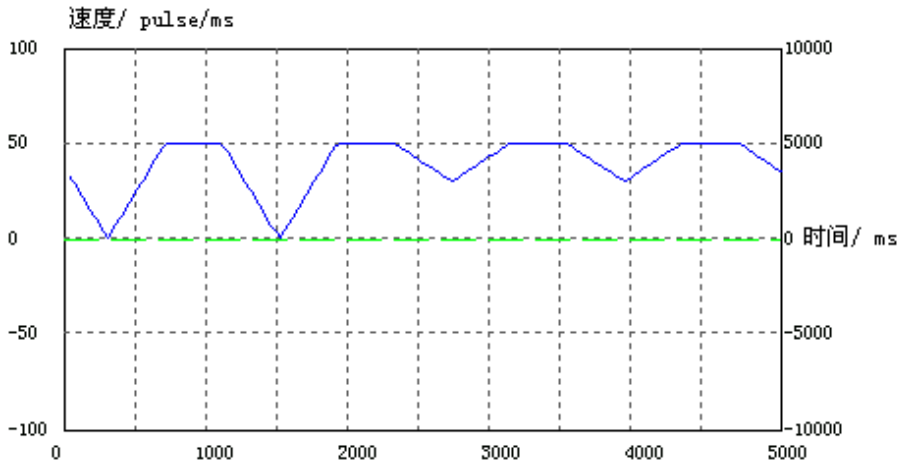

图 3-8　Follow 双 FIFO 切换从轴速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

#define MASTER                  1
#define SLAVE                   2

#define STAGE_FIFO1             1
#define STAGE_TO_FIFO2          2
#define STAGE_TO_FIFO1          3
#define STAGE_END               4

int main(int argc, char* argv[])
{
    short sRtn;
    double prfVel[8];
    TJogPrm jog;
    short space;
    long masterPos;
    double slavePos;
    short stage, pressKey;

    // 打开运动控制器
    sRtn = GTN_Open();
    // 指令返回值检测，请查阅例3-1
    commandhandler("GTN_Open", sRtn);
    // 复位运动控制器
    sRtn = GTN_Reset (1);
    commandhandler("GTN_Reset ", sRtn);
    // 配置运动控制器
    // 注意：配置文件test.cfg取消了各轴的报警和限位
    sRtn = GTN_LoadConfig (1,"test.cfg");
    commandhandler("GTN_LoadConfig ", sRtn);
    // 清除各轴的报警和限位
    sRtn = GTN_ClrSts(1,1, 8);
    commandhandler("GTN_ClrSts", sRtn);
    // 伺服使能
    sRtn = GTN_AxisOn (1,MASTER);
    commandhandler("GTN_AxisOn", sRtn);
    sRtn = GTN_AxisOn (1,SLAVE);
    commandhandler("GTN_AxisOn", sRtn);
    // 位置清零
    sRtn = GTN_ZeroPos (1,MASTER);
    commandhandler("GTN_ZeroPos", sRtn);
    sRtn = GTN_ZeroPos (1,SLAVE);
    commandhandler("GTN_ZeroPos", sRtn);
```

```
// 将主轴设为Jog模式
sRtn = GTN_PrfJog (1,MASTER);
commandhandler("GTN_PrfJog", sRtn);
// 设置主轴运动参数
sRtn = GTN_GetJogPrm (1,MASTER, &jog);
commandhandler("GTN_GetJogPrm", sRtn);
jog.acc = 1;
sRtn = GTN_SetJogPrm (1,MASTER, &jog);
commandhandler("GTN_SetJogPrm", sRtn);
sRtn = GTN_SetVel (1,MASTER, 50);
commandhandler("GTN_SetVel", sRtn);
// 启动主轴
sRtn = GTN_Update (1,1<<(MASTER-1));
commandhandler("GTN_Update", sRtn);
// 将从轴设为Follow模式
sRtn = GTN_PrfFollow(1,SLAVE);
commandhandler("GTN_PrfFollow", sRtn);
// 清空从轴FIFO1
sRtn = GTN_FollowClear(1,SLAVE, 0);
commandhandler("GTN_FollowClear", sRtn);
// 清空从轴FIFO2
sRtn =GTN_FollowClear(1,SLAVE, 1);
commandhandler("GTN_FollowClear", sRtn);
// 设置主轴，默认跟随主轴规划位置
sRtn = GTN_SetFollowMaster(1,SLAVE, MASTER);
commandhandler("GTN_SetFollowMaster", sRtn);
// 查询Follow模式FIFO1的剩余空间
sRtn = GTN_FollowSpace(1,SLAVE, &space, 0);
printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
// 向FIFO1中增加运动数据
masterPos = 20000;
slavePos = 10000;
sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos, FOLLOW_SEGMENT_NORMAL, 0);
commandhandler("GTN_FollowData", sRtn);
// 查询Follow模式FIFO1的剩余空间
sRtn = GTN_FollowSpace(1,SLAVE, &space, 0);
printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
// 向FIFO1中增加运动数据
masterPos += 20000;
slavePos   += 20000;
sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos, FOLLOW_SEGMENT_NORMAL, 0);
commandhandler("GTN_FollowData", sRtn);
// 查询Follow模式FIFO1的剩余空间
sRtn = GTN_FollowSpace(1,SLAVE, &space, 0);
printf("GTN_FollowSpace()=%d space=%d\n", sRtn, space);
// 向FIFO1中增加运动数据
```

```
masterPos += 20000;
slavePos   += 10000;
sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos, FOLLOW_SEGMENT_NORMAL, 0);
commandhandler("GTN_FollowData", sRtn);
// 设置从轴循环次数为无限循环
sRtn = GTN_SetFollowLoop(1,SLAVE, 0);
commandhandler("GTN_SetFollowLoop", sRtn);

// 设置启动跟随条件
sRtn = GTN_SetFollowEvent(1,SLAVE, FOLLOW_EVENT_START, 1);
commandhandler("GTN_SetFollowEvent", sRtn);
// 启动从轴Follow运动
sRtn = GTN_FollowStart(1,1<<(SLAVE-1));
commandhandler("GTN_FollowStart", sRtn);
stage = STAGE_FIFO1;
pressKey = 0;

while(1)
{
    // 检查是否有按键
    if(kbhit())
    {
        getch();
        pressKey = 1;
    }
    // 如果当前运行FIFO1中的数据并且检测到按键，则切换运行FIFO2中的数据
    if(STAGE_FIFO1 == stage )
    {
        if( 1 == pressKey )
        {
            pressKey = 0;
            stage = STAGE_TO_FIFO2;
            // 向FIFO2中发送过渡数据
            sRtn = GTN_FollowClear(1,SLAVE, 1);
            masterPos = 20000;
            slavePos   = 10000;
            sRtn =GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                 FOLLOW_SEGMENT_CONTINUE, 1);
            masterPos+= 20000;
            slavePos += 20000;
            sRtn =GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                 FOLLOW_SEGMENT_NORMAL, 1);
            masterPos+= 20000;
            slavePos += 16000;
            sRtn =GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                 FOLLOW_SEGMENT_NORMAL, 1);
```

```
            // 切换到FIFO2
            // 当前工作FIFO中的数据遍历完以后才会切换FIFO
            sRtn =GTN_FollowSwitch(1,1<<(SLAVE-1));
        }
    }

    // 检查 FIFO1 是否被清空，如果已被清空，则将新的速度规划传入 FIFO1 中，并且切换运行 FIFO1 中数据
    if(STAGE_TO_FIFO2 == stage )
    {
        // 查询 FIFO1 的剩余空间
        GTN_FollowSpace(1,SLAVE, &space, 0);
        // 如果 FIFO1 被清空，说明已经切换到 FIFO2
        if( 16 == space )
        {
            stage = STAGE_TO_FIFO1;
            masterPos = 20000;
            slavePos   = 16000;
            sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                  FOLLOW_SEGMENT_CONTINUE, 0);
            masterPos+= 20000;
            slavePos += 20000;
            sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                  FOLLOW_SEGMENT_NORMAL, 0);
            masterPos+= 20000;
            slavePos += 16000;
            sRtn = GTN_FollowData(1,SLAVE, masterPos, slavePos,
                                  FOLLOW_SEGMENT_NORMAL, 0);
            // 切换到 FIFO1
            // 当前工作 FIFO 遍历完以后才会切换 FIFO
            sRtn = GTN_FollowSwitch(1,1<<(SLAVE-1));
        }
    }

    if(STAGE_TO_FIFO1 == stage )
    {
        // 查询 FIFO2 的剩余空间
        GTN_FollowSpace(1,SLAVE, &space, 1);
        // 如果 FIFO2 被清空，说明已经切换到 FIFO1
        if( 16 == space )
        {
            stage = STAGE_END;
        }
    }

    // 查询各轴的规划速度
```

```
            sRtn = GTN_GetPrfVel (1,1, prfVel, 8);
            printf("master=%-10.2lf\tslave=%-10.2lf\r",
            prfVel[MASTER-1], prfVel[SLAVE-1]);

            if(STAGE_END == stage )
            {
                if( 1 == pressKey )
                {
                    pressKey = 0;
                    break;
                }
            }
        }

        // 伺服关闭
        sRtn = GTN_AxisOff(1,MASTER);
        printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, MASTER);
        sRtn = GTN_AxisOff(1,SLAVE);
        printf("\nGTN_AxisOff()=%d\n", sRtn, SLAVE);
        getch();
        return 0;
}
```

第4章　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 GTN_FollowData）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《GTN 系列运动控制器编程手册之基本功能》。

4.1 指令列表

表 4-1　PVT 运动模式指令列表

指令	说明	页码
GTN_PrfPvt	设置指定轴为 PVT 运动模式	61
GTN_SetPvtLoop	设置 PVT 运动模式循环次数	71
GTN_GetPvtLoop	查询 PVT 运动模式循环次数	60
GTN_PvtTable	向 PVT 运动模式指定数据表传送数据，采用 PVT 描述方式	66
GTN_PvtTableComplete	向 PVT 运动模式指定数据表传送数据，采用 Complete 描述方式	66
GTN_PvtTablePercent	向 PVT 运动模式指定数据表传送数据，采用 Percent 描述方式	67
GTN_PvtPercentCalculate	计算 PVT 运动模式 Percent 描述方式下各数据点的速度	64
GTN_PvtTableContinuous	向 PVT 运动模式指定数据表传送数据，采用 Continuous 描述方式	67
GTN_PvtContinuousCalculate	计算 PVT 运动模式 Continuous 描述方式下各数据点的时间	64
GTN_PvtTableSelect	选择 PVT 运动模式数据表	68
GTN_PvtStart	启动 PVT 运动	65
GTN_PvtStatus	读取 PVT 运动状态	65

4.2 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 GTN_PvtTable、GTN_PvtTableComplete、GTN_PvtTablePercent 或 GTN_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 GTN_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 GTN_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 GTN_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。

假设有如表 4-2 所示的 4 个数据点，采用 PVT 方式进行描述。

表 4-2　用 PVT 方式描述的数据点

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

- 调用 GTN_PrfPvt 将轴切换到 PVT 模式
- 调用 GTN_PvtTable 将 4 个数据点传递到数据表
- 调用 GTN_SetPvtLoop 设置为循环执行
- 调用 GTN_PvtStart 启动运动

由于 P1 的时间为 1000 毫秒，因此调用 GTN_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 4-1 所示。

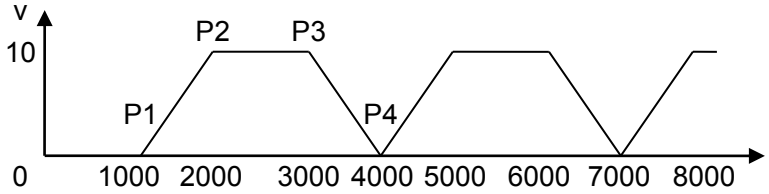

图 4-1　循环执行数据表

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 4-3 所示的 4 组数据点，采用 PVT 描述方式。

表 4-3　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 4-2 合理的 PVT 描述方式运动规律所示。

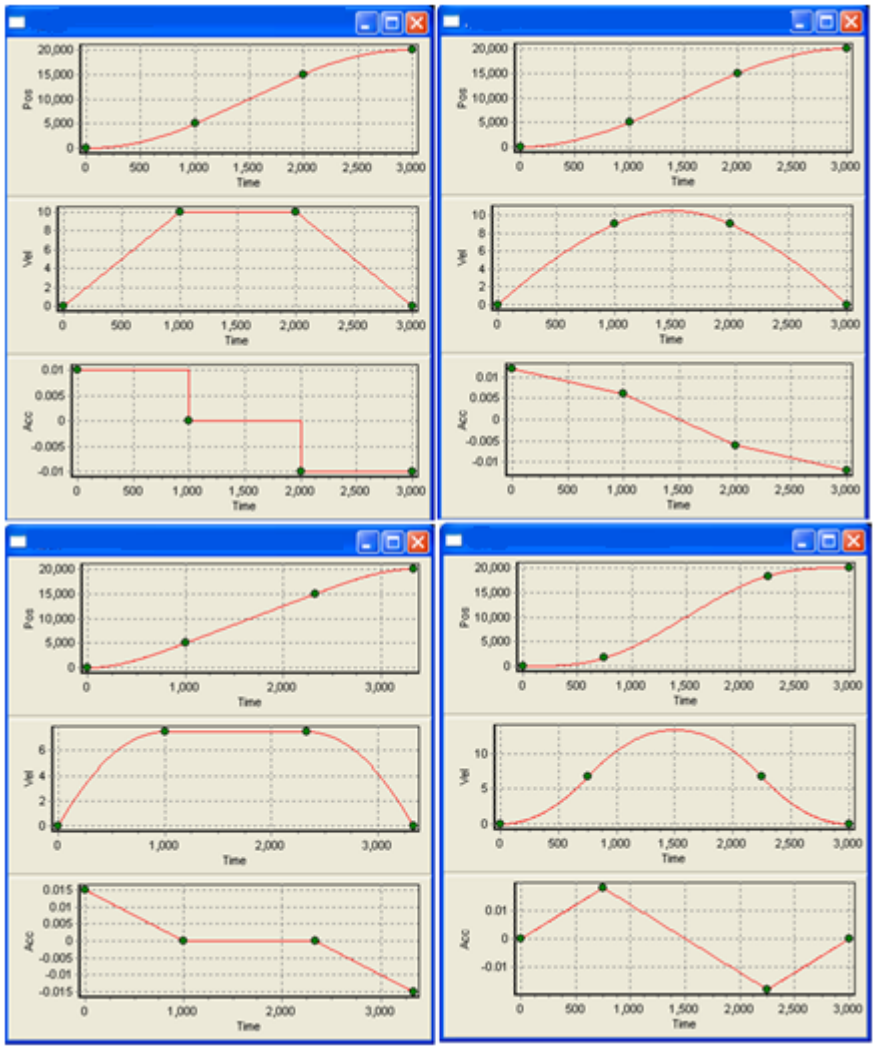

图 4-2　合理的 PVT 描述方式运动规律

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 4-4 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 4-4　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 4-3 所示。

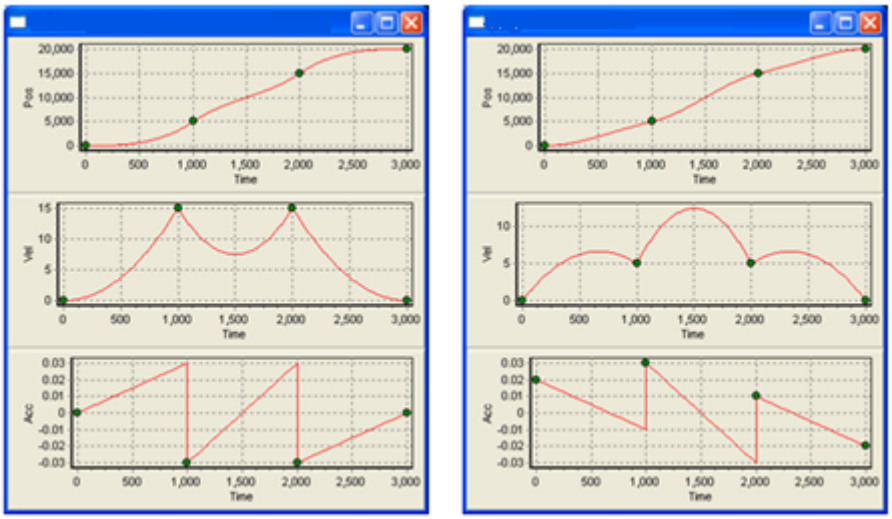

图 4-3　不合理的 PV 描述方式运动规律

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 4-5 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 4-5　Complete 描述方式的一组数据点

数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
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 4-4 所示。

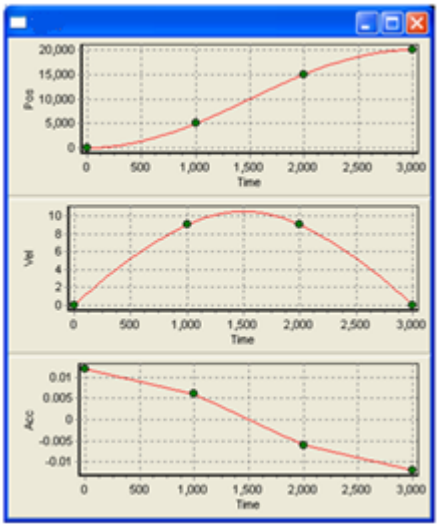

图 4-4　Complete 描述方式运动规律

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 4-6 所

示。

表 4-6　Complete 方式描述三角函数的数据点

数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
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 4-5 所示。

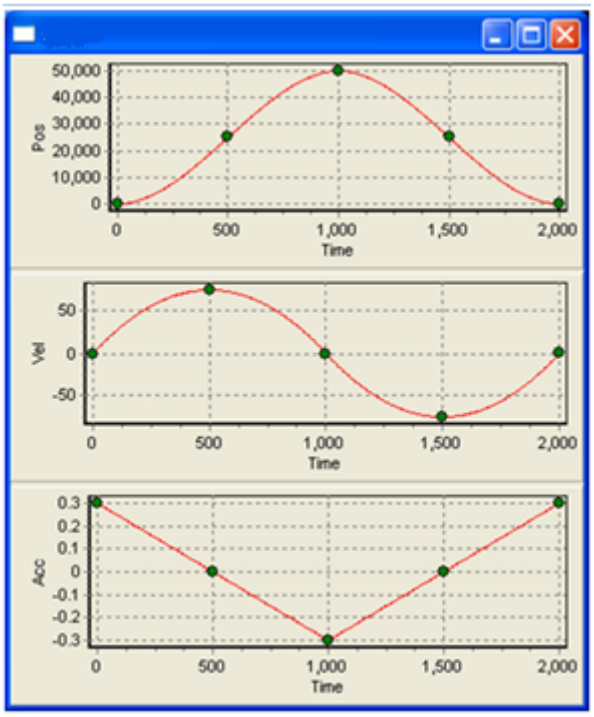

图 4-5　Complete 方式描述三角函数运动规律

增加数据点可以减小与函数 P=50000sin^2 (π/2000*t)的逼近误差。下图 4-6 给出了数据点数为 5、10、50 时的位置误差。

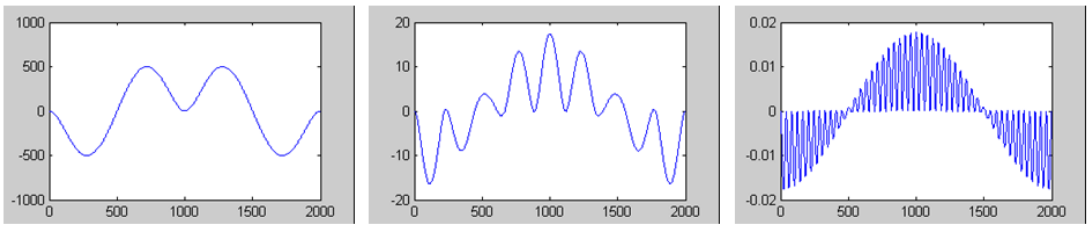

图 4-6　Complete 方式下数据点数分别为 5、10、50 时的位置误差

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”。以下图 4-7 为例来进行说明。

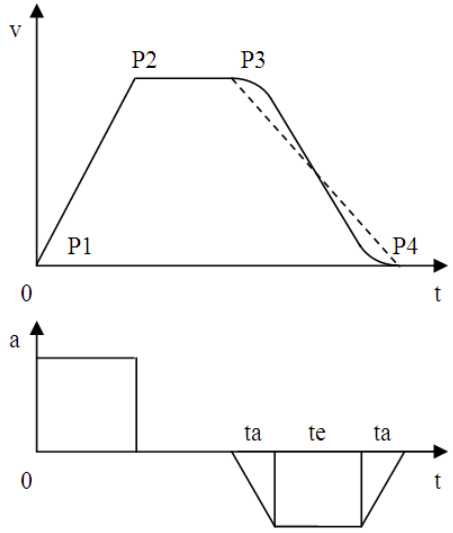

图 4-7　Percent 描述方式下的百分比定义

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 4-7 所示的这组数据点，采用 Percent 描述方式。

表 4-7　Percent 描述方式下的数据点

数据点	时间（ms）	位置（pulse）	百分比	速度（pulse/ms）
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 4-8 所示。

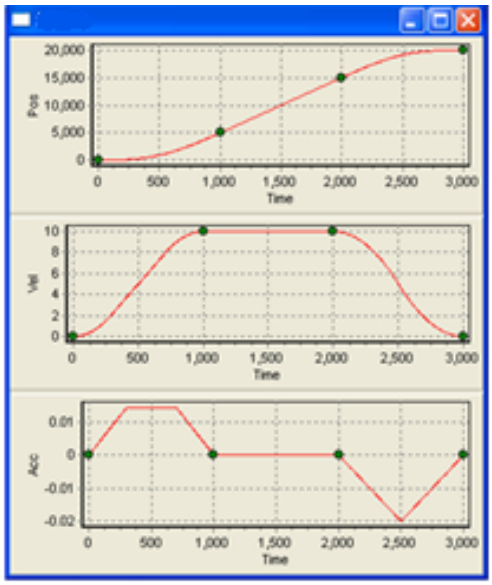

图 4-8　Percent 描述方式下的运动规律

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，下图 4-9 示例了一些可能的分段情况。

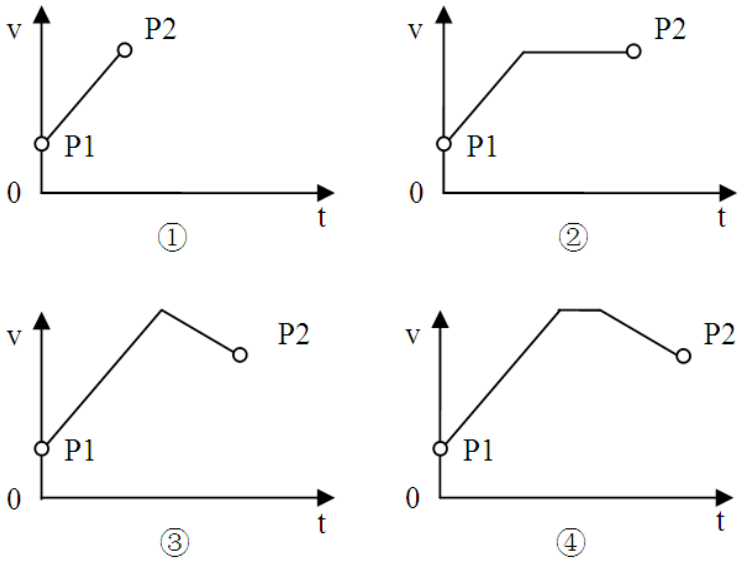

图 4-9　Continuous 描述方式

例如表 4-8 所示的这两组数据点，采用 Continuous 描述方式。

表 4-8　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 4-10 所示。

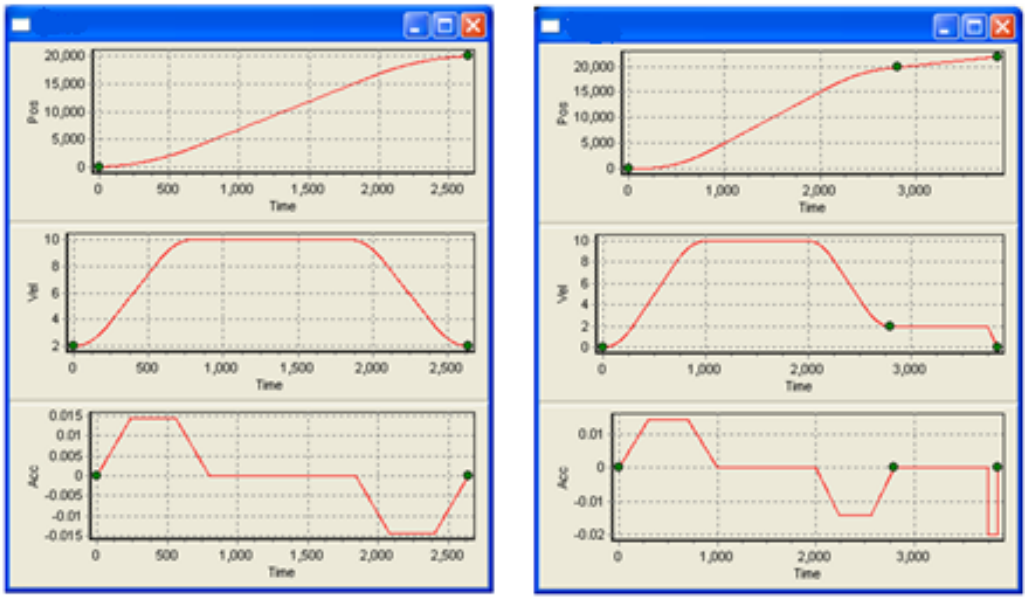

图 4-10　Continuous 描述方式下的运动规律

4.3 例程

1. PVT 描述方式

如图 4-11 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。

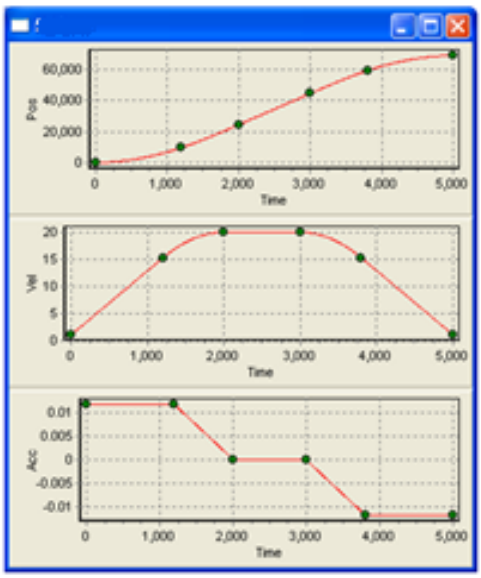

图 4-11　PVT 例程描述方式下的运动规律

可以满足上述要求的一组数据表如表 4-9 所示。

表 4-9　PVT 例程描述方式下的数据点

数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
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 4-1　PVT 描述方式

```
#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "gts.h"

#define AXIS                1
#define TABLE               1

int main(int argc, char* argv[])
{
    short sRtn;
    long mask;
    // X轴的数据点参数
    double time[6]={0, 1200, 2000, 3000, 3800, 5000};
    double pos[6]={0, 9750, 24483, 44483, 59216, 68966};
    double vel[6]={1, 15.25, 20, 20, 15.25, 1};
```

```
double prfVel, prfPos, t;
short tableId;

// 打开运动控制器
sRtn = GTN_Open ();
commandhandler("GTN_Open ", sRtn);
// 复位运动控制器
sRtn = GTN_Reset (1);
commandhandler("GTN_Reset ", sRtn);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn = GTN_LoadConfig (1,"test.cfg");
commandhandler("GTN_LoadConfig ", sRtn);
// 清除各轴报警和限位
sRtn = GTN_ClrSts (1,1, 8);
commandhandler("GTN_ClrSts", sRtn);
sRtn = GTN_AxisOn (1,AXIS);
commandhandler("GTN_AxisOn", sRtn);
// 等待伺服使能就绪
Sleep(1000);
// 设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS);
commandhandler("GTN_PrfPvt", sRtn);
// 发送数据
sRtn = GTN_PvtTable(1,TABLE, 6, &time[0], &pos[0], &vel[0]);
commandhandler("GTN_PvtTable", sRtn);
// 选择数据表
sRtn = GTN_PvtTableSelect(1,AXIS, TABLE);
commandhandler("GTN_PvtTableSelect", sRtn);
mask = 1<<(AXIS-1);
sRtn = GTN_PvtStart(1,mask);
commandhandler("GTN_PvtStart", sRtn);
while(!kbhit())
{
    // 读取数据表和运动时间
    sRtn = GTN_PvtStatus(1,AXIS, &tableId, &t);
    // 读取规划速度
    sRtn = GTN_GetPrfVel (1,AXIS, &prfVel);
    // 读取规划位置
    sRtn =GTN_GetPrfPos (1,AXIS, &prfPos);
    printf("%2d %10.0lf %10.2lf %10.1lf\r", tableId, t, prfVel, prfPos);
}
// 伺服关闭
sRtn = GTN_AxisOff(1,AXIS);
printf("\nGTN_AxisOff()=%d\n", sRtn);
getch();
```

```
    return 0;
}
```

2. Complete 描述方式

假设位置和时间之间的关系由函数 P=40000sin² (π/2000*t)确定。要求启动以后能够循环运动，按 A 键幅值增大 50%，按 B 键幅值减小 50%，Complete 描述方式下的速度曲线如图 4-12 所示。

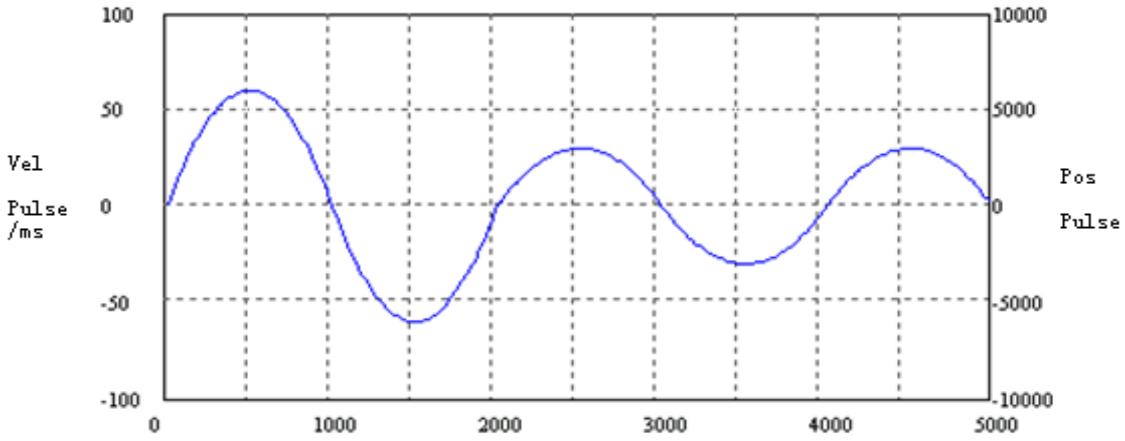

图 4-12　Complete 描述方式下的速度曲线

例程 4-2　Complete 描述方式

```
#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "math.h"
#include "stdlib.h"
#include "gts.h"

#define AXIS        1
#define TABLE1      1
#define TABLE2      2
#define PI          3.1415926

void Calculate(double amplitude, long n, double *pTime, double *pPos)
{
    long i;
    for(i=0;i<n;++i)
    {
        pPos[i] = amplitude*sin(PI/2000*pTime[i])*sin(PI/2000*pTime[i]);
    }
}

int main(int argc, char* argv[])
{
    short sRtn;
```

```
long mask;
// X轴的数据点参数
double time[5]={0, 500, 1000, 1500, 2000};
double pos[5];
double a[5], b[5], c[5];
double prfVel, prfPos, t;
short tableId;
double amplitude = 40000;
short table = TABLE1;
char key;

// 打开运动控制器
sRtn = GTN_Open ();
commandhandler("GTN_Open ", sRtn);
// 复位运动控制器
sRtn = GTN_Reset (1);
commandhandler("GTN_Reset ", sRtn);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn = GTN_LoadConfig (1,"test.cfg");
commandhandler("GTN_LoadConfig ", sRtn);
// 清除各轴报警和限位
sRtn = GTN_ClrSts(1,1, 8);
commandhandler("GTN_ClrSts", sRtn);
sRtn = GTN_AxisOn (1,AXIS);
commandhandler("GTN_AxisOn", sRtn);
// 等待伺服使能就绪
Sleep(1000);
// 设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS);
commandhandler("GTN_PrfPvt", sRtn);
Calculate(amplitude, 5, &time[0], &pos[0]);
// 发送数据
sRtn = GTN_PvtTableComplete(1,table, 5, &time[0], &pos[0], &a[0], &b[0], &c[0], 0, 0);
commandhandler("GTN_PvtTableComplete", sRtn);
// 选择数据表
sRtn = GTN_PvtTableSelect(1,AXIS, table);
commandhandler("GTN_PvtTableSelect", sRtn);
// 设置为循环执行
sRtn = GTN_SetPvtLoop(1,AXIS, 0);
commandhandler("GTN_SetPvtLoop", sRtn);
mask = 1<<(AXIS-1);
sRtn = GTN_PvtStart(1,mask);
commandhandler("GTN_PvtStart", sRtn);

while(1)
```

```
{
    // 读取数据表和运动时间
    sRtn = GTN_PvtStatus(1,AXIS, &tableId, &t);
    // 读取规划速度
    sRtn = GTN_GetPrfVel (1,AXIS, &prfVel);
    // 读取规划位置
    sRtn = GTN_GetPrfPos(1,AXIS, &prfPos);
    if(kbhit() )
    {
        key = getch();
        if('A' == toupper(key) )
        {
            amplitude *= 1.5;
        }
        if('B' == toupper(key) )
        {
            amplitude *= 0.5;
        }
        if( ( 'A' == toupper(key) ) || ( 'B' == toupper(key) ) )
        {
            Calculate(amplitude, 5, &time[0], &pos[0]);
            table = TABLE1 + TABLE2 - tableId;
            // 发送数据
            sRtn = GTN_PvtTableComplete(1,table, 5, &time[0], &pos[0], &a[0],
                                        &b[0], &c[0], 0, 0);
            if( 0 != sRtn )
            {
                commandhandler("GTN_PvtTableComplete", sRtn);
                exit(0);
            }
            // 选择数据表
            sRtn = GTN_PvtTableSelect(1,AXIS, table);
            if( 0 != sRtn )
            {
                commandhandler("\nGTN_PvtTableSelect", sRtn);
                exit(0);
            }
        }

        if('Q' == toupper(key) )
        {
            mask = 1 << (AXIS-1);
            GT_Stop(1,mask, 0);
            exit(0);
        }
    }
```

```
            printf("%2d %10.0lf %10.2lf %10.1lf\r", tableId, t, prfVel, prfPos);
        }
        // 伺服关闭
        sRtn = GTN_AxisOff(1,AXIS);
        printf("\nGTN_AxisOff ()=%d\n", sRtn);
        getch();
        return 0;
}
```

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 4-13 所示。

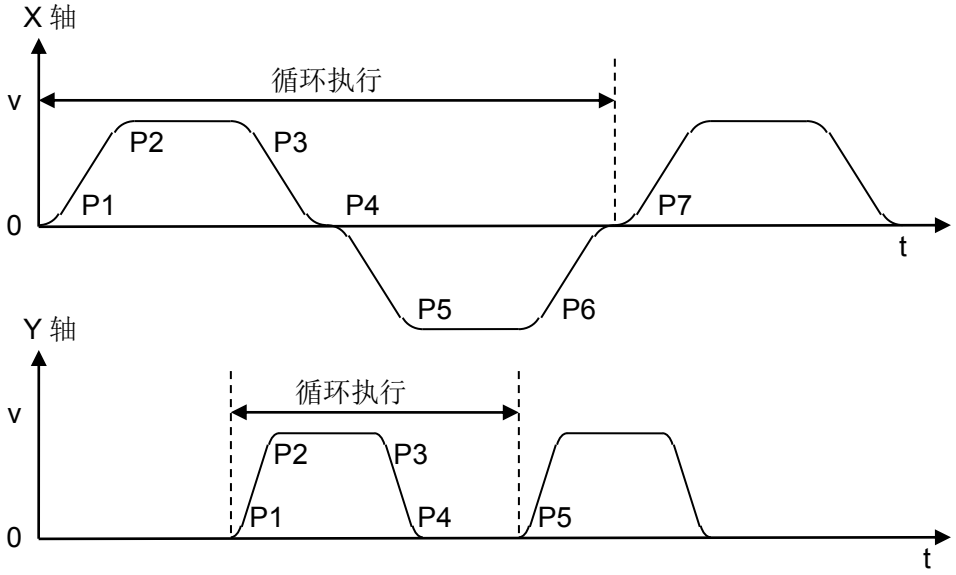

图 4-13　Percent 描述方式下 X 轴和 Y 轴的运动规律

X 轴取 7 个数据点，设置为循环模式。数据点参数如表 4-10 和表 4-11 所示。

表 4-10　Percent 描述方式下的数据点 1

数据点	时间（ms）	位置（pulse）	百分比	速度（pulse/ms）
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 4-11　Percent 描述方式下的数据点 2

数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
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 4-12 和表 4-13 所示。

表 4-12　Percent 描述方式下的数据点 3

数据点	时间（ms）	位置（脉冲）	百分比	速度（pulse/ms）
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 4-13　Percent 描述方式下的数据点 4

数据点	时间（ms）	位置（pulse）	速度（pulse/ms）
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。下图是当 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。实际的运动轨迹如图 4-14 所示。

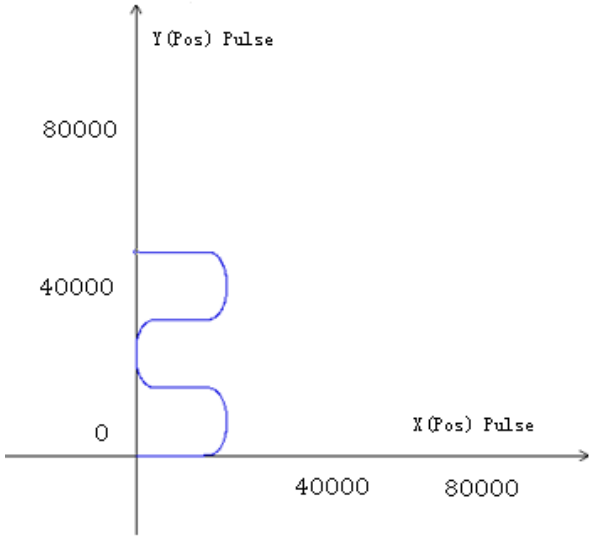

图 4-14　Percent 描述方式下的 X-Y 位置图

例程 4-3　Percent 描述方式

```
#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "gts.h"

#define AXIS_X          1
#define AXIS_Y          2
#define TABLE_X         1
#define TABLE_Y         2
#define LOOP_COUNT 2

int main(int argc, char* argv[])
{
    short sRtn;
    long mask;
    // X轴的数据点参数
    double time_x[7] = {0, 1000, 2000, 3000, 4000, 5000, 6000};
    double pos_x[7] = {0, 5000, 15000, 20000, 15000, 5000, 0};
    double percent_x[7] = {60, 0, 60, 60, 0, 60, 0};
    // Y轴的数据点参数
    double time_y[5] = {2000, 2500, 3500, 4000, 5000};
    double pos_y[5] = {0, 2500, 12500, 15000, 15000};
    double percent_y[5] = {60, 0, 60, 0, 0};
    double prfVel[2], prfPos[2], time[2];
    short tableId[2];

    // 打开运动控制器
    sRtn = GTN_Open();
    commandhandler("GTN_Open", sRtn);
```

```
// 复位运动控制器
sRtn = GTN_Reset(1);
commandhandler("GTN_Reset", sRtn);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn = GTN_LoadConfig(1,"test.cfg");
commandhandler("GTN_LoadConfig", sRtn);
// 清除各轴报警和限位
sRtn = GTN_ClrSts (1,1, 8);
commandhandler("GTN_ClrSts", sRtn);
sRtn = GTN_AxisOn (1,AXIS_X);
commandhandler("GTN_AxisOn", sRtn);
sRtn = GTN_AxisOn (1,AXIS_Y);
commandhandler("GTN_AxisOn", sRtn);
// 等待伺服使能就绪
Sleep(1000);
// 将X轴设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS_X);
commandhandler("GTN_PrfPvt", sRtn);
// 将Y轴设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS_Y);
commandhandler("GTN_PrfPvt", sRtn);
// 向X轴的数据表发送数据
sRtn = GTN_PvtTablePercent(1,TABLE_X, 7, &time_x[0], &pos_x[0], &percent_x[0], 0);
commandhandler("GTN_PvtTablePercent", sRtn);
// 向Y轴的数据表发送数据
sRtn = GTN_PvtTablePercent(1,TABLE_Y, 5, &time_y[0], &pos_y[0], &percent_y[0], 0);
commandhandler("GTN_PvtTablePercent", sRtn);
// X轴选择数据表TABLE_X
sRtn = GTN_PvtTableSelect(1,AXIS_X, TABLE_X);
commandhandler("GTN_PvtTableSelect", sRtn);
// Y轴选择数据表TABLE_Y
sRtn = GTN_PvtTableSelect(1,AXIS_Y, TABLE_Y);
commandhandler("GTN_PvtTableSelect", sRtn);
// 设置循环次数
sRtn = GTN_SetPvtLoop(1,AXIS_X, LOOP_COUNT);
commandhandler("GTN_SetPvtLoop", sRtn);
// 设置循环次数
sRtn = GTN_SetPvtLoop(1,AXIS_Y, 2*LOOP_COUNT-1);
commandhandler("GTN_SetPvtLoop", sRtn);
// 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 = 1<<(AXIS_X-1);
mask = 1<<(AXIS_Y-1);
sRtn = GTN_PvtStart(1,mask);
```

```
    commandhandler("GTN_PvtStart", sRtn);

    while(!kbhit())
    {
        // 读取数据表和运动时间
        sRtn = GTN_PvtStatus(1,AXIS_X, &tableId[0], &time[0]);
        sRtn = GTN_PvtStatus(1,AXIS_Y, &tableId[1], &time[1]);
        // 读取规划速度
        sRtn = GTN_GetPrfVel (1,AXIS_X, &prfVel[0]);
        sRtn = GTN_GetPrfVel (1,AXIS_Y, &prfVel[1]);
        // 读取规划位置
        sRtn = GTN_GetPrfPos (1,AXIS_X, &prfPos[0]);
        sRtn = GTN_GetPrfPos (1,AXIS_Y, &prfPos[1]);
        printf("x:%2d %10.0lf %10.2lf %10.1lf y:%2d %10.0lf %10.2lf %10.1lf\r",
            tableId[0], time[0], prfVel[0], prfPos[0], tableId[1], time[1], prfVel[1], prfPos[1]);
    }
    // 伺服关闭
    sRtn = GTN_AxisOff (1,AXIS_X);
    printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_X);
    sRtn = GTN_AxisOff (1,AXIS_Y);
    printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_Y);
    getch();
    return 0;
}
```

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 GTN_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 4-15 所示。

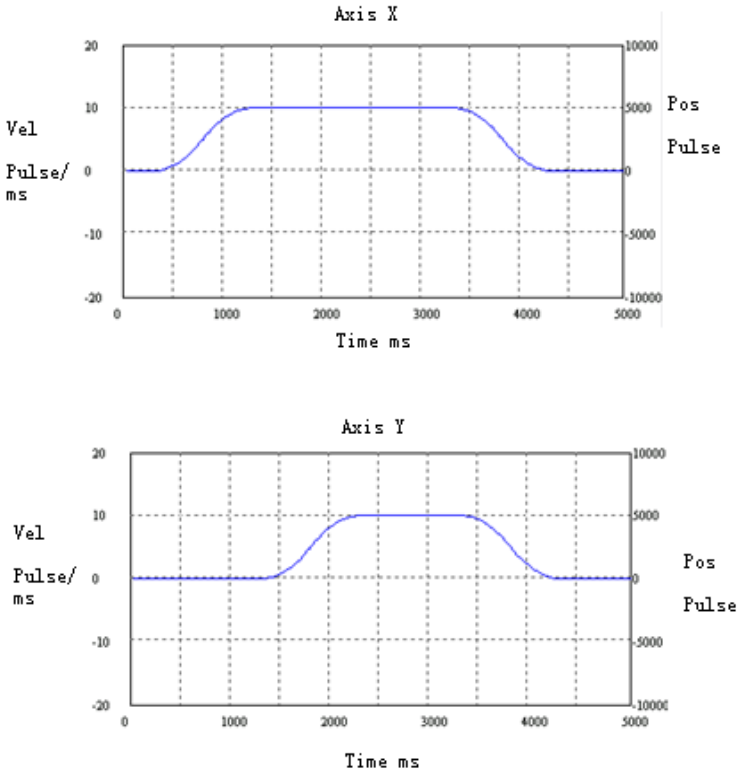

图 4-15　Continuous 描述方式下的 X 轴和 Y 轴速度曲线

例程 4-4　Continuous 描述方式

```
#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "gts.h"

#define AXIS_X          1
#define AXIS_Y          2
#define TABLE_X         1
#define TABLE_Y         2

int main(int argc, char* argv[])
{
    short sRtn;
    long mask;
    // X轴的数据点参数
    double pos_x[2] = {0, 30000};
    double vel_x[2] = {0, 0};
    double percent_x[2] = {100, 100};
    doublevelMax_x[2] = {10, 10};
    double acc_x[2] = {0.01, 0.01};
    double dec_x[2] = {0.01, 0.01};
    double time_x[2];
    double timeBegin_x;
```

```
// Y轴的数据点参数
double pos_y[2] = {0, 20000};
double vel_y[2] = {0, 0};
double percent_y[2] = {100, 100};
double velMax_y[2] = {10, 10};
double acc_y[2] = {0.01, 0.01};
double dec_y[2] = {0.01, 0.01};
double time_y[2];
double timeBegin_y;
double prfVel[2], prfPos[2], time[2];
short tableId[2];

// 打开运动控制器
sRtn = GTN_Open();
commandhandler("GTN_Open", sRtn);
// 复位运动控制器
sRtn = GTN_Reset (1);
commandhandler("GTN_Reset", sRtn);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn = GTN_LoadConfig(1,"test.cfg");
commandhandler("GTN_LoadConfig", sRtn);
// 清除各轴报警和限位
sRtn = GTN_ClrSts (1,1, 8);
commandhandler("GTN_ClrSts", sRtn);
sRtn = GTN_AxisOn (1,AXIS_X);
commandhandler("GTN_AxisOn", sRtn);
sRtn = GTN_AxisOn (1,AXIS_Y);
commandhandler("GTN_AxisOn", sRtn);
// 等待伺服使能就绪
Sleep(1000);
// 将X轴设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS_X);
commandhandler("GTN_PrfPvt", sRtn);
// 将Y轴设置为PVT模式
sRtn = GTN_PrfPvt(1,AXIS_Y);
commandhandler("GTN_PrfPvt", sRtn);
// 计算X轴运动时间
sRtn = GTN_PvtContinuousCalculate(1,2, &pos_x[0], &vel_x[0], &percent_x[0],
                                  &velMax_x[0], &acc_x[0], &dec_x[0], &time_x[0]);
commandhandler("GTN_PvtContinuousCalculate", sRtn);
// 计算Y轴运动时间
sRtn = GTN_PvtContinuousCalculate(1,2, &pos_y[0], &vel_y[0], &percent_y[0],
                                  &velMax_y[0], &acc_y[0], &dec_y[0], &time_y[0]);
commandhandler("GTN_PvtContinuousCalculate", sRtn);
// 计算启动延时
```

```
if(time_x[1] <time_y[1] )
{
    timeBegin_x = time_y[1] - time_x[1];
    timeBegin_y = 0;
}
else
{
    timeBegin_x = 0;
    timeBegin_y = time_x[1] - time_y[1];
}
// 发送X轴数据点
sRtn = GTN_PvtTableContinuous(1,TABLE_X, 2, &pos_x[0], &vel_x[0], &percent_x[0],
                              &velMax_x[0], &acc_x[0], &dec_x[0], timeBegin_x);
commandhandler("GTN_PvtTableContinuous", sRtn);
// 发送Y轴数据点
sRtn = GTN_PvtTableContinuous(1,TABLE_Y, 2, &pos_y[0], &vel_y[0], &percent_y[0],
                              &velMax_y[0], &acc_y[0], &dec_y[0], timeBegin_y);
commandhandler("GTN_PvtTableContinuous", sRtn);
// X轴选择数据表TABLE_X
sRtn = GTN_PvtTableSelect(1,AXIS_X, TABLE_X);
commandhandler("GTN_PvtTableSelect", sRtn);
// Y轴选择数据表TABLE_Y
sRtn = GTN_PvtTableSelect(1,AXIS_Y, TABLE_Y);
commandhandler("GTN_PvtTableSelect", sRtn);
// 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 = 1<<(AXIS_X-1);
mask= 1<<(AXIS_Y-1);
sRtn = GTN_PvtStart(mask);
commandhandler("GTN_PvtStart", sRtn);
while(!kbhit())
{
    // 读取数据表和运动时间
    sRtn = GTN_PvtStatus(1,AXIS_X, &tableId[0], &time[0]);
    sRtn = GTN_PvtStatus(1,AXIS_Y, &tableId[1], &time[1]);
    // 读取规划速度
    sRtn = GTN_GetPrfVel (1,AXIS_X, &prfVel[0]);
    sRtn = GTN_GetPrfVel (1,AXIS_Y, &prfVel[1]);
    // 读取规划位置
    sRtn = GTN_GetPrfPos (1,AXIS_X, &prfPos[0]);
    sRtn = GTN_GetPrfPos (1,AXIS_Y, &prfPos[1]);
    printf("x:%2d %10.0lf %10.2lf %10.1lf y:%2d %10.0lf %10.2lf %10.1lf\r",
    tableId[0], time[0], prfVel[0], prfPos[0], tableId[1], time[1], prfVel[1], prfPos[1]);
}
// 伺服关闭
```

```
    sRtn = GTN_AxisOff (1,AXIS_X);
    printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_X);
    sRtn = GTN_AxisOff (1,AXIS_Y);
    printf("\nGTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_Y);
    getch();
    return 0;
}
```

第5章　指令详细说明

提示	本手册中所有字体为蓝色的指令（如 GTN_FollowData）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《GTN 系列运动控制器编程手册之基本功能》。

指令 1　GTN_FollowClear

指令原型	short GTN_FollowClear (short core, short profile, short fifo=0)		
指令说明	清除 Follow 运动模式指定 FIFO 中的数据。 运动状态下该指令无效。		
指令类型	立即指令，调用后立即生效。	**章节页码**	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
fifo	指定需要清除的 FIFO，取值范围：0、1 两个值。默认为 0。		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 (2) 请检查要清除的 FIFO 是否正在使用，运动是否结束。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 2　GTN_FollowData

指令原型	short GTN_FollowData (short core, short profile, long masterSegment, double slaveSegment, short type= FOLLOW_SEGMENT_NORMAL, short fifo=0)		
指令说明	向 Follow 运动模式指定 FIFO 增加数据。		
指令类型	立即指令，调用后立即生效。	**章节页码**	17
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
masterSegment	主轴位移。单位：pulse。		
slaveSegment	从轴位移。单位：pulse。		
type	数据段类型。 FOLLOW_SEGMENT_NORMAL（该宏定义为 0）普通段。默认为该类型。 FOLLOW_SEGMENT_EVEN（该宏定义为 1）匀速段。 FOLLOW_SEGMENT_STOP（该宏定义为 2）减速到 0 段。 FOLLOW_SEGMENT_CONTINUE（该宏定义为 3）保持 FIFO 之间速度连续。		

fifo	指定存放数据的 FIFO，取值范围：0、1 两个值。默认为 0。
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 (2) 请检查是否有足够的空间放新的数据。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	无。
指令示例	**例程 3-2　Follow 单 FIFO 模式**

指令 3　GTN_FollowSpace

指令原型	short GTN_FollowSpace (short core, short profile, short *pSpace, short fifo=0)		
指令说明	查询 Follow 运动模式指定 FIFO 的剩余空间。		
指令类型	立即指令，调用后立即生效。	**章节页码**	17
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pSpace	读取 FIFO 的剩余空间。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 4　GTN_FollowStart

指令原型	short GTN_FollowStart (short core, long mask, long option)		
指令说明	启动 Follow 运动。		
指令类型	立即指令，调用后立即生效。	**章节页码**	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
mask	按位指示需要启动 Follow 运动的轴号。当 bit 位为 1 时表示启动对应的轴。		
option	按位指示所使用的 FIFO，默认为 0。当 bit 位为 0 时表示对应的轴使用 FIFO1。当 bit 位为 1 时表示对应的轴使用 FIFO2。		

Bit	5	4	3	2	1	0
对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	11	10	9	8	7	6
对应轴	12 轴	11 轴	10 轴	9 轴	8 轴	7 轴

Bit	5	4	3	2	1	0
对应轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	11	10	9	8	7	6

指令返回值	对应轴 | 12 轴 | 11 轴 | 10 轴 | 9 轴 | 8 轴 | 7 轴 若返回值为 1： (1) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 (2) 检查运动是否结束，运动进行时，指令调用会失败； (3) 检查相应轴是否设置了跟随主轴； (4) 检查 FIFO 是否有数据； (5) 检查 mask 参数是否设置了启动相应的轴。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	无。
指令示例	**例程 3-2　Follow 单 FIFO 模式**

指令 5　GTN_FollowSwitch

指令原型	short GTN_FollowSwitch(short core, long mask)		
指令说明	切换 Follow 运动模式所使用的 FIFO。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
mask	按位指示需要切换 Follow 工作 FIFO 的轴号。当 bit 位为 1 时表示切换对应的轴的 FIFO。 Bit | 5 | 4 | 3 | 2 | 1 | 0 对应轴 | 6 轴 | 5 轴 | 4 轴 | 3 轴 | 2 轴 | 1 轴 Bit | 11 | 10 | 9 | 8 | 7 | 6 对应轴 | 12 轴 | 11 轴 | 10 轴 | 9 轴 | 8 轴 | 7 轴		
指令返回值	若返回值为 1： (1) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 (2) 检查运动是否进行，只有运动中才能切换。 (3) 检查目标 FIFO 是否为空。 (4) 检查 mask 参数是否设置了启动相应的轴。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 3-3　Follow 双 FIFO 切换**		

指令 6　GTN_GetFollowEvent

指令原型	short GTN_GetFollowEvent (short core, short profile, short *pEvent, short *pMasterDir, long *pPos)		
指令说明	读取 Follow 运动模式启动跟随条件。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 5 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围[1,2]
profile	规划轴号，正整数，取值范围与控制轴数相同。
pEvent	读取启动跟随条件。 FOLLOW_EVENT_START（该宏定义为 1）表示调用 GTN_FollowStart 以后立即启动。 FOLLOW_EVENT_PASS（该宏定义为 2）表示主轴穿越设定位置以后启动跟随。
pMasterDir	读取主轴运动方向。 1 主轴正向运动，-1 主轴负向运动。
pPos	读取穿越位置，单位：pulse。 当 event 为 FOLLOW_EVENT_PASS 时有效。
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	GTN_SetFollowEvent
指令示例	无。

指令 7　GTN_GetFollowLoop

指令原型	short GTN_GetFollowLoop(short core, short profile, long *pLoop)		
指令说明	读取 Follow 运动模式循环已经执行完成的次数。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pLoop	读取 Follow 模式循环已经执行完成的次数。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_SetFollowLoop		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 8　GTN_GetFollowMaster

指令原型	short GTN_GetFollowMaster (short core, short profile, short *pMasterIndex, short *pMasterType, short *pMasterItem)		
指令说明	读取 Follow 运动模式跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pMasterIndex	读取主轴索引。 正整数。对于 GTN-SLAVE-400 型端子板，取值范围：[1, 4]。		

pMasterType	对于 GTN-SLAVE-600 型端子板，取值范围：[1, 6]。 主轴索引不能与规划轴号相同，最好主轴索引号小于规划轴号，如主轴索引为 1 轴，规划轴号为 2 轴。
	读取主轴类型。 FOLLOW_MASTER_PROFILE（该宏定义为 2）表示跟随规划轴(profile)的输出值，默认为此类型。 FOLLOW_MASTER_ENCODER（该宏定义为 1）表示跟随编码器(encoder)的输出值。 FOLLOW_MASTER_AXIS（该宏定义为 3）表示跟随轴(axis)的输出值。
pMasterItem	合成轴类型，当 masterType= FOLLOW_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值，默认为该值。 1 表示 axis 的编码器位置输出值。
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	GTN_SetFollowMaster
指令示例	无。

指令 9　GTN_GetFollowMemory

指令原型	short GTN_GetFollowMemory (short core, short profile, short *pMemory)		
指令说明	读取 Follow 运动模式的缓存区大小。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pMemory	读取 Follow 运动缓存区大小标志。 0：每个 Follow 运动缓存区有 16 段空间。 1：每个 Follow 运动缓存区有 512 段空间。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetFollowMemory		
指令示例	无。		

指令 10　GTN_GetPtLoop

指令原型	short GTN_GetPtLoop(short core, short profile, long *pLoop)		
指令说明	查询 PT 运动模式循环执行的次数。动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		

pLloop	查询 PT 模式循环已经执行完成的次数。动态模式下该参数无效。
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	GTN_SetPtLoop
指令示例	无。

指令 11　GTN_GetPtMemory

指令原型	short GTN_GetPtMemory(short core, short profile, short *pMemory)		
指令说明	读取 PT 运动模式的缓存区大小。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pMemory	读取 PT 运动缓存区大小标志。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_SetPtMemory		
指令示例	无。		

指令 12　GTN_GetPvtLoop

指令原型	short GTN_GetPvtLoop(short core, 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	33
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
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 GTN_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_SetPvtLoop		
指令示例	无。		

指令 13　GTN_PrfFollow

指令原型	short GTN_PrfFollow (short core, short profile, short dir)

指令说明	设置指定轴为 Follow 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
dir	设置跟随方式。 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随。		
指令返回值	若返回值为 1： (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。 (2) 当前已经是 Follow 模式，但再次设置的 dir 与当前的 dir 不一致。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 14　GTN_PrfPt

指令原型	short GTN_PrfPt(short core, short profile, short mode=PT_MODE_STATIC)		
指令说明	设置指定轴为 PT 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
mode	指定 FIFO 使用模式。 PT_MODE_STATIC（该宏定义为 0）静态模式。默认为该模式。 PT_MODE_DYNAMIC（该宏定义为 1）动态模式。		
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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 2-1　PT 静态 FIFO**		

指令 15　GTN_PrfPvt

指令原型	short GTN_PrfPvt(short core, 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	33
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
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	无。		

指令 16　GTN_PtClear

指令原型	short GTN_PtClear(short core, short profile, short fifo)		
指令说明	清除 PT 运动模式指定 FIFO 中的数据。 运动状态下该指令无效。 动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	**章节页码**	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1： (1) 静态模式下，检查要清除的 FIFO 是否正在使用，在运动； (2) 动态模式下，不能在运动时清 FIFO； (3) 请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 2-1　PT 静态 FIFO**		

指令 17　GTN_PtData

指令原型	short GTN_PtData(short core, short profile, double pos, long time, short type, short fifo=0)		
指令说明	向 PT 运动模式指定 FIFO 增加数据。		
指令类型	立即指令，调用后立即生效。	**章节页码**	8
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pos	段末位置。单位：pulse。		
time	段末时间。单位：ms。		
type	数据段类型。 PT_SEGMENT_NORMAL（该宏定义为 0）普通段。默认为该类型。 PT_SEGMENT_EVEN（该宏定义为 1）匀速段。 PT_SEGMENT_STOP（该宏定义为 2）减速到 0 段。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1： (1) 请检查 Space 是否小于 0，若是，则等待 Space 大于 0； (2) 请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其		

	意义。
相关指令	无。
指令示例	**例程 2-1　PT 静态 FIFO**

指令 18　GTN_PtSpace

指令原型	short GTN_PtSpace(short core, short profile, short *pSpace, short fifo=0)		
指令说明	查询 PT 运动模式指定 FIFO 的剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
pSpace	读取 PT 指定 FIFO 的剩余空间。		
fifo	指定所要查询的 FIFO，取值范围：0、1 两个值。默认为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 2-1　PT 静态 FIFO**		

指令 19　GTN_PtStart

指令原型	short GTN_PtStart(short core, long mask, long option)
指令说明	启动 PT 运动。
指令类型	立即指令，调用后立即生效。　章节页码　8
指令参数	该指令共有 3 个参数，参数的详细信息如下。
core	内核，正整数，取值范围[1,2]
mask	按位指示需要启动 PT 运动的轴号。当 bit 位为 1 时表示启动对应的轴。 Bit 5 4 3 2 1 0 对应轴 6 轴 5 轴 4 轴 3 轴 2 轴 1 轴 Bit 11 10 9 8 7 6 对应轴 12 轴 11 轴 10 轴 9 轴 8 轴 7 轴
option	按位指示所使用的 FIFO，默认为 0。 Bit 5 4 3 2 1 0 对应轴 6 轴 5 轴 4 轴 3 轴 2 轴 1 轴 Bit 11 10 9 8 7 6 对应轴 12 轴 11 轴 10 轴 9 轴 8 轴 7 轴 当 bit 位为 0 时表示对应的轴使用 FIFO1（即 fifo=0） 当 bit 位为 1 时表示对应的轴使用 FIFO2（即 fifo=1） 动态模式下该参数无效。
指令返回值	若返回值为 1：请检查相应轴的 FIFO 是否有数据，若没有，请先压入数据； 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其

	意义。
相关指令	无。
指令示例	**例程 2-1　PT 静态 FIFO**

指令 20　GTN_PvtContinuousCalculate

指令原型	short GTN_PvtContinuousCalculate(short core, 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	33
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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/ms^2”，数组长度为 count。		
pDec	数据点减速度数组。单位是“pulse/ms^2”，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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-4　Continuous 描述方式**		

指令 21　GTN_PvtPercentCalculate

指令原型	short GTN_PvtPercentCalculate (short core, 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	33
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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 GTN_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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	无。
指令示例	**例程 4-3　Percent 描述方式**

指令 22　GTN_PvtStart

指令原型	short GTN_PvtStart(short core, 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	33
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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。 Bit 5 4 3 2 1 0 对应轴 6 轴 5 轴 4 轴 3 轴 2 轴 1 轴 Bit 11 10 9 8 7 6 对应轴 12 轴 11 轴 10 轴 9 轴 8 轴 7 轴 在启动运动之前，可以调用 GTN_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 GTN_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-1　PVT 描述方式**		

指令 23　GTN_PvtStatus

指令原型	short GTN_PvtStatus(short core, 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	33
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
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 GTN_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其		

	意义。
相关指令	无。
指令示例	**例程 4-1　PVT 描述方式**

指令 24　GTN_PvtTable

指令原型	short GTN_PvtTable (short core, 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	33
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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 GTN_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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-1　PVT 描述方式**		

指令 25　GTN_PvtTableComplete

指令原型	short GTN_PvtTableComplete (short core, 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	33
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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 GTN_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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	无。
指令示例	无。

指令 26　GTN_PvtTableContinuous

指令原型	short GTN_PvtTableContinuous (short core, 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	33
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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 GTN_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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-4　Continuous 描述方式**		

指令 27　GTN_PvtTablePercent

指令原型	short GTN_PvtTablePercent (short core, 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	33
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
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 GTN_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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-3　Percent 描述方式**		

指令 28　GTN_PvtTableSelect

指令原型	short GTN_PvtTableSelect(short core, 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	33
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
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《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 4-1　PVT 描述方式**		

指令 29　GTN_SetFollowEvent

指令原型	short GTN_SetFollowEvent(short core, short profile, short event, short masterDir, long pos)
指令说明	设置 Follow 运动模式启动跟随条件。

指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
event	启动跟随条件。 FOLLOW_EVENT_START（该宏定义为 1）表示调用 GTN_FollowStart 以后立即启动。 FOLLOW_EVENT_PASS（该宏定义为 2）表示主轴穿越设定位置以后启动跟随。		
masterDir	穿越启动时，主轴的运动方向。 1 主轴正向运动，-1 主轴负向运动。		
pos	穿越位置，单位：pulse。 当 event 为 FOLLOW_EVENT_PASS 时有效。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_GetFollowEvent		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 30　GTN_SetFollowLoop

指令原型	short GTN_SetFollowLoop(short core, short profile, long loop)		
指令说明	设置 Follow 运动模式下的循环次数。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
loop	指定 Follow 模式循环执行的次数。 取值范围：[-21474836482147483647]。注：loop 小于 1 表示无限次循环。		
指令返回值	若返回值为 1：请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_GetFollowEvent		
指令示例	**例程 3-2　Follow 单 FIFO 模式**		

指令 31　GTN_SetFollowMaster

指令原型	short GTN_SetFollowMaster (short core, short profile, short masterIndex, short masterType = FOLLOW_MASTER_PROFILE, short masterItem)		
指令说明	设置 Follow 运动模式下的跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	17
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		

profile	规划轴号，正整数，取值范围与控制轴数相同。
masterIndex	主轴索引。 正整数。对于 GTN-SLAVE-400 型端子板，取值范围：[1, 4]。 对于 GTN-SLAVE-600 型端子板，取值范围：[1, 6]。 主轴索引不能与规划轴号相同，最好主轴索引号小于规划轴号，如主轴索引为 1 轴，规划轴号为 2 轴。
masterType	主轴类型。 FOLLOW_MASTER_PROFILE（该宏定义为 2）表示跟随规划轴(profile)的输出值。默认为该类型。 FOLLOW_MASTER_ENCODER（该宏定义为 1）表示跟随编码器(encoder)的输出值。 FOLLOW_MASTER_AXIS（该宏定义为 3）表示跟随轴(axis)的输出值。 FOLLOW_MASTER_AXIS_OTHER(该宏定义为 103) 表示 core2 轴跟随 core1 中轴(axis)的输出值 FOLLOW_MASTER_ENCODER_OTHER(该宏定义为 101) 表示 core2 轴跟随 core1 中编码器(encoder)的输出值
masterItem	合成轴类型，当 masterType= FOLLOW_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值，默认为该值。 1 表示 axis 的编码器位置输出值。
指令返回值	若返回值为 1： (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。 (2) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。
相关指令	GTN_GetFollowMaster
指令示例	**例程 3-2　Follow 单 FIFO 模式**

指令 32　GTN_SetFollowMemory

指令原型	short GTN_SetFollowMemory (short core, short profile, short memory)		
指令说明	设置 Follow 运动模式的缓存区大小。		
指令类型	立即指令，调用后立即生效。	**章节页码**	17
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
memory	Follow 运动缓存区大小标志。 0：每个 Follow 运动缓存区有 16 段空间。 1：每个 Follow 运动缓存区有 512 段空间。		
指令返回值	若返回值为 1： (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。 (2) 请检查当前轴是否为 Follow 模式，若不是，请先调用 GTN_PrfFollow 将当前轴设置为 Follow 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		

相关指令	GTN_GetFollowMemory
指令示例	无。

指令 33　GTN_SetPtLoop

指令原型	short GTN_SetPtLoop(short core, short profile, long loop)		
指令说明	设置 PT 运动模式循环执行的次数。动态模式下该指令无效。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
loop	指定 PT 模式循环执行的次数。 取值范围：非负整数。如果需要无限循环，设置为 0。 动态模式下该参数无效。		
指令返回值	若返回值为 1：请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_GetPtLoop		
指令示例	无。		

指令 34　GTN_SetPtMemory

指令原型	short GTN_SetPtMemory(short core, short profile, short memory)		
指令说明	设置 PT 运动模式的缓存区(FIFO)大小。		
指令类型	立即指令，调用后立即生效。	章节页码	8
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
memory	PT 运动缓存区大小标志： 0：每个 PT 运动缓存区有 32 段空间。 1：每个 PT 运动缓存区有 1024 段空间。		
指令返回值	若返回值为 1： (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。 (2) 请检查当前轴是否为 PT 模式，若不是，请先调用 GTN_PrfPt 将当前轴设置为 PT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_GetPtMemory		
指令示例	无。		

指令 35　GTN_SetPvtLoop

指令原型	short GTN_SetPvtLoop(short core, 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**	33
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围[1,2]		
profile	规划轴号，正整数，取值范围与控制轴数相同。		
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 GT_PvtTable 是否为 PVT 模式，若不是，请先调用 GTN_PrfPvt 将当前轴设置为 PVT 模式。 其他返回值：请参照《GTN 系列运动控制器之基本功能》第 3 章指令返回值及其意义。		
相关指令	GTN_GetPvtLoop		
指令示例	无。		

第6章 索引

6.1 指令索引

6.2 表格索引

6.3 图片索引

6.4 例程索引