



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
二层 W211 室

电 话：0755-26970823 26970817 26970824

传 真：0755-26970846

电子邮件：support@gogoltech.com

网 址：<http://www.gogoltech.com.cn>

固高科技（香港）有限公司

地 址：香港九龙清水湾香港科技大学新翼楼 3639 室

电 话：(852) 2358-1033

传 真：(852) 2358-4931

电子邮件：info@gogoltech.com

网 址：<http://www.gogoltech.com/>

Web: <http://www.gogoltech.com>

GE 系列运动控制器 编程手册



务必将此手册交给用户

- 非常感谢您选购 GE 连续轨迹运动控制器
- 在您使用之前，请仔细阅读此手册，确保正确使用。
- 请将此手册妥善保存，以备随时查阅。


版权申明

固高科技有限公司
保留所有权力

固高科技有限公司（以下简称固高科技）保留在不事先通知的情况下，修改本手册中的产品和产品规格等文件的权力。

固高科技不承担由于使用本手册或本产品不当，所造成直接的、间接的、特殊的、附带的或相应产生的损失或责任。

固高科技具有本产品及其软件的专利权、版权和其它知识产权。未经授权，不得直接或者间接地复制、制造、加工、使用本产品及其相关部分。

 注意	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。
--	--

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 - 26970839）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) 26970835
- ◆ 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 GE 系列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，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。

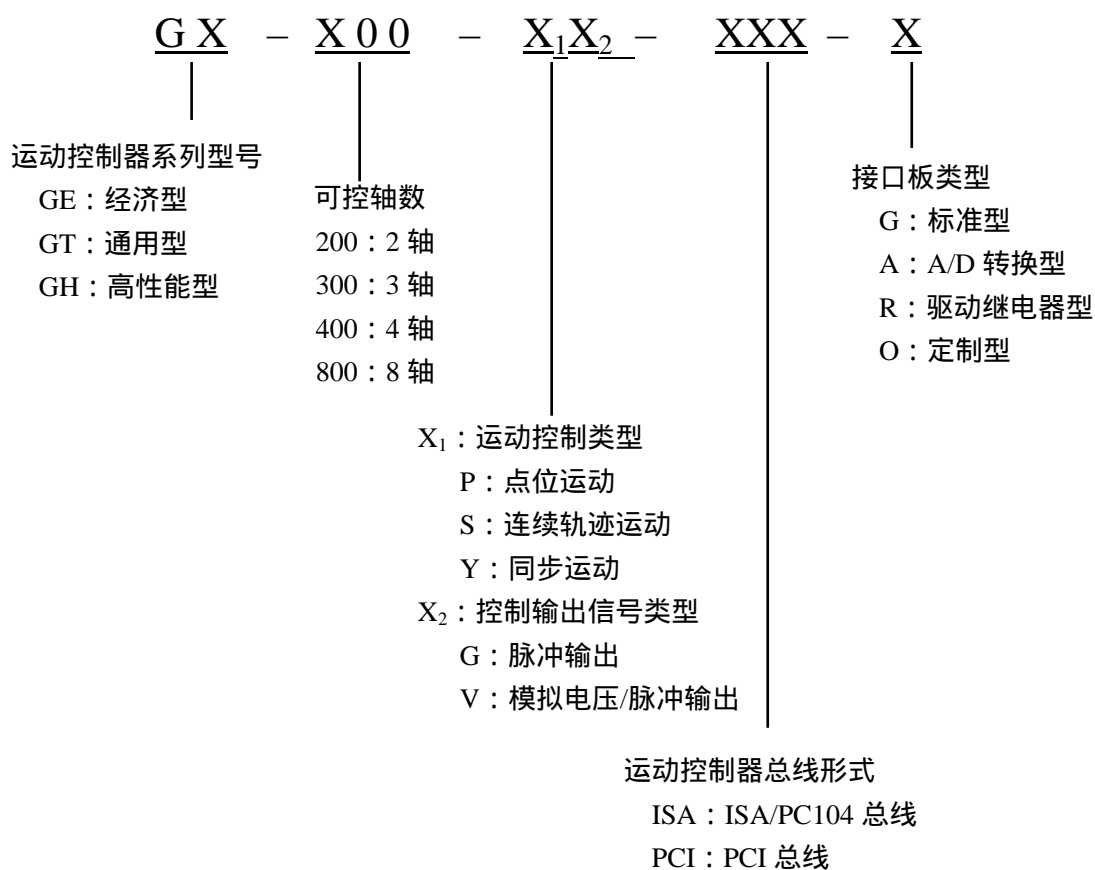
编程手册的主要内容

本手册由十二章内容组成。详细介绍了 GE 系列运动控制器的控制功能及编程实现。

相关文件

关于 GE 系列运动控制器调试和安装，请参见随产品配套的《GE 运动控制器用户手册》。

型号说明



型 号	运 动 控 制 类 型	控 制 轴 数	控 制 电 机	输 出 方 式
GE-X00-SV	连续轨迹运动	4/3/2	伺服/步进电机	模拟电压输出/脉冲输出
GE-X00-SG	连续轨迹运动	4/3/2	伺服/步进电机	脉冲输出
GE-X00-PV	点位运动	8/4/2	伺服/步进电机	模拟电压输出/脉冲输出
GE-X00-PG	点位运动	8/4/2	伺服/步进电机	脉冲输出

目录

第一章 运动控制器函数库的使用.....	1
1.1 DOS 系统下函数库的使用.....	1
1.2 Windows 系统下动态链接库的使用.....	2
第二章 指令返回值及其意义.....	4
2.1 指令返回值	4
2.2 指令错误寄存器	5
2.3 记录指令执行流程.....	8
第三章 控制系统初始化	10
3.1 运动控制器初始化.....	10
3.2 专用输入信号参数设置.....	11
3.3 控制轴初始化	13
3.4 控制轴实际位置设置.....	16
3.5 运动控制器初始化例程.....	16
第四章 运动控制器状态检测.....	19
4.1 指令列表	19
4.2 重点说明	19
4.3 控制轴状态寄存器.....	19
4.4 控制轴模式寄存器.....	22
4.5 连续轨迹运动状态寄存器.....	22
5.1 梯形曲线	24
5.2 S 曲线	28
5.3 速度控制	31
5.4 参数刷新机制	34
5.5 停止控制轴的运动.....	34
第六章 连续轨迹运动	36
6.1 进给速度、加速度设置.....	36
6.2 轨迹运动的轨迹描述.....	38
6.3 多段连续轨迹运动的基本实现.....	40
6.4 小线段连续轨迹运动的速度规划策略.....	44
6.5 连续轨迹运动中的速度倍率.....	47
6.6 连续轨迹运动的机床坐标系和加工坐标系.....	48
6.7 连续轨迹运动的运动信息反馈.....	48
7.1 指令列表	51
7.2 Home 回原点.....	51
7.3 Home + Index 回原点.....	51
7.3 Home + Index 回原点.....	52
7.4 例程	53
7.5 重点说明	58
第八章 安全机制	59
8.1 设置跟随误差超限自动停止.....	59
8.2 设置输出电压饱和极限.....	59

8.3 限位状态处理	60
8.4 控制轴驱动报警处理.....	60
第九章 数字 I/O	61
9.1 通用数字 I/O	61
9.2 专用数字 I/O	61
第十章 运动控制器可选功能.....	63
10.1 运动控制器数据监测.....	63
10.2 模拟电压输入	71
10.3 手脉功能	73
10.4 主轴转速控制	75
第十一章 指令列表	77
第十二章 指令说明	84

第一章 运动控制器函数库的使用

GE 系列运动控制器提供 DOS 下的 C 语言函数库和 Windows 下的动态链接库。用户只要调用函数库中的指令，就可以实现运动控制器的各种功能。下面分别讲述 DOS、Windows 系统下函数库的使用方法。

1.1 DOS 系统下函数库的使用

GE 连续轨迹运动控制器 (GE-X00-SX) DOS 系统下的函数库和头文件存放在产品配套光盘的 DOS\ UserLib 文件夹下，共七个文件。分别为：

ges.h	头文件
gest.lib	微模式(Tiny)的函数库
gess.lib	小模式(Small)的函数库
gesm.lib	中模式(Medium)的函数库
gesc.lib	紧凑模式(Compact)的函数库
gesl.lib	大模式(Large)的函数库
gesh.lib	巨大模式(Huge)的函数库

GE 点位运动控制器 (GE-X00-PX) DOS 系统下的函数库和头文件存放在产品配套光盘的 DOS\UserLib 文件夹下，共七个文件。分别为：

gep.h	头文件
gept.lib	微模式(Tiny)的函数库
geps.lib	小模式(Small)的函数库
gepm.lib	中模式(Medium)的函数库
gepc.lib	紧凑模式(Compact)的函数库
gepl.lib	大模式(Large)的函数库
geph.lib	巨大模式(Huge)的函数库

该函数库是用 Borland C + + 3.1 编译生成的，用户可在 Borland C + + 3.1 或更高版本的开发环境下链接该函数库。

使用函数库开发应用程序的流程如下：

1. 确定编译模式——
 - 启动 Borland C + + 3.1 ；
 - 选择“ Project ” 菜单下“ Open Project ”，打开已有工程文件或建立新工程文件；
 - 选择“ Options ” 菜单下“ Compiler ” 中的“ Code generation...” 菜单项；
 - 在“ Model ” 一栏中选择当前工程文件的编译模式。
2. 添加函数库——
 - 选择相应编译模式的函数库和头文件，复制到用户当前的工程文件

夹中；

- 选择“Window”菜单下的“Project”菜单项，切换到工程窗口；
- 选择“Project”菜单下的“Add Item...”菜单项；
- 在“Name”栏中输入“*.lib”后回车；
- 在“Files”栏中选择将要加入工程的函数库，然后点击“Add”按钮。

3. 添加源程序文件——

- 选择“File”菜单下的“New”菜单项，新建一个.CPP或.C文件；
- 在新建的.CPP或.C文件中加入函数库头文件的声明，例如：
`#include "ges.h" ;` //声明连续轨迹运动控制器函数库头文件
- 保存该文件，然后将该源程序文件添加到当前工程当中。

1.2 Windows 系统下动态链接库的使用

在Windows系统下使用GE系列运动控制器，首先要安装驱动程序，GE系列运动控制器的驱动程序存放在产品配套光盘的Windows\Driver文件夹下。

运动控制器指令函数动态链接库存放在产品配套光盘的Windows\VC（Windows\VB或Windows\Delphi）文件夹下。连续轨迹运动控制器（GE-X00-SX）的动态链接库文件名为ges.dll，点位运动控制器（GE-X00-PX）的动态链接库文件名为gep.dll。

在Windows系统下，用户可以使用任何能够支持动态链接库的开发工具来开发应用程序。下面分别以Visual C++、Visual Basic和Delphi为例讲解如何在这些开发工具中使用运动控制器的动态链接库。

1.2.1 Visual C++中的使用

1. 启动Visual C++，新建一个工程；
2. 将产品配套光盘Windows\VC文件夹中的动态链接库、头文件和lib文件复制到工程文件夹中；
3. 选择“Project”菜单下的“Settings...”菜单项；
4. 切换到“Link”标签页，在“Object/library modules”栏中输入lib文件名（例如ges.lib）；
5. 在应用程序文件中加入函数库头文件的声明，例如：
`#include "ges.h"`
6. 至此，用户就可以在Visual C++中调用函数库中的任何函数，开始编写应用程序。

1.2.2 Visual Basic 中的使用

1. 启动Visual Basic，新建一个工程；
2. 将产品配套光盘Windows\VB文件夹中的动态链接库和函数声明文件

复制到工程文件夹中；

3. 选择“工程”菜单下的“添加模块”菜单项；
4. 切换到“现存”标签页，选择函数声明文件（例如 ges.bas），将其添加到工程当中；
5. 至此，用户就可以在 Visual Basic 中调用函数库中的任何函数，开始编写应用程序。

1.2.3 Delphi 中的使用

1. 启动 Delphi，新建一个工程；
2. 将产品配套光盘 Windows\Delphi 文件夹中的动态链接库和函数声明文件复制到工程文件夹中；
3. 选择“Project”菜单下的“Add to Project...”菜单项；
4. 将函数声明文件添加到工程当中；
5. 在代码编辑窗口中，切换到用户的单元文件；
6. 选择“File”菜单下的“Use Unit...”菜单项，添加对函数声明文件的引用；
7. 至此，用户就可以在 Delphi 中调用函数库中的任何函数，开始编写应用程序。

第二章 指令返回值及其意义

2.1 指令返回值

运动控制器按照主机发送的指令工作。运动控制器指令封装在 C 语言函数库（DOS 环境）和动态链接库（Windows 环境）中。用户在编写应用程序时，通过调用运动控制器指令来操纵运动控制器。

运动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表 2-1 所示：

表 2-1 GE-X00-PX 运动控制器指令返回值定义

返回值	意义	处理方法
-3	指令重入	不要在多个线程当中同时调用运动控制器指令。
-1	通讯出错	检查运动控制器是否工作正常。
0	指令执行成功	继续。
1	指令错误	调用 GT_GetCmdSts 指令，确定出错原因，予以改正。
7	指令参数错误	检查指令参数是否正确。

表 2-2 GE-X00-SX 运动控制器指令返回值定义

返回值	意义	处理方法
-3	指令重入	不要在多个线程当中同时调用运动控制器指令。
-1	通讯出错	检查运动控制器是否工作正常。
0	指令执行成功	继续。
1	指令错误	调用 GT_GetCmdSts 指令，确定出错原因，予以改正。
2	圆弧半径为零或弦长为零（整圆除外）	圆弧指令相关参数出错时返回该值。检查该指令的参数，改正后重发此指令。
3	直线长度为零或溢出	直线插补指令相关参数出错时返回该值。检查该指令的参数，改正后重发此指令。
4	（加）速度小于等于零或溢出	合成速度、合成加速度指令相关参数出错时返回该值。检查该指令的参数，改正后重发此指令。
5	圆弧弦长大于直径	圆弧指令相关参数出错时返回该值。检查该指令参数是否正确合理，改正后重发。
7	指令参数错误	检查该指令的参数是否正确，改正后重发此指令。



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。

如果指令返回值为-1，而且重复调用该指令仍返回-1，说明运动控制器的通讯出现故障，运动控制器没有正确地接收主机的指令；或是运动控制器工作不正常，不能正确处理主机指令。此时请按照如下步骤进行检查：

1. 在 Windows 系统下使用运动控制器时，请检查驱动程序是否正确安装；
2. 检查运动控制器、接插件和连接线是否插牢；
3. 检查运动控制器的基地址跳线是否正确，请确认运动控制器的基地址跳线与应用程序中所指定的基地址相一致（针对 ISA/PC104 总线的运动控制器）；
4. 检查 PC 机的插槽是否能够正常工作，并尝试更换插槽或 PC 机；
5. 检查 JP4 跳线是否正确，JP4 跳线默认为 1-2 脚短接。

2.2 指令错误寄存器

当运动控制器指令的返回值为 1 时，说明指令执行错误，此时应当调用指令 GT_GetCmdSts 进一步确认具体出错原因。指令错误寄存器的具体定义如表 2-3，当某个状态位为 1 时，指示了相应的错误原因。

表 2-3 GE-X00-PX 运动控制器指令错误寄存器定义

位	定 义
0	指令参数溢出。
1	指令参数非法。
2	GT_MltiUpdt 的参数值为 0。
3	伺服使能时调用 GT_DrvRst。
4	保留。
5	设置捕获状态指令出错： 已经对控制轴设置了捕获方式，但还没有捕获到信息，或者捕获了信息后没有清除该标志，主机就再次调用设置捕获方式的指令。
6	在运动状态下修改控制轴的工作模式。
7	保留。
8	保留。
9	在控制轴产生驱动报警时调用 GT_AxisOn。
10	保留。
11	控制轴在运动状态时调用 GT_ZeroPos 或 GT_SetAtlPos； 控制轴在运动状态时调用 GT_Update 或 GT_MltiUpdt 刷新在运动状态下不允许修改的运动控制参数（例如梯形曲线不允许在运动状态下修改加速度，S 曲线不允许在运动状态下修改速度和加速度。 ）。

第二章 指令返回值及其意义

12	保留。
13	保留。
14	保留。
15	保留。

表 2-4 GE-X00-SX 运动控制器指令错误寄存器定义

位	定 义
0	保留。
1	控制轴指令参数非法。
2	GT_MltiUpdt 的参数值为 0。
3	控制轴电机状态错误： 控制轴处在伺服使能状态时，调用 GT_DrvRst 指令复位驱动器，或调用 GT_AlarmOff/GT_AlarmOn 指令管理报警信号，或调用 GT_EncOff / GT_EncOn 指令关闭或打开编码器； 控制轴产生驱动报警时，调用 GT_AxisOn 指令。
4	保留。
5	设置捕获状态指令出错： 已经对控制轴设置了捕获方式，但还没有捕获到信息，或者捕获了信息后没有清除该标志，主机就再次调用设置捕获方式的指令。
6	控制轴状态指令错误： 控制轴处于脉冲输出方式下时，调用 GT_SetKp 等与伺服控制相关的指令； 控制轴处于伺服使能状态下时，调用 GT_CtrlMode 指令； 控制轴处于模拟量输出方式下时，调用 GT_StepDir/GT_StepPulse 指令； 调用 GT_MapCnt 指令修改坐标系偏移量时，有控制轴处于运动状态，或者缓冲区没有关闭，或者参数超限。
7	保留。
8	连续轨迹运动指令错误： 调用 GT_StrtList 指令打开缓冲区后，没有发送定位指令就直接调用连续轨迹运动指令（如 GT_LnXY 指令）； 调用立即轨迹指令时，轨迹运动没有停止； 关闭缓冲区后，仍调用定位指令； 重复调用定位指令，或定位指令参数不正确； 缓冲区输入方式下调用指令 GT_SetSynAcc； 设置最大速度、启动速度、加速度和急停加速度时，轨迹运动没有停止； 轨迹运动参数指令的参数不正确。
9	缓冲区状态管理指令出错： 调用 GT_StrtMtn 指令时有轨迹运动且没有停止； 调用 GT_StrtList 指令时缓冲区连续轨迹运动没有停止或者缓冲区已经打开； 缓冲区处于打开状态时调用 GT_AddList 指令；

第二章 指令返回值及其意义

位	定 义
	调用 GT_RestoreMtn 指令时轨迹运动没有停止 ,或缓冲区中没有未执行的轨迹运动描述指令。
10	保留。
11	控制轴处于运动状态时调用 GT_ZeroPos 或 GT_SetAtlPos 指令。
12	保留。
13	保留。
14	该指令不属于连续轨迹运动控制器的指令集。
15	由于运动控制器缓冲区满 ,刚刚调用的运动描述指令没有被运动控制器接收 ;主机需要继续重复调用该指令 ,直至接收该指令为止。

下面这段程序以标准 C 语言为例 , 简单说明了利用指令返回值进行错误处理的方法。当指令返回值异常时 , 发出蜂鸣声并显示错误信息。在实际应用系统中 , 应当根据具体情况采取相应的处理方法。

例程 2-1 利用指令返回值进行错误处理

```
void error(short rtn)
{
    switch(rtn)
    {
        case -1:
            printf("\a\nCommunication Error !");          break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");                break;
        case 2:
            printf("\a\nRadius or chord is 0 !");          break;
        case 3:
            printf("\a\nLength is 0 or overflow !");        break;
        case 4:
            printf("\a\nVelocity or acceleration is less than 0 !");
                                                                break;
        case 5:
            printf("\a\nChord is greater than diameter !"); break;
        case 7:
            printf("\a\nParameter error !");              break;
        default:
            printf("\a\nError Code = %d ",rtn);            break;
    }
}

void main()
```

```
{
    short  rtn;
    rtn = GT_Open();           error(rtn);      //打开运动控制器
    rtn = GT_Reset();          error(rtn);      //复位运动控制器
    rtn = GT_ClrSts(1);        error(rtn);      //清除 1 轴状态寄存器
    ...                        //其它指令
    rtn = GT_Close();          error(rtn);      //关闭运动控制器
}
```

2.3 记录指令执行流程

在调试应用程序时，需要检查发送到运动控制器的指令流和指令参数，使用函数库所提供的 GT_HookCommand 指令可以简化这方面的工作。

用户通过调用 GT_HookCommand 指令将自己编写的指令处理程序挂接到函数库。当用户调用运动控制器指令时，函数库在指令执行完成以后自动调用所挂接的指令处理程序，并将当前的指令名称、指令参数和指令返回值传递给指令处理程序。指令处理程序可以将这些信息记录到文件当中，以使用户对其进行分析。

用户可在指令处理程序中检查指令返回值，集中进行错误处理，简化错误处理程序。

2.3.1 Visual C++中的指令处理函数的声明和挂接

在 Visual C++中调用 GT_HookCommand 指令挂接指令处理程序时，所挂接的指令处理程序必须声明为标准调用（在函数名之前添加关键字__stdcall）。

例程 2-2 在 Visual C++中使用

```
void __stdcall CommandHandle(char *command,short error)
{
    ...
}

void main()
{
    GT_HookCommand(CommandHandle);      //挂接指令处理函数
    ...
}
```

2.3.2 Visual Basic 中指令处理过程的声明和挂接

在 Visual Basic 中调用 GT_HookCommand 指令挂接指令处理程序时，所挂接的指令处理程序必须定义在独立的标准模块当中，参数必须定义为传值方式。

指令名称参数应当调用函数 StrConv () 转化为 Unicode 编码。

例程 2-3 在 Visual Basic 中使用

```
Sub CommandHandle(ByVal command As String, ByVal error As Integer)
    Dim Msg As String
    Msg = StrConv(command, vbUnicode)    //转化成为 Unicode 编码
    ...
End Sub

Private Sub Command1_Click()
    GT_HookCommand AddressOf CommandHandle//挂接指令处理程序
    ...
End Sub
```

2.3.3 Delphi 中指令处理过程的声明和挂接

在 Delphi 中调用 GT_HookCommand 指令挂接指令处理程序时，所挂接的指令处理程序必须声明为标准调用（在函数名之后添加关键字 stdcall）。

例程 2-4 在 Delphi 中使用

```
procedure CommandHandle(command:PChar;error:SmallInt);stdcall;
begin
    ...
end

procedure TForm1.Button1Click(Sender: TObject);
begin
    GT_HookCommand(CommandHandle);        //挂接指令处理程序
    ...
end;
```

第三章 控制系统初始化

GE 系列运动控制器包含 GE 连续轨迹运动控制器 (GE-X00-SX) 和 GE 点位运动控制器 (GE-X00-PX)。在开发应用程序之前,必须结合实际系统正确选择和设置运动控制器。

3.1 运动控制器初始化

本章描述如何将 GE 系列运动控制器进行初始化,并给出了相关的例程。如没有特别标注,本章中使用的指令可适用于 GE 系列中所有的运动控制器。

3.1.1 指令列表

表 3-1 运动控制器初始化指令列表

指 令	说 明
GT_Open	打开运动控制器
	ISA 总线的运动控制器需要设置基地址
	PCI 总线的运动控制器不需要设置基地址
GT_Reset	复位运动控制器

3.1.2 重点说明

3.1.2.1 打开运动控制器

必须首先调用指令 GT_Open 打开运动控制器,和运动控制器建立通讯。

对于 ISA 总线的运动控制器,调用指令 GT_Open 时必须指定运动控制器的基地址。基地址必须和运动控制器的基地址跳线一致,基地址跳线的设置方法请参见《GE 系列运动控制器用户手册》中的“基地址选择:JP1”一栏。

对于 PCI 总线的运动控制器,调用指令 GT_Open 时不必指定运动控制器的基地址。

在退出应用程序时应当调用指令 GT_Close 关闭运动控制器。

3.1.2.2 复位运动控制器

调用指令 GT_Reset 将使运动控制器的所有寄存器恢复到默认状态,一般在打开运动控制器之后调用该指令。

3.2 专用输入信号参数设置

3.2.1 指令列表

表 3-2 专用输入信号参数设置指令列表

指 令	说 明
GT_LmtSns	设置运动控制器各轴限位开关触发电平（默认为 0，高电平触发）
GT_HomeSns	设置运动控制器各轴 Home 信号触发沿（默认为 0，下降沿触发）
GT_EncSns	设置运动控制器各轴编码器计数方向

3.2.2 重点说明

3.2.2.1 设置限位开关触发电平

运动控制器为每个控制轴都提供了正负限位信号输入接口。各轴限位开关应与端子板上相应的限位信号输入端子和 OGND 相连。其接线方式请参见《GE 系列运动控制器用户手册》中的“专用输入、输出连接方法”一栏。当某个方向上的限位开关触发时，运动控制器将自动停止该方向上的运动，以保障系统安全。当离开限位开关以后，必须调用指令 GT_ClrSts 才能清除该轴的限位状态。

运动控制器默认的限位开关为常闭开关，即各轴处于正常工作状态时，其限位开关信号输入为低电平；当限位开关信号输入为高电平时，与其对应轴的限位状态将被触发。如果使用常开开关，需要通过调用指令 GT_LmtSns 改变限位开关触发电平。

指令 GT_LmtSns 的参数设置各轴正负限位开关的触发电平，当该参数的某个状态位为 0 时，表示将对应的限位开关设置为高电平触发，当某个状态位为 1 时表示将对应的限位开关设置为低电平触发。指令参数和各轴限位的对应关系如下所示：

表 3-3 指令 GT_LmtSns 的参数和各轴限位的对应关系

状态位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
限位开关	轴 8		轴 7		轴 6		轴 5		轴 4		轴 3		轴 2		轴 1	
	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+

3.2.2.2 原点信号触发沿设置

运动控制器为每个控制轴提供了一个原点信号输入接口。各轴原点开关应与端子板上相应的原点信号输入端子和 OGND 相连。其接线方式请参见《GE 系列运动控制器用户手册》中的“专用输入、输出连接方法”一栏。

第三章 控制系统初始化

默认情况下运动控制器原点信号为下降沿触发，调用指令 GT_HomeSns 能够改变运动控制器原点信号的边沿触发方式。

当指令参数的某个状态位为 0 时，所对应控制轴的原点信号为下降沿触发；当指令参数的某个状态位为 1 时，所对应控制轴的原点信号为上升沿触发。指令参数的状态位和各轴原点的对应关系如表 3-4 所示：

表 3-4 指令 GT_HomeSns 的参数和各轴原点的对应关系

状态位	7	6	5	4	3	2	1	0
Home 开关	Home7	Home6	Home5	Home4	Home3	Home2	Home1	Home0

3.2.2.3 编码器计数方向设置

只有电机旋转的正方向和编码器计数的正方向一致，才能保证运动控制器正常工作。在实际应用中，可能由于接线错误或者电机设置不当，造成电机转向与编码器计数方向相反，导致运动控制器不能正常工作。此时可以调用指令 GT_EncSns 修改运动控制器各轴的编码器计数方向。

当指令参数的某个状态位为 1 时，将所对应的控制轴的编码器计数方向取反，指令参数的状态位定义如表 3-5 所示：

表 3-5 指令 GT_EncSns 的参数和各轴编码器的对应关系

状态位	7	6	5	4	3	2	1	0
对应轴	轴 8	轴 7	轴 6	轴 5	轴 4	轴 3	轴 2	轴 1

3.2.2.4 驱动报警信号

运动控制器为每个控制轴都提供了驱动报警信号的输入接口。如果不需要驱动报警输入信号，可调用指令 GT_AlarmOff，使驱动报警无效，其接线方式请参见《GE 系列运动控制器用户手册》中的“专用输入、输出连接方法”一栏。

当控制轴处于驱动报警状态时，运动控制器拒绝针对该控制轴的运动指令。在伺服电机驱动器的报警消除以后，必须调用指令 GT_ClrSts 清除该轴的驱动报警状态。

GE-X00-XG 运动控制器默认驱动报警输入信号无效。

GE-X00-XV 运动控制器默认驱动报警输入信号有效。

3.3 控制轴初始化

3.3.1 指令列表

表 3-6 运动控制轴初始化指令列表

指 令	说 明
GT_AxisOn	使能指定控制轴
GT_ClrSts	清除指定控制轴状态
GT_StepDir	设置指定控制轴的脉冲输出方式为“脉冲/方向”方式
GT_StepPulse	设置指定控制轴的脉冲输出方式为“正/负脉冲”方式
GT_CtrlMode	设置指定控制轴的模拟量输出或脉冲输出模式（GE-X00-XV）
GT_CloseLp	将指定控制轴设置为闭环控制模式（GE-X00-PV）
GT_OpenLp	将指定控制轴设置为开环模式（GE-X00-PV）
GT_SetKp	设置指定控制轴的误差比例增益（GE-X00-XV）
GT_SetKi	设置指定控制轴的误差积分增益（GE-X00-XV）
GT_SetKd	设置指定控制轴的误差微分增益（GE-X00-XV）
GT_SetKvff	设置指定控制轴的速度前馈增益（GE-X00-XV）
GT_SetKaff	设置指定控制轴的加速度前馈增益（GE-X00-XV）
GT_SetILmt	设置指定控制轴的误差积分极限（GE-X00-XV）
GT_SetMtrLmt	设置指定控制轴输出电压饱和极限（GE-X00-XV）
GT_SetMtrBias	设置指定控制轴的零漂电压补偿（GE-X00-XV）

3.3.2 例程

例程 3-1 指令返回值处理函数

```

void error(short rtn)
{
    switch(rtn)
    {
        case -1:
            printf("\a\nCommunication Error !");           break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");                   break;
        case 2:
            printf("\a\nRadius or chord is 0 !");             break;
        case 3:
            printf("\a\nLength is 0 or overflow !");          break;
        case 4:
    
```

```
        printf("\a\nVelocity or acceleration is less then 0 !");
        break;

    case 5:
        printf("\a\nChord is greater than diameter !"); break;
    case 7:
        printf("\a\nParameter error !"); break;
    default:
        printf("\a\nError Code =%d",rtn); break;
    }
}
```

例程 3-2 GE-X00-XG 运动控制器的控制轴初始化

```
void AxisInitial(short axis_num,unsigned short limit)
{
    //控制轴初始化函数
    short rtn;
    rtn=GT_LmtSns(limit); //设置限位开关触发电平
    error(rtn);
    for(short i=1;i<=axis_num;++i)
    {
        rtn=GT_StepPulse(i); //设置轴输出正负脉冲信号
        error(rtn);
        rtn=GT_AxisOn(i); //驱动使能
        error(rtn);
        rtn=GT_ClrSts(i); //控制轴状态寄存器复位
        error(rtn);
        delay(200); //插入适当延时
        //等待驱动器伺服就绪
    }
}
```

例程 3-3 GE-X00-XV 运动控制器控制轴初始化（输出脉冲）

```
void AxisInitial(short axis_num,unsigned short limit)
{
    //控制轴初始化函数
    short rtn;
    rtn=GT_LmtSns(limit); //设置限位开关触发电平
    error(rtn);
    for(short i=1;i<=axis_num;++i)
    {
        rtn=GT_CtrlMode(i,1); //设置控制轴为脉冲输出模式
        error(rtn);
        rtn=GT_StepPulse(i); //设置轴输出正负脉冲信号
        error(rtn);
        rtn=GT_AxisOn(i); //驱动使能
        error(rtn);
        rtn=GT_ClrSts(i); //控制轴状态寄存器复位
    }
}
```

```
        error(rtn);
        delay(200);                //插入适当延时
                                    //等待驱动器伺服就绪
    }
}
```

例程 3-4 GE-X00-XV 运动控制器控制轴初始化（输出模拟电压）

```
void AxisInitial(short axis_num,unsigned short limit,double kp)
{
    //控制轴初始化函数
    short rtn;
    rtn=GT_LmtSns(limit);           //设置限位开关触发电平
    error(rtn);
    for(short i=1;i<=axis_num;++i)
    {
        rtn=GT_SetKp(i,kp);         //设置控制轴比例增益
        error(rtn);
        rtn=GT_Update(i);           //参数刷新，使所设 Kp 生效
        error(rtn);
        rtn=GT_AxisOn(i);           //伺服使能控制轴
        error(rtn);
        rtn=GT_ClrSts(i);           //控制轴状态寄存器复位
        error(rtn);
        delay(200);                 //插入适当延时
                                    //等待驱动器伺服就绪
    }
}
```

3.2.3 重点说明

3.2.3.1 设置控制轴为 “模拟电压输出” 或 “脉冲输出”

GE-X00-XG 运动控制器只能实现脉冲输出，可以控制步进电机，也可以控制接收脉冲输入的伺服电机。GE-X00-XV 运动控制器可以实现脉冲输出或者模拟电压输出，上电默认为模拟电压输出。通过调用指令 GT_CtrlMode 可以切换控制轴的输出方式。

3.2.3.2 设置脉冲输出方式 “脉冲+方向” / “正负脉冲”

运动控制器默认脉冲输出方式为“脉冲+方向”。调用 GT_StepPulse 指令可将控制轴设置为“正负脉冲”方式；调用 GT_StepDir 指令可将控制轴设置为“脉冲+方向”方式。

3.2.3.3 设置输出模拟电压

GE-X00-XV 运动控制器默认输出模拟电压。

各控制轴可能存在零漂电压，应当调用指令 GT_SetMtrBias 正确设置控制轴零漂电压补偿，才能保证运动控制器的准确定位。

3.4 控制轴实际位置设置

3.4.1 指令列表

表 3-7 设置实际位置指令列表

指 令	说 明
GT_ZeroPos	控制轴实际位置和目标位置清零
GT_SetAtlPos	设置控制轴的实际位置

3.4.2 重点说明

3.4.2.1 实际位置和目标位置清零

调用指令 GT_ZeroPos 将控制轴的实际位置和目标位置清为 0。该指令只在指定轴运动停止时有效，否则被视为非法指令。

3.4.2.2 设置指定轴的实际位置

调用指令 GT_SetAtlPos 将指定轴的实际位置和目标位置修改为设定值。该指令只在指定轴运动停止时有效，否则被视为非法指令。

3.5 运动控制器初始化例程

例程 3-5 GE-300-SG 运动控制器初始化一

```
#include <stdio.h>
#include "ges.h"

void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !");          break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");                  break;
        case 2:
            printf("\a\nRadius or chord is 0 !");            break;
        case 3:
            printf("\a\nLength is 0 or overflow !");         break;
        case 4:
```

```
        printf("\a\nVelocity or acceleration is less then 0 !");
        break;
    case 5:
        printf("\a\nChord is greater than diameter !"); break;
    case 7:
        printf("\a\nParameter error !"); break;
    default:
        printf("\a\nError Code = %d",error); break;
    }
}

void main()
{
    GT_HookCommand(CommandHandle); //挂接指令处理函数
    GT_Open(0x300); //打开 ISA 总线的控制器 ,
                    //其基地址为 0x300。
                    //打开 PCI 总线的控制器 ,
                    //不需要传入参数。
    GT_Reset(); //复位运动控制器
    GT_LmtSns(0); //设置限位开关触发电平
    for(short i=1;i<=3;++i)
    {
        GT_StepPulse(i); //设置轴输出正负脉冲信号
        GT_AxisOn(i); //驱动使能
        GT_ClrSts(i); //控制轴状态寄存器复位
    }
}
```

例程 3-6 GE-300-SG 运动控制器初始化二

```
#include <stdio.h>
#include "ges.h"

void error(short rtn)
{
    switch(rtn)
    {
        case -1:
            printf("\a\nCommunication Error !"); break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !"); break;
        case 2:
            printf("\a\nRadius or chord is 0 !"); break;
    }
}
```

```
        case 3:
            printf("\a\nLength is 0 or overflow !");          break;
        case 4:
            printf("\a\nVelocity or acceleration is less then 0 !");
                                                                    break;
        case 5:
            printf("\a\nChord is greater than diameter !"); break;
        case 7:
            printf("\a\nParameter error !");                  break;
        default:
            printf("\a\nError Code = %d", rtn);                break;
    }
}

void main()
{
    short rtn;
    rtn=GT_Open(0x300);          //打开 ISA 总线的控制器，其基地址
                                //为 0x300。
                                //打开 PCI 总线的控制器，则不需传
                                //入参数。

    error(rtn);
    rtn=GT_Reset();
    error(rtn);
    rtn=GT_LmtSns(0);           //设置限位开关触发电平
    error(rtn);
    for(short i=1; i<=3; ++i)
    {
        rtn=GT_StepPulse(i);    //设置轴输出正负脉冲信号
        error(rtn);
        rtn=GT_AxisOn(i);       //驱动使能
        error(rtn);
        rtn=GT_ClrSts(i);       //控制轴状态寄存器复位
        error(rtn);
    }
}
```


第四章 运动控制器状态检测

用户可以从运动控制器提供的状态寄存器读取控制轴状态、连续轨迹运动状态（GE-X00-SX）和指令状态。

4.1 指令列表

表 4-1 状态检测指令列表

指 令	说 明
GT_GetSts	读取指定控制轴状态。
GT_GetCrdSts	读取连续轨迹运动状态（仅用于 GE-X00-SX）。
GT_ClrSts	清除指定控制轴状态。
GT_GetMode	读取指定控制轴工作模式（仅用于 GE-X00-PX）。
GT_RstSts	清除指定控制轴的指定状态位。

4.2 重点说明

对于 GE-X00-PX 运动控制器，调用指令 GT_GetSts 读取控制轴的状态寄存器，控制轴状态寄存器的 bit10 指示控制轴的运动状态。

对于 GE-X00-SX 运动控制器，调用指令 GT_GetCrdSts 读取连续轨迹运动状态寄存器，连续轨迹运动状态寄存器的 bit0 指示连续轨迹运动状态。

对于 GE-X00-SX 运动控制器，不建议调用指令 GT_GetSts 检测 bit10 来判断是否有轨迹运动（轨迹运动的准备和预处理需要一定时间，启动轨迹运动时该标志位尚未置 1，因此存在一段时间的滞后）。

控制轴运动状态标志位和连续轨迹运动状态标志位表示控制器内部加/减速运动规划的状态，而不是控制轴的实际运动状态。即只表示控制器的加/减速控制是否已完成一次规划任务，规划到目标位置。控制轴是否真正到达目标位置还取决于实际系统的伺服滞后，稳定性及其它条件。

4.3 控制轴状态寄存器

运动控制器为每个控制轴提供一个状态寄存器。在运动过程中，用户可以通过调用指令 GT_GetSts 查询这些状态位，全面了解控制轴的运动情况。GE-X00-SX 的控制轴状态寄存器为 16 位，各状态位的定义如表 4-2 所示；GE-X00-PX 的控制轴状态寄存器为 32 位，各状态位定义如表 4-3 所示。其中标志位 8-15 指示控制轴的运行状态信息和轴号，用户不能对其进行修改，而 0-7 位表示控制轴不同的事件状态。这些事件一旦发生，相应的标志位置 1，并一直保持。用户可调用指令 GT_ClrSts 清除标志位。

表 4-2 GE-X00-SX 运动控制器控制轴状态寄存器定义

位	定 义												
0	保留。												
1	伺服电机驱动器报警标志位。如果控制轴连接的伺服电机驱动器报警，该位置 1。												
2	保留。												
3	Index/Home 标志位。在设置位置捕获指令后，控制器检测到要求的 Index/Home 捕获条件，该位置 1。												
4	控制轴跟随误差超过所设定的位置误差极限时置 1。												
5	正向限位开关触发标志位。如果正向限位开关触发，该位置 1。												
6	负向限位开关触发标志位。如果负向限位开关触发，该位置 1。												
7	保留。												
8	保留。												
9	电机伺服使能/禁止状态：1 表示使能，0 表示禁止（默认）。												
10	运动状态标志位。1 表示运动，0 表示静止（默认）。												
11	限位开关输入“有效/无效”状态：1 表示有效（默认），0 表示无效。												
12 13	控制轴号标志(13bit=高位,12bit=低位)。控制轴号的编码如下： <table><tr><td>Bit 13</td><td>Bit12</td><td>轴</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>1</td><td>0</td><td>3</td></tr></table>	Bit 13	Bit12	轴	0	0	1	0	1	2	1	0	3
Bit 13	Bit12	轴											
0	0	1											
0	1	2											
1	0	3											
14	指示控制轴 Home 捕获使能：0 表示禁止（默认），1 表示使能。												
15	指示控制轴 Index 捕获使能：0 表示禁止（默认），1 表示使能。												

表 4-3 GE-X00-PX 运动控制器控制轴状态寄存器定义

位	定 义
0	保留。
1	伺服电机驱动器报警标志位。如果控制轴连接的伺服电机驱动器报警，该位置 1。
2	保留。
3	Index/Home 标志位。在设置位置捕获指令后，控制器检测到要求的 Index/Home 捕获条件，该位置 1。
4	控制轴跟随误差超过所设定的位置误差极限时置 1。
5	正向限位开关触发标志位。如果正向限位开关触发，该位置 1。
6	负向限位开关触发标志位。如果负向限位开关触发，该位置 1。
7	控制轴指令出错时置 1。
8	指示控制轴控制模式：0 表示开环，1 表示闭环（默认）。
9	电机伺服使能/禁止状态：1 表示使能，0 表示禁止（默认）。
10	运动状态标志位。1 表示运动，0 表示静止（默认）。
11	限位开关输入“有效/无效”状态：1 表示有效（默认），0 表示无效。
12~13	保留。
14	指示控制轴 Home 捕获使能：0 表示禁止（默认），1 表示使能。
15	指示控制轴 Index 捕获使能：0 表示禁止（默认），1 表示使能。
16~26	保留。
27	控制轴的编码器异常时置 1。
28~30	指示控制轴的轴号：
	Bit30 Bi29 Bit28 轴
	0 0 0 1
	0 0 1 2
	0 1 0 3
	0 1 1 4
	1 0 0 5
	1 0 1 6
	1 1 0 7
	1 1 1 8
31	保留。

4.4 控制轴模式寄存器

GE-X00-PX 运动控制器为每个控制轴提供一个 16 位的模式寄存器。用户调用指令 GT_GetMode 查询运动控制器控制轴的工作模式。控制轴模式寄存器的定义如下表所示：

表 4-4 GE-X00-PX 运动控制器控制轴模式寄存器定义

位	定 义																
0~6	保留。																
7	指示运动出错自动停止是否有效：0 表示无效（默认），1 表示有效。																
8~10	保留。																
11~13	指示当前轴的运动模式： <table><tr><td>Bit13</td><td>Bit12</td><td>Bit11</td><td>运动模式</td></tr><tr><td>0</td><td>0</td><td>0</td><td>梯形曲线（默认）</td></tr><tr><td>0</td><td>0</td><td>1</td><td>速度模式</td></tr><tr><td>0</td><td>1</td><td>0</td><td>S 曲线</td></tr></table>	Bit13	Bit12	Bit11	运动模式	0	0	0	梯形曲线（默认）	0	0	1	速度模式	0	1	0	S 曲线
Bit13	Bit12	Bit11	运动模式														
0	0	0	梯形曲线（默认）														
0	0	1	速度模式														
0	1	0	S 曲线														
14~15	保留。																

4.5 连续轨迹运动状态寄存器

GE-X00-SX 控制器提供一个寄存器记录连续轨迹运动状态。该寄存器由运动控制器统一管理。用户调用指令 GT_GetCrdSts 读取该寄存器，用户程序不能直接清除或设置该寄存器的状态。连续轨迹运动状态寄存器定义如下表所示：

表 4-5 连续轨迹运动状态寄存器定义

位	定 义
0	指示轨迹运动状态：0 表示有轨迹运动，1 表示无轨迹运动（默认）。
1	缓冲区是否打开：0 表示打开，1 表示关闭（默认）。
2	轨迹运动预处理是否正常：0 表示正常（默认），1 表示预处理时间不够，运动控制器不能正常工作。
3	保留。
4	指示一段轨迹运动状态，0 表示正在进行轨迹运动，1 表示一段轨迹运动完成（默认）。
5~6	保留。
7	指示指令输入状态，0 表示缓冲区指令输入或执行状态，1 表示立即指令输入状态（默认）。
8	保留。
9	轨迹运动中相关控制轴是否出现异常（如限位开关触发），0 表示相关控制轴正常（默认），1 表示相关控制轴出现异常，并自动停止连续轨迹运动。
10	指示脉冲输出是否出现异常，0 表示正常（默认），1 表示脉冲输出异常。

位	定 义
11~12	保留。
13	缓冲区是否空，0 表示不空，1 表示空（默认）。
14~15	保留。

第五章 点位运动 (GE-X00-PX)

GE-X00-PX 运动控制器可以进行位置控制或速度控制。

在位置控制模式下提供了 2 种加减速方式：梯形曲线加减速和 S 曲线加减速。梯形曲线加减速允许在运动过程当中随时修改目标位置和目标速度；S 曲线加减速允许在运动过程当中随时修改目标位置。在加速度相等的情况下梯形曲线具有较短的加减速时间，而 S 曲线的运动比较平滑。应当针对具体应用场合选择相应的加减速模式。

在速度控制模式下，运动控制器按照所设定的加速度加速或减速到目标速度，在运动过程当中可以随时修改目标速度。

5.1 梯形曲线

5.1.1 指令列表

表 5-1 设置梯形曲线指令列表

指 令	说 明
GT_PrflT	设置指定控制轴为梯形曲线
GT_SetAcc	设置指定控制轴的加速度
GT_SetVel	设置指定控制轴的目标速度
GT_SetPos	设置指定控制轴的目标位置
GT_Update	刷新指定控制轴的运动控制参数

梯形曲线参数设置范围

参 数	取 值 范 围	单 位
目标位置	-1,073,741,824 ~ 1,073,741,823	pulse
最大速度	0 ~ 40,957	pulse/ms
加速度	0 ~ 102,400	pulse/ms ²

5.1.2 例程

控制轴 1 所对应的伺服电机工作在速度模式下，丝杆导程为 10mm，电机每转脉冲数为 10000pulse/r（四倍频后），则脉冲当量为 1 um/pulse。

该轴采用梯形曲线加减速，加速度为 4m/s²，目标速度为 10m/min，目标位置为 200mm。

$$\text{加速度} = \frac{4m/s^2}{1\mu m/pulse} = \frac{4*10^6\mu m/(10^3ms)^2}{1\mu m/pulse} = 4pulse/ms^2$$

$$\text{速度} = \frac{10m/min}{1\mu m/pulse} = \frac{10*10^6\mu m/(60*10^3ms)}{1\mu m/pulse} = 166.7pulse/ms$$

$$\text{位置} = \frac{200mm}{1\mu m/pulse} = \frac{200*10^3\mu m}{1\mu m/pulse} = 200000pulse$$

例程 5-1 梯形曲线例程

```
void AxisRunT(unsigned short axis,long pos,double vel,double acc)
{
    GT_PrflT(axis);                //设置为梯形曲线加减速
    GT_SetPos(axis,pos);            //设置控制轴目标位置
    GT_SetVel(axis,vel);            //设置控制轴速度
    GT_SetAcc(axis,acc);            //设置控制轴加速度
    GT_Update(axis);                //刷新控制轴参数
}

void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !"); break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");      break;
        case 7:
            printf("\a\nParameter Error !");      break;
        default:
            printf("\a\nError Code =%d",error);  break;
    }
}

void main()
{
    GT_HookCommand(CommandHandle); //挂接错误处理函数
    GT_Open(0x300)                 //打开运动控制器
    GT_Reset();                    //复位运动控制器
}
```

```
AxisInitial(1,0,10);           //控制轴初始化，引用例程 3-4
AxisRunT(1,200000,166.7,4);    //控制轴按照梯形曲线运动
}
```

5.1.3 重点说明

梯形曲线加减速的“速度 - 时间”曲线如图所示：

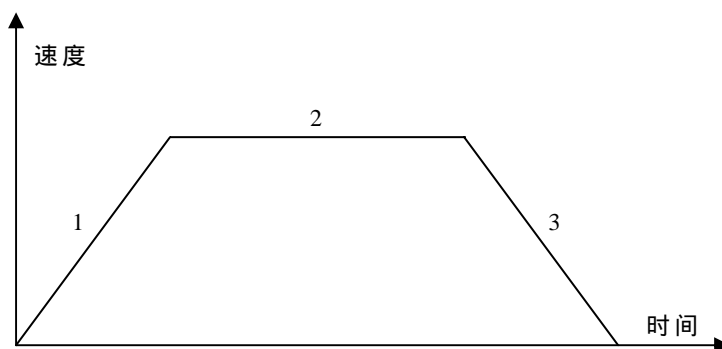


图 5-1 梯形曲线速度曲线

梯形曲线的运动规律可以划分为 3 个阶段：

1. 加速段：速度按照设定的加速度从 0 加速到目标速度。
2. 匀速段：保持目标速度，直至到达减速点。
3. 减速段：速度按照设定的加速度从目标速度减速到 0。

梯形曲线可以在运动过程当中修改目标位置和目标速度。

如果在运动过程中增大目标位置，运动控制器根据新的目标位置重新计算减速点，到达新的减速点后开始减速。

如果在运动过程当中减小目标位置，运动控制器根据新的目标位置重新计算减速点。如果当前位置已经超越新的减速点，运动控制器首先减速到 0，然后反向运动到新的目标位置。如果当前位置没有超越新的减速点，运动控制器到达新的减速点后开始减速。

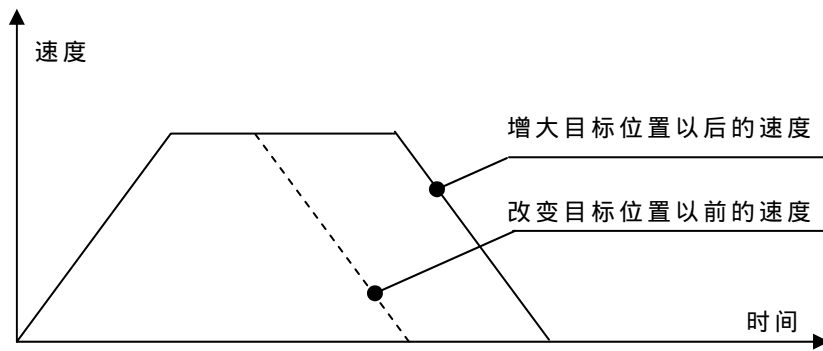


图 5-2 梯形曲线在运动过程中增大目标位置

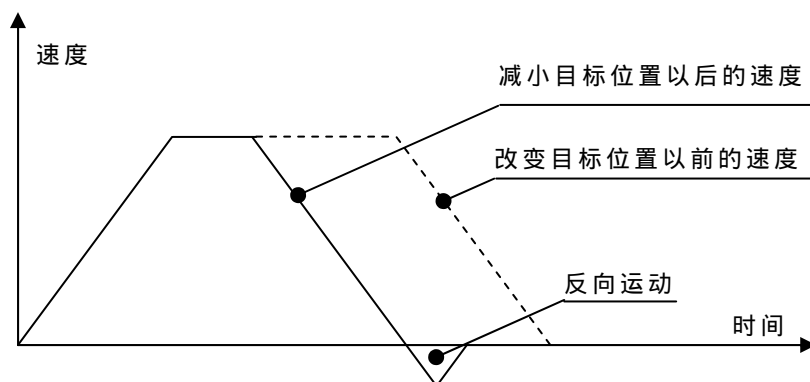


图 5-3 梯形曲线在运动过程中减小目标位置

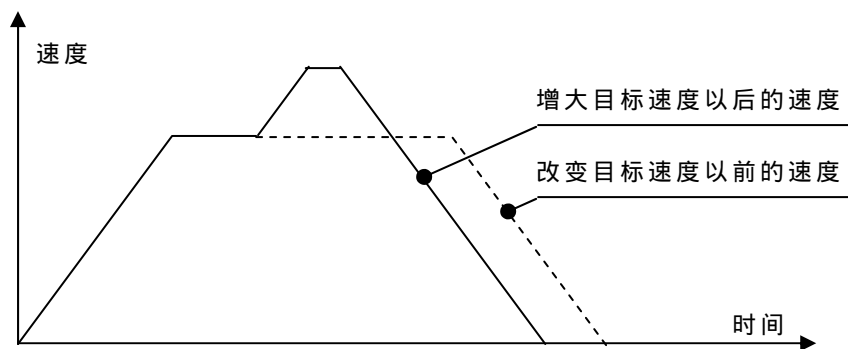


图 5-4 梯形曲线在运动过程中增大目标速度

5.2 S 曲线

5.2.1 指令列表

表 5-2 设置 S 曲线指令列表

指 令	说 明
GT_PrflS	设置指定控制轴为 S 曲线
GT_SetMAcc	设置指定控制轴的最大加速度
GT_SetJerk	设置指定控制轴的加加速度
GT_SetVel	设置指定控制轴的目标速度
GT_SetPos	设置指定控制轴的目标位置
GT_Update	更新指定控制轴的运动控制参数

S 曲线参数设置范围

参 数	取 值 范 围	单 位
目标位置	-1,073,741,824 ~ 1,073,741,823	pulse
最大速度	0 ~ 40,957	pulse/ms
加速度	0 ~ 102,400	pulse/ms ²
加加速度	0 ~ 256,000	pulse/ms ³

5.2.2 例程

控制轴 1 所对应的伺服电机工作在速度模式，丝杆导程为 10mm，电机每转脉冲数为 10000pulse/r（四倍频后），则脉冲当量为 1 μm/pulse。

该轴采用 S 曲线加减速，加加速度为 1m/s³，加速度为 4m/s²，目标速度为 10m/min，目标位置为 200mm。

$$\text{加加速度} = \frac{1\text{m/s}^3}{1\mu\text{m/pulse}} = \frac{10^6\mu\text{m}/(10^3\text{ms})^3}{1\mu\text{m/pulse}} = 0.001\text{pulse/ms}^3$$

$$\text{加速度} = \frac{4\text{m/s}^2}{1\mu\text{m/pulse}} = \frac{4*10^6\mu\text{m}/(10^3\text{ms})^2}{1\mu\text{m/pulse}} = 4\text{pulse/ms}^2$$

$$\text{速度} = \frac{10\text{m/min}}{1\mu\text{m/pulse}} = \frac{10*10^6\mu\text{m}/(60*10^3\text{ms})}{1\mu\text{m/pulse}} = 166.7\text{pulse/ms}$$

$$\text{位置} = \frac{200\text{mm}}{1\mu\text{m} / \text{pulse}} = \frac{200 * 10^3 \mu\text{m}}{1\mu\text{m} / \text{pulse}} = 200000 \text{pulse}$$

例程 5-2 S 曲线例程

```
void AxisRunS(unsigned short axis,long pos,double vel,double macc,double
jerk)
{
    GT_PrflS(axis);                //设置为 S 曲线加减速
    GT_SetPos(axis,pos);           //设置控制轴目标位置
    GT_SetVel(axis,vel);           //设置控制轴速度
    GT_SetMAcc(axis,macc);         //设置控制轴加速度
    GT_SetJerk(axis,jerk);         //设置控制轴加加速度
    GT_Update(axis);               //刷新控制轴参数
}

void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !"); break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");      break;
        case 7:
            printf("\a\nParameter Error !");    break;
        default :
            printf("\a\nError Code =%d",error); break;
    }
}

void main()
{
    GT_HookCommand(CommandHandle); //挂接错误处理函数
    GT_Open(0x300)                 //打开运动控制器
    GT_Reset();                    //复位运动控制器

    AxisInitial(1,0,10);           //控制轴初始化 , 引用例程 3-4
    AxisRunS(1,200000,166.7,4,0.001);
}
```

5.2.3 重点说明

S 曲线加减速的“速度 - 时间”曲线如下所示：

S 曲线的运动规律可以划分为 7 个阶段：

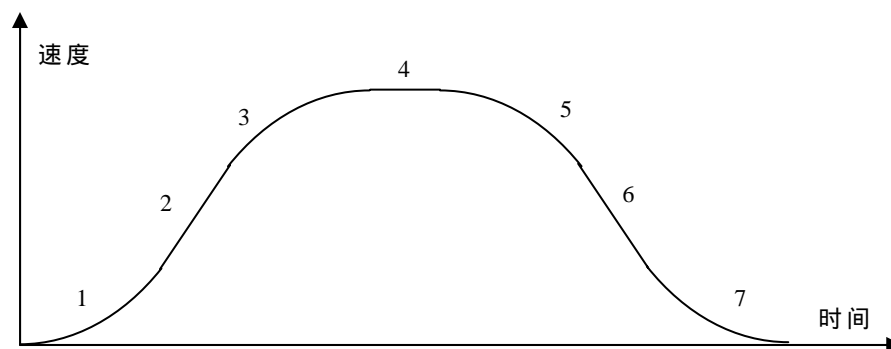


图 5-5 S 曲线速度曲线

1. 变加速段：加速度按照设定的加加速度从 0 递增到最大加速度，速度按照加速度递增。
2. 匀加速段：加速度保持最大加速度不变，速度按照最大加速度递增。
3. 变加速段：加速度按照设定的加加速度从最大加速度递减到 0，速度按照加速度递增。
4. 匀速段：加速度为 0，速度保持目标速度不变。
5. 变减速段：加速度按照设定的加加速度从 0 递增到最大加速度，速度按照加速度递减。
6. 匀减速段：加速度保持最大加速度不变，速度按照最大加速度递减。
7. 变减速段：加速度按照设定的加加速度从最大加速度递减到 0，速度按照加速度递减。

S 曲线可以在运动过程当中修改目标位置。

如果在运动过程当中增大目标位置，运动控制器根据新的目标位置重新计算减速点，到达新的减速点后才开始减速。

如果在运动过程当中减小目标位置，运动控制器根据新的目标位置重新计算减速点。如果当前位置已经超越新的减速点，运动控制器首先减速到 0，然后反向运动到新的目标位置。如果当前位置没有超越新的减速点，运动控制器到达新的减速点后开始减速。

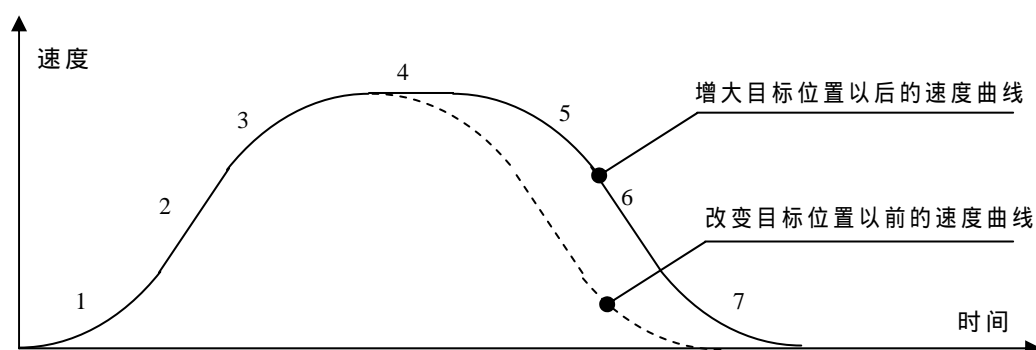


图 5-6 S 曲线在运动过程中增大目标位置

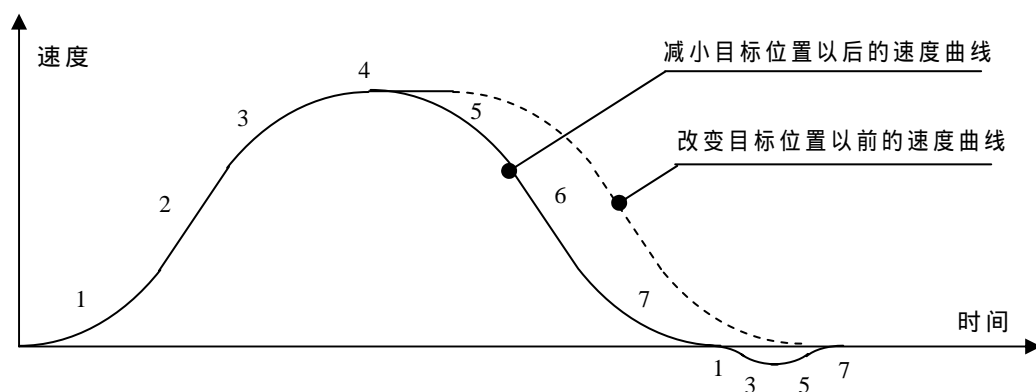


图 5-7 S 曲线在运动过程中减小目标位置

5.3 速度控制

5.3.1 指令列表

表 5-3 设置速度控制指令列表

指令	说明
GT_PrflV	设置指定控制轴为速度控制模式
GT_SetAcc	设置指定控制轴的加速度
GT_SetVel	设置指定控制轴的目标速度
GT_Update	更新指定控制轴的运动控制参数

参数设置范围

参 数	取 值 范 围	单 位
最大速度	-40,958 ~ 40,957	pulse/ms
加速度	0 ~ 102,400	pulse/ms ²

5.3.2 例程

控制轴 1 所对应的伺服电机工作在速度模式，丝杆导程为 10mm，电机每转脉冲数为 10000pulse/r（四倍频后），则脉冲当量为 1 μm/pulse。

该轴采用速度控制模式，加速度为 4m/s²，目标速度为 10m/min，脉冲当量为 1μm/pulse。

$$\text{加速度} = \frac{4m/s^2}{1\mu m/pulse} = \frac{4*10^6\mu m/(10^3ms)^2}{1\mu m/pulse} = 4pulse/ms^2$$

$$\text{速度} = \frac{10m/min}{1\mu m/pulse} = \frac{10*10^6\mu m/(60*10^3ms)}{1\mu m/pulse} = 166.7pulse/ms$$

例程 5-3 速度控制例程

```
void AxisRunV(unsigned short axis, double vel,double acc)
{
    GT_PrflV(axis);           //设置为速度控制模式
    GT_SetVel(axis,vel);      //设置控制轴速度
    GT_SetAcc(axis,acc);      //设置控制轴加速度
    GT_Update(axis);          //刷新控制轴参数
}

void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !"); break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");      break;
        case 7:
            printf("\a\nParameter Error !");      break;
        default:
            printf("\a\nError Code=%d",error);  break;
    }
}
```

```
    }  
}  
  
void main()  
{  
    GT_HookCommand(CommandHandle); //挂接错误处理函数  
    GT_Open(0x300)                  //打开运动控制器  
    GT_Reset();                     //复位运动控制器  
    AxisInitial(1,0,10);             //控制轴初始化，引用例程 3-4  
    AxisRunV(1,166.7,4);  
}
```

5.3.3 重点说明

在速度控制模式下，控制轴以设定的加速度加速到目标速度以后，一直保持目标速度。在运动过程当中可以随时改变目标速度。

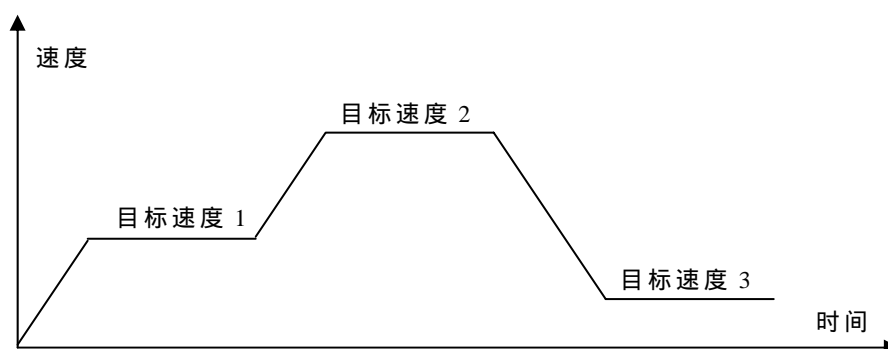


图 5-8 速度控制模式下改变目标速度

5.4 参数刷新机制

GE-X00-PX 运动控制器的指令按照生效方式分为 2 类：立即生效指令和刷新生效指令。

立即生效指令在指令发送成功以后生效。

刷新生效指令在调用指令 GT_Update 或 GT_MltiUpdt 之后生效。

指令 GT_Update 可实现控制轴多个参数的同时更新。

指令 GT_MltiUpdt 可实现多个控制轴多个参数的同时更新。

表 5-4 刷新生效指令列表

指 令	说 明
GT_SetPos	设置指定控制轴的目标位置
GT_SetVel	设置指定控制轴的目标速度
GT_SetAcc	设置指定控制轴的加速度
GT_SetMAcc	设置指定控制轴的最大加速度，仅用于 S 曲线
GT_SetJerk	设置指定控制轴的加加速度，仅用于 S 曲线
GT_SetKp	设置指定控制轴的比例增益
GT_SetKi	设置指定控制轴的积分增益
GT_SetKd	设置指定控制轴的微分增益
GT_SetKvff	设置指定控制轴的速度前馈
GT_SetKaff	设置指定控制轴的加速度前馈
GT_SetMtrBias	设置指定控制轴的零漂电压补偿
GT_SetMtrLmt	设置指定控制轴的输出电压饱和极限
GT_SetILmt	设置指定控制轴的误差积分饱和极限
GT_SetPosErr	设置指定控制轴的跟随误差极限

5.5 停止控制轴的运动

5.5.1 指令列表

表 5-5 停止控制轴运动指令列表

指 令	说 明
GT_SmthStp	平滑停止控制轴
GT_AbptStp	急停控制轴

5.5.2 重点说明

如果需要在运动过程当中停止控制轴，可以调用指令 GT_SmthStp 或 GT_AbptStp。

GT_SmthStp 是立即指令，能够平滑停止控制轴的运动。调用指令 GT_SmthStp 时，运动控制器将目标速度设置为 0，并根据用户所设定的加速度平滑减速到 0。当控制轴平滑停止运动完成之后，用户才能重新设置控制轴的运动参数继续运动。指令 GT_SmthStp 的效果如下所示：

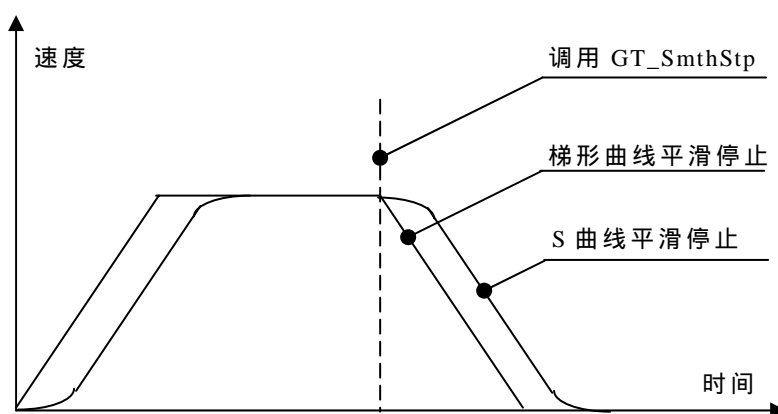


图 5-9 平滑停止速度曲线

GT_AbptStp 是立即指令，能够紧急停止控制轴的运动。调用 GT_AbptStp 指令时，无论控制轴处于何种加减速模式，都将立即减速到 0。指令 GT_AbptStp 的效果如下所示：

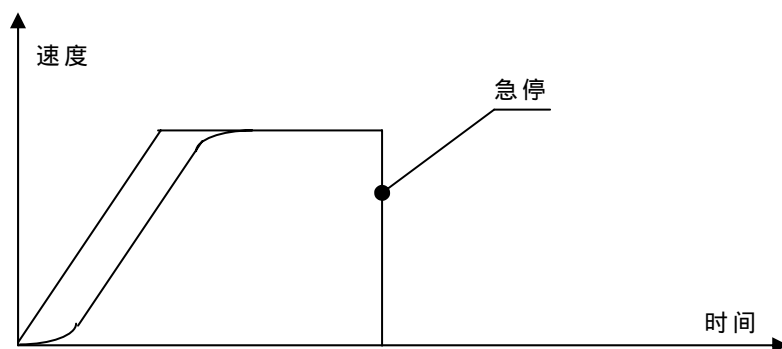


图 5-10 急停速度曲线

第六章 连续轨迹运动

GE-X00-SX 运动控制器（GE 连续轨迹运动控制器）可以实现直线插补、圆弧插补，通过前瞻预处理能够实现小线段高速平滑的连续轨迹运动。

6.1 进给速度、加速度设置

6.1.1 指令列表

表 6-1 设置轨迹运动进给速度、加速度指令列表

指令	说明
GT_SetSynVel	设置轨迹运动的进给速度
GT_SetSynAcc	设置轨迹运动的加速度
GT_SetStrtVel	设置轨迹运动的启动速度
GT_SetStpAcc	设置轨迹运动的急停加速度
GT_SetMaxVel	设置轨迹运动的最大速度

6.1.2 例程

例程 6-1 设置轨迹运动的进给速度、加速度

该程序示例一个轨迹运动的进给速度、加速度的设置以及单位说明：

丝杆导程 5mm，电机每转脉冲数为 10000pulse（四倍频后）。

进给速度（WorkVel）为 3m/min，进给加速度（WorkAcc）为 0.9m/s^2 ，启动速度（StartVel）为 300pulse/s，最大速度（MaxVel）为 15m/min，急停加速度（StopAcc）为 3m/s^2 。

$$WorkVel = \frac{3000\text{mm}}{60 * 10^3 \text{ms}} \times \frac{10000 \text{pulse}}{5\text{mm}} = 100 \text{pulse/ms}$$

$$WorkAcc = \frac{900\text{mm}}{(10^3 \text{ms})^2} \times \frac{10000 \text{pulse}}{5\text{mm}} = 1.8 \text{pulse/ms}^2$$

$$StartVel = \frac{300 \text{pulse}}{10^3 \text{ms}} = 0.3 \text{pulse/ms}$$

$$MaxVel = \frac{15000\text{mm}}{60 * 10^3 \text{ms}} \times \frac{10000 \text{pulse}}{5\text{mm}} = 500 \text{pulse/ms}$$

$$StopAcc = \frac{3000\text{mm}}{(10^3 \text{ms})^2} \times \frac{10000 \text{pulse}}{5\text{mm}} = 6 \text{pulse/ms}^2$$

```
void MotionInitial()
{
    short rtn;
    double StartVel, MaxVel, StopAcc;
    StartVel=0.3;                //启动速度为 300HZ
    MaxVel=500;                  //最大速度为 15m/min
    StopAcc=6;                   //急停加速度为 3m/s2

    rtn=GT_SetStrtVel(StartVel); //设置系统启动速度
    error(rtn);
    rtn=GT_SetMaxVel(MaxVel);    //设置系统最大速度
    error(rtn);
    rtn=GT_SetStpAcc(StopAcc);   //设置系统急停加速度
    error(rtn);
}

void main()
{
    short rtn;
    double WorkVel, WorkAcc;
    AxisInitial(3,0);            //参考例程 3-2 ~ 3-4
    MotionInitial();
    WorkVel=100;                  //进给速度为 3m/min
    WorkAcc=1.8;                  //进给加速度为 0.9m/s2
    rtn=GT_SetSynAcc(WorkAcc);    //设置进给加速度
    error(rtn);
    rtn=GT_SetSynVel(WorkVel);    //设置进给速度
    error(rtn);
}
```

6.1.3 重点说明

6.1.3.1 GT_SetStrtVel

该指令设置轨迹运动的启动速度，单位为 **pulse/ms**。一般在对运动控制器进行参数初始化时调用该指令，该指令调用后立即生效。在运动过程中可以改变启动速度，但会影响运动控制器的速度规划，**建议用户只在初始化时调用该指令**。

对于步进电机，总存在一个起跳频率，步进电机可以直接以起跳频率为启动速度（不必是零速）开始加速到进给速度，可减少加速时间，提高加工效率。运动控制器的默认起跳速度为：500HZ。

6.1.3.2 GT_SetMaxVel

该指令设置轨迹运动的最大速度，单位为 **pulse/ms**。一般在对运动控制器进行参数初始化时调用该指令，该指令在调用后立即生效。在运动过程中可以改变最大速度，但会影响运动控制器的速度规划，**建议用户只在初始化时调用该指令**。

根据步进电机的矩频特性，运行频率高于某个频率时，输出力矩下降非常快，可能出现带动不了负载且电机堵转的情况。建议用户在初始化时根据机床负载和电机工作情况调用指令 GT_SetMaxVel 设置最大速度。运动控制器的默认最大速度为：256KHZ。

6.3.1.3 GT_SetSynVel

该指令设置轨迹运动的进给速度，单位为 **pulse/ms**。该指令影响此后所有直线插补和圆弧插补指令的进给速度，直到再次调用该指令设置新的进给速度为止。

6.3.1.4 GT_SetSynAcc

该指令设置轨迹运动的进给加速度，单位为 **pulse/ms²**。该指令只能在立即指令工作方式下调用，对于缓冲区中的连续轨迹运动，该指令无效。

6.3.1.5 GT_SetStpAcc

该指令设置轨迹运动的急停加速度，单位为 **pulse/ms²**。在运动过程中，系统出现限位触发、伺服报警等异常情况时，GE-X00-SX 将按照该加速度实现减速停止，防止从高速立即停止下来造成对电机和系统的损害。一般在对运动控制器进行参数初始化时调用该指令。

用户在设置急停加速度时应注意使急停加速度大于进给加速度，并保证当系统运行在最高速度时急停，不会对系统和电机产生过度损伤。

6.2 轨迹运动的轨迹描述

6.2.1 指令列表

表 6-2 轨迹运动的轨迹描述指令列表

指 令	说 明
GT_LnXYG0	二维直线插补（含完整的加减速速度规划曲线）
GT_LnXYZG0	三维直线插补（含完整的加减速速度规划曲线）
GT_LnXYZAG0	四维直线插补（含完整的加减速速度规划曲线）
GT_LnXY	二维直线插补
GT_LnXYZ	三维直线插补
GT_LnXYZA	四维直线插补

指令	说明
GT_ArcXY	XY 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcXYP	XY 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcYZ	YZ 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcYZP	YZ 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcZX	ZX 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcZXP	ZX 平面圆弧插补（以终点位置和半径为输入参数）

6.2.2 例程

例程 6-2 单段轨迹运动的实现

本例在例程 6-1 的基础上调用轨迹描述指令，实现立即执行的单段轨迹运动。

```
void main()
{
    short rtn;
    double WorkVel, WorkAcc;

    AxisInitial(3,0);           //参考例程 3-2 ~ 3-4
    MotionInitial();           //参考例程 6-1

    WorkVel=100;                //进给速度为 3m/min
    WorkAcc=1.8;                //进给加速度为 0.9m/s2
    rtn=GT_SetSynAcc(WorkAcc);  //设置进给加速度
    error(rtn);
    rtn=GT_SetSynVel(WorkVel);  //设置进给速度
    error(rtn);
    rtn=GT_LnXY(10,10);         //两轴直线插补从当前点
    error(rtn);                 //运动到坐标（10，10）
}
```

6.2.3 重点说明

GE-X00-SX 运动控制器提供的轨迹描述指令都是在正交坐标系中描述的。

轨迹描述指令的长度单位为 pulse，要求用户在系统机械设计方面确保各控制轴的脉冲当量一致，以保证轨迹描述的正确和速度的一致性。

GT_LnXYG0/GT_LnXYZG0/GT_LnXYZAG0 和 GT_LnXY/GT_LnXYZ/GT_LnXYZA 直线插补指令的区别在于：

1. 用户若调用 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 指令，产生一个完整的加减速速度规划过程，即以设定加速度，从启动速度向目标

速度加速，到达减速点时，以加速度减速到启动速度后停止；

2. 用户若调用 GT_LnXY、GT_LnXYZ、GT_LnXYZA 指令，且运动控制器处于缓冲区连续轨迹运动方式时，运动控制器根据前后轨迹描述指令的进给速度、终点速度决定当前轨迹的速度规划曲线。

为正确描述圆弧插补运动，将圆弧插补指令描述的运动轨迹放在正交坐标系平面内，圆弧插补的旋转正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，逆时针方向为正（图 6-1）。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旋转的正方向。

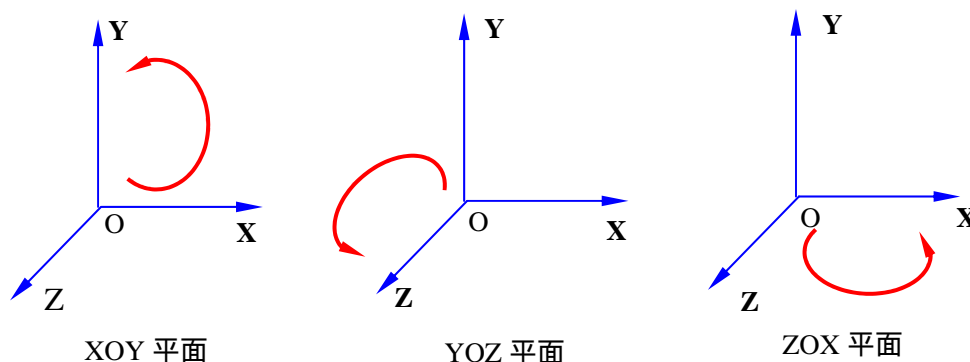


图 6-1 圆弧插补正方向

6.3 多段连续轨迹运动的基本实现

例程 6-2 给出的是在立即运动指令工作方式下，调用相关指令产生立即轨迹运动的范例。在这种工作方式下，执行了最后一行 `rtn=GT_LnXY(10,10)` 后，若运动控制器连接的外部设备（如机床）的工作状态正常（无限位触发、伺服报警等），将产生一个立即的轨迹运动，从当前点运动到（10pulse，10pulse）位置。

当运动控制器处于立即指令工作方式且轨迹运动未完成时（GT_GetCrdSts 返回状态的 BIT0 为 0），运动控制器不接受新的轨迹描述指令。

如果用户希望实现多段连续轨迹运动，必须启动缓冲区指令输入和缓冲区指令执行方式。

为了方便地实现多段连续轨迹运动，运动控制器提供一个大小为 8k 字的缓冲区。用户可先将部分轨迹描述或参数指令存放在该缓冲区中（以缓冲区满为限），然后发出执行指令。在运动控制器执行缓冲区内轨迹描述指令的同时，用户可继续向这个缓冲区内下载轨迹描述或参数指令。

运动控制器通过对缓冲区内多段轨迹描述指令的前瞻预处理，能够获得

良好的速度规划特性。

下面分两部分讲述利用缓冲区实现连续轨迹运动的方法：

- 将轨迹描述指令和参数指令输入缓冲区；
- 执行缓冲区连续轨迹运动。

6.3.1 将轨迹描述指令和参数指令输入缓冲区

6.3.1.1 指令列表

表 6-3 缓冲区输入管理指令列表

指 令	说 明
GT_StrtList	打开并清空缓冲区
GT_MvXY	定位运动起点位置（二维），设置进给速度和加速度
GT_MvXYZ	定位运动起点位置（三维），设置进给速度和加速度
GT_MvXYZA	定位运动起点位置（四维），设置进给速度和加速度
GT_EndList	关闭缓冲区
GT_AddList	再次打开缓冲区

可输入缓冲区的轨迹描述指令和参数指令，见表 6-4：

表 6-4 可输入缓冲区的指令列表

指 令	说 明
GT_SetSynVel	设置进给速度
GT_LnXYG0	二维直线插补（含完整的加减速速度规划曲线）
GT_LnXYZG0	三维直线插补（含完整的加减速速度规划曲线）
GT_LnXYZAG0	四维直线插补（含完整的加减速速度规划曲线）
GT_LnXY	二维直线插补
GT_LnXYZ	三维直线插补
GT_LnXYZA	四维直线插补
GT_ArcXY	XY 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcXYP	XY 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcYZ	YZ 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcYZP	YZ 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcZX	ZX 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcZXP	ZX 平面圆弧插补（以终点位置和半径为输入参数）
GT_SetDccVel	设置轨迹段的终点速度

6.3.1.2 重点说明

打开并清空缓冲区

GT_StrtList：打开并清空缓冲区。

启动缓冲区连续轨迹运动之前，必须先调用指令 **GT_StrtList** 进入缓冲区指令输入状态。该指令在缓冲区关闭且无缓冲区连续轨迹运动时有效。

定位运动起点位置，设置进给速度和加速度

运动控制器规定，打开并清空缓冲区（**GT_StrtList**）后，必须紧接定位运动起点位置指令（**GT_MvXY**、**GT_MvXYZ**、**GT_MvXYZA**）。

当用户调用指令 **GT_StrtMtn** 启动缓冲区连续轨迹运动后，运动控制器将从当前位置按照直线插补的方式，以定位指令中给定的加速度和速度，运动到指定的起点位置，并减速到 0，然后顺序执行缓冲区中的轨迹描述指令。

调用 **GT_MvXY**、**GT_MvXYZ** 或 **GT_MvXYZA** 设定的进给速度和加速度参数，作为此后输入缓冲区的轨迹描述指令的进给速度和加速度，用户可以调用指令 **GT_SetSynVel** 修改进给速度，但加速度将保持为定位指令中的设定值。

关闭缓冲区

GT_EndList：关闭缓冲区。该指令在缓冲区处于打开状态时有效。

再次打开缓冲区

GT_AddList：打开已经被关闭的缓冲区，允许用户向缓冲区增加新的缓冲区指令。

当增加的连续轨迹指令发送完毕后，可再次调用指令 **GT_EndList** 关闭缓冲区。该指令在缓冲区处于关闭状态时有效。用户可多次调用指令 **GT_AddList** 打开被关闭的缓冲区。

连续轨迹运动指令存入缓冲区的注意事项

用户能够在调用指令 **GT_StrtList** 之后的任意时刻，调用指令 **GT_EndList** 结束缓冲区指令输入状态；此后，可以调用指令 **GT_AddList** 继续缓冲区指令输入状态，又可再次调用指令 **GT_EndList** 结束缓冲区指令输入状态。

GT_AddList 和 **GT_EndList** 构成的组合，可以使用任意多次。当整个运动轨迹描述完成时，一定要调用 **GT_EndList** 指令通知运动控制器运动轨迹描述完成。

控制器对缓冲区中轨迹运动指令的处理机制

用户可以不断向缓冲区中添加轨迹描述或参数指令，直到缓冲区满。

缓冲区满时，运动控制器拒绝接收用户输入的轨迹描述指令和参数指令，并返回缓冲区满的信息。

启动缓冲区连续轨迹运动后，随着轨迹描述指令的执行完成，缓冲区会有新的空间，用户可以继续发送轨迹描述或参数指令。

6.3.2 启动、停止缓冲区连续轨迹运动

6.3.2.1 指令列表

表 6-5 启动、停止指令列表

指 令	说 明
GT_StrtMtn	启动缓冲区连续轨迹运动
GT_StpMtn	平滑停止轨迹运动
GT_EstpMtn	紧急停止轨迹运动
GT_RestoreMtn	暂停后恢复缓冲区连续轨迹运动

6.3.2.2 例程

例程 6-3 缓冲区连续轨迹运动的基本实现

例 6-3 实现三段连续轨迹运动，用户可参考实现多段的连续轨迹运动。

```

void main()
{
    short rtn;
    AxisInitial(3,0);                //参考例程 3-2 ~ 3-4
    rtn=GT_StrtList();
    error ( rtn );
    rtn=GT_MvXYZ(0, 0, 0, 100, 1.8);
    error ( rtn );

    //定位运动起点坐标
    // ( 0,0,0,0 ), 设置
    //进给速度为 3m/min
    //加速度为 0.9m/s2
    rtn=GT_LnXY(10,10);              //运动到坐标 ( 10 , 10 )
    error(rtn) ;
    rtn=GT_ArcXY(0,0,123);
    error(rtn);

    //以坐标 ( 0 , 0 ) 为圆心
    //以坐标 ( 10 , 10 ) 为起点
    //正向 123 度圆弧
    rtn=GT_LnXYZ(0,0,10);            //运动到坐标 ( 0,0,10 )
    error(rtn);
    rtn=GT_EndList();                //关闭缓冲区
    error(rtn);
    rtn=GT_StrtMtn();                //启动缓冲区
    //连续轨迹运动
    error(rtn) ;
}

```

6.3.2.3 重点说明

启动缓冲区连续轨迹运动

用户调用指令 **GT_StrtMtn** 成功后，缓冲区中的指令被顺序执行。

启动缓冲区连续轨迹运动前，用户应确保运动控制器的缓冲区内有轨迹描述指令。

启动缓冲区连续轨迹运动后，运动控制器将用户传送的轨迹描述或参数指令放入缓冲区的同时，执行缓冲区连续轨迹运动，直到用户调用指令 **GT_EndList** 关闭缓冲区指令输入方式。

中断缓冲区中指令的执行

调用指令 **GT_StpMtn** 后，运动控制器按照设定的加速度平滑停止轨迹运动；调用指令 **GT_EStpMtn** 后，运动控制器按照急停加速度停止轨迹运动。

调用指令 **GT_StpMtn** (**GT_EStpMtn**) 成功后，缓冲区被立即关闭，相当于调用了指令 **GT_EndList**。当缓冲区连续轨迹运动规划停止后，运动控制器记录当前位置信息和当前段段号（又称断点信息），以确保恢复缓冲区轨迹运动时，能够准确地回到断点位置继续缓冲区连续轨迹运动。

当轨迹运动停止且缓冲区没有被再次打开，用户调用的轨迹描述指令将被运动控制器立即执行。

GT_StpMtn 和 **GT_EStpMtn** 指令也可中断正在进行的立即轨迹运动，但不保留立即轨迹运动的断点信息，即立即轨迹运动被中断后无法恢复，必须重新调用轨迹描述指令启动立即轨迹运动。

恢复缓冲区中指令的执行

当调用 **GT_StpMtn** (**GT_EStpMtn**) 使缓冲区连续轨迹运动停止（调用 **GT_GetCrdSts** 获取的状态信息中的 **BIT0** 为 1），用户可以调用 **GT_RestoreMtn** 指令，再次启动缓冲区连续轨迹运动。

这时运动控制器从当前坐标位置以直线插补方式运动到断点位置处，并减速到启动速度，然后继续缓冲区连续轨迹运动。

调用 **GT_RestoreMtn** 指令前，用户必须确保曾经调用 **GT_StrtMtn** 启动了缓冲区的运行，否则可能引起错误的运动。

暂停后恢复缓冲区连续轨迹运动必须调用 **GT_RestoreMtn** 指令，若调用 **GT_StrtMtn** 来恢复运动，可能会引起错误的运动。

6.4 小线段连续轨迹运动的速度规划策略

连续轨迹运动的进给速度采用梯形曲线加减速策略，定位指令中的加速度参数设置了连续轨迹运动的进给加速度，指令 **GT_SetSynVel** 可改变进给速度。如果设定的进给速度超过了用户设定的最大速度，取最大速度为进给速度。

为解决速度和精度的矛盾，GE-X00-SX 运动控制器提供基于前瞻预处理的速​​度规划策略。用户通过设定本身机床的工艺特征参数（脉冲当量、进给速度、最大加速度、允许拐弯时间等），结合 GE-X00-SX 运动控制器的前瞻预处理功能模块，可实现小线段连续轨迹加工。

运动控制器将实现速度规划预处理功能的指令称为**前瞻预处理指令**。

图 6-2 所示为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同样精度下，对速度的显著提升：

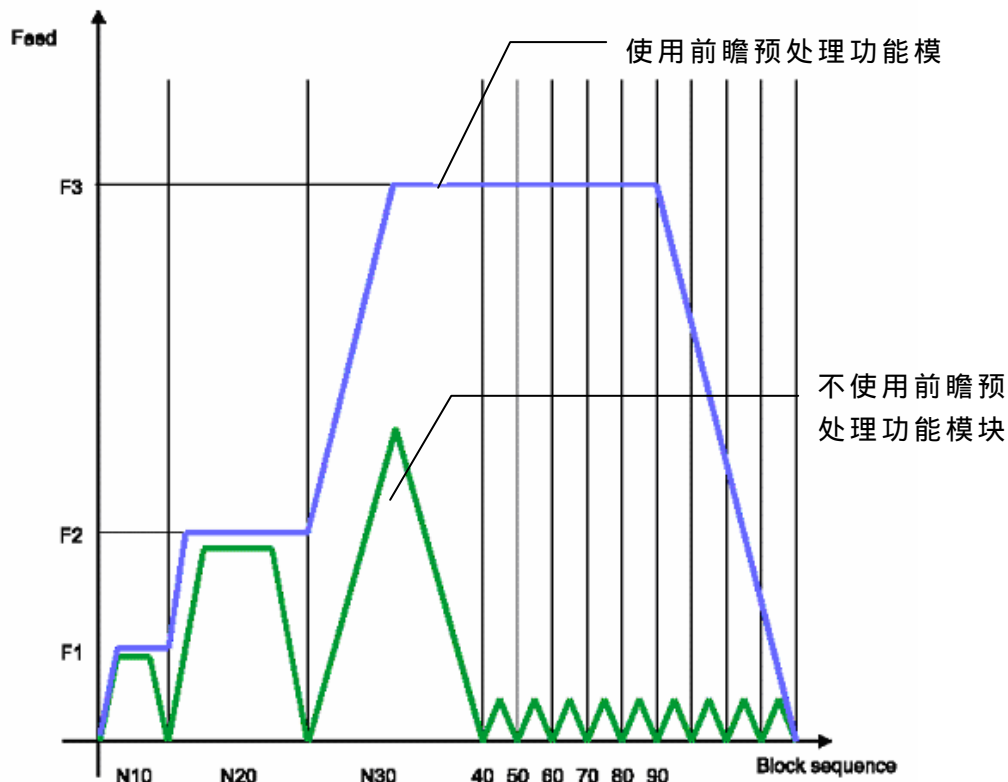


图 6-2 使用和不使用前瞻预处理功能模块的速度曲线对

6.4.1 前瞻预处理指令列表

表 6-6 前瞻预处理指令列表

指令	说明
GT_InitLookAhead	初始化前瞻预处理参数
GT_AddLookData	将轨迹段信息添加到预处理缓冲区
GT_CalVel	计算轨迹段终点速度
GT_SetDccVel	设置轨迹段终点速度

6.4.2 重点说明

6.4.2.1 初始化前瞻预处理参数

GT_InitLookAhead：设置前瞻预处理功能模块的初始化参数

表 6-7 前瞻预处理初始化参数列表

参数名称	参 数 意 义	说 明
T	拐弯时间（单位：s）	T 越大，计算出来的终点速度越大，但会降低加工精度；反之，提高了加工的精度，但计算出的终点速度偏低
acc_max	系统能承受的最大加速度（单位： mm/s^2 ）	根据不同的机械系统和电机驱动器取值不同，经验范围：100~10000 mm/s^2
acc_run	系统加速度（单位： mm/s^2 ）	设定加速度一定小于最大加速度
vel_run	系统进给速度（单位： mm/s ）	

参数 T 和 acc_max 会影响拐点处的位置误差：

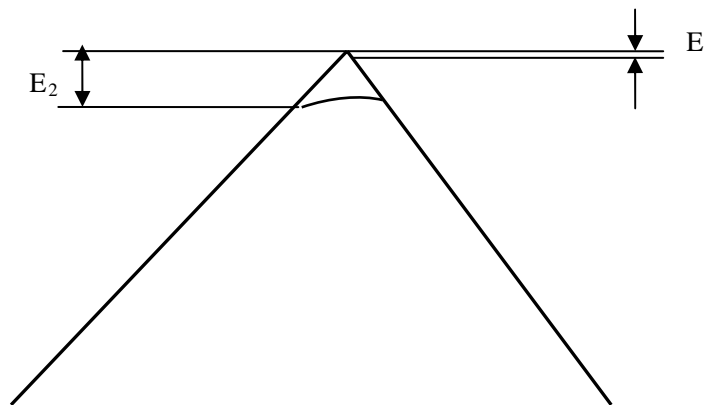


图 6-3 不同 T 和 Acc_max 参数的位置误差比较

观察图 6-3，当 T 或 acc_max 参数设置较大时，降速较少，但位置误差 E_2 较大；若 T 和 acc_max 参数设置合理，可以保证位置误差 E 进入系统能接受的精度范围。

6.4.2.2 传送轨迹特征数据进入预处理缓冲区

GT_AddLookData：将轨迹特征数据加入预处理缓冲区

表 6-8 传递轨迹特征数据列表

参数名称	参 数 意 义	说 明	
Code	曲线类型	0	G00 直线
		1	G01 直线

		2	G02 顺圆弧
		3	G03 逆圆弧
PlaneGroup	平面代码	17	XY 平面
		18	ZX 平面
		19	YZ 平面
R	圆弧半径	若当前段不是圆弧，设为 0	
x , y , z	坐标终点位置	单位：mm	
F	当前段进给速度	允许每段有不同的进给速度（单位：mm/s）	
cx , cy	圆心坐标	若当前段不是圆弧，设为 0	

6.4.2.3 计算终点速度

GT_Calvel：计算预处理当前轨迹段的终点速度；调用该指令的返回值若为 0 表示预处理正在进行，若为 1 表示预处理结束。

6.5 连续轨迹运动中的速度倍率

6.5.1 指令列表

表 6-8 速度倍率指令列表

指 令	说 明
GT_OverrideG0	修改 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 指令以及定位指令的速度倍率
GT_Override	修改除 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 指令以及定位指令以外的轨迹描述指令的速度倍率

6.5.2 重点说明

用户可调用 GT_Override、GT_OverrideG0 指令修改当前的速度倍率，达到在线修改当前轨迹运动的进给速度的目的，同时也改变了以后所有轨迹运动的进给速度。一旦成功修改速度倍率，该倍率值将一直有效，直到再次调用 GT_Override 或 GT_OverrideG0 指令修改速度倍率。

运动控制器默认的速度倍率均为 100%。

一般来说速度倍率的可调整范围在 0%~200% 之间，如果用户调整速度倍率使进给速度大于系统初始化设置的最大速度，运动控制器取最大速度为进给速度。

6.6 连续轨迹运动的机床坐标系和加工坐标系

6.6.1 指令说明

表 6-9 坐标偏移指令

指 令	说 明
GT_MapCnt	设置指定控制轴坐标偏移量

6.6.2 重点说明

机床坐标系是以机床零点为原点的坐标系 ;加工坐标系是相对于加工原点坐标系。GE-X00-SX 运动控制器上电时默认两个坐标系重合。当用户需要建立加工坐标系时，可调用 GT_MapCnt 指令设置指定控制轴的坐标偏移量。

6.7 连续轨迹运动的运动信息反馈

6.7.1 指令列表

表 6-10 连续轨迹运动信息反馈指令列表

指 令	说 明
GT_GetPrfPnt	读取所有控制轴的规划位置
GT_GetAtlPos	读取指定控制轴的实际位置
GT_GetSegPnt	读取当前轨迹段的所有控制轴终点目标位置
GT_GetPrfVel	读取规划速度
GT_GetCrdSts	读取连续轨迹运动状态
GT_GetSts	读取指定控制轴限位、报警、原点捕获信号状态
GT_GetMtnNm	读取当前轨迹段编号
GT_GetBrkPnt	读取所有控制轴的断点位置

6.7.2 重点说明

6.7.2.1 读取规划位置和实际位置

GT_GetPrfPnt：调用该指令读取连续轨迹运动中所有控制轴的规划位置。

GT_GetAtlPos：调用该指令读取指定控制轴的实际位置。

GE-X00-SX 运动控制器处于脉冲输出方式时，系统处于开环控制，默认关闭编码器。

用户可以调用 **GT_EncOn** 指令打开指定控制轴的编码器，调用

GT_GetAtlPos 读取编码器的实际位置值。

若关闭编码器，调用 GT_GetAtlPos 读取的位置信息为规划位置值。

6.7.2.2 轨迹段号

GT_GetMtnNm：该指令返回当前轨迹段编号。

轨迹段编号规则

1. 启动缓冲区中指令执行后，轨迹段编号随输入缓冲区段数递增，GT_MvXY、GT_MvXYZ 或 GT_MvXYZA 定位指令不计入段号（段号为 0）；
2. 段号由 1 开始累加，当计数到最大值 32768 时段号溢出并从 1 开始重新累计；
3. 当缓冲区内的轨迹段已经执行完毕，但用户又没有执行 GT_EndList 指令关闭缓冲区，连续轨迹运动状态字的 bit0 和 bit1 为 0，轨迹段号保持在当前段；
4. 用户若调用 GT_StpMtn 或 GT_EStpMtn 指令中断缓冲区连续轨迹运动，连续轨迹运动状态字的 Bit1 和 Bit0 为 1，段号保持当前值，直到恢复缓冲区连续轨迹运动；
5. 当前段运动结束时，若没有连续轨迹运动段，段号保持当前值；
6. 调用 GT_StrtList 指令，段号清零。

6.7.2.3 连续轨迹运动状态和其它状态

GT_GetCrdSts：调用该指令读取连续轨迹运动状态，其标志位定义见表 6-11。

表 6-11 连续轨迹运动状态定义

位	定义
0	指示轨迹运动状态：0 表示有轨迹运动，1 表示无轨迹运动（默认）。
1	缓冲区是否打开：0 表示打开，1 表示关闭（默认）。
2	轨迹运动预处理是否正常：0 表示正常（默认），1 表示预处理时间不够，运动控制器不能正常工作。
3	保留。
4	指示一段轨迹运动状态，0 表示正在进行轨迹运动，1 表示一段轨迹运动完成（默认）。
5~6	保留。
7	指示指令输入状态，0 表示缓冲区指令输入或执行状态，1 表示立即指令输入状态（默认）。
8	保留。
9	轨迹运动中相关控制轴是否出现异常（如限位开关触发），0 表示相关控制轴正常（默认），1 表示相关控制轴出现异常，

位	定 义
	并自动停止连续轨迹运动。
10	指示脉冲输出是否出现异常，0 表示正常（默认），1 表示脉冲输出异常。
11~12	保留。
13	缓冲区是否空，0 表示不空，1 表示空（默认）。
14~15	保留。

GT_GetSts :调用该指令获取指定控制轴的外部信号状态。正在进行连续轨迹运动的控制轴出现限位、伺服报警等异常状态时，GE-X00-SX 运动控制器自动停止轨迹运动，并将控制轴状态相应标志位置位，便于用户检测后进行相应处理。

第七章 Home/Index 高速捕获

7.1 指令列表

表 7-1 Home/Index 捕获指令列表

指令	说明
GT_CaptHome	将指定控制轴设置为 Home 捕获
GT_CaptIndex	将指定控制轴设置为 Index 捕获
GT_GetCapt	读取指定控制轴的捕获位置

7.2 Home 回原点

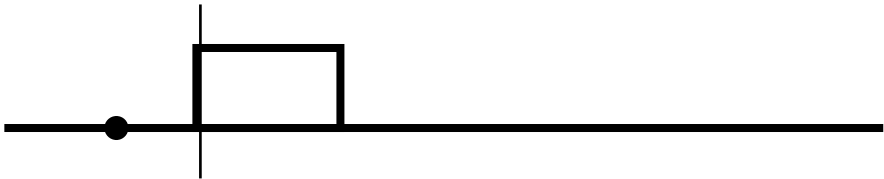
(1) 工作台向原点 (Home) 开关方向运动，启动 Home 捕获；



(2) 当 Home 信号产生时，平滑停止工作台；

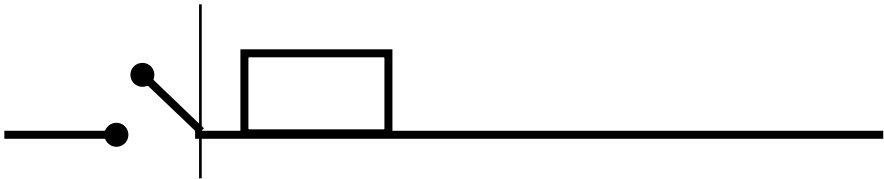


(3) 调用指令 GT_GetCapt 读取 Home 信号触发时工作台的实际位置，然后反向



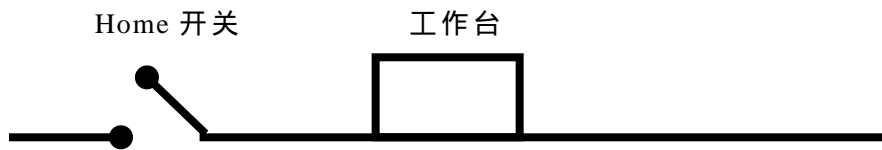
运动回到该位置；

(4) 继续向前运动一段指定距离，离开 Home 开关，等工作台停稳以后调用指令 GT_ZeroPos 将工作台位置清零，建立机床坐标系。

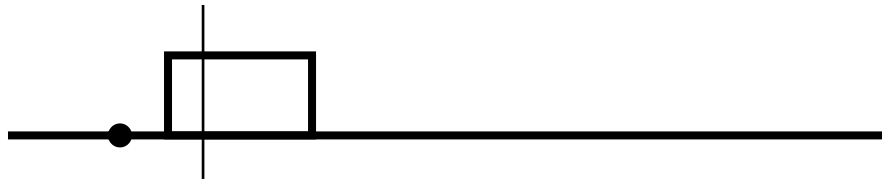


7.3 Home + Index 回原点

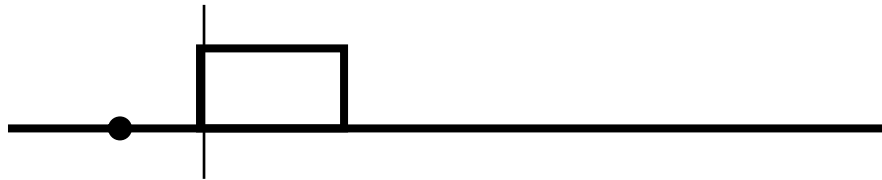
(1) 工作台向原点 (Home) 开关方向运动，启动 Home 捕获；



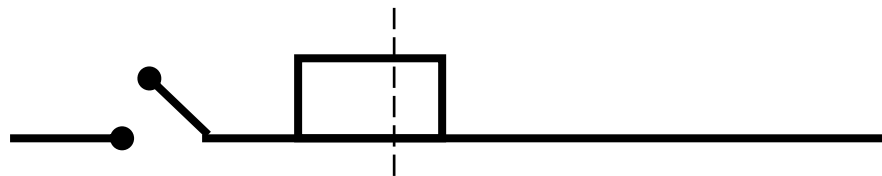
(2) 当 Home 信号产生时，平滑停止工作台；



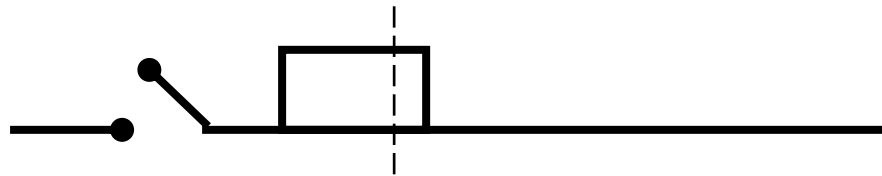
(3) 调用指令 GT_GetCapt 读取 Home 信号触发时工作台的实际位置，然后反向运动回到该位置；



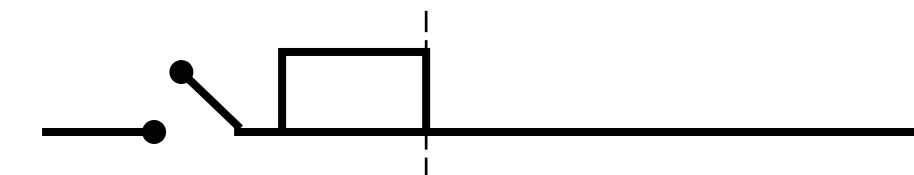
(4) 启动 Index 捕获，继续运动一圈多一点，当 Index 信号产生时，平滑停止工作台；



(5) 调用指令 GT_GetCapt 读取 Index 信号触发时工作台的实际位置，然后反向运动到该位置；



(6) 反向运动一段指定距离，消除丝杠的反向间隙，等工作台停稳以后调用指令 GT_ZeroPos 将工作台位置清零，建立机床坐标系。



7.4 例程

例程 7-1 Home 回原点例程 (以 GE-300-SG 运动控制器为例)

```
void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !");          break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");                  break;
        case 2:
            printf("\a\nRadius or chord is 0 !");           break;
        case 3:
            printf("\a\nLength is 0 or overflow !");        break;
        case 4:
            printf("\a\nVelocity or acceleration is less then 0 !");
                                                                break;
        case 5:
            printf("\a\nChord is greater than diameter !"); break;
        case 7:
            printf("\a\nParameter error !");                break;
        default:
            printf("\a\nError Code = %d",error);            break;
    }
}

short Home(unsigned short axis,long pos,long offset,double
vel_high,double vel_low)
{
    double prf_pos[4];
    unsigned short status,crd_status;

    GT_ClrSts(axis);                //清回原点轴状态
    GT_CaptHome(axis);              //将回原点轴设置为
                                    //原点捕获方式
    GT_SetSynVel(vel_high);         //设置回原点的速度
    GT_SetSynAcc(0.01);             //设置回原点的加速度
    GT_GetPrfPnt(prf_pos);          //读取各轴的规划位置
    prf_pos[axis-1] = pos;          //设置回原点轴的目标位置
}
```

```

GT_LnXYZ(prf_pos[0],prf_pos[1],prf_pos[2]);
do
{
    GT_GetCrdSts(&crd_status);           //读取坐标系状态
    if(crd_status&1)                       //如果运动已经完成
    {                                     //仍然没有触发 Home 信号
        return 1;                         //返回错误代码 1
    }
    GT_GetSts(axis,&status);               //读取控制轴状态
}while(!(status&0x8));                    //等待 Home 信号触发
GT_StpMtn();                             //平滑停止
do
{
    GT_GetCrdSts(&crd_status);           //读取坐标系状态
}while(!(crd_status&1));                  //等待运动完成
GT_GetCapt(axis,&pos);                   //读取捕获位置
GT_SetSynVel(vel_low);                   //低速运动到原点
prf_pos[axis-1]= pos;
rtn=GT_LnXYZ(prf_pos[0],prf_pos[1],prf_pos[2]);
do
{
    GT_GetCrdSts(&crd_status);           //读取坐标系状态
}while(!(crd_status&1));                  //等待运动完成
prf_pos[axis-1]+= offset;                 //运动一小段距离离开 Home 开关
GT_LnXYZ(prf_pos[0],prf_pos[1],prf_pos[2]);
do
{
    GT_GetCrdSts(&crd_status);           //读取坐标系状态
}while(!(crd_status&1));                  //等待运动完成

delay(1000);                             //插入适当延时 ,等待工作台停稳

GT_ZeroPos(axis);                         //工作台位置清零建立机床坐标系
return 0;
}

void main()
{
    GT_HookCommand(CommandHandle);
    GT_Open();                             //打开运动控制器
    GT_Reset();                             //复位运动控制器
    AxisInitial(1,0);                      //引用例程 3-2 或 3-3
    if(0!=Home(1,-1000000,1000,20,2))     //调用回零函数

```

```

    {
        printf("\n Home Error !");
    }
}

```

例程 7-2 Home + Index 回原点例程（以 GE-800-PV 运动控制器为例）

```

void CommandHandle(char *command,short error)
{
    switch(error)
    {
        case -1:
            printf("\a\nCommunication Error !");          break;
        case 0:
            break;
        case 1:
            printf("\a\nCommand Error !");                break;
        case 7:
            printf("\a\nParameter error !");              break;
        default:
            printf("\a\nError Code = %d",error);          break;
    }
}

short Home(unsigned short axis,long pos,long offset,double
vel_high,double vel_low)
{
    unsigned long status;

    GT_ClrSts(axis);                                     //清回原点轴状态
    GT_CaptHome(axis);                                   //将回原点轴设置为原点捕获方式
    GT_SetPos(axis, pos);
    GT_SetVel(axis,vel_high);                            //高速查找原点
    GT_SetAcc(axis,0.01);
    GT_Update(axis);

    do
    {
        GT_GetSts(axis,&status);                         //读取回原点轴状态
        if(!(status&0x400))
        {
            return 1;                                     //回原点轴已经停止
                                                    //Home 信号仍然没有触发
        }
    }while(!(status&0x8))                                //等待 Home 捕获
}

```

```
GT_SmthStp(axis);                                //平滑停止回原点轴

do
{
    GT_GetSts(axis,&status);
}while(status&0x400);                            //等待运动完成

GT_GetCapt(axis,&pos);                          //读取 Home 捕获位置
GT_SetPos(axis,pos);                            //低速运动到原点捕获位置
GT_SetVel(axis,vel_low);
GT_Update(axis);

do
{
    GT_GetSts(axis,&status);
}while(status&0x400);                            //等待运动完成

GT_ClrSts(axis);                                //清除 Home 捕获触发标志
GT_CaptIndex(axis);                            //将回原点轴设置为
                                                //Index 捕获
GT_SetPos(axis,pos+11000);                      //继续运动一圈多一点
GT_Update(axis);

do
{
    GT_GetSts(axis,&status);
    if(!(status&x400))
    {
        return 2;                                //回原点轴已经停止
                                                //Index 信号仍然没有触发
    }
}while(!(status&0x8));                          //等待 Index 信号触发

GT_SmthStp(axis);                                //平滑停止回原点轴

do
{
    GT_GetSts(axis,&status);
}while(status&0x400);                            //等待运动完成

GT_GetCapt(axis,&pos);                          //读取 Index 捕获位置
GT_SetPos(axis,pos);                            //运动到 Index 捕获位置
GT_SetVel(axis,vel_low);
```

```
GT_Update(axis);

do
{
    GT_GetSts(axis,&status);
}while(status&0x400);           //等待运动完成

GT_SetPos(axis,pos+offset);     //运动到 Index 捕获位置
GT_Update(axis);

do
{
    GT_GetSts(axis,&status);
}while(status&0x400);           //等待运动完成

delay(1000);                     //插入适当延时 ,等待工作台停稳

GT_ZeroPos(axis);               //工作台位置清零建立机床坐标系
return 0;
}

void main()
{
    GT_HookCommand(CommandHandle);
    GT_Open();                   //打开运动控制器
    GT_Reset();                  //复位运动控制器
    AxisInitial(1,0);            //引用例程 3-2 或 3-3
    if(0!=Home(1,-1000000,1000,20,2)) //调用回零函数
    {
        printf("\n Home Error !");
    }
}
```

7.5 重点说明

GE 系列运动控制器为每个控制轴提供一个高速位置捕获寄存器，用来保存触发信号产生时控制轴的实际位置。

Home 信号来自原点开关，调用指令 GT_CaptHome 将指定控制轴的位置捕获方式设置为 Home 信号触发。如果指定控制轴已经处于 Home 捕获模式或者 Index 捕获模式，且捕获信号尚未触发，不能再次调用该指令。如果捕获信号已经触发，必须调用指令 GT_ClrSts 清除指定控制轴“捕获触发标志位”bit3 以后，才能再次调用该指令。

Index 信号来自编码器 C 相（编码器每转一圈产生一个 C 相脉冲）或者光栅尺（每隔固定距离产生一个脉冲），调用指令 GT_CaptIndex 将指定控制轴的位置捕获方式设置为 Index 信号触发。如果指定控制轴已经处于 Home 捕获模式或者 Index 捕获模式，且捕获信号尚未触发，不能再次调用该指令。如果捕获信号已经触发，必须调用指令 GT_ClrSts 清除指定控制轴“捕获触发标志位”bit3 以后，才能再次调用该指令。

Home 信号和 Index 信号都是边沿触发模式。可调用指令 GT_HomeSns 设置 Home 信号是上升沿触发还是下降沿触发，运动控制器默认为下降沿触发。当 Home 信号或者 Index 信号产生时，运动控制器将自动锁存 Home/Index 信号产生时触发轴的实际位置（来自编码器或光栅尺），并将触发轴状态寄存器的 bit3 “Home/Index 捕获触发标志位”置 1，关闭 Home/Index 捕获（触发轴状态寄存器的 bit14/bit15“Home/Index 捕获状态位”清 0）。调用指令 GT_GetCapt 可以读取指定控制轴的捕获位置值。

注意：使用 Home+Index 方式回原点时，应将 Home 开关安装在相邻的 2 个 Index 信号的中间。当 Home 开关和 Index 信号重叠时可能造成 Index 信号捕获错误。

第八章 安全机制

8.1 设置跟随误差超限自动停止

8.1.1 指令列表

表 8-1 设置跟随误差超限自动停止指令列表

指令	说明
GT_SetPosErr	设置指定控制轴的跟随误差极限
GT_GetPosErr	读取指定控制轴的跟随误差极限
GT_AuStpOn	使能指定控制轴跟随误差超限自动停止模式 (GE-X00-PX)
GT_AuStpOff	关闭指定控制轴跟随误差超限自动停止模式 (GE-X00-PX)(默认)

8.1.2 重点说明

对于伺服控制系统而言，在某些异常情况下，电机的实际位置可能与规划位置差距很大。这时通常存在一些危险情况，例如电机故障、编码器 A、B 相信号接反或断线、机械摩擦太大或者机械故障造成电机堵转等。为了及时检测这些情况，增强系统的安全性并延长设备使用寿命，运动控制器提供跟随误差超限自动停止的安全保护机制。

运动控制器在每个控制采样周期内都检查控制轴的实际位置与规划位置的误差是否超越所设定的跟随误差极限。

对于 GE-X00-PX 运动控制器，如果位置误差超越所设定的跟随误差极限，运动控制器将该轴状态寄存器的 bit4 标志位置 1；如果已使能跟随误差超限自动停止功能，运动控制器自动紧急停止该轴的运动。

对于 GE-X00-SX 运动控制器，如果正在进行连续轨迹运动的控制轴位置误差超越所设定的跟随误差极限，运动控制器紧急停止连续轨迹运动，并将该轴状态寄存器的 bit4 标志位置 1。

8.2 设置输出电压饱和极限

8.2.1 指令列表

表 8-2 设置输出电压饱和极限

指令	说明
GT_SetMtrLmt	设置指定控制轴的输出电压饱和极限

8.2.2 重点说明

运动控制器各控制轴默认电压输出范围是 $\pm 10V$ 。在某些场合下，需要对控制轴输出电压进行限制，可以调用指令 `GT_SetMtrLmt` 设定控制轴输出电压饱和值。

8.3 限位状态处理

运动控制器能够通过限位开关限制各控制轴的运动范围。装有限位开关的控制轴的“安全运动范围”如图所示：

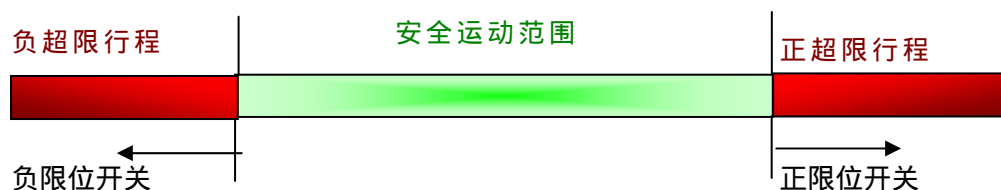


图 8-1 控制轴运动行程范围定义

工作台碰到限位开关时，运动控制器紧急停止工作台的运动，此时会产生较大的冲击。建议用户不要使用限位开关回原点，而应当使用原点开关完成回原点的操作，详细步骤请参阅第七章。

限位开关触发以后，运动控制器禁止向触发限位方向上继续运动，同时将控制轴状态寄存器的相应状态位置 1。离开限位开关回到安全运动范围以后，需要调用指令 `GT_ClrSts` 将控制轴状态寄存器的限位触发状态位清 0，才能使控制轴从超限状态回到正常状态，否则控制轴不能正常运动。

8.4 控制轴驱动报警处理

运动控制器提供专用的驱动报警信号输入接口，若已经打开监视驱动报警信号功能，且检测到驱动器报警信号，运动控制器将控制轴状态寄存器的相应标志位置 1，紧急停止发生报警轴的运动（GE-X00-PX）或连续轨迹运动（GE-X00-SX），并关闭驱动器的伺服使能信号。

驱动器报警信号产生以后，应当执行以下操作：

1. 确定引起驱动器报警的原因，并加以改正；
2. 复位驱动器；
3. 重新回机床原点。

第九章 数字 I/O

9.1 通用数字 I/O

9.1.1 指令列表

表 9-1 通用数字 I/O 指令列表

指 令	说 明
GT_ExInpt	读取 16 路通用数字 I/O 输入的电平状态
GT_ExOpt	设置 16 路通用数字 I/O 输出的电平状态

9.1.2 重点说明

运动控制器提供带光电隔离的 16 路通用数字量输入和 16 路通用数字量输出接口。

调用指令 GT_ExInpt 可以读取 16 路通用数字量输入。指令参数的状态位与端子板上外部输入接口的对应关系如表 9-2 所示：

表 9-2 通用 I/O 输入接口定义

状态位	15	14	13	12	11	10	9	8
端子板	EXI15	EXI14	EXI13	EXI12	EXI11	EXI10	EXI9	EXI8
状态位	7	6	5	4	3	2	1	0
端子板	EXI7	EXI6	EXI5	EXI4	EXI3	EXI2	EXI1	EXI0

调用指令 GT_ExOpt 可以设置 16 路通用数字量输出。指令参数的状态位与端子板上外部输出接口的对应关系如表 9-3 所示：

表 9-3 通用 I/O 输出接口定义

状态位	15	14	13	12	11	10	9	8
端子板	EXO15	EXO14	EXO13	EXO12	EXO11	EXO10	EXO9	EXO8
状态位	7	6	5	4	3	2	1	0
端子板	EXO7	EXO6	EXI5	EXO4	EXO3	EXO2	EXO1	EXO0

9.2 专用数字 I/O

9.2.1 指令列表

表 9-4 专用数字 I/O 指令列表

指 令	说 明
GT_GetLmtSwt	读取限位开关的电平状态
GT_GetHomeSwt	读取原点开关的电平状态

9.2.2 重点说明

调用指令 GT_GetLmtSwt 能够读取限位开关的电平状态，指令参数的状态位与端子板上各控制轴的限位信号输入接口的对应关系如表 9-5 所示：

表 9-5 指令参数状态位与端子板上各控制轴限位信号输入接口的对应关系

状态位	15	14	13	12	11	10	9	8
端子板	Limit7-	Limt7+	Limit6-	Limt6+	Limit5-	Limt5+	Limit4-	Limt4+
状态位	7	6	5	4	3	2	1	0
端子板	Limit3-	Limt3+	Limit2-	Limt2+	Limit1-	Limt1+	Limit0-	Limit0+

调用指令 GT_GetHomeSwt 能够读取原点开关的电平状态，指令参数的状态位与端子板上各控制轴的原点信号输入接口的对应关系如表 9-6 所示：

表 9-6 指令参数状态位与端子板上各控制轴原点信号输入接口的对应关系

状态位	7	6	5	4	3	2	1	0
端子板	Home7	Home6	Home5	Home4	Home3	Home2	Home1	Home0

第十章 运动控制器可选功能

10.1 运动控制器数据监测

通过采集和分析运动控制器各控制轴的相关数据，调整各控制轴 PID 控制参数和运动参数，可以充分发挥伺服系统的性能优势，达到理想的控制效果。

GE 系列运动控制器能够实时监测各控制轴的规划位置、实际位置、规划速度、实际速度、控制电压。用户可以指定需要监测的控制轴参数以及监测频率。

10.1.1 指令列表

表 10-1 数据监测指令列表

指 令	说 明
GT_SetWatch	指定需要监测的控制轴参数。
GT_GetWatch	读取监测数据。
GT_StartWatch	指定监测频率，启动数据监测。
GT_StopWatch	停止数据监测。

10.1.2 例程

以 GE-800-PV 运动控制器为例，监测第 2 轴每个控制采样周期的规划位置 and 实际位置，并将所监测的数据写入文件。

例程 10-1 使用 Visual C++ 进行编程

```
#include "gep.h" //引用指令函数库
#define BUFFER_SIZE 400 //定义监测缓冲区大小

void __stdcall CommandHandle(char * command,short error)
{ //错误处理函数
    switch(error)
    {
        case -1:
            printf("\n Communication Error : %s = %d",command,error);
            exit(3);
        case 0:
            return;
        case 1:
            printf("\n Command Error : %s = %d",command,error);
            exit(2);
        case 7:
```

```
        printf("\n Parameter Error : %s = %d",command,error);
                                                exit(1);
    default:
        printf("\n Error Code = %d ", error);          exit(1);
    }
}

void main()
{
    FILE *fp;                                     //创建文件，用于保存数据
    if((fp=fopen("output.txt","wt"))==NULL)
    {
        printf("Can't Open output.txt\n");
        exit(0);
    }
    GT_HookCommand(CommandHandle); //挂接错误处理程序
    GT_Open(0x300);                 //打开运动控制器
    GT_Reset();                     //复位运动控制器
    unsigned short config[AXIS_NUM] = {0,3,0,0,0,0,0,0};
                                                //AXIS_NUM 定义于函数库指示控
                                                //制轴数，参见表 10-2
    GT_SetWatch(config,BUFFER_SIZE); //监测第 2 轴的规划位置和实际位置
    long PrfPos[BUFFER_SIZE],AtlPos[BUFFER_SIZE];
                                                //创建数据缓冲区
    TWatchBuffer buffer;                 //TWatchBuffer 定义于函数库
    buffer.Axis[1].PrfPos = PrfPos;      //挂接规划位置数据缓冲区
    buffer.Axis[1].AtlPos = AtlPos;      //挂接实际位置数据缓冲区
    GT_SetKp(2,10);                     //设置第 2 轴的比例增益
    GT_Update(2);                         //刷新第 2 轴参数
    GT_AxisOn(2);                         //第 2 轴伺服使能

    printf("\nPress any key to continue...");//按任意键继续
    getch();

    GT_SetPos(2,100000);                 //设置第 2 轴的目标位置
    GT_SetVel(2,10);                     //设置第 2 轴的目标速度
    GT_SetAcc(2,1);                       //设置第 2 轴的加速度
    GT_StartWatch(0);                     //启动数据监测每个控制周期
                                                //监测 1 次
    GT_Update(2);                         //刷新第 2 轴参数

    unsigned short available;
```

```

unsigned long status;
do
{
    GT_GetWatch(&buffer,&available); //检测数据缓冲区是否放满
    if(available)                    //当数据缓冲区放满以后
    {                                //存储到数据文件当中
        for(short i=0;i<BUFFER_SIZE;++i)
        {
            fprintf(fp,"%d\t%d\n",PrfPos[i],AtlPos[i]);
        }
    }
    GT_GetSts(2,&status);            //检测第 2 轴的状态
}while(status&0x400);              //等待第 2 轴运动停止

do                                  //第 2 轴运动停止以后
{                                  //再次监测 1 批数据
    GT_GetWatch(&buffer,&available); //检测数据缓冲区是否放满
    if(available)                  //当数据缓冲区放满以后
    {                              //存储到数据文件当中
        for(short i=0;i<BUFFER_SIZE;++i)
        {
            fprintf(fp,"%d\t%d\n",PrfPos[i],AtlPos[i]);
        }
    }
}while(!available);               //等待数据缓冲区放满

GT_StopWatch();                   //停止数据监测
GT_Close();                       //关闭运动控制器
fclose(fp);                      //关闭数据文件
}

```

例程 10-2 使用 Delphi 进行编程

```

const
    BUFFER_SIZE = 400;            //定义缓冲区的数据容量
procedure CommandHandle(Command:PChar;Error:short);stdcall;
begin
    //错误处理函数
    case error of
        -1:  ShowMessageFmt('Communication Error %s =
            %d',[command,error]);
        0:   exit;
    end;
end;

```

```
1:  ShowMessageFmt('Command Error %s =
%d',[command,error]);
7:  ShowMessageFmt('Param Error %s = %d',[command,error]);
end;
end;

procedure TForm1.Button1Click(Sender: TObject); //点击按钮 Button1 时
var
    config : TConfig;           //TConfig 定义于函数库
    available : word;
    buffer : TWatchBuffer;      //TWatchBuffer 定义于
                                //函数库
    status : LongWord;
    i : LongInt;
    list : TStringList;
begin
    list := TStringList.Create; //list 用于保存数据
    GT_HookCommand(CommandHandle); //挂接错误处理程序
    GT_Open($300);              //打开运动控制器
    GT_Reset;                   //复位运动控制器
    config[0] := $0;
    config[1] := $3;            //监测第 2 轴的
    config[2] := $0;            //规划位置 and 实际位置
    config[3] := $0;
    config[4] := $0;
    config[5] := $0;
    config[6] := $0;
    config[7] := $0;
    GT_SetWatch(config,BUFFER_SIZE); //设置所要监测数据
                                    //和缓冲区大小
    SetLength(buffer.Axis[1].PrfPos,BUFFER_SIZE);
                                    //创建规划位置数据缓冲区
    SetLength(buffer.Axis[1].AtlPos,BUFFER_SIZE);
                                    //创建实际位置数据缓冲区
    GT_SetKp(2,10);              //设置第 2 轴的比例增益
    GT_Update(2);                //刷新第 2 轴参数
    GT_AxisOn(2);                //第 2 轴伺服使能
    ShowMessage('Click button to continue...'); //点击按钮继续

    GT_SetPos(2,100000);         //设置第 2 轴的目标位置
    GT_SetVel(2,10);             //设置第 2 轴的目标速度
    GT_SetAcc(2,1);              //设置第 2 轴的加速度
    GT_StartWatch(0);            //启动数据监测
                                //每个控制周期监测 1 次
```



```

GT_Update(2);                //刷新第 2 轴参数
repeat
    GT_GetWatch(buffer,available); //检测数据缓冲区是否放满
    if 1 = available then        //当数据缓冲区放满以后
    begin                        //存储到 list 当中
        for i:=0 to BUFFER_SIZE - 1 do
        begin
            list.Add(Format('%d  %d',[buffer.Axis[1].PrfPos[i],
                                buffer.Axis[1].AtlPos[i]]));
        end;
    end;
    GT_GetSts(2,status);        //检测第 2 轴的状态
until 0 = (status and $400);    //等待第 2 轴运动停止

repeat                          //第 2 轴停止以后
    GT_GetWatch(buffer,available); //再次监测 1 批数据
    if 1 = available then        //当数据缓冲区放满以后
    begin                        //存储到 list 当中
        for i:=0 to BUFFER_SIZE - 1 do
        begin
            list.Add(Format('%d  %d',[buffer.Axis[1].PrfPos[i],
                                buffer.Axis[1].AtlPos[i]]));
        end;
    end;
until (1 = available);          //等待缓冲区放满

GT_StopWatch;                  //停止数据监测
GT_Close;                      //关闭运动控制器
list.SaveToFile('output.txt'); //将数据写入文件
list.Free;
end;

```

例程 10-3 使用 Visual Basic 进行编程

```

Const BUFFER_SIZE = 400        //定义缓冲区的数据容量

Private Sub Command1_Click()    //点击按钮开始数据监测
    Dim fso As New FileSystemObject
    Dim ts As TextStream
    Set ts = fso.CreateTextFile("output.txt") //创建文件，用于保存数据
    GT_Open &H300              //打开运动控制器

```

```

GT_Reset //复位运动控制器
Dim Config(Axis_NUM) As Integer //Axis_NUM 定义于函数库
Config(0) = &H0
Config(1) = &H3 //监测第 2 轴的规划位置和
Config(2) = &H0 //实际位置
Config(3) = &H0
Config(4) = &H0
Config(5) = &H0
Config(6) = &H0
Config(7) = &H0
GT_SetWatch Config(0), BUFFER_SIZE //设置所要监测数据和缓冲区数据
//容量将数组第 1 个元素按照引用方
//式传递给函数库
Dim PrfPos(BUFFER_SIZE) As Long //创建规划位置数据缓冲区
Dim AtlPos(BUFFER_SIZE) As Long //创建实际位置数据缓冲区
Dim buffer As TWatchBuffer
buffer.Axis(1).PrfPos = VarPtr(PrfPos(0)) //挂接规划位置数据缓冲区
buffer.Axis(1).AtlPos = VarPtr(AtlPos(0)) //挂接实际位置数据缓冲区
GT_SetKp 2, 10 //设置第 2 轴的比例增益
GT_Update 2 //刷新第 2 轴参数
GT_AxisOn 2 //第 2 轴伺服使能

MsgBox "Click button to continue..."

GT_SetPos 2, 100000 //设置第 2 轴的目标位置
GT_SetVel 2, 10 //设置第 2 轴的目标速度
GT_SetAcc 2, 1 //设置第 2 轴的加速度
GT_StartWatch 0 //启动数据监测
//每个控制周期监测 1 次
GT_Update 2 //刷新第 2 轴参数
Dim i As Long
Dim status As Long
Dim available As Integer
Do
    GT_GetWatch buffer, available //检测数据缓冲区是否放满
    If available = 1 Then //当数据缓冲区放满以后
        For i = 0 To BUFFER_SIZE - 1 //存储到文件当中
            ts.WriteLine PrfPos(i) & "," & AtlPos(i)
        Next i
    End If
    GT_GetSts 2, status //检测第 2 轴的状态
Loop Until (status And &H400) = 0 //等待第 2 轴运动停止

```

```

Do                                     //第 2 轴运动停止以后
    GT_GetWatch buffer, available      //再次监测 1 批数据
    If available = 1 Then
        For i = 0 To BUFFER_SIZE - 1
            ts.WriteLine PrfPos(i) & "," & AtlPos(i)
        Next i
    End If
    Loop Until 1 = available           //等待缓冲区放满
    GT_StopWatch                       //停止数据监测
    GT_Close                           //关闭运动控制器
    ts.Close
End Sub

```

10.1.3 重点说明

10.1.3.1 指定监测的控制轴参数和数据缓冲区容量

调用 GT_SetWatch 指令指定监测的控制轴参数和数据缓冲区容量。该指令有 2 个参数，第一个参数指定监测的控制轴参数（按位指定），具体定义如表 10-2：

表 10-2 监视参数位定义

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
保留	控制电压	保留	保留	实际速度	规划速度	实际位置	规划位置

当某个标志位为 1 时，表示监测对应的参数，标志位为 0 时，表示不监测该参数。

第二个参数指定数据缓冲区容量，如果用户指定的数据缓冲区太大（大小与监视的参数个数有关），该指令返回 7。

下面这段程序将运动控制器设置为监测 1、2、3 轴的实际位置，缓冲区容量为 400。

```

unsigned short config[8]={0x2,0x2,0x2,0,0,0,0,0}; //监测 1、2、3 轴实际位置
short rtn;
rtn=GT_SetWatch(config,400);           //设置监测参数和数据缓冲区容量
if(rtn!=0)                             //检查指令返回值
{
    cout<<"GT_SetWatch =\t"<<rtn<<endl;
    return;
}

```

10.1.3.2 数据缓冲区数据结构

各轴数据结构如下所示：

```
typedef struct
{
    long *PrfPos;
    long *AtlPos;
    double *PrfVel;
    double *AtlVel;
    short *Voltage;
} TAxis;
```

数据缓冲区数据结构如下所示：

```
typedef struct
{
    TAxis Axis[AXIS_NUM];           //AXIS_NUM 为轴数
} TWatchBuffer;
```

用户所监测的数据必须挂接相应的数据缓冲区,否则将产生内存访问错误。

下面这段程序挂接 1、2、3 轴的实际位置的数据缓冲区。

```
long AtlPos[3][400];                //创建数据缓冲区
TWatchBuffer buffer;
buffer.Axis[0].AtlPos = AtlPos[0];   //挂接控制轴缓冲区
buffer.Axis[1].AtlPos = AtlPos[1];
buffer.Axis[2].AtlPos = AtlPos[2];
```

10.1.3.3 读取缓冲区数据

调用 GT_GetWatch 指令查询缓冲区状态,读取缓冲区数据。第一个参数指定数据缓冲区,第二个参数指示数据缓冲区是否放满。当第二个参数返回值为 1 时,表示数据缓冲区已经放满,此时用户可以访问缓冲区中的数据。

```
long AtlPos[3][400];                //创建数据缓冲区
TWatchBuffer buffer;

buffer.Axis[0].AtlPos = AtlPos[0];   //挂接 1 轴实际位置缓冲区
buffer.Axis[1].AtlPos = AtlPos[1];   //挂接 2 轴实际位置缓冲区
buffer.Axis[2].AtlPos = AtlPos[2];   //挂接 3 轴实际位置缓冲区

short available;
int status;
do{
    GT_GetWatch(&buffer,&available);
    if(1==available)                  //检查数据缓冲区是否放满
```

```

{
    for(i=0;i<400;++i)                //将数据写入文件
    {
        fprintf(fp,"%ld\t%ld\t%ld\r\n",AtlPos[0][i],AtlPos[1][i],AtlPos[2][i]);
    }
}
GT_GetSts(1,&status);                //读取 1 轴的状态
}while(0x400==(status&0x400));        //等待 1 轴运动停止
```

10.1.3.4 “启动/停止”数据监测

调用 GT_StartWatch 启动数据监测，指令参数指定每隔多少控制采样周期监测一次。

调用 GT_StopWatch 停止数据监测。

10.2 模拟电压输入

10.2.1 指令列表

表 10-3 模拟电压输入指令列表

指令	说明
GT_GetAdc	读取指定通道输入的模拟电压。
GT_SetAdcChn	设置模拟电压输入通道数目（GE-X00-SX）。

10.2.2 例程

例程 10-4 读取通道 1 的模拟电压输入

```

void main()
{
    short voltage;
    GT_Open();                //打开运动控制器
    GT_Reset();              //复位运动控制器
    if(0==GT_GetAdc(1,&voltage)) //读取通道 1 的模拟电压值
    {
        printf("\n Channel 1 Voltage : %d",voltage);
        //打印通道 1 的模拟电压值
    }
    else
    {
        printf("\nCommand Error !");
    }
    GT_Close();              //关闭运动控制器
}
```

}

10.2.3 重点说明

GE 系列运动控制器提供了 8 路 12 位精度的外部模拟电压输入通道。调用指令 GT_GetAdc 读取指定通道的外部输入模拟电压值。输入电压按照表 10-4 进行计算。

表 10-4 模拟电压值和指令读取数值的对应关系

输入电压值 (V)	所对应的数值
+10	2047
0	0
-10	-2048

GE-X00-PX 的 8 路 A/D 的采样频率为 4KHZ ,GE-X00-SX 的 8 路 A/D 的采样频率为 770HZ。

GE-X00-SX 提供 GT_SetAdcChn 指令设置 A/D 通道数目 ,提高 A/D 的采样频率。当只有一路 A/D 时 , 采样频率可以提高到 $8 \times 770\text{HZ}=6.16\text{KHZ}$ 。

10.3 手脉功能

GE-X00-SX 运动控制器可通过手脉直接控制电机的运动，实现手动模式下的精确定位。

10.3.1 指令列表

表 10-5 手脉功能指令列表

指令	说明
GT_HandWheel	设置指定控制轴为手脉控制模式，并设置跟随倍率。
GT_ExitHWheel	设置指定控制轴退出手脉控制模式。
GT_SetSynVel	设置处于手脉控制模式下控制轴的最大跟随速度。
GT_SetSynAcc	设置处于手脉控制模式下控制轴的加速度。

10.3.2 例程

例程 10-5 根据通用 I/O 的输入，将对应的控制轴切换到手脉控制模式

```
short InMPG()
{
    short rtn;
    unsigned short axis=1;
    double rate=0.1;
    unsigned short CurInpt;
    rtn=GT_ExInpt(&CurInpt);           //读取当前外部输入值
    if(rtn!=0) return rtn;
    if(CurInpt & 0x1) axis=1;           //0 号 IO 输入，选中第一轴
    if(CurInpt & 0x2) axis=2;           //1 号 IO 输入，选中第二轴
    if(CurInpt & 0x4) axis=3;           //2 号 IO 输入，选中第三轴
    if(CurInpt & 0x8) rate=0.001;       //3 号 IO 输入，倍率设定位
    if(CurInpt & 0x10) rate=0.01;       //4 号 IO 输入，倍率设定位
    if(CurInpt & 0x20) rate=0.1;        //5 号 IO 输入，倍率设定位
    rtn=GT_HandWheel(axis,rate);        //将指定控制轴切换到
                                        //手脉控制模式并设置手脉倍率
    if(rtn!=0) return rtn;
}
```

10.3.3 重点说明

GE-X00-SX 提供一路专用的手脉输入接口 (CN10)，该接口可接收差动或单端的脉冲输入信号。

只能有一个控制轴处于手脉控制模式。如果将某个控制轴切换到手脉控制模式，原先处于手脉控制模式的控制轴自动退出手脉控制模式。用户只能在没

有任何轨迹运动时将控制轴切换到手脉控制模式。

控制轴处于手脉控制模式时，必须设定控制轴的最大跟随速度和加速度。控制轴按照跟随倍率跟踪手脉的速度和位置，并保证到位准确。控制轴从启动速度开始，按照给定加速度加速到目标跟随速度（由跟随倍率和手脉速度计算求出），在运动过程中按照倍率跟随手脉速度的变化，并在手脉运动停止时降速到启动速度且停止。若根据倍率和当前手脉速度计算出的跟随速度大于设定的最大跟随速度，取最大跟随速度为当前跟随速度。

对于步进电机，应当使控制轴跟随速度尽量避开系统的共振区（即设置合理的启动速度、加速度和最大跟随速度），否则在跟随过程中可能产生较大的振动。

10.4 主轴转速控制

10.4.1 指令列表

表 10-6 主轴转速控制指令列表

指 令	说 明
GT_SetSpindleVel	设置主轴转速。
GT_CloseSpindle	关闭主轴。

10.4.2 例程

例程 10-6 设定主轴转速

```
void main()
{
    GT_Open();                //打开运动控制器
    GT_Reset();               //复位运动控制器
    if(0== GT_SetSpindleVel (32767)    //设置主轴转速
    {
        printf("\n GT_SetSpindleVel OK!");
    }
    delay(10000);             //延时 10s
    if(0== GT_CloseSpindle ()         //关闭主轴
    {
        printf("\n GT_ CloseSpindle OK!");
    }
    GT_Close();               //关闭运动控制器
}
```

10.4.3 重点说明

GE-X00-SX 提供一路专用的主轴控制输出接口（CN8）。调用指令 GT_SetSpindleVel 可立即改变主轴控制输出电压，调节主轴的转速。

表 10-7 输出电压值和设定数值的对应关系

输入电压值（V）	所对应的数值
+10	36767
0	0
-10	-32768

调用指令 `GT_CloseSpindle` 可以停止主轴。不要采取调用指令 `GT_SetSpindleVel` 将控制电压设置为 0 的方式停止主轴，避免由于零漂电压使主轴不能完全停止而产生意外。

第十一章 指令列表

表 11-1 GE-X00-PX 运动控制器指令列表

指 令	说 明
设置运动控制器	
GT_Open	打开运动控制器
GT_Close	关闭运动控制器
GT_Reset	复位运动控制器
GT_HardRst	硬件复位运动控制器
GT_HookCommand	挂接用户自定义的指令处理函数
设置控制轴	
GT_AlarmOn	控制轴驱动报警信号有效
GT_AlarmOff	控制轴驱动报警信号无效
GT_LmtsOn	控制轴限位信号有效
GT_LmtsOff	控制轴限位信号无效
GT_LmtSns	设置控制轴限位触发电平
GT_EncSns	设置控制轴编码器的记数方向
GT_HomeSns	设置 Home 信号的触发沿
GT_IndexSns	设置 Index 信号的触发沿
GT_CloseLp	将控制轴设置为闭环控制模式
GT_OpenLp	将控制轴设置为直接电压输出模式
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出
GT_StepDir	将控制轴的脉冲输出模式设置为“脉冲 + 方向”
GT_StepPulse	将控制轴的脉冲输出模式设置为“正负脉冲”
GT_AuStpOn	控制轴跟随误差超限自动停止有效
GT_AuStpOff	控制轴跟随误差超限自动停止无效
GT_DrvRst	清除指定控制轴的伺服报警信号
GT_EncOn	打开控制轴的编码器输入
GT_EncOff	关闭控制轴的编码器输入
设置控制轴运动模式	
GT_PrflT	将控制轴设置为梯形曲线运动模式
GT_PrflS	将控制轴设置为 S 曲线运动模式
GT_PrflV	将控制轴设置为速度控制模式
设置控制轴运动参数	
GT_SetPos	设置控制轴的目标位置
GT_GetPos	读取控制轴的目标位置
GT_SetVel	设置控制轴的目标速度
GT_GetVel	读取控制轴的目标速度
GT_SetAcc	设置控制轴的加速度

第十一章 指令列表

指 令	说 明
GT_GetAcc	读取控制轴的加速度
GT_SetMAcc	设置控制轴的加速度，仅用于 S 曲线
GT_GetMAcc	读取控制轴的加速度，仅用于 S 曲线
GT_SetJerk	设置控制轴的加加速度
GT_GetJerk	读取控制轴的加加速度
GT_ZeroPos	控制轴的实际位置清零
GT_SetAtlPos	设置控制轴的实际位置
GT_GetAtlPos	读取控制轴的实际位置
GT_GetPrfPos	读取控制轴的规划位置
GT_GetAtlErr	读取控制轴的跟随误差
GT_GetAtlVel	读取控制轴的实际速度
GT_GetPrfVel	读取控制轴的规划速度
设置控制轴控制参数	
GT_SetKp	设置控制轴的比例增益
GT_GetKp	读取控制轴的比例增益
GT_SetKi	设置控制轴的积分增益
GT_GetKi	读取控制轴的积分增益
GT_SetKd	设置控制轴的微分增益
GT_GetKd	读取控制轴的微分增益
GT_SetKvff	设置控制轴的速度前馈增益
GT_GetKvff	读取控制轴的速度前馈增益
GT_SetKaff	设置控制轴的加速度前馈增益
GT_GetKaff	读取控制轴的加速度前馈增益
GT_SetMtrBias	设置控制轴的零漂电压补偿值
GT_GetMtrBias	读取控制轴的零漂电压补偿值
GT_SetMtrLmt	设置控制轴的输出电压饱和极限值
GT_GetMtrLmt	读取控制轴的输出电压饱和极限值
GT_SetILmt	设置控制轴的积分饱和极限值
GT_GetILmt	读取控制轴的积分饱和极限值
GT_GetIntgr	读取控制轴的误差积分值
GT_SetPosErr	设置跟随误差极限值
GT_GetPosErr	读取跟随误差极限值
控制轴运动控制指令	
GT_AxisOn	打开控制轴伺服使能
GT_AxisOff	关闭控制轴伺服使能
GT_Update	控制轴参数（包括运动参数和控制参数）刷新
GT_MltiUpdt	多个控制轴参数（包括运动参数和控制参数）刷新
GT_SmthStp	平滑停止控制轴
GT_AbptStp	急停控制轴

指 令	说 明
状态检测	
GT_GetSts	读取控制轴状态
GT_GetCmdSts	读取指令执行出错原因
GT_ClrSts	清除控制轴状态
GT_RstSts	清除控制轴的指定状态
GT_GetMode	读取控制轴模式
Home/Index 捕获	
GT_CaptHome	将控制轴设置为 Home 信号捕获
GT_CaptIndex	将控制轴设置为 Index 信号捕获
GT_GetCapt	读取位置捕获值
数字 I/O	
GT_ExInpt	读取通用数字 I/O 的输入信号
GT_ExOpt	设置通用数字 I/O 的输出信号
GT_GetHomeSwt	读取原点开关电平状态
GT_GetLmtSwt	读取限位开关电平状态
D/A	
GT_SetMtrCmd	设置控制轴的输出电压, 仅在直接电压输出模式下有效
GT_GetMtrCmd	读取控制轴的输出电压
多卡操作指令	
GT_GetCurrentCardNo	读取当前工作的运动控制器卡号
GT_SwitchtoCardNo	设置当前工作的运动控制器卡号
A/D (可选)	
GT_GetAdc	读取指定通道的模拟电压输入值
数据监测 (可选)	
GT_SetWatch	设置所要监测的参数
GT_GetWatch	读取监测数据
GT_StartWatch	启动监测
GT_StopWatch	停止监测

表 11-2 GE-X00-SX 运动控制器指令列表

指 令	说 明
设置运动控制器	
GT_Open	打开运动控制器
GT_Close	关闭运动控制器
GT_Reset	复位运动控制器
GT_HardRst	硬件复位运动控制器
GT_HookCommand	挂接用户自定义的指令处理函数
设置控制轴	
GT_AlarmOn	控制轴驱动报警信号有效
GT_AlarmOff	控制轴驱动报警信号无效
GT_LmtsOn	控制轴限位信号有效
GT_LmtsOff	控制轴限位信号无效
GT_LmtSns	设置控制轴限位触发电平
GT_EncSns	设置控制轴编码器的记数方向
GT_HomeSns	设置 Home 信号的触发沿
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出
GT_StepDir	将控制轴的脉冲输出模式设置为“脉冲 + 方向”
GT_StepPulse	将控制轴的脉冲输出模式设置为“正负脉冲”
GT_DrvRst	清除指定控制轴所连接驱动器的伺服报警信号
GT_EncOn	打开控制轴的编码器输入
GT_EncOff	关闭控制轴的编码器输入
设置控制轴运动参数	
GT_ZeroPos	控制轴的实际位置清零
GT_SetAtlPos	设置控制轴的实际位置
GT_GetAtlPos	读取控制轴的实际位置
GT_GetPrfPnt	读取轨迹运动中所有控制轴的规划位置值
GT_GetSegPnt	读取轨迹运动中所有控制轴的当前段终点位置
GT_GetBrkPnt	读取所有控制轴缓冲区连续轨迹运动中断时的断点位置
GT_GetPrfVel	读取控制轴的规划速度
设置控制轴控制参数	
GT_SetKp	设置控制轴的比例增益
GT_GetKp	读取控制轴的比例增益
GT_SetKi	设置控制轴的积分增益
GT_GetKi	读取控制轴的积分增益
GT_SetKd	设置控制轴的微分增益
GT_GetKd	读取控制轴的微分增益
GT_SetKvff	设置控制轴的速度前馈增益

第十一章 指令列表

指令	说明
GT_GetKvff	读取控制轴的速度前馈增益
GT_SetKaff	设置控制轴的加速度前馈增益
GT_GetKaff	读取控制轴的加速度前馈增益
GT_SetMtrBias	设置控制轴的零漂电压补偿值
GT_GetMtrBias	读取控制轴的零漂电压补偿值
GT_SetMtrLmt	设置控制轴的输出电压饱和极限值
GT_GetMtrLmt	读取控制轴的输出电压饱和极限值
GT_SetILmt	设置控制轴的积分饱和极限值
GT_GetILmt	读取控制轴的积分饱和极限值
GT_SetPosErr	设置跟随误差极限值
GT_GetPosErr	读取跟随误差极限值
控制轴运动控制指令	
GT_AxisOn	打开控制轴伺服使能
GT_AxisOff	关闭控制轴伺服使能
GT_Update	控制轴参数（包括运动参数和控制参数）刷新
GT_MltiUpdt	多个控制轴参数（包括运动参数和控制参数）刷新
连续轨迹运动指令	
GT_LnXY	二维直线插补运动
GT_LnXYG0	二维直线插补定位运动
GT_LnXYZ	三维直线插补运动
GT_LnXYZG0	三维直线插补定位运动
GT_LnXYZA	四维直线插补运动
GT_LnXYZAG0	四维直线插补定位运动
GT_ArcXY	XY 平面内的两轴圆弧插补（圆心位置，角度）
GT_ArcXYP	XY 平面内的两轴圆弧插补（终点位置，半径）
GT_ArcYZ	YZ 平面内的两轴圆弧插补（圆心位置，角度）
GT_ArcYZP	YZ 平面内的两轴圆弧插补（终点位置，半径）
GT_ArcZX	ZX 平面内的两轴圆弧插补（圆心位置，角度）
GT_ArcZXP	ZX 平面内的两轴圆弧插补（终点位置，半径）
GT_SetSynVel	设置轨迹运动的进给速度
GT_SetSynAcc	设置轨迹运动的进给加速度
GT_SetStrtVel	设定轨迹运动的启动速度
GT_SetStpAcc	设定轨迹运动的停止加速度
GT_SetMaxVel	设定轨迹运动的最大速度
GT_Override	设定除 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 和定位指令之外的轨迹运动的速度倍率
GT_OverrideG0	设定 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 和定位指令的速度倍率
GT_StpMtn	平滑停止连续轨迹运动

第十一章 指令列表

指令	说明
GT_EStpMtn	紧急停止连续轨迹运动
GT_MapCnt	设置坐标偏移量
缓冲区管理指令	
GT_StrtList	打开并清空缓冲区
GT_MvXY	在二维坐标系中，定位缓冲区连续轨迹运动坐标起点
GT_MvXYZ	在三维坐标系中，定位缓冲区连续轨迹运动坐标起点
GT_MvXYZA	在四维坐标系中，定位缓冲区连续轨迹运动坐标起点
GT_EndList	关闭缓冲区
GT_AddList	重新打开被关闭的缓冲区
GT_StrtMtn	启动多段连续轨迹运动
GT_RestoreMtn	恢复被中断的多段连续轨迹运动
GT_GetMtnNm	读取轨迹段编号
状态检测	
GT_GetSts	读取控制轴状态
GT_GetCmdSts	读取指令执行出错原因
GT_GetCrdSts	读取连续轨迹运动状态
GT_ClrSts	清除指定控制轴状态
GT_RstSts	清除指定控制轴的指定状态
Home/Index 捕获	
GT_CaptHome	将控制轴设置为 Home 信号捕获
GT_CaptIndex	将控制轴设置为 Index 信号捕获
GT_CaptProb	设置探针捕获功能
GT_GetCapt	读取位置捕获值
数字 I/O	
GT_ExInpt	读取通用数字 I/O 的输入信号
GT_ExOpt	设置通用数字 I/O 的输出信号
GT_GetHomeSwt	读取原点开关电平状态
GT_GetLmtSwt	读取限位开关电平状态
GT_BufIO	缓冲区内通用 IO 输出指令
GT_BufIOBit	缓冲区内设置高速 IO 输出电平，控制激光开关
运动速度控制指令	
GT_SetDccVel	设定线段的终点速度
前瞻预处理指令	
GT_AddLookData	将轨迹特征数据加入预处理缓冲区
GT_CalVel	计算预处理当前轨迹段的终点速度
GT_InitLookAhead	设置前瞻预处理功能模块的初始化参数
辅助编码器指令	
GT_EncPos	读取辅助编码器的位置
多卡操作指令	

指令	说明
GT_GetCurrentCardNo	读取当前工作卡号
GT_SwitchtoCardNo	设置当前工作卡号
手脉控制指令（可选）	
GT_HandWheel	将控制轴切换到手脉控制模式
GT_ExitHWheel	退出手脉控制模式
主轴控制指令（可选）	
GT_CloseSpindle	关闭主轴控制输出
GT_SetSpindleVel	设置主轴控制输出
A/D（可选）	
GT_GetAdc	读取指定通道的模拟电压输入值
GT_SetAdcChn	设置指定通道数
数据监测（可选）	
GT_SetWatch	设置所要监测的参数
GT_GetWatch	读取监测数据
GT_StartWatch	启动监测
GT_StopWatch	停止监测

第十二章 指令说明



运动控制器所有指令按照字母顺序排列，指令原型以 C 语言描述。
如果正在学习使用运动控制器，或者应用程序正处于调试阶段，请不要连接机械系统，以免误操作损坏设备。在首次连接机械系统时，仔细检查限位开关的设置、编码器计数方向是否正确、PID 参数是否合适、运动控制参数是否合理，以确保机械系统的安全。

GT_AbptStp

指令原型：short GT_AbptStp(unsigned short axis)

指令说明：急停指定控制轴，并将指定控制轴的目标速度和规划速度同时设置为 0。该指令执行以后立即生效。详细说明参阅第五章。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX。

指令示例：当通用数字量输入接口的 EXI15 为高电平时，急停 1 轴。

```
void main()
{
    unsigned short exInpt;
    GT_HookCommand(CommandHandle);           //例程 5-1
    GT_Open();                               //打开运动控制器
    GT_Reset();                              //复位运动控制器
    AxisInitial(1,0);                        //例程 3-2
    AxisRunT(1,1000000,1,0.1);              //例程 5-1
    do
    {
        GT_ExInpt(&exInpt); //读取通用数字量输入接口状态
        if((exInpt&0x8000)==0x8000)
        {
            GT_AbptStp(1);                  //急停 1 轴
        }
    }while((exInpt&0x8000)!=0x8000);
}
```

GT_AddList

指令原型：short GT_AddList(void)

指令说明：在调用指令 GT_EndList 关闭缓冲区之后，如果用户需要继续增加轨迹

描述或参数指令，可以调用本指令再次打开缓冲区。当缓冲区处于打开状态，该指令无效。

适用板卡：GE-X00-SX。

指令示例：在用 GT_EndList 指令关闭缓冲区后，再次打开缓冲区。

```
short  AddList ()
{
    short rtn;
    rtn=GT_StrtList();                //打开缓冲区
    if(0!=rtn) return  rtn;
    rtn=GT_MvXYZ(0,0,0,0,5,0.1);      //定位运动的起点
    if(0!=rtn) return  rtn;
    ...                               //继续添加指令
    rtn=GT_EndList();                //指令发送结束后 ,结束缓冲区
    if(0!=rtn) return  rtn;
    rtn=GT_AddList();                //重新打开缓冲区
    if(0!=rtn) return  rtn;
    rtn=GT_LnXY(20000,30000);         //向缓冲区中添加运动指令
    if(0!=rtn) return  rtn;
    ...                               //继续添加指令
}
```

GT_AddLookData

指令原型：short GT_AddLookData(char code, char plane_group, double r, double x, double y, double z, double vel, double cx, double cy, int i, long n, bool flag)

指令说明：将数据发送到预处理缓冲区。指令的返回值若为“0”表示预处理缓冲区接收成功，为“1”表示该指令传送的位置数据与前次相同，此时需要发下一条。

指令参数：参数说明——

参 数 名 称	参 数 意 义	说 明	
code	曲线类型	0	G00 直线
		1	G01 直线
		2	顺圆弧
		3	逆圆弧
plane_group	平面代码	17	XY 平面
		18	ZX 平面
		19	YZ 平面
r	圆弧半径	若当前段不是圆弧，r 设为 0	
x , y , z	终点位置	单位：mm	
vel	当前段目标速度	单位：mm/s	
cx , cy	圆弧圆心位置	若当前段不是圆弧，cx , cy 设为 0	
i	压入到缓冲区的具体位置		

n	当前段的标识	用户加工代码的段号
flag	当前段是否降速到 0	一般情况下设为 false。当工艺特征需要系统停止时（如换刀等操作），flag 为 true。

适用板卡：GE-X00-SX。

GT_AlarmOff

指令原型：short GT_AlarmOff(unsigned short axis)

指令说明：将指定控制轴的驱动报警输入信号设置为无效。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_AlarmOn。

GT_AlarmOn

指令原型：short GT_AlarmOn(unsigned short axis)

指令说明：将指定控制轴的驱动报警输入信号设置为有效，默认情况下驱动报警有效。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：

```
short AlarmConfig(unsigned short axis,short enable)
{
    if(enable)
    {
        return GT_AlarmOn(axis);           //控制轴轴驱动报警输入信号有效
    }
    else
    {
        return GT_AlarmOff(axis);          //控制轴轴驱动报警输入信号无效
    }
}
```

GT_ArcXY

指令原型：short GT_ArcXY(double x_center, double y_center, double angle)

指令说明：该指令实现 XOY 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

指令参数：x_center、y_center 是圆弧圆心位置，单位是 pulse；angle 是旋转角度，其单位是度，正负代表旋转方向，旋转角度的取值范围为[-360,360]度。旋转方向参见图 6-1 圆弧插补正方向的相关说明。

适用板卡：GE-X00-SX。

指令示例：举例——立即执行 XOY 平面圆弧插补命令。

```
short  ArcXY ()
{
    short rtn;
    rtn=GT_SetSynVel(5);                //设定运动速度
    if(0!=rtn) return  rtn;
    rtn=GT_SetSynAcc(0.1);              //设定运动加速度
    if(0!=rtn) return  rtn;
    rtn=GT_ArcXY(40000,30000,180);      //开始 XOY 平面圆弧插补运动
    if(0!=rtn) return  rtn;
    return 0;
}
```

GT_ArcXYP

指令原型：short GT_ArcXYP(double x_end, double y_end, double r, short direction)

指令说明：该指令实现 XOY 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

指令参数：x_end、y_end 是圆弧终点位置；r 是圆弧半径并带符号，其符号表示此段圆弧是优弧还是劣弧（正：劣弧，负：优弧），位置和半径的单位是 pulse；direction 是圆弧旋转方向，取值为 1 表示正向旋转，-1 表示负向旋转。旋转方向参见图 6-1 圆弧插补正方向的说明。

适用板卡：GE-X00-SX。

指令示例：举例——发送 XOY 平面圆弧插补命令到缓冲区。

```
short  ArcXYP ()
{
    short rtn;
    rtn=GT_StrtList();
    if(0!=rtn) return  rtn;
    rtn=GT_MvXY(0,0, 5,0.1);
    if(0!=rtn) return  rtn;
    rtn=GT_ArcXYP(40000,0,20000,-1);
    if(0!=rtn) return  rtn;
    rtn=GT_EndList();
    if(0!=rtn) return  rtn;
    return 0;
}
```

GT_ArcYZ

指令原型：short GT_ArcYZ(double y_center, double z_center, double angle)

指令说明：该指令实现 YOZ 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命

令);如果是缓冲区第一条运动轨迹段,则起点是定位指令指定的位置。
指令参数: y_center、z_center 是圆弧圆心位置,单位是 pulse; angle 是旋转角度,其单位是度,正负代表旋转方向,旋转角度的取值范围为[-360,360]度。

适用板卡: GE-X00-SX。

指令示例: 举例——立即执行 YOZ 平面圆弧插补命令。

```
short  ArcYZ ()
{
    short rtn;
    rtn=GT_SetSynVel(5);
    if(0!=rtn)  return  rtn;
    rtn=GT_SetSynAcc(0.1);
    if(0!=rtn)  return  rtn;
    rtn=GT_ArcYZ(40000,30000,180);
    if(0!=rtn)  return  rtn;
    reurn 0;
}
```

GT_ArcYZP

指令原型: short GT_ArcYZP(double y_end, double z_end, double r, short direction)

指令说明: 该指令实现 YOZ 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置(缓冲区命令)或者是当前位置(立即命令);如果是缓冲区第一条运动轨迹段,则起点是定位指令指定的位置。

指令参数: y_end、z_end 是圆弧终点位置; r 是圆弧半径并带符号,其符号表示此段圆弧是优弧还是劣弧(正:劣弧,负:优弧),位置和半径的单位是 pulse; direction 是圆弧旋转方向,取值为 1 表示正向旋转,-1 表示负向旋转。

适用板卡: GE-X00-SX。

指令示例: 举例——立即执行 YOZ 平面圆弧插补命令。

```
short  ArcYZP ()
{
    short rtn;
    rtn=GT_StrtList();
    if(0!=rtn)  return  rtn;
    rtn=GT_MvXYZ(0,0,0,5,0.1);
    if(0!=rtn)  return  rtn;
    rtn=GT_ArcYZP(40000,0,20000,-1);
    if(0!=rtn)  return  rtn;
    rtn=GT_EndList();
    if(0!=rtn)  return  rtn;
    return 0;
}
```

GT_ArcZX

指令原型：short GT_ArcZX(double z_center, double x_center, double angle)

指令说明：该指令实现 ZOX 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

指令参数：z_center、x_center 是圆弧圆心位置，单位是 pulse；angle 是旋转角度，其单位是度，正负代表旋转方向，旋转角度的取值范围为[-360,360]度。

适用板卡：GE-X00-SX。

指令示例：举例——立即执行 ZOX 平面圆弧插补命令。

```
short ArcZX ()
{
    short rtn;
    rtn=GT_SetSynVel(5);
    if(0!=rtn) return rtn;
    rtn=GT_SetSynAcc(0.1);
    if(0!=rtn) return rtn;
    rtn=GT_ArcZX(40000,30000,180);
    if(0!=rtn) return rtn;
    return 0;
}
```

GT_ArcZXP

指令原型：short GT_ArcZXP(double z_end, double x_end, double r, short direction)

指令说明：该指令实现 ZOX 平面内的两轴圆弧插补。圆弧插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

指令参数：z_end、x_end 是圆弧终点位置；r 是圆弧半径并带符号，其符号表示此段圆弧是优弧还是劣弧（正：劣弧，负：优弧），位置和半径的单位是 pulse；direction 是圆弧旋转方向，取值为 1 表示正向旋转，-1 表示负向旋转。

适用板卡：GE-X00-SX。

指令示例：举例——发送 ZOX 平面圆弧插补命令到缓冲区。

```
short ArcZxp()
{
    short rtn;
    rtn=GT_StrtList();
    if(0!=rtn) return rtn;
    rtn=GT_MvXYZ(0,0,0,5,0.1);
    if(0!=rtn) return rtn;
    rtn=GT_ArcZXP(40000,0,20000,-1);
    if(0!=rtn) return rtn;
}
```

```
    rtn=GT_EndList();  
    if(0!=rtn)  return  rtn;  
    return 0;  
}
```

GT_AuStpOff

指令原型：short GT_AuStpOff(unsigned short axis)

指令说明：关闭指定控制轴的跟随误差超限自动停止功能。执行该指令以后，如果该轴的实际位置和规划位置之间的误差超过所设定的跟随误差极限，那么运动控制器不会停止该轴的运动。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PV。

指令示例：参阅 GT_AuStpOn。

GT_AuStpOn

指令原型：short GT_AuStpOn(unsigned short axis)

指令说明：使能指定控制轴的跟随误差超限自动停止功能。执行该指令以后，如果该轴的实际位置和理论位置之间的误差超过所设定的跟随误差极限，那么运动控制器自动停止该轴的运动，并将指定控制轴状态寄存器的相应标志位置 1。跟随误差极限默认值为 32767，可以调用 GT_SetPosErr 指令重新设置跟随误差极限。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PV。

指令示例：

```
short AutoStopConfig(unsigned short axis,short enable)  
{  
    if(enable)  
    {  
        return GT_AuStpOn(axis);           //控制轴跟随误差超限自动停止有效  
    }  
    else  
    {  
        return GT_AuStpOff(axis);          //控制轴跟随误差超限自动停止无效  
    }  
}
```

GT_AxisOff

指令原型：short GT_AxisOff(unsigned short axis)

指令说明：关闭指定控制轴所连电机的伺服使能信号。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_AxisOn。

GT_AxisOn

指令原型：short GT_AxisOn(unsigned short axis)

指令说明：打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。如果指定控制轴连接伺服电机，并且输出控制量为模拟电压，那么必须首先设置指定控制轴位置环的 PID 参数。为了保证系统安全，调用该指令时，运动控制器会自动将目标位置和规划位置同步到实际位置。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：当通用数字量输入接口的 EXI15 为高电平时，1 轴伺服使能；当通用数字量输入接口的 EXI15 为低电平时，1 轴伺服关闭。

```
short AxisEnable(unsigned short axis)
{
    unsigned short exInpt;
    rtn = GT_ExInpt(&exInpt);           //读取通用数字量输入接口的状态
    if(0!=rtn) return rtn;
    if(exInpt&0x8000)
    {
        return GT_AxisOn(axis);        //如果 EI15 为高电平，打开控制轴的伺服
    }
    else
    {
        return GT_AxisOff(axis);       //如果 EI15 为低电平，关闭控制轴的伺服
    }
}
```

GT_BufIO

指令原型：short GT_BufIO(unsigned short io_status)

指令说明：在缓冲区方式下，控制通用 IO 的输出。该指令只能在运动控制器处于缓冲区输入方式下被调用。如果在立即指令下调用，返回值为-1。

两条运动指令之间最多允许两条该指令，用于在前一条轨迹运动指令结束和后一条轨迹运动指令开始之前控制通用 IO 的输出状态。

指令参数：io_status：表示 16 位通用 IO 输出的状态，取值范围为[0,0xffff]。

适用板卡：GE-X00-SX

GT_BufIOBit

指令原型：short GT_BufIOBit(unsigned short io_bit, short bit_status)

指令说明：在缓冲区方式下，设置指定的通用输出端口的高低电平状态。

该指令只能用于缓冲区方式下。

该指令一旦执行不能马上生效，必须有轨迹运动时才生效。如果有连续的两条 GT_BufIOBit 指令，则距离轨迹指令最近的一条有效。

指令参数：io_bit：表示指定的通用输出位通道，取值[0, 15]。

bit_status : 表示指定的通用输出的状态 (只能为 “ 0 ” 或者 “ 1 ”)。 “ 1 ” 表示使能高速 IO 的输出为高电平。 “ 0 ” 表示高速 IO 输出为低电平。

适用板卡 : GE-X00-SX

GT_Calvel

指令原型 : short GT_CalVel(double *vel,long *number)

指令说明 : 计算预处理缓冲区中当前轨迹段的终点速度。指令返回值若为 “ 0 ” 表示预处理缓冲区中还存在需要处理的数据 , 为 “ 1 ” 表示预处理结束。

指令参数 : vel : 返回终点速度。

number 对应的用户加工代码段号 , 对应 GT_AddLookData() 参数中的 n 数。

适用板卡 : GE-X00-SX。

GT_CaptHome

指令原型 : short GT_CaptHome(unsigned short axis)

指令说明 : 将指定控制轴的位置捕获设置为由 Home 信号触发。执行 GT_CaptHome 指令以后 , 当 Home 信号产生时 , 运动控制器自动锁存该控制轴的实际位置 , 并将该控制轴状态寄存器的位置捕获触发标志位 (bit3) 置 1。因此可以通过检查该控制轴状态寄存器的位置捕获触发标志位 (bit3) 确定 Home 捕获是否已经触发。当 Home 捕获触发以后 , 调用 GT_GetCapt 指令可以读取 Home 信号触发时该控制轴的实际位置。完成了一次 Home 捕获以后 , 在启动下一次 Home 捕获之前 , 必须首先调用 GT_ClrSts 指令清除该控制轴状态寄存器的位置捕获触发标志位 (bit3) , 然后重新调用 GT_CaptHome 指令。

指令参数 : 指定控制轴编号。

适用板卡 : GE-X00-PX , GE-X00-SX。

指令示例 : 参阅例程 7-1。

GT_CaptIndex

指令原型 : short GT_CaptIndex(unsigned short axis)

指令说明 : 将指定控制轴的位置捕获设置为由编码器 C 相脉冲信号或者光栅尺的 Index 信号触发。执行 GT_CaptIndex 指令以后 , 当编码器 C 脉冲信号产生时 , 运动控制器自动锁存该控制轴的实际位置 , 并将该控制轴状态寄存器的位置捕获触发标志位 (bit3) 置 1。因此可以通过检查该控制轴状态寄存器的位置捕获触发标志位 (bit3) 确定 Index 捕获是否已经触发。当 Index 捕获触发以后 , 调用 GT_GetCapt 指令可以读取 Index 信号触发时该控制轴的实际位置。完成了一次 Index 捕获以后 , 在启动下一次 Index 捕获之前 , 必须首先调用 GT_ClrSts 指令清除该控制轴状态寄存器的位置捕获触发标志位 (bit3) , 然后重新调用 GT_CaptIndex 指令。

指令参数 : 指定控制轴编号。

适用板卡 : GE-X00-PX , GE-X00-SX。

指令示例 : 参阅例程 7-2。

GT_Close

指令原型：short GT_Close(void)

指令说明：关闭运动控制器。当退出应用程序时，调用该指令以释放运动控制器的访问权。

指令参数：无。

适用板卡：GE-X00-PX，GE-X00-SX。

GT_CaptProb

函数原型：short GT_CaptProb(void)；

函数说明：该函数设置捕获探针输入信号事件。本运动控制器以通用 IO 输入中的 0 号通道（EXI0）作为探针输入，调用该函数后，所有控制轴的位置捕获寄存器以及辅助编码器的位置捕获寄存器将记录探针信号到来时的实际位置。

当捕获事件发生时，所有控制轴状态字中表示有捕获发生的标志（即 bit3）置位。

执行一次 GT_CaptProb() 只能捕获到一次探针输入信号的位置信息。要想捕获下一个位置信息，必须重新调用本函数设置捕获探针输入信号事件。捕获到的实际位置同样可以作为控制轴的精确坐标原点。捕获探针输入信号的逻辑采用硬件完成，与运动轴的速度无关。

指令参数：无

适用板卡：所有 GT 系列卡

指令示例：举例——设置捕获探针输入信号事件，在运动过程中循环检测状态，在产生捕获时读取第二轴捕获到的实际位置值并打印。

```
void main()
{
    short rtn;
    unsigned short status;
    long actl_pos;
    rtn=GT_CaptProb();      error(rtn);
    ....
    rtn=GT_Axis(2);         error(rtn);
    rtn=GT_GetSts(&status); error(rtn);
    while(status&0x400)
    {
        if(status&0x8)
        {
            rtn=GT_GetAtlPos(&actl_pos); error(rtn);
            printf("the capture pos of axis 2 is: %ld\n",actl_pos);
            break;
        }
        rtn=GT_GetSts(&status);          error(rtn);
    }
}
```

}

GT_CloseLp

指令原型：short GT_CloseLp(unsigned short axis)

指令说明：将指定控制轴设置为闭环控制模式。各控制轴默认情况下为闭环控制模式。为了保证系统安全，在调用该指令之前，应当首先关闭伺服，然后再调用该指令切换到闭环控制模式。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PV。

GT_CloseSpindle

指令原型：short GT_CloseSpindle(void)

指令说明：关闭主轴信号输出。

适用板卡：GE-X00-SV。

指令示例：参阅例程 10-6。

GT_ClrSts

指令原型：short GT_ClrSts(unsigned short axis)

指令说明：将指定控制轴状态寄存器的 bit0 ~ 7 清 0，其它状态位不受影响。状态寄存器的定义请参阅第四章。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 3-2、3-3、3-4。

GT_CtrlMode

指令原型：short GT_CtrlMode(unsigned short axis,unsigned short mode)

指令说明：设置指定控制轴为模拟量输出或脉冲输出。GE-X00-XV 运动控制器的所有控制轴都能输出模拟电压或脉冲。调用 GT_CtrlMode(axis, 0)可将控制轴的设置为输出模拟电压，调用 GT_CtrlMode(axis, 1)可将控制轴设置为输出脉冲。

当伺服电机工作在速度模式下时，运动控制器应当设置为输出模拟电压。当伺服电机工作在位置模式下，或者使用步进电机，运动控制器应当设置为输出脉冲。

GE-X00-XV 运动控制器所有控制轴上电时默认为电压输出模式。

如果运动控制器的输出模式和电机工作模式不匹配，电机将无法运动。

指令参数：axis 指定控制轴编号；mode 设置控制量的输出模式：0 表示模拟量控制方式，1 表示脉冲控制方式。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：

```
short ControlModeConfig(unsigned short axis)
```

```
{
```

```
    unsigned short exInpt;
```

```
    rtn = GT_ExInpt(&exInpt);           //读取通用数字量输入接口的状态
```

```

if(0!=rtn) return rtn;
if(exInpt&0x8000)
{
    return GT_CtrlMode(axis,1);    //如果 EI15 为高电平 ,设置为脉冲输出
}
else
{
    return GT_CtrlMode(axis,0);    //如果 EI15 为低电平 ,设置为模拟电压输出
}
}

```

GT_DrvRst

指令原型：short GT_DrvRst(unsigned short axis)

指令说明：清除指定控制轴的驱动报警。控制轴状态寄存器的 bit1 指示控制轴是否处于报警状态。如果需要清除驱动器报警，那么必须首先调用 GT_AxisOff 指令关闭控制轴的伺服使能，然后延时一段时间，再调用 GT_DrvRst 指令即可清除控制轴的报警信号。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：清除指定控制轴的驱动报警信号。

```

short ClearAlarmSignal(unsigned short axis)
{
    short rtn;
    rtn = GT_AxisOff(axis);           //关闭控制轴的伺服使能
    if(0!=rtn) return rtn;
    delay(10000);                     //延时 10 秒
    rtn = GT_DrvRst(axis);            //清除控制轴的驱动报警
    return rtn;
}

```

GT_EncOff

指令原型：short GT_EncOff(unsigned short axis)

指令说明：关闭指定控制轴的编码器输入，若指定控制轴处于伺服使能的工作状态（已调用过 GT_AxisOn()命令）时，调用该指令非法。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

GT_EncOn

指令原型：short GT_EncOn(unsigned short axis)

指令说明：打开指定控制轴的编码器输入，若指定控制轴处于伺服使能的工作状态（已调用过 GT_AxisOn()命令）时，调用该指令非法。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

GT_EncPos

指令原型：GT_EncPos(unsigned short axis,long *pos)

指令说明：该指令用来读取辅助编码器实际位置值。

指令参数：axis 表示需要读取的辅助编码器号，*pos 读取指定辅助编码器的实际位置。对于带有辅助编码器的运动控制器（可选），可调用该函数读取辅助编码器的位置采样值。

适用板卡：GE-X00-SX。

指令示例：该例程读取 1 号辅助编码器位置值并打印。

```
short  ReadEncPos()
{
    short rtn;
    long pos
    rtn=GT_EncPos(1,&pos);
    if(0!=rtn) return rtn;
    printf("pos:%ld",pos);
}
```

GT_EncSns

指令原型：short GT_EncSns(unsigned short sense)

指令说明：设置运动控制器各轴编码器计数方向。只有电机旋转的正方向和编码器计数的正方向相一致，运动控制器才能正常工作。在实际应用中，可能由于电机设置不当或是接线错误，造成电机转向与编码器计数反向相反，导致运动控制器不能正常工作。此时可以调用 GT_EncSns 指令修改运动控制器所有控制轴的编码器计数方向。

指令参数：设置各轴编码器的记数方向。

适用板卡：GE-X00-PX，GE-X00-SV。

指令示例：参阅例程 3-2。

GT_EndList

指令原型：short GT_EndList(void)

指令说明：在缓冲区命令输入状态下，调用本指令结束缓冲区命令输入状态。

适用板卡：GE-X00-SX。

指令示例：举例——在缓冲区命令输入状态下，结束缓冲区命令输入状态。

```
short  EndList()
{
    short rtn;
    rtn=GT_StrtList();                //打开缓冲区
    if(0!=rtn) return rtn;
    rtn=GT_MvXYZ(0,0,0,16,3.7);      //定位缓冲区运动的起点
    if(0!=rtn) return rtn;
    rtn=GT_LnXYZ(1234,5678,9013);    //向缓冲区中添加直线指令
    if(0!=rtn) return rtn;
```

```
    rtn=GT_ArcXY(2345,6789,360);           //向缓冲区中添加圆弧指令
    if(0!=rtn) return rtn;
    rtn=GT_EndList();                       //结束缓冲区
    if(0!=rtn) return rtn;
}
```

GT_EStpMtn

指令原型：short GT_EStpMtn(void)

指令说明：该指令立即紧急停止轨迹插补运动。调用该指令后，运动控制器按照设定的急停加速度减速停止连续轨迹运动。若缓冲区连续轨迹运动状态，该指令关闭缓冲区，并在运动停止后保存断点信息，以便此后调用 GT_RestoreMtn 指令继续缓冲区连续轨迹运动。
用户若在执行定位指令时发出急停指令使运动停止，可能无法正确恢复缓冲区连续轨迹运动。

适用板卡：GE-X00-SX。

GT_ExInpt

指令原型：short GT_ExInpt(unsigned short *input)

指令说明：读取 16 路通用数字量输入。指令参数的状态位与端子板外部输入接口的对应关系请参阅第九章。

指令参数：读取 16 路通用数字 I/O 的输入信号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_ExOpt。

GT_ExitHWheel

指令原型：short GT_ExitHWheel(void)

指令说明：使处于手脉控制模式的控制轴退出手脉控制模式。

适用板卡：GE-X00-SX。

GT_ExOpt

指令原型：short GT_ExOpt(unsigned short output)

指令说明：设置 16 路通用数字量输出。指令参数的状态位与端子板外部输出接口的对应关系请参阅第九章。

指令参数：设置 16 路通用数字 I/O 的输出信号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：当外部输入接口 EXI1 为高电平时，外部输出接口 EXO2 为高电平。

```
short DigitalInputOutput()
{
    short rtn;
    unsigned short exInpt;
    rtn = GT_ExInpt(&exInpt);           //读取外部输入
    if(0!=rtn) return rtn;
    if(exInpt&0x2)
```

```

    {
        return GT_ExOpt(0x4);           //外部输出接口 EXO2 为高电平
    }
}

```

GT_GetAcc

指令原型：short GT_GetAcc(unsigned short axis,double *acc)

指令说明：读取指定控制轴的加速度。

指令参数：axis 是指定控制轴编号；*acc 读取指定控制轴的加速度。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetPos。

GT_GetAdc

指令原型：short GT_GetAdc(unsigned short channel,short *voltage)

指令说明：运动控制器提供了 8 路 12 位精度的外部模拟电压输入通道。调用 GT_GetAdc 指令可以读取指定通道的外部输入模拟电压值。外部输入模拟电压值和 GT_GetAdc 指令读取数值的对应关系请参阅第十章。

指令示例：参阅例程 10-4。

指令参数：channel 指定模拟电压输入通道；*voltage 读取指定输入通道的模拟电压。

适用板卡：GE-X00-PX，GE-X00-SX。

GT_GetAtlErr

指令原型：short GT_GetAtlErr(unsigned short axis,long *error)

指令说明：读取指定控制轴规划位置与实际位置之间的偏差。

指令参数：axis 指定控制轴编号；*error 读取指定控制轴的跟随误差。

适用板卡：GE-X00-PV。

指令示例：1 轴规划运动到位时（控制轴状态寄存器的 bit10 为 0），读取控制轴规划位置与实际位置之间的偏差。

```

void main()
{
    unsigned long status;
    long error;
    GT_HookCommand(CommandHandle);
    AxisInitial(1,0,1);           //引用例程 3-4
    AxisRunT(1,10000,10,1);       //引用例程 5-1
    do
    {
        GT_GetSts(1,&status);
    }while(status&0x400);         //等待控制轴运动停止
    GT_GetAtlErr(1,&error);        //读取控制轴的实际位置
    printf("\nError = %ld",error);
}

```


GT_GetAtlPos

指令原型：short GT_GetAtlPos(unsigned short axis,long *pos)

指令说明：读取指定控制轴的实际位置。

指令参数：axis 指定控制轴编号；*pos 读取指定控制轴的实际位置。GE-X00-SX 控制卡中调用该指令时，在步进控制卡或伺服控制卡工作在步进模式下时，取回当前规划位置值，在伺服控制卡工作在伺服模式时，指令取回编码器的实际位置值。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：1 轴规划运动到位时（控制轴状态寄存器的 bit10 为 0），读取控制轴的实际位置和规划位置。

```
void main()
{
    unsigned long status;
    long atlPos, prfPos;
    GT_HookCommand(CommandHandle);
    AxisInitial(1,0,1);                //引用例程 3-4
    AxisRunT(1,10000,10,1);            //引用例程 5-1
    do
    {
        GT_GetSts(1, &status);
    }while(status&0x400);               //等待控制轴运动停止
    GT_GetAtlPos(1, &atlPos);           //读取控制轴的实际位置
    GT_GetPrfPos(1, &prfPos);          //读取控制轴的规划位置
    printf("\nActual Position = %ld Profile Position = %ld", atlPos, prfPos);
}
```

GT_GetAtlVel

指令原型：short GT_GetAtlVel(unsigned short axis,double *vel)

指令说明：读取指定控制轴的实际速度。

指令参数：axis 指定控制轴编号；*vel 读取指定控制轴的实际速度。

适用板卡：GE-X00-PX。

指令示例：1 轴规划运动到位时（控制轴状态寄存器的 bit10 为 0），读取控制轴的实际速度和规划速度。

```
void main()
{
    unsigned long status;
    double atlVel, prfVel;
    GT_HookCommand(CommandHandle);
    AxisInitial(1,0,1);                //引用例程 3-4
    AxisRunT(1,10000,10,1);            //引用例程 5-1
    do
    {
```

```

        GT_GetSts(1, &status);
    }while(status&0x400);                //等待控制轴运动停止
    GT_GetAtlVel(1, &atlVel);            //读取控制轴的实际速度
    GT_GetPrfVel(1, &prfVel);            //读取控制轴的规划速度
    printf("\nActual Vel = %lf Profile Vel = %lf", atlVel, prfVel);
}

```

GT_GetBrkPnt

指令原型：short GT_GetBrkPnt(double *point)

指令说明：在缓冲区运行过程中，主机能够使用 GT_EStpMtn 和 GT_StpMtn 指令中断缓冲区连续轨迹运动。调用该指令可读取断点位置信息。

指令参数：*point 返回中断点的位置。按照 X, Y, Z 的顺序存放在数组中。

适用板卡：GE-X00-SX。

GT_GetCapt

指令原型：short GT_GetCapt(unsigned short axis, long *pos)

指令说明：读取指定控制轴的 Home 信号或者 Index 信号触发时，其实际位置捕获值。该值捕获以后一直不变，直到下次位置捕获触发。

指令参数：axis 指定控制轴编号；*pos 读取指定控制轴的捕获位置。

适用板卡：GE-X00-PX, GE-X00-SX。

指令示例：参阅例程 7-1。

GT_GetCmdSts

指令原型：short GT_GetCmdSts(unsigned short *status)

指令说明：确定指令执行出错的原因。运动控制器函数库的每条指令都有返回值，指示该指令是否执行成功。当指令返回值为 1 时，表示指令执行出错，这时可以调用该指令进一步确定出错原因。错误定义请参阅第二章。

指令参数：读取指令执行出错原因。

适用板卡：GE-X00-PX, GE-X00-SX。

GT_GetCrdSts

指令原型：short GT_GetCrdSts(unsigned short *status)

指令说明：该指令获得轨迹运动状态。

指令参数：*status 返回该状态，意义参阅第六章 - 连续轨迹运动的运动信息反馈。

适用板卡：GE-X00-SX。

指令示例：举例——读取轨迹运动状态，当发现 bit2 置位时，将所有轴伺服关断。

```

short CoordState()
{
    short rtn;
    unsigned short status;
    rtn=GT_GetCrdSts(&status);
    if(0==rtn) return rtn;
    if(status&0x4)

```

```

{
    rtn=GT_AxisOff(1);
    if(0==rtn) return rtn;
    rtn=GT_AxisOff(2);
    if(0==rtn) return rtn;
    rtn=GT_AxisOff(3);
    if(0==rtn) return rtn;
}
return 0;
}

```

GT_GetCurrentCardNo

指令原型：short GT_GetCurrentCardNo(void)

指令说明：该指令获取当前运动控制器的卡号。运动控制器的卡号取值范围为0—15。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：打开两个运动控制器，关闭第一个运动控制器，打印第二个运动控制器的卡号并关闭第二个运动控制器。

```

short main()
{
    long pos[3];
    short CardNo ;
    rtn=GT_Open();                //打开第一块控制卡
    if(0!=rtn) return rtn;
    rtn=GT_Open();                //打开第二块控制卡
    if(0!=rtn) return rtn;
    CardNo = GT_GetCurrentCardNo ();    //读取当前控制卡卡号
    printf("Card No:%d",CardNo);
    rtn=GT_SwitchtoCardNo(0);        //切换到第一块控制卡
    if(0!=rtn) return rtn;
    rtn=GT_Close();                //关闭当前控制卡
    if(0!=rtn) return rtn;

    rtn=GT_SwitchtoCardNo(1);        //切换到第二块控制卡
    if(0!=rtn) return rtn;
    CardNo = GT_GetCurrentCardNo ();    //读取当前控制卡卡号
    if(0!=rtn) return rtn;
    printf("Card No:%d",CardNo);
    rtn=GT_Close();
    if(0!=rtn) return rtn;
    return 0;
}

```

GT_GetHomeSwt

指令原型：short GT_GetHomeSwt(unsigned short *home)

指令说明：读取运动控制器 Home 开关的状态。当 Home 开关为高电平时，指令参数所对应的状态位为 1，当 Home 开关为低电平时，指令参数所对应的状态位为 0。指令示例：读取运动控制器各轴 Home 开关的状态。详细说明请参阅第九章。

指令参数：读取各轴原点开关的电平状态。

适用板卡：GE-X00-PX，GT-X00-SX。

指令示例：读取运动控制器各轴 Home 开关的状态。

```
short DisplayHomeSwitch()
{
    unsigned short home,i,bit=1;
    short rtn;
    rtn = GT_GetHomeSwt(&home);
    if(0!=rtn) return rtn;
    for(i=0;i<8;++i)
    {
        if(home&bit)
        {
            printf("\nHome%d is high",i);
        }
        else
        {
            printf("\nHome%d is low",i);
        }
        bit<<=1;
    }
    return 0
}
```

GT_GetILmt

指令原型：short GT_GetILmt(unsigned short axis, unsigned short *limit)

指令说明：读取指定控制轴的误差积分极限。

指令参数：axis 指定控制轴编号；*limit 读取指定控制轴的误差积分饱和极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetIntgr

指令原型：short GT_GetIntgr(unsigned short axis, short *integral)

指令说明：读取指定控制轴的误差积分值。

指令参数：axis 指定控制轴编号；*integral 读取指定控制轴的误差积分。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetJerk

指令原型：short GT_GetJerk(unsigned short axis, double *jerk)

指令说明：读取指定控制轴的加加速度。

指令参数：axis 指定控制轴编号；*jerk 读取指定控制轴的加加速度。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetPos。

GT_GetKaff

指令原型：short GT_GetKaff(unsigned short axis, double *kaff)

指令说明：读取指定控制轴的加速度前馈系数。

指令参数：axis 指定控制轴编号；*kaff 读取指定控制轴的加速度前馈系数。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetKd

指令原型：short GT_GetKd(unsigned short axis, double *kd)

指令说明：读取指定控制轴的误差微分增益。

指令参数：axis 指定控制轴编号；*kd 读取指定控制轴的误差微分增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetKi

指令原型：short GT_GetKi(unsigned short axis, double *ki)

指令说明：读取指定控制轴的误差积分增益。

指令参数：axis 指定控制轴编号；*ki 读取指定控制轴的误差积分增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetKp

指令原型：short GT_GetKp(unsigned short axis, double *kp)

指令说明：读取指定控制轴的误差增益。

指令参数：axis 指定控制轴编号；*kp 读取指定控制轴的误差比例增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：读取 1 轴的误差比例增益、误差积分增益、误差微分增益、速度前馈系数、加速度前馈系数、误差积分饱和极限、输出电压饱和极限。

```
short GetPid()
{
    short rtn;
    double kp,ki,kd,kaff,kvff;
    short bias;
    unsigned short motorLimit, intgrLimit,intgr,posErr;
```

```

    rtn = GT_GetKp(1,&kp);                //读取 1 轴的误差比例增益
    if(0!=rtn) return rtn;
    rtn = GT_GetKi(1,&ki);                //读取 1 轴的误差积分增益
    if(0!=rtn) return rtn;
    rtn = GT_GetILmt(1,&intgrLimit);        //读取 1 轴的误差积分饱和极限
    if(0!=rtn) return rtn;
    rtn = GT_GetKd(1,&kd);                //读取 1 轴的误差微分增益
    if(0!=rtn) return rtn;
    rtn = GT_GetKaff(1,&kaff);            //读取 1 轴的加速度前馈
    if(0!=rtn) return rtn;
    rtn = GT_GetKvff(1,&kvff);            //读取 1 轴的速度前馈
    if(0!=rtn) return rtn;
    rtn = GT_GetMtrBias(1,&bias);          //读取 1 轴的零漂电压补偿值
    if(0!=rtn) return rtn;
    rtn = GT_GetMtrLmt(1,&motorLimit);      //读取 1 轴的输出电压饱和极限
    if(0!=rtn) return rtn;
    rtn = GT_GetIntgr(1,&intgr);           //读取 1 轴的输出电压饱和极限
    if(0!=rtn) return rtn;
    rtn = GT_GetPosErr(1,&posErr);         //读取 1 轴的跟随误差极限
    if(0!=rtn) return rtn;
    printf("\n kp=%lf ki=%lf kd=%lf kvff=%lf kaff=%lf bias=%d IntgrLimit=%d
    MotorLimit=%d                                intgr=%d
    PosErr=%d",kp,ki,kd,kvff,kaff,bias,intgrLimit,motorLimit,intgr,posErr);
}

```

GT_GetKvff

指令原型：short GT_GetKvff(unsigned short axis, double *kvff)

指令说明：读取指定控制轴的速度前馈系数。

指令参数：axis 指定控制轴编号；*kvff 读取指定控制轴的速度前馈系数。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetLmtSwt

指令原型：short GT_GetLmtSwt(unsigned short *limit)

指令说明：读取运动控制器各轴的限位开关状态。当限位开关为高电平时，指令参数所对应的状态位为 1，当限位开关为低电平时，指令参数所对应的状态位为 0。详细说明请参阅第九章。

指令参数：读取各轴正负限位开关的电平状态。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：读取运动控制器各轴限位开关的电平状态。

```

short DisplayLimitSwitch()
{
    unsigned short limit,i,bit=1;

```

```
short rtn;
rtn = GT_GetLmtSwt(&limit);
if(0!=rtn) return rtn;
for(i=0;i<8;++i)
{
    if(limit&bit)
    {
        printf("\nLimit%d + is high",i);
    }
    else
    {
        printf("\nLimit%d + is low",i);
    }
    bit<<=1;
    f(limit&bit)
    {
        printf("\nLimit%d - is high",i);
    }
    else
    {
        printf("\nLimit%d - is low",i);
    }
    bit<<=1;
}
return 0;
}
```

GT_GetMAcc

指令原型：short GT_GetMAcc(unsigned short axis,double *macc)

指令说明：读取指定控制轴的最大加速度，仅用于 S 曲线加减速。

指令参数：axis 指定控制轴编号；*macc 读取指定控制轴的最大加速度。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetPos。

GT_GetMode

指令原型：short GT_GetMode(unsigned short axis,unsigned short *mode)

指令说明：读取指定控制轴的模式寄存器。模式寄存器的定义请参阅第四章。

指令参数：axis 指定控制轴编号；*mode 读取指定控制轴的模式寄存器。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetPos。

GT_GetMtnNm

指令原型：short GT_GetMtnNm(unsigned short *number)

指令说明：本指令返回当前缓冲区中正在执行的轨迹段编号。当主机使用 GT_EStpMtn 和 GT_StpMtn 指令中断缓冲区命令执行状态后，调用本指令则返回中断处的轨迹段编号。缓冲区中，轨迹描述命令可产生有效段号的累加，轨迹参数命令（如 GT_SetSynVel，GT_SetDccVel 等）不产生段号的变化。段号范围为：[1,32767]，当执行段号超过 32767 时，运动控制器从 1 开始重新累加段号。

指令参数：*number 返回轨迹段编号。

适用板卡：GE-X00-SX。

GT_GetMtrBias

指令原型：short GT_GetMtrBias(unsigned short axis,short *bias)

指令说明：读取指定控制轴的零漂电压补偿。

指令参数：axis 指定控制轴编号；*bias 读取指定控制轴的零漂电压补偿值。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetMtrCmd

指令原型：short GT_GetMtrCmd(unsigned short axis,short *voltage)

指令说明：读取指定控制轴在直接电压输出模式下的输出电压值。

指令参数：axis 指定控制轴编号；*voltage 读取指定控制轴的输出电压值。

适用板卡：GE-X00-PV。

GT_GetMtrLmt

指令原型：short GT_GetMtrLmt(unsigned short axis, unsigned short *limit)

指令说明：读取指定控制轴的输出电压饱和极限。

指令参数：axis 指定控制轴编号；*limit 读取指定控制轴的输出电压饱和极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetPos

指令原型：short GT_GetPos(unsigned short axis,long *pos)

指令说明：读取指定控制轴的目标位置。

指令参数：axis 指定控制轴编号；*pos 读取指定控制轴的目标位置。

适用板卡：GE-X00-PX。

指令示例：读取 1 轴的目标位置、目标速度、加速度、加加速度、最大加速度、控制轴状态、控制轴模式。

short GetMotionParameter()

```
{  
    short rtn;  
    long pos;  
    double vel,acc,macc,jerk;  
    unsigned long status;  
    unsigned short mode;
```



```

    rtn = GT_GetPos(1,&pos);
    if(0!=rtn) return rtn;
    rtn = GT_GetVel(1,&vel);
    if(0!=rtn) return rtn;
    rtn = GT_GetAcc(1,&acc);
    if(0!=rtn) return rtn;
    rtn = GT_GetMAcc(1,&macc);
    if(0!=rtn) return rtn;
    rtn = GT_GetJerk(1,&jerk);
    if(0!=rtn) return rtn;
    rtn = GT_GetSts(1,&status);
    if(0!=rtn) return rtn;
    rtn = GT_GetMode(1,&mode);
    if(0!=rtn) return rtn;
    printf("\n Pos = %ld Vel = %lf Acc = %lf MAcc = %lf Jerk = %lf Status = %lx
    Mode = %x",
    pos, vel, acc, macc, jerk, status, mode);
    return 0;
}

```

GT_GetPosErr

指令原型：short GT_GetPosErr(unsigned short axis,unsigned short *error)

指令说明：读取指定控制轴的跟随误差极限。

指令参数：axis 指定控制轴编号；*error 读取指定控制轴的跟随误差极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_GetKp。

GT_GetPrfPnt

指令原型：short GT_GetPrfPnt(double *point)

指令说明：该指令获得轨迹运动中各轴的规划位置值。

指令参数：双精度型*point 作为指针，指向一个长度为 3 的数组，按 X、Y、Z 轴的顺序返回各轴的当前规划位置值，即各轴的内部计数值。

用户调用该指令时，应自行定义一个长度为 3 的双精度整型数组，并将数组首地址作为该指令的实参。

适用板卡：GE-X00-SX。

指令示例：举例——获得轨迹运动中各轴的规划位置值，并打印。

```

short GetPrfPnt()
{
    short rtn;
    double crd_prf_pos[3];
    rtn=GT_GetPrfPnt(crd_prf_pos);
    if(0!=rtn) return rtn;
    printf("the coordinate pos are: %ld, %ld, %ld \n", crd_prf_pos[0],

```

```
    crd_prf_pos[1], crd_prf_pos[2]);  
    return 0;  
}
```

GT_GetPrfPos

指令原型：short GT_GetPrfPos(unsigned short axis, long *pos)

指令说明：读取指定控制轴的规划位置。

指令参数：axis 指定控制轴编号；*pos 读取指定控制轴的规划位置。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetAtlPos。

GT_GetPrfVel

指令原型：short GT_GetPrfVel(unsigned short axis, double *vel)

指令说明：读取指定控制轴的规划速度。

指令参数：axis 指定控制轴编号；*vel 读取指定控制轴的规划速度。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetAtlVel。

GT_GetPrfVel

指令原型：short GT_GetPrfVel(double *vel)

指令说明：该指令返回轨迹运动的当前进给速度。

指令参数：*vel 返回当前进给速度。

适用板卡：GE-X00-SX。

GT_GetSegPnt

指令原型：short GT_GetSegPnt(double *point)

指令说明：该指令读取所有轴的目标位置，在进行轨迹运动时，可调用该指令读取当前执行轨迹段的终点位置。该指令读回三个轴的终点位置。

指令参数：*point 返回目标位置值。

适用板卡：GE-X00-SX。

GT_GetSts

指令原型：short GT_GetSts(unsigned short axis, unsigned long *status)

指令说明：读取指定控制轴的状态寄存器。状态寄存器的定义请参阅第四章。

指令参数：axis 指定控制轴编号；*status 读取指定控制轴的状态寄存器。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_GetPos。

GT_GetVel

指令原型：short GT_GetVel(unsigned short axis, double *vel)

指令说明：读取指定控制轴的目标速度。

指令参数：axis 指定控制轴编号；*vel 读取指定控制轴的目标速度。

适用板卡：GE-X00-PX。

指令示例：参阅 GT_GetPos。

GT_HardRst

指令原型：short GT_HardRst(void)

指令说明：调用该指令以后将复位运动控制器，并重新加载 DSP 程序。调用该指令后运动控制器寄存器的状态如下：

1. 所有运动参数寄存器设置为 0
2. 所有位置捕获寄存器设置为 0
3. 各轴为梯形曲线加减速模式
4. 所有 PID 参数设置为 0 (GE-X00-XV)
5. 所有控制轴的误差积分极限为 32767 (GE-X00-XV)
6. 所有控制轴的输出电压饱和极限为 32767 (GE-X00-XV)
7. 所有控制轴的跟随误差极限为 32767 (GE-X00-XV)
8. 禁止跟随误差超限自动停止 (GE-X00-PV)
9. 所有控制轴工作在闭环控制模式 (GE-X00-XV)
10. 所有控制轴的输出控制量为默认值

指令参数：无。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：当通用数字 I/O 的 EXI0 为高电平时硬件复位运动控制器。

```
short HardReset()
{
    short rtn;
    unsigned short exInpt;
    rtn = GT_ExInpt(&exInpt);
    if(0!=rtn) return rtn;
    if(exInpt&1)
    {
        return GT_HardRst();
    }
}
```

GT_HomeSns

指令原型：short GT_HomeSns(unsigned short sense)

指令说明：设置运动控制器各轴 Home 信号的触发沿。详细内容请参阅第三章。

指令参数：设置所有控制轴 Home 信号的触发沿。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：当通用数字 I/O 的 EXI0 为高电平时将 1 轴的 Home 捕获设置为上升沿触发，为低电平时将 1 轴的 Home 捕获设置为下降沿触发。

```
short HomeConfig()
{
    short rtn;
    unsigned short exInpt;
    rtn = GT_ExInpt(&exInpt);
```

```

if(0!=rtn) return rtn;
if(exInpt&1)
{
    return GT_HomeSns(1);
}
else
{
    return GT_HomeSns(0);
}
}

```

GT_HandWheel

指令原型：short GT_HandWheel(unsigned short axis, double ratio)

指令说明：手脉功能。GE-300-SX 运动控制器提供一个手脉输入接口，可以接收单端和差动的手脉或辅助编码器信号。

当系统运动停止时，用户调用该指令将指定控制轴设置为手脉输出的方式，并设定好跟随倍率后，若用户转动手脉，指定控制轴会跟随手脉运动。

一旦进入手脉输出工作方式，系统不接收任何运动命令和启动缓冲区等命令——除非调用解除手脉输出方式的指令。当某个控制轴工作在手脉控制模式下时，若经过一段时间（2s）的采样发现手脉位置没有发生变化，即认为手脉运动停止，可以改变手脉的跟随轴和跟随倍率。

指令参数：axis 指定要跟随手脉运动的轴号，ratio 设定跟随轴的倍率。

适用板卡：GE-X00-SX。

指令示例：参阅例程 10-5。

GT_HookCommand

指令原型：short GT_HookCommand(TCommandHandle CommandHandle)

指令说明：挂接指令处理程序。详细内容请参阅第二章。

指令参数：CommandHandle 是所挂接的指令处理程序的地址，TCommandHandle 的定义为：

```
typedef void (__stdcall * TCommandHandle)(char * command, short error);
```

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 2-2、2-3、2-4。

GT_IndexSns

指令原型：short GT_IndexSns(unsigned short sense)

指令说明：设置运动控制器各轴编码器 C 相信号或者光栅尺 Index 信号的触发沿。

指令参数：设置各控制轴编码器 C 相信号或者光栅尺 Index 信号的触发沿。

适用板卡：GE-X00-PX。

GT_InitLookAhead

指令原型：short GT_InitLookAhead(double t, double acc_max, double acc, double vel, int n, double con)

指令说明：初始化预处理功能模块的各项参数。

指令参数：参数说明——

t：拐弯时间。单位：s，用户在调用预处理功能指令时注意其时间单位的统一，即速度和加速度的时间单位均为 S，求出的终点速度也以秒为时间单位。

t 的经验范围：0.001~0.01S。

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的偏低。因此要合理选择 T 值。

acc_max：系统能承受的最大加速度，单位：mm/s²。根据不同的机械系统和电机驱动器取值不同，经验范围：100~10000 mm/s²。

acc：系统设定加速度，单位：mm/s²。

vel：用户加工的目标速度，单位：mm/s。

n：需预处理的段数。

con：脉冲数与标准长度单位对应值，单位：pulse/mm。

适用板卡：GE-X00-SX。

GT_LmtSns

指令原型：short GT_LmtSns(unsigned short sense)

指令说明：设置运动控制器各轴限位开关的触发电平。当指令参数某个状态位为 1 时，所对应的限位开关为低电平触发；当指令参数某个状态位为 0 时，所对应的限位开关为高电平触发。默认情况下各轴限位开关为高电平触发。

指令参数：设置运动控制器各轴限位开关的触发电平。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 3-2、3-3、3-4。

GT_LmtsOff

指令原型：short GT_LmtsOff(unsigned short axis)

指令说明：将指定控制轴的正负限位开关设置为无效。调用该指令以后，即使指定控制轴的限位开关触发，运动控制器也不会停止越限轴的运动。但是仍然可以调用 GT_GetLmtSwt 指令读取限位开关的状态。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_LmtsOn。

GT_LmtsOn

指令原型：short GT_LmtsOn(unsigned short axis)

指令说明：将指定控制轴的正负限位开关设置为有效。默认情况下，所有控制轴的正负限位开关有效。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：将 1 轴的正负限位开关设置为无效，2 轴的正负限位开关设置为有效。

```
short LimitConfig()
```

```
{  
    short rtn;  
    rtn = GT_LmtsOff(1);  
    if(0!=rtn) return rtn;  
    rtn = GT_LmtsOn(2);  
    if(0!=rtn) return rtn;  
    return GT_ClrSts(1);  
}
```

GT_LnXY

指令原型：short GT_LnXY(double x, double y)

指令说明：该指令实现二维直线插补运动。

参数 x、y 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

适用板卡：GE-X00-SX。

指令示例：二维直线插补命令。

```
short LineXY()
```

```
{  
    short rtn;  
    rtn=GT_SetSynVel(3);  
    if(0==rtn) retrun rtn;  
    rtn=GT_SetSynAcc(1);  
    if(0==rtn) retrun rtn;  
    rtn=GT_LnXY(12345,67890);  
    if(0==rtn) retrun rtn;  
    return 0;  
}
```

GT_LnXYZ

指令原型：short GT_LnXYZ(double x, double y, double z)

指令说明：该指令实现三维直线插补运动。参数 x、y、z 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

适用板卡：GE-X00-SX。

指令示例：缓冲区中的三维直线插补命令。

```
short LineXYZ()
```

```
{
```

```

short rtn;
rtn=GT_StrtList();
if(0==rtn) retrun rtn;
rtn=GT_MvXYZ(0,0,0,6,0.3);
if(0==rtn) retrun rtn;
rtn=GT_LnXYZ(12945,58372,65473);
if(0==rtn) retrun rtn;
rtn=GT_StrtMtn();
if(0==rtn) retrun rtn;
...
}

```

GT_LnXYZA

指令原型：short GT_LnXYZA(double x, double y, double z, double a)

指令说明：该指令实现四维直线插补运动。参数 x、y、z、a 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。

适用板卡：GE-X00-SX。

指令示例：缓冲区中的三维直线插补命令。

```

short LineXYZA()
{
    short rtn;
    rtn=GT_StrtList();
    if(0==rtn) retrun rtn;
    rtn=GT_MvXYZA(0,0,0,6,0.3,0.5);
    if(0==rtn) retrun rtn;
    rtn=GT_LnXYZA(12945,58372,65473,65473);
    if(0==rtn) retrun rtn;
    rtn=GT_StrtMtn();
    if(0==rtn) retrun rtn;
    ...
}

```

GT_LnXYG0

指令原型：short GT_LnXYG0(double x, double y)

指令说明：该指令实现二维直线插补运动。

参数 x、y 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。用户在使用速度倍率时，GT_OverrideG0 影响该指令产生的轨迹运动的速度。

适用板卡：GE-X00-SX。

指令示例：二维直线插补命令。

```
short LineXYG0()
{
    short rtn;
    rtn=GT_SetSynVel(30);
    if(0==rtn) retrun rtn;
    rtn=GT_SetSynAcc(1);
    if(0==rtn) retrun rtn;
    rtn=GT_LnXYG0(12345,67890);
    if(0==rtn) retrun rtn;
    return 0;
}
```

GT_LnXYZG0

指令原型：short GT_LnXYZG0(double x, double y, double z)

指令说明：该指令实现三维直线插补运动。参数 x、y、z 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。用户在使用速度倍率时，GT_OverrideG0 影响该指令产生的轨迹运动的速度。

适用板卡：GE-X00-SX。

指令示例：缓冲区中三维直线插补命令。

```
short LineXYZG0()
{
    short rtn;
    rtn=GT_StrtList();
    if(0==rtn) retrun rtn;
    rtn=GT_MvXYZ(0,0,0,6,0.3);
    if(0==rtn) retrun rtn;
    rtn=GT_LnXYZG0(12945,58372,65473);
    if(0==rtn) retrun rtn;
    rtn=GT_StrtMtn();
    if(0==rtn) retrun rtn;
    ...
}
```

GT_LnXYZAG0

指令原型：short GT_LnXYZG0(double x, double y, double z, double a)

指令说明：该指令实现四维直线插补运动。参数 x、y、z、a 分别为相应控制轴的终点位置，单位是 pulse。直线插补运动的起点位置是前一段轨迹描述的终点位置（缓冲区命令）或者是当前位置（立即命令）；如果是缓冲区第一条运动轨迹段，则起点是定位指令指定的位置。用户在使用速

度倍率时，GT_OverrideG0 影响该指令产生的轨迹运动的速度。

适用板卡：GE-X00-SX。

指令示例：缓冲区中三维直线插补命令。

```
short LineXYZAG0()
{
    short rtn;
    rtn=GT_StrtList();
    if(0==rtn) retrun rtn;
    rtn=GT_MvXYZA(0,0,0,6,0.3,0.5);
    if(0==rtn) retrun rtn;
    rtn=GT_LnXYZAG0(12945,58372,65473,65473);
    if(0==rtn) retrun rtn;
    rtn=GT_StrtMtn();
    if(0==rtn) retrun rtn;
    ...
}
```

GT_MapCnt

指令原型：short GT_MapCnt(unsigned short axis, double count)

指令说明：设置坐标偏移量。调用该指令专门管理坐标偏移量，建立区别于机床坐标系的加工坐标系。

指令参数：axis 指定控制轴编号，count 设定要偏移的脉冲数。

适用板卡：GE-X00-SX。

GT_MltiUpdt

指令原型：short GT_MltiUpdt(unsigned short mask)

指令说明：同时更新多个控制轴的参数。当指令参数某个状态位为 1 时，所对应控制轴的参数将被更新。指令参数状态位和控制轴的对应关系如下所示：

状态位	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
控制轴	8	7	6	5	4	3	2	1

指令参数：控制轴刷新参数。

适用板卡：GE-X00-PX。

指令示例：同时更新 1 轴和 3 轴的参数。

```
void main()
{
    GT_HookCommand(CommandHandle);
    GT_Open();                //打开运动控制器
    GT_Reset();               //复位运动控制器
    GT_SetKp(1,10);           //设置 1 轴比例增益
}
```

```

GT_SetKp(3,10);           //设置 3 轴比例增益
GT_MltiUpdt(5);           //同时更新 1 轴和 3 轴
GT_AxisOn(1);             //1 轴伺服使能
GT_AxisOn(3);             //3 轴伺服使能
printf("\nPress any key to continue..."); //按任意键继续
getch();
GT_PrflV(1);              //设置 1 轴运动参数
GT_SetVel(1,10);
GT_SetAcc(1,0.1);
GT_PrflV(3);              //设置 3 轴运动参数
GT_SetVel(3,10);
GT_SetAcc(3,0.1);
GT_MltiUpdt(5);           //同时更新 1 轴和 3 轴
}

```

GT_MvXY

指令原型：short GT_MvXY(double x, double y, double vel, double acc)

指令说明：对于二维坐标系，该指令必须作为指令 GT_StrtList 后的第一条缓冲区命令，放在命令缓冲区中用来定义二维连续轨迹的缓冲区命令的起点位置。

参数 x、y 是起点位置，单位是 pulse；参数 vel 设定进给速度，单位是 pulse/ms；参数 acc 设定进给加速度，单位是 pulse/ms²。此后调用 GT_StrtMtn 指令时，运动控制器控制 X、Y 轴（1、2 轴）从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

适用板卡：GE-X00-SX。

相关指令：GT_MvXYZ

指令示例：该指令只能在打开缓冲区，还没有传送轨迹描述和参数命令的情况下调用。一旦缓冲区打开时调用了该指令，缓冲区中以后的轨迹描述命令只能是二维轨迹命令。

举例——在缓冲区打开状态下，将缓冲区运动的起点定位在 X 轴 1000 个 pulse，Y 轴 2000 个 pulse 的位置，并设速度为 100pulse/ms，加速度为 1.8pulse/ms² 移动到定位点。

```

short MoveXY()
{
    short rtn;
    rtn=GT_StrtList();
    if(0==rtn) retrun rtn;
    rtn=GT_MvXY (1000,2000,100,1.8);
    if(0==rtn) retrun rtn;
    ...
    rtn=GT_StrtMtn();
    if(0==rtn) retrun rtn;
    return 0;
}

```

}

GT_MvXYZ

指令原型：short GT_MvXYZ (double x, double y, double z, double vel, double acc)

指令说明：对于三维坐标系，该指令必须作为指令 GT_StrtList 后的第一条缓冲区命令，放在命令缓冲区中用来定义三维连续轨迹的缓冲区命令的起点位置。

参数 x、y、z 是起点位置，单位是 pulse；参数 vel 设定进给速度，单位是 pulse/ms；参数 acc 设定进给加速度，单位是 pulse/ms²。此后调用 GT_StrtMtn 指令时，运动控制器控制 X、Y、Z 轴（1、2、3 轴）从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

适用板卡：GE-X00-SX。

相关指令：GT_MvXY

指令示例：参阅 GT_MvXY。

GT_MvXYZA

指令原型：short GT_MvXYZA (double x, double y, double z, double a, double vel, double acc)

指令说明：对于四维坐标系，该指令必须作为指令 GT_StrtList 后的第一条缓冲区命令，放在命令缓冲区中用来定义四维连续轨迹的缓冲区命令的起点位置。

参数 x、y、z、a 是起点位置，单位是 pulse；参数 vel 设定进给速度，单位是 pulse/ms；参数 acc 设定进给加速度，单位是 pulse/ms²。此后调用 GT_StrtMtn 指令时，运动控制器控制 X、Y、Z、A 轴（1、2、3、4 轴）从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

适用板卡：GE-X00-SX。

相关指令：GT_MvXY

指令示例：参阅 GT_MvXY。

GT_Open

指令原型：short GT_Open(unsigned long address=65535)

指令说明：调用 GT_Open 指令打开运动控制器，以获取对运动控制器的访问权。对于 ISA 总线的运动控制器，需要指定运动控制器的基地址。对于 PCI 总线的运动控制器，不需要指定运动控制器的基地址，只需将默认参数 65535 传递给函数库即可。

指令参数：ISA 总线的运动控制器需要指定基地址；PCI 总线的运动控制器不用指定基地址。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 5-1。

GT_OpenLp

指令原型：short GT_OpenLp(unsigned short axis)

指令说明：将指定控制轴设置为直接电压输出模式。在该模式下可以调用 GT_SetMtrCmd 指令直接设置指定控制轴的输出电压。输出电压可以按照下表进行计算。

输出 电压 值(V)	所 对 应 的 数 值
+ 10	32767
0	0
- 10	-32768

指令参数：指定控制轴编号。

适用板卡：GE-X00-PV。

指令示例：参阅 GT_SetMtrCmd。

GT_Override

指令原型：short GT_Override(double override)

指令说明：设置除 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 以及定位指令以外的轨迹描述指令所产生运动的速度倍率。

调用该指令可以在线改变后续轨迹运动的进给速度，使设定速度改变为原设定速度的 override 倍，且一直有效，直到再次调用该指令。

指令参数：override 是设置的速度倍率。

适用板卡：GE-X00-SX。

GT_OverrideG0

指令原型：short GT_OverrideG0(double override)

指令说明：设置 GT_LnXYG0、GT_LnXYZG0、GT_LnXYZAG0 以及定位指令的速度倍率。调用该指令后，使原设定速度改变为原设定速度的 override 倍，且一直有效，直到再次调用该指令。

指令参数：override 是设置的速度倍率。

适用板卡：GE-X00-SX。

GT_PrflS

指令原型：short GT_PrflS(unsigned short axis)

指令说明：将指定控制轴设置为 S 曲线加减速。只能在指定控制轴静止时执行该指令，否则该指令无效，并返回 1。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-2。

GT_PrflT

指令原型：short GT_PrflT(unsigned short axis)

指令说明：将指定控制轴设置为梯形曲线加减速。只能在指定控制轴静止时执行该指令，否则该指令无效，并返回 1。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-1。

GT_PrflV

指令原型：short GT_PrflV(unsigned short axis)

指令说明：将指定控制轴设置为速度追踪模式。只能在指定控制轴静止时执行该指令，否则该指令无效，并返回 1。

指令参数：指定控制轴编号。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-3。

GT_Reset

指令原型：short GT_Reset(void)

指令说明：复位运动控制器。调用该指令以后，各寄存器的状态和调用 GT_HardRst 指令相同，但是 GT_Reset 指令不会重新装载 DSP 程序。该指令一般用于运动控制器初始化。

指令参数：无。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 5-1。

GT_RestoreMtn

指令原型：short GT_RestoreMtn(void)

指令说明：在缓冲区运动暂停后，使用此命令可使中断的运动从断点处继续运动。
注意：调用该指令之前，确保已经发送 GT_StrtMtn，否则电机可能出现不正确运动。

适用板卡：GE-X00-SX。

GT_RstSts

指令原型：short GT_RstSts(unsigned short axis, unsigned short mask)

指令说明：将指定控制轴状态寄存器的指定状态位清 0。指令参数的状态位和状态寄存器定义相同。当指令参数的某个状态位为 1 时，保留状态寄存器的相应位；当指令参数的某个状态位为 0 时，将状态寄存器的相应位清 0。该指令只能清除状态寄存器的 bit0~bit7。

指令参数：axis 指定控制轴编号；mask 指定清除指定控制轴状态寄存器的哪些状态位。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_SetAtlPos。

GT_SetAcc

指令原型：short GT_SetAcc(unsigned short axis, double acc)

指令说明：设置指定控制轴的加速度。参数取值范围是[0，102400]。该指令仅对速度追踪模式和梯形曲线加减速有效。指令参数的单位是“ pulse/ms² ”，

换算方法请参阅第五章。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；acc 设置指定控制轴的加速度。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-1、5-3。

GT_SetAdcChn

指令原型：short GT_SetAdcChn(unsigned short channel)

指令说明：设置 A/D 通道数目，用来提高 A/D 的采样频率。当只有一个 A/D 通道时，采样频率可以提高到 $8 \times 770\text{Hz} = 6.16\text{KHz}$ 。

指令参数：通道数。

适用板卡：GE-X00-SX。

GT_SetAtlPos

指令原型：short GT_SetAtlPos(unsigned short axis, long pos)

指令说明：将指定控制轴的实际位置、目标位置和规划位置设置为指定值。参数取值范围是 $[-2^{30}, 2^{30} - 1]$ 。只能在指定控制轴静止时执行该指令，否则该指令无效，并返回 1。

指令参数：axis 指定控制轴编号；pos 设置指定控制轴的实际位置。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：1 轴运动停止以后，将该轴的实际位置设置为 0，并且清除该轴状态寄存器的 bit0。

```
void main()
{
    GT_HookCommand(CommandHandle);
    GT_Open();
    GT_Reset();
    AxisInitial(1,0); //引用例程 3-2。
    AxisRunT(1,10000,1,0.1); //引用例程 5-1。
    unsigned long status;
    do
    {
        GT_GetSts(1,&status); //读取 1 轴的控制轴状态寄存器
    } while(status & 0x400);
    GT_SetAtlPos(1,0);
    GT_RstSts(1,0xfe);
}
```

GT_SetDccVel

指令原型：short GT_SetDccVel(double vel)

指令说明：设定线段的终点位置，单位是 pulse/ms。

该指令在缓冲区下确定轨迹段的终点速度，运动控制器将按照该指令设定的速度进行加减速控制。

指令参数：vel 设定轨迹段终点速度。

适用板卡：GE-X00-SX。

GT_SetILmt

指令原型：short GT_SetILmt(unsigned short axis,unsigned short limit)

指令说明：设置指定控制轴的误差积分饱和极限。参数取值范围是 $[0, 2^{15} - 1]$ 。
第一次设置误差积分增益 Ki 时要小心。如果系统正在运行而积分项的值比较大，把 Ki 设置为非 0 值将引起电机突然跳跃。为了避免这种情况，应当首先将误差积分饱和极限设置为 0，这样就能够将误差积分项清 0，保证电机平稳运动。然后再将误差积分增益 Ki 和误差积分极限设置为期望值。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；limit 设置指定控制轴的误差积分饱和极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetJerk

指令原型：short GT_SetJerk(unsigned short axis,double jerk)

指令说明：设置指定控制轴的加加速度。参数取值范围是 $[0, 256000]$ 。该指令仅对 S 曲线加减速有效。指令参数的单位是 pulse/ms³，换算方法请参阅第五章。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；jerk 设置指定控制轴的加加速度。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-2。

GT_SetKaff

指令原型：short GT_SetKaff(unsigned short axis,double kaff)

指令说明：设置指定控制轴的加速度前馈系数。参数取值范围是 $[0, 2^{15}-1]$ 。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；kaff 设置指定控制轴的加速度前馈系数。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetKd

指令原型：short GT_SetKd(unsigned short axis,double kd)

指令说明：设置指定控制轴的误差微分增益。参数取值范围是 $[0, 2^{15}-1]$ 。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；kd 设置指定控制轴的误差微分增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetKi

指令原型：short GT_SetKi(unsigned short axis,double ki)

指令说明：设置指定控制轴的误差积分增益。参数取值范围是 $[0, 2^{15}-1]$ 。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；ki 设置指定控制轴的误差积分增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetKp

指令原型：short GT_SetKp(unsigned short axis,double kp)

指令说明：设置指定控制轴的误差比例增益。参数取值范围是 $[0, 2^{15}-1]$ 。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；kp 设置指定控制轴的误差比例增益。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：设置 1 轴的误差比例增益、误差积分增益、误差微分增益、加速度前馈系数、速度前馈系数、零漂电压补偿值、输出电压饱和极限、误差积分饱和极限。

```
short PidConfig()
{
    short rtn;
    rtn = GT_SetKp(1,10);                //设置 1 轴的误差比例增益
    if(0!=rtn) return rtn;
    rtn = GT_SetKi(1,20);                //设置 1 轴的误差积分增益
    if(0!=rtn) return rtn;
    rtn = GT_SetILmt(1,20000);           //设置 1 轴的误差积分饱和极
    限
    if(0!=rtn) return rtn;
    rtn = GT_SetKd(1,5);                 //设置 1 轴的误差微分增益
    if(0!=rtn) return rtn;
    rtn = GT_SetKaff(1,10);              //设置 1 轴的加速度前馈
    if(0!=rtn) return rtn;
    rtn = GT_SetKvff(1,10);              //设置 1 轴的速度前馈
    if(0!=rtn) return rtn;
    rtn = GT_SetMtrBias(1,-200);          //设置 1 轴的零漂电压补偿值
    if(0!=rtn) return rtn;
    rtn = GT_SetMtrLmt(1,6*3276);        //设置 1 轴的输出电压饱和极
    限
    if(0!=rtn) return rtn;
    rtn = GT_SetPosErr(1,32767);         //设置 1 轴的跟随误差极限
    if(0!=rtn) return rtn;
    return GT_Update(1);                 //使 1 轴的控制参数生效
}
```

GT_SetKvff

指令原型：short GT_SetKvff(unsigned short axis,double kvff)

指令说明：设置指定控制轴的速度前馈系数。参数取值范围是 $[0, 2^{15}-1]$ 。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；kvff 设置指定控制轴的速度前馈系数。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetMAcc

指令原型：short GT_SetMAcc(unsigned short axis, double macc)

指令说明：设置指定控制轴的最大加速度。参数取值范围是[0, 102400]。该指令仅对 S 曲线加减速有效。指令参数的单位是“pulse/ms²”，换算方法请参阅第五章。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；macc 设置指定控制轴的最大加速度。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-2。

GT_SetMaxVel

指令原型：short GT_SetMaxVel (double vel)

指令说明：该指令设定系统最高运行速度，当用户设定速度大于此速度时按此速度运行。单位：pulse/ms。

指令参数：设定控制卡最高运行速度。

适用板卡：GE-X00-SX。

GT_SetMtrBias

指令原型：short GT_SetMtrBias(unsigned short axis, short bias)

指令说明：设置指定控制轴的零漂电压补偿。参数取值范围是[-32768, 32767]。因为运动控制器各控制轴可能存在不同的零漂电压，所以必须正确设置零漂电压补偿，才能保证运动控制器的准确定位。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；bias 设置指定控制轴的零漂电压补偿值。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetMtrCmd

指令原型：short GT_SetMtrCmd(unsigned short axis, short voltage)

指令说明：在直接电压输出模式下，设置指定控制轴的输出电压。参数取值范围是[-32768, 32767]。输出电压值与 GT_SetMtrCmd 指令参数的对应关系参阅 GT_OpenLp。

指令参数：axis 指定控制轴编号；voltage 设置指定控制轴的输出电压。

适用板卡：GE-X00-PV。

指令示例：设置控制轴 1 输出 3V 模拟电压。

short OutputVoltage()

```
{  
    short rtn;  
    rtn = GT_OpenLp(1);           //将控制轴 1 切换到直接电压  
    输出模
```

```
                                //式
    if(0!=rtn) return rtn;
    return GT_SetMtrCmd(1, 3*3276);           //控制轴 1 输出 3V 模拟电压
}
```

GT_SetMtrLmt

指令原型：short GT_SetMtrLmt(unsigned short axis, unsigned short limit)

指令说明：设置指定控制轴输出电压饱和极限。参数取值范围是[0, 32767]。指定控制轴默认电压输出范围是 $\pm 10V$ 。在某些场合下，需要对输出电压进行限制，这时可以调用 GT_SetMtrLmt 指令设定输出电压饱和值。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；limit 设置指定控制轴的输出电压饱和极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetPos

指令原型：short GT_SetPos(unsigned short axis, long pos)

指令说明：设置指定控制轴的目标位置。参数取值范围是[-1073741824, 1073741823]。指令参数的单位是 pulse，换算方法请参阅第五章。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；pos 设置指定控制轴的目标位置。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-1、5-2、5-3。

GT_SetPosErr

指令原型：short GT_SetPosErr(unsigned short axis, unsigned short error)

指令说明：设置指定控制轴的跟随误差极限。参数取值范围是[0, 32767]。各控制轴默认的跟随误差极限为 32767，调用 GT_SetPosErr 指令可以修改跟随误差极限。该指令为刷新生效指令。详细内容参阅第八章。

指令参数：axis 指定控制轴编号；error 设置指定控制轴的跟随误差极限。

适用板卡：GE-X00-PV，GE-X00-SV。

指令示例：参阅 GT_SetKp。

GT_SetSpindleVel

指令原型：short GT_SetSpindleVel(short vel)

指令说明：设定主轴转速并开始主轴运动。

指令参数：vel：主轴速度，16 位模拟量输出值，对应-10V ~ +10V，取值范围 [-32768,32767]。

适用板卡：GE-X00-SV。

指令示例：参阅例程 10-6。

GT_SetStpAcc

指令原型：short GT_SetStpAcc (double acc)

指令说明：设定系统的停止加速度。

该指令用来设定系统紧急停止时的加速度，例如碰到限位开关。一般情况下取正常加速度的 2 - 3 倍，单位：pulse/ms²。

指令参数：acc 停止加速度值。

适用板卡：GE-X00-SX。

GT_SetStrtVel

指令原型：short GT_SetStrtVel (double vel)

指令说明：设定连续轨迹运动的启动速度。

该指令用来设定在连续轨迹运动的启动速度，启动速度与电机特性和机械系统特性有直接关系，请用户根据实际系统进行设置。运动控制器上电默认的启动速度为 500HZ，单位：pulse/ms。

指令参数：vel 启动速度值。

适用板卡：GE-X00-SX。

GT_SetSynAcc

指令原型：short GT_SetSynAcc(double acc)

指令说明：该指令设置连续轨迹运动的进给加速度。

指令参数：acc 是设定的进给加速度值。其单位是 pulse/ms²。该指令在立即指令输入方式下有效，且决定此后所有立即轨迹运动的加速度，直到再次调用该指令为止。

适用板卡：GE-X00-SX。

GT_SetSynVel

指令原型：short GT_SetSynVel(double vel)

指令说明：该指令设置轨迹段的进给速度。

指令参数：vel 是设定的轨迹运动的进给速度值。其单位是 pulse/ms。该指令影响此后调用的所有直线插补和圆弧插补轨迹描述指令的进给速度，直到再次调用该指令为止。

适用板卡：GE-X00-SX。

GT_SetVel

指令原型：short GT_SetVel(unsigned short axis, double vel)

指令说明：设置指定控制轴的目标速度。速度控制模式下参数取值范围是[-40958, 40957]，梯形曲线加减速和 S 曲线加减速模式下参数取值范围是[0, 40957]。该指令为刷新生效指令。

指令参数：axis 指定控制轴编号；vel 设置指定控制轴的目标速度。

适用板卡：GE-X00-PX。

指令示例：参阅例程 5-1、5-2、5-3。

GT_SmthStp

指令原型：short GT_SmthStp(unsigned short axis)

指令说明：根据指定控制轴的加减速模式平滑减速到 0。详细内容参阅第五章。

该指令为刷新生效指令。

指令参数：参数是指定控制轴编号。

适用板卡：GE-X00-PX。

指令示例：参阅例程 7-1、7-2。

GT_StepDir

指令原型：short GT_StepDir(unsigned short axis)

指令说明：将指定控制轴的脉冲输出方式设置为“脉冲 + 方向”。

指令参数：参数是指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：当通用数字 I/O 的 EXI0 为高电平时将 1 轴的输出脉冲设置为“脉冲 + 方向”，为低电平时将 1 轴的输出脉冲设置为“正负脉冲”。

```
short PulseConfig()
{
    short rtn;
    unsigned short exInpt;
    rtn = GT_ExInpt(&exInpt);           //读取通用数字 I/O 的输入
    if(0!=rtn) return rtn;
    if(exInpt&1)
    {
        return GT_StepDir(1);           //将 1 轴的输出脉冲设置为
                                         // “脉冲 + 方向”
    }
    else
    {
        return GT_StepPulse(1);         //将 1 轴的输出脉冲设置为
                                         // “正负脉冲”
    }
}
```

GT_StepPulse

指令原型：short GT_StepPulse(unsigned short axis)

指令说明：将指定控制轴的脉冲输出方式设置为“正负脉冲”。

指令参数：参数是指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅 GT_StepDir。

GT_StpMtn

指令原型：short GT_StpMtn (void)

指令说明：该指令按照设定的进给加速度停止轨迹运动。若处于缓冲区连续轨迹运动状态，该指令将关闭缓冲区，并在运动停止后保存断点信息，以便此后调用 GT_RestoreMtn 指令继续缓冲区连续轨迹运动。

用户若在执行定位指令时发出停止指令停止运动，可能无法恢复正确的缓冲区连续轨迹运动。

适用板卡：GE-X00-SX。

GT_StrtList

指令原型：short GT_StrtList (void)

指令说明：该指令打开并清空缓冲区。已经处于缓冲区命令输入状态时，该指令无效。

适用板卡：GE-X00-SX。

指令示例：请参阅 GT_LnXYZ。

GT_StrtMtn

指令原型：short GT_StrtMtn (void)

指令说明：该指令启动缓冲区连续轨迹运动。运动控制器顺序执行存储在缓冲区中的轨迹运动描述指令。

适用板卡：GE-X00-SX。

指令示例：请参阅 GT_LnXYZ。

GT_SwitchtoCardNo

指令原型：short GT_SwitchtoCardNo (unsigned short number)

指令说明：该指令用于切换当前运动控制器卡号。一台 PC 机上使用多个运动控制器时，本指令用于指定当前运动控制器。当指令执行成功后，后随的所有指令只操作当前运动控制器。在多运动控制器系统中，每个运动控制器在操作系统启动时被分配一个卡号（0 - 15），该卡号在系统重新启动之前一直有效，用于区别不同控制卡。卡号分配原则遵循 PNP 规则，第一个被系统识别的运动控制器卡号为 0，所以在硬件配置没有改变的情况下，系统每次分配的卡号是相同的。

指令参数：number 将被设为当前运动控制器的卡号。取值范围为 0 - 15。

适用板卡：GE-X00-SX。

指令示例：参阅例程 GT_GetCurrentCardNo。

GT_Update

指令原型：short GT_Update(unsigned short axis)

指令说明：使指定控制轴参数生效。

指令参数：参数是指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SV。

指令示例：参阅例程 5-1、5-2。

GT_ZeroPos

指令原型：short GT_ZeroPos(unsigned short axis)

指令说明：将指定控制轴的实际位置、目标位置和规划位置设置为 0。只能在指定控制轴静止（GE 点位运动控制器）或无轨迹运动（GE 连续轨迹运动控制器）时执行该指令，否则该指令无效，并返回 1。

指令参数：参数是指定控制轴编号。

适用板卡：GE-X00-PX，GE-X00-SX。

指令示例：参阅例程 7-1、7-2。

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
基地西座二层 W211 室

电 话：0755-26970823 26970819 26970824

传 真：0755-26970821

电子邮件：support@gogoltech.com

网 址：<http://www.gogoltech.com.cn/>

固高科技（香港）有限公司

地 址：香港九龙清水湾香港科技大学新翼楼
3639 室

电 话：(852) 2358-1033

传 真：(852) 2358-4931

电子邮件：info@gogoltech.com

网 址：<http://www.gogoltech.com/>
