



运动控制器编程手册

基本功能

R1.5

2021 年 4 月

© 1999-2021 固高科技 版权所有

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。



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

联系我们

固高科技（深圳）有限公司

地址：深圳市高新技术产业园南区深港产学研基地西座
二楼 W211 室

电话：0755-26970817 26737236 26970824

传真：0755-26970821

电子邮件：support@googoltech.com

网址：<http://www.googoltech.com.cn>

固高科技（香港）有限公司

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

電話：+(852) 2358-1033

傳真：+(852) 2719-8399

電子郵件：info@googoltech.com

網址：<http://www.googoltech.com>

臺灣固高科技股份有限公司

地址：台中市西屯区福中二街 10 巷 22 号 2 楼

電話：+886-4-23588245

傳真：+886-4-23586495

電子郵件：twinfo@googoltech.com

文档版本

版本号	修订日期
1.0	2015 年 11 月 30 日
1.1	2018 年 7 月 23 日
1.2	2019 年 3 月 4 日
1.3	2019 年 7 月 8 日
1.4	2019 年 11 月 15 日
1.5	2021 年 4 月 20 日

前言

感谢选用固高运动控制器

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

固高产品的更多信息

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

你也可以通过电话（0755-26970817）咨询关于公司和产品的更多信息。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

电子邮件：support@googoltech.com；

电话：0755-26970843

发函至：深圳市高新技术产业园南区园深港产学研基地西座二楼 W211 室

固高科技（深圳）有限公司

邮编：518057

编程手册的用途

用户通过阅读本手册，能够了解运动控制器的基本控制功能，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

本手册由十三章内容组成，详细介绍了运动控制器的基本控制功能及编程实现。

相关文件

关于控制器的调试和安装，请参见随产品配套的运动控制器用户手册。

关于更复杂的控制器功能，请参见随产品配套的《运动控制器编程手册之高级功能》

关于扩展模块的使用，请参见随产品配套的扩展模块编程手册。



注意

通过固高科技官网可下载如驱动程序、dll 文件、例程、Demo 等相关文件，网址为：
www.googoltech.com.cn/pro_view-3.html

目录

前言	3
目录	错误!未定义书签。
第 1 章 指令列表	8
第 2 章 运动控制器函数库的使用	14
2.1 WINDOWS 系统下动态链接库的使用	14
2.1.1 Visual C++ 6.0 中的使用	14
2.1.2 Visual Basic 6.0 中的使用	15
2.1.3 Delphi 中的使用	15
2.1.4 VB.NET 中的使用	15
2.1.5 Visual C#中的使用	16
第 3 章 指令返回值及其意义	17
3.1 本章简介	17
3.2 指令返回值	17
3.3 例程	17
第 4 章 系统配置	19
4.1 本章简介	19
4.2 系统配置基本概念	19
4.2.1 硬件资源	19
4.2.2 软件资源	19
4.2.3 资源组合	20
4.3 系统配置工具	21
4.3.1 配置 axis	22
4.3.2 配置 step	25
4.3.3 配置 dac	26
4.3.4 配置 encoder	28
4.3.5 配置 control	30
4.3.6 配置 profile	31
4.3.7 配置 di	32
4.3.8 配置 do	33
4.4 配置文件生成和下载	34
4.5 使用 MCT2008 系统配置举例	35
4.5.1 开环控制模式	35
4.5.2 闭环控制模式	43
4.6 配置信息修改指令	50
4.6.1 指令列表	50
4.6.2 重点说明	51
4.6.3 例程	53
4.7 控制器配置初始化状态	55

第 5 章 运动状态检测	57
5.1 本章简介	57
5.2 指令列表	57
5.3 重点说明	57
5.3.1 轴状态定义.....	57
5.3.2 轴的运动参数.....	58
5.4 例程	59
第 6 章 运动模式	63
6.1 本章简介	63
6.2 点位运动模式.....	63
6.2.1 指令列表.....	63
6.2.2 重点说明.....	64
6.2.3 例程	64
6.3 JOG 运动模式	66
6.3.1 指令列表.....	66
6.3.2 重点说明.....	67
6.3.3 例程	67
6.4 电子齿轮 (GEAR) 运动模式	70
6.4.1 指令列表.....	70
6.4.2 重点说明.....	70
6.4.3 例程	72
6.5 插补运动模式.....	74
6.5.1 指令列表.....	74
6.5.2 重点说明.....	75
第 7 章 访问硬件资源	103
7.1 本章简介	103
7.2 访问数字 IO	103
7.2.1 指令列表.....	103
7.2.2 重点说明.....	103
7.2.3 例程	105
7.3 访问编码器	107
7.3.1 指令列表.....	107
7.3.2 例程	108
7.4 访问 DAC	109
7.4.1 指令列表.....	109
7.4.2 重点说明.....	109
7.4.3 例程	109
7.5 访问模拟量输入(仅适用于带模拟量功能端子板)	110
7.5.1 指令列表.....	110
7.5.2 重点说明.....	110
7.5.3 例程	110
第 8 章 高速硬件捕获	111
8.1 本章简介	111

8.2	指令列表	111
8.3	HOME/INDEX 硬件捕获	111
8.3.1	重点说明	111
8.3.2	例程	112
8.4	HOME 回原点	113
8.4.1	重点说明	113
8.4.2	例程	114
8.5	HOME+INDEX 回原点	117
8.5.1	重点说明	117
8.5.2	例程	119
8.6	探针捕获	123
8.6.1	重点说明	123
8.6.2	例程	123
8.7	HSIO 捕获	125
8.7.1	重点说明	125
8.7.2	例程	125
8.8	重复捕获	126
第 9 章	安全机制	127
9.1	本章简介	127
9.2	限位	127
9.2.1	指令列表	127
9.2.2	重点说明	127
9.2.3	例程	128
9.3	报警	129
9.4	平滑停止和急停	130
9.5	跟随误差极限	130
第 10 章	运动程序	131
10.1	本章简介	131
10.2	运动程序概述	131
10.3	运动程序的使用	132
10.3.1	编写运动程序	132
10.3.2	编译	132
10.3.3	指令列表	133
10.3.4	下载	133
10.3.5	绑定线程、函数和数据页	134
10.3.6	启动，停止，暂停线程	134
10.3.7	查询线程状态	134
10.3.8	例程	134
10.4	如何编写运动程序	139
10.4.1	语言元素	139
10.4.2	运算指令	139
10.4.3	流程控制与标准 C 语言的流程控制对比	140
10.5	可在运动程序中使用的指令	142
第 11 章	其他指令	144

11.1	本章简介	144
11.2	打开/关闭运动控制器	144
11.3	读取固件版本号.....	144
11.4	读取系统时钟.....	145
11.5	打开/关闭电机使能信号.....	145
11.6	维护位置值	145
11.7	电机到位检测.....	146
11.8	设置 PID 参数	150
11.9	反向间隙补偿.....	152
11.10	螺距误差补偿.....	152
11.11	位置比较输出.....	153
11.11.1	指令列表.....	153
11.11.2	重点说明.....	153
11.11.3	例程	154
第 12 章	SMART HOME（智能回原点）	157
12.1	本章简介	157
12.2	指令列表	157
12.3	重点说明	157
12.4	注意事项	160
12.5	例程	161
第 13 章	指令详细说明	163
第 14 章	索引	231
14.1	指令索引	231
14.2	例程索引	234
14.3	表格索引	235
14.4	图片索引	236

第1章 指令列表



提示

(1) 本手册中所有字体为蓝色的指令（如 [GT_PrfrTrap](#)）均带有超级链接，点击可跳转至指令说明。

(2) 本手册的指令全部为单卡库的指令说明，当工控机插入多张控制卡时，请调用多卡库指令，多卡库指令相比单卡库指令仅多出一个参数 `cardNum`（控制卡卡号），用于控制相应的控制卡。控制卡卡号确定：一般情况下，靠近 CPU 的 PCI 插槽为 0 号卡，依次排列（不排除有特殊的工控机，看实际情况做一些控制操作也可以确认卡号）。

表 1-1 指令列表

第 4 章 系统配置		19
4.4 配置文件生成和下载		34
GT_LoadConfig	下载配置信息到运动控制器，调用该指令后需再调用 GT_ClrSts 才能使该指令生效	209
4.6 配置信息修改指令		50
GT_AlarmOff	控制轴驱动报警信号无效	163
GT_AlarmOn	控制轴驱动报警信号有效	163
GT_LmtsOn	控制轴限位信号有效	206
GT_LmtsOff	控制轴限位信号无效	205
GT_ProfileScale	设置控制轴的规划器当量变换值	211
GT_EncScale	设置控制轴的编码器当量变换值	180
GT_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”	227
GT_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”	227
GT_StepOrthogonal	将脉冲输出通道的脉冲输出模式设置为“正交脉冲”	228
GT_SetMtrBias	设置模拟量输出通道的零漂电压补偿值	222
GT_GetMtrBias	读取模拟量输出通道的零漂电压补偿值	196
GT_SetMtrLmt	设置模拟量输出通道的输出电压饱和和极限值	222
GT_GetMtrLmt	读取模拟量输出通道的输出电压饱和和极限值	196
GT_EncSns	设置编码器的计数方向	181
GT_EncOn	设置为“外部编码器”计数方式	180
GT_EncOff	设置为“脉冲计数器”计数方式	180
GT_SetPosErr	设置跟随误差极限值	223
GT_GetPosErr	读取跟随误差极限值	197
GT_SetStopDec	设置平滑停止减速度和急停减速度	224
GT_GetStopDec	读取平滑停止减速度和急停减速度	200
GT_LmtSns	设置运动控制器各轴限位开关触发电平	205
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出	179
GT_SetStopIo	设置平滑停止和紧急停止数字量输入的信息	225
GT_GpiSns	设置运动控制器数字量输入的电平逻辑	204
GT_SetAdcFilter	设置模拟量输入的滤波器时间参数	212

第 5 章 运动状态检测		57
GT_GetSts	读取轴状态	200
GT_ClrSts	清除驱动器报警标志、跟随误差超限标志、限位触发标志 1. 只有当驱动器没有报警时才能清除轴状态字的报警标志 2. 只有当跟随误差正常以后，才能清除跟随误差超限标志 3. 只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志	173
GT_GetPrfMode	读取轴运动模式	198
GT_GetPrfPos	读取规划位置	198
GT_GetPrfVel	读取规划速度	199
GT_GetPrfAcc	读取规划加速度	198
GT_GetAxisPrfPos	读取轴(axis)的规划位置值	184
GT_GetAxisPrfVel	读取轴(axis)的规划速度值	184
GT_GetAxisPrfAcc	读取轴(axis)的规划加速度值	184
GT_GetAxisEncPos	读取轴(axis)的编码器位置值	183
GT_GetAxisEncVel	读取轴(axis)的编码器速度值	183
GT_GetAxisEncAcc	读取轴(axis)的编码器加速度值	182
GT_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值	183
GT_Stop	停止一个或多个轴的规划运动，停止坐标系运动	228

第 6 章 运动模式		63
6.2 点位运动模式		63
GT_PrflTrap	设置指定轴为点位运动模式	211
GT_SetTrapPrm	设置点位运动模式下的运动参数	225
GT_GetTrapPrm	读取点位运动模式下的运动参数	201
GT_SetPos	设置目标位置	223
GT_GetPos	读取目标位置	197
GT_SetVel	设置目标速度	227
GT_GetVel	读取目标速度	202
GT_Update	启动点位运动	230
6.3 Jog 运动模式		66
GT_PrflJog	设置指定轴为 Jog 运动模式	210
GT_SetJogPrm	设置 Jog 运动模式下的运动参数	221
GT_GetJogPrm	读取 Jog 运动模式下的运动参数	194
GT_SetVel	设置目标速度	227
GT_GetVel	读取目标速度	202
GT_Update	启动 Jog 运动	230
6.4 电子齿轮（Gear）运动模式		70
GT_PrflGear	设置指定轴为电子齿轮运动模式	210
GT_SetGearMaster	设置电子齿轮运动跟随主轴	219
GT_GetGearMaster	读取电子齿轮运动跟随主轴	193
GT_SetGearRatio	设置电子齿轮比	219

第 1 章 指令列表

GT_GetGearRatio	读取电子齿轮比	194
GT_GearStart	启动电子齿轮运动	181
6.5 插补运动模式		74
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	215
GT_GetCrdPrm	查询坐标系参数	188
GT_CrdData	向插补缓存区增加插补数据	177
GT_LnXY	缓存区指令，二维直线插补	206
GT_LnXYZ	缓存区指令，三维直线插补	207
GT_LnXYZA	缓存区指令，四维直线插补	208
GT_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)	207
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	209
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	208
GT_ArcXYR	缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)	164
GT_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)	163
GT_ArcYZR	缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)	165
GT_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)	165
GT_ArcZXR	缓存区指令，ZX 平面圆弧插补(以终点位置和半径为输入参数)	166
GT_ArcZXC	缓存区指令，ZX 平面圆弧插补(以终点位置和圆心位置为输入参数)	166
GT_ArcXYZ	缓冲区指令，空间圆弧插补	167
GT_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令	170
GT_BufDelay	缓存区指令，缓存区内延时设置指令	169
GT_BufDA	缓存区指令，缓存区内输出 DA 值	169
GT_BufLmtsOn	缓存区指令，缓存区内有效限位开关	171
GT_BufLmtsOff	缓存区指令，缓存区内无效限位开关	171
GT_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息	172
GT_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动	172
GT_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动	170
GT_CrdSpace	查询插补缓存区剩余空间	178
GT_CrdClear	清除插补缓存区内的插补数据	177
GT_CrdStart	启动插补运动	178
GT_CrdStatus	查询插补运动坐标系状态	178
GT_SetUserSegNum	缓存区指令，设置自定义插补段段号	226
GT_GetUserSegNum	读取自定义插补段段号	201
GT_GetRemainderSegNum	读取未完成的插补段段数	199
GT_SetOverride	设置插补运动目标合成速度倍率	222
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	216
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	188
GT_GetCrdPos	查询该坐标系的当前坐标位置值	188
GT_GetCrdVel	查询该坐标系的合成速度值	189
GT_InitLookAhead	初始化插补前瞻缓存区	205
GT_SetCrdBufferMode	获取插补缓冲区的模式（静态模式或动态模式）	214
GT_GetCrdBufferMode	获取插补缓冲区的模式（静态模式或动态模式）	187
GT_GetCrdSegmentTime	获取插补段的执行时间	188

GT_GetCrdTime	获取插补运动的执行时间	189
---------------	-------------	-----

第 7 章 访问硬件资源		103
7.2 访问数字 IO		103
GT_GetDi	读取数字 IO 输入状态	190
GT_GetDiRaw	读取数字 IO 输入状态的原始值	191
GT_GetDiReverseCount	读取数字量输入信号的变化次数	191
GT_SetDiReverseCount	设置数字量输入信号的变化次数的初值	217
GT_SetDo	设置数字 IO 输出状态	217
GT_SetDoBit	按位设置数字 IO 输出状态	218
GT_SetDoBitReverse	使数字量输出信号输出定时脉冲信号	218
GT_GetDo	读取数字 IO 输出状态	192
7.3 访问编码器		107
GT_GetEncPos	读取编码器位置	192
GT_GetEncVel	读取编码器速度	192
GT_SetEncPos	修改编码器位置	219
7.4 访问 DAC		109
GT_SetDac	设置 DAC 输出电压	216
GT_GetDac	读取 DAC 输出电压	190
7.5 访问模拟量输入(仅适用于带模拟量功能端子板)		110
GT_GetAdc	读取模拟量输入的电压值	181
GT_GetAdcValue	读取模拟量输入的数字转换值	182

第 8 章 高速硬件捕获		111
GT_SetCaptureMode	设置编码器捕获方式，并启动捕获	213
GT_GetCaptureMode	读取编码器捕获方式	186
GT_GetCaptureStatus	读取编码器捕获状态	186
GT_SetCaptureSense	设置捕获电平	214
GT_ClearCaptureStatus	清除捕获状态	173
GT_SetCaptureRepeat	设置重复捕获	213
GT_GetCaptureRepeatStatus	查询重复捕获状态	185
GT_GetCaptureRepeatPos	查询重复捕获位置值	185
GT_StopCapture	停止捕获	229

第 9 章 安全机制		127
GT_SetSoftLimit	设置轴正向软限位和负向软限位	224
GT_GetSoftLimit	读取轴正向软限位和负向软限位	199

第 10 章 运动程序		131
GT_Download	下载运动程序到运动控制器	179
GT_GetFunId	读取运动程序中函数的标识	193
GT_GetVarId	读取运动程序中变量的标识	201
GT_Bind	绑定线程、函数、数据页	169

第 1 章 指令列表

GT_RunThread	启动线程	212
GT_StopThread	停止正在运行的线程	229
GT_PauseThread	暂停正在运行的线程	210
GT_GetThreadSts	读取线程的状态	200
GT_SetVarValue	设置运动程序中变量的值	226
GT_GetVarValue	读取运动程序中变量的值	202

第 11 章 其它指令		142
11.2 打开/关闭运动控制器		144
GT_Open	打开运动控制器	210
GT_Close	关闭运动控制器	173
GT_SetCardNo	切换当前运动控制器卡号	214
GT_GetCardNo	读取当前运动控制器卡号	186
GT_Reset	复位运动控制器	211
11.3 读取固件版本号		144
GT_GetVersion	读取运动控制器固件的版本号	202
11.4 读取系统时钟		145
GT_GetClock	读取运动控制器系统时钟	187
GT_GetClockHighPrecision	读取运动控制器系统高精度时钟	187
11.5 打开/关闭电机使能信号		145
GT_AxisOn	打开驱动器使能	168
GT_AxisOff	关闭驱动器使能	168
11.6 维护位置值		145
GT_SetPrfPos	修改指定轴的规划位置	224
GT_SynchAxisPos	axis 合成规划位置和所关联的 profile 同步 axis 合成编码器位置和所关联的 encoder 同步	229
GT_ZeroPos	清零规划位置和实际位置，并进行零漂补偿	230
11.7 电机到位检测		146
GT_SetAxisBand	设置轴到位误差带 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志	212
GT_GetAxisBand	读取轴到位误差带	182
11.8 设置 PID 参数		150
GT_SetControlFilter	设定 PID 索引，支持 3 组 PID 参数	214
GT_GetControlFilter	读取当前 PID 索引	187
GT_SetPid	设置 PID 参数	222
GT_GetPid	读取 PID 参数	197
11.9 反向间隙补偿		152
GT_SetBacklash	设置反向间隙补偿的相关参数	212
GT_GetBacklash	读取反向间隙补偿的相关参数	185
11.10 螺距误差补偿		152
GT_SetLeadScrewComp	设置螺距误差补偿的相关参数	220
GT_EnableLeadScrewComp	使能螺距误差补偿功能	179

11.11 位置比较输出		153
GT_ComparePulse	在 HSIO 口输出 IO	176
GT_CompareData	设置位置比较参数并启动位置比较输出	174
GT_CompareStatus	查询位置比较状态	176
GT_CompareStop	取消位置比较输出	177
GT_CompareLinear	设置等间距重复位置比较输出	175
GT_CompareContinuePulseMode	设置一维位置比较输出是否为连续脉冲模式	174

第 12 章 Smart Home（智能回原点）		157
GT_GoHome	启动 Smart Home 实现各种方式回原点	202
GT_GetHomePrm	读取设置到控制器的 Smart Home 回原点参数	204
GT_GetHomeStatus	获取 Smart Home 回原点的状态	194

第2章 运动控制器函数库的使用



注意

运动控制器驱动程序、dll 文件、例程、Demo 等相关文件请通过固高科技官网下载，网址为：www.googoltech.com.cn/pro_view-3.html

2.1 Windows 系统下动态链接库的使用

在 Windows 系统下使用运动控制器，首先要安装驱动程序。在安装前需要提前下载运动控制器的驱动程序请通过固高科技官网下载。

运动控制器指令函数动态链接库也请通过官网提前下载至本地硬盘中。运动控制器的动态链接库文件名为 gts.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。

目前提供了 Visual C++、Visual Basic、C#、VB.NET 和 Delphi 等编程语言的动态链接库（这些动态链接库均为 C++ 编译，只是函数、结构体的声明因具体编译语言而异），它们的动态库文件夹分别为 VC、VB6.0、C#、VB.NET 和 Delphi，它们的子目录包含 32bit 和 64bit 两个文件夹。其中，32bit 为使用 32 位编译器生成的动态链接库，64bit 为使用 64 位编译器生成的动态链接库；每个子文件夹包含了 single_card 和 multi_card 两个子目录。其中，single_card 为单卡库，multi_card 为多卡库。对于 LabVIEW 用户，仍然可以使用 VC 文件夹提供的 VC 版本动态链接库，通过 LabVIEW 工具把函数指令导出成 VI 供 LabVIEW 程序调用（具体方法参考 LabVIEW 关于调用 VC 动态链接库的相关说明）。



注意

1. 后文涉及的函数指令说明均为单卡库函数指令。
2. 单卡库和多卡库函数的区别在于多卡库的函数多了一个“卡号”参数（第一张卡的卡号为 0）。
3. 如果应用程序需要操作多张卡，仍然可以使用单卡库提供的函数，但是为了编程方便，建议直接使用多卡库，即使是只操作一张卡也可以使用多卡库。
4. 与本说明书配套发行的 gts.dll 动态链接库包含了扩展 IO 模块的函数指令，用户可以直接调用操作扩展 IO 模块，函数指令的详细说明请参考扩展 IO 模块配套《扩展模块编程手册》。安装 GTS 控制卡的驱动后，无需再安装扩展 IO 模块驱动，两个驱动是一样的。

2.1.1 Visual C++ 6.0 中的使用

- (1) 启动 Visual C++ 6.0，新建一个工程；
- (2) 将下载的动态链接库压缩包里的 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gts.lib;
- (5) 在应用程序文件中加入函数库头文件的声明, 例如: #include “gts.h”;

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。

对于步骤(4), 还有一种比较简便的方法, 那就是在应用程序文件里面添加包含链接文件的声明, 例如: #pragma comment (lib,"gts.lib") 。

上述关于Visual C++ 6.0调用GTS库函数的方法同样适用于使用Visual Studio平台开发。

2.1.2 Visual Basic 6.0 中的使用

- (1) 启动Visual Basic, 新建一个工程;
- (2) 将下载的动态链接库压缩包里VB6.0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gts.bas, 将其添加到工程当中;

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。

2.1.3 Delphi 中的使用

- (1) 启动Delphi, 新建一个工程;
- (2) 将下载的动态链接库压缩包里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;

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。

2.1.4 VB.NET 中的使用

- (1) 启动Visual Studio, 按照“File”->“New”, 选择建立VB工程;
- (2) 将下载的动态链接库压缩包里VB.NET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, 注意: gts.dll应复制到“.\bin”文件夹中的debug或者release文件夹中;
- (3) 选择“project”菜单下的“Add existing Item”菜单项, 选择函数声明文件, 如gts.vb, 将其添加到工程当中;

至此, 用户就可以在Visual Studio中使用VB.NET模块调用函数库中的任何函数, 开始编写应用程序。

2.1.5 Visual C#中的使用

- (1) 启动Visual Studio，按照“File”->“New”，选择建立C#工程；
- (2) 将下载的动态链接库压缩包里C#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，注意：gts.dll应复制到“.\bin”文件夹中的debug或者release文件夹中；
- (3) 选择“project”菜单下的“Add existing Item”菜单项，选择函数声明文件，如gts.cs，将其添加到工程当中；

至此，用户就可以在Visual Studio中使用C#模块调用函数库中的任何函数，开始编写应用程序。

第3章 指令返回值及其意义

3.1 本章简介

本章主要介绍了运动控制器指令的所有返回值及其意义。在“第 13 章 指令详细说明”，每一条指令介绍的“指令返回值”一项中，都有更加详细的关于返回值意义和操作的介绍。

3.2 指令返回值

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动态链接库中。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 3-1 所示。

表 3-1 运动控制器指令返回值定义

返回值	意义	处理方法
0	指令执行成功	无
1	指令执行错误	1. 检查当前指令的执行条件是否满足
2	license 不支持	1. 如果需要此功能，请与生产厂商联系。
7	指令参数错误	1. 检查当前指令输入参数的取值
8	不支持该指令	DSP 固件不支持该指令对应的功能
-1~-5	主机和运动控制器通讯失败	1. 是否正确安装运动控制器驱动程序 2. 检查运动控制器是否接插牢靠 3. 更换主机 4. 更换控制器 5. 运动控制器的金手指是否干净
-6	打开控制器失败	1. 是否正确安装运动控制器驱动程序 2. 是否调用了 2 次 GT_Open 指令 3. 其他程序是否已经打开运动控制器，或进程中是否还驻留着打开控制器的程序
-7	运动控制器没有响应	1. 更换运动控制器
-8	多线程资源忙	指令在线程里执行超时才返回，有可能是 PCI 通信异常，导致指令无法及时返回

3.3 例程

例程 3-1 检测 GT 指令是否正常执行

检测 GT 指令是否正常执行，该程序在 VC 6.0 的 win32 console application 工程下运行。

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

```
// 该函数检测某条GT指令的执行结果，command为指令名称，error为指令执行返回值
void commandhandler(char *command, short error)
{
    // 如果指令执行返回值为非0，说明指令执行错误，向屏幕输出错误结果
    if(error)
    {
        printf("%s = %d\n", command, error);
    }
}

int main(int argc, char* argv[])
{
    // 指令返回值变量
    short sRtn;
    sRtn = GT_Open();
    // 指令返回值校验
    commandhandler("GT_Open", sRtn);
    return 0;
}
```



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

第4章 系统配置

4.1 本章简介

在使用运动控制器进行各种操作之前，需要对运动控制器进行配置，使运动控制器的状态和各种工作模式能够满足客户的要求。这个过程，叫做系统配置。

在运动控制器管理软件 Motion Controller Toolkit 2008（以下简称 MCT2008）中包括一个系统配置的组件，用户可以利用该组件来对运动控制器进行配置，配置完成之后，生成相应的配置文件*.cfg，用户在编程时，调用相关的指令，将配置信息传递给运动控制器，即可完成整个运动控制器的配置工作。用户也可以利用相关的指令完成运动控制器的配置。



本手册中所有字体为蓝色的指令（如 [GT_PrffTrap](#)）均带有超级链接，点击可跳转至指令说明。

4.2 系统配置基本概念

运动控制器内部包含了各种软硬件资源，各种软硬件资源之间相互组合，即可实现运动控制器的各种应用。

4.2.1 硬件资源

数字量输出资源(do): 包括伺服使能数字量输出、伺服报警清除数字量输出、通用数字量输出。

数字量输入资源(di): 包括正限位数字量输入、负限位数字量输入、驱动报警数字量输入、原点信号数字量输入、通用数字量输入。

编码器计数资源(encoder): 用来对外部编码器的脉冲输出进行计数。

脉冲输出资源(step): 脉冲输出通道，可以输出“脉冲+方向”或者“CCW/CW”控制脉冲。

电压输出资源(dac): 电压输出通道，输出-10V~+10V 的控制电压。

4.2.2 软件资源

规划器资源(profile): 根据运动模式和运动参数实时计算规划位置和规划速度，生成所需的速度曲线，实时地输出规划位置。

伺服控制器资源(control): 根据伺服控制算法、控制参数、跟随误差实时地计算控制量。

轴资源(axis): 将软件资源、硬件资源进行组合，作为整体进行操作。其中包括驱动报警信号、正限位信号、负限位信号、平滑停止信号、紧急停止信号的管理；规划器输出的规划位置的当量变换；编码器计数位置的当量变换等功能。

4.2.3 资源组合

系统配置就是将上述的硬件资源和软件资源相互组合，并对各个资源的基本属性进行配置的过程。下面的两个例子描述了资源组合的基本概念。

1. 开环控制模式（脉冲控制）

使用步进电机，或使用伺服电机的位置控制模式的运动控制系统，其基本配置如图 4-1 所示。

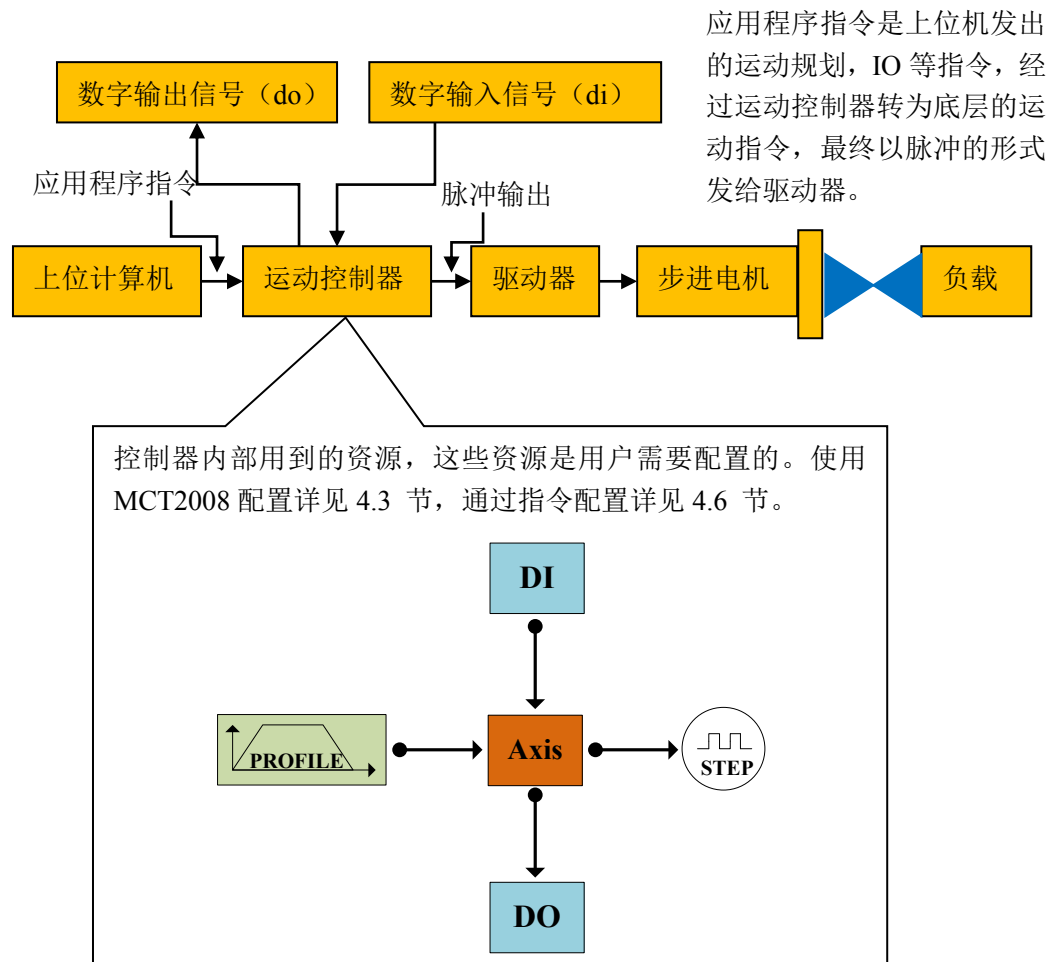


图 4-1 开环运动控制系统的配置

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到 step，由 step 产生控制脉冲，驱动电机运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信号，让电机使能。

2. 闭环控制模式（模拟量控制）

一个闭环伺服电机运动控制系统的常见组成部分如图 4-2 所示。

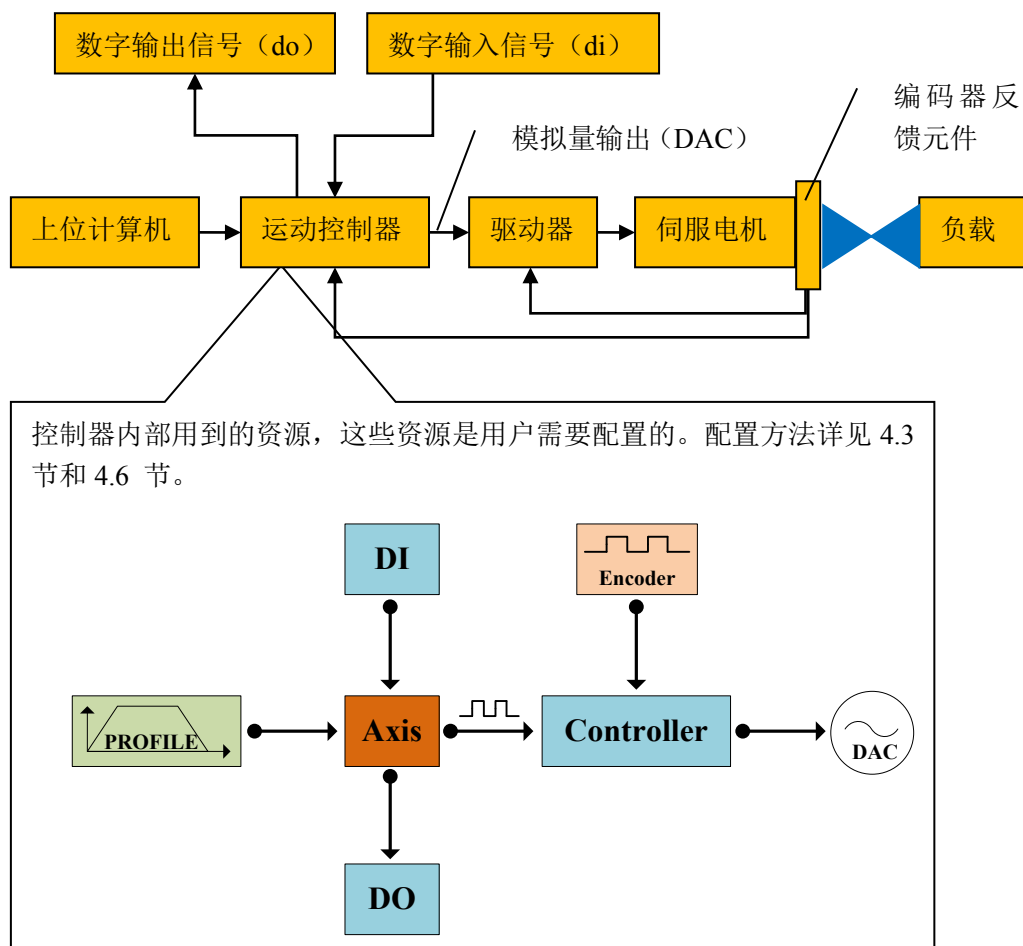


图 4-2 闭环运动控制系统的配置

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到伺服控制器中，伺服控制器将规划位置与 encoder 的计数位置进行比较，获得跟随误差，并通过一定的伺服控制算法，得到实时的控制量，将控制量传递给 dac，由 dac 转换成控制电压来控制电机的运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信号，让电机使能。

4.3 系统配置工具

使用固高科技提供的 Motion Controller Toolkit 2008 运动控制器管理软件能够方便地对系统进行配置，启动软件以后显示如图 4-3 所示界面。



图 4-3 MCT2008 运动控制器管理软件界面

选择“工具”菜单，点击“控制器配置”，打开运动控制器配置面板就可以对系统进行配置。如图 4-4 所示。



图 4-4 打开控制器配置

4.3.1 配置 axis

Axis 配置界面如图 4-5 所示。

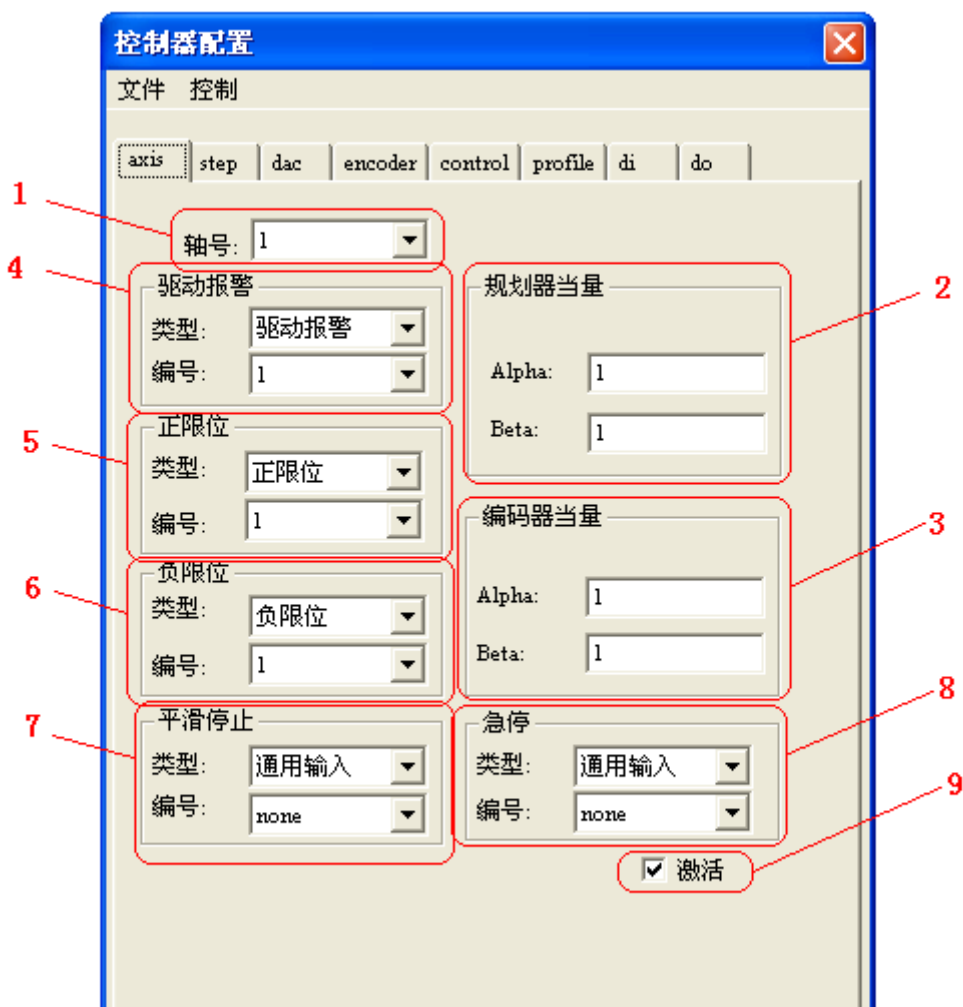


图 4-5 axis 配置界面

配置说明：“axis”选项主要用来配置轴控制的相关信息。配置后对控制系统可能产生的影响如图 4-6 所示。

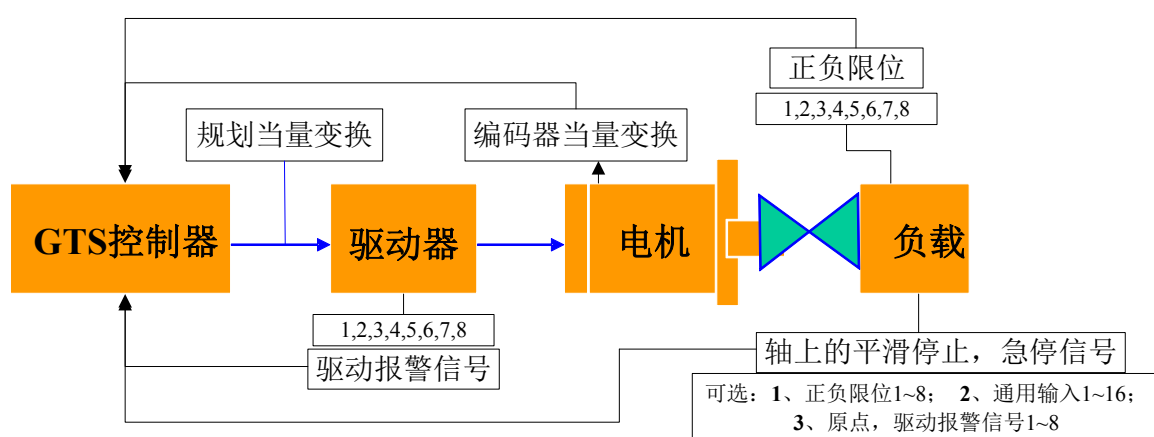


图 4-6 axis 配置对控制系统的影响

一般情况下，驱动报警、正负限位信号均保持默认设置状态，即“驱动报警”，“正限位”和“负限位”的“编号”同“轴号”相对应。为了帮助用户提高配置的灵活性，可选用其他配置，如将轴 2 的限位信号配给轴 1 使用。用户设置时，一般情况下，该选项保持默认设置，除非出现特殊需求。

1. axis 编号选择：选择需要进行配置的 axis 的编号。

2. 规划器当量变换参数：如果需要在 axis 中对规划器输出的规划位置进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta P_{profile}}{\Delta P_{axis}} = \frac{Alpha}{Beta}$$

其中：

$\Delta P_{profile}$ ——规划器输出的规划位置的变化量

ΔP_{axis} ——axis 输出的规划位置的变化量

系统默认的 Alpha 和 Beta 都为 1，因此，规划器输出的规划位置在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767, 0)和(0, 32767)；Beta 的取值范围：(-32767, 0)和(0, 32767)。该项可以通过指令 [GT_ProfileScale](#) 来设置。

3. 编码器当量变换参数：如果需要在 axis 中对编码器计数的位置值进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta E_{enc}}{\Delta E_{axis}} = \frac{Alpha}{Beta}$$

其中：

ΔE_{enc} ——编码器计数的位置值的变化量

ΔE_{axis} ——axis 输出的编码器位置值的变化量

系统默认的 Alpha 和 Beta 都为 1，因此，编码器计数的位置值在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767, 0)和(0, 32767)；Beta 的取值范围：(-32767, 0)和(0, 32767)。该项可以通过指令 [GT_EncScale](#) 来设置。



注意

规划器当量和编码器当量的设置参数要满足 $Alpha \geq Beta$ 。

4. 驱动报警信号数字量输入选择：选择驱动报警信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为驱动报警信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择驱动报警数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的驱动报警信号无效。驱动报警无效可以通过指令 [GT_AlarmOff](#) 设置，驱动报警有效可以通过指令 [GT_AlarmOn](#) 设置。
5. 正限位信号数字量输入选择：选择正限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为正限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择正限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的正限位信号无效。限位开关无效可以通过指令 [GT_LmtsOff](#) 设置，限位开关有效可以通过指令 [GT_LmtsOn](#) 设置。

6. 负限位信号数字量输入选择：选择负限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为负限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择负限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的负限位信号无效。限位开关无效可以通过指令 `GT_LmtsOff` 设置，限位开关有效可以通过指令 `GT_LmtsOn` 设置。



驱动报警信号、正负限位信号在控制器复位状态下，都默认各轴对应相应的报警和限位信号，如 1 轴的驱动报警以及正负限位的编号都是 1 号。
但若有特殊情况，可以通过设置将不同轴上的限位和报警信号挂接到本轴上，如将 1 轴的正限位设置为 2 号。这种情况下，若 2 号正限位触发了，该信号将传递给 1 轴。

7. 平滑停止信号数字量输入选择：选择平滑停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为平滑停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有平滑停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 没有平滑停止信号。平滑停止信号数字量输入选择可以通过指令 `GT_SetStopIo` 设置。
8. 紧急停止信号数字量输入选择：选择紧急停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为紧急停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有紧急停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 没有紧急停止信号。紧急停止信号数字量输入选择可以通过指令 `GT_SetStopIo` 设置。
9. Axis 激活选项：如果 axis 不被激活，则与该 axis 相关的所有计算和管理任务将会无效。默认 axis 都是激活的。但是如果没有用到某个 axis 的相关功能，则可以不把该 axis 激活，这样可以节约运动控制器的处理资源。

4.3.2 配置 step

Step 配置界面如图 4-7 所示。



图 4-7 step 配置界面

配置说明：“step”选项主要用于配置脉冲控制模式。若采用的脉冲控制，而且是“脉冲+方向”的方式，该选项保持默认设置。如果轴 1 不使用脉冲控制模式，应当将 step 索引选 1，并选择不激活。以此类推。

1. Step 索引：选择需要进行配置的 step 的编号。
2. Step 输出脉冲信号模式选择：可以选择 step 脉冲输出通道的脉冲输出模式，可以为“脉冲+方向”或者“CCW/CW”，默认为“脉冲+方向”。设置为“脉冲+方向”模式，可以调用指令 [GT_StepDir](#) 来实现；设置为“CCW/CW”模式，可以调用指令 [GT_StepPulse](#) 来实现。（补充说明：可以调用指令 [GT_StepOrthogonal](#) 将脉冲输出模式设置为正交脉冲）
3. Step 激活选项：如果 step 不被激活，则该脉冲输出通道将不可用，不会输出脉冲。默认 step 都是激活的。但是如果没用用到某个 step，则可以不把该 step 激活，这样可以节约运动控制器的处理资源。

4.3.3 配置 dac

Dac 配置界面如图 4-8 所示。

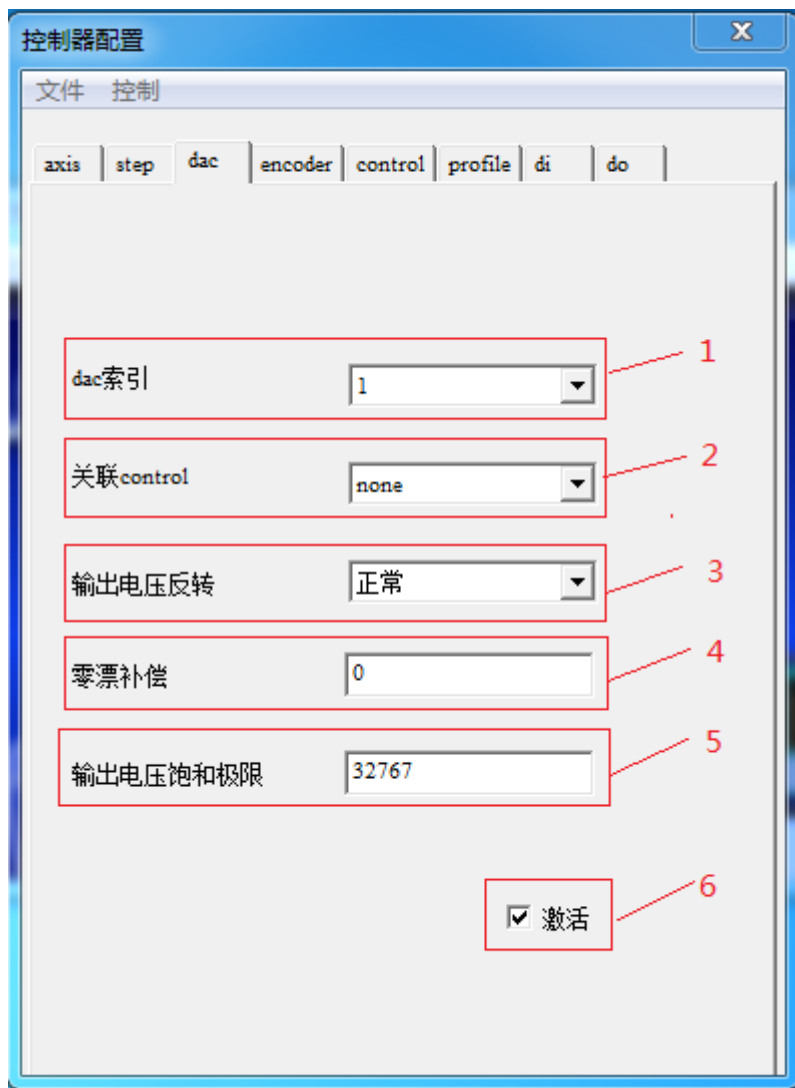


图 4-8 dac 配置界面

配置说明：“dac”选项主要用于设置 dac 的输出状态。dac 的输出在闭环控制时，作为伺服电压输出控制，用户不可控制伺服电压输出大小。但在开环状态下，可作为通用电压输出控制，用户可调用 [GT_SetDac](#) 设置输出电压的大小。

1. Dac 索引：选择需要进行配置的 dac 的编号。注意：当电机采用闭环控制模式时，默认是将轴号和 Dac 编号相对应。如轴 1 的电压控制的通道为 Dac1。
2. 关联 control：把该 Dac 关联到 control 模块，此处为关联的 control 模块编号（对于 4 轴卡为 1~3，对于 8 轴卡为 1~8），取值为 none 表示没有关联 control，则无法实现闭环控制。**当 Dac 关联到 control 后就不能调用 [GT_SetDac](#) 单独控制 Dac 输出了。**
3. Dac 输出电压反转：选择是否需要将 dac 的输出电压取反，如果为“正常”，则向 dac 中写入正值时，dac 输出正电压，向 dac 中写入负值时，dac 输出负电压；如果为“取反”，则反之。
4. Dac 的零漂补偿值：如果需要对 dac 进行零漂补偿时，在这里设置具体的零漂补偿值。取值范围：[-32768, 32767]。该项可以通过指令 [GT_SetMtrBias](#) 来设置。
5. Dac 输出电压饱和极限：该项设置 dac 能够输出的最大电压绝对值。取值范围：(0, 32767]。换算关系如下：已知允许输出的电压范围为 -aV~aV，则设置的值为 $32767 * a / 10$ ，取整数。例如，如果设置为 32767，则允许输出的电压范围为：[-10V, +10V]，设置为 16384，则允

许输出的电压范围为：[-5V, +5V]。如果 control 输出的控制量绝对值，或者用户使用 GT_SetDac 指令设置的电压值的绝对值超过设定值时，将会按照该项设置的参数被限制在指定电压范围之内。该项可以通过指令 GT_SetMtrLmt 来设置。

6. Dac 激活选项：如果 dac 不被激活，则该电压输出通道将不可用，不会输出电压值。默认 dac 都是激活的。但是如果没有用到某个 dac，则可以不把该 dac 激活，这样可以节约运动控制器的处理资源。

4.3.4 配置 encoder

Encoder 配置界面如图 4-9 所示。

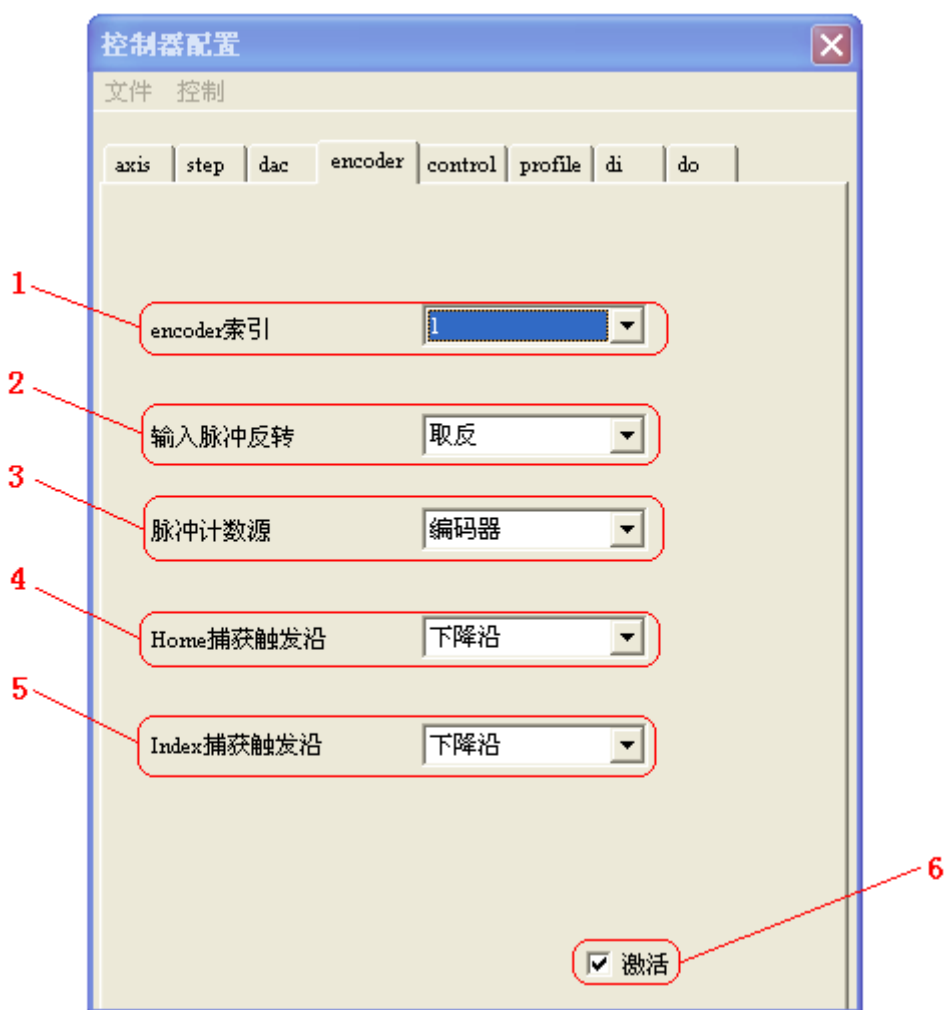


图 4-9 encoder 配置界面

配置说明：“encoder”选项主要配置与编码器有关的参数。配置后对控制系统可能产生的影响如图 4-10 所示。

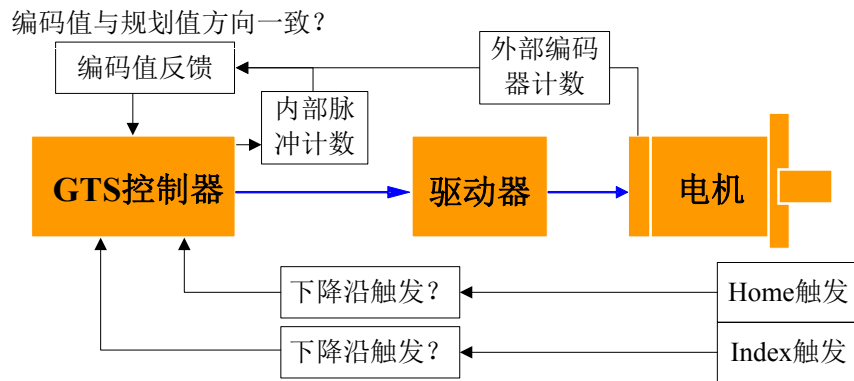


图 4-10 encoder 配置对控制系统的影响

当编码器值与规划值方向相反时，可以通过修改“输入脉冲反转”来校正。在闭环控制模式下，若出现“飞车”现象，也可通过修改该选项来校正。脉冲计数源的设置，一般情况下，保持默认设置。若没安装编码器，则可通过设置该选项为“脉冲计数器”。对于“Home 触发沿”和“Index 触发沿”的设置，取决于传感器的安装，若不能触发，可以通过修改这部分。

1. Encoder 索引：选择需要进行配置的 encoder 的编号。
2. 输入脉冲反转：运动控制器可以接收正交编码器信号，该项选项与反馈脉冲方向以及编码器计数方向的关系如图 4-11 所示，该项可以通过指令 `GT_EncSns` 来修改。

	正常		取反	
A 相				
B 相				
编码器	计数增加	计数减少	计数增加	计数减少

图 4-11 encoder 输入脉冲反转项的影响

3. 脉冲计数源：表示编码器计数来源，默认情况下是外部编码器计数。如果没有外接编码器，则可以将其设置为脉冲计数器，encoder 将会对 step 输出的脉冲个数进行计数。设置为外部编码器，可以调用指令 `GT_EncOn` 来实现；设置为脉冲计数器，可以调用指令 `GT_EncOff` 来实现。

设置为外部编码器是指通过外部安装的编码器计数值来计算，而脉冲计数器则是指通过控制器内部硬件来计算发出去的脉冲个数。在闭环控制方式下，必须设置成外部编码器计数方式。

4. Home 捕获触发沿：用来设置 Home 捕获的触发沿，默认为下降沿触发。如果选择了常闭开关，可以将捕获沿设置为上升沿触发。该项可以通过指令 `GT_SetCaptureSense` 来修改。
5. Index 捕获触发沿：用来设置 Index 捕获的触发沿，默认为下降沿触发。该项可以通过指令 `GT_SetCaptureSense` 来修改。
6. Encoder 激活选项：如果 encoder 不被激活，则将不会对输入脉冲进行计数。默认 encoder 都是激活的。但是如果没用用到某个 encoder，则可以不把该 encoder 激活，这样可以节约运动控制器的处理资源。

4.3.5 配置 control

Control 配置界面如图 4-12 所示。

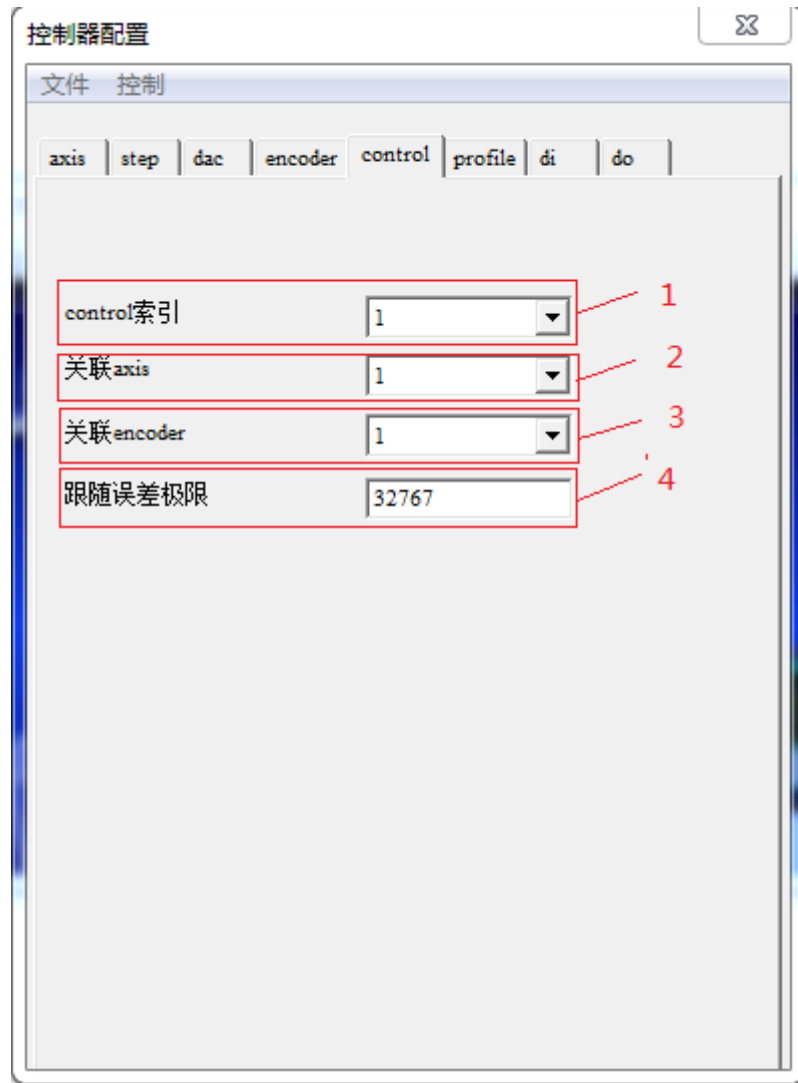


图 4-12 control 配置界面

配置说明：“control”选项主要设置闭环控制的参数。其中跟随误差表示如图 4-13 所示。

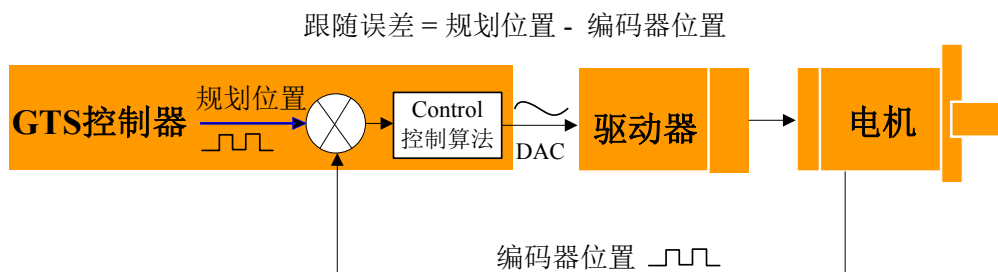


图 4-13 跟随误差解释图

1. Control 索引：选择需要进行配置的 control 的编号。
2. 关联 axis：默认为不关联（none），如果需要伺服闭环控制模式，应该使 control 与 axis 关联，它是闭环控制模块的输入，若无特殊要求，按照一一对应进行关联即可，即 control1

关联到 axis1, control2 关联到 axis2, 以此类推。

3. 关联 encoder: 默认为不关联 (none), 如果需要伺服控制模式, 应该使 control 与 encoder 关联, 它是闭环控制模块的反馈, 若无特殊要求, 按照一一对应进行关联即可, 即 control1 关联到 encoder1, control2 关联到 encoder2, 以此类推。
4. 跟随误差极限: 表示规划位置和实际位置的误差的极限。当跟随误差超过设定的极限时, 自动关闭 axis 的驱动器使能信号。默认值为: 32767, 单位: pulse。该项可以通过指令 `GT_SetPosErr` 来设置。

4.3.6 配置 profile

Profile 配置界面如图 4-14 所示。

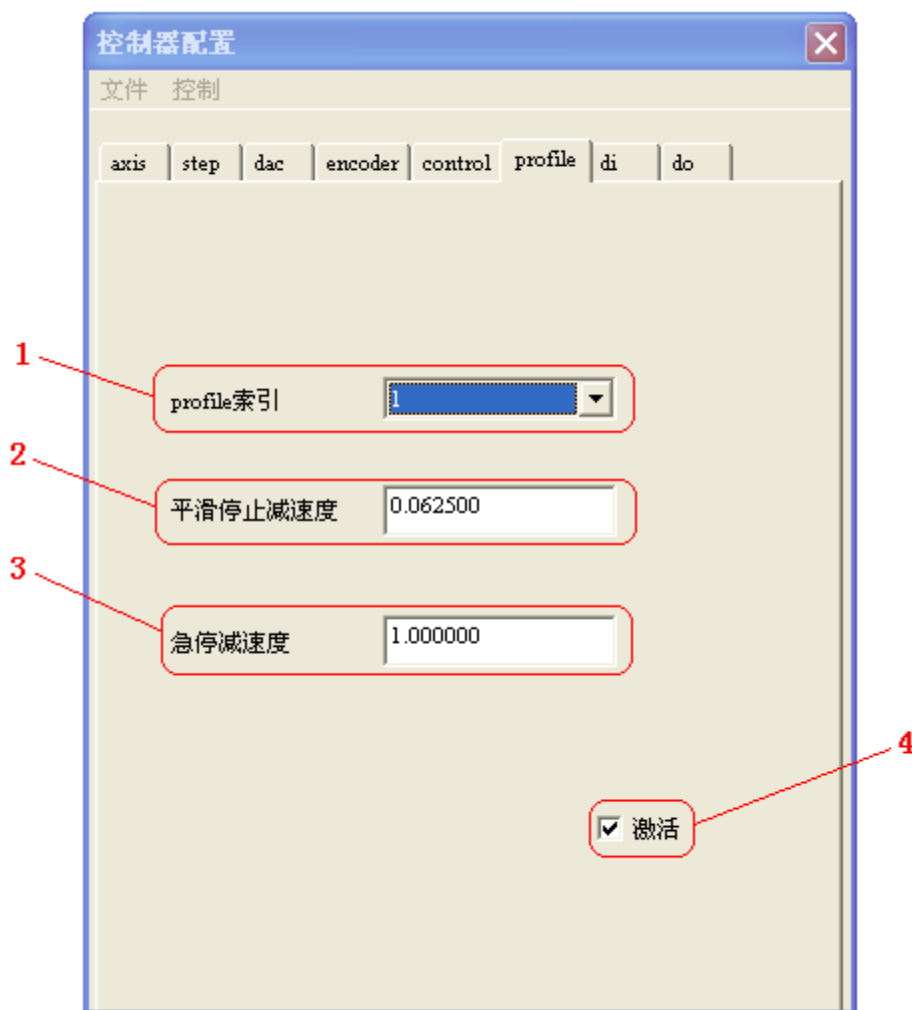


图 4-14 profile 配置界面

1. Profile 索引: 选择需要进行配置的 profile 的编号。
2. 平滑停止减速度: 表示调用 `GT_Stop` 指令平滑停止时所使用的减速度, 默认为 1pulse/ms²。该值可以通过调用指令 `GT_SetStopDec` 来修改。
3. 紧急停止减速度: 表示调用 `GT_Stop` 指令急停时所使用的减速度, 默认为 1000pulse/ms²。该值可以通过调用指令 `GT_SetStopDec` 来修改。

4. Profile 激活选项: 如果 profile 不被激活, 则将不会进行运动规划。默认 profile 都是激活的。但是如果没用用到某个 profile, 则可以不把该 profile 激活, 这样可以节约运动控制器的处理资源。

4.3.7 配置 di

Di 配置界面如图 4-15 所示。



图 4-15 di 配置界面

1. Di 类型选择: 选择需要配置的 di 的类型, 包括: 驱动报警、正限位、负限位、原点、通用输入、手轮。
2. Di 索引: 选择需要配置的 di 的编号。
3. 输入反转: 表示 di 的输入逻辑取反。默认情况下, 0 表示输入低电平, 1 表示输入高电平; 输入反转后, 0 表示输入高电平, 1 表示输入低电平。该项可以通过指令 [GT_GpiSns](#) 设置。
4. 滤波时间: 表示 di 输入信号维持设定时间才有效, 默认滤波时间为 3, 单位: 250us, 即 750us。



注意

设置该滤波参数需根据具体情况进行调整, 比如, 若信号维持时间很短, 但又需要获取到, 此时需要将滤波时间设置 0。

5. Di 激活选项：如果 di 不被激活，则该数字量输入将不起作用。默认 di 都是激活的。但是如果没用用到某个 di，则可以不把该 di 激活，这样可以节约运动控制器的处理资源。

4.3.8 配置 do

Do 配置界面如图 4-16 所示。

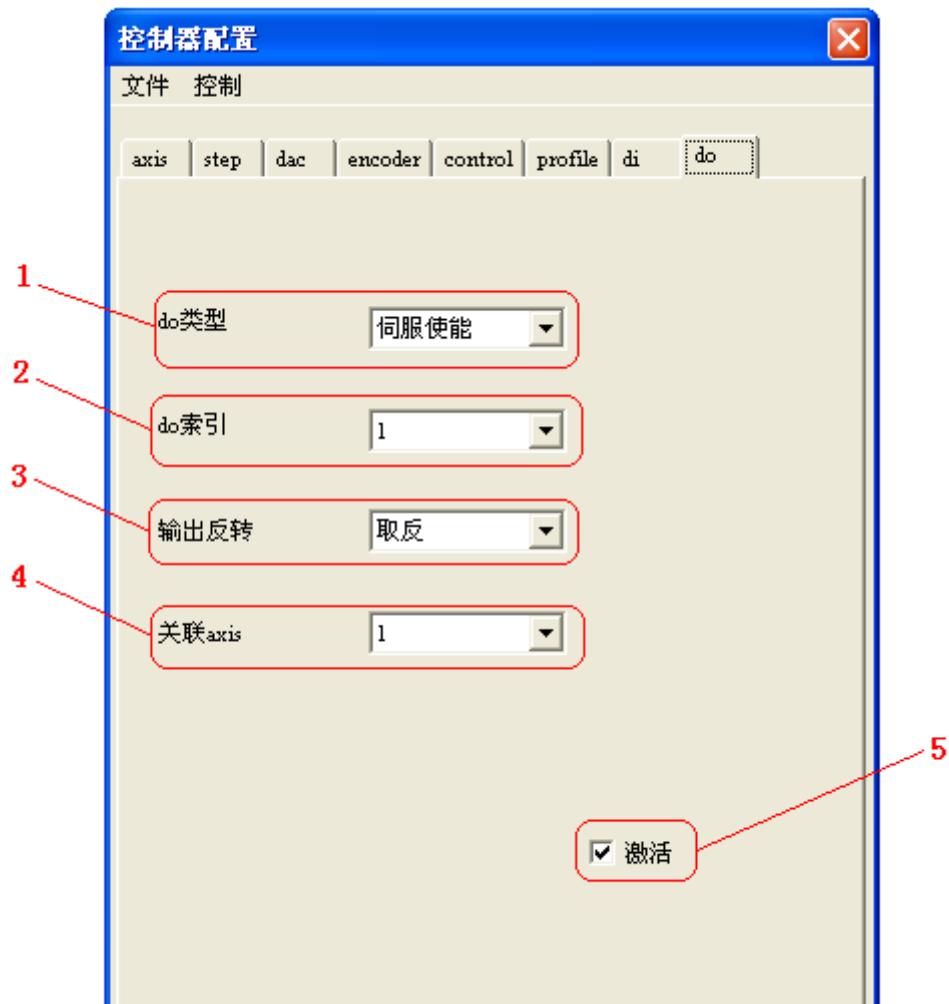


图 4-16 do 配置界面

1. Do 类型选择：选择需要配置的 do 的类型，包括：伺服使能、清除报警、通用输出。



注意

清除报警信号并不是所有驱动器上都有的。即使驱动器上有此信号，此信号也不能清除所有的驱动报警信号。例如，在编码器接线不良引起驱动报警时，就无法使用清除报警信号来清除。用户可以调用指令 `GT_SetDo` 来发出清除报警信号，但是当报警无法清除时，用户需要查询驱动器所提示的报警信号，然后查阅驱动器相关手册，来确定驱动器报警信号的清除方法。

2. Do 索引：选择需要配置的 do 的编号。
3. 输出反转：表示 do 的输出逻辑取反。默认值为：正常，0 表示输出低电平，1 表示输出高电平。输出反转后，0 表示输出高电平，1 表示输出低电平。

4. 关联 axis: 表示该 do 关联到指定 axis 的驱动器使能。默认情况下, 各轴都具有独立的驱动器使能 do 作为输出, 当调用 [GT_AxisOn](#) 指令时, 该 do 置 1。如果驱动器是低电平使能, 必须把“输出反转”设置为取反。



Do 一旦和 axis 关联, 就不能调用 [GT_SetDo](#) 或 [GT_SetDoBit](#) 指令直接设置伺服使能输出电平。如果不需要驱动器使能信号, 可以取消和驱动器使能 do 的关联。取消关联以后, 驱动器使能 do 就可以作为普通 do 来使用, 可以调用 [GT_SetDo](#) 或 [GT_SetDoBit](#) 直接设置其输出电平。

5. Do 激活选项: 如果 do 不被激活, 则该数字量输出将不起作用。默认 do 都是激活的。但是如果没用用到某个 do, 则可以不把该 do 激活, 这样可以节约运动控制器的处理资源。

4.4 配置文件生成和下载

表 4-1 下载配置文件指令

指令	说明	页码
GT_LoadConfig	下载配置信息到运动控制器, 调用该指令后需再调用 GT_ClrSts 才能使该指令生效	209

按照 4.3 节中的描述进行运动控制器的配置之后, 如图 4-17 所示, 在“控制器配置”界面的“文件”菜单, 点击“写入到文件”, 即可对配置信息进行保存, 生成配置文件 (*.cfg)。



图 4-17 生成配置文件界面

用户可以调用 `GT_LoadConfig` 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 `GT_LoadConfig` 指令时，参数需要包含配置文件的绝对路径。



注意

`GT_LoadConfig` 中的文件名长度不允许超过 125 个字，否则调用指令将会失败。

4.5 使用 MCT2008 系统配置举例

本节将以 4.2.3 节的两个例子为基础，介绍在使用 MCT2008 配置的流程。



注意

在调试阶段会频繁使用本流程。提醒用户，调试时请不要将电机与机械本体连接，否则可能会因操作失误而造成机械本体的严重损坏。

4.5.1 开环控制模式

请参考图 4-1，将这一套系统配置到轴 1 上，假设端子板上接了 limit0+，limit0-一对正负限位开关，以及一个输入开关 EXI0，一个输出开关 EXO0。需要配置 5 项内容，分别是 axis，step，profile，di，do。步骤：

- (1) 打开 MCT2008 软件，点击“工具”->“控制器配置”->“axis”页，开环模式下的 axis 配置流程如图 4-18 所示。

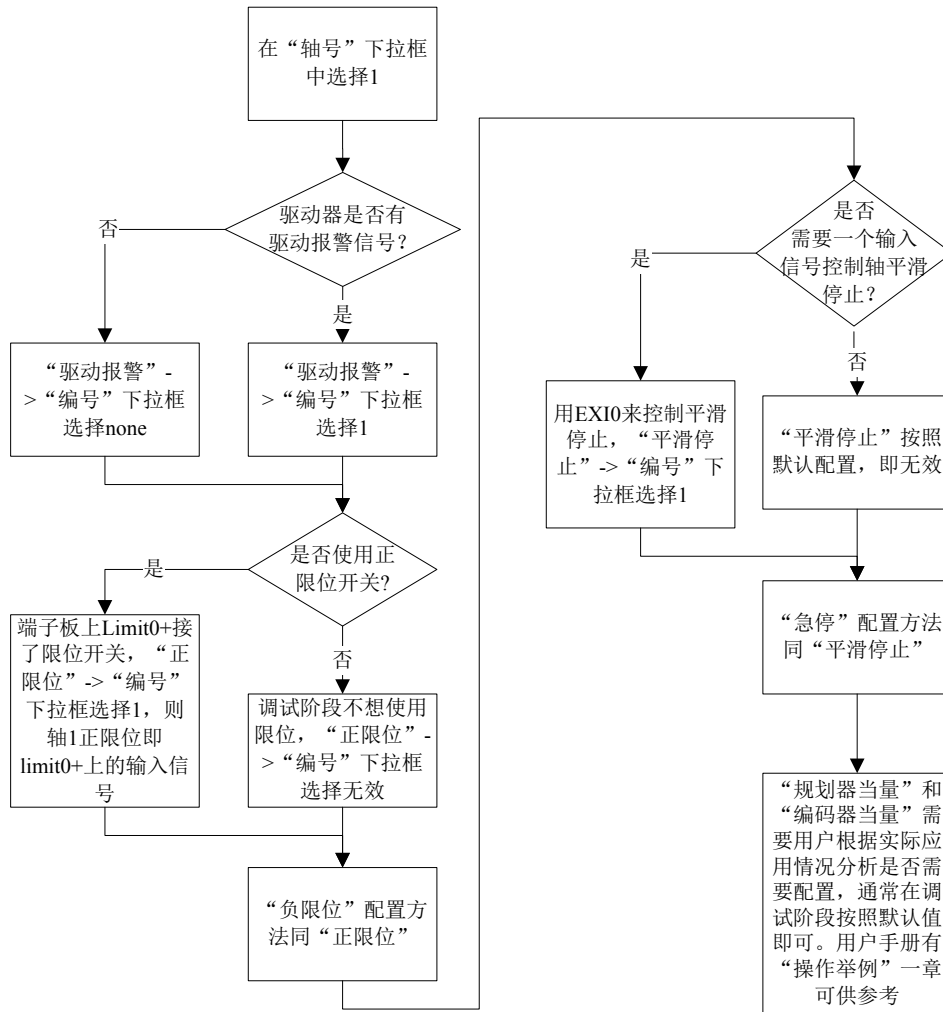


图 4-18 开环模式下 axis 配置流程

- (2) 切换至“step”页。开环模式下 step 配置流程如图 4-19 所示。

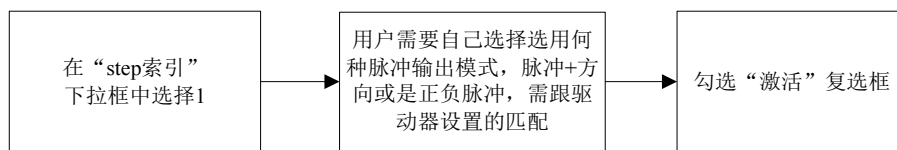


图 4-19 开环模式下 step 配置流程

- (3) 切换至“profile”页。开环模式下 profile 配置流程如图 4-20 所示。

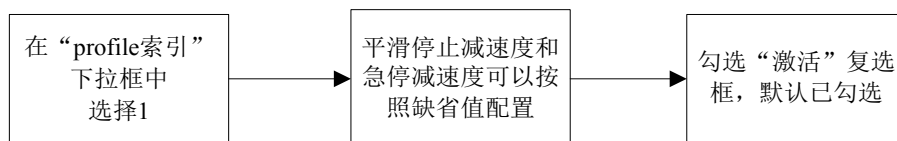


图 4-20 开环模式下 profile 配置流程

- (4) 切换至“di”页。调用 `GT_GetDi` 指令读取某一个输入开关的值时，0 表示输入低电平，1 表示输入高电平，一般情况下按默认值配置即可。

(5) 切换至“do”页。开环模式下 do 配置流程如图 4-21 所示。

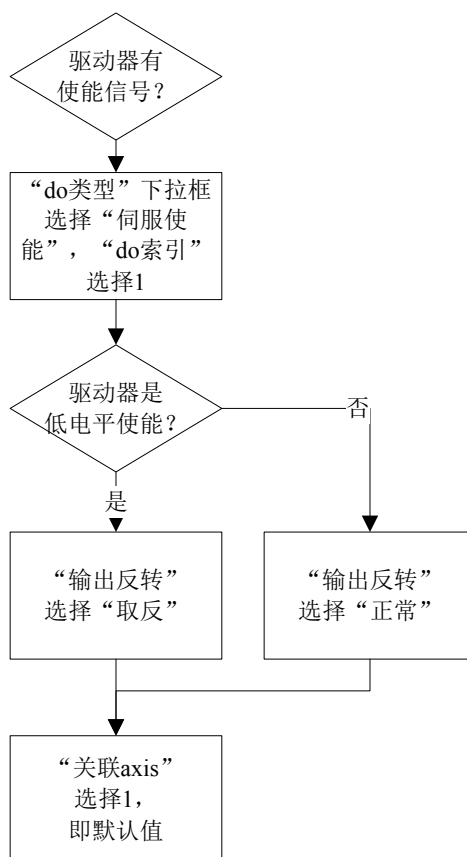


图 4-21 开环模式下 do 配置流程

- (6) 点击“控制器配置”->“文件”->“写入到文件”。命名为 OpenLoop.cfg 文件。
- (7) 在应用程序中调用指令 `GT_LoadConfig("OpenLoop.cfg")`，将配置文件加载到控制器中。完成配置过程。
- (8) 例：要配置第 1 轴控制步进电机，电机的控制模式是“脉冲+方向”，输出的脉冲不经过任何当量转换，没有报警信号，电机使能信号的电平是低电平，限位的触发电平是低电平，无急停和平滑停止信号，在运动过程中只读取控制器发出的脉冲值。配置过程如下：
 - 1) 打开固高科技提供的运动控制器管理软件：MCT2008，启动后的界面如图 4-3 所示。选择“工具”菜单，点击“控制器配置”，打开运动控制器配置面板就可以对系统进行配置。
 - 2) 选择 axis（选项主要用来配置轴控制的相关信息），配置成如图 4-22 所示。

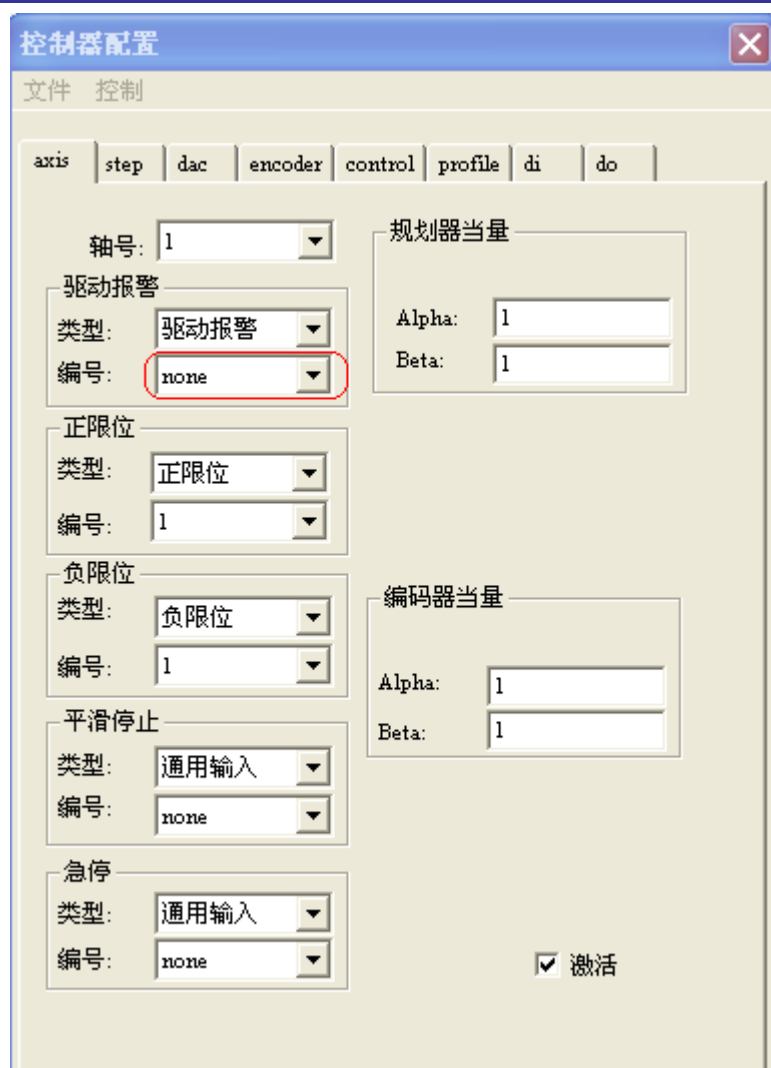


图 4-22 开环配置示例之 axis 配置界面

- (9) 选择 step（选项用来配置轴脉冲输出方式），配置成如图 4-7 所示。此示例中保持默认配置即可。
- (10) 选择 encoder（选项用来配置脉冲计数源、Home 捕获以及 Index 捕获边沿），配置成如图 4-23 所示。

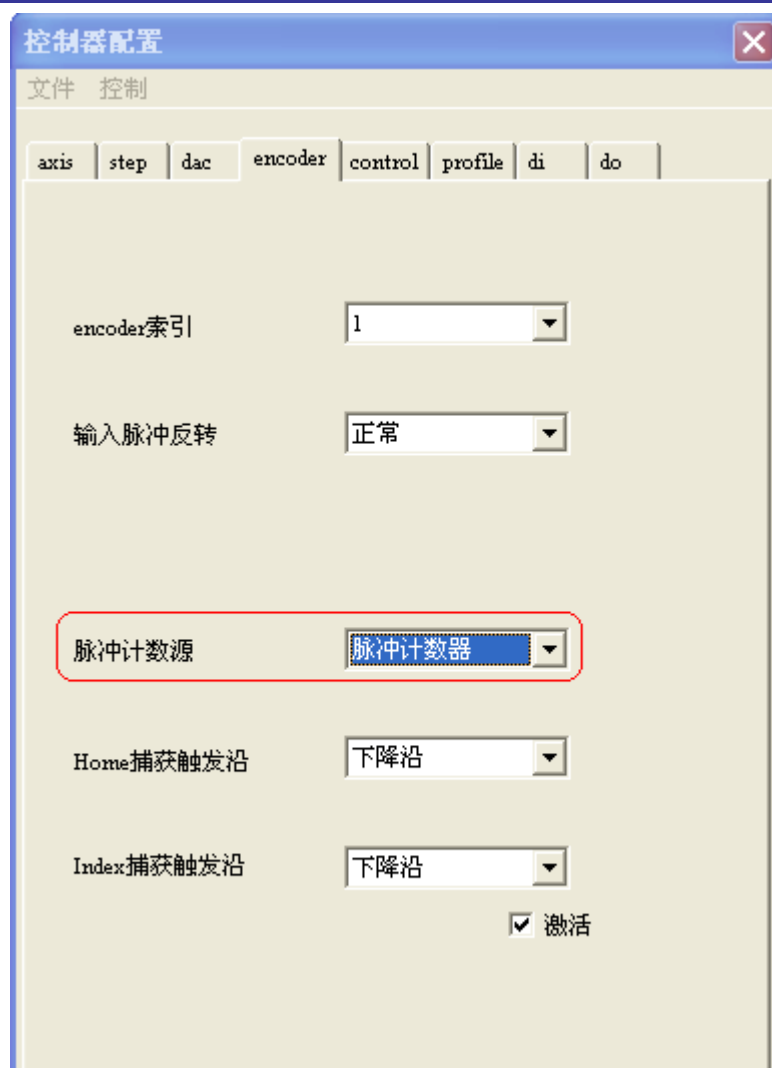


图 4-23 开环配置示例之 encoder 配置界面

- (11) 选择 control（选项用来脱离 PID 闭环控制），配置成如图 4-12 所示。此示例中保持默认配置即可。
- (12) profile 配置，配置成如图 4-14 所示。此示例中保持默认配置即可。
- (13) di/do 配置，配置电机正负限位触发电平为低电平，如图 4-24 和图 4-25 所示；配置电机伺服使能时输出为低电平，如图 4-16 所示。此示例中 do 保持默认配置即可。



图 4-24 开环配置示例之 di 配置界面 1

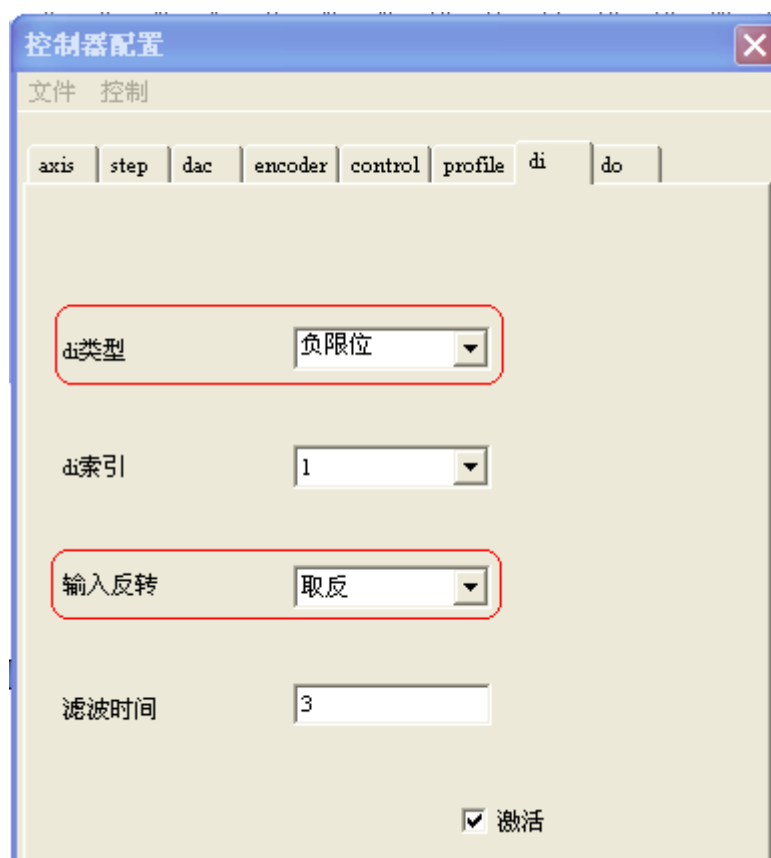


图 4-25 开环配置示例之 di 配置界面 2

(14) 如图 4-17 所示，在“控制器配置”界面的“文件”菜单，点击“写入到文件”，即可对配置信息

进行保存，生成配置文件(*.cfg)。

用户可以调用 `GT_LoadConfig` 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 `GT_LoadConfig` 指令时，参数需要包含配置文件的绝对路径。此时控制器配置完成。

- (15) 检验配置是否成功：把配置写入控制器状态，如图 4-26 所示。



图 4-26 写入控制器状态

- (16) 查看轴状态是否异常，如图切换到 MCT2008 主界面，点击轴状态菜单进入轴状态界面如图 4-27 所示。



图 4-27 点击进入轴状态界面

- (17) 查看轴状态是否正常，如图 4-28 所示，图中已按了“伺服使能按钮”，电机已伺服使能。请按“清除状态”按钮，如果实际状态与下图 4-28 不一致，请重复第一至第十步（即“1”至“10”）查看配置是否正确；如状态还不一致，请查看电气接线上是否存在问题。



图 4-28 轴状态界面

- (18) 单击“视图”->“JOG”菜单项调出 Jog 面板，在“轴号”下拉框中选择控制轴。如图 4-29 所示。

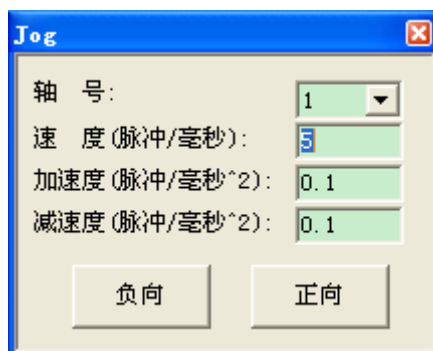


图 4-29 Jog 模块界面

设置合适的运动参数。加速度减速度大于零，速度可正可负。按下“负向”或者“正向”按钮，此时电机开始运动。

如果控制电机没有运动，请先调出相应轴的状态面板，确认该轴不存在报警、限位，并且已经上伺服。如果状态正常，且规划位置会变化，则需要确认驱动器和控制器的控制模式是否匹配，电气接线上是否存在问题。

4.5.2 闭环控制模式

请参考图 4-2，将这一套系统配置到轴 1 上，假设端子板上接了 limit0+，limit0-一对正负限位开关，以及一个输入开关 EXI0，一个输出开关 EXO0。需要配置 7 项内容，分别是 axis，dac，encoder，control，profile，di，do。步骤如下。

- (1) 打开 MCT2008 软件，点击“工具”->“控制器配置”->“axis”页，如图 4-30 所示。

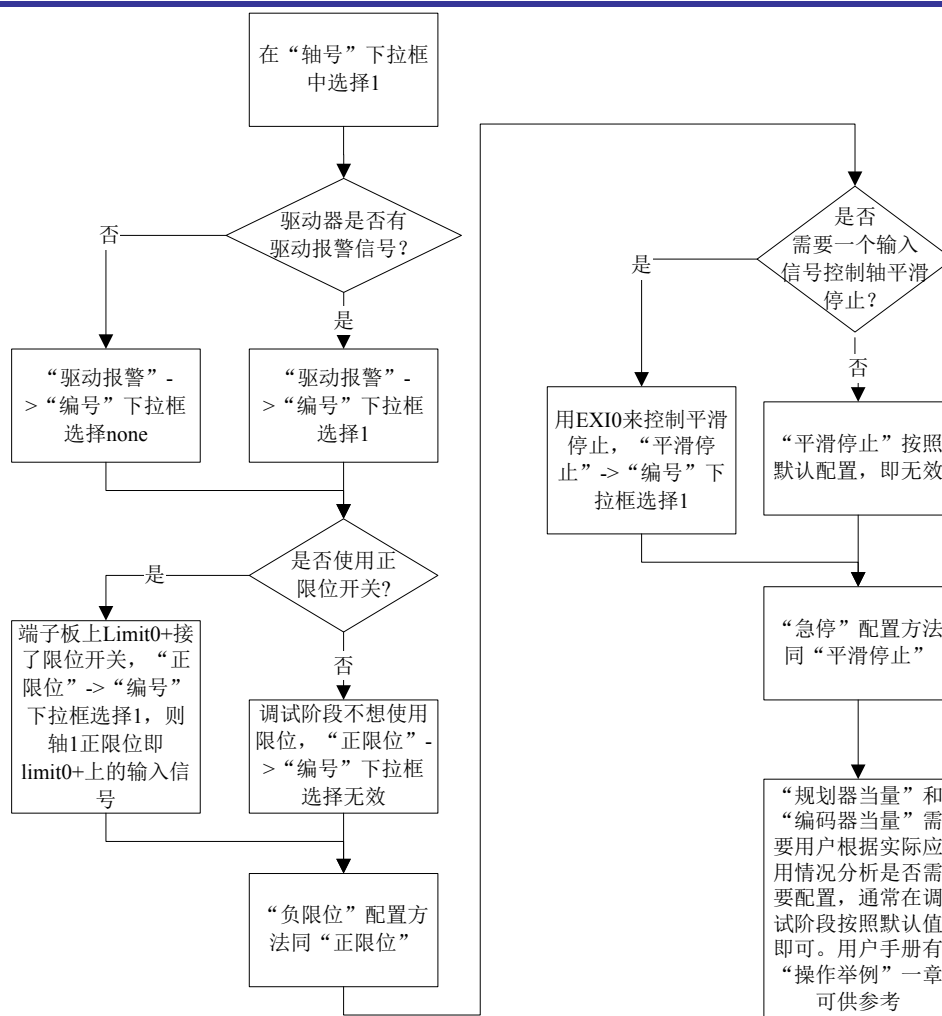


图 4-30 闭环模式下 axis 配置流程

(2) 切换到“step”页。取消勾选“激活”复选框。

(3) 切换至“dac”页，如图 4-31 所示。

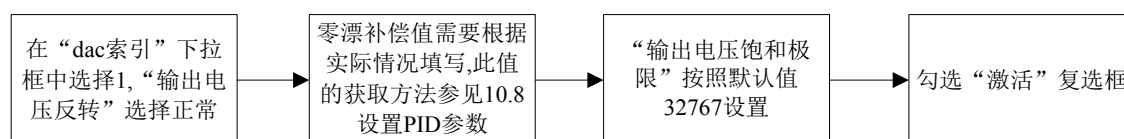


图 4-31 闭环模式下 dac 配置流程

(4) 切换至“encoder”页，如图 4-32 所示。

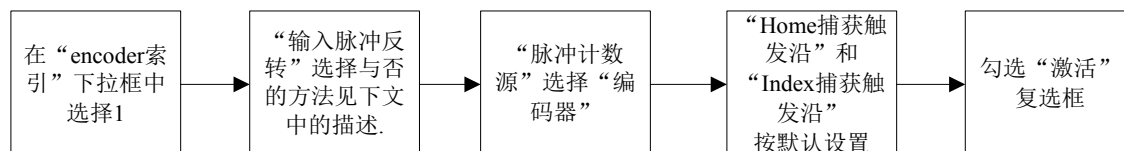


图 4-32 闭环模式下 encoder 配置流程

(5) “输入脉冲反转”与否的确认方法：

1) 将驱动器设置为电压方式，注意接好电机，但是电机不要连接任何机械本体，以免发

生危险；

- 2) 打开 MCT2008，将控制器配置在开环模式(脉冲方式)下，打开“视图”，选择“轴状态”，清除状态后，伺服使能（此时电机可能会有缓慢转动，这是由于零漂引起的，可以暂时不用理会）；
 - 3) 打开“视图”，选择“电压输出”，填入一个较小的电压值（如 0.1V，约 30r/min，一般电压与速度间对应关系如下：1V~300r/min，用户需要根据实际情况确定此值），并输出；
 - 4) 观察实际位置（也即编码器位置）的变化，如果电压设为正值时，实际位置也在正向增加，则“输入脉冲反转”选“正常”；反之若实际位置反向增加，则选“取反”；
 - 5) 选择完成后，再重新配置，确认给定正电压时，实际位置正相增加；给定负电压时，实际位置反向增加。
- (6) 切换至“control”页，如图 4-33 所示。

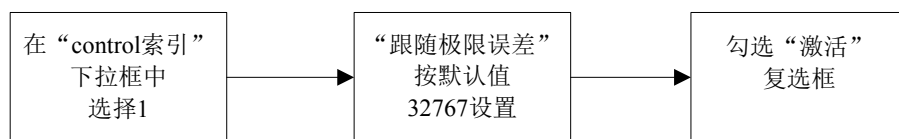


图 4-33 闭环模式下 control 配置流程

- (7) 切换至“profile”页，如图 4-34 所示。

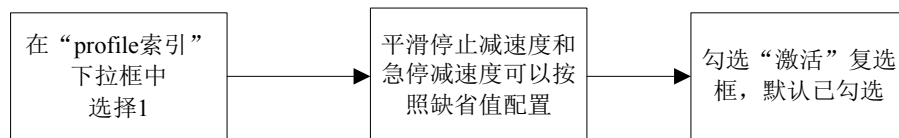


图 4-34 闭环模式下 profile 配置流程

- (8) 切换至“di”页。调用 `GT_GetDi` 指令读取某一个输入开关的值时，0 表示输入低电平，1 表示输入高电平，一般情况下按默认值配置即可。
- (9) 切换至“do”页，如图 4-35 所示。

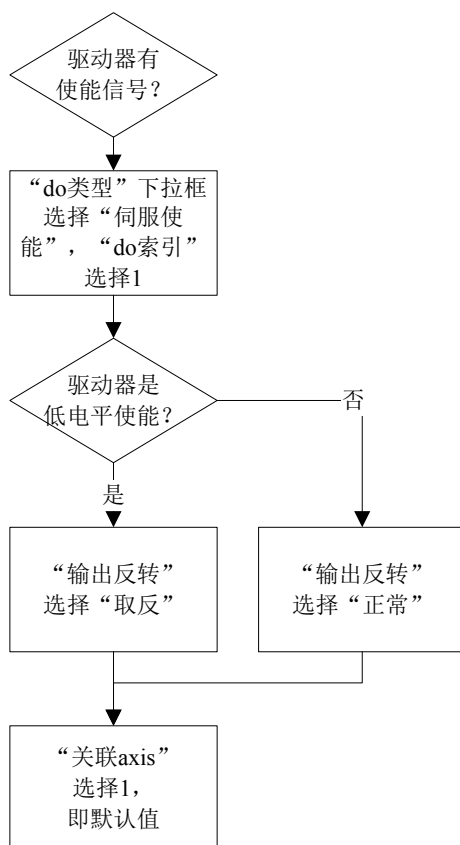


图 4-35 闭环模式下 do 配置流程

- (10) 点击“控制器配置”->“文件”->“写入到文件”。命名为 CloseLoop.cfg 文件。
- (11) 在应用程序中调用指令 `GT_LoadConfig("CloseLoop.cfg")`，将配置文件加载到控制器中。完成配置过程。
- (12) 注意，闭环模式下还需要设置 pid 参数，才能伺服使能。点击“视图”->“PID”，选择“轴号”为 1，将“比例增益”设为正数，点击“更新”。
- (13) 例：要配置第一轴闭环控制伺服电机，电机的控制模式是速度，脉冲不经过任何当量转换，报警信号触发电平是高电平，电机使能信号的电平是低电平，限位的触发电平是低电平，无急停和平滑停止信号。配置过程如下：
 - 1) 打开固高科技提供的运动控制器管理软件：MCT2008，启动后的界面如图 4-3 所示。选择“工具”菜单，点击“控制器配置”，打开运动控制器配置面板就可以对系统进行配置。
 - 2) 选择 axis（选项主要用来配置轴控制的相关信息），配置成如图 4-5 所示。此示例中保持默认配置即可。
 - 3) 选择 dac（选项来激活模拟电模输出），配置成如图 4-8 所示，也即默认配置。如果电机正反馈，可以通过修改“输出电压反转”，使电机控制正常，如图 4-36 所示。

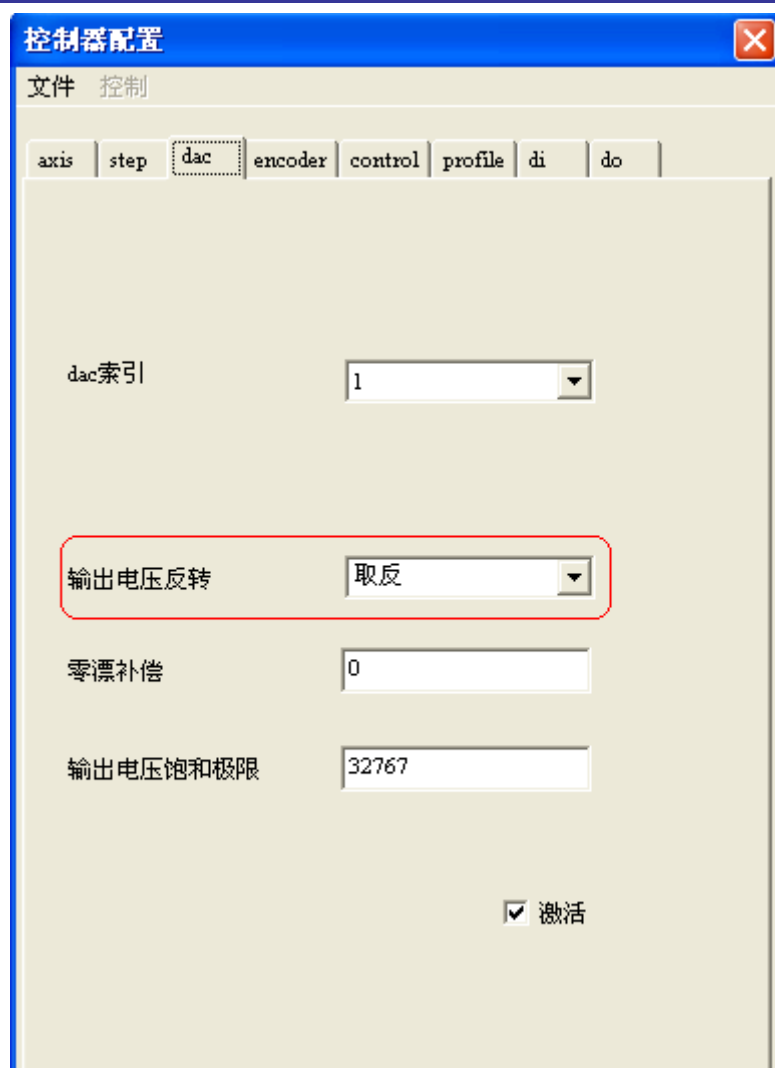


图 4-36 闭环配置示例之 dac 配置界面

- (14) 选择 encoder（选项用来配置脉冲计数源以及 Home 捕捉以及 Index 捕捉边沿），配置成如图 4-9 所示，也即默认配置。如果电机正反馈，可以修改“输入脉冲反转”为“正常”（默认为“取反”），使电机控制正常，如图 4-37 所示。

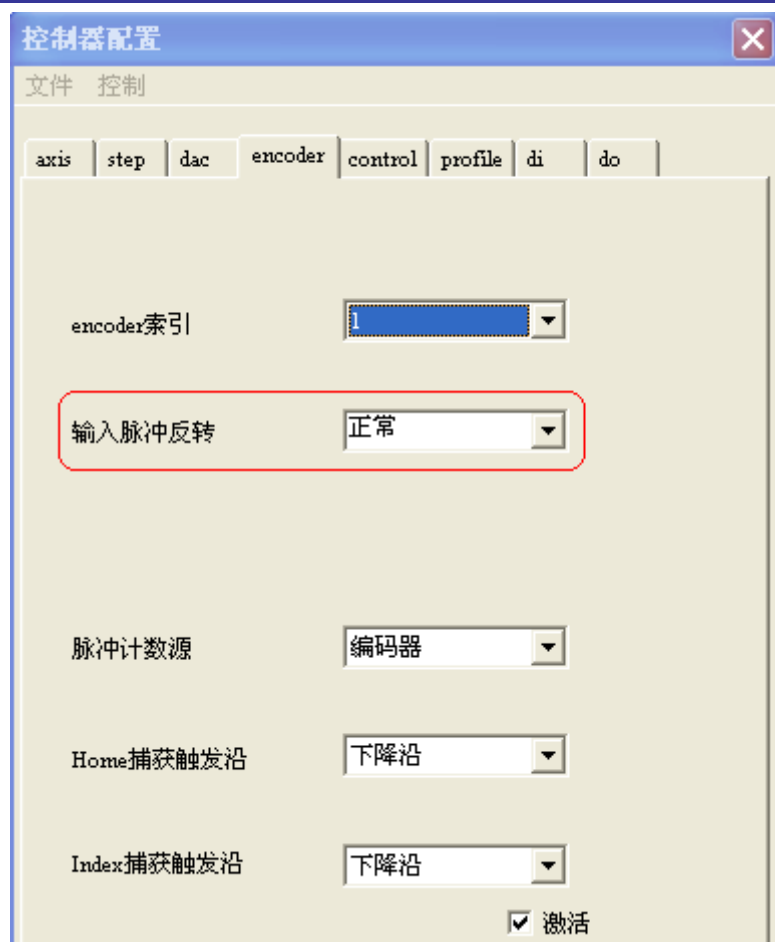


图 4-37 闭环配置示例之 encoder 配置界面

(15) 选择 control（选项用来关联 PID 闭环控制），配置成如图 4-38 所示。



图 4-38 闭环配置示例之 control 配置界面

- (16) profile 配置。配置成如图 4-14 所示。此示例中保持默认配置即可。
- (17) di/do 配置，配置电机正负限位触发电平为低电平，如图 4-24 和图 4-25 所示；配置电机伺服使能时输出为低电平，如图 4-16 所示。此示例中 do 保持默认配置即可。
- (18) 如图 4-17 所示，在“控制器配置”界面的“文件”菜单，点击“写入到文件”，即可对配置信息进行保存，生成配置文件(*.cfg)。

用户可以调用 [GT_LoadConfig](#) 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 [GT_LoadConfig](#) 指令时，参数需要包含配置文件的绝对路径。此时控制器配置完成。

- (19) 检验配置是否成功：把配置写入控制器状态，如图 4-26 所示。

点击“视图”->“PID”，调出 PID 面板，轴号选择为 1 轴，比例增益设置为适当的值（比如 3），其他保持默认值，最后点击“更新”，第一轴就是模拟量控制模式了，如图 4-39 所示。

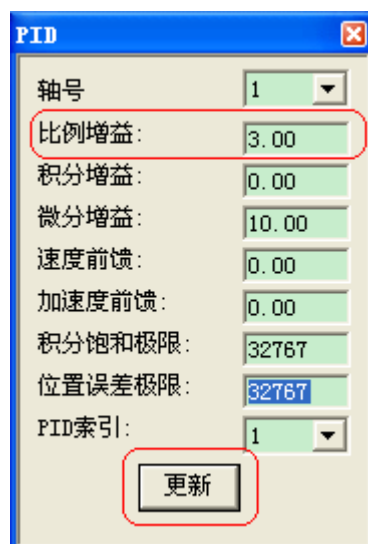


图 4-39 PID 设置界面

- (20) 查看轴状态是否异常，如图切换到 MCT2008 主界面，点击轴状态菜单进入轴状态界面如图 4-27 示。
- (21) 查看轴状态是否正常，如图 4-28 所示，图中已按了“伺服使能按钮”，电机已伺服使能。请按“清除状态”按钮，如果实际状态与图 4-28 不一致，请重复第一至第十一步（即“1”至“11”）查看配置是否正确；如状态还不一致，请查看电气接线上是否存在问题。
- (22) 单击“视图”——“JOG”菜单项调出 Jog 面板，在“轴号”下拉框中选择控制轴。如图 4-29 所示。

设置合适的运动参数。加速度减速度大于零，速度可正可负。按下“负向”或者“正向”按钮，此时电机开始运动。

如果控制电机没有运动，请先调出相应轴的状态面板，确认该轴不存在报警、限位，并且已经上伺服。如果状态正常，且规划位置会变化，则需要确认驱动器和控制器的控制模式是否匹配，电气接线上是否存在问题。

4.6 配置信息修改指令

用户除了可以使用上面所述的配置文件的方式实现运动控制器的初始化配置外，还可以使用指令的方式来实现初始化配置。

4.6.1 指令列表

表 4-2 配置信息修改指令列表

指令	说明	页码
GT_AlarmOff	控制轴驱动报警信号无效	163
GT_AlarmOn	控制轴驱动报警信号有效	163
GT_LmtsOn	控制轴限位信号有效	206
GT_LmtsOff	控制轴限位信号无效	205
GT_ProfileScale	设置控制轴的规划器当量变换值	211
GT_EncScale	设置控制轴的编码器当量变换值	180

指令	说明	页码
GT_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”	227
GT_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”	227
GT_StepOrthogonal	将脉冲输出通道的脉冲输出模式设置为“正交脉冲”	228
GT_SetMtrBias	设置模拟量输出通道的零漂电压补偿值	222
GT_GetMtrBias	读取模拟量输出通道的零漂电压补偿值	196
GT_SetMtrLmt	设置模拟量输出通道的输出电压饱和极限值	222
GT_GetMtrLmt	读取模拟量输出通道的输出电压饱和极限值	196
GT_EncSns	设置编码器的计数方向	181
GT_EncOn	设置为“外部编码器”计数方式	180
GT_EncOff	设置为“脉冲计数器”计数方式	180
GT_SetPosErr	设置跟随误差极限值	223
GT_GetPosErr	读取跟随误差极限值	197
GT_SetStopDec	设置平滑停止减速度和急停减速度	224
GT_GetStopDec	读取平滑停止减速度和急停减速度	200
GT_LmtSns	设置运动控制器各轴限位开关触发电平	205
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出	179
GT_SetStopIo	设置平滑停止和紧急停止数字量输入的信息	225
GT_GpiSns	设置运动控制器数字量输入的电平逻辑	204
GT_SetAdcFilter	设置模拟量输入的滤波器时间参数	212

4.6.2 重点说明

1. 编码器计数方向设置

指令 `GT_EncSns` 可以修改运动控制器各编码器的计数方向，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 4-3 所示。

表 4-3 `GT_EncSns` 指令参数状态位定义

状态位	8	7	6	5	4	3	2	1	0
编码器	辅助编码器	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1

例程 4-1 修改编码器计数方向

修改编码器 1 和 4 的计数方向为正向，其他编码器（包括辅助编码器）的计数方向取反。状态位应按表 4-4 所示。

表 4-4 例程 4-1 中的编码器计数方向设定表

状态位	8	7	6	5	4	3	2	1	0
编码器	辅助编码器	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1
位值（二进制）	1	1	1	1	1	0	1	1	0

```

.....
// 返回值变量
short sRtn;
// 状态值变量
unsigned short sValue;

```

```

// 初始化使所有位都为1
sValue = 0xffff;
// 使第0位为0，即轴1所对应的位
sValue&= ~(1);
// 使第3位为0，即轴4所对应的位
sValue&= ~(1<<3);
// 设置编码器计数方向
sRtn = GT_EncSns(sValue);
.....

```

2. 设置限位开关触发电平

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表 4-5 所示。

表 4-5 `GT_LmtSns` 指令参数状态位定义

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	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+

例程 4-2 修改限位开关触发电平

修改轴 1，轴 4 的正负限位为低电平触发。状态位应按下表 4-6 所示：

表 4-6 例程 3-2 限位开关设定表

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	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+
位值（二进制）	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

```

.....
// 返回值变量
short sRtn;
// 状态值变量
unsigned short sValue;
// 初始化使所有位都为0
sValue = 0;
// 使第0, 1位为1，即轴1正负限位所对应的位
sValue |= 0x3;
// 使第6, 7位为1，即轴4正负限位所对应的位
sValue |= 1<<6;
sValue |= 1<<7;
// 设置限位开关触发电平
sRtn = GT_LmtSns(sValue);

```

4.6.3 例程

1. 开环（脉冲）控制模式

- (1) 打开运动控制器，`GT_Open`；
- (2) 复位运动控制器，`GT_Reset`。复位后默认的控制模式为“脉冲+方向”的脉冲控制方式。若不是采用“脉冲+方向”的控制方式，则可调用 `GT_StepPulse` 修改；当需要还原为“脉冲+方向”的控制方式时，则可调用 `GT_StepDir` 指令（补充说明：可以调用指令 `GT_StepOrthogonal` 将脉冲输出模式设置为正交脉冲）；
- (3) 检查相关轴驱动器报警信号有没有连接。（一般若采用步进电机，可能没有驱动器报警信号），若没有连接，则应该调用 `GT_AlarmOff` 指令，使驱动器报警无效，默认是有效的；
- (4) 检查相关轴的限位开关，若没有连接，则需要通过调用 `GT_LmtsOff`，使限位无效，默认是有效的；若有连接，则要检查触发电平是否设置正确，可通过 `GT_LmtSns` 指令修改；
- (5) 在确认前面两步操作之后，调用 `GT_ClrSts`，更新设置的状态；
- (6) 调用 `GT_AxisOn`，使能驱动器，这样相应的电机便能工作了；
- (7) 使轴运动，运动后，若出现编码器位置和规划位置方向不一致，则可通过调用 `GT_EncSns` 改变编码器的计数方向。

例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式

```

.....
// 指令返回值变量
short sRtn;
// 打开运动控制器
sRtn = GT_Open();
// 指令返回值检测，请查阅例程3-1
commandhandler("GT_Open", sRtn);
// 系统复位
sRtn = GT_Reset();
commandhandler("GT_Reset", sRtn);
// 配置轴1脉冲输出方式为脉冲+方向
sRtn = GT_StepDir(1);
commandhandler("GT_StepDir", sRtn);
// 配置轴1报警输出无效
sRtn = GT_AlarmOff(1);
commandhandler("GT_AlarmOff", sRtn);
// 配置轴1正负限位无效
sRtn = GT_LmtsOff(1, -1);
commandhandler("GT_LmtsOff", sRtn);
// 配置完成后要更新状态
sRtn = GT_ClrSts(1);
commandhandler("GT_ClrSts", sRtn);

```

```
// 轴1伺服使能
sRtn = GT_AxisOn(1);
commandhandler("GT_AxisOn", sRtn);
.....
```

2. 闭环（模拟量）控制模式

- (1) 打开运动控制器，GT_Open;
- (2) 复位运动控制器，GT_Reset。复位后的控制模式为“脉冲+方向”的脉冲控制方式，因此需要调用 GT_CtrlMode 来修改相应轴的控制模式；
- (3) 检查相关轴驱动器报警信号有没有连接。（一般若采用步进电机，可能没有驱动器报警信号），若没有连接，则应该调用 GT_AlarmOff 指令，使驱动器报警无效，默认是有效的；
- (4) 检查相关轴的限位开关，若没有连接，则需要通过调用 GT_LmtsOff，使限位无效，默认是有效的；若有连接，则要检查触发电平是否设置正确，可通过 GT_LmtSns 指令修改；
- (5) 在确认前面两步操作之后，调用 GT_ClrSts，更新设置的状态；
- (6) 设置 Pid 参数，通过调用 GT_SetPid，将相应轴的 Pid 参数中的 Kp 设置为大于 0 的数，如 Kp=1；
- (7) 调用 GT_AxisOn，使能驱动器，若出现飞车现象（注意，调试状态下不要接任何机械本体），则可通过调用 GT_EncSns 改变编码器的计数方向，最后再使能驱动器，这样相应的电机便能工作了。

例程 4-4 设置第 1 轴为闭环控制方式

```
.....
// 指令返回值变量
short sRtn;
// PID结构体变量
TPid pid;
// 打开运动控制器
sRtn = GT_Open();
// 指令返回值检测，请查阅例程3-1
commandhandler("GT_Open", sRtn);
// 系统复位
sRtn = GT_Reset();
commandhandler("GT_Reset", sRtn);
// 配置轴1为模拟量输出模式
sRtn = GT_CtrlMode(1, 0);
commandhandler("GT_CtrlMode", sRtn);
// 配置轴1报警输出无效
sRtn = GT_AlarmOff(1);
commandhandler("GT_AlarmOff", sRtn);
// 配置轴1正负限位无效
sRtn = GT_LmtsOff(1, -1);
commandhandler("GT_LmtsOff", sRtn);
```

```

// 配置完成后要更新状态
sRtn = GT_ClrSts(1);
commandhandler("GT_ClrSts", sRtn);
// 获取当前pid参数
sRtn = GT_GetPid(1, 1, &pid);
commandhandler("GT_GetPid", sRtn);
// 调试阶段，只需修改一下kp到一个较小的值
pid.kp = 1;
// 设置PID参数
sRtn = GT_SetPid(1, 1, &pid);
commandhandler("GT_SetPid", sRtn);
// 轴1伺服使能
sRtn = GT_AxisOn(1);
commandhandler("GT_AxisOn", sRtn);
// 伺服使能后若出现飞车，则需要通过调用GT_EncSns指令来修改编码器计数方向
// 等待伺服稳定
Sleep(200);
.....

```



注意

通过指令配置控制器之后，用户必须调用指令 `GT_ClrSts` 更新状态，使得配置生效。很多初次使用的客户会容易忘记这一点。

4.7 控制器配置初始化状态

控制器初始化状态，是指调用指令 `GT_Open` 成功后，调用指令 `GT_Reset` 复位后的状态。此时所有的状态都为默认状态。想要让控制器回到初始状态，只要调用 `GT_Reset` 指令即可。

表 4-7 所示为复位后，控制器配置选项的默认状态，调用第四栏“相关指令”中的指令可以修改对应选项的状态。

表 4-7 控制器配置初始化状态

资源	配置选项	默认状态	相关指令
axis	该资源是否激活	激活	/
	规划器当量	Alpha 和 Beta 都为 1	<code>GT_ProfileScale</code>
	编码器当量	Alpha 和 Beta 都为 1	<code>GT_EncScale</code>
	平滑停止和急停数字输入信号	不配置	<code>GT_SetStopIo</code>
	正负限位输入	有效，编号与轴号对应	<code>GT_LmtsOn</code> <code>GT_LmtsOff</code>
	驱动报警输出	有效，编号与轴号对应	<code>GT_AlarmOff</code>
step	该资源是否激活	激活	/
	脉冲输出方式	脉冲+方向 正交脉冲	<code>GT_StepDir</code> <code>GT_StepPulse</code> <code>GT_StepOrthogonal</code>
dac	该资源是否激活	激活	-

资源	配置选项	默认状态	相关指令
	DAC 输出电压零漂补偿	0	GT_SetMtrBias
	DAC 输出电压反转	正常	/
	DAC 输出电压饱和极限	32767	GT_SetMtrLmt
encoder	该资源是否激活	激活	/
	脉冲计数源	外部编码器	GT_EncOn GT_EncOff
	输入脉冲反转	取反	GT_EncSns
	Home 及 Index 捕获触发沿	下降沿	GT_SetCaptureSense
control	该资源是否与轴关联	不关联	/
	跟随误差极限	32767	GT_SetPosErr
profile	该资源是否激活	激活	/
	平滑停止减速度	1 pulse/ms ²	GT_SetStopDec
	急停的减速度	1000 pulse/ms ²	
di	该资源是否激活	激活	/
	限位开关	常闭开关，常为低电平，高电平限位触发	GT_LmtSns
	所有 DI 输入反转	正常	GT_GpiSns
	所有 DI 滤波时间	3，即 750μs	/
	所有 DI 状态	常为低电平	/
do	该资源是否激活	激活	/
	所有 DO 输出反转	正常，1 代表输出高电平，0 代表输出低电平	/
	驱动报警触发电平	1 代表输出高电平，0 代表输出低电平	/

第5章 运动状态检测

5.1 本章简介

当用户连接好一整套系统后（包括运动控制器，驱动器，电机），如何查看这套系统的运动状态？控制器可以帮助用户在应用程序中查看驱动器报警，当前运动位置，运动速度和加速度等一系列状态参数。本章将介绍用户可以检测到哪些状态以及如何检测。



本手册中所有字体为蓝色的指令（如 [GT_Prfrap](#)）均带有超级链接，点击可跳转至指令说明。

5.2 指令列表

表 5-1 运动状态检测指令列表

指令	说明	页码
GT_GetSts	读取轴状态	200
GT_ClrSts	清除驱动器报警标志、跟随误差超限标志、限位触发标志 1. 只有当驱动器没有报警时才能清除轴状态字的报警标志 2. 只有当跟随误差正常以后，才能清除跟随误差超限标志 3. 只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志	173
GT_GetPrfMode	读取轴运动模式	198
GT_GetPrfPos	读取规划位置	198
GT_GetPrfVel	读取规划速度	199
GT_GetPrfAcc	读取规划加速度	198
GT_GetAxisPrfPos	读取轴(axis)的规划位置值	184
GT_GetAxisPrfVel	读取轴(axis)的规划速度值	184
GT_GetAxisPrfAcc	读取轴(axis)的规划加速度值	184
GT_GetAxisEncPos	读取轴(axis)的编码器位置值	183
GT_GetAxisEncVel	读取轴(axis)的编码器速度值	183
GT_GetAxisEncAcc	读取轴(axis)的编码器加速度值	182
GT_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值	183
GT_Stop	停止一个或多个轴的规划运动，停止坐标系运动	228

5.3 重点说明

5.3.1 轴状态定义

用户可以从控制器的运动状态寄存器中读取轴状态。当调用 [GT_GetSts](#) 指令时，将返回一个 32 位的轴状态字，该轴状态字的定义如表 5-2 所示。

表 5-2 轴状态定义

位	定义
0	保留
1	驱动器报警标志 控制轴连接的驱动器报警时置 1
2	保留
3	保留
4	跟随误差超限标志 控制轴规划位置 and 实际位置的误差大于设定极限时置 1
5	正限位触发标志 正限位开关电平状态为限位触发电平时置 1 规划位置大于正向软限位时置 1
6	负限位触发标志 负限位开关电平状态为限位触发电平时置 1 规划位置小于负向软限位时置 1
7	IO 平滑停止触发标志 如果轴设置了平滑停止 IO，当其输入为触发电平时置 1，并自动平滑停止该轴
8	IO 急停触发标志 如果轴设置了急停 IO，当其输入为触发电平时置 1，并自动急停该轴
9	电机使能标志 电机使能时置 1
10	规划运动标志 规划器运动时置 1
11	电机到位标志 规划器静止，规划位置 and 实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志
12~31	保留

驱动器报警标志、限位触发标志、IO 停止、跟随误差超限标志触发以后，不会自动清 0。只有当产生异常的原因已经消除以后，调用 [GT_ClrSts](#) 指令才能清除相应的异常标志。

规划运动状态(bit10)只表示理论上的运动状态。置 1 表示处于规划运动状态，清 0 表示处于规划静止状态。由于电机跟随滞后、机械系统震荡等原因，一般在规划静止一段时间以后，机械系统才能完全停止。

电机到位标志(bit11)表示实际到位状态。该标志位是电机到位检测功能的一部分。控制器复位后，默认该功能未开启。若要开启该功能，请参考 11.7 节的详细说明。该标志位置 1 表示已经处于规划静止状态(bit10=0)，并且规划位置 and 编码器位置的误差在设定的误差带内保持了设定时间。当规划运动或者规划位置 and 编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长，影响加工效率。

5.3.2 轴的运动参数

调用 [GT_GetPrfMode](#) 可以读取当前轴的运动模式。运动模式将在“第 6 章 运动模式”中详细介绍。控制器有如下几种运动模式：点位运动模式、Jog 模式、PT 模式、电子齿轮模式、Follow 模式、

插补运动模式、PVT 模式，其中 PT 模式、Follow 模式和 PVT 模式详见《运动控制器编程手册之高级功能》。

“4.2.2 软件资源”中介绍过规划器的概念。规划器是位于控制器内部的，是根据用户所设置的运动模式和运动参数计算规划位置和规划速度的软件资源。它不代表电机系统的位置和速度。调用 `GT_GetPrfPos`，`GT_GetPrfVel`，`GT_GetPrfAcc` 可以读取规划器当前的位置，速度和加速度。

“4.2.2 软件资源”中介绍过轴的概念。轴亦是位于控制器内部的，是将规划器和编码器硬件资源整合起来的软件资源。轴的规划位置是规划器位置经过当量变换后的值，轴的编码器位置也是编码器硬件的位置经过当量变换后的值。默认情况下，规划器位置的当量与轴的规划位置的当量之比是 1:1，轴的编码器位置当量与编码器硬件的位置当量之比也是 1:1。调用 `GT_GetAxisPrfPos`，`GT_GetAxisPrfVel`，`GT_GetAxisPrfAcc` 可以读取轴的规划器当前的位置，速度和加速度。调用 `GT_GetAxisEncPos`，`GT_GetAxisEncVel`，`GT_GetAxisEncAcc` 可以读取轴的编码器当前的位置，速度和加速度。调用 `GT_GetAxisError` 读取轴的规划位置和轴的编码器位置的差值。

调用 `GT_Stop` 停止正在运动的轴。停止方式可以分为急停和平滑停止。具体介绍请参考“8.3 平滑停止和急停”。

5.4 例程

例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度

```
#define AXIS 1 // 定义轴号
int main(int argc, char* argv[])
{
    short bFlagAlarm = FALSE; // 伺服报警标志
    short bFlagMError = FALSE; // 跟随误差超限标志
    short bFlagPosLimit = FALSE; // 正限位触发标志
    short bFlagNegLimit = FALSE; // 负限位触发标志
    short bFlagSmoothStop = FALSE; // 平滑停止标志
    short bFlagAbruptStop = FALSE; // 急停标志
    short bFlagServoOn = FALSE; // 伺服使能标志
    short bFlagMotion = FALSE; // 规划器运动标志
    double dPrfPos; // 规划位置
    double dPrfVel; // 规划速度
    double dPrfAcc; // 规划加速度
    long lPrfMode; // 运动模式
    char chPrfMode[20]; // 运动模式，字符串变量
    short sRtn; // 指令返回值变量
    long lAxisStatus; // 轴状态

    // 打开运动控制器
    sRtn = GT_Open();
    // 指令返回值检测，请查阅例程3-1
    commandhandler("GT_Open", sRtn);
    // 系统复位
    sRtn = GT_Reset();
    commandhandler("GT_Reset", sRtn);
```

```
// 读取轴状态
sRtn = GT_GetSts(AXIS, &lAxisStatus);
commandhandler("GT_GetSts", sRtn);
```

```
// 伺服报警标志
if (lAxisStatus& 0x2)
{
    bFlagAlarm = TRUE;
    printf("伺服报警\n");
}
else
{
    bFlagAlarm = FALSE;
    printf("伺服正常\n");
}
```

```
// 跟随误差超限标志
if (lAxisStatus& 0x10)
{
    bFlagMError = TRUE;
    printf("运动出错\n");
}
else
{
    bFlagMError = FALSE;
    printf("运动正常\n");
}
```

```
// 正向限位
if (lAxisStatus& 0x20)
{
    bFlagPosLimit = TRUE;
    printf("正限位触发\n");
}
else
{
    bFlagPosLimit = FALSE;
    printf("正限位未触发\n");
}
```

```
// 负向限位
if (lAxisStatus& 0x40)
{
    bFlagNegLimit = TRUE;
    printf("负限位触发\n");
}
else
{
    bFlagNegLimit = FALSE;
```

```
    printf("负限位未触发\n");
}
// 平滑停止
if (lAxisStatus & 0x80)
{
    bFlagSmoothStop = TRUE;
    printf("平滑停止触发\n");
}
else
{
    bFlagSmoothStop = FALSE;
    printf("平滑停止未触发\n");
}
// 急停标志
if (lAxisStatus & 0x100)
{
    bFlagAbruptStop = TRUE;
    printf("急停触发\n");
}
else
{
    bFlagAbruptStop = FALSE;
    printf("急停未触发\n");
}
// 伺服使能标志
if (lAxisStatus & 0x200)
{
    bFlagServoOn = TRUE;
    printf("伺服使能\n");
}
else
{
    bFlagServoOn = FALSE;
    printf("伺服关闭\n");
}
// 规划器正在运动标志
if (lAxisStatus & 0x400)
{
    bFlagMotion = TRUE;
    printf("规划器正在运动\n");
}
else
{
    bFlagMotion = FALSE;
    printf("规划器已停止\n");
}
```

```

// 读取运动数据
sRtn = GT_GetPrfPos(Axis, &dPrfPos);
commandhandler("GT_GetPrfPos", sRtn);
printf("规划位置 %8.2f\n", dPrfPos);
sRtn = GT_GetPrfVel(Axis, &dPrfVel);
commandhandler("GT_GetPrfVel", sRtn);
printf("规划速度 %8.2f\n", dPrfVel);
sRtn = GT_GetPrfAcc(Axis, &dPrfAcc);
commandhandler("GT_GetPrfAcc", sRtn);
printf("规划加速度 %8.2f\n", dPrfAcc);

// 读取运动模式
sRtn = GT_GetPrfMode(Axis, &lPrfMode);
commandhandler("GT_GetPrfMode", sRtn);
// 清空字符串
memset(chPrfMode, '\0', 20);
switch(lPrfMode)
{
    case 0:
        sprintf(chPrfMode, "Trap");
        break;
    case 1:
        sprintf(chPrfMode, "Jog");
        break;
    case 2:
        sprintf(chPrfMode, "PT");
        break;
    case 3:
        sprintf(chPrfMode, "Gear");
        break;
    case 4:
        sprintf(chPrfMode, "Follow");
        break;
    case 5:
        sprintf(chPrfMode, "Interpolation");
        break;
    case 6:
        sprintf(chPrfMode, "PVT");
        break;
    default:
        break;
}
printf("运动模式 %s\n", chPrfMode);
return TRUE;
}

```

第6章 运动模式

6.1 本章简介

运动模式是指规划一个或多个轴运动的方式。运动控制器支持的运动模式有点位运动模式、Jog 运动模式、电子齿轮（即 Gear）运动模式和插补运动模式。



本手册中所有字体为蓝色的指令（如 [GT_PrfrTrap](#)）均带有超级链接，点击可跳转至指令说明。



用户需注意，每一个轴在任一时刻只能处在一种运动模式下。

表 6-1 设置运动模式指令列表

指令	说明	页码
GT_PrfrTrap	设置指定轴为点位模式	211
GT_PrfrJog	设置指定轴为 Jog 模式	210
GT_PrfrGear	设置指定轴为电子齿轮模式	210
GT_GetPrfrMode	查询指定轴的运动模式	198



在设置或切换运动模式时，要保证轴处于规划静止状态。如果轴正在运动，请先调用 [GT_Stop](#) 指令停止一个或多个轴的运动，然后再调用上表中的指令切换到想要的运动模式下。

6.2 点位运动模式

6.2.1 指令列表

表 6-2 点位运动模式指令列表

指令	说明	页码
GT_PrfrTrap	设置指定轴为点位运动模式	211
GT_SetTrapPrm	设置点位运动模式下的运动参数	225
GT_GetTrapPrm	读取点位运动模式下的运动参数	201
GT_SetPos	设置目标位置	223
GT_GetPos	读取目标位置	197
GT_SetVel	设置目标速度	227
GT_GetVel	读取目标速度	202
GT_Update	启动点位运动	230

6.2.2 重点说明

每一个轴在规划静止时都可以设置为点位运动。在点位运动模式下，各轴可以独立设置目标位置、目标速度、加速度、减速度、起跳速度、平滑时间等运动参数，能够独立运动或停止。

调用 `GT_Update` 指令启动点位运动以后，控制器根据设定的运动参数自动生成相应的梯形曲线速度规划，并且在运动过程中可以随时修改目标位置和目标速度。

设定平滑时间能够得到平滑的速度曲线，从而使加减速过程更加平稳，如图 6-1 所示。

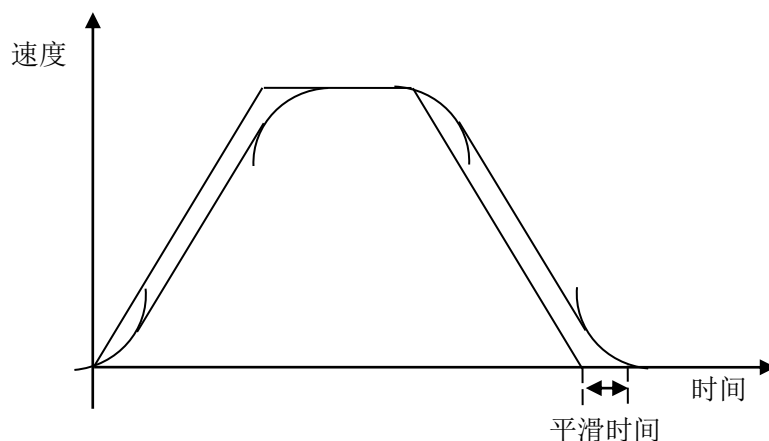


图 6-1 点位运动速度曲线

平滑时间是指加速度的变化时间，单位：ms，取值范围：[0, 50]。

6.2.3 例程

例程 6-1 点位运动

将第 1 轴设定为点位运动模式，并且以速度 50pulse/ms ，加速度 0.25pulse/ms^2 ，减速度 0.125pulse/ms^2 ，平滑时间为 25ms 的运动参数正向运动 50000 个脉冲。

该例程生成一段梯形曲线速度规划，如图 6-2 所示。

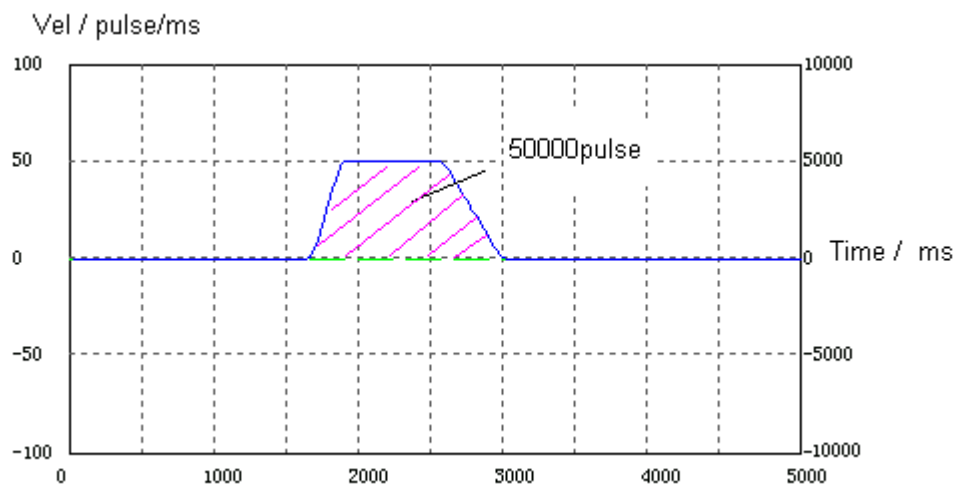


图 6-2 点位运动速度规划

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

// 定义轴号
#define AXIS 1

int main(int argc, char* argv[])
{
    short sRtn;
    TTrapPrm trap;
    long sts;
    double prfPos;

    // 打开运动控制器
    sRtn = GT_Open();
    // 指令返回值检测, 请查阅例程3-1
    commandhandler("GT_Open", sRtn);
    // 复位运动控制器
    sRtn = GT_Reset();
    commandhandler("GT_Reset", sRtn);
    // 配置运动控制器
    // 注意: 配置文件取消了各轴的报警和限位
    sRtn = GT_LoadConfig("test.cfg");
    commandhandler("GT_LoadConfig", sRtn);
    // 清除各轴的报警和限位
    sRtn = GT_ClrSts(1, 8);
    commandhandler("GT_ClrSts", sRtn);
    // 伺服使能
    sRtn = GT_AxisOn(AXIS);
    commandhandler("GT_AxisOn", sRtn);

    // 位置清零
    sRtn = GT_ZeroPos(AXIS);
    commandhandler("GT_ZeroPos", sRtn);
    // AXIS轴规划位置清零
    sRtn = GT_SetPrfPos(AXIS, 0);
    commandhandler("GT_SetPrfPos", sRtn);
    // 将 AXIS 轴设为点位模式
    sRtn = GT_PrftTrap(AXIS);
    commandhandler("GT_PrftTrap", sRtn);
    // 读取点位运动参数(需要读取所有运动参数到上位机变量)
    sRtn = GT_GetTrapPrm(AXIS, &trap);
    commandhandler("GT_GetTrapPrm", sRtn);
    // 设置需要修改的运动参数
```

```

trap.acc = 0.25;
trap.dec = 0.125;
trap.smoothTime = 25;
// 设置点位运动参数
sRtn = GT_SetTrapPrm(AXIS, &trap);
commandhandler("GT_SetTrapPrm", sRtn);
// 设置 AXIS 轴的目标位置
sRtn = GT_SetPos(AXIS, 50000L);
commandhandler("GT_SetPos", sRtn);
// 设置AXIS轴的目标速度
sRtn = GT_SetVel(AXIS, 50);
commandhandler("GT_SetVel", sRtn);
// 启动AXIS轴的运动
sRtn = GT_Update(1<<(AXIS-1));
commandhandler("GT_Update", sRtn);

do
{
    // 读取AXIS轴的状态
    sRtn = GT_GetSts(AXIS, &sts);
    // 读取AXIS轴的规划位置
    sRtn = GT_GetPrfPos(AXIS, &prfPos);
    printf("sts=0x%-10lprfPos=%-10.1lfr", sts, prfPos);
} while(sts&0x400); // 等待AXIS轴规划停止

// 伺服关闭
sRtn = GT_AxisOff(AXIS);
printf("\nGT_AxisOff()=%d\n", sRtn);
getch();
return 0;
}

```

6.3 Jog 运动模式

6.3.1 指令列表

表 6-3 Jog 运动模式指令列表

指令	说明	页码
GT_PrJog	设置指定轴为 Jog 运动模式	210
GT_SetJogPrm	设置 Jog 运动模式下的运动参数	221
GT_GetJogPrm	读取 Jog 运动模式下的运动参数	194
GT_SetVel	设置目标速度	227
GT_GetVel	读取目标速度	202
GT_Update	启动 Jog 运动	230

6.3.2 重点说明

在 Jog 运动模式下，各轴可以独立设置目标速度、加速度、减速度、平滑系数等运动参数，能够独立运动或停止。

调用 `GT_Update` 指令启动 Jog 运动以后，按照设定的加速度加速到目标速度后保持匀速运动，在运动过程中可以随时修改目标速度，如图 6-3 所示。

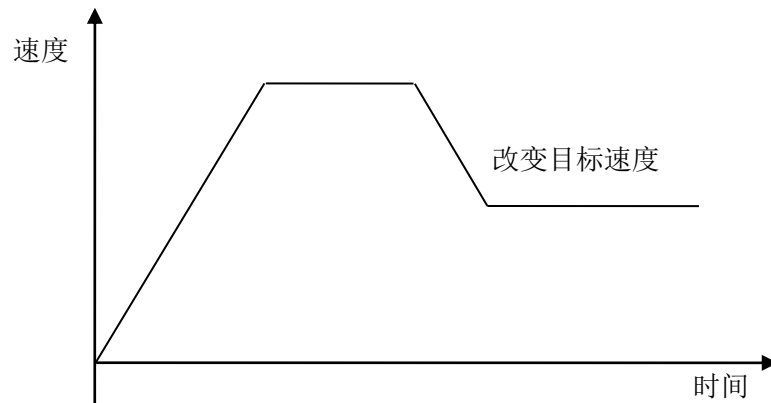


图 6-3 Jog 模式速度曲线

设定平滑系数能够得到平滑的速度曲线，从而使加减速过程更加平稳。平滑系数的取值范围是 $[0, 1)$ ，越接近 1，加速度变化越平稳。

6.3.3 例程

例程 6-2 Jog 运动

轴 1 运动在 Jog 模式下，初始目标速度为 100pulse/ms。动态改变目标速度，当规划位置超过 100000pulse 时，修改目标速度为 50 pulse/ms。如图 6-4 所示。

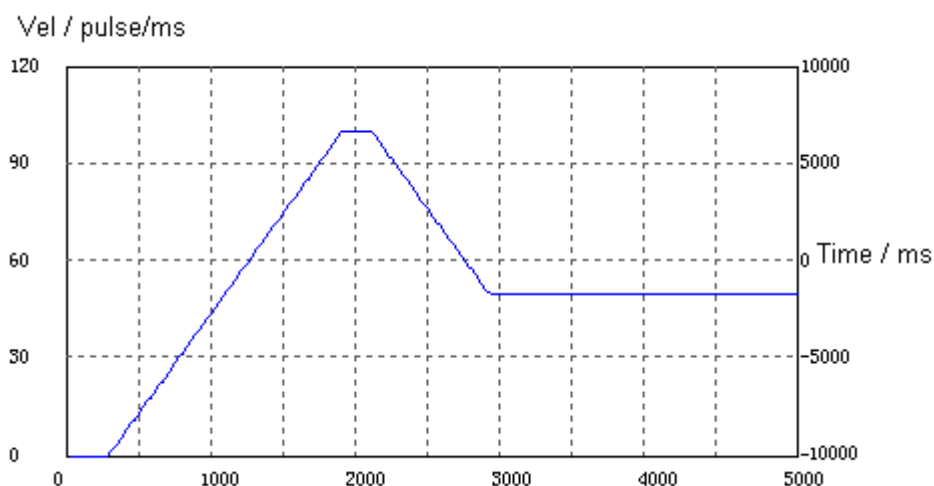


图 6-4 Jog 模式动态改变目标速度

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

```
// 定义轴号
#define AXIS 1

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

    // 打开运动控制器
    sRtn = GT_Open();
    // 指令返回值检测, 请查阅例程 3-1
    commandhandler("GT_Open", sRtn);
    // 复位运动控制器
    sRtn = GT_Reset();
    commandhandler("GT_Reset", sRtn);
    // 配置运动控制器
    // 注意: 配置文件取消了各轴的报警和限位
    sRtn = GT_LoadConfig("test.cfg");
    commandhandler("GT_LoadConfig", sRtn);
    // 清除各轴的报警和限位
    sRtn = GT_ClrSts(1, 8);
    commandhandler("GT_ClrSts", sRtn);
    // 伺服使能
    sRtn = GT_AxisOn(AXIS);
    commandhandler("GT_AxisOn", sRtn);
    // 位置清零
    sRtn = GT_ZeroPos(AXIS);
    commandhandler("GT_ZeroPos", sRtn);
    // 将 AXIS 轴设为 Jog 模式
    sRtn = GT_PrkJog(AXIS);
    commandhandler("GT_PrkJog", sRtn);
    // 读取 Jog 运动参数(需要读取全部运动参数到上位机变量)
    sRtn = GT_GetJogPrm(AXIS, &jog);
    commandhandler("GT_GetJogPrm", sRtn);
    // 设置需要修改的运动参数
    jog.acc = 0.0625;
    jog.dec = 0.0625;
    // 设置 Jog 运动参数
    sRtn = GT_SetJogPrm(AXIS, &jog);
    commandhandler("GT_SetJogPrm", sRtn);
    // 设置 AXIS 轴的目标速度
    sRtn = GT_SetVel(AXIS, 100);
    commandhandler("GT_SetVel", sRtn);
```

```

// 启动 AXIS 轴的运动
sRtn = GT_Update(1<<(AXIS-1));
commandhandler("GT_Update", sRtn);

while(1)
{
    // 读取AXIS轴的状态
    sRtn = GT_GetSts(AXIS, &sts);
    // 读取AXIS轴的规划位置
    sRtn = GT_GetPrfPos(AXIS, &prfPos);
    // 读取AXIS轴的规划速度
    sRtn = GT_GetPrfVel(AXIS, &prfVel);
    printf("sts=0x%-10lprfPos=%-10.1lfprfVel=%-10.1lfr", sts, prfPos, prfVel);
    if(prfPos>= 100000)
    {
        // 设置AXIS轴新的目标速度
        sRtn = GT_SetVel(AXIS, 50);
        commandhandler("GT_SetVel", sRtn);
        // AXIS轴新的目标速度生效
        sRtn = GT_Update(1<<(AXIS-1));
        commandhandler("GT_Update", sRtn);
        break;
    }
}
while(!kbhit())
{
    // 读取AXIS轴的状态
    sRtn = GT_GetSts(AXIS, &sts);
    // 读取AXIS轴的规划位置
    sRtn = GT_GetPrfPos(AXIS, &prfPos);
    // 读取AXIS轴的规划速度
    sRtn = GT_GetPrfVel(AXIS, &prfVel);
    printf("sts=0x%-10lprfPos=%-10.1lfprfVel=%-10.1lfr", sts, prfPos, prfVel);
}
// 停止轴运动
sRtn = GT_Stop(1<<(AXIS-1), 1<<(AXIS-1));
// 伺服关闭
sRtn = GT_AxisOff(AXIS);
printf("\nGT_AxisOff()=%d\n", sRtn);
getch();
return 0;
}

```

6.4 电子齿轮（Gear）运动模式

6.4.1 指令列表

表 6-4 电子齿轮运动模式指令列表

指令	说明	页码
GT_PrGfGear	设置指定轴为电子齿轮运动模式	210
GT_SetGearMaster	设置电子齿轮运动跟随主轴	219
GT_GetGearMaster	读取电子齿轮运动跟随主轴	193
GT_SetGearRatio	设置电子齿轮比	219
GT_GetGearRatio	读取电子齿轮比	194
GT_GearStart	启动电子齿轮运动	181

6.4.2 重点说明

电子齿轮模式能够将两轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接。

我们把被跟随的轴叫主轴，把跟随的轴叫从轴。电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置、编码器位置。

传动比：主轴速度与从轴速度的比例。电子齿轮模式能够灵活的设置传动比，节省机械系统的安装时间。当主轴速度变化时，从轴会根据设定好的传动比自动改变速度。电子齿轮模式也能够能够在运动过程中修改传动比。

离合区：当改变传动比时，可以设置离合区，实现平滑变速，如图所示，阴影区域为离合区。

电子齿轮模式速度曲线如图 6-5 所示。



离合区位移是指从轴平滑变速过程中主轴运动的位移。不要计算成从轴变速时走过的位移。

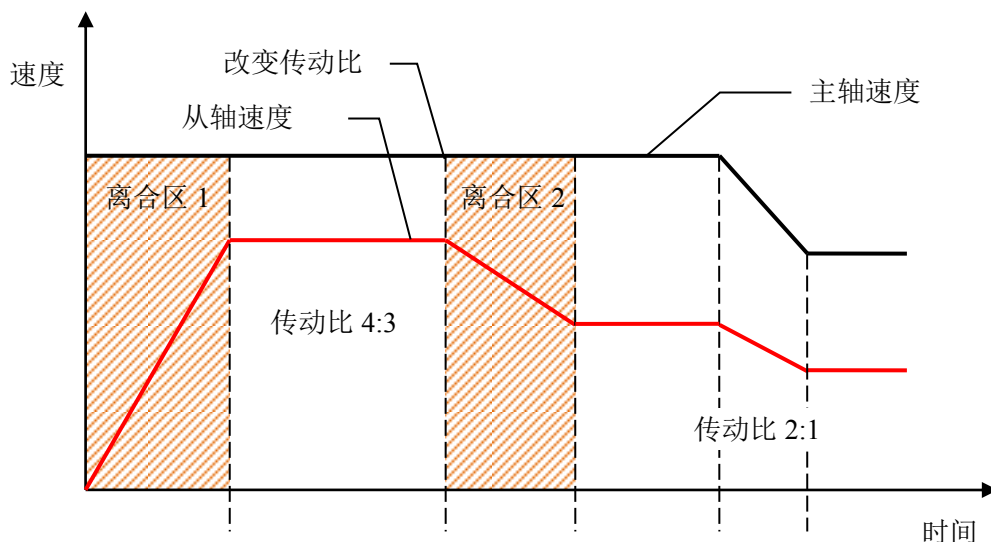


图 6-5 电子齿轮模式速度曲线

主轴匀速运动，从轴为电子齿轮模式。在离合区 1 从轴速度从 0 逐渐增大，直到到达传动比 4:3。当改变传动比至 2:1 时，在离合区 2 从轴速度逐渐变化直到满足新的传动比。离合区越大，从轴传动比的变化过程越平稳。当主轴速度变化时，从轴速度也随着变化，保持固定的传动比。

(1) 如何切换到电子齿轮模式？

用户必须要调用 `GT_PrGearing(short profile, short dir)`，才能将指定轴设定为 Gear 模式。应将 从轴设定为 Gear 模式。

(2) 如何设置主轴？

调用 `GT_SetGearMaster(short profile, short masterIndex, short masterType, short masterItem)`。
profile 为从轴轴号，masterIndex 为主轴轴号。



为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

(3) 如何设定传动比和离合区？

- 1) 调用 `GT_SetGearRatio(short profile, long masterEven, long slaveEven, long masterSlope)` 指令来设置传动比和离合区。profile 是从轴轴号；masterEven 是主轴位移，slaveEven 是从轴位移，masterEven/slaveEven 等于传动比；slope 是主轴离合区位移，即主轴在离合区内走过的位移，用户应自行计算出来。
- 2) 如果从轴轴号为 slave，当主轴位移 alpha 时从轴位移 beta，主轴运动 slope 位移后从轴到达设定传动比，应当调用以下指令：

`GT_SetGearRatio(slave, alpha, beta, slope);`

(4) 如何停止电子齿轮功能？

如果需要停止电子齿轮功能，可以调用 `GT_Stop` 停止从轴，就可以退出电子齿轮功能。

6.4.3 例程

例程 6-3 电子齿轮跟随

主轴为 Jog 模式，从轴为电子齿轮模式，传动比为主轴速度：从轴速度=2:1，主轴运动离合区位移后（图中阴影部分的区域），从轴达到设定的传动比，如图 6-6 和图 6-7 所示。

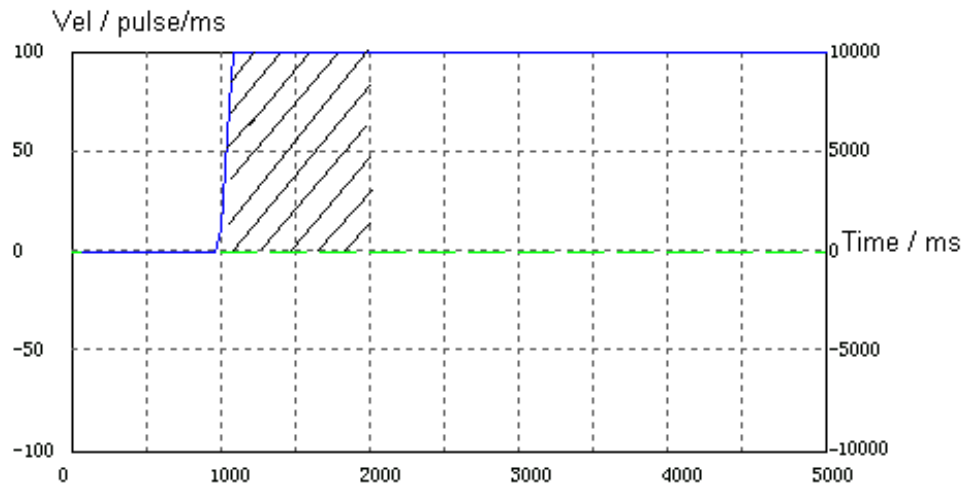


图 6-6 电子齿轮模式主轴速度规划

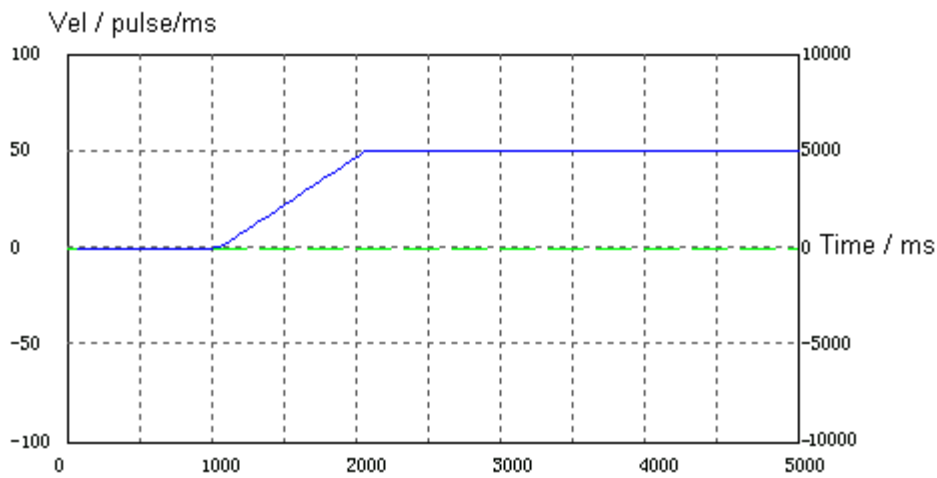


图 6-7 电子齿轮模式从轴速度规划

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

#define MASTER      1
#define SLAVE       2

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

```

double prfVel[8];
TJogPrm jog;

// 打开运动控制器
sRtn = GT_Open();
// 指令返回值检测, 请查阅例程3-1
commandhandler("GT_Open", sRtn);
// 复位运动控制器
sRtn = GT_Reset();
commandhandler("GT_Reset", sRtn);
// 配置运动控制器
// 注意: 配置文件 test.cfg 取消了各轴的报警和限位
sRtn = GT_LoadConfig("test.cfg");
commandhandler("GT_LoadConfig", sRtn);
// 清除各轴的报警和限位
sRtn = GT_ClrSts(1, 8);
commandhandler("GT_ClrSts", sRtn);
// 伺服使能
sRtn = GT_AxisOn(MASTER);
commandhandler("GT_AxisOn", sRtn);
sRtn = GT_AxisOn(SLAVE);
commandhandler("GT_AxisOn", sRtn);
// 位置清零
sRtn = GT_ZeroPos(MASTER);
commandhandler("GT_ZeroPos", sRtn);
sRtn = GT_ZeroPos(SLAVE);
commandhandler("GT_ZeroPos", sRtn);
// 将主轴设为 Jog 模式
sRtn = GT_PrkJog(MASTER);
commandhandler("GT_PrkJog", sRtn);
// 设置主轴运动参数
sRtn = GT_GetJogPrm(MASTER, &jog);
commandhandler("GT_GetJogPrm", sRtn);
jog.acc = 1;
sRtn = GT_SetJogPrm(MASTER, &jog);
commandhandler("GT_SetJogPrm", sRtn);
sRtn = GT_SetVel(MASTER, 100);
commandhandler("GT_SetVel", sRtn);
// 启动主轴
sRtn = GT_Update(1<<(MASTER-1));
commandhandler("GT_Update", sRtn);
// 将从轴设为 Gear 模式
sRtn = GT_PrjGear(SLAVE);
commandhandler("GT_PrjGear", sRtn);
// 设置主轴, 默认跟随主轴规划位置
sRtn = GT_SetGearMaster(SLAVE, MASTER);

```

```

commandhandler("GT_SetGearMaster", sRtn);
// 设置从轴的传动比和离合区
sRtn = GT_SetGearRatio(SLAVE, 2, 1, 100000);
commandhandler("GT_SetGearRatio", sRtn);
// 启动从轴
sRtn = GT_GearStart(1<<(SLAVE-1));
commandhandler("GT_GearStart",sRtn);

while(!kbhit())
{
    // 查询各轴的规划速度
    sRtn = GT_GetPrfVel(1, prfVel, 8);
    printf("master vel=%-10.2lftslave vel=%-10.2lfr",
        prfVel[MASTER-1], prfVel[SLAVE-1]);
}

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

```

6.5 插补运动模式

6.5.1 指令列表

表 6-5 插补运动模式指令列表

指令	说明	页码
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	215
GT_GetCrdPrm	查询坐标系参数	188
GT_CrdData	向插补缓存区增加插补数据	177
GT_LnXY	缓存区指令，二维直线插补	206
GT_LnXYZ	缓存区指令，三维直线插补	207
GT_LnXYZA	缓存区指令，四维直线插补	208
GT_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)	207
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	209
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	208
GT_ArcXYR	缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)	164
GT_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)	163
GT_ArcYZR	缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)	165
GT_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)	165

指令	说明	页码
GT_ArcZXR	缓存区指令, ZX 平面圆弧插补(以终点位置和半径为输入参数)	166
GT_ArcZXC	缓存区指令, ZX 平面圆弧插补(以终点位置和圆心位置为输入参数)	166
GT_ArcXYZ	缓冲区指令, 空间圆弧插补	167
GT_BufIO	缓存区指令, 缓存区内数字量 IO 输出设置指令	170
GT_BufDelay	缓存区指令, 缓存区内延时设置指令	169
GT_BufDA	缓存区指令, 缓存区内输出 DA 值	169
GT_BufLmtsOn	缓存区指令, 缓存区内有效限位开关	171
GT_BufLmtsOff	缓存区指令, 缓存区内无效限位开关	171
GT_BufSetStopIo	缓存区指令, 缓存区内设置 axis 的停止 IO 信息	172
GT_BufMove	缓存区指令, 实现刀向跟随功能, 启动某个轴点位运动	172
GT_BufGear	缓存区指令, 实现刀向跟随功能, 启动某个轴跟随运动	170
GT_CrdSpace	查询插补缓存区剩余空间	178
GT_CrdClear	清除插补缓存区内的插补数据	177
GT_CrdStart	启动插补运动	178
GT_CrdStatus	查询插补运动坐标系状态	178
GT_SetUserSegNum	缓存区指令, 设置自定义插补段段号	226
GT_GetUserSegNum	读取自定义插补段段号	201
GT_GetRemainderSegNum	读取未完成的插补段段数	199
GT_SetOverride	设置插补运动目标合成速度倍率	222
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	216
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	188
GT_GetCrdPos	查询该坐标系的当前坐标位置值	188
GT_GetCrdVel	查询该坐标系的合成速度值	189
GT_InitLookAhead	初始化插补前瞻缓存区	205
GT_SetCrdBufferMode	获取插补缓冲区的模式(静态模式或动态模式)	214
GT_GetCrdBufferMode	获取插补缓冲区的模式(静态模式或动态模式)	187
GT_GetCrdSegmentTime	获取插补段的执行时间	188
GT_GetCrdTime	获取插补运动的执行时间	189

6.5.2 重点说明

1. 直线插补与圆弧插补

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

插补最常见的两种方式是直线插补和圆弧插补。

直线插补方式中, 两点间的插补沿着直线的点群来逼近。首先假设在实际轮廓起始点处沿 x 方向走一小段(如一个脉冲当量), 发现终点在实际轮廓的下方, 则下一条线段沿 y 方向走一小段, 此时如果线段终点还在实际轮廓下方, 则继续沿 y 方向走一小段, 直到在实际轮廓上方以后, 再向 x 方向走一小段。依次循环类推。直到到达轮廓终点为止。这样实际轮廓是由一段段的折线拼接而

成，虽然是折线，如果我们每一段走刀线段都在精度允许范围内，那么此段折线还是可以近似看做一条直线段。这就是直线插补。假设某数控机床刀具在 xy 平面上从点 (x_0, y_0) 运动到点 (x_1, y_1) ，其直线插补的加工过程如图 6-8 所示。

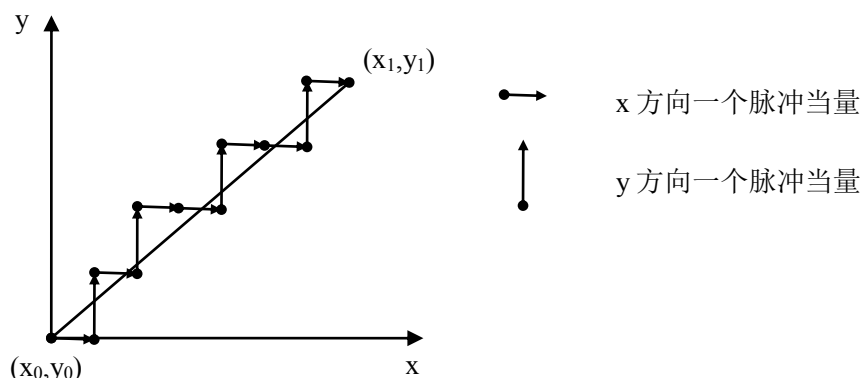


图 6-8 直线插补示意图

圆弧插补是给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制刀具沿这些点运动，加工出圆弧曲线。圆弧插补只能在某一平面进行。假设某数控机床刀具在 xy 平面第一象限走一段逆圆弧，圆心为原点，半径为 5，起点 $A(5, 0)$ ，终点 $B(0, 5)$ ，其圆弧插补的加工过程如图 6-9 所示。

控制器还支持空间圆弧插补功能，根据前一个点和该指令参数设置的终点和中间点，由三个点确定圆弧，并实现圆弧插补运动。对应的指令 [GT_ArcXYZ](#)。

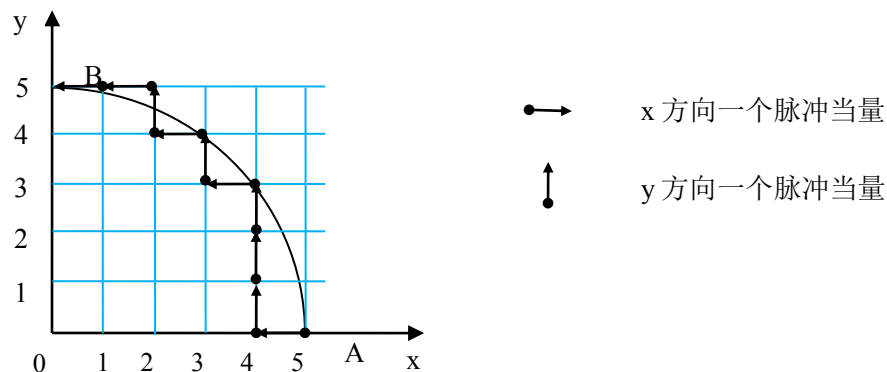


图 6-9 圆弧插补示意图

2. 运动控制器的插补模式

运动控制器的插补运动模式具有以下功能：

- (1) 可以实现直线插补和圆弧插补；
- (2) 可以同时有两个坐标系进行插补运动；
- (3) 每个坐标系含有两个缓存区，可以实现缓存区暂停、恢复等功能；
- (4) 具有缓存区延时和缓存区数字量输出的功能；
- (5) 具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

3. 使用插补模式的步骤

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

(1) 建立坐标系

在运动控制器的初始状态下，复位之后或者还未使用过插补运动状态下，所有的规划轴都处于单轴运动模式下，两个坐标系也是无效的。所以，进行插补运动时，首先需要建立坐标系，将规划轴映射到相应的坐标系中。每个坐标系最多支持四维(X-Y-Z-A)，用户根据自己的需求，也可以利用二维(X-Y)、三维(X-Y-Z)坐标系描述运动轨迹。

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

例程 6-4-1 建立坐标系

建立了一个二维坐标系，规划轴 1 对应为 x 轴，规划轴 2 对应为 y 轴，坐标系原点的规划位置是(100, 100)，单位：pulse，在此坐标系内运动的最大合成速度为 500pulse/ms，最大合成加速度为 1pulse/ms²，最小匀速时间为 50ms。如图 6-10 所示。

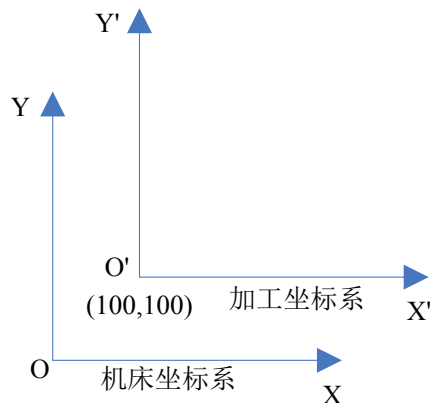


图 6-10 加工坐标系偏移量示意图

```

.....
// 指令返回值变量
short sRtn;
// TCrdPrm结构体变量，该结构体定义了坐标系
TCrdPrm crdPrm;

// 将结构体变量初始化为0
memset(&crdPrm, 0, sizeof(crdPrm));
// 为结构体赋值
crdPrm.dimension = 2;           // 坐标系为二维坐标系
crdPrm.synVelMax = 500;         // 最大合成速度：500pulse/ms
crdPrm.synAccMax = 1;          // 最大加速度：1pulse/ms^2
crdPrm.evenTime = 50;          // 最小匀速时间：50ms
crdPrm.profile[0] = 1;         // 规划器1(即profile[0])对应到X轴（即1）
crdPrm.profile[1] = 2;         // 规划器2(即profile[1])对应到Y轴（即2）

```

```

crdPrm.setOriginFlag = 1;           // 表示需要指定坐标系的原点坐标的规划位置
crdPrm.originPos[0] = 100;         // 坐标系的原点坐标的规划位置为 (100, 100)
crdPrm.originPos[1] = 100;

```

```

// 建立1号坐标系，设置坐标系参数

```

```

sRtn = GT_SetCrdPrm(1, &crdPrm);

```

```

.....

```

例程 6-5-2 建立坐标系

建立了一个二维坐标系，规划轴 2 对应为 x 轴，规划轴 3 对应为 y 轴，坐标系原点的规划位置是(100, 100)，单位：pulse，在此坐标系内运动的最大合成速度为 500pulse/ms，最大合成加速度为 1pulse/ms²，最小匀速时间为 50ms。如图 6-10 所示。

```

.....

```

```

// 指令返回值变量

```

```

short sRtn;

```

```

// TCrdPrm结构体变量，该结构体定义了坐标系

```

```

TCrdPrm crdPrm;

```

```

// 将结构体变量初始化为0

```

```

memset(&crdPrm, 0, sizeof(crdPrm));

```

```

// 为结构体赋值

```

```

crdPrm.dimension=2;           // 坐标系为二维坐标系

```

```

crdPrm.synVelMax=500;         // 最大合成速度：500pulse/ms

```

```

crdPrm.synAccMax=1;          // 最大加速度：1pulse/ms^2

```

```

crdPrm.evenTime = 50;         // 最小匀速时间：50ms

```

```

crdPrm.profile[0] = 0;        // 规划器1不映射轴

```

```

crdPrm.profile[1] = 1;        // 规划器2(即profile[1])对应到X轴（即1）

```

```

crdPrm.profile[2] = 2;        // 规划器3(即profile[2])对应到Y轴（即2）

```

```

crdPrm.setOriginFlag = 1;     // 表示需要指定坐标系的原点坐标的规划位置

```

```

crdPrm.originPos[0] = 0;

```

```

crdPrm.originPos[1] = 100;    // 坐标系的原点坐标的规划位置为 (100, 100)

```

```

crdPrm.originPos[2] = 100;

```

```

// 建立1号坐标系，设置坐标系参数

```

```

sRtn = GT_SetCrdPrm(1, &crdPrm);

```

```

.....

```

例程说明：

- **dimension**：表示所建立的坐标系的维数，取值范围：[1, 4]，该例程中所建立的坐标系是二维，即 X-Y 坐标系。
- **synVelMax**：表示该坐标系所能承受的最大合成速度，如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。
- **synAccMax**：表示该坐标系所能承受的最大合成加速度，如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。

- **evenTime**: 每个插补段的最小匀速时间。当用户设置的插补段比较短时, 而该插补段的目标速度又设置的比较大, 则会造成合成速度的曲线如图 6-11 (a) 所示, 只有加速段和减速段, 形成一个速度尖角, 加速度在尖角处瞬间由正值变为了负值, 造成较大的冲击; 设置了 **evenTime** 之后, 可以减小目标速度, 使速度曲线如图 6-11 (b) 所示, 减小了加速度突变的冲击。

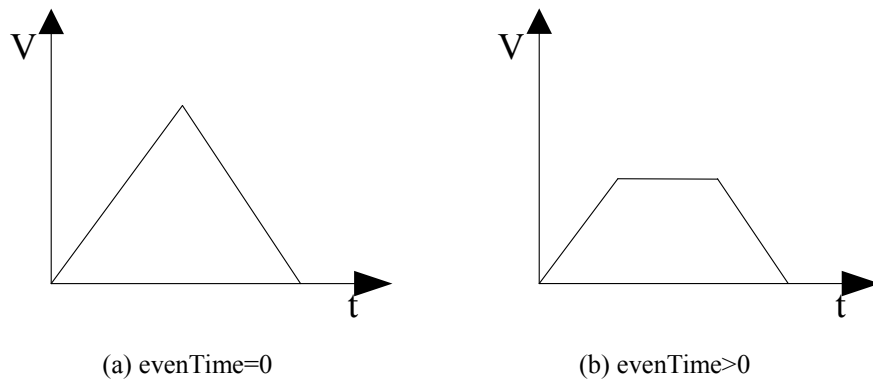


图 6-11 不同 evenTime 下的速度曲线

(2) 向缓存区存入数据

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

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

例程 6-6 直线插补例程

假设某数控机床刀具在 xy 平面从原点出发, 走一段如图 6-12 所示的正六边形轨迹。一共需要走七段轨迹, 图中标号已标出。每走完一段轨迹会输出一一次 IO 信号, 并且暂停 400ms, 其直线插补的例程如下, 直线插补例程运动轨迹如图 6-12 所示。

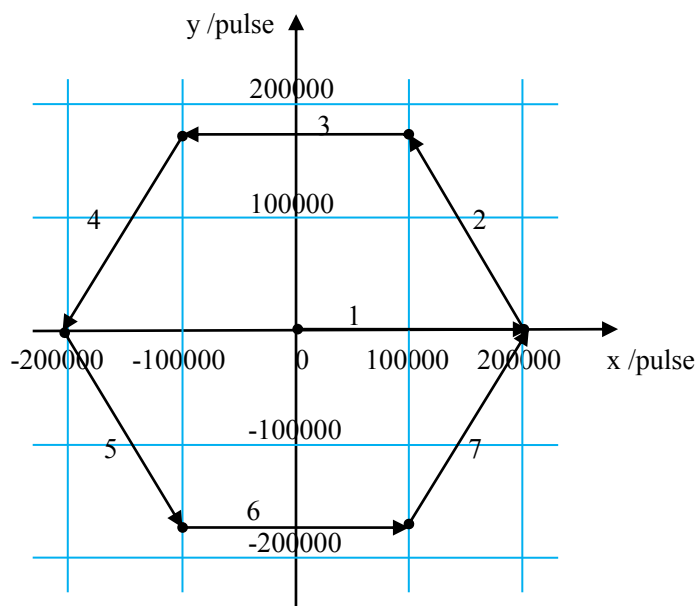


图 6-12 直线插补例程运动轨迹

```

.....
// 指令返回值变量
short sRtn;
// 坐标系运动状态查询变量
short run;
// 坐标系运动完成段查询变量
long segment;
// 坐标系的缓存区剩余空间查询变量
long space;

// 即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据
sRtn = GT_CrdClear(1, 0);
// 向缓存区写入第一段插补数据
sRtn = GT_LnXY(
    1,                // 该插补段的坐标系是坐标系1
    200000, 0,        // 该插补段的终点坐标(200000, 0)
    100,              // 该插补段的目标速度：100pulse/ms
    0.1,              // 插补段的加速度：0.1pulse/ms^2
    0,                // 终点速度为0
    0);               // 向坐标系1的FIFO0缓存区传递该直线插补数据

// 向缓存区写入第二段插补数据
sRtn = GT_LnXY(1, 100000, 173205, 100, 0.1, 0, 0);
// 缓存区数字量输出
sRtn = GT_BufIO(
    1,                // 坐标系是坐标系1
    MC_GPO,           // 数字量输出类型：通用输出
    0xffff,           // bit0~bit15全部都输出
    0x55,             // 输出的数值为0x55
    0);               // 向坐标系1的FIFO0缓存区传递该数字量输出
// 第三段插补数据
sRtn = GT_LnXY(1, -100000, 173205, 100, 0.1, 0, 0);
// 缓存区数字量输出
sRtn = GT_BufIO(1, MC_GPO, 0xffff, 0xaa, 0);
// 第四段插补数据
sRtn = GT_LnXY(1, -200000, 0, 100, 0.1, 0, 0);
// 缓存区延时指令
sRtn = GT_BufDelay(
    1,                // 坐标系是坐标系1
    400,              // 延时400ms
    0);               // 向坐标系1的FIFO0缓存区传递该延时
// 第五段插补数据
sRtn = GT_LnXY(1, -100000, -173205, 100, 0.1, 0, 0);
// 缓存区数字量输出
sRtn = GT_BufIO(1, MC_GPO, 0xffff, 0x55, 0);

```

```

// 缓存区延时指令
sRtn = GT_BufDelay(1, 100, 0);
// 第六段插补数据
sRtn = GT_LnXY(1, 100000, -173205, 100, 0.1, 0, 0);
// 第七段插补数据
sRtn = GT_LnXY(1, 200000, 0, 100, 0.1, 0, 0);
// 查询坐标系1的FIFO0所剩余的空间
sRtn = GT_CrdSpace(1, &space, 0);
// 启动坐标系1的FIFO0的插补运动
sRtn = GT_CrdStart(1, 0);
// 等待运动完成
sRtn = GT_CrdStatus(1, &run, &segment, 0);
do
{
    // 查询坐标系1的FIFO的插补运动状态
    sRtn = GT_CrdStatus(
        1,           // 坐标系是坐标系1
        &run,        // 读取插补运动状态
        &segment,    // 读取当前已经完成的插补段数
        0);         // 查询坐标系1的FIFO0缓存区
    // 坐标系在运动, 查询到的run的值为1
} while(run == 1);
.....

```

(3) 圆弧插补

运动控制器的插补模式支持在 XY 平面、YZ 平面和 ZX 平面的圆弧插补。其中圆弧插补的旋转方向按照右手螺旋定则定义为：从坐标平面的“上方”(即垂直于坐标平面的第三个轴的正方向)看，来确定逆时针方向和顺时针方向。可以这样简单记忆：将右手拇指前伸，其余四指握拳，拇指指向第三个轴的正方向，其余四指的方向即为逆时针方向。映射坐标系为二维坐标系(X-Y)时，XOY 坐标平面内的圆弧插补逆时针方向同样定义，如图 6-13 示。

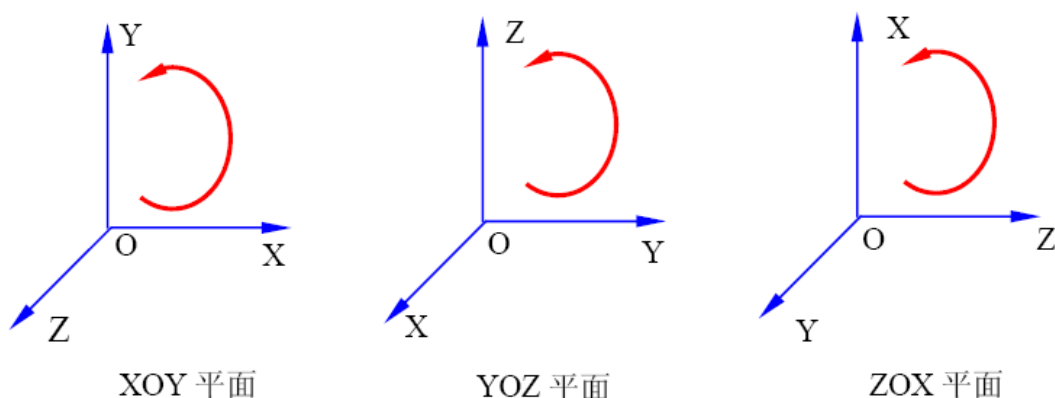


图 6-13 圆弧插补逆时针方向

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

a) 半径描述方法

调用指令 `GT_ArcXYR`、`GT_ArcYZR`、`GT_ArcZXR` 是使用半径描述方法对圆弧进行描述。使用半径描述方法，用户需要输入圆弧终点坐标、圆弧半径、圆弧的旋转方向、速度和加速度等。其中参数半径可为正值，也可为负值，其绝对值为圆弧的半径，正值表示圆弧的旋转角度 $\leq 180^\circ$ ，负值表示圆弧的旋转角度 $> 180^\circ$ ，如图 6-14 所示，半径描述方法无法描述 360° 的整圆。

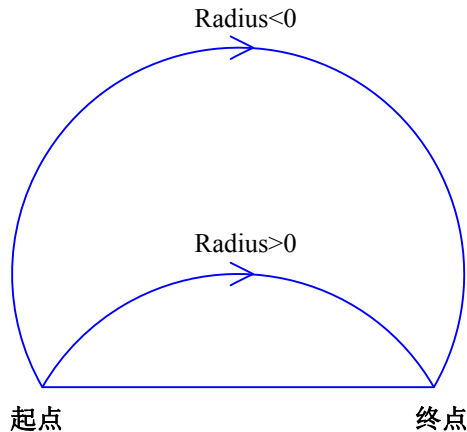


图 6-14 半径取正值/负值圆弧插补示意图

b) 圆心坐标描述方法

调用指令 `GT_ArcXYC`、`GT_ArcYZC`、`GT_ArcZXC` 是使用圆心坐标描述方法对圆弧进行描述。使用圆心描述方法，用户需要输入圆弧终点坐标、圆心相对于起点坐标的相对位置值、圆弧的旋转方向、速度和加速度等。其中，圆心位置值参数的定义如图 6-15 所示。

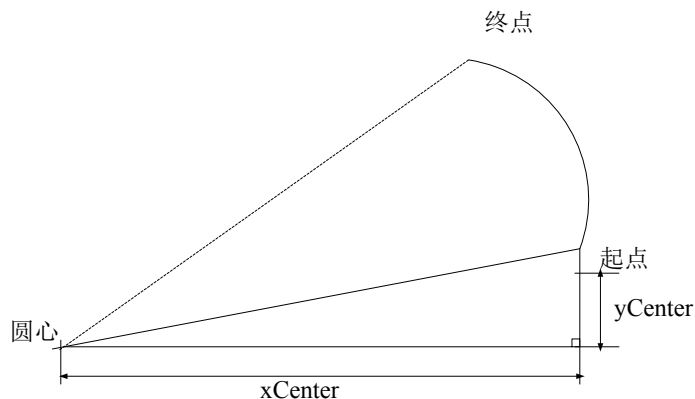


图 6-15 圆心坐标描述方法示意图

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



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

例程 6-7 圆弧插补例程

假设某数控机床刀具在 xy 平面走一段如图 6-16 所示的圆弧。该例程使用圆心坐标描述方法描述一个整圆，使用半径描述方法描述一个 $1/4$ 圆弧，最后回到原点位置。一共需要走三段轨迹，图

中用标号标出。整圆的圆心坐标为(100000, 0)，半径 100000pulse。圆弧的圆心坐标为原点(0, 0)，半径 200000pulse。

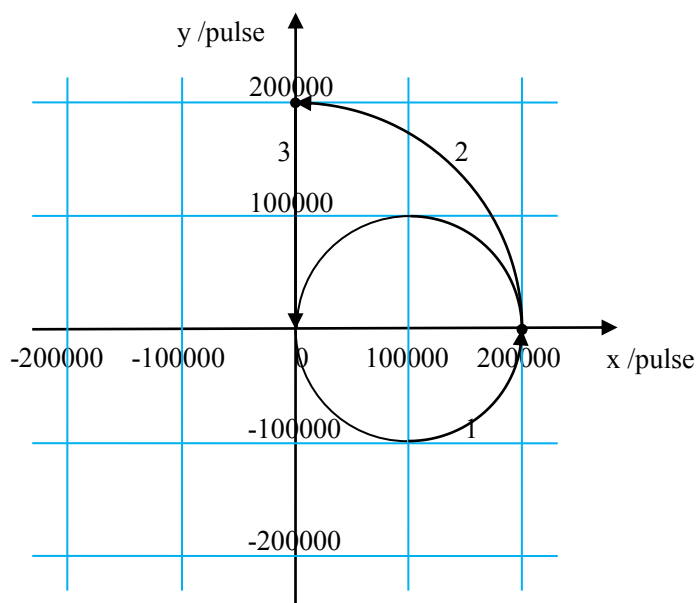


图 6-16 圆弧插补例程运动轨迹

```

.....
// 指令返回值变量
short sRtn;
// 坐标系运动状态查询变量
short run;
// 坐标系运动完成段查询变量
long segment;
// 坐标系的缓存区剩余空间查询变量
long space;

// 即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据
sRtn = GT_CrdClear(1, 0);
// 向缓存区写入第一段插补数据
sRtn = GT_LnXY(
    1,                // 该插补段的坐标系是坐标系1
    200000, 0,        // 该插补段的终点坐标(200000, 0)
    100,              // 该插补段的目标速度：100pulse/ms
    0.1,              // 插补段的加速度：0.1pulse/ms^2
    0,                // 终点速度为0
    0);               // 向坐标系1的FIFO0缓存区传递该直线插补数据

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

```

```

0.1,          // 该插补段的加速度: 0.1pulse/ms^2
0,            // 终点速度为0
0);           // 向坐标系1的FIFO0缓存区传递该直线插补数据

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

// 向缓存区写入第四段插补数据, 回到原点位置
sRtn = GT_LnXY(1, 0, 0, 100, 0.1, 0, 0);
// 查询坐标系1的FIFO0所剩余的空间
sRtn = GT_CrdSpace(1, &space, 0);
// 启动坐标系1的FIFO0的插补运动
sRtn = GT_CrdStart(1, 0);
// 等待运动完成
sRtn = GT_CrdStatus(1, &run, &segment, 0);
do
{
    // 查询坐标系1的FIFO的插补运动状态
    sRtn = GT_CrdStatus(
        1,          // 坐标系是坐标系1
        &run,       // 读取插补运动状态
        &segment,   // 读取当前已经完成的插补段数
        0);         // 查询坐标系1的FIFO0缓存区
    // 坐标系在运动, 查询到的run的值为1
} while(run == 1);

.....

```

4. 以 GT_Ln 开头 G0 结尾的指令

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

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

5. 缓存区 FIFO 的管理 (运动暂停与恢复)

每个坐标系包含两个缓存区(FIFO): FIFO0 和 FIFO1, 其中 FIFO0 为主要运动 FIFO, FIFO1 为

辅助运动 FIFO，每个 FIFO 都含有 4096 段插补数据的空间。



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

在运动控制器的插补模式下，不能随意切换到其他运动模式，否则会导致插补坐标系破坏，并且原来压入插补缓存区的数据会丢失。但是在实际应用中，经常会有类似下面例子描述的情况。假如机床在走一段轨迹的途中需要暂停下来，更换路径换刀或者移到安全的地方以查看加工效果，然后再回到暂停时的坐标继续完成剩余的轨迹。为了实现上述操作，应利用每个坐标系提供的两个缓存区 FIFO0 和 FIFO1。

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

FIFO1 是辅助运动 FIFO，用户的辅助插补运动的插补数据可以放在 FIFO1 中，FIFO1 的插补数据必须在 FIFO0 的运动停止的情况下才能输入，如果 FIFO0 在运动，向 FIFO1 中传递插补数据，将会提示错误。FIFO1 的插补运动也可以暂停、恢复，但是，在 FIFO1 暂停时不可以进行 FIFO0 的插补运动，否则，FIFO1 缓存区将会被清空，不可以再恢复 FIFO1 的运动，插补主运动与辅助运动流程如图 6-17 所示。



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

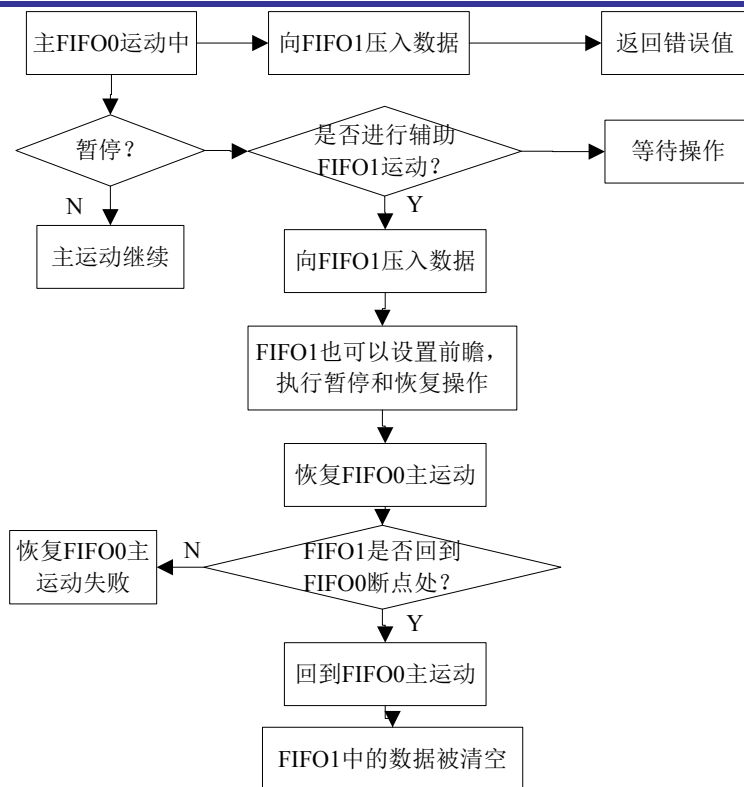


图 6-17 插补主运动与辅助运动流程

例程 6-8 插补 FIFO 管理

假设机床需要做运动: 主加工运动需要从(0, 0)运动到(100000, 100000)位置, 中途在(50000, 50000)附近出现了断刀, 需要暂停运动去换刀。由于主加工运动已经将预定的轨迹数据压入了缓存区, 若重新压入新的数据则会破坏原来的运动, 因此需要启动辅助运动来协助完成。

假设换刀轨迹是先走到(70000, 30000), 然后走到(110000, 50000)位置完成换刀动作。但为了能继续主加工运动, 需要回到暂停点, 如图 6-18 所示。

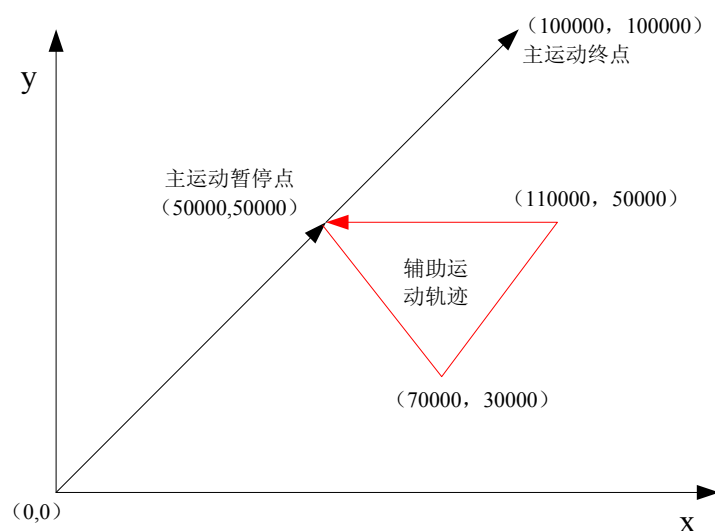


图 6-18 插补 FIFO 管理例程之换刀轨迹

.....
// 函数返回值

```

short sRtn;
// 循环变量
short i;
// 查询坐标系运行标志
short run;
// 查询坐标系运行的段数
long seg;
// 本程序局部运行标志，主运动启动时为1，停止时为0
short flag;
// 坐标系规划位置
double crdPos[2];
// 坐标系暂停位置
double crdstoppos[2];
// 坐标系结构体
TCrdPrm crdPrm;

// 清除fifo数据
sRtn = GT_CrdClear(1, 0);
sRtn = GT_CrdClear(1, 1);

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

do
{
    // 查询插补坐标位置
    sRtn = GT_GetCrdPos(1, crdPos);
    // 当X轴的坐标位置大于50000时，暂停
    if(crdPos[0] > 50000.0)
    {
        // 停止坐标系1的运动
        sRtn = GT_Stop(1<<8, 1<<8);
        // 注意:要等坐标系真正停止下来去获取当前的位置
        do
        {
            // 读取坐标系的状态，查看是否停止
            sRtn = GT_CrdStatus(1, &run, &seg, 0);

```

```

    }while(run == 1);
    // 获取暂停后当前的位置, 并将标志 flag 置 0, 退出 while 循环
    sRtn = GT_GetCrdPos(1, crdstoppos);
    flag = 0;
}
printf("crdPos[0]=%-10.1lf    crdPos[1]=%-10.1lfr", crdPos[0], crdPos[1]);
}while(flag == 1);

// 向FIFO1缓存区写入辅助运动插补数据
sRtn = GT_LnXY(1, 70000, 30000, 10, 1, 0, 1);
sRtn = GT_LnXY(1, 110000, 50000, 10, 1, 0, 1);
// 启动坐标系1的FIFO1中运动
sRtn = GT_CrdStart(1, 1);
do
{
    // 查询插补规划位置
    sRtn = GT_GetCrdPos(1, crdPos);
    // 等待辅助运动完毕
    sRtn = GT_CrdStatus(1, &run, &seg, 1);
    printf("crdPos[0]=%-10.1lf    crdPos[1]=%-10.1lfr", crdPos[0], crdPos[1]);
}while(1 == run);

// 向FIFO1压入暂停后的位置点
sRtn = GT_LnXY(1, (long)crdstoppos[0], (long)crdstoppos[1], 10, 1, 0, 1);
// 启动回暂停点运动
sRtn = GT_CrdStart(1, 1);
do
{
    // 查询插补规划位置
    sRtn = GT_GetCrdPos(1, crdPos);
    // 等待回到暂停点位置运动完毕
    sRtn = GT_CrdStatus(1, &run, &seg, 1);
    printf("crdPos[0]=%-10.1lf    crdPos[1]=%-10.1lfr", crdPos[0], crdPos[1]);
}while(1 == run);

// 恢复主运动
sRtn = GT_CrdStart(1, 0);
do
{
    // 获取当前的位置
    sRtn = GT_GetCrdPos(1, crdPos);
    // 等待主运动完毕
    sRtn = GT_CrdStatus(1, &run, &seg, 0);
    printf("crdPos[0]=%-10.1lf    crdPos[1]=%-10.1lfr", crdPos[0], crdPos[1]);
}while(1 == run);

```

6. 前瞻预处理

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

下面用一个实例来说明前瞻预处理的机制优势。假设机床要加工一个长方形的零件，刀具所走的轨迹如图 6-19 (a)所示。假设 m 点到 n 点距离 3000 个单位长度，有 30 段规划。n 点到 p 点距离 2000 个单位长度，有 20 段规划。每段规划 100 个单位长度。

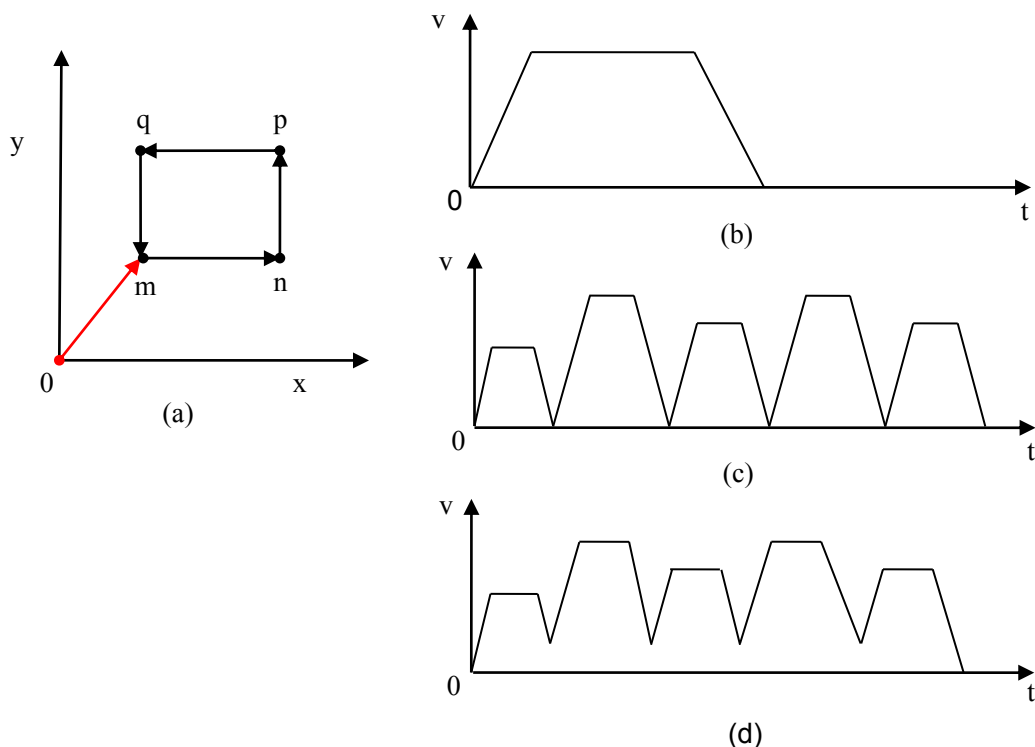


图 6-19 使用前瞻与不使用前瞻的速度规划区别

如果按照图 6-19(b)所示的速度规划，即在拐角处不减速，则加工精度一定会较低，而且可能在拐弯时对刀具和零件造成较大冲击。如果按照图 6-19(c)所示的速度规划，即在拐角处减速为 0，可以最大限度保证加工精度，但加工速度就会慢下来。如果按照图 6-19(d)所示的速度规划，在拐角处将速度减到一个合理值，既可以满足加工精度又能提高加工速度，就是一个好的速度规划。

为了实现类似图 6-19(d)所示的好的速度规划，前瞻预处理模块不仅要知道当前运动的位置参数，还要提前知道后面若干段运动的位置参数，这就是所谓的前瞻。例如在对图 6-19(a)中的轨迹做前瞻预处理时，我们设定控制器预先读取 50 段运动轨迹到缓存区中，则它会自动分析出在第 30 段将会出现拐点，并依据用户设定的拐弯时间计算在拐弯处的终点速度。前瞻预处理模块也会依照用户设定的最大加速度值计算速度规划，使任何加减速过程都不会超过这个值，防止对机械部分产生破坏性冲击力。

从下图 6-20 可以直观地了解，使用前瞻预处理功能模块来规划速度，在小线段加工过程中的对速度的显著提升。前瞻预处理流程图如图 6-21 所示。

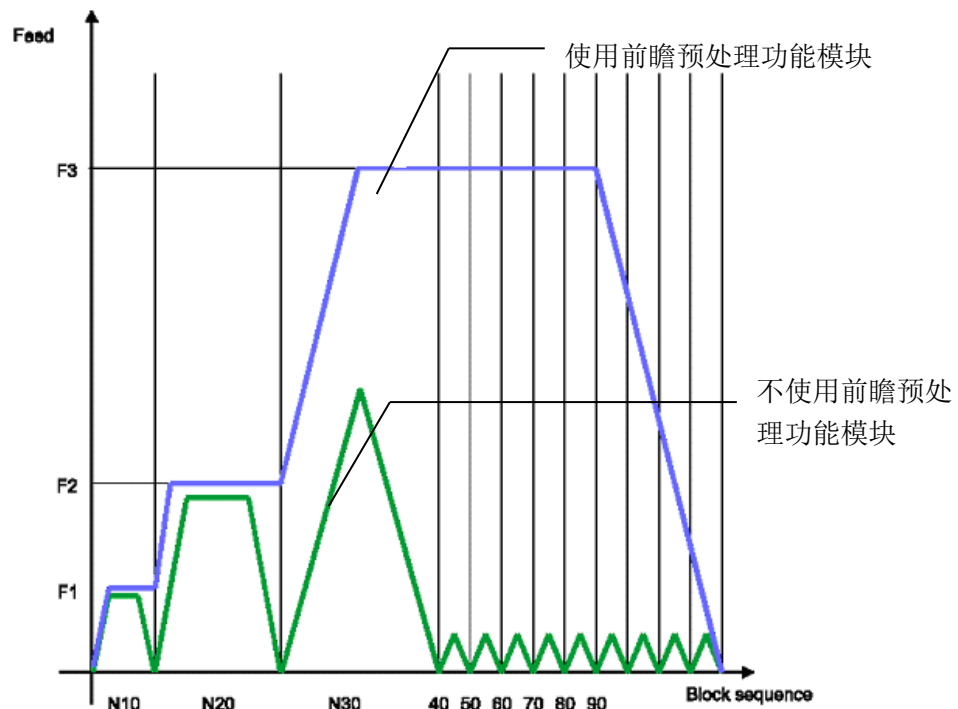


图 6-20 使用和不使用前瞻预处理功能模块的速度曲线对比图

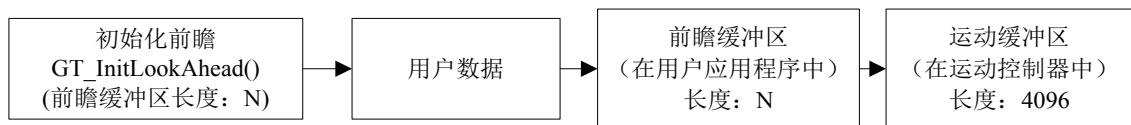


图 6-21 前瞻预处理流程图

- (1) 用户在应用程序中定义一个结构体 TCrdData 的数组作为前瞻缓存区，数组大小用户自己设定，比如数组大小为 200，那么前瞻缓存区长度 $N=200$ 段；然后调用指令 `GT_InitLookAhead` 作初始化前瞻。
- (2) 用户调用如 `GT_LnXY` 等直线插补指令和圆弧插补指令将数据段输入缓存区。这时，插补数据先流入开辟的前瞻缓存区，当流入前瞻缓存区的数据段数大于了 N 之后，才能逐段流入运动缓存区。运动缓存区是控制器内部资源，大小 4096 段，每一段可以存放一条指令。控制器只能执行压入运动缓存区中的数据，所以用户一定要确保前瞻缓存区的数据进入运动缓存区。分不同情况分析：
 - 1) 假设用户数据只有 190 段，则当用户调用完后，数据会一直停留在前瞻缓存区，因此，用户需要调用 `GT_CrdData(1, NULL, 0)` 来将前瞻缓存区数据压入运动缓存区。
 - 2) 假设用户数据只有 300 段，则当用户调用完后，有 100 段数据已经流入运动缓存区，但还有 200 段留在前瞻缓存区，同样，用户需要调用 `GT_CrdData(1, NULL, 0)` 来将前瞻缓存区数据压入运动缓存区。
 - 3) 假设用户有数据 5000 段，则在用户调用插补指令过程中，会出现前瞻缓存区和运动缓存区都被压满的情况，因此，需要注意，若一直压数据，没有启动插补运动，当压入第 4297 段时（前瞻缓存区和运动缓存区一共 4296 段大小），由于两缓存区都满了，

所以调用指令会返回 1。此时需要调用 `GT_CrdSpace` 查询运动缓存区的空间，只有当查询到当前运动缓存区有空间时，才能继续调用插补指令压入剩下的数据。同样，最后用户需要调用 `GT_CrdData(1, NULL, 0)` 来将前瞻缓存区数据压入运动缓存区。



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

例程 6-9 前瞻预处理例程

假设机床加工过程中，需要走一长直线，该直线由 300 条小直线段组成，现对这段路径进行前瞻预处理。其轨迹如图 6-22 所示。红色线段为起始轨迹。

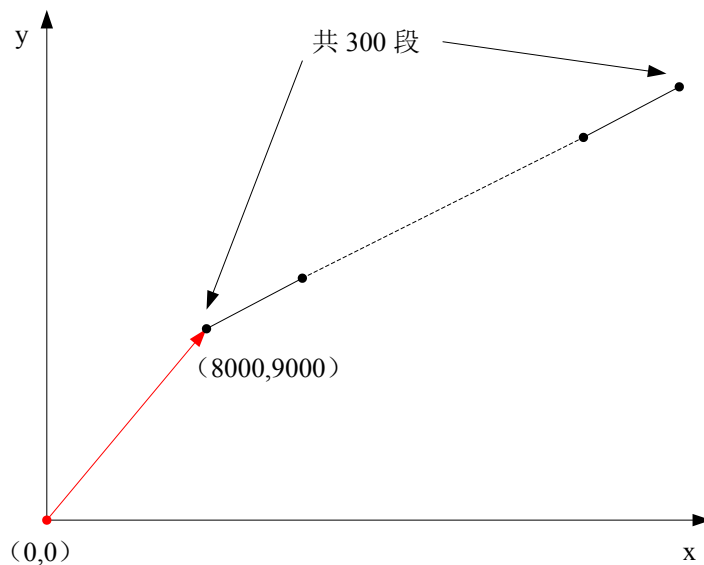


图 6-22 前瞻预处理例程之运动轨迹图

```
.....
// 指令返回值
short sRtn;
// 循环变量
int i;
// 定义前瞻缓存区内存区
TCrdData crdData[200];
long posTest[2];
long space;

// 初始化坐标系1的FIFO0的前瞻模块
sRtn = GT_InitLookAhead(1, 0, 5, 1, 200, crdData);
// 压插补数据：小线段加工
posTest[0] = 0;
posTest[1] = 0;
for(i=0;i<300;++i)
{
    sRtn = GT_LnXY(1, 8000+posTest[0], 9000+posTest[1], 100, 0.8, 0, 0);
}
```

```

// 查询返回值是否成功
if(0 != sRtn)
{
    do
    {
        // 查询运动缓存区空间，直至空间不为0
        sRtn = GT_CrdSpace(1, &space, 0);
    }while(0 == space);
    // 重新调用上次失败的插补指令
    sRtn = GT_LnXY(1, 8000+posTest[0], 9000+posTest[1], 100, 0.8, 0, 0);
}
posTest[0] += 1600;
posTest[1] += 1852;
}

// 将前瞻缓存区中的数据压入控制器
sRtn = GT_CrdData(1, NULL, 0);
// 启动运动
sRtn = GT_CrdStart(1, 0);
.....

```

例程说明：

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



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

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

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

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

对于 C#用户, **GT_CrdData** 的声明如下:

```
public static extern short GT_CrdData(short crd, System.IntPtr pCrdData, short fifo);
```

程序调用为: `gts.mc.GT_CrdData(1, System.IntPtr.Zero, 0);`

对于 VB.net 用户, `GT_CrdData` 的声明如下:

```
Public Declare Function GT_CrdData Lib "gts.lib" (ByVal crd As Short, ByVal pCrdData As Long, Optional ByVal fifo As Short = 0) As Short
```

程序调用为: `GT_CrdData(crd,0,fifo)`

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

- 没有前瞻预处理和经过前瞻预处理的区别如图 6-23 和图 6-24 所示。

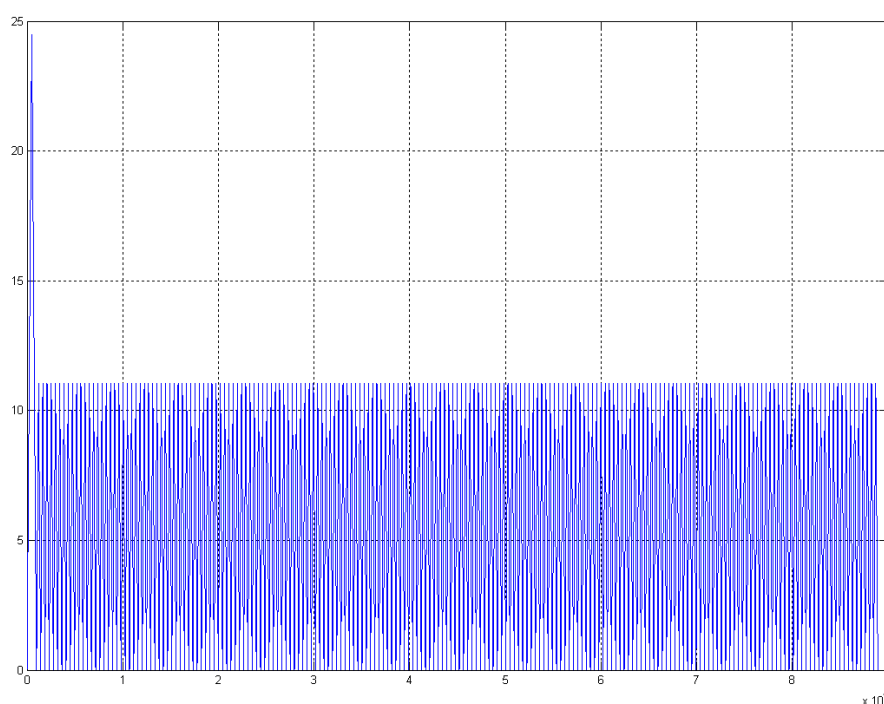


图 6-23 没有进行前瞻预处理的合成速度曲线

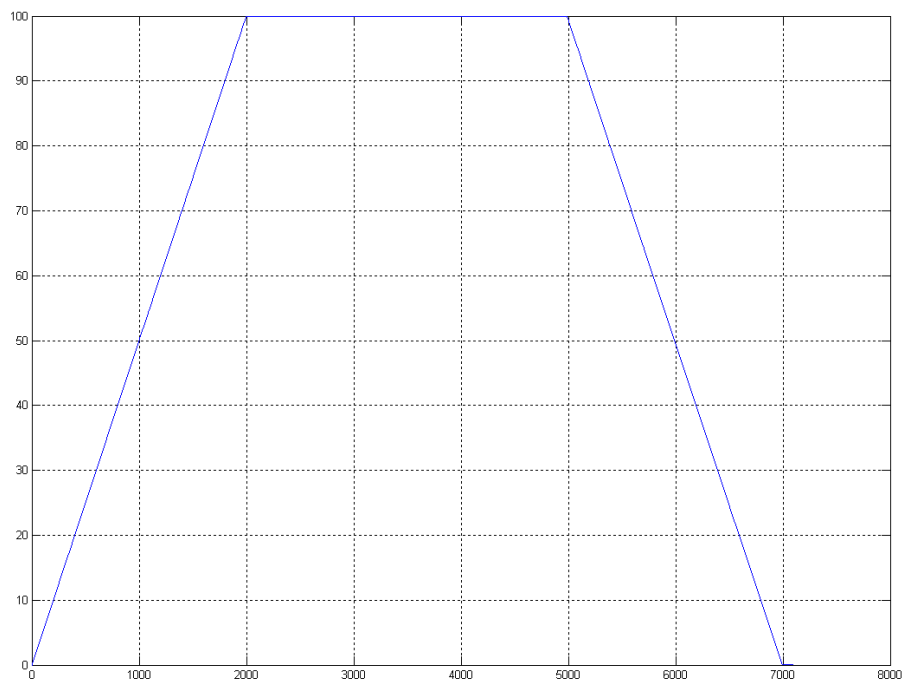


图 6-24 进行了前瞻预处理后的合成速度曲线

7. 刀向跟随功能

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

(1) GT_BufMove

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

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

该指令的第二个参数是需要进行点位运动的轴号。



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

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

例程 6-10 刀向跟随功能 GT_BufMove

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

运动到(200000, 0)的同时,在 A 轴方向以 30 pulse/ms 的速度运动到 50000pulse 的位置(第 2 段轨迹)。这里用到 `GT_BufMove` 的非模态方式。然后,在 A 轴方向以 30 pulse/ms 的速度运动 100000 的位置。等到 A 轴方向运动完成,在 XY 坐标内画一个圆心为原点,半径 200000,位于三四象限的半圆弧(第 3 段轨迹)。这里用到 `GT_BufMove` 的模态方式,刀向跟随功能 `GT_BufMove` 的运动轨迹如图 6-25 所示。

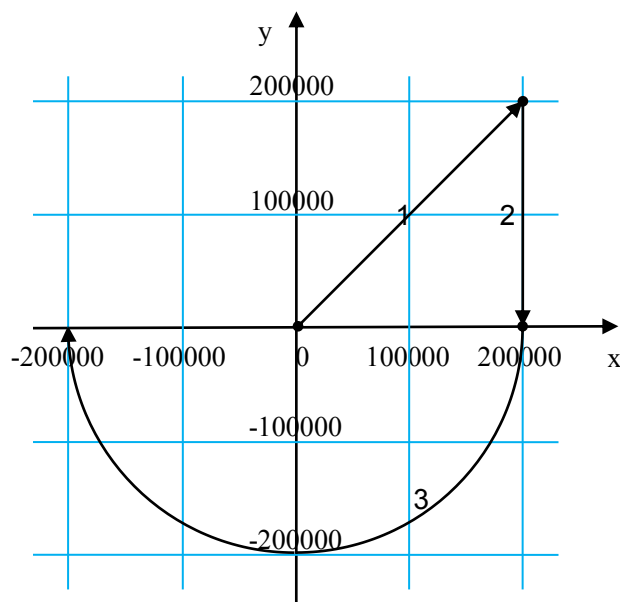


图 6-25 刀向跟随功能 `GT_BufMove` 的运动轨迹

```

.....
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;

// 清除坐标系1的FIFO0中的数据
sRtn = GT_CrdClear(1, 0);
// 向FIFO0缓存区写入一段直线插补数据
sRtn = GT_LnXY(
    1,                // 该插补段的坐标系是坐标系1
    200000, 200000,   // 该插补段的终点坐标(200000, 200000)
    100,              // 该插补段的目标速度: 100pulse/ms
    0.1,              // 插补段的加速度: 0.1pulse/ms^2
    0,                // 终点速度为0
    0);               // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 向FIFO0缓存区写入一段非模态点位运动数据
sRtn = GT_BufMove(
    1,                // 该插补段的坐标系是坐标系1
    4,                // 点位运动的轴号: 第4轴
    50000,            // 点位运动的目标位置: 50000 pulse

```

```

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

do
{
    // 坐标系在运动, 查询到的run的值为1
    sRtn = GT_CrdStatus(1, &run, &segment, 0);
}while(run == 1);
.....

```

例程插补合成速度和点位速度的运行结果如图 6-26 所示。

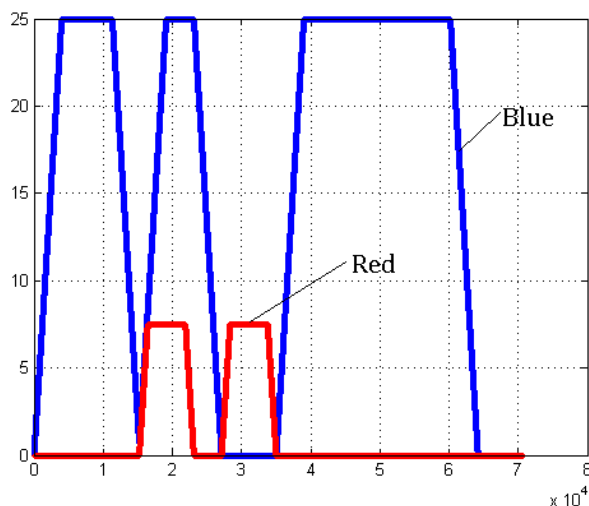


图 6-26 插补缓存区内的点位运动速度图

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



注意

在使用插补缓存区中的点位运动功能时，需要注意以下内容：

1. 点位运动的目标位置是相对于机床原点的绝对位置。
2. 如果在上一轮的缓存区点位运动没有运动完成，又发送了新的点位运动，则会按照新的点位运动指令进行规划，即可以在插补缓存区中修改点位运动的目标位置和目标速度。
3. 如果在运动过程中停止插补缓存区的运动，则点位运动将不会停止，如果需要停止点位运动，则需要调用 `GT_Stop` 指令停止响应轴的运动。恢复缓存区运动时，用户需自行保证之前点位运动的轴在合适的位置上。
4. 当在模态点位运动的过程中，进行点位运动的轴由于触发限位等异常原因停止时，插补缓存区将不会继续运行，此时用户需排查异常情况，重新设置相应参数，使系统正常后才可以工作。

(2) `GT_BufGear`

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



注意

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

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



注意

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

例程 6-11 刀向跟随功能 `GT_BufGear`

假设一个机床加工环境，有一个 XY 的二维坐标系，和一个不在坐标系内的轴 A。在 XY 坐标系内，刀具先从(0, 0)运动到(200000, 200000)（第 1 段轨迹）。然后，在 XY 方向从(200000, 200000)运动到(200000, 0)的同时，在 A 轴方向运动到 50000pulse 的位置，两者将同时到达各自指定的规划位置（第 2 段轨迹）。然后，在 XY 坐标内画一个圆心为原点，半径 200000，位于三四象限的半圆弧，同时在 A 轴方向运动 100000 的位置，两者将同时到达各自指定的规划位置（第 3 段轨迹），刀向跟随功能 GT_BufGear 的运动轨迹如图 6-27 所示。

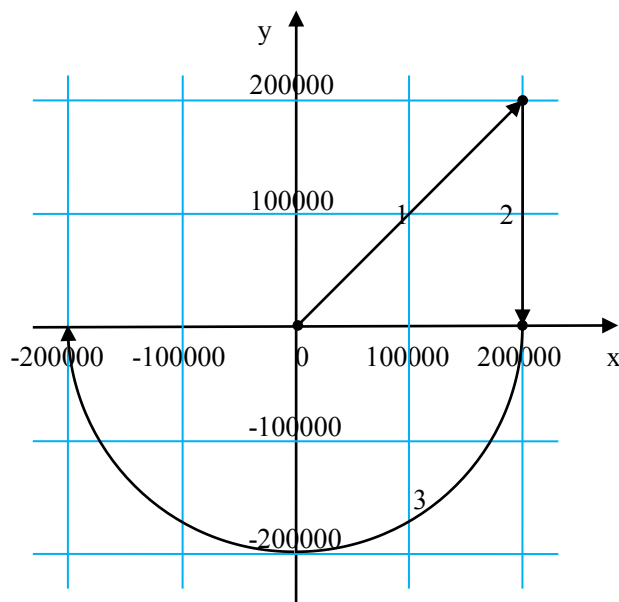


图 6-27 刀向跟随功能 GT_BufGear 的运动轨迹

```

.....
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;

// 清除坐标系1的FIFO0中的数据
sRtn = GT_CrdClear(1, 0);
// 向FIFO0缓存区写入一段直线插补数据
sRtn = GT_LnXY(
    1,                // 该插补段的坐标系是坐标系1
    200000, 200000,   // 该插补段的终点坐标(200000, 200000)
    100,              // 该插补段的目标速度: 100pulse/ms
    0.1,              // 插补段的加速度: 0.1pulse/ms^2
    0,                // 终点速度为0
    0);              // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 向FIFO0缓存区写入一段跟随运动数据,
// GT_BufGear()指令需要在所要跟随的插补段前
sRtn =GT_BufGear(

```

```

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

do
{
    // 坐标系在运动，查询到的run的值为1
    sRtn = GT_CrdStatus(1, &run, &segment, 0);
} while(1 == run);
.....

```

例程的运行结果如图 6-28 所示。

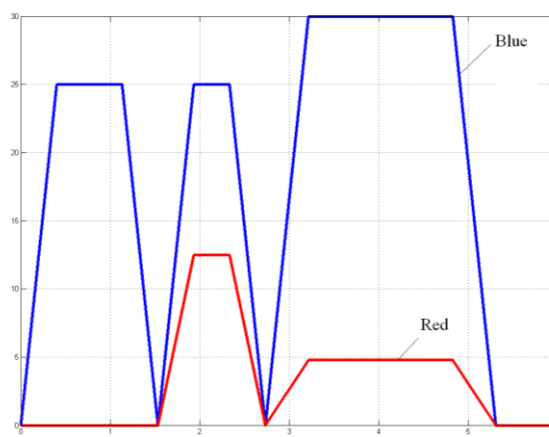


图 6-28 插补缓存区内的跟随运动速度图

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

在使用插补缓存区中的跟随运动功能时，需要注意以下内容：

- 1) **GT_BufGear** 指令需要在所要跟随的插补段前，不要间隔其他种类的指令，可以同时调用多个 **GT_BufGear** 指令来实现多个轴跟随插补运动。
- 2) 当暂停坐标系运动时，插补的合成速度减速到 0，跟随轴的速度也会为 0，如果希望重新恢复坐标系运动时，跟随轴仍能够实现跟随，不要调用 **GT_Stop** 指令停止跟随轴的运动，否则重新启动插补缓存区的运动时，跟随轴将无法完成正在进行的跟随运动。

例程 6-12 刀向跟随功能——实际工件加工

假设机床加工过程中，机床需要加工一个工件，XY 平面的尺寸如图 6-29 所示，工件的厚度为 500（单位均为：pulse），采用的工艺是用砂轮磨削工件外轮廓，加工时不但要调整刀具的 Z 方向高度，同时需要调整砂轮的 C 向角度（假设砂轮旋转一周的脉冲总数是 10000）。在加工工艺中，首先为避免撞到工件，需要将砂轮抬到一定的高度:2000（假设安全高度为 2000），从原点空走到 A(6000, 6000)位置，之后调整砂轮逆时针转 45°（对应为 1250 pulse），使之与工件的加工切向方向一致，然后降低砂轮高度到加工位置:-100。接着，沿直线方向加工到 B(6000, 12000)位置，从 B 到 C 为一圆弧，需要实时更新砂轮的方向，使之保持与工件的加工切向方向一致，所以需要调用 **GT_BufGear** 指令，跟随的位移量是 2500(90°)。随之后面的加工与之前类似，直至到加工点 F(36000, 6000)位置，此时需要抬刀使砂轮改变 90°，使之与 FA 段的加工方向一致，之后再放下砂轮至加工位置，直至加工完工件。

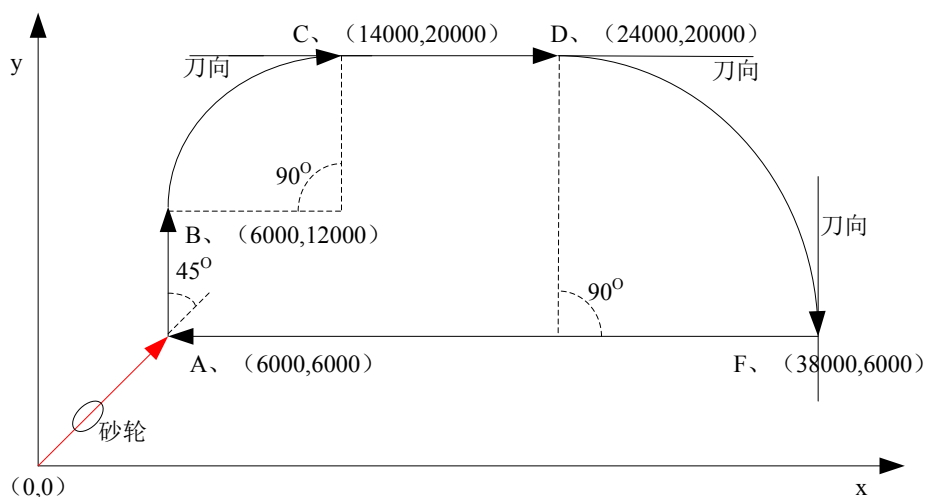


图 6-29 刀向跟随功能之工件尺寸和刀运动轨迹

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

```
.....
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;
// 清除坐标系1的FIFO0中的数据
sRtn = GT_CrdClear(1, 0);
```

```

// 向FIFO0缓存区写入一段直线插补数据
sRtn = GT_BufMove(
    1,           // 该插补段的坐标系是坐标系1
    3,           // 点位运动的轴号：第3轴
    500,         // 点位运动的目标位置：500 pulse
    10,          // 点位运动的目标速度：10 pulse/ms
    0.1,         // 点位运动的目标加速度：0.1 pulse/(ms*ms)
    1,           // 该点位运动是模态指令，等砂轮抬高后才执行下面的指令
    0);          // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 直线插补指令，到达A点
sRtn = GT_LnXY(1, 6000, 6000, 50, 0.1, 0, 0);
sRtn = GT_BufMove(
    1,           // 该插补段的坐标系是坐标系1
    4,           // 点位运动的轴号：第4轴
    -1250,       // 使其逆时针旋转45°与BA方向一致
    10,          // 点位运动的目标速度：10 pulse/ms
    0.1,         // 点位运动的目标加速度：0.1 pulse/(ms*ms)
    1,           // 该点位运动是模态指令，等角度到位后才执行下面的指令
    0);          // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 模态指令，将砂轮放下到加工高度位置
sRtn = GT_BufMove(1, 3, -100, 10, 0.1, 1, 0);
// 直线插补指令，到达B点
sRtn = GT_LnXY(1, 6000, 12000, 50, 0.1, 0, 0);
sRtn = GT_BufGear(
    1,           // 该插补段的坐标系是坐标系1
    4,           // 跟随运动的轴号：第4轴
    2500,        // 跟随运动的位移量：2500 pulse（相应为90°）
    0);          // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 到达C点位置
sRtn = GT_ArcXYR(
    1,           // 坐标系是坐标系1
    0, 14000,    // 该圆弧的终点坐标(0, 200000)
    8000,        // 半径：8000pulse
    0,           // 该圆弧是顺时针圆弧
    50,          // 该插补段的目标速度：50pulse/ms
    0.1,         // 该插补段的加速度：0.1pulse/ms^2
    0,           // 终点速度为0
    0);          // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 直线插补指令，到达D点
sRtn = GT_LnXY(1, 24000, 20000, 50, 0.1, 0, 0);
// 跟随运动的位移量：2500 pulse（相应为90°）
sRtn = GT_BufGear(1, 4, 2500, 0);
// 到达F点
sRtn = GT_ArcXYR(1, 38000, 6000, 14000, 0, 50, 0.1, 0, 0);
// 模态指令，将砂轮抬高到安全高度位置
sRtn = GT_BufMove(1, 3, 500, 10, 0.1, 1, 0);

```

```

// 模态指令, 将砂轮抬旋转至与FA方向一致
//该位置相对初始位置为225°, 即目标位置6250
sRtn = GT_BufMove(1, 4, 6250, 10, 0.1, 1, 0);
// 模态指令, 将砂轮放下到加工高度位置
sRtn = GT_BufMove(1, 3, -100, 10, 0.1, 1, 0);
// 直线插补指令, 到达A点
sRtn = GT_LnXY(1, 6000, 6000, 50, 0.1, 0, 0);
// 启动运动
sRtn = GT_CrdStart(1, 0);
do
{
    sRtn = GT_CrdStatus(1, &run, &segment, 0);
    // 坐标系在运动, 查询到的 run 的值为 1
    }while(run == 1);
.....

```

8. 静态插补缓冲区

插补缓冲区默认为“动态缓冲区”，即本次压入插补缓冲区的数据在这些数据段运动完成后会被清除，不会保留在缓冲区。在需要使用插补运动重复执行相同轨迹的运动时，就需要在每次运动结束后向缓冲区压入同样的轨迹数据，这样显然空耗了 CPU 资源。“静态插补缓冲区”模式下，压入插补缓冲区的数据在运动结束后不会被清除，仍然保留在缓冲区内，只需要重新启动运动，就可以重复执行先前的运动轨迹。目前，静态插补缓冲区的大小为 1000。若需要工作在静态缓冲区模式，需要调用相应的指令进行模式切换。

例程 6-12 静态插补缓冲区模式

```

short sRtn;
short crd = 1, fifo = 0;
.....
//控制器初始化, 建立插补坐标系

//先使用静态模式输入数据
sRtn = GT_SetCrdBufferMode(crd, CRD_BUFFER_STATIC_INPUT, fifo); //输入数据模式
sRtn = GT_CrdClear(crd, fifo);
sRtn = GT_LnXY(crd, 20000, 10000, 50, fifo);
sRtn = GT_LnXY(crd, 0, 0, 50, fifo);
sRtn = GT_SetCrdBufferMode(crd, CRD_BUFFER_STATIC_READY, fifo); //结束输入数据
sRtn = GT_CrdStart(crd, fifo);
//再次调用启动插补运动可以继续执行上述插补段数据
sRtn = GT_CrdStart(crd, fifo);

//切换到动态模式
sRtn = GT_SetCrdBufferMode(crd, CRD_BUFFER_DYNAMIC_DEFAULT, fifo);
sRtn = GT_CrdClear(crd, fifo);
sRtn = GT_LnXY(crd, 20000, 20000, 50, fifo);
sRtn = GT_CrdStart(crd, fifo);

```

第7章 访问硬件资源

7.1 本章简介



本手册中所有字体为蓝色的指令（如 [GT_PrflTrap](#)）均带有超级链接，点击可跳转至指令说明。

运动控制器包含的硬件资源分为如下几类：

表 7-1 运动控制器硬件资源

资源	说明
数字量输入	正负限位、驱动报警、原点、通用输入
数字量输入	电机到位信号输入（arrive）
数字量输出	报警清除、伺服使能、通用输出
编码器	对外部编码器的脉冲输出进行计数
模拟量输出（DAC）	电压输出
模拟量输入（ADC）	电压输入

本章介绍如何在应用程序中使用这些硬件资源。

7.2 访问数字 IO

7.2.1 指令列表

表 7-2 访问数字 IO 指令列表

指令	说明	页码
GT_GetDi	读取数字 IO 输入状态	190
GT_GetDiRaw	读取数字 IO 输入状态的原始值	191
GT_GetDiReverseCount	读取数字量输入信号的变化次数	191
GT_SetDiReverseCount	设置数字量输入信号的变化次数的初值	217
GT_SetDo	设置数字 IO 输出状态	217
GT_SetDoBit	按位设置数字 IO 输出状态	218
GT_SetDoBitReverse	使数字量输出信号输出定时脉冲信号	218
GT_GetDo	读取数字 IO 输出状态	192

7.2.2 重点说明

调用 [GT_GetDi](#) 指令可以读取限位、驱动报警、原点、通用输入、手轮接口这些数字量输入接口的输入电平状态。其中，手轮接口为 5V 电平输入，其余 IO 为 24V 电平输入。

调用 [GT_SetDo](#) 指令可以设置伺服使能、报警清除、通用输出这些数字量输出接口的输出电平状态。

调用 `GT_GetDiRaw` 指令可以读取数字量输入接口的原始电平状态。

`GT_GetDiReverseCount` 指令用来读取数字量输入的变化次数，当数字量输入由 0 变为 1，或者由 1 变为 0，该次数就会增加一次。`GT_SetDiReverseCount` 指令用来设置数字量变化次数计数器的初值。

`GT_SetDoBitReverse` 指令用来使数字量输出信号输出一个定时的脉冲，例如，假设当前通用数字量输出信号 1 是高电平，当调用指令 `GT_SetDoBitReverse(MC_GPO, 1, 0, 100)`；则该数字量信号将会发出一个 $100 \times 250\mu\text{s} = 25\text{ms}$ 时间宽度的负脉冲。

下面详细说明专用数字量 IO 的含义和用途。

1. 正负限位（di）

控制器为每个轴提供两个输入作为行程控制开关。如图 7-1 所示。

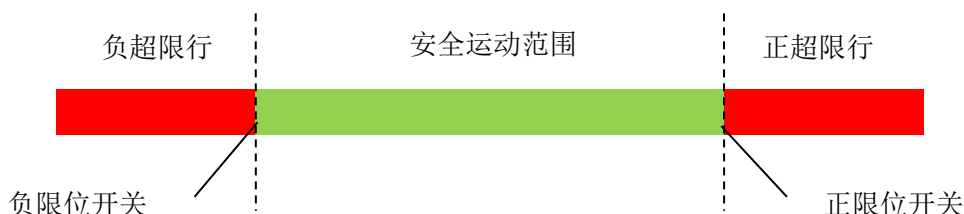


图 7-1 限位开关示意图

把轴配置成正负限位有效后，如果轴运动超越了限位开关所在位置，限位开关触发，运动控制器禁止触发限位方向上运动，同时该轴的限位触发状态置 1（通过调用 `GT_GetSts` 可以查看此状态标志位）。离开限位回到安全运动范围以后，需要调用指令 `GT_ClrSts` 清除限位触发状态，才能使控制轴回到正常运动状态。



注意

限位触发后，轴紧急停止，调用 `GT_GetSts` 读取状态，会发现正限位触发标志位或负限位触发标志位置 1。用户应按如下步骤操作：

1. 调用指令让轴向反方向运动到安全范围内停下。即如果正限位触发就让轴往负方向运动，负限位触发就让轴往正方向运动。
2. 调用 `GT_ClrSts` 清除限位触发状态。
3. 限位触发后的紧急停止可能会因为运动过冲而引入位置误差，建议用户重新回零。回零方法详见 Home 回原点。

如果用户不使用限位开关，可以通过控制器配置，将轴的正负限位配置为无效。

限位开关默认为高电平触发。通过调用 `GT_LmtSns` 指令可以修改限位开关的触发电平。

2. 驱动器报警信号（di）

控制器提供专用的驱动报警信号输入接口。该接口应该与驱动器的报警输出信号接口相接。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能，急停运动规划，同时该轴报警触发标志置 1（通过调用 `GT_GetSts` 可以查看此状态标志位）。

驱动报警信号是低电平触发。

**注意**

驱动报警后，轴紧急停止，调用 `GT_GetSts` 读取状态，会发现驱动器报警标志位置 1。用户应当执行以下操作：

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. 调用 `GT_ClrSts` 清除报警，重新回机床原点。

3. 原点信号 (di)

原点信号又叫 Home 信号，即电机回到原点时控制它停止的信号。用户负责在应用程序中捕获到 Home 信号，然后控制电机停止，具体操作请参考“第八章高速硬件捕获”。

原点信号默认为下降沿触发，通过调用 `GT_SetCaptureSense` 指令可以修改原点开关的触发沿。

4. 伺服使能 (do)

伺服使能信号默认与 axis 关联，用户不能直接调用 `GT_SetDo` 或 `GT_SetDoBit` 对伺服使能设置输出电平。默认情况下，调用 `GT_AxisOn`，伺服使能输出低电平，该 do 置 0。如果驱动器是低电平使能，则必须在控制器配置将伺服使能输出设置为输出电平取反，具体操作请参考配置 do。

伺服使能信号的硬件接口在端子板 25pin 轴接口中可以找到（请参考硬件说明书）。如果用户使用的驱动器不含伺服使能输入信号，或者用户不需要使用此信号，可以取消伺服使能 do 与 axis 的关联（操作请参考配置 do），这样用户可以直接调用 `GT_SetDo` 或 `GT_SetDoBit` 对伺服使能设置输出电平。

5. 报警清除 (do)

某些驱动器上有报警清除数字输入接口，控制器提供报警清除数字量输出接口，方便用户使用此功能。但是此信号不能清除所有的驱动报警情况。例如，在编码器接线不良引起驱动报警时，就无法使用清除报警信号来清除。因此，当报警无法清除时，用户需要具体问题具体分析。用户可以调用 `GT_SetDo` 或 `GT_SetDoBit` 来设置清除报警信号的电平。

7.2.3 例程

例程 7-1 访问数字 IO

有一个按键连接端子板上通用输入 EXI3，常为低电平，按下按键，输入高电平。端子板通用输出 EXO6 上连接一个指示灯，当 EXO6 输出低电平时指示灯亮，输出高电平时指示灯灭。开始指示灯一直灭，用户希望不断读取轴 1 正负限位的数字量输入变化次数，并将结果输出到显示器。在正负限位变化次数都超过 10 次之后归零重新计数，并且使指示灯一直亮起来。用户可以随时按键停止这种检测。

```

.....
// 指令返回值
short sRtn;
// 通用输入读取值
long lGpiValue;
// 正限位输入变化次数
unsignedlong lPosLmtReverseCount;
// 负限位输入变化次数
unsignedlong lNegLmtReverseCount;

```

```

// 初始化变化次数为0
lPosLmtReverseCount = 0;
lNegLmtReverseCount = 0;

// 读取EXI3输入值
sRtn = GT_GetDi(MC_GPI, &lGpiValue);
// 此函数为检测指令返回值函数, 参看例程3-1检测GT指令是否正常执行
commandhandler("GT_GetDi", sRtn);
// 如果为高电平, 说明按键正在被按下, 则不检测, 返回 1
if( lGpiValue & (1<<3))
    return 1;
// EXO6输出高电平, 使指示灯灭
sRtn = GT_SetDo(MC_GPO, 1<<6);
commandhandler("GT_SetDo", sRtn);
// 按键没有被按下, 循环。如果EXI3输入值为高电平, 即按键按下, 则退出循环
while( ! (lGpiValue & (1<<3)))
{
    // 读取正限位输入变化次数
    sRtn = GT_GetDiReverseCount(
        MC_LIMIT_POSITIVE,      // 指定数字IO类型是正限位
        1,                      // 指定正限位1
        &lPosLmtReverseCount,    // 读取的值
        1);                    // 一次读取一个数字量
    commandhandler("GT_GetDiReverseCount", sRtn);
    // 读取负限位输入变化次数
    sRtn = GT_GetDiReverseCount(
        MC_LIMIT_NEGATIVE,      // 指定数字IO类型是负限位
        1,                      // 指定负限位1
        &lNegLmtReverseCount,    // 读取的值
        1);                    // 一次读取一个数字量
    commandhandler("GT_GetDiReverseCount", sRtn);
    // 将结果输出到显示器
    printf("PosLmtReverseCount = %d\n NegLmtReverseCount = %d\n",
        lPosLmtReverseCount, lNegLmtReverseCount);
    // 如果正负限位的输入变化次数都超过10次
    if(( lPosLmtReverseCount >= 10) && ( lNegLmtReverseCount >= 10) )
    {
        // 重新归为0
        lPosLmtReverseCount = 0;
        lNegLmtReverseCount = 0;
        // 设置正限位输入变化次数
        sRtn = GT_SetDiReverseCount(
            MC_LIMIT_POSITIVE,    // 指定数字IO类型是正限位
            1,                    // 指定正限位1
            &lPosLmtReverseCount, // 读取的值

```

```

1); // 一次读取一个轴
commandhandler("GT_SetDiReverseCount", sRtn);
// 设置负限位输入变化次数
sRtn = GT_SetDiReverseCount(
    MC_LIMIT_NEGATIVE, // 指定数字IO类型是负限位
    1, // 指定负限位1
    &lNegLmtReverseCount, // 读取的值
    1); // 一次读取一个轴
commandhandler("GT_SetDiReverseCount", sRtn);
// EXO6输出高电平，使指示灯亮
sRtn = GT_SetDoBit(
    MC_GPO, // 指定数字IO类型是通用输出
    7, // 指定第7个通用输出，即EXO6
    0); // 输出低电平
commandhandler("GT_SetDoBit", sRtn);
}
// 不断读取通用输入，已检测EXI3的电平状态
sRtn = GT_GetDi(MC_GPI, &lGpiValue);
commandhandler("GT_GetDi", sRtn);
}
.....

```

7.3 访问编码器

7.3.1 指令列表

表 7-3 访问编码器指令列表

指令	说明	页码
GT_GetEncPos	读取编码器位置	192
GT_GetEncVel	读取编码器速度	192
GT_SetEncPos	修改编码器位置	219

控制器内部为每个轴配置了脉冲计数装置。控制器默认的脉冲计数源是外部编码器。如果用户在接线时将外部编码器的信号与端子板 25pin 轴接口的编码器信号接在一起，就可以调用上述指令读取外部编码器的值。如果用户没有接外部编码器反馈信号，例如，使用步进电机时没有编码器反馈部件，如图 7-2 所示，则用户调用 GT_GetEncPos 读取的编码器位置为 0。



图 7-2 开环控制系统示意图

控制器还可以配置脉冲计数源是脉冲计数器（操作详见配置 step）。调用 GT_GetEncPos 读取的将是运动控制器向驱动器发出的脉冲个数。因此，即使不接反馈部件，也可以读取变化的位置值。

调用 GT_SetEncPos 修改编码器位置的值。例如，设置轴 1 的编码器位置为 0，则接下来的编码器计数从 0 开始。若设置为 1000，则从 1000 开始。

7.3.2 例程

例程 7-2 读取 8 个轴编码器和辅助编码器位置值

```
#include "gts.h"
int main(int argc, char* argv[])
{
    short sRtn, i;
    double enc[9];

    sRtn = GT_Open();
    commandhandler("GT_Open", sRtn);
    while(1)
    {
        // 读取8个编码的位置
        GT_GetEncPos(1, &enc[0], 8);
        // 读取辅助编码器的位置
        GT_GetEncPos(9, &enc[8], 1);
        for(i=0;i<9;++i)
        {
            printf("%.0lf", enc[i]);
        }
        printf("\r");
    }
    return 0;
}
```

例程 7-2-2 读取 MPG 位置值和信号

```
#include "gts.h"
int main(int argc, char* argv[])
{
    short sRtn;
    double mpgPos;
    long lGpiValue;
    sRtn = GT_Open();
    while(1)
    {
        // 读取辅助编码器的位置
        sRtn = GT_GetEncPos(11, &mpgPos, 1);
        sRtn = GT_GetDi(MC_MPG, &lGpiValue);
        printf("mpgPos =%lf,lGpiValue = %ld\r", mpgPos,lGpiValue);
    };
    return 0;
}
```

7.4 访问 DAC

7.4.1 指令列表

表 7-4 访问 DAC 指令列表

指令	说明	页码
GT_SetDac	设置 DAC 输出电压	216
GT_GetDac	读取 DAC 输出电压	190

7.4.2 重点说明

控制器的模拟量输出通道在闭环控制模式下，作为伺服电压输出控制通道，与驱动器连接，是不允许用户操作的。在开环控制模式下，可以作为通用的电压输出通道。上电复位后，控制器默认为脉冲（开环）控制模式，用户可以调用 DAC 指令列表中的指令来设置和读取 dac 输出电压。轴控制接口的 8 路模拟电压值与指令读取数值对应关系如表 7-5 所示。

表 7-5 轴控模拟电压值与指令读取数值对应关系

输入电压值 (V)	指令读取数值
-10	-32768
0	0
10	32767

非轴接口的 4 路模拟电压值与指令读取数值对应关系如表 7-6 所示。

表 7-6 非轴模拟电压值与指令读取数值对应关系

输入电压值 (V)	指令读取数值
0	0
10	32767

7.4.3 例程

例程 7-3 访问 DAC

轴 4 的 dac 输出通道输出 5V 电压，设置完后也可以读取该通道电压。

```

.....
// 指令返回值
short sRtn;
// 电压值
short sSetValue;
short sGetValue;
// 控制器复位，所有轴都为脉冲控制模式
sRtn = GT_Reset();
// 计算轴4的电压输出值

```

```

sSetValue = (short) 32767*5/10;
// 设置轴4的输出电压
sRtn = GT_SetDac(4, &sSetValue, 1);
// 读取轴4的输出电压值
sRtn = GT_GetDac(4, &sGetValue, 1);
.....

```

7.5 访问模拟量输入(仅适用于带模拟量功能端子板)

7.5.1 指令列表

表 7-7 访问模拟量输入指令列表

指令	说明	页码
GT_GetAdc	读取模拟量输入的电压值	181
GT_GetAdcValue	读取模拟量输入的数字转换值	182

7.5.2 重点说明

运动控制器提供了 8 路 12 位/16 位精度的外部模拟电压输入通道，最大采样频率为 8kHz。调用指令 GT_GetAdc 读取指定通道的外部输入模拟电压值。模拟电压值与指令读取数值对应关系如表 7-8 所示。

表 7-8 模拟电压值与指令读取数值对应关系

输入电压值 (V)	指令读取数值
-10	-26214
0	0
10	26214

7.5.3 例程

例程 7-4 访问 ADC

将 5V 电源并联接入 4 个 adc 输入通道，读取该输入通道的电压值和数字转换值。

```

.....
// 指令返回值
short sRtn;
// 电压值
double dGetVoltageValue[4];
// 数字转换值
short sGetDigitalValue[4];
// 读取4个通道的输入电压
sRtn = GT_GetAdc(1, &dGetVoltageValue[0], 4);
// 读取4个通道输入电压的数字转换值
sRtn = GT_GetAdcValue(1, &sGetDigitalValue[0], 4);
.....

```

第8章 高速硬件捕获

8.1 本章简介

捕获即当某一种信号触发时，运动控制器能准确记录触发时刻轴的位置信息。控制器提供四种捕获方式，Home 捕获，Index 捕获、探针(Probe)捕获和 HSIO 捕获。本章将详细介绍这四种捕获方式以及如何在应用程序中实现。



本手册中所有字体为蓝色的指令（如 [GT_PrfrTrap](#)）均带有超级链接，点击可跳转至指令说明。

8.2 指令列表

表 8-1 高速硬件捕获指令列表

指令	说明	页码
GT_SetCaptureMode	设置编码器捕获方式，并启动捕获	213
GT_GetCaptureMode	读取编码器捕获方式	186
GT_GetCaptureStatus	读取编码器捕获状态	186
GT_SetCaptureSense	设置捕获电平	214
GT_ClearCaptureStatus	清除捕获状态	173
GT_SetCaptureRepeat	设置重复捕获	213
GT_GetCaptureRepeatStatus	查询重复捕获状态	185
GT_GetCaptureRepeatPos	查询重复捕获位置值	185
GT_StopCapture	停止捕获	229

8.3 Home/Index 硬件捕获

8.3.1 重点说明

控制器提供 Home 信号数字量输入接口（详见“7.2 访问数字 IO”）。Home 捕获模式下，当运动控制器检测到 Home 信号接口下降沿（控制器默认下降沿捕获触发）时，FPGA 立刻锁存 Home 开关所对应的编码器位置，同时将该编码器轴的捕获状态标志位置 1，然后退出 Home 捕获模式。

Index 捕获模式下，当出现 Index（编码器 C 相）下降沿时，FPGA 立刻锁存该编码器位置，同时将该编码器轴的捕获触发标志位置 1，然后退出 Index 捕获模式。

Home 捕获和 Index 捕获默认都是下降沿触发。调用 [GT_SetCaptureSense](#) 可以修改三种捕获模式的触发沿。

调用 [GT_SetCaptureMode](#) 就会开启相应的捕获模式。控制器会一直检测是否有触发信号。当 Home 或 Index 捕获触发以后，对应的轴的触发标志会置 1。调用 [GT_GetCaptureStatus](#) 可以查看捕获状态和捕获时的位置信息。触发标志会保持为 1（触发），直到重新启动 Home 或 Index 捕获时，控制器会自动清除对应轴的捕获触发标志位。

8.3.2 例程

例程 8-1 Home/Index 捕获

该例程描述了怎样设置 Home 捕获和 Index 捕获。用户如直接使用此例程还无法实现整个捕获过程，需要控制端子上 home 数字量输入电平使 home 信号触发才行。具体回原点过程可以参考 8.4 和 8.5 节。

```
#include "gts.h"
#define ENCODER 1

void CaptureIndex(void)
{
    short sRtn, status;
    long pos;
    double encPos;

    // 启动Index捕获
    sRtn = GT_SetCaptureMode(ENCODER, CAPTURE_INDEX);
    commandhandler("GT_SetCaptureMode", sRtn);
    printf("\n");
    do
    {
        // 查询捕获状态
        GT_GetCaptureStatus(ENCODER, &status, &pos);
        // 读取编码器位置
        // 该指令和捕获无关，仅用于显示编码器位置
        GT_GetEncPos(ENCODER, &encPos);
        // 显示捕获状态和编码器位置
        printf("status = %d enc = %-8.0lf\r", status, encPos);
        // 当指定轴捕获触发时退出循环
    } while( 0 == status );

    // 显示捕获位置
    printf("\ncapture = %-8ld\n", pos);
}

void CaptureHome(void)
{
    short sRtn, status;
    long pos;
    double encPos;

    // 启动Home捕获
    sRtn = GT_SetCaptureMode(ENCODER, CAPTURE_HOME);
    commandhandler("GT_SetCaptureMode", sRtn);
}
```

```

printf("\n");
do
{
    // 查询捕获状态
    GT_GetCaptureStatus(ENCODER, &status, &pos);
    // 读取编码器位置
    // 该指令和捕获无关，仅用于显示编码器位置
    GT_GetEncPos(ENCODER, &encPos);
    // 显示捕获状态和编码器位置
    printf("status = %d enc = %-8.0lf\r", status, encPos);
    // 当指定轴捕获触发时退出循环
} while( 0 == status );
// 显示捕获位置
printf("\ncapture = %-8ld\n", pos);
}

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

    sRtn = GT_Open();
    commandhandler("GT_Open", sRtn);

    while(1)
    {
        // 选择捕获模式，H: Home捕获，I: Index捕获，Q: 退出程序
        printf("\n[H:Home|I:Index|Q:Quit]:");
        ch=toupper(getche());
        switch(ch)
        {
            case'H': CaptureHome();break;
            case'I': CaptureIndex(); break;
            case'Q': return 0;
        }
    }
}

```

8.4 Home 回原点

8.4.1 重点说明

- (1) 首先启动 Home 捕获。令工作台（轴）向原点（Home）开关方向运动（点位运动模式）。搜索距离就是在这个距离以内，工作台肯定可以运动到原点开关处。如图 8-1 所示。应该注意，不要让轴跑过原点开关位置时才启动 Home 捕获，这样自然会捕获失败。

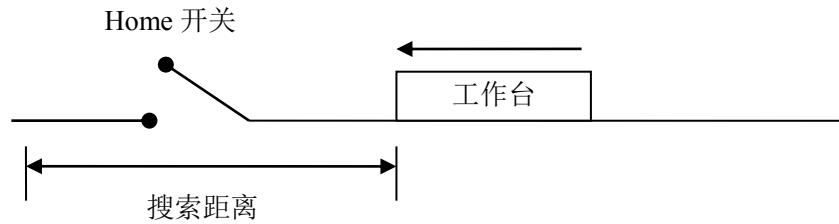


图 8-1 Home 回原点示意图 1

- (2) 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置。这时轴依然向前运动，将点位运动目标位置修改为“Home 捕获位置+偏移量”。一般会将原点开关处偏移一段距离处设为目标位置，如图 8-2 所示。

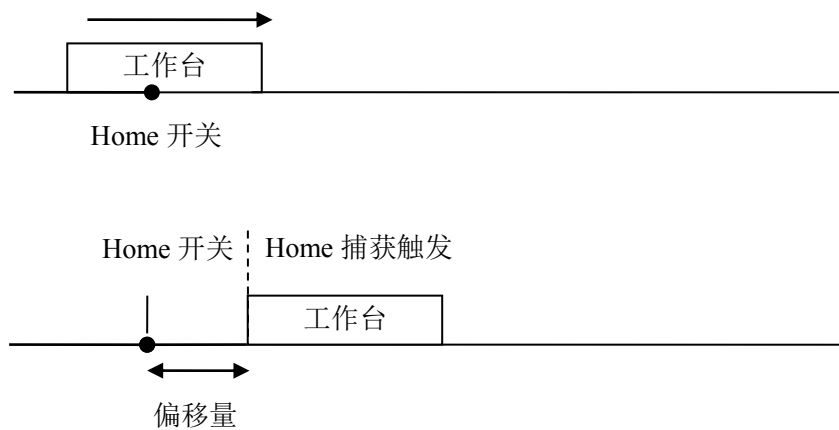


图 8-2 Home 回原点示意图 2

- (3) 等工作台停稳以后，调用 `GT_ZeroPos` 设置机械原点，如图 8-3 所示。



图 8-3 Home 回原点示意图 3

8.4.2 例程

例程 8-2 Home 回原点

该例程实现了上一节讲述的回原点过程。其中，搜索距离为 200000，偏移量为 2000。

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

#define AXIS 1

#define SEARCH_HOME -200000
```

```

#define HOME_OFFSET -2000

int main(int argc, char* argv[])
{
    short sRtn, capture;
    TTrapPrm trapPrm;
    long status, pos;
    double prfPos, encPos, axisPrfPos, axisEncPos;
    // 打开运动控制器
    sRtn = GT_Open();
    commandhandler("GT_Open", sRtn);
    // 复位运动控制器
    sRtn = GT_Reset();
    commandhandler("GT_Reset", sRtn);
    // 配置运动控制器
    // 注意：配置文件test.cfg取消了各轴的报警和限位
    sRtn = GT_LoadConfig("test.cfg");
    commandhandler("GT_LoadConfig", sRtn);
    // 清除指定轴的报警和限位
    sRtn = GT_ClrSts(AXIS);
    commandhandler("GT_ClrSts", sRtn);
    // 驱动器使能
    sRtn = GT_AxisOn(AXIS);
    commandhandler("GT_AxisOn", sRtn);
    // 启动Home捕获
    sRtn = GT_SetCaptureMode(AXIS, CAPTURE_HOME);
    commandhandler("GT_SetCaptureMode", sRtn);
    // 切换到点位运动模式
    sRtn = GT_PrftTrap(AXIS);
    commandhandler("GT_PrftTrap", sRtn);
    // 读取点位模式运动参数
    sRtn = GT_GetTrapPrm(AXIS, &trapPrm);
    commandhandler("GT_GetTrapPrm", sRtn);
    trapPrm.acc = 0.25;
    trapPrm.dec = 0.25;
    // 设置点位模式运动参数
    sRtn = GT_SetTrapPrm(AXIS, &trapPrm);
    commandhandler("GT_SetTrapPrm", sRtn);
    // 设置点位模式目标速度，即回原点速度
    sRtn = GT_SetVel(AXIS, 10);
    commandhandler("GT_SetVel", sRtn);
    // 设置点位模式目标位置，即原点搜索距离
    sRtn = GT_SetPos(AXIS, SEARCH_HOME);
    commandhandler("GT_SetPos", sRtn);
    // 启动运动
    sRtn = GT_Update(1<<(AXIS-1));

```

```

commandhandler("GT_Update", sRtn);
printf("\nWaiting for home signal...\n");
do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取捕获状态
    sRtn = GT_GetCaptureStatus(Axis, &capture, &pos);
    // 读取规划位置
    sRtn = GT_GetPrfPos(Axis, &prfPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(Axis, &encPos);
    printf("capture=%d prfPos=%.2lf encPos=%.2lfr", capture, prfPos, encPos);
    // 如果运动停止, 返回出错信息
    if( 0 == ( status& 0x400 ) )
    {
        printf("\nno home found\n");
        getch();
        return 1;
    }
    // 等待捕获触发
} while( 0 == capture );
// 显示捕获位置
printf("\ncapture pos = %ld\n", pos);
// 运动到"捕获位置+偏移量"
sRtn = GT_SetPos(Axis, pos + HOME_OFFSET);
commandhandler("GT_SetPos", sRtn);
// 在运动状态下更新目标位置
sRtn = GT_Update(1<<(Axis-1));
commandhandler("GT_Update", sRtn);

do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取规划位置
    sRtn = GT_GetPrfPos(Axis, &prfPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(Axis, &encPos);
    printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lfr", status, prfPos, encPos);
    // 等待运动停止
} while( status& 0x400 );
// 检查是否到达"Home捕获位置+偏移量"
if( prfPos != pos+HOME_OFFSET )
{
    printf("\nmove to home pos error\n");
}

```

```

    getch();
    return 2;
}
printf("\nHome finish\n");
// 延时一段时间, 等待电机停稳
Sleep(200);
printf("\nPress any key to set pos as 0...\n");
getch();
// 位置清零
sRtn = GT_ZeroPos(Axis);
commandhandler("GT_ZeroPos", sRtn);
// 读取规划位置
sRtn = GT_GetPrfPos(Axis, &prfPos);
// 读取编码器位置
sRtn = GT_GetEncPos(Axis, &encPos);
// 读取axis规划位置
sRtn = GT_GetAxisPrfPos(Axis, &axisPrfPos);
// 读取axis编码器位置
sRtn = GT_GetAxisEncPos(Axis, &axisEncPos);
printf("\nprfPos=%-10.0lf encPos=%-10.0lf axisPrfPos=%-10.0lf axisEncPos=%-10.0lf",
        prfPos, encPos, axisPrfPos, axisEncPos);
// 伺服关闭
sRtn = GT_AxisOff(Axis);
printf("\nGT_AxisOff()=%d\n", sRtn);
getch();
return 0;
}

```

8.5 Home+Index 回原点

8.5.1 重点说明

前三步与 Home 回原点完全相同。

- (1) 首先启动 Home 捕获。令工作台（轴）向原点（Home）开关方向运动（点位运动模式）。搜索距离就是在这个距离以内，工作台肯定可以运动到原点开关处。如图 8-4 所示。应该注意，不要让轴跑过原点开关位置时才启动 Home 捕获，这样自然会捕获失败。

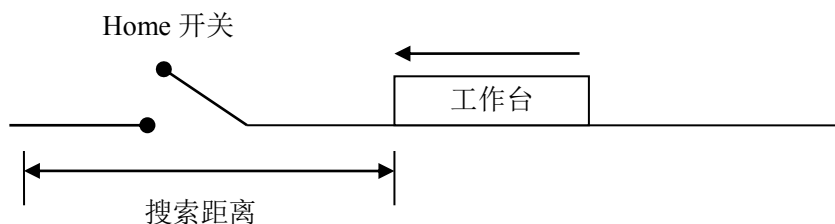


图 8-4 Home+Index 回原点示意图 1

- (2) 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置。这时轴依然向前运动，将点位运动目标位置修改为“Home 捕获位置+偏移量”。一般会在原点开关处偏移一段距离处设为原点。如图 8-5 所示。

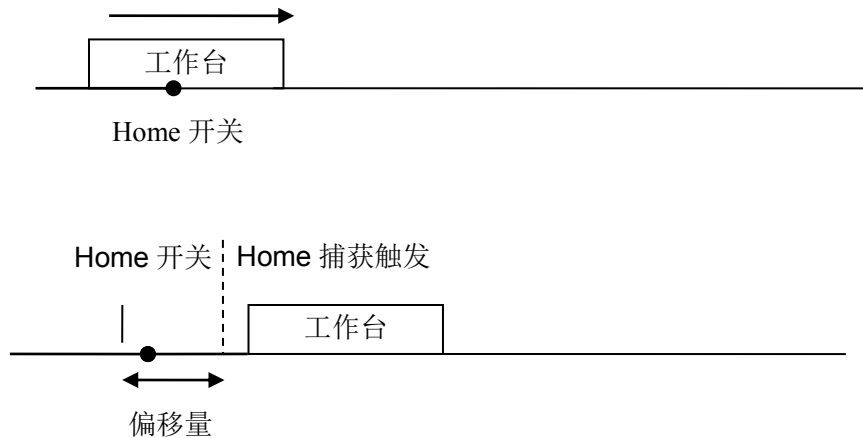


图 8-5 Home+Index 回原点示意图 2

- (3) 等工作台停稳以后，启动 Index 捕获。工作台继续点位运动，让其向反方向运动一段搜索距离。如图 8-6 所示。



图 8-6 Home+Index 回原点示意图 3

- (4) 工作台继续运动，寻找 Index 信号。当 Index 信号产生时，读取 Index 信号触发时工作台的实际位置。这时轴依然向前运动，将点位运动目标位置修改为“Index 捕获位置+偏移量”。如图 8-7 所示。



图 8-7 Home+Index 回原点示意图 4

- (5) 等工作台停稳以后，调用 `GT_ZeroPos` 设置机械原点。如图 8-8 所示。

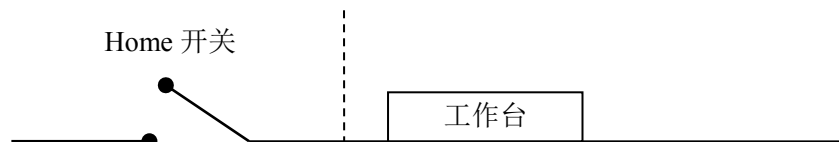


图 8-8 Home+Index 回原点示意图 5

8.5.2 例程

例程 8-3 Home+Index 回原点

```

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

// 定义轴号
#define AXIS 1
// 定义home回零搜索距离
#define SEARCH_HOME -200000
// 定义到home捕获位置的偏移量
#define HOME_OFFSET -2000
// 定义index回零搜索距离
#define SEARCH_INDEX 15000
// 定义到index捕获位置的偏移量
#define INDEX_OFFSET 1000

int main(int argc, char* argv[])
{
    // 定义返回值变量
    short sRtn;
    // 捕获状态
    short capture;
    // 点位运动参数结构体
    TTrapPrm trapPrm;
    // 轴状态
    long status;
    // 捕获位置
    long pos;
    // 分别是规划位置，编码器位置，轴的规划位置，轴的编码器位置
    double prfPos, encPos, axisPrfPos, axisEncPos;
    // 打开运动控制器
    sRtn = GT_Open();
    commandhandler("GT_Open", sRtn);
    // 复位
    sRtn = GT_Reset();
    commandhandler("GT_Reset", sRtn);
    // 加载配置文件
    sRtn = GT_LoadConfig("test.cfg");
    commandhandler("GT_LoadConfig", sRtn);
    // 清状态
    sRtn = GT_ClrSts(AXIS);
    commandhandler("GT_ClrSts", sRtn);

```

```

// 轴伺服使能
sRtn = GT_AxisOn(Axis);
commandhandler("GT_AxisOn", sRtn);
// 开启轴的home捕获功能
sRtn = GT_SetCaptureMode(Axis, CAPTURE_HOME);
commandhandler("GT_SetCaptureMode", sRtn);
// 设置轴为点位运动模式
sRtn = GT_PrflTrap(Axis);
commandhandler("GT_PrflTrap", sRtn);
// 读取点位运动参数
sRtn = GT_GetTrapPrm(Axis, &trapPrm);
commandhandler("GT_GetTrapPrm", sRtn);

trapPrm.acc = 0.25;
trapPrm.dec = 0.25;
// 设置点位运动参数
sRtn = GT_SetTrapPrm(Axis, &trapPrm);
commandhandler("GT_SetTrapPrm", sRtn);
// 设置目标速度
sRtn = GT_SetVel(Axis, 10);
commandhandler("GT_SetVel", sRtn);
// 设置目标位置
sRtn = GT_SetPos(Axis, SEARCH_HOME);
commandhandler("GT_SetPos", sRtn);
// 启动运动, 等待home信号触发
sRtn = GT_Update(1 <= (Axis-1));
commandhandler("GT_Update", sRtn);
printf("\nWaiting for home signal...\n");
// 如果home捕获没有触发, 就一直循环
do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取捕获状态
    sRtn = GT_GetCaptureStatus(Axis, &capture, &pos);
    // 读取规划位置
    sRtn = GT_GetPrflPos(Axis, &prflPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(Axis, &encPos);
    printf("capture=%d prflPos=%-10.1lf encPos=%-10.1lfr", capture, prflPos, encPos);
    // 电机已经停止, 说明整个搜索过程中 home 信号一直没有触发
    if( 0 == ( status & 0x400 ) )
    {
        printf("\nno home found");
        return 1;
    }
}

```

```

// 如果home信号已经触发, 则退出循环, 捕获位置已经在pos变量中保存
} while( 0 == capture );
printf("\ncapture pos = %ld\n", pos);
// 设定目标位置为捕获位置+偏移量
sRtn = GT_SetPos(Axis, pos + HOME_OFFSET);
commandhandler("GT_SetPos", sRtn);
// 启动运动
sRtn = GT_Update(1 << (Axis-1));
commandhandler("GT_Update", sRtn);

do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取规划位置
    sRtn = GT_GetPrfPos(Axis, &prfPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(Axis, &encPos);
    printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lf\r", status, prfPos, encPos);
} while( status & 0x400 );

if( prfPos != pos + HOME_OFFSET )
{
    printf("\nmove to home pos error");
    return 2;
}
// home捕获完毕
printf("\nHome finish\n");
Sleep(200);
// 启动index捕获
sRtn = GT_SetCaptureMode(Axis, CAPTURE_INDEX);
commandhandler("GT_SetCaptureMode", sRtn);
// 设置当前位置+index 搜索距离为目标位置
sRtn = GT_SetPos(Axis, (long)(prfPos + SEARCH_INDEX));
commandhandler("GT_SetPos", sRtn);
// 启动运动
sRtn = GT_Update(1 << (Axis-1));
commandhandler("GT_Update", sRtn);
// 等待index捕获信号触发
printf("\nWaiting for index signal...\n");

do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取捕获状态

```

```

sRtn = GT_GetCaptureStatus(Axis, &capture, &pos);
// 读取规划位置
sRtn = GT_GetPrfPos(Axis, &prfPos);
// 读取编码器位置
sRtn = GT_GetEncPos(Axis, &encPos);
printf("capture=%d prfPos=%-10.1lf encPos=%-10.1lf\r", capture, prfPos, encPos);
// 电机已经停止, 说明整个搜索过程中 index 信号一直没有触发
if( 0 == ( status& 0x400 ) )
{
    printf("\nno index found\n");
    getch();
    return 1;
}
// 如果index信号已经触发, 则退出循环, 捕获位置已经在pos变量中保存
}while( 0 == capture );

printf("\ncapture pos = %ld\n", pos);
// 设置捕获位置+index偏移量为目标位置
sRtn = GT_SetPos(Axis, pos+ INDEX_OFFSET);
commandhandler("GT_SetPos", sRtn);
// 启动运动
sRtn = GT_Update(1<<(Axis-1));
commandhandler("GT_Update", sRtn);

do
{
    // 读取轴状态
    sRtn = GT_GetSts(Axis, &status);
    // 读取规划位置
    sRtn = GT_GetPrfPos(Axis, &prfPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(Axis, &encPos);
    printf("status=0x%-8lx prfPos=%-10.1lf encPos=%-10.1lf\r", status, prfPos, encPos);
}while( status& 0x400 );

if( prfPos != pos+ INDEX_OFFSET )
{
    printf("\nmove to index pos error\n");
    getch();
    return 2;
}
// home+index捕获完毕
printf("\nHome+Index finish\n");
printf("\nPress any key to set pos as 0...\n");
getch();

```

```

Sleep(200);
// 更新原点位置
sRtn = GT_ZeroPos(Axis);
printf("GT_ZeroPos", sRtn);

sRtn = GT_GetPrfPos(Axis, &prfPos);
sRtn = GT_GetEncPos(Axis, &encPos);
sRtn = GT_GetAxisPrfPos(Axis, &axisPrfPos);
sRtn = GT_GetAxisEncPos(Axis, &axisEncPos);
printf("\nprfPos=%-10.0lf encPos=%-10.0lf axisPrfPos=%-10.0lf axisEncPos=%-10.0lf",
      prfPos, encPos, axisPrfPos, axisEncPos);
getch();
// 伺服关闭
sRtn = GT_AxisOff(Axis);
printf("\nGT_AxisOff()=%d\n", sRtn);
return 0;
}

```

8.6 探针捕获

8.6.1 重点说明

探针捕获一般用于测量行业。当探针到达某个物体的边沿，控制器立刻捕获当前轴的位置并使轴停止。多测量边沿上的几个点，就可以准确定位物体。控制器探针捕获的硬件接口是通用输入输出（GPIO）。当探针触发后，设置了探针捕获模式的轴将能得到当前轴的编码位置，若没有设置探针捕获的轴，调用 `GT_GetCaptureStatus` 时获取的捕获值为 0。

8.6.2 例程

例程 8-4 探针捕获

该例程启动 1 轴的探针捕获，其余轴不启动。当轴运动在搜索范围内，GPIO 高电平触发时，捕获才能成功。

```

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

#define Axis 1

#define SEARCH_PROBE -2000000 // 假设在该路程内能找到探针信号
int main(int argc, char* argv[])
{
    short sRtn, capture;

```

```

TTrapPrm trapPrm;
long status, pos;
double prfPos, encPos;

// 打开运动控制器
sRtn = GT_Open();
// 复位运动控制器
sRtn = GT_Reset();
// 配置运动控制器
sRtn = GT_LoadConfig("test.cfg");
// 清除指定轴的报警和限位
sRtn = GT_ClrSts(AXIS);
// 驱动器使能
sRtn = GT_AxisOn(AXIS);
// 启动Probe 捕获
sRtn = GT_SetCaptureMode(AXIS, CAPTURE_PROBE);
// 切换到点位运动模式
sRtn = GT_PrftTrap(AXIS);
// 读取、设置点位模式运动参数
sRtn = GT_GetTrapPrm(AXIS, &trapPrm);
trapPrm.acc = 0.25;
trapPrm.dec = 0.25;
sRtn = GT_SetTrapPrm(AXIS, &trapPrm);
// 设置点位模式目标速度，即回原点速度
sRtn = GT_SetVel(AXIS, 10);
// 设置点位模式目标位置，即原点搜索距离
sRtn = GT_SetPos(AXIS, SEARCH_PROBE);
// 启动运动
sRtn = GT_Update(1<<(AXIS-1));
do
{
    // 读取轴状态
    sRtn = GT_GetSts(AXIS, &status);
    // 读取捕获状态
    sRtn = GT_GetCaptureStatus(AXIS, &capture, &pos);
    // 读取规划位置
    sRtn = GT_GetPrfPos(AXIS, &prfPos);
    // 读取编码器位置
    sRtn = GT_GetEncPos(AXIS, &encPos);
    printf("capture=%d prfPos=%lf encPos=%lfr", capture, prfPos, encPos);
    // 如果运动停止，返回出错信息
    if( 0 == ( status& 0x400 ) )
    {
        printf("\nno Probe found\n");
        getch();
        return 1;
    }
}

```

```

    }
    // 等待捕获触发
    }while( 0 == capture );
    // 显示捕获位置
    printf("\nProbe captured pos = %ld\n", pos);
    // 伺服关闭
    sRtn = GT_AxisOff (AXIS);
    printf("\nGT_AxisOff()=%d\n", sRtn);
    getch();
    return 0;
}

```

8.7 HSIO 捕获

8.7.1 重点说明

运动控制器端子板共有 2 个位置比较输出通道（也即 HSIO 口），CN14 HSIO 的 1、6 脚差分输出 HSIO0，2、7 脚差分输出 HSIO1。5 脚是 5V，9 脚是地。位置比较脉冲输出电压为 5V。

使用 `GT_SetCaptureMode` 指令，把某一编码器的捕获模式设置为 `CAPTURE_HSIO0` 或 `CAPTURE_HSIO1`，HSIO 作为捕获源时使用上升沿触发，即可在 HSIO0 或 HSIO1 口输出上升沿的同时锁存指定编码器的值。

8.7.2 例程

例程 8-5 HSIO 捕获用法示例

```

long pBuf = 1000, pSts;
short pVal;
// 设置当1轴编码器位置为1000时HSIO0输出脉冲
GT_CompareData(1,           // 对1轴进行位置比较输出
               1,           // 需要进行比较的数据源为外部编码器
               0,           // 到达比较位置后输出脉冲
               0,           // 初始状态
               500,         // 脉冲宽度为 500us
               pBuf,        // HSIO0 的比较位置缓冲区数据起始地址
               1,           // HSIO0 的比较位置缓冲区数据长度
               NULL,
               0);

// 设置 HSIO0 输出时，同时锁存 1、2 轴编码器
rtn = GT_SetCaptureMode(1, CAPTURE_HSIO0);
rtn = GT_SetCaptureMode(2, CAPTURE_HSIO0);

// 当编码器到达 1000 脉冲时锁存 1、2 轴编码器，此时可使用以下代码查询捕获情况
rtn = GT_GetCaptureStatus(1, &pSts, &pVal);

```

```
rtn = GT_GetCaptureStatus(2, &pSts, &pVal);
```

8.8 重复捕获

使用 `GT_SetCaptureRepeat` 指令可以进行多次捕获，即重复捕获。设置重复捕获后，当捕获源被触发时，控制器会锁存指定编码器的值，并重新等待捕获信号再次触发。每个轴每次最多可以连续锁存 256 个位置值，捕获次数可调用 `GT_GetCaptureRepeatStatus` 查询，捕获到的位置值可以使用 `GT_GetCaptureRepeatPos` 获得。

例程 8-6 重复捕获使用说明

用法如下：

```
// 若需使用 HSIO0 作为捕获源，设置 1 轴编码器重复捕获 150 次，可调用以下指令
```

```
rtn = GT_SetCaptureMode(1, CAPTURE_HSIO0);
```

```
rtn = GT_SetCaptureRepeat(1, 150);
```

```
// 在运动过程中，可随时调用以下指令查询捕获状态以及捕获位置值
```

```
short *pSts;
```

```
longpValue[256];
```

```
unsignedlongpClk;
```

```
rtn = GT_GetCaptureRepeatStatus(1, &pSts);
```

```
// 假设调用 GT_GetCaptureRepeatPos 指令时 HSIO0 已输出 38 个脉冲，则 pSts 的值为 38，  
再 调用以下指令可读取已捕获的 38 个位置值。
```

```
rtn = GT_GetCaptureRepeatPos(1, &pValue, 1, 38);
```

```
// 若再次调用 GT_GetCaptureRepeatPos 指令时 HSIO0 已输出 48 个脉冲，可以调用以下指  
令 读取第 39 个到 48 个位置值。
```

```
rtn = GT_GetCaptureRepeatPos(1, &pValue, 39, 10);
```

第9章 安全机制

9.1 本章简介

本章介绍运动控制器提供给用户的所有可用安全机制，包括：限位、报警、平滑停止、紧急停止、跟随误差极限停止。



本手册中所有字体为蓝色的指令（如 `GT_PrflTrap`）均带有超级链接，点击可跳转至指令说明。

9.2 限位

运动控制器能够通过安装限位开关或者设置软限位来限制各轴的运动范围，如图 9-1 所示。

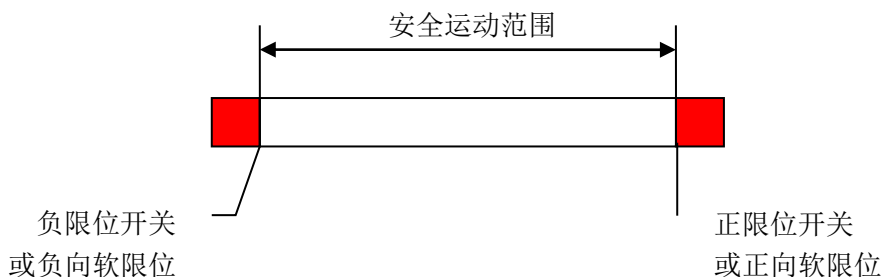


图 9-1 轴运动范围

工作台碰到限位开关或者规划位置超越软限位时，运动控制器紧急停止工作台的运动。限位触发以后，运动控制器禁止触发限位方向上运动，同时该轴的限位触发状态置 1。离开限位回到安全运动范围以后，需要调用指令 `GT_ClrSts` 清除限位触发状态，才能使控制轴回到正常运动状态。

控制器复位后，默认软限位是无效的，没有触发的。

9.2.1 指令列表

表 9-1 软限位指令列表

指令	说明	页码
<code>GT_SetSoftLimit</code>	设置轴正向软限位和负向软限位	224
<code>GT_GetSoftLimit</code>	读取轴正向软限位和负向软限位	199

9.2.2 重点说明

应当在回原点以后再设置软限位。正向软限位必须大于负向软限位。软限位和限位开关可以同时使用，当软限位触发时也会置起限位触发标志。

限位触发以后使用急停加速度紧急停止。默认急停加速度为 1000pulse/ms^2 ，如何设置急停加速度请参见“4.3.6 配置 profile”。

9.2.3 例程

例程 9-1 软限位使用

该例程设置了第一轴的软限位，正向在 20000 处，负向在-20000 处。启动第一轴的点位运动后，通过下图 9-2 可以看到，当运动超过正向软限时，限位信号触发，运动停止。

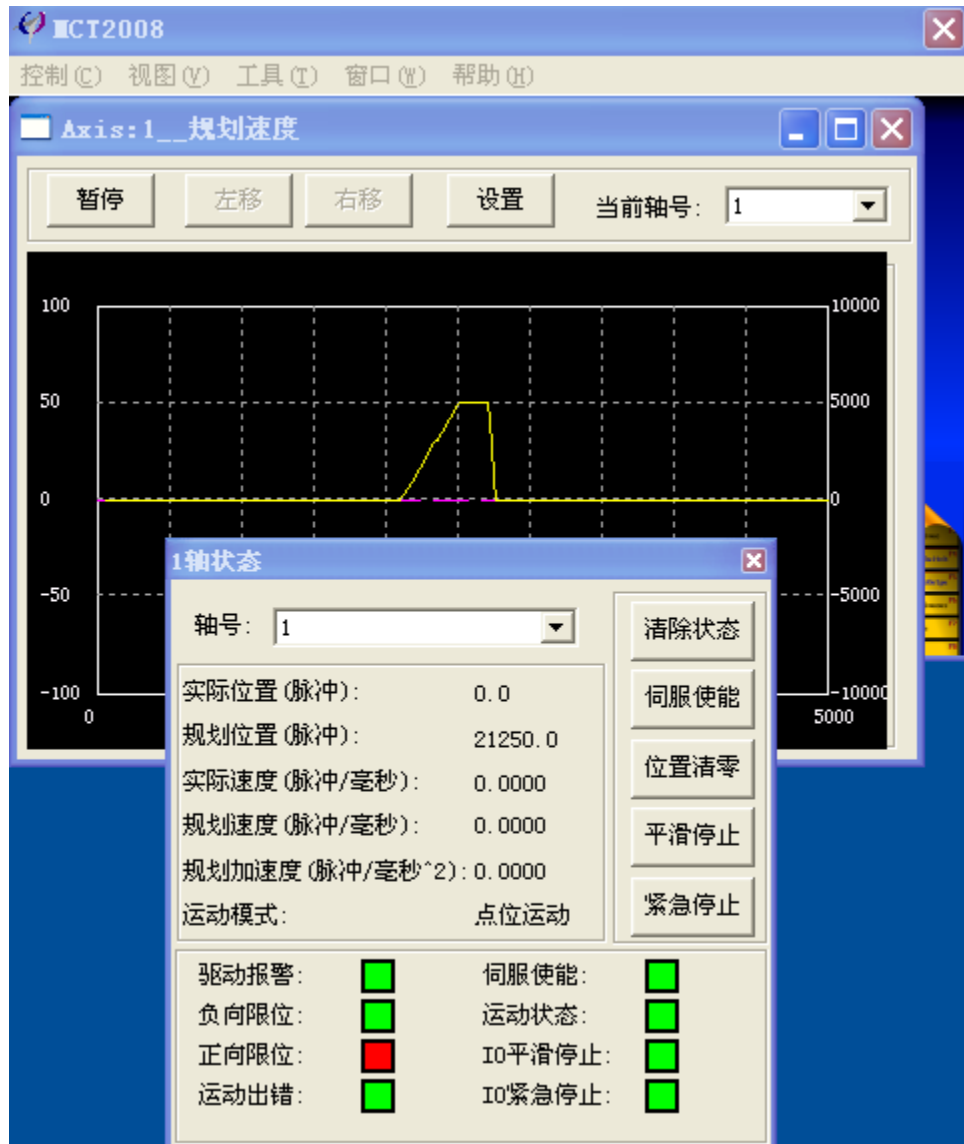


图 9-2 软限位触发

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

#define AXIS 1
int main(int argc, char* argv[])
{
    // 指令返回值
    short sRtn;
```

```

// 轴状态变量
long sts;
// 点位运动结构体变量
TTrapPrm trap;
// 规划位置
double prfPos;

// 打开运动控制器
sRtn = GT_Open();
// 复位
sRtn = GT_Reset();
// 控制器配置
sRtn = GT_LoadConfig("test.cfg");
// 清除轴状态
sRtn = GT_ClrSts(1, 8);
// 设置软限位
sRtn = GT_SetSoftLimit(Axis, 20000, -20000);
// 将第一轴设置为点位运动模式
sRtn = GT_PrftTrap(Axis);
// 设置点位运动参数
sRtn = GT_GetTrapPrm(Axis, &trap);
trap.acc = 0.125;
trap.dec = 0.125;
sRtn = GT_SetTrapPrm(Axis, &trap);
// 设置点位运动目标速度
sRtn = GT_SetVel(Axis, 50);
// 设置点位运动目标位置
sRtn = GT_SetPos(Axis, 1000000L);
// 启动点位运动
sRtn = GT_Update(1<=<(Axis-1));

while(!kbhit())
{
    // 读取第一轴轴状态
    sRtn = GT_GetSts(Axis, &sts);
    // 读取第一轴规划位置
    sRtn = GT_GetPrfPos(Axis, &prfPos);
    printf("sts=0x%-8lx prfPos=%-10.2lfr", sts, prfPos);
}
return 0;
}

```

9.3 报警

运动控制器提供专用的驱动报警信号输入接口。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能，急停运动规划，同时该轴报警触发标志置 1。

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) 调用 `GT_ClrSts` 清除报警，重新回机床原点。

9.4 平滑停止和急停

运动控制器的每个轴都可以定义平滑停止 IO 和急停 IO。

当平滑停止 IO 输入为触发电平时（触发电平可以设置），运动控制器自动平滑停止所关联的控制轴，并将轴状态字（bit7）置 1。

当急停 IO 输入为触发电平时（触发电平可以设置），运动控制器自动紧急停止所关联的控制轴，并将轴状态字（bit8）置 1。

IO 平滑停止或者 IO 急停完成以后，必须调用 `GT_ClrSts` 指令清除停止标志位（bit7 和 bit8），才能继续运动。

9.5 跟随误差极限

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。图 4-13 为跟随误差示意图，如果位置误差超越所设定的跟随误差极限，运动控制器自动紧急停止该轴的运动，同时该轴跟随误差超限标志置 1。

如何设置跟随误差极限请参考 4.3.5 节。

第10章 运动程序

10.1 本章简介

本章将介绍下载在控制器中运行的程序——运动程序。用户想要使用运动程序，需要了解相应的语法，以及下载步骤。本章将一一介绍。



提示

本手册中所有字体为蓝色的指令（如 [GT_PrflTrap](#)）均带有超级链接，点击可跳转至指令说明。

10.2 运动程序概述

为了表述方便，直接在 PC 机上调用动态链接库发送指令访问控制器的程序称为“应用程序”，下载到运动控制器上执行的程序称为“运动程序”。运动程序与应用程序的关系如图 10-1 所示。

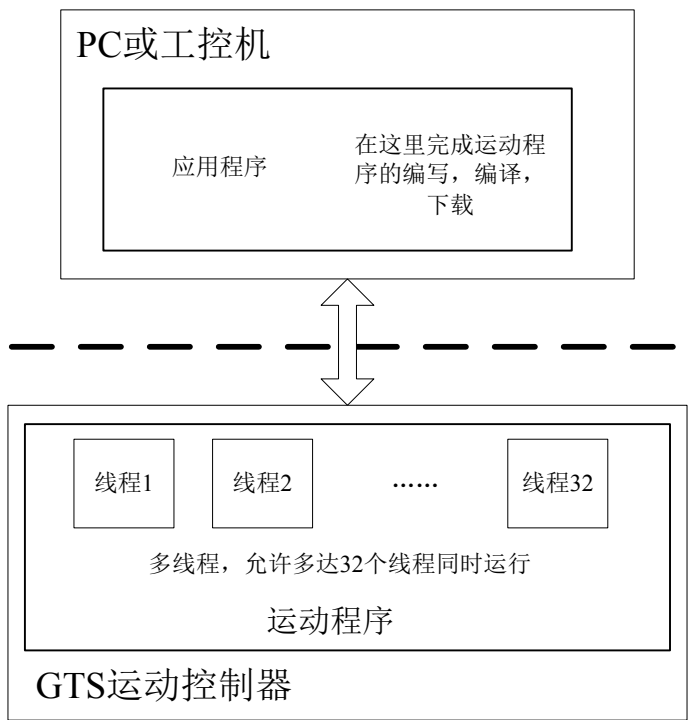


图 10-1 运动程序与应用程序的关系

运动程序的三个特点：独立性，实时性，并行性。

独立性：运动程序能够脱离主机在运动控制器上独立执行，主机能够将CPU资源分配给其它任务，从而将主机从繁琐的运动逻辑管理中解放出来。当然，如果需要，主机仍然可以在任何时候向控制器发送指令，即使运动控制器上的运动程序正在执行。



注意

当主机指令和运动控制器上的运动程序控制相同的轴时，需仔细设计运动逻辑，以免造成混乱。

实时性：运动控制器上执行运动程序由于不需要通过总线和主机进行频繁的数据交换，因此具有更高的实时性。和在主机上执行的应用程序不同之处在于，运动程序对GT指令调用不必再通过PC总线，因此具有更高的执行效率。平均执行速度约为100指令/毫秒，是PC机执行API指令速度的5倍。

并行性：支持多任务，允许多达32个运动程序在运动控制器上同时执行。



在多线程环境下，一个线程中连续的 2 条指令在执行时有可能被插入其它线程的指令。当启动多个线程并行执行时，应当仔细考虑线程之间是否会相互影响。

10.3 运动程序的使用

10.3.1 编写运动程序

运动程序可以使用C语言编写。但是一些编写规则和C语言略有不同。用户编写运动程序时应遵照“10.4.1 语言元素”、“10.4.2 运算指令”中的说明，否则有可能编译不通过。

运动程序可以和应用程序一样调用GT指令。用户可查阅“10.5 可在运动程序中使用的指令”知道哪些指令可以在运动程序中调用。



运动程序中，调用 GT 指令必须完整描述函数的每一个参数。

例如：short `GT_GetClock(unsigned long *pClock, unsigned long *pLoop=NULL)`。

在应用程序中调用，可以写成如下形式，使用 VC 可以编译通过：

```
long lClock;
```

```
GT_GetClock(&lClock);
```

在运动程序中调用，必须写成如下形式，否则编译不通过：

```
long lClock, lLoop;
```

```
GT_GetClock(&lClock, &lLoop);
```

10.3.2 编译

为了让运动控制器能够执行用户用C语言编写的运动程序，必须对运动程序进行编译。使用MCT2008编译运动程序，生成目标程序文件 (*.bin) 和符号文件 (*.ini)。目标文件用来下载到运动控制器。符号文件用来保存运动程序编译信息。应用程序必须使用这2个文件才能正确下载和启动运动程序，访问运动程序的变量。

关于如何使用MCT2008编译运动程序，请参考“MCT2008使用帮助”。具体操作：启动MCT2008，点击主界面菜单“帮助”-->“MCT2008使用帮助”，将会弹出MCT2008使用帮助对话框。在对话框左侧“目录”一页点击“工具”-->“运动程序编译器.mht”，可以查看详细的如何编译运动程序的说明。如图10-2所示。

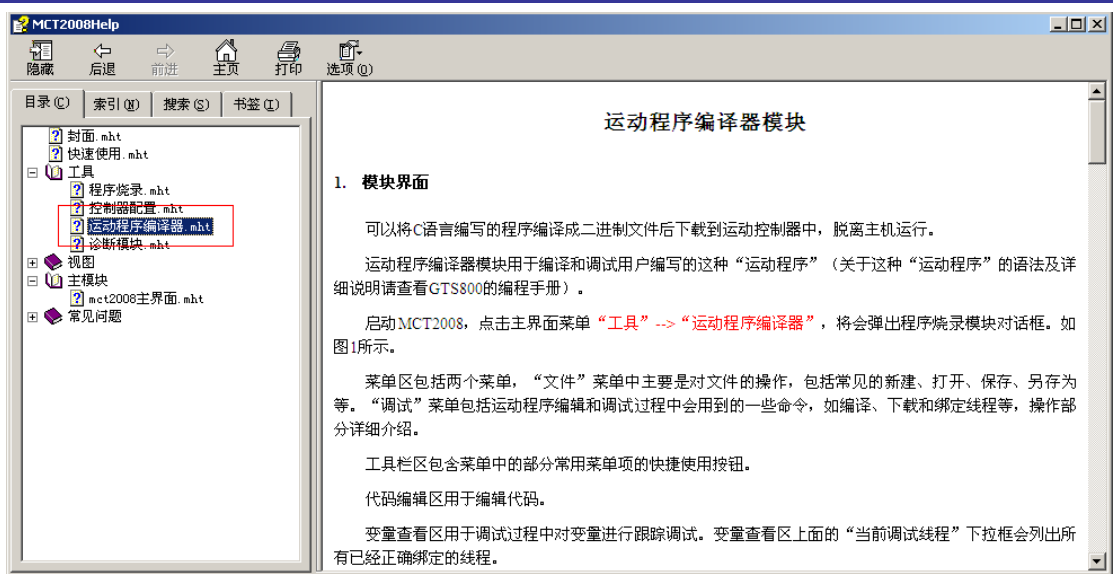


图 10-2 MCT2008 运动程序编译说明界面

编译成功后，目标程序文件 (*.bin) 和符号文件 (*.ini) 会出现在运动程序文件 (*.c) 的同一目录下。

注意

如果编译不成功，则无法进行下一步操作。用户应按照编译器的提示仔细检查错误原因。

10.3.3 指令列表

后面的操作需要用户在应用程序中调用相关的运动程序指令来完成。

表 10-1 运动程序指令列表

指令	说明	页码
GT_Download	下载运动程序到运动控制器	179
GT_GetFunId	读取运动程序中函数的标识	193
GT_GetVarId	读取运动程序中变量的标识	201
GT_Bind	绑定线程、函数、数据页	169
GT_RunThread	启动线程	212
GT_StopThread	停止正在运行的线程	229
GT_PauseThread	暂停正在运行的线程	210
GT_GetThreadSts	读取线程的状态	200
GT_SetVarValue	设置运动程序中变量的值	226
GT_GetVarValue	读取运动程序中变量的值	202

10.3.4 下载

编译成功后，用户需要在应用程序中调用 [GT_Download](#) 指令将目标文件 (*.bin) 下载到运动控制器的 SDRAM 中。用户应保证目标文件和符号文件与应用程序在同一目录下。

当下载新的运动程序时会覆盖原有的运动程序。运动控制器每次上电以后需要重新下载运动程

序。

10.3.5 绑定线程、函数和数据页

运动程序下载到运动控制器后，还不能立即执行。运动控制器会自动为运动程序中的每个函数，全局变量和局部变量定义好 ID，调用 `GT_GetFunId` 读取函数 ID，调用 `GT_GetVarId` 读取变量 ID。

为了执行运动程序，必须调用 `GT_Bind` 指令绑定线程、函数和数据页。

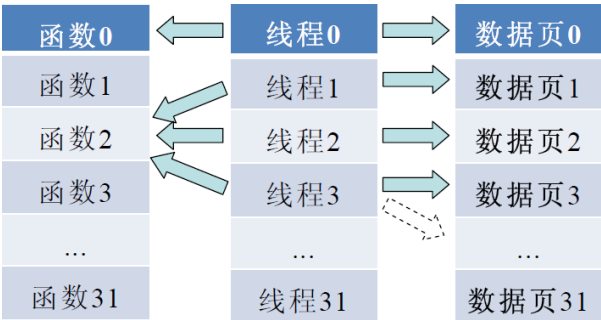


图 10-3 线程、函数和数据页的关系

运动控制器支持 32 个线程同时运行，一个线程只能分配一个函数，但是一个函数可以分配给多个线程同时执行，例如多轴回零时，可以让多个线程绑定同一个回零函数，然后同时启动这些线程就可以实现多轴同时回零。在线程执行过程中不允许绑定新的函数，除非线程执行完毕。

各函数的局部变量放在相互独立的数据页中。运动控制器提供 32 个数据页。在绑定线程和函数时，必须指明所使用的数据页。一个数据页只能分配给一个线程，但是一个线程可以使用多个数据页。线程在执行过程中可以切换数据页。其关系示意如图 10-3 所示。

10.3.6 启动，停止，暂停线程

将线程、函数和数据页绑定好后，调用 `GT_RunThread` 指令就可以启动某一个线程。调用 `GT_StopThread` 停止某个正在运行的线程，调用 `GT_PauseThread` 暂停线程。

10.3.7 查询线程状态

应用程序可以随时调用 `GT_GetThreadSts` 指令查询线程的执行状态。包括该线程是否正在运行，线程绑定的函数的返回值，当前正在执行的指令所在行数，当前正在执行的指令的返回值。

应用程序可以随时调用 `GT_SetVarValue` 指令更新运动程序中所有变量的值。

应用程序可以随时调用 `GT_GetVarValue` 指令查询运动程序中所有变量的值。

10.3.8 例程

1. 单线程累加求和

例程 10-1 运动程序单线程累加求和

运动程序完成累加求和任务。定义了全局变量 `sum` 用于保存累加和，局部变量 `begin` 用于保存累加起点，局部变量 `end` 用于保存累加终点。累加完成以后程序结束。

```

// -----
// 累加求和
// begin 累加起点
// end 累加终点
// -----

int sum;

int add(int begin, int end)
{
    int i;
    int cc;

    i = begin;
lbl_loop:
    cc = i>end;
    if(cc) goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
    return sum;
}

```

应用程序负责编译、下载、初始化、启动运动程序。

```

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

int main(int argc, char* argv[])
{
    short rtn;
    short funId;
    TVarInfo sum, begin, end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GT_Open();
    printf("GT_Open()=%d\n", rtn);
    // 复位运动控制器
    rtn = GT_Reset();
    printf("GT_Reset()=%d\n", rtn);
    // 下载运动程序sum.bin,
    // 必须保证sum.bin文件位于工程文件夹中
    rtn = GT_Download("sum.bin");
    printf("GT_Download()=%d\n", rtn);
}

```

```

// 获取函数ID
rtn = GT_GetFunId("add", &funId);
printf("GT_GetFunId()=%d\n", rtn);
// 获取全局变量sum的ID
rtn = GT_GetVarId(NULL, "sum", &sum);
printf("GT_GetVarId()=%d\n", rtn);
// 获取局部变量begin的ID
rtn = GT_GetVarId("add", "begin", &begin);
printf("GT_GetVarId()=%d\n", rtn);
// 获取局部变量end的ID
rtn = GT_GetVarId("add", "end", &end);
printf("GT_GetVarId()=%d\n", rtn);
// 绑定线程，函数，数据页
rtn = GT_Bind(0, funId, 0);
printf("GT_Bind()=%d\n", rtn);
value = 0;
// 初始化运动程序的全局变量sum
rtn = GT_SetVarValue(-1, &sum, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 1;
// 初始化运动程序的局部变量begin
rtn = GT_SetVarValue(0, &begin, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 100;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue(0, &end, &value);
printf("GT_SetVarValue()=%d\n", rtn);
// 启动线程
rtn = GT_RunThread(0);
printf("GT_RunThread()=%d\n", rtn);

do
{
    // 查询线程状态
    rtn = GT_GetThreadSts(0, &thread);
    // 查询全局变量sum的值
    rtn = GT_GetVarValue(-1, &sum, &value);
    printf("run=%d sum=%-10.0lf\n", thread.run, value);
} while( 1 == thread.run );    // 等待线程运行结束

getch();
return 0;
}

```

2. 多线程累加求和

例程 10-2 运动程序多线程累加求和

运动程序代码和例程 10-1 相同。

应用程序负责编译、下载、初始化、启动运动程序。和例程 10-1 不同之处在于启动 2 个线程完成累加运算任务。

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

int main(int argc, char* argv[])
{
    short rtn;
    short funId;
    TVarInfo sum, begin, end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GT_Open();
    printf("GT_Open()=%d\n", rtn);
    // 复位运动控制器
    rtn = GT_Reset();
    printf("GT_Reset()=%d\n", rtn);
    // 下载运动程序sum.bin
    // 必须保证sum.bin文件位于工程文件夹中
    rtn = GT_Download("sum.bin");
    printf("GT_Download()=%d\n", rtn);
    // 获取函数ID
    rtn = GT_GetFunId("add", &funId);
    printf("GT_GetFunId()=%d\n", rtn);
    // 获取全局变量sum的ID
    rtn = GT_GetVarId(NULL, "sum", &sum);
    printf("GT_GetVarId()=%d\n", rtn);
    // 获取局部变量begin的ID
    rtn = GT_GetVarId("add", "begin", &begin);
    printf("GT_GetVarId()=%d\n", rtn);
    // 获取局部变量end的ID
    rtn = GT_GetVarId("add", "end", &end);
    printf("GT_GetVarId()=%d\n", rtn);
    // 绑定线程，函数，数据页
    rtn = GT_Bind(0, funId, 0);
    printf("GT_Bind()=%d\n", rtn);
    // 绑定线程，函数，数据页
    rtn = GT_Bind(1, funId, 1);
    printf("GT_Bind()=%d\n", rtn);
    value = 0;
    // 初始化运动程序的全局变量sum
```

```

rtn = GT_SetVarValue(-1, &sum, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 1;
// 初始化运动程序的局部变量begin
rtn = GT_SetVarValue(0, &begin, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 50;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue(0, &end, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 51;
// 初始化运动程序的局部变量begin
rtn = GT_SetVarValue(1, &begin, &value);
printf("GT_SetVarValue()=%d\n", rtn);
value = 100;
// 初始化运动程序的局部变量end
rtn = GT_SetVarValue(1, &end, &value);
printf("GT_SetVarValue()=%d\n", rtn);
// 启动线程
rtn = GT_RunThread(0);
printf("GT_RunThread()=%d\n", rtn);
// 启动线程
rtn = GT_RunThread(1);
printf("GT_RunThread()=%d\n", rtn);

do
{
    // 查询线程状态
    rtn = GT_GetThreadSts(0, &thread);
}while( 1 == thread.run );    // 等待线程运行结束
do
{
    // 查询线程状态
    rtn = GT_GetThreadSts(1, &thread);
}while( 1 == thread.run );    // 等待线程运行结束

// 查询全局变量sum的值
rtn = GT_GetVarValue(-1, &sum, &value);
printf("sum=%-10.0lf", value);

getch();
return 0;
}

```

10.4 如何编写运动程序

10.4.1 语言元素

1. 数据类型

支持整型和浮点型 2 种数据类型。

- 整型 32 位，取值范围是-2, 147, 483, 648 ~ 2, 147, 483, 647。
- 浮点型采用定点格式，32 位整数，16 位小数。所能表示的最小精度为 $(1/2)^{16}=0.0000152587890625$ 。

2. 常量

可以在程序中直接使用立即数和宏。立即数可以是 10 进制整数、16 进制整数和浮点数。

3. 变量

可以声明局部变量和全局变量。每个函数最多可声明 1024 个局部变量。全局变量最多可声明 1024 个。整型类型说明符为 int。浮点型类型说明符为 double。

4. 数组

支持一维数组，支持常量下标索引和变量下标索引。

不支持多维数组，不支持用数组元素进行下标索引。

5. 函数

函数可以定义返回值类型和输入形参类型。

不支持在函数中调用自定义函数，但是可以调用 GT 运动控制指令。

6. 数据类型转换

支持强制数据类型转换。强制数据类型转换符有 int, double。

- 数据类型转换符必须加括号，如 `a=(int)b`
- 数据类型转换不会改变变量本身的数据类型定义

10.4.2 运算指令

支持算术运算、逻辑运算、关系运算、位运算，语法规则和 C 语言相同，但是不支持复杂表达式，只能使用 2 个操作数进行运算，而且这 2 个操作数的数据类型必须相同。

注意：由于运动程序中的浮点数据类型只有 16 位小数精度，请不要在运动程序中进行高精度浮点运算。

1. 算术运算

用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)共五种。

2. 逻辑运算

逻辑运算包括与(&&)、或(||)、非(!)三种。参数可以是整型变量、或者整型常数。

3. 关系运算

关系运算符用于比较运算。包括大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=)和不等(<!=)六种。参与比较的参数类型必须一致。

4. 位运算

参与运算的量，按二进制位进行运算。包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)六种。

5. 流程控制

支持条件跳转、条件返回，语法规则和 C 语言相同。

1) 条件跳转如下所示：

```
if(var) goto label;
```

当条件变量 var 非 0 时，跳转到标记为 label 的指令。

不支持表达式作为判断条件。

2) 条件返回如下所示：

```
if(var) return value;
```

当条件变量 var 非 0 时，程序返回，返回值为 value。

不支持表达式作为判断条件。

10.4.3 流程控制与标准 C 语言的流程控制对比

(1) While 语句

运动程序实现：

```
int i,sum,cc;
i = 1;
sum = 0;
lbl_loop:
    cc = i > 10;
    if(cc) goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
.....
```

标准 C 实现：

```
int i,sum;
i = 1;
sum = 0;
while(i <= 10)
{
    sum = sum + i;
    i = i + 1;
}
```

以上程序等同于：

运动程序实现:

```

int i,sum,cc;
i = 1;
sum = 0;
lbl_loop:
    sum = sum + i;
    i = i + 1;

    cc = i<= 10;
    if(cc) goto lbl_loop;
lbl_end:
.....

```

标准 C 实现:

```

int i,sum;
i = 1;
sum = 0;
while(i<= 10)
{
    sum = sum + i;
    i = i + 1;
}

```

(2) for 语句

运动程序实现:

```

int i,sum,cc;
i = 1;
sum = 0;
lbl_loop:
    cc = i> 10;
    if(cc) goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
.....

```

标准 C 实现:

```

int i,sum;
sum = 0;
for(i=1;i<= 10;i++)
{
    sum = sum + i;
}

```

(3) if... else 语句

运动程序实现:

```

int a,b;
int cc;
a = 5;
b = 10;
cc = a<=b;
    if(cc) goto lbl_Jump;
a = a - b;
goto lbl_end;
lbl_Jump:
    a = b - a;
lbl_end:
.....

```

标准 C 实现:

```

int a,b;
a = 5;
b = 10;
if(a>b)
{
    a = a - b;
}
else
{
    a = b - a;
}

```

(4) switch...case

运动程序实现： <pre>#define MC_GPO 12 //注意：该宏定义应放在函数体外 int a; int cc; a = 5; cc = a == 1; if(cc) goto lbl_1; cc = a == 2; if(cc) goto lbl_2; cc = a == 5; if(cc) goto lbl_3; goto lbl_end; lbl_1: GT_SetDo(MC_GPO,0x01); goto lbl_end; lbl_2: GT_SetDo(MC_GPO,0x02); goto lbl_end; lbl_3: GT_SetDo(MC_GPO,0x04); goto lbl_end; lbl_end:</pre>	标准 C 实现： <pre>int a; a = 5; switch(a) { case 1: GT_SetDo(MC_GPO,0x01); break; case 2: GT_SetDo(MC_GPO,0x02); break; case 5: GT_SetDo(MC_GPO,0x04); break; default: break; }</pre>
---	---



注意

编写运动程序时：

1. 对于 GT 指令，其指令的参数个数必须写全；
2. GT 指令中的参数宏定义，在运动程序当中不支持，需要重新定义。

10.5 可在运动程序中使用的指令

表 10-2 可在运动程序中使用的指令其它指令

编号	指令	指令	指令
1	GT_SetDo	GT_PrflTrap	GT_LmtsOn
4	GT_SetDoBit	GT_SetTrapPrm	GT_LmtsOff
7	GT_GetDo	GT_GetTrapPrm	GT_ProfileScale
10	GT_GetDi	GT_PrflJog	GT_EncScale
13	GT_GetDiReverseCount	GT_SetJogPrm	GT_StepDir
16	GT_SetDiReverseCount	GT_GetJogPrm	GT_StepPulse
19	GT_SetDac	GT_PrflGear	GT_StepOrthogonal
22	GT_GetDac	GT_SetGearMaster	GT_SetMtrBias
25	GT_GetAdcValue	GT_GetGearMaster	GT_GetMtrBias
28	GT_GetEncPos	GT_SetGearRatio	GT_SetMtrLmt

编号	指令	指令	指令
31	GT_SetCaptureMode	GT_GetGearRatio	GT_GetMtrLmt
34	GT_GetCaptureStatus	GT_GearStart	GT_EncSns
37	GT_GetClock	GT_ZeroPos	GT_EncOn
40	GT_GetSts	GT_SetPid	GT_EncOff
43	GT_AxisOn	GT_GetPid	GT_SetPosErr
46	GT_Stop	GT_Update	GT_GetPosErr
49	GT_SynchAxisPos	GT_SetPos	GT_SetStopDec
52	GT_GetSoftLimit	GT_GetPos	GT_GetStopDec
55	GT_GetAxisBand	GT_SetVel	GT_LmtSns
58	GT_GetPrfVel	GT_GetVel	GT_CtrlMode
61	GT_GetPrfMode	GT_ZeroPos	GT_CrdSpace
64	GT_GetAxisPrfVel	GT_CrdStatus	GT_CrdStart
67	GT_GetAxisEncPos	GT_AlarmOff	GT_SetOverride
70	GT_GetAxisEncAcc	GT_AlarmOn	GT_GetCrdVel
73	GT_SetControlFilter		
76	GT_GetControlFilter		

第11章 其他指令

11.1 本章简介

本章介绍 GTS 运动控制器为用户提供的其他指令。包括：打开/关闭运动控制器、读取固件版本号、读取系统时钟、打开/关闭电机使能信号、维护位置值、电机到位检测、设置 PID 参数、反向间隙补偿、自动回原点、位置比较输出。



本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。

11.2 打开/关闭运动控制器

表 11-1 打开/关闭运动控制器指令列表

指令	说明	页码
GT_Open	打开运动控制器	210
GT_Close	关闭运动控制器	173
GT_SetCardNo	切换当前运动控制器卡号	214
GT_GetCardNo	读取当前运动控制器卡号	186
GT_Reset	复位运动控制器	211

在使用运动控制器之前，首先需要使用 [GT_Open](#) 指令打开运动控制器，和运动控制器建立通讯；在使用运动控制器结束之后，退出应用程序时，应当调用 [GT_Close](#) 指令关闭运动控制器。用户不应当在程序中反复地调用 [GT_Open](#) 和 [GT_Close](#) 指令，去频繁地打开和关闭运动控制器。只要在应用程序初始化阶段调用一次 [GT_Open](#)，在应用程序退出时调用一次 [GT_Close](#) 即可。

调用 [GT_Reset](#) 指令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。

[GT_SetCardNo](#) 用于切换当前运动控制器卡号，一台计算机上使用多个运动控制器时，该指令用于指定当前运动控制器。当指令执行成功后，之后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)，用于区别不同控制器。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。

11.3 读取固件版本号

表 11-2 读取固件版本号指令列表

指令	说明	页码
GT_GetVersion	读取运动控制器固件的版本号	202

为了方便用户核对运动控制器固件版本，提供 [GT_GetVersion](#) 指令来读取运动控制器固件版本号，版本号是一个含有 21 个字符的字符串：aaa bbbbbb ccc dddddd eee。具体的定义如表 11-3 所示。

表 11-3 固件版本号的定义格式

aaa	固件 1 的版本号，如 100，即表示版本号为：1.00
bbbbbb	固件 1 的版本号的生成时间，如 090908，即表示该版本生成于：2009 年 9 月 8 日
ccc	固件 2 的版本号
dddddd	固件 2 的版本号的生成时间
eee	固件 3 的版本号

例程 11-1 读取运动控制器版本号

```

short rtn;
char *pVersion;                // 定义指向版本号字符串的指针
rtn = GT_Open();
rtn = GT_GetVersion(&pVersion); // 读取版本号
printf("%s\n", pVersion);
rtn = GT_Close()

```

11.4 读取系统时钟

表 11-4 读取系统时钟指令列表

指令	说明	页码
GT_GetClock	读取运动控制器系统时钟	187
GT_GetClockHighPrecision	读取运动控制器系统高精度时钟	187

运动控制器上电初始化之后，内部计数时钟从 0 开始计数，每 1 毫秒增加 1，通过 [GT_GetClock](#) 指令可以读取该计数时钟的值。[GT_GetClockHighPrecision](#) 读取的时钟每 125 微秒增加 1，调用 [GT_Reset](#) 指令将会使计数时钟值清零。

11.5 打开/关闭电机使能信号

表 11-5 打开/关闭电机使能信号指令列表

指令	说明	页码
GT_AxisOn	打开驱动器使能	168
GT_AxisOff	关闭驱动器使能	168

调用 [GT_AxisOn](#) 指令将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。如果在系统配置时，没有数字量输出与该 axis 关联，则该指令将会无效(4.3.8 配置 do)。如果运动控制器被配置成了伺服控制方式(第 4 章 系统配置)，那么必须首先设置指定轴的位置环的 PID 参数。

11.6 维护位置值

表 11-6 维护位置值指令列表

指令	说明	页码
GT_SetPrfPos	修改指定轴的规划位置	224
GT_SynchAxisPos	axis 合成规划位置和所关联的 profile 同步 axis 合成编码器位置和所关联的 encoder 同步	229
GT_ZeroPos	清零规划位置 and 实际位置，并进行零漂补偿	230

第三章系统配置里提到,axis 含有 profile 和 encoder 的当量变换的功能,如果调用了 GT_SetPrfPos 指令或者 GT_SetEncPos 指令之后,profile 的输出值或者 encoder 的输出发生了变化,如果需要将 axis 当量变换之后的值与 profile 或者 encoder 的值同步,需要调用 GT_SynchAxisPos 指令。

11.7 电机到位检测

表 11-7 电机到位检测指令列表

指令	说明	页码
GT_SetAxisBand	设置轴到位误差带 规划器静止,规划位置和实际位置的误差小于设定误差带,并且在误差带内保持设定时间后,轴状态的电机到位标志位置起到位标志	212
GT_GetAxisBand	读取轴到位误差带	182

用户使用伺服电机时,由于伺服电机在运动的过程中可能会存在运动滞后,会出现规划停止,而实际位置并没有到位的情况。用户可以使用运动控制器的运动到位检测功能来判断电机是否实际到位。运动控制器默认该功能是无效的,当调用 GT_SetAxisBand 指令设置了相应的误差带和保持时间之后,该功能生效。

该功能生效后,当规划器处于静止状态,即轴状态寄存器 bit10 为 0(参见 5.3.1),并且规划位置和编码器位置的误差在设定的误差带内保持了设定时间,轴状态寄存器 bit11 将被置 1(参见 5.3.1)。当规划器运动,或者规划位置和编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度,应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小,或者误差带保持时间太长,都会使到位时间增长,影响加工效率。

如图 11-1 所示。当轴 1 启动电机到位检测功能,设置误差带为 100pulse,误差带保持时间为 500μs,当电机在 1000pulse 位置处静止时,会有震荡。控制器判断其震荡位于误差带内 500μs 后,就将轴状态的 bit11 电机到位标志置 1。蓝色阴影区域表示电机到位标志还未置 1,橙色阴影区域表示已置 1。可以看出,误差带保持时间越短,误差带越大,控制器会在越短时间内将电机到位标志置 1,但是并不是越短越好。电机是否连接机械本体,也会影响控制器判断电机到位所需的时间。

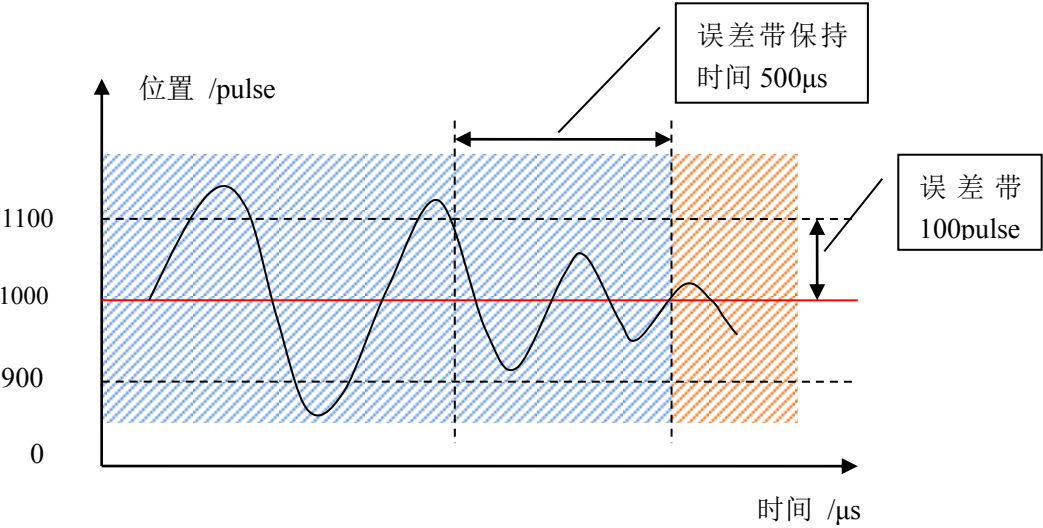


图 11-1 电机到位检测功能



使用电机到位检测功能必须注意以下几点：

- axis 正确关联编码器，并且编码器方向和规划运动方向必须一致。
- 正确设置到位误差带，默认情况下到位误差带无效
- 调用 `GT_ZeroPos` 进行对位置进行清零，同时进行自动零漂补偿。

例程 11-2 电机到位检测功能

下面这个例程示例电机到位检测的使用方法。一个轴运动到位以后，启动另一个轴的运动。电机到位的运动状态检测如图 11-2 所示。

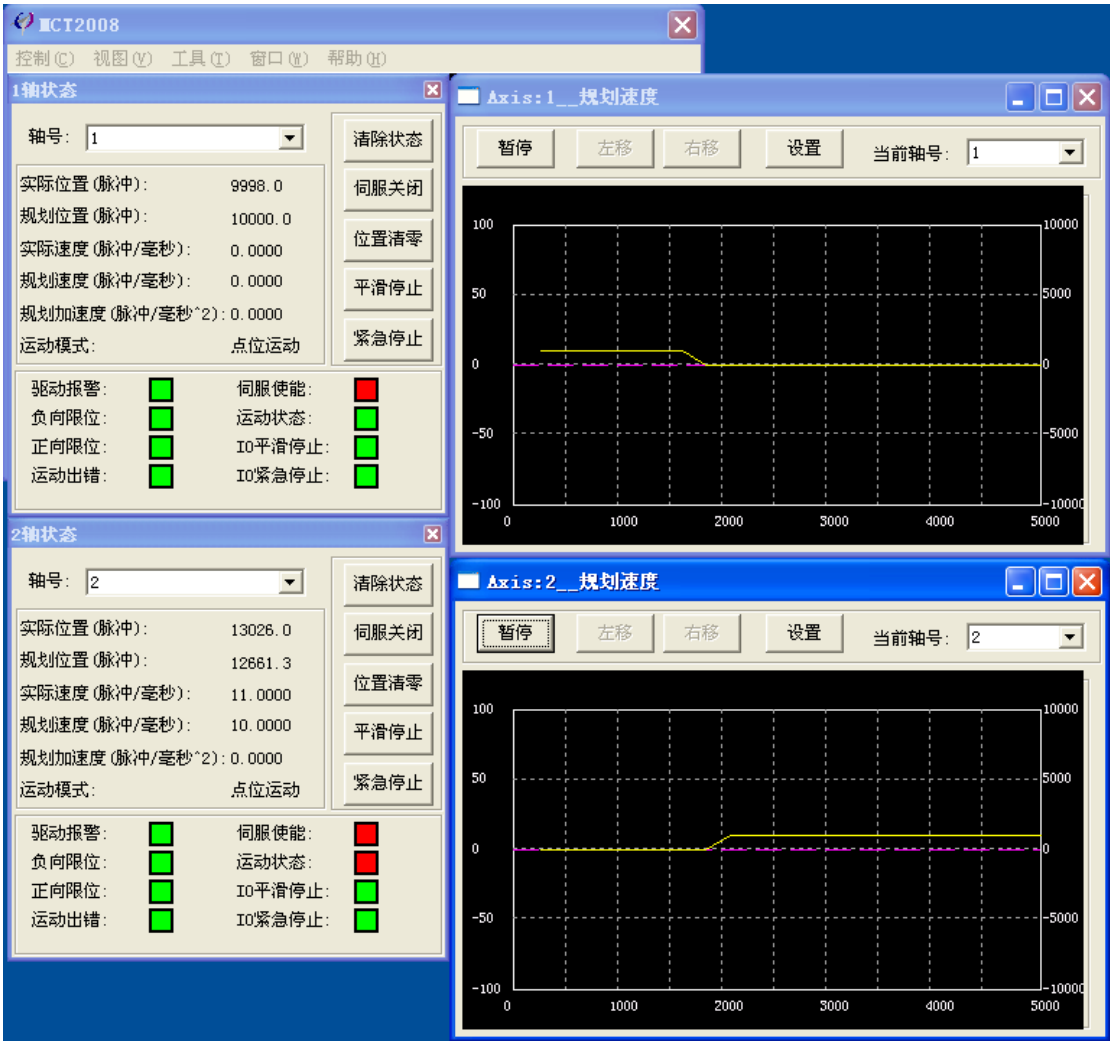


图 11-2 电机到位的运动状态检测

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

#define AXIS_X 1
#define AXIS_Y 2

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

```
TPid pid;
TTrapPrm trap;
long sts;
long posX, posY;
double prfPos, prfVel;

// 打开运动控制器
sRtn = GT_Open();
commandhandler("GT_Open", sRtn);
// 复位控制器
sRtn = GT_Reset();
commandhandler("GT_Reset", sRtn);
// 配置运动控制器为伺服模式
sRtn = GT_LoadConfig("servo.cfg");
commandhandler("GT_LoadConfig", sRtn);
// 延时一段时间
Sleep(100);
// 清除各轴的报警和限位
sRtn = GT_ClrSts(1, 8);
commandhandler("GT_ClrSts", sRtn);
// 读取X轴PID参数
sRtn = GT_GetPid(AXIS_X, 1, &pid);
commandhandler("GT_GetPid", sRtn);
pid.kp = 10;
// 更新X轴PID参数
sRtn = GT_SetPid(AXIS_X, 1, &pid);
commandhandler("GT_SetPid", sRtn);
// 读取Y轴PID参数
sRtn = GT_GetPid(AXIS_Y, 1, &pid);
commandhandler("GT_GetPid", sRtn);
pid.kp = 10;
// 更新Y轴PID参数
sRtn = GT_SetPid(AXIS_Y, 1, &pid);
commandhandler("GT_SetPid", sRtn);
// X轴伺服使能
sRtn = GT_AxisOn(AXIS_X);
commandhandler("GT_AxisOn", sRtn);
// Y轴伺服使能
sRtn = GT_AxisOn(AXIS_Y);
commandhandler("GT_AxisOn", sRtn);
// 延时一段时间，等待伺服稳定
Sleep(200);
// 位置清零，并进行自动零漂补偿
sRtn = GT_ZeroPos(AXIS_X);
commandhandler("GT_ZeroPos", sRtn);
// 设置X轴到位误差带
```

```

sRtn = GT_SetAxisBand(AXIS_X, 20, 5);
commandhandler("GT_SetAxisBand", sRtn);
// 位置清零, 并进行自动零漂补偿
sRtn = GT_ZeroPos(AXIS_Y);
commandhandler("GT_ZeroPos", sRtn);
// 设置Y轴到位误差带
sRtn = GT_SetAxisBand(AXIS_Y, 20, 5);
commandhandler("GT_SetAxisBand", sRtn);
// X轴设为点位模式
sRtn = GT_PrftTrap(AXIS_X);
commandhandler("GT_PrftTrap", sRtn);
// 读取X轴点位运动参数
sRtn = GT_GetTrapPrm(AXIS_X, &trap);
commandhandler("GT_GetTrapPrm", sRtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置X轴点位运动参数
sRtn = GT_SetTrapPrm(AXIS_X, &trap);
commandhandler("GT_SetTrapPrm", sRtn);
// 设置X轴的目标速度
sRtn = GT_SetVel(AXIS_X, 10);
commandhandler("GT_SetVel", sRtn);
// Y轴设为点位模式
sRtn = GT_PrftTrap(AXIS_Y);
commandhandler("GT_PrftTrap", sRtn);
// 读取Y轴点位运动参数
sRtn = GT_GetTrapPrm(AXIS_Y, &trap);
commandhandler("GT_GetTrapPrm", sRtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置Y轴点位运动参数
sRtn = GT_SetTrapPrm(AXIS_Y, &trap);
commandhandler("GT_SetTrapPrm", sRtn);
// 设置Y轴的目标速度
sRtn = GT_SetVel(AXIS_Y, 10);
commandhandler("GT_SetVel", sRtn);

posX = 10000;
posY = 20000;
while(!kbhit())
{
    // 设置X轴目标位置
    sRtn = GT_SetPos(AXIS_X, posX);
    commandhandler("GT_SetPos", sRtn);
    // 启动X轴的运动
    sRtn = GT_Update(1<<(AXIS_X-1));

```

```

commandhandler("GT_Update", sRtn);
posX = - posX;
// 等待X轴进入误差带
do
{
GT_GetSts(Axis_X, &sts);
GT_GetPrfPos(Axis_X, &prfPos);
GT_GetPrfVel(Axis_X, &prfVel);
printf("x pos=%-10.2lf vel=%-6.2lfr", prfPos, prfVel);
}while( 0x800 != ( sts& 0x800 ) );

printf("\n");
// 设置Y轴目标位置
sRtn = GT_SetPos(Axis_Y, posY);
commandhandler("GT_SetPos", sRtn);
// 启动Y轴的运动
sRtn = GT_Update(1<<(Axis_Y-1));
commandhandler("GT_Update", sRtn);
posY = - posY;
// 等待Y轴进入误差带
do
{
GT_GetSts(Axis_Y, &sts);
GT_GetPrfPos(Axis_Y, &prfPos);
GT_GetPrfVel(Axis_Y, &prfVel);
printf("y pos=%-10.2lf vel=%-6.2lfr", prfPos, prfVel);
}while( 0x800 != ( sts& 0x800 ) );
printf("\n");
}
// 伺服关闭
sRtn = GT_AxisOff (Axis_X);
printf("\nGT_AxisOff()=%d, Axis:%d\n", sRtn, Axis_X);
sRtn = GT_AxisOff (Axis_Y);
printf("\nGT_AxisOff()=%d, Axis:%d\n", sRtn, Axis_Y);
return 0;
}

```

11.8 设置 PID 参数

表 11-8 设置 PID 参数指令列表

指令	说明	页码
GT_SetControlFilter	设定 PID 索引, 支持 3 组 PID 参数	214
GT_GetControlFilter	读取当前 PID 索引	187
GT_SetPid	设置 PID 参数	222
GT_GetPid	读取 PID 参数	197

PID 参数说明如下：

- (1) **kp**: 比例增益；该系数的作用是改变控制系统的动态响应速度。
- (2) **ki**: 积分增益；该系数的作用是消除控制系统的稳态误差。
- (3) **kd**: 微分增益；该系数的作用是改善控制系统的动态性能。其作用与输出的偏差变化速度成比例，能够预测偏差的变化，产生超前控制作用，以阻止偏差的变化。一般不需要设置。
- (4) **kvff**: 速度前馈系数；该参数的作用是减小跟随误差。跟随误差即某一时刻的规划值与编码反馈值的差值。
- (5) **kaff**: 加速度前馈系数；一般不需要调节。
- (6) **integralLimit**: 积分饱和极限；该参数一般默认设置。
- (7) **derivativeLimit**: 微分饱和极限；该参数一般默认设置。
- (8) **limit**: 控制量输出饱和极限；该参数与驱动器的电压接收范围有关，默认是 32767，即对应范围是[-10V, 10V]。若驱动器的接收电压范围是[-5V, 5V]，则该参数为 16384，如果第一次调 PID 参数可以把此值设小，以防止发生正反馈，保护设备。当确定反馈正常时，需把此值设成对应的电压值，如很多驱动器默认的接收的范围是[-10V, 10V]。

调节 PID 的相关说明：

- (1) 首先调节 **kp**，当电机很容易被移动时，说明 **kp** 太小，应慢慢加大 **kp**，但不能将电机调节的“过硬”，**kp** 太大了可能会引起高频振荡，这时就需要减小该参数。一般可以从 **kp = 1** 慢慢增大调试，利用 MCT2008 的 TUNING 功能，查看响应曲线，一般如果控制旋转电机，走点位运动，只调此参数即可。
- (2) 在响应速度满足的情况下，若跟随误差较大，则需要调节速度前馈系数，以减小跟随误差。此参数在连续轨迹运动的场合使用较多；但在点位运动，如果用直线电机，调整此参数，对电机快速响应也有一定的效果。
- (3) 如果按照第一、第二步调试，效果还不能满足客户的需求，请调整驱动器的相关参数，如果驱动器的相关参数已调整好，请重复第一到第三步的操作，直到效果能满足用户的要求。
- (4) 若需要消除稳态误差，调节一般先将 DA 零漂清除，建议先不要调节 **ki** 值。**清除零漂方法：**
 - 1) **GT_Open** 之后，调用 **GT_Reset**，然后将相应轴设置成闭环控制模式（注意：设置 PID 参数时，只需要设置 **kp=1**，其他 PID 参数保持默认），之后调用 **GT_AxisOn**。
 - 2) 待电机稳定之后，调用 **GT_GetEncPos** 获取对应轴的位置编码值，接着调用 **GT_SetMtrBias** 来设置 DA 零漂值，该零漂值为编码值的负值。设置完之后，延时一段时间，然后再调用 **GT_GetEncPos**，若此时的值还不为 0，则再次用此次的编码值与上次的编码值相加，将其和的负值再次设置 DA 零漂。

运动控制器支持设置 3 组 PID 参数，并且支持运动时在各组 PID 参数之间进行切换，通过调用 **GT_SetControlFilter** 指令来切换 PID 参数。

11.9 反向间隙补偿

表 11-9 反向间隙补偿指令列表

指令	说明	页码
GT_SetBacklash	设置反向间隙补偿的相关参数	212
GT_GetBacklash	读取反向间隙补偿的相关参数	185

反向间隙误差是指由于传动链中机械间隙的存在，执行部件在运动过程中，从正向运动变为负向运动时，或者从负向运动变为正向运动时，执行部件的运动量与理论量存在误差，最后将反映为叠加至工件上的加工精度的误差。为了消除反向间隙误差，提高机器的加工精度和定位精度，该控制器提供了反向间隙误差补偿功能。用户只要在初始化的时候调用相应的反向间隙误差补偿功能指令 GT_SetBacklash 设置了相应的参数，反向间隙误差补偿功能将会生效；也可以通过指令 GT_SetBacklash 来关闭反向间隙误差补偿功能。

用户可以设置反向间隙误差补偿量的叠加速度，可以瞬间(一个控制周期内)叠加到输出量上，也可以选择以一定的速度叠加到输出量上。通过设置指令 GT_SetBacklash 的 compChangeValue 参数来实现，当 compChangeValue 的值为 0 或者大于等于 compValue 的值时，则表示误差补偿量将瞬间叠加到输出量上，当为其他值时，表示误差补偿量的叠加速度，单位是：pulse/ms。

反向间隙误差补偿方向指的是，反向间隙误差补偿是沿正方向补偿还是沿负方向补偿。如果指令 GT_SetBacklash 的参数 compDir 参数设置为 0 时，则只有电机从正方向转为负方向运动时，反向间隙补偿量生效，当电机向正方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从正方向转为负方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从负方向转为正方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向正方向运动一定的距离，以保证正方向运动没有间隙存在。

当指令 GT_SetBacklash 的参数 compDir 参数设置为 1 时，则只有电机从负方向转为正方向运动时，反向间隙补偿量生效，当电机向负方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从负方向转为正方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从正方向转为负方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向负方向运动一定的距离，以保证负方向运动没有间隙存在。

反向间隙补偿量会直接叠加到运动控制器的输出量上，当用户读取规划位置时，不会读到反向间隙补偿量。但是用户如果读取电机编码器的值，将会读到反向间隙的补偿量。

11.10 螺距误差补偿

表 11-10 螺距误差补偿指令

指令	说明	页码
GT_SetLeadScrewComp	设置螺距误差补偿的相关参数	220
GT_EnableLeadScrewComp	使能螺距误差补偿功能	179

如果机械传动系统使用了滚珠丝杠结构，由于加工制造存在偏差，丝杠每一处的螺距不可能做到完全一致，为了补偿某个运动区域的螺距误差，控制器提供了螺距误差补偿功能。用户在使用螺距误差功能之前，需要事先测量出补偿区域的补偿量。测量方法：确定一段补偿区域及每次测量的步长，先让电机轴往正方向运动，每走完一个步长得到一个理论值与实际值偏差作为补偿值；然后再反向运动，相应地得到反向的各个步长对应的补偿值。调用 GT_SetLeadScrewComp 设置补偿区域

和正反方向的补充值，最后调用 `GT_EnableLeadScrewComp` 使用电机轴的螺距误差补偿功能，这样控制器在电机运动过程中就会按照设置的参数在对应位置执行相应的补偿。

一般在回零之后进行测量补偿量。注意每次回零时，均从同一方向往原点运动。如最后均从负限位往原点运动，而不是此次从从负限位往原点运动，下次又从正限位往原点运动去找原点信号。

11.11 位置比较输出

11.11.1 指令列表

表 11-11 位置比较输出指令列表

指令	说明	页码
<code>GT_ComparePulse</code>	在 HSIO 口输出脉冲或者电平	176
<code>GT_CompareData</code>	设置位置比较参数并启动位置比较输出	174
<code>GT_CompareStatus</code>	查询位置比较状态	176
<code>GT_CompareStop</code>	取消位置比较输出	177
<code>GT_CompareLinear</code>	设置等间距重复位置比较输出	175
<code>GT_CompareContinuePulseMode</code>	设置一维位置比较输出是否为连续脉冲模式	174

11.11.2 重点说明

运动控制器内有位置比较单元，可用于将编码器位置或内部脉冲计数器位置与设定位置比较，当编码器位置或内部脉冲计数器位置到达设定位置时，在高速 IO（HSIO）口输出脉冲或反转电平。

控制器端子板共有 2 个位置比较输出通道，CN14 的 1、6 脚差分输出 HSIO0，2、7 脚差分输出 HSIO1。5 脚是 5V，9 脚是地。位置比较脉冲输出电压为 5V。

输入位置比较数据有两种方式：间距方式（`GT_CompareLinear` 指令）和数组方式（`GT_CompareData` 指令）。若比较位置为大量连续的等间距输出，可使用间距方式输入位置比较数据，此时仅需输入起始比较位置、间隔距离以及重复比较次数。若比较位置为非等间距位置值，可使用数组方式进行输入，以当前位置为起始点，输入比较位置相对起始位置的距离数组。此时需保证位置数组中的数据为单调增加的正向距离值或单调减少的负向距离值。

一旦重新调用 `GT_CompareData` 或 `GT_CompareLinear`，控制器将停止上次位置比较输出、清零位置比较脉冲输出个数，重新设置位置比较参数并启动新的位置比较。

使用 `GT_CompareData` 指令可设置位置比较输出的输出方式。启动位置比较以后，当外部编码器或者内部脉冲计数器到达设定的比较位置时，若选择定宽脉冲方式输出，立即从端子板 HSIO 的对应管脚输出一个脉冲，脉冲宽度可以设置为 1~32767us；若选择电平方式输出，立即从对应 HSIO 输出反转电平。当所有比较位置都触发以后，位置比较结束。按电平方式输出时，可以设置比较输出的起始电平，但脉冲方式不能设置输出的初始电平。

位置比较过程中，可调用 `GT_CompareStatus` 查询位置比较输出是否已经完成，以及已输出的上升沿个数。需要注意的是，该计数器按已输出的上升沿计数。若使用 `GT_CompareData` 以电平方式输出，则返回的 `pCount[0]` 及 `pCount[1]` 为已输出的上升沿个数，而非已完成的位置比较个数；若以脉冲方式输出，则返回的 `pCount[0]` 及 `pCount[1]` 为已输出的上升沿个数，与已输出的脉冲个数，也

即已完成的位置比较个数，相等。

若位置比较输出未启动或已结束，可调用 `GT_ComparePulse` 输出脉冲或指定电平。如果位置比较没有结束，该指令无效。

在某些工艺下，需要在每一次位置比较都输出若干个宽度和间距可控的脉冲，这时可以通过调用 `GT_CompareContinuePulseMode` 指令把一维位置比较输出的脉冲形式切换为连续脉冲模式。具体使用方法查看该指令说明。

一维位置比较的 2 个通道不能独立挂接不同的比较源。通过 `GT_CompareData` 可以设置两个 Buffer 的比较数据，但是只能设置一个比较源。通过 `GT_CompareLinear` 可以指定通道，但是最后一次调用该指令的参数生效，即无法同时把不同通道挂接到不同的比较源（即使两个通道挂接了同一个比较源，那么也是最后一条指令生效）。



一维位置比较和二维位置比较都是以启动位置比较后当前位置点为基准点，进行比较输出的。

11.11.3 例程

例程 11-3 位置比较输出指令详细的用法

(1) `GT_ComparePulse` 输出指定电平或脉冲：

```
// 在 HSIO 通道 0 输出一个宽度为 500 微秒的脉冲。
GT_ComparePulse(1,           // HSIO 通道 0
                 0,           // 输出脉冲
                 500);        // 脉冲宽度为 500us
// HSIO 通道 1 和通道 2 的引脚输出电平取反。
GT_ComparePulse(3,           // HSIO 通道 0 和通道 1 引脚输出电平取反
                 1,           // 输出电平
                 0);          // 无效参数
```

(2) `GT_CompareData` 设置位置比较输出数据

```
long pBuf1[3] = {1000, 2000, 3500}; // HSIO0 的比较位置缓冲区数据
long pBuf2[3] = {1500, 2000, 3000}; // HSIO1 的比较位置缓冲区数据

// 1 轴编码器位置为 1000, 3500 时 HSIO0 输出脉冲，为 1500, 3000 时 HSIO1 输出脉冲，
// 为 2000 时 HSIO0, HSIO1 同时输出脉冲。
GT_CompareData(1,           // 对 1 轴进行位置比较输出
               1,           // 需要进行比较的数据源为外部编码器
               0,           // 到达比较位置后输出脉冲
               0,           // 初始状态
               500,         // 脉冲宽度为 500us
```

```
pBuf1, // HSIO0 的比较位置缓冲区数据起始地址
3, // HSIO0 的比较位置缓冲区数据长度
pBuf2, // HSIO1 的比较位置缓冲区数据起始地址
3); // HSIO1 的比较位置缓冲区数据长度
// 初始电平为复位状态，1 轴编码器位置为 1000，2000，3500 位置反转 HSIO0 的电平。
GT_CompareData(1, // 对 1 轴进行位置比较输出
1, // 需要进行比较的数据源为外部编码器
1, // 到达比较位置后输出反转电平
0, // 初始电平为复位状态
0, // 该参数无效
pBuf1, // HSIO0 的比较位置缓冲区数据起始地址
3, // HSIO0 的比较位置缓冲区数据长度
NULL,
0);
```

下图 11-3 为 1 轴编码器在比较位置缓冲区数据为：pBuf1[3] = {1000, 2000, 3500}时，输出方式为脉冲时和输出方式为电平时的比较图。

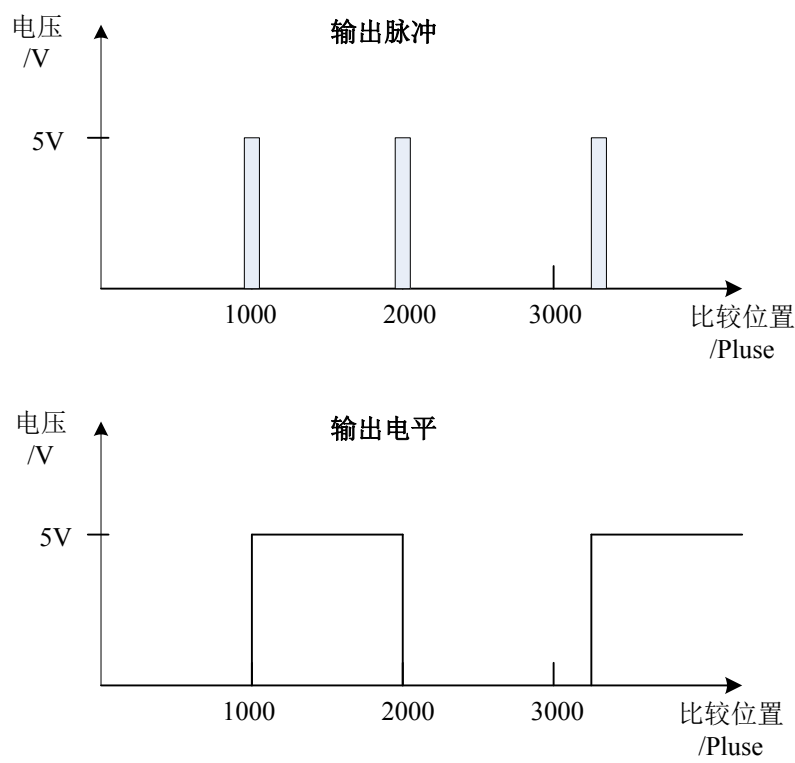


图 11-3 输出脉冲和输出电平比较

(3) GT_CompareLinear 设置位置比较数据

```
// 1 轴编码器位置为 1000 开始位置比较，每隔 100 个脉冲在 HSIO1 输出一个宽度为 10us 的脉冲，重复 400000 次。
rtn = GT_CompareLinear(1, // 对 1 轴进行位置比较输出
1, // 输出通道为 HSIO0
1000, // 当外部编码器位置到达 1000pulse 时，
// 开始进行位置比较输出，此位置为相对位置
```

```
400000, // 进行 400000 次的位置比较输出
100, // 每隔 100pulse, 在 HSIO0 口输出 1 个脉冲
10, // 在 HSIO0 口输出的脉冲宽度为 10us
1); // 需要进行比较的数据源为外部编码器
```

第12章 Smart Home（智能回原点）

12.1 本章简介

为了方便用户使用 GTS 控制器回原点功能，增加了 Smart Home（智能回原点）运动模式，在该模式下，用户只需根据回原点的方式配置相应的参数，调用一条指令就可以实现自动回原点。

12.2 指令列表

表 12-1 Smart Home 指令列表

指令	说明	页码
GT_GoHome	启动 Smart Home 实现各种方式回原点	202
GT_GetHomePrm	读取设置到控制器的 Smart Home 回原点参数	204
GT_GetHomeStatus	获取 Smart Home 回原点的状态	194

12.3 重点说明

名称解释：

Home：原点

Index：编码器转动一圈产生的 Z 相信号（也称为 C 相信号）

Smart Home 是对 GTS 系列运动控制器的“Home/Index 回原点”和“自动回原点”的优化和扩展。Smart Home 仍然采用高速硬件捕获机制实现回原点，把原来较为繁琐的回零过程固化到控制器，只需要调用简单指令就能够实现回原点，简化了用户程序。此外，Smart Home 汲取了工控界较为常见的回原点方式，集成到控制器给予实现，具体包括以下回原点方式：

- （1）限位回原点：**调用回原点指令，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动，脱离限位后再以较低的速度往限位方向运动，触发限位后停止运动，此处即为原点。（这种回原点方式没有用到高速硬件捕获功能，适用于对回原点精度要求不高或者不易于安装 Home 开关的场合）。

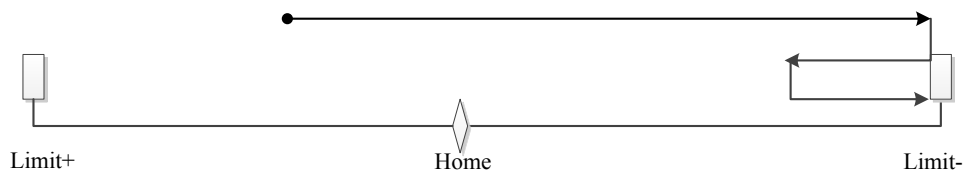


图 12-1 限位回原点

- （2）限位+Home 回原点：**调用回原点指令，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动并启动高速硬件捕获，在设定的搜索范围内搜索 Home，当触发 Home 开关后，电机将以较低的速度运动到捕获的位置处（即 Home 开关）。

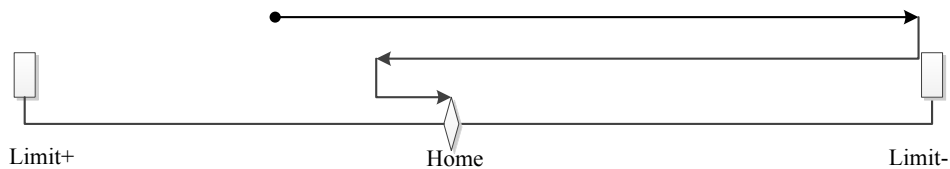


图 12-2 限位+Home 回原点

- (3) **限位+Index 回原点**：调用回原点指令，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则以较低的速度反方向运动并启动高速硬件捕获，在设定的搜索范围内寻找 Index，捕获到编码器的 Index 信号后，电机运动到捕获的位置处（编码器 Index 位置）。

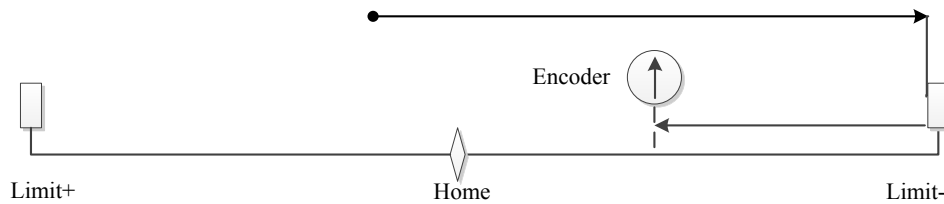


图 12-3 限位+Index 回原点

- (4) **限位+Home+Index 回原点**：调用回原点指令，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动并启动高速硬件捕获，在设定是范围内搜索 Home，当触发 Home 开关后，如果设定的搜索 Index 方向与搜索 Home 方向相同，则电机直接以较低的速度在设定的搜索范围内寻找 Index；反之，电机以较低的速度运动到捕获的 Home 位置处，之后再启动运动，在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

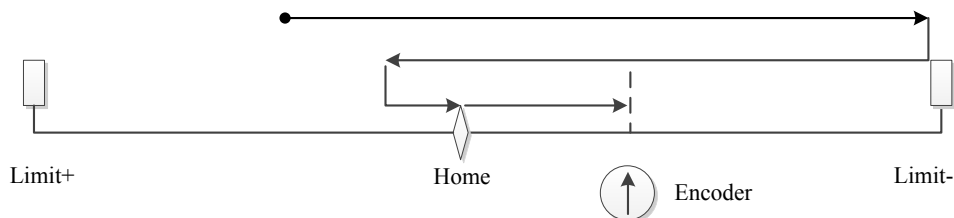


图 12-4 限位+Home+Index 回原点

- (5) **Home 回原点**：调用回原点指令，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的搜索范围内寻找 Home，当触发 Home 开关后，电机以较低的速度运动到捕获的位置处。

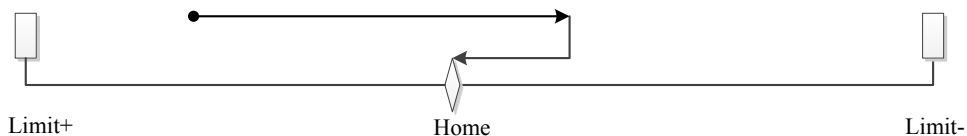


图 12-5 Home 回原点

- (6) **Home+Index 回原点**：调用回原点指令，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的范围内搜索 Home，当触发 Home 开关后，如果设定的搜索 Index 方向与搜索 Home 方向相同，则电机直接以较低的速度在设定的搜索范围内寻找 Index；反之，电机以较低的速度运动到捕获的 Home 位置处，之后再启动运动，在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

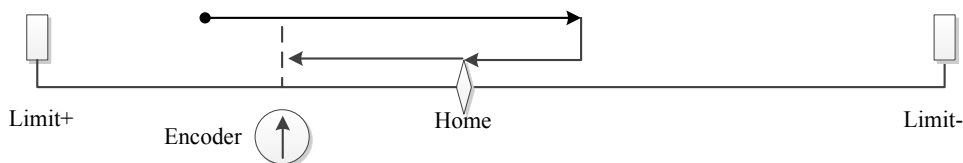


图 12-6 Home+Index 回原点

- (7) **Index 回原点**：调用回原点指令，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的范围内搜索 Index，捕获到编码器的 Index 信号后，电机再以较低的速度运动到 Index 处的位置停止。

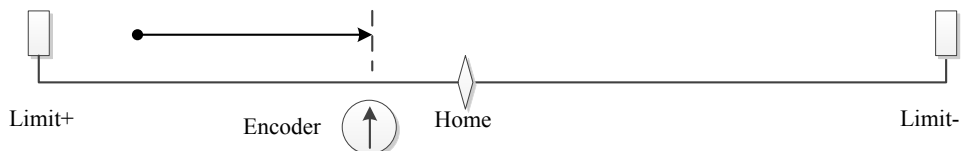


图 12-7 Index 回原点

- (8) **强制 Home 回原点**：当运动方向朝向 Home，则按照（5）实现；当运动方向背向 Home 时，则电机以较高的速度运动，在搜索范围内若碰到限位或运动到指定的搜索距离则反方向运动，继续寻找 Home。

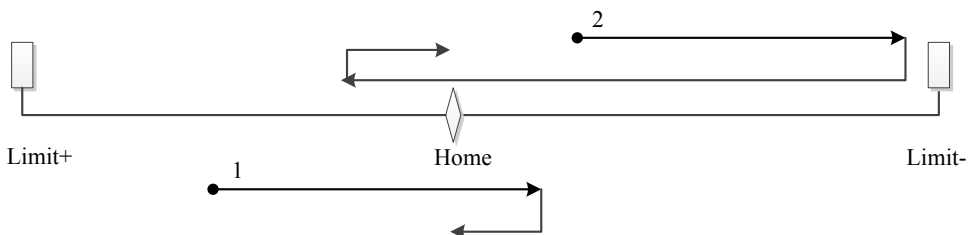


图 12-8 强制 Home 回原点

- (9) **强制 Home+Index 回原点**：当运动方向朝向 Home，则按照（6）实现；当运动方向背向 Home 时，则电机以较高的速度运动，在搜索范围内若碰到限位或运动到指定的搜索距离则反方向运动，继续寻找 Home。当触发 Home 开关后，如果设定的搜索 Index 方向与搜索 Home 方向相同，则电机直接以较低的速度在设定的搜索范围内寻找 Index；反之，电机以较低的速度运动到捕获的 Home 位置处，之后再次启动运动，在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

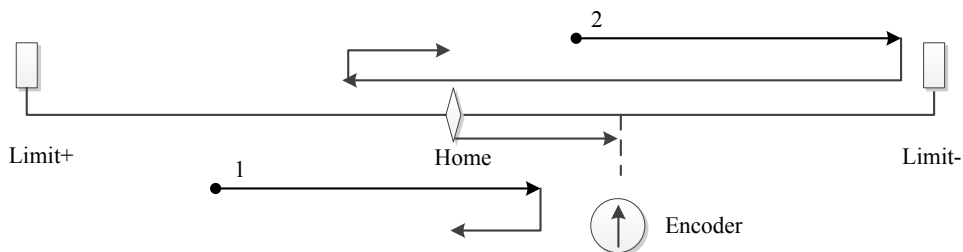


图 12-9 强制 Home+Index 回原点

实际当中，电机在回零开始时所在的位置有可能正处于 Home 开关或限位开关处，这种情况下，不同的回零模式有不同的处理办法：

对于（1）限位回原点，（2）限位+Home 回原点，（3）限位+Index 回原点，（4）限位+Home+Index 回原点，如果电机所处位置触发的限位开关与运动方向一致（往正方向运动且处于正限位或者往负方向运动处于负限位），则控制器会让电机回退一定距离，脱离限位后再开始执行回零动作：

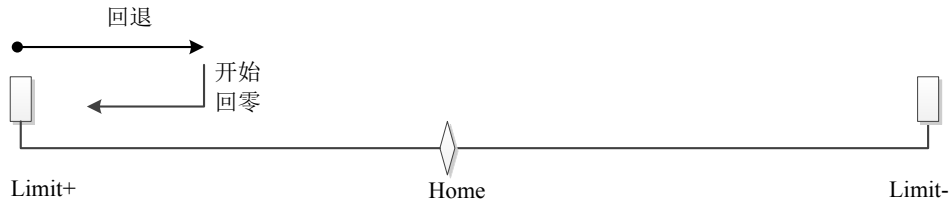


图 12-10 强制 Home+Index 回原点

对于(5)Home 回原点，(6)Home+Index 回原点，(8)强制 Home 回原点，(9)强制 Home+Index 回原点，如果电机所处位置触发的限位开关与运动方向一致（往正方向运动且处于正限位或者往负方向运动处于负限位），则控制器会让电机脱离限位，再反方向去寻找 Home 执行回零动作：

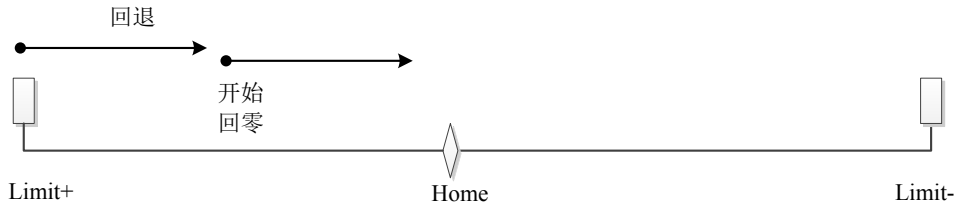


图 12-11 强制 Home+Index 回原点

对于(5)Home 回原点，(6)Home+Index 回原点，(8)强制 Home 回原点，(9)强制 Home+Index 回原点，如果电机所处的位置位于 Home 开关上，则控制器会让电机往搜索 Home 的反方向运动脱离 Home 开关，再开始执行回零动作：

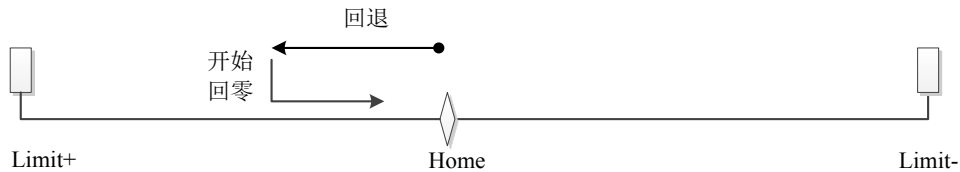


图 12-12 强制 Home+Index 回原点

上面只是简单介绍了几种回原点方式的基本概况，每一种方式都包含正向和负向两种方法，正向指规划位置为正数的方向，负方向指规划位置为负数的方向，例如“限位+Home 回原点”即包括了“正限位+Home 回原点”和“负限位+Home 回原点”；每一种回原点方式，都可以通过设置偏移量使得最终电机停止的位置离原点位置有一个偏移量；每种回原点方式可能由于设定的搜索距离、电机意外停止等因素而找不到原点，大部分异常情况都可以通过查看回原点状态来进行辨识。此外，Smart Home 功能支持各轴单独回原点，互不影响，即可以实现多轴同时回原点。

12.4 注意事项

回零前，确定规划位置与实际位置是否同向，即 `GT_GetPrfPos` 和 `GT_GetEncPos`（如果使用了当量变化，请参照相应指令）获取的位置值同正或同负。如果使用的电机为步进电机，没有配备编码器，那么需要把控制器的编码器选项配置为脉冲计数器。

回零前，需要使能轴，否则无法执行回零。

回零前，先调用 `GT_ZeroPos` 把规划位置 and 实际位置清零，防止规划位置 and 实际位置不一致造成回零不准确。

回零前，需要用户判断电机当前位置是否处于原点触发位置，如果处于原点触发位置，应该先把电机移开远离原点一段距离后再启动回零。

回零时，要查看回零状态，如果发现电机不动了，并已经运动到设定的位置，但是回零状态没有显示回零完成，需要检查一下零点搜索范围是否合理，原点开关是否异常。

回零完成，Smart Home 不会自动清零位置，需要用户调用 `GT_ZeroPos` 把当前位置清零来确定零点。

强制 Home 回原点和强制 Home+Index 回原点是为了方便用户提高回零效率而设置的，但是由于 Home 开关实际是有一定长度，从 Home 开关的左向右运动触发开关和从右向左触发开关得到的捕获位置是不一样的。这两种回零模式下由于机器初始位置可能位于开关的左边也可能处于右边，最终会影响得到的实际原点，用户需要注意这点。

12.5 例程

```
int main(int argc, char* argv[])
{
    short sRtn;
    short gAxis = 1;
    THomeStatus tHomeSts;

    sRtn = GT_Open();
    sRtn = GT_Reset();
    sRtn = GT_AlarmOff(gAxis);
    sRtn = GT_LmtsOn(gAxis);    //如果没有接限位开关则需要把限位取消
    sRtn = GT_LmtSns(3);        //限位为低电平触发（按实际情况设置）
    sRtn = GT_EncSns(1);        //编码器方向设置（按实际编码器设置，保证规划位置与实际位置方向一致）
    sRtn = GT_ClrSts(gAxis);
    sRtn = GT_ZeroPos(gAxis);
    sRtn = GT_AxisOn(gAxis);    //必须使能轴
    //设置 Smart Home 回原点参数
    THomePrm tHomePrm;
    sRtn = GT_GetHomePrm(1,&tHomePrm);
    tHomePrm.mode = 20;
    tHomePrm.moveDir = 1;
    tHomePrm.indexDir = 1;
    tHomePrm.edge = 0;
    tHomePrm.velHigh = 5;
    tHomePrm.velLow = 1;
    tHomePrm.acc = 0.1;
    tHomePrm.dec = 0.1;
    tHomePrm.searchHomeDistance = 200000;
    tHomePrm.searchIndexDistance = 30000;
    tHomePrm.escapeStep = 1000;
```

```
sRtn = GT_GoHome(1,&tHomePrm);           //启动 Smart Home 回原点
do
{
    sRtn = GT_GetHomeStatus(1,&tHomeSts); //获取回原点状态
} while(tHomeSts.run);                   //等待搜索原点停止
```

第13章 指令详细说明



提示

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

指令 1 GT_AlarmOff

指令原型	short GT_AlarmOff(short axis)		
指令说明	控制相应轴驱动报警信号无效。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	控制轴号，正整数。		
指令返回值	若返回值为 1：请检查相应轴在配置文件中是否已经配置了报警无效。 其他返回值：请参照指令返回值列表。		
相关指令	GT_AlarmOn		
指令示例	例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式		

指令 2 GT_AlarmOn

指令原型	short GT_AlarmOn(short axis)		
指令说明	控制轴驱动报警信号有效。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	控制轴号，正整数。		
指令返回值	若返回值为 1：请检查相应轴在配置文件中是否已经配置了报警无效。 其他返回值：请参照指令返回值列表。		
相关指令	GT_AlarmOff		
指令示例	例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式		

指令 3 GT_ArcXYC

指令原型	short GT_ArcXYC(short crd, long x, long y, double xCenter, double yCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。使用圆心描述方法描述圆弧。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。		

	1: 逆时针圆弧。
synVel	插补段的目标合成速度。取值范围: (0, 32767), 单位: pulse/ms。
synAcc	插补段的合成加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。
velEnd	插补段的终点速度。取值范围: [0, 32767], 单位: pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义, 否则该值无效。默认值为: 0。
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。
相关指令	无。
指令示例	例程 6-6 圆弧插补例程

指令 4 GT_ArcXYR

指令原型	short GT_ArcXYR (short crd, long x, long y, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 9 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
x	圆弧插补 x 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
y	圆弧插补 y 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
radius	圆弧插补的圆弧半径值。取值范围: [-1073741823, 1073741823], 单位: pulse。 半径为正时, 表示圆弧为小于等于 180°圆弧。 半径为负时, 表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0: 顺时针圆弧。 1: 逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围: (0, 32767), 单位: pulse/ms。		
synAcc	插补段的合成加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围: [0, 32767], 单位: pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义, 否则该值无效。默认值为: 0		
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。		
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-6 圆弧插补例程		

指令 5 GT_ArcYZC

指令原型	short GT_ArcYZC (short crd, long y, long z, double yCenter, double zCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-6 圆弧插补例程		

指令 6 GT_ArcYZR

指令原型	short GT_ArcYZR (short crd, long y, long z, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。		

	1: 逆时针圆弧。
synVel	插补段的目标合成速度。取值范围: (0, 32767), 单位: pulse/ms。
synAcc	插补段的合成加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。
velEnd	插补段的终点速度。取值范围: [0, 32767], 单位: pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义, 否则该值无效。默认值为: 0。
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。
相关指令	无。
指令示例	例程 6-6 圆弧插补例程

指令 7 GT_ArcZXC

指令原型	short GT_ArcZXC (short crd, long z, long x, double zCenter, double xCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	ZX 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 10 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
z	圆弧插补 z 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围: [-1073741823, 1073741823], 单位: pulse。		
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。		
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0: 顺时针圆弧。 1: 逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围: (0, 32767), 单位: pulse/ms。		
synAcc	插补段的合成加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围: [0, 32767], 单位: pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义, 否则该值无效。默认值为: 0。		
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。		
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-6 圆弧插补例程		

指令 8 GT_ArcZXR

指令原型	short GT_ArcZXR (short crd, long z, long x, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	ZX 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-6 圆弧插补例程		

指令 9 GT_ArcXYZ

指令原型	short GT_ArcXYZ(short crd,long x,long y,long z,double interX,double interY,double interZ,double synVel,double synAcc,double velEnd=0,short fifo=0)		
指令说明	空间圆弧插补，根据前一个点和该指令参数设置的终点和中间点，由三个点确定圆弧，并实现圆弧插补运动。		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 11 个参数，参数的详细信息如下。		
crd	插补坐标系，取值范围：[1,2]		
x	圆弧终点坐标 x 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	圆弧终点坐标 y 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	圆弧终点坐标 z 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
interX	圆弧经过的中间点 x 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
interY	圆弧经过的中间点 y 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
interZ	圆弧经过的中间点 z 值，取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		

syneAcc	合成加速度。取值范围： (0, 32767)， 单位： pulse/ms2。
velEnd	终点速度。取值范围： [0, 32767)， 单位： pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。 默认值为： 0。
fifo	插补缓存区号。 取值范围： [0, 1]， 默认值为： 0。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	<pre>TCrdPrm crdPrm; //建立坐标系 memset(&crdPrm, 0, sizeof(crdPrm)); crdPrm.dimension=3; // 坐标系为二维坐标系 crdPrm.synVelMax=500; // 最大合成速度： 500pulse/ms crdPrm.synAccMax=1; // 最大加速度： 1pulse/ms^2 crdPrm.evenTime = 50; // 最小匀速时间： 50ms crdPrm.profile[0] = 1; // 规划器 1 对应到 X 轴 crdPrm.profile[1] = 2; // 规划器 2 对应到 Y 轴 crdPrm.profile[2] = 3; // 规划器 3 对应到 Z 轴 sRtn = GT_SetCrdPrm(1, &crdPrm); //建立 1 号坐标系， 设置坐标系参数 sRtn = GT_CrdClear(1, 0); //清除此缓存区 sRtn = GT_ArcXYZ(1, 100000, 100000, 0, 50000, 50000, 50000, 50, 1); //一个空间半圆 sRtn = GT_CrdStart(1, 0); //启动插补运动</pre>

指令 10 GT_AxisOff

指令原型	short GT_AxisOff(short axis)		
指令说明	关闭驱动器使能。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	关闭伺服使能的轴的编号。正整数。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_AxisOn		
指令示例	例程 6-1 点位运动		

指令 11 GT_AxisOn

指令原型	short GT_AxisOn(short axis)		
指令说明	打开驱动器使能。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
axis	打开伺服使能的轴的编号。正整数。		
指令返回值	若返回值为 1：		
	(1) 若在配置文件中报警有效，请检查驱动器是否有报警。		
	(2) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。		
	(3) 在闭环控制模式下，采用 MCT2008 对控制器配置，请检查配置 control 一项中是否选择了关联，若没有，选择关联。若已选择，请检查配置 encoder 一项是否选择激活，若没有，选择激活。若已选择，则请检查 PID 参数中的 Kp 是否设置为 0，Kp 必须大于 0，调试阶段可设为一个较小值 5。		

相关指令	其他返回值：请参照指令返回值列表。
指令示例	GT_AxisOff 例程 6-1 点位运动

指令 12 GT_Bind

指令原型	short GT_Bind(short thread, short funId, short page)		
指令说明	绑定线程、函数、数据页。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
thread	线程编号，取值范围：[0, 31]。		
funId	函数标识，可以调用 GT_GetFunId 查询。		
page	数据页编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1：请检查绑定的线程号是否已经有线程绑定并正在运行。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 10-1 运动程序单线程累加求和		

指令 13 GT_BufDA

指令原型	short GT_BufDA (short crd, short chn, short daValue, short fifo=0)		
指令说明	缓存区内输出 DA 值。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
chn	模拟量输出的通道号。取值范围：[1, 8]。		
daValue	模拟量输出的值。取值范围：[-32768, 32767]，其中：-32768 对应-10V，32767 对应+10V。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 14 GT_BufDelay

指令原型	short GT_BufDelay (short crd, unsigned short delayTime, short fifo=0)		
指令说明	缓存区内延时设置指令。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
delayTime	延时时间。取值范围：[0, 16383]，单位：ms。		

fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	例程 6-5 直线插补例程

指令 15 GT_BufGear

指令原型	short GT_BufGear (short crd, short gearAxis, long pos, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴跟随运动。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
gearAxis	需要进行跟随运动的轴号，取值范围：[1, 8]。该轴不能处于坐标系中。		
pos	跟随运动的位移量，单位：pulse。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-10 刀向跟随功能		

指令 16 GT_BufIO

指令原型	short GT_BufIO (short crd, unsigned short doType, unsigned short doMask, unsigned short doValue, short fifo=0)		
指令说明	缓存区内数字量 IO 输出设置指令。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
doType	数字量输出的类型。 MC_ENABLE(该宏定义为 10)：输出驱动器使能。 MC_CLEAR(该宏定义为 11)：输出驱动器报警清除。 MC_GPO(该宏定义为 12)：输出通用输出。		
doMask	从 bit0~bit15 按位表示指定的数字量输出是否有操作。 0：该路数字量输出无操作。1：该路数字量输出有操作。		
doValue	从 bit0~bit15 按位表示指定的数字量输出的值。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：		

	(1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。
相关指令	无。
指令示例	例程 6-5 直线插补例程

指令 17 GT_BufLmtsOff

指令原型	short GT_BufLmtsOff (short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内无效限位开关。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
axis	需要将限位无效的轴的编号。取值范围: [1, 8]。		
limitType	需要无效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0): 需要将该轴的正限位无效。 MC_LIMIT_NEGATIVE(该宏定义为 1): 需要将该轴的负限位无效。 -1: 需要将该轴的正限位和负限位都无效, 默认为该值。		
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。		
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。		
相关指令	GT_BufLmtsOn		
指令示例	无。		

指令 18 GT_BufLmtsOn

指令原型	short GT_BufLmtsOn(short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内有效限位开关。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
crd	坐标系号。正整数, 取值范围: [1, 2]。		
axis	需要将限位有效的轴的编号, 取值范围: [1, 8]。		
limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0): 需要将该轴的正限位设置为有效。 MC_LIMIT_NEGATIVE(该宏定义为 1): 需要将该轴的负限位设置为有效。 -1: 需要将该轴的正限位和负限位都设置为有效, 默认为该值。		
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。		
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动,		

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

指令 19 GT_BufMove

指令原型	short GT_BufMove (short crd, short moveAxis, long pos, double vel, double acc, short modal, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴点位运动。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
moveAxis	需要进行点位运动的轴号，取值范围：[1, 8]。该轴不能处于坐标系中。		
pos	点位运动的目标位置，单位：pulse。		
vel	点位运动的目标速度，单位：pulse/ms。		
acc	点位运动的加速度，单位：pulse/ms ² 。		
modal	点位运动的模式。 0：该指令为非模态指令，即不阻塞后续的插补缓存区指令的执行。 1：该指令为模态指令，将会阻塞后续的插补缓存区指令的执行。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-9 刀向跟随功能		

指令 20 GT_BufSetStopIo

指令原型	short GT_BufSetStopIo (short crd, short axis, short stopType, short inputType, short inputIndex, short fifo=0)		
指令说明	缓存区内设置 axis 的停止 IO 信息。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
axis	需要设置停止 IO 信息的轴的编号。取值范围：[1, 8]。		
stopType	需要设置停止 IO 信息的停止类型。 0：紧急停止类型。 1：平滑停止类型。		
inputType	设置的数字量输入的类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。		

	MC_ALARM(该宏定义为 2): 驱动报警。 MC_HOME(该宏定义为 3): 原点开关。 MC_GPI(该宏定义为 4): 通用输入。 MC_ARRIVE(该宏定义为 5): 电机到位信号。
inputIndex	设置的数字量输入的索引号, 取值范围根据 inputType 的取值而定。 当 inputType= MC_LIMIT_POSITIVE 时, 取值范围: [1, 8]。 当 inputType= MC_LIMIT_NEGATIVE 时, 取值范围: [1, 8]。 当 inputType= MC_ALARM 时, 取值范围: [1, 8]。 当 inputType= MC_HOME 时, 取值范围: [1, 8]。 当 inputType= MC_GPI 时, 取值范围: [1, 16]。 当 inputType= MC_ARRIVE 时, 取值范围: [1, 4]。
fifo	插补缓存区号。正整数, 取值范围: [0, 1], 默认值为: 0。
指令返回值	若返回值为 1: (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据, 若是, 则检查 fifo0 是否使用并运动, 若运动, 则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值: 请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 21 GT_ClearCaptureStatus

指令原型	short GT_ClearCaptureStatus(short encoder)		
指令说明	清除捕获状态。		
指令类型	立即指令, 调用后立即生效。	章节页码	112
指令参数	该指令共有 1 个参数, 参数的详细信息如下。		
encoder	需要被清除捕获状态的编码器轴号。正整数。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 22 GT_Close

指令原型	short GT_Close()		
指令说明	关闭运动控制器。		
指令类型	立即指令, 调用后立即生效。	章节页码	145
指令参数	无。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_Open		
指令示例	例程 11-1 读取运动控制器版本号		

指令 23 GT_ClrSts

指令原型	short GT_ClrSts(short axis, short count=1)
指令说明	清除驱动器报警标志、跟随误差超限标志、限位触发标志。

	1. 只有当驱动器没有报警时才能清除轴状态字的报警标志; 2. 只有当跟随误差正常以后, 才能清除跟随误差超限标志; 3. 只有当离开限位开关, 或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志。		
指令类型	立即指令, 调用后立即生效。	章节页码	58
指令参数	该指令共有 2 个参数, 参数的详细信息如下。		
axis	起始轴号。正整数。		
count	读取的轴数, 默认为 1。正整数。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetSts		
指令示例	例程 4-1 修改编码器计数方向		

指令 24 GT_CompareContinuePulseMode

指令原型	short GT_CompareContinuePulseMode(short mode,short count ,short standTime)		
适用板卡	GTS-400-PG, GTS-400-PV, GTS-800-PG, GTS-800-PV		
指令说明	设置一维位置比较输出是否为连续脉冲模式。控制器复位后如果没有调用该指令则一维位置比较输出脉冲为默认模式（比较输出单个脉冲）。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数, 参数的详细信息如下。		
mode	输出脉冲模式, 取值范围[0,1], 0-表示默认模式, 即达比较位置时只输出单个脉冲, 1-表示连续脉冲模式, 即达到比较位置时输出 count 个脉冲。		
count	设置为连续脉冲模式时, 调用 GT_ComparePulse 也能够输出多个脉冲, 只是无法连续脉冲模式输出的脉冲个数, 当 mode=0 时, 该值不起作用。		
standTime	连续脉冲模式输出时脉冲与脉冲之间的时间间隔, 取值范围(0,65535]us。		
相关指令	GT_ComparePulse , GT_CompareData 。		
指令示例	<pre>//一维位置比较直接输出脉宽为500us的单个脉冲 sRtn = GT_CompareContinuePulseMode(0,3,500); sRtn=GT_ComparePulse(1,0,500); // 一维位置比较直接输出3个脉宽10ms、间隔8ms的连续脉冲 sRtn = GT_CompareContinuePulseMode(1,3,8000); sRtn=GT_ComparePulse(1,0,10000); 另外, 下面的例子显示了在每个比较点都会输出脉宽为 500us、间隔为 500us 的连续 3 个脉冲, 如果这种模式不是您希望的, 请设置位置比较输出脉冲模式为默认模式。 long pBuf1[5]={20000,40000,60000,80000,100000}; long pBuf2[5]={0}; sRtn = GT_CompareContinuePulseMode(1,3,500); sRtn = GT_CompareData(1,0,0,0,500,pBuf1,5,pBuf2,0);</pre>		

指令 25 GT_CompareData

指令原型	short GT_CompareData(short encoder, short source, short pulseType, short startLevel, short time, long *pBuf1, short count1, long *pBuf2, short count2)
指令说明	设置位置比较参数并启动位置比较输出

指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
encoder	需要进行位置比较的编码器轴号，正整数。		
source	位置比较数据源。0 表示使用内部脉冲计数器，1 表示使用外部编码器。 比较源必须与控制器配置文件中设置的“脉冲计数源”一致，脉冲计数源为“脉冲计数器”时，则此处取 0，脉冲计数源为“编码器”时，此处取 1。		
pulseType	HSIO 输出方式： 0 表示输出脉冲，脉冲宽度由 time 参数设定， 1 表示输出电平。		
startLevel	按位设置 HSIO 输出的初始电平， bit0 表示通道 1，bit1 表示通道 2； 0：表示位置比较输出引脚初始电平为复位状态（恢复到初始化后的电平状态）； 1：表示位置比较输出引脚初始电平为取反状态（与初始化后的电平状态相反）。 若 pulseType 为 1，本参数建议设置为 0。		
time	pulseType 为 0 时，该参数用来设定脉冲输出宽度，取值范围：[1, 32767]，单位：us。 pulseType 为 1 时，本参数无效。		
pBuf1	HSIO0 的比较位置缓冲区，位置值为相对于调用该指令时的当前位置的偏移，必须是单调上升的正数序列或单调下降的负数序列		
count1	pBuf1 的长度，最大值为 4096		
pBuf2	HSIO1 的比较位置缓冲区，位置值为相对于调用该指令时的当前位置的偏移，必须是单调上升的正数序列或单调下降的负数序列。且 pBuf1 和 pBuf2 的单调性必须相同		
count2	pBuf2 的长度，最大值为 4096		
指令返回值	若返回值为 7： 检查位置数组是否单调上升的正数序列或单调下降的负数序列。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 11-3 位置比较输出指令详细的用法		

指令 26 GT_CompareLinear

指令原型	short GT_CompareLinear(short encoder, long startPos, long repeatTimes, short interval, short time, short source)		
指令说明	设置等间距重复位置比较输出。		
指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
encoder	位置比较的编码器轴号，正整数。		
channel	位置比较输出通道，取值为 1~2		
startPos	开始进行比较的起始位置，单位：pulse，这个起始位置是相对位置，即相对于调用该指令时的当前位置的偏移量。		
repeatTimes	比较输出重复次数		
interval	输出位置间隔，单位：pulse，要求与 startPos 同为正数或同为负数，且不能为 0		
time	输出脉冲宽度，取值范围 1~32767，单位为微秒。		
source	位置比较数据源。0 表示使用内部脉冲计数器，1 表示使用外部编码器。		

指令返回值	比较源必须与配置控制器时设置的“脉冲计数源”一致，脉冲计数源为“脉冲计数器”时，则此处取 0，脉冲计数源为“编码器”时，此处取 1。
	若返回值为 7： (1) 检查 startPos 与 interval 是否同为正数或同为负数。 (2) 检查 interval 是否为 0。
	其他返回值：请参照指令返回值列表。
	无。
相关指令	无。
指令示例	例程 11-3 位置比较输出指令详细的用法

指令 27 GT_ComparePulse

指令原型	short GT_ComparePulse(short level, short pulsetype, short time)		
指令说明	在 HSIO 口输出脉冲或者电平		
指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
level	pulseType 为 0 时，本参数按位设置 HSIO 是否输出脉冲。 bit0 表示通道 1，bit1 表示通道 2； 0：表示本通道不输出脉冲； 1：表示本通道输出脉冲。		
	pulseType 为 1 时，本参数按位设置 HSIO 的输出电平。 bit0 表示通道 1，bit1 表示通道 2； 0：位置比较输出引脚电平复位（恢复到初始化后的电平状态）； 1：位置比较输出引脚电平取反（与初始化后的电平状态相反）。		
pulsetype	HSIO 输出方式： 0 表示输出脉冲，脉冲宽度由 time 参数设定， 1 表示输出电平。		
time	pulseType 为 0 时，该参数用来设定脉冲输出宽度，取值范围：[1, 32767]，单位：us。 pulseType 为 1 时，本参数无效。		
指令返回值	若返回值为 1： 检查是否已经启动位置比较输出，若正在进行位置比较输出，无法在 HSIO 口输出电平或脉冲。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 11-3 位置比较输出指令详细的用法		

指令 28 GT_CompareStatus

指令原型	short GT_CompareStatus(short *pStatus, long *pCount)		
指令说明	查询位置比较状态		
指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
pStatus	指示位置比较是否结束。1 表示位置比较结束，0 表示正在进行位置比较。		
pCount	读取位置比较已输出的上升沿个数， 必须传递一个长度为 2 的数组。		
	若使用调用指令 GT_CompareData，即以电平方式输出，则返回的 pCount[0]及 pCount[1]为通道 1 及通道 2 已输出的上升沿个数，但与已完成的位置比较个数不等		

	（以电平方式输出时，已完成的位置比较个数为已输出上升沿和已输出下降沿个数的总和）； 若以脉冲方式输出，则返回的 pCount[0]及 pCount[1]为通道 1 及通道 2 已输出的上升沿个数，与已完成的位置比较个数相等。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 29 GT_CompareStop

指令原型	short GT_CompareStop()		
指令说明	取消位置比较输出		
指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令无参数。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 30 GT_CrdClear

指令原型	short GT_CrdClear(short crd, short fifo)		
指令说明	清除插补缓存区内的插补数据。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
fifo	所要清除的插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-5 直线插补例程		

指令 31 GT_CrdData

指令原型	short GT_CrdData (short crd, TCrdData *pCrdData, short fifo=0)		
指令说明	用于在使用前瞻时。调用该指令表示后续没有新的数据，将会一次性把前瞻缓存区的数据压入运动缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pCrdData	只能设置为：NULL。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]。默认值为：0。		
指令返回值	若返回值为非零值，说明前瞻缓存区还有数据没有被压入运动缓存区，而运动缓存区没有空间了。此时需要检查运动缓存区的空间（调用 GT_CrdSpace 检查）。当检		

相关指令	查运动缓存区有空间时，再次调用 GT_CrdData 指令，直至返回值为 0 时，前瞻缓存区的数据才被完全送入运动缓存区。
指令示例	无。
指令示例	例程 6-8 前瞻预处理例程

指令 32 GT_CrdSpace

指令原型	short GT_CrdSpace (short crd, long *pSpace, short fifo=0)		
指令说明	查询插补缓存区剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSpace	读取插补缓存区中的剩余空间。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-8 前瞻预处理例程		

指令 33 GT_CrdStart

指令原型	short GT_CrdStart (short mask, short option)		
指令说明	启动插补运动。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
mask	从 bit0~bit1 按位表示需要启动的坐标系。 bit0 对应坐标系 1，bit1 对应坐标系 2。 0：不启动该坐标系，1：启动该坐标系。		
option	从 bit0~bit1 按位表示坐标系需要启动的缓存区的编号。 bit0 对应坐标系 1，bit1 对应坐标系 2。 0：启动坐标系中 FIFO0 的运动，1：启动坐标系中 FIFO1 的运动。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 若使用了辅助 fifo1 运动，检查当前坐标系位置没有恢复到 fifo0 断点坐标系位置。 (3) 检查参数设置是否启动了坐标系。 (4) 检查坐标系是否在运动。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-5 直线插补例程		

指令 34 GT_CrdStatus

指令原型	short GT_CrdStatus (short crd, short *pRun, long *pSegment, short fifo=0)		
指令说明	查询插补运动坐标系状态。		
指令类型	立即指令，调用后立即生效。	章节页码	76

指令参数	该指令共有 4 个参数，参数的详细信息如下。
crd	坐标系号。正整数，取值范围：[1, 2]。
pRun	读取插补运动状态。0：该坐标系的该 FIFO 没有在运动；1：该坐标系的该 FIFO 正在进行插补运动。
pSegment	读取当前已经完成的插补段数。当重新建立坐标系或者调用 GT_CrdClear 指令后，该值会被清零。
fifo	所要查询运动状态的插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	例程 6-5 直线插补例程

指令 35 GT_CtrlMode

指令原型	short GT_CtrlMode(short axis, short mode)		
指令说明	设置控制轴为模拟量输出或脉冲输出。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	控制轴号。正整数。		
mode	切换的模式。 0：将指定轴切换为闭环控制模式(即电压控制方式)。 1：将指定轴切换为开环控制模式(即脉冲控制方式)。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 4-4 设置第 1 轴为闭环控制方式		

指令 36 GT_Download

指令原型	short GT_Download(char *pFileName)		
指令说明	下载运动程序到运动控制器。注：文件包含的线程数不能大于 32。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pFileName	下载到运动控制器的运动程序文件名		
指令返回值	若返回值为 2007： (1) 请检查文件名是否太长。 (2) 请检查需要下载的文件是否存在。 若返回值为 2008：表示打开文件失败。 (1) 请检查文件是否损坏，编译是否成功。 (2) 请检查 ini 和 bin 文件是否在同一根目录。 若返回值为 7：请检查线程数量是否大于 32。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 10-1 运动程序单线程累加求和		

指令 37 GT_EnableLeadScrewComp

指令原型	short GT_EnableLeadScrewComp(short axis,short mode)		
指令说明	使能螺距误差补偿功能。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	轴的编号。正整数。		
mode	是否使能螺距误差补偿功能，0-不使能，1-使能		
指令返回值	请参照指令返回值列表		
相关指令	GT_SetLeadScrewComp		
指令示例	无。		

指令 38 GT_EncOff

指令原型	short GT_EncOff(short encoder)		
指令说明	设置为“脉冲计数器”计数方式。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
encoder	编码器通道号。正整数。		
指令返回值	请参照指令返回值列表		
相关指令	GT_EncOn		
指令示例	无。		

指令 39 GT_EncOn

指令原型	short GT_EncOn(short encoder)		
指令说明	设置为“外部编码器”计数方式。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下：		
encoder	编码器通道号。正整数。		
指令返回值	请参照指令返回值列表		
相关指令	GT_EncOff		
指令示例	无。		

指令 40 GT_EncScale

指令原型	short GT_EncScale(short axis, short alpha, short beta)		
指令说明	设置控制轴的编码器当量变换值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	控制轴号。正整数。		
alpha	规划器当量的alpha值，取值范围：(-32767, 0)和(0, 32767)。		
beta	规划器当量的beta值，取值范围：(-32767, 0)和(0, 32767)。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GT_ProfileScale		
指令示例	无。		

指令 41 GT_EncSns

指令原型	short GT_EncSns(unsigned short sense)		
指令说明	设置编码器的计数方向。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
sense	按位标识编码器的计数方向。 对于 8 轴控制器：bit0~bit7 依次对应编码器 1~8，bit8 对应辅助编码器。 对于 4 轴控制器：bit0~bit3 依次对应编码器 1~4，bit8 对应辅助编码器 1，bit9 对应辅助编码器 2。 0：该编码器计数方向不取反。 1：该编码器计数方向取反。 系统复位后，默认为编码器计数方向取反。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 42 GT_GearStart

指令原型	short GT_GearStart(long mask)									
指令说明	启动电子齿轮运动。									
指令类型	立即指令，调用后立即生效。							章节页码		72
指令参数	该指令共有 1 个参数，参数的详细信息如下。									
mask	按位指示需要启动 Gear 运动的轴号。当 bit 位为 1 时表示启动对应的轴。									
	对于四轴控制器：									
	Bit		3	2	1	0				
	对应轴		4 轴	3 轴	2 轴	1 轴				
指令返回值	对于 8 轴控制器：									
	Bit		7	6	5	4	3	2	1	0
	对应轴		8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
	若返回值为 1：									
指令返回值	(1) 请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_PrflGear 将当前轴设置为电子齿轮模式。									
	(2) 请检查主轴是否已设置。									
	(3) 请检查传动比是否已设置。									
指令返回值	其他返回值：请参照指令返回值列表。									
	无。									
相关指令	无。									
指令示例	例程 6-3 电子齿轮跟随									

指令 43 GT_GetAdc

指令原型	short GT_GetAdc (short adc, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取模拟量输入的电压值。		
指令类型	立即指令，调用后立即生效。	章节页码	111
指令参数	该指令共有 4 个参数，参数的详细信息如下。		

adc	adc 起始通道号，取值范围：[1, 8]。
pValue	读取的输入电压值。单位：伏特。
count	读取的通道数，默认为 1。 1 次最多可以读取 8 路 adc 输入电压值。
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	GT_GetAdcValue
指令示例	例程 7-4 访问 ADC

指令 44 GT_GetAdcValue

指令原型	short GT_GetAdcValue (short adc, short *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取模拟量输入的数字转换值。		
指令类型	立即指令，调用后立即生效。	章节页码	111
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
adc	adc 起始通道号，取值范围：[1, 8]。		
pValue	读取的输入电压数值。单位：bit，取值范围：[-26214, 26214]，对应的电压值为[-10, 10]伏特。		
count	读取的通道数。默认为 1。 1 次最多可以读取 8 路 adc 输入电压值。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetAdc		
指令示例	例程 7-4 访问 ADC		

指令 45 GT_GetAxisBand

指令原型	short GT_GetAxisBand(short axis, long *pBand, long *pTime)		
指令说明	读取轴到位误差带。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号。正整数。		
pBand	读取误差带大小。		
pTime	读取误差带保持时间。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetAxisBand		
指令示例	无。		

指令 46 GT_GetAxisEncAcc

指令原型	short GT_GetAxisEncAcc(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 encoder 输出值经过当量变换之后的编码器加速度值。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		

axis	起始轴号。正整数。
pValue	轴的编码器加速度。单位: pulse/ms ² 。
count	读取的轴数, 默认为 1。
pClock	读取控制器时钟, 默认值为: NULL, 即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 47 GT_GetAxisEncPos

指令原型	short GT_GetAxisEncPos(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 encoder 输出值经过当量变换之后的编码器位置值。		
指令类型	立即指令, 调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
axis	起始轴号。正整数。		
pValue	轴的编码器位置。单位: pulse。		
count	读取的轴数, 默认为 1。正整数。		
pClock	读取控制器时钟, 默认值为: NULL, 即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 8-2 Home 回原点		

指令 48 GT_GetAxisEncVel

指令原型	short GT_GetAxisEncVel(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 encoder 输出值经过当量变换之后的编码器速度值。		
指令类型	立即指令, 调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		
axis	起始轴号。正整数。		
pValue	轴的编码器速度。单位: pulse/ms。		
count	读取的轴数, 默认为 1。正整数。		
pClock	读取控制器时钟, 默认值为: NULL, 即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 49 GT_GetAxisError

指令原型	short GT_GetAxisError(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 经过当量变换之后的规划位置与 encoder 经过当量变换之后的编码器位置的差值。		
指令类型	立即指令, 调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数, 参数的详细信息如下。		

axis	起始轴号。正整数。
pValue	轴的规划位置与编码器位置的差值。单位：pulse。
count	读取的轴数，默认为 1。正整数。
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 50 GT_GetAxisPrfAcc

指令原型	short GT_GetAxisPrfAcc(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。正整数。		
pValue	轴的规划加速度。单位：pulse/ms ² 。		
count	读取的轴数，默认为 1。正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 51 GT_GetAxisPrfPos

指令原型	short GT_GetAxisPrfPos(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划位置。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。正整数。		
pValue	轴的规划位置。单位：pulse。		
count	读取的轴数，默认为 1。正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 8-2 Home 回原点		

指令 52 GT_GetAxisPrfVel

指令原型	short GT_GetAxisPrfVel(short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划速度。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号。正整数。		

pValue	轴的规划速度。单位：pulse/ms。
count	读取的轴数，默认为 1。正整数。
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 53 GT_GetBacklash

指令原型	short GT_GetBacklash (short axis, long *pCompValue, double *pCompChangeValue, long *pCompDir)		
指令说明	读取反向间隙补偿的相关参数。		
指令类型	立即指令，调用后立即生效。	章节页码	153
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	查询的轴号。正整数。		
pCompValue	读取的反向间隙补偿值。		
pCompChangeValue	读取的反向间隙补偿值的变化量。		
pCompDir	读取的反向间隙补偿的补偿方向。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetBacklash		
指令示例	无。		

指令 54 GT_GetCaptureRepeatPos

指令原型	short GT_GetCaptureRepeatPos(short encoder, long *pValue, short startNum, short count)		
指令说明	查询重复捕获位置值。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
encoder	作为重复捕获的编码器轴号，正整数。		
pValue	读取编码器捕获值，可能有多组值，应传入一个长度比 GT_SetCaptureRepeat 的 count 参数大的数组。		
startNum	起始读取捕获位置索引号		
count	总读取个数，最大为已捕获次数		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 55 GT_GetCaptureRepeatStatus

指令原型	short GT_GetCaptureRepeatStatus(short encoder, short *pCount)		
指令说明	查询重复捕获状态。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
encoder	重复捕获的编码器轴号，正整数。		
pCount	读取编码器捕获次数，0 表示未触发，大于 0 代表已触发捕获，该值等于已触发次		

指令返回值	数
相关指令	请参考指令返回值列表。
指令示例	例程 8-6 重复捕获使用说明

指令 56 GT_GetCaptureMode

指令原型	short GT_GetCaptureMode(short encoder, short *pMode, short count=1)		
指令说明	读取编码器捕获方式。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
encoder	编码器起始轴号。正整数。		
pMode	编码器捕获模式。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
指令返回值	请参考指令返回值列表。		
相关指令	GT_SetCaptureMode		
指令示例	例程 8-1 Home/Index 捕获		

指令 57 GT_GetCaptureStatus

指令原型	short GT_GetCaptureStatus (short encoder, short *pStatus, long *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取编码器捕获状态。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
encoder	编码器起始轴号。正整数。		
pStatus	读取编码器捕获状态。 为 1 时表示对应轴捕获触发。		
pValue	读取编码器捕获值。 当捕获触发时，编码器捕获值会自动更新。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
pClock	读取控制器时钟。		
指令返回值	请参考指令返回值列表。		
相关指令	GT_GetCaptureMode		
指令示例	例程 8-1 Home/Index 捕获		

指令 58 GT_GetCardNo

指令原型	short GT_GetCardNo(short *pIndex)		
指令说明	读取当前运动控制器卡号。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pIndex	读取的当前运动控制器的卡号。		
指令返回值	请参考指令返回值列表。		

相关指令	GT_SetCardNo
指令示例	无。

指令 59 GT_GetClock

指令原型	short GT_GetClock(unsigned long *pClock, unsigned long *pLoop=NULL)		
指令说明	读取运动控制器系统时钟。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
pClock	读取的运动控制器的时钟，单位：ms		
pLoop	内部使用，默认值为：NULL，即不读取该值		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 60 GT_GetClockHighPrecision

指令原型	shortGT_GetClockHighPrecision(unsigned long *pClock)		
指令说明	读取运动控制器系统高精度时钟。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pClock	读取的运动控制器的时钟，单位：125us。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 61 GT_GetControlFilter

指令原型	short GT_GetControlFilter(short control, short *pIndex)		
指令说明	读取当前 PID 索引。		
指令类型	立即指令，调用后立即生效。	章节页码	152
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号。正整数。		
pIndex	读取的伺服控制参数的索引号。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetControlFilter		
指令示例	无。		

指令 62 GT_GetCrdBufferMode

指令原型	short GT_GetCrdBufferMode(short crd,short *pBufferMode,short fifo);		
指令说明	获取插补缓冲区的模式（静态模式或动态模式）。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pBufferMode	插补缓冲区模式： CRD_BUFFER_MODE_DYNAMIC_DEFAULT 0:		

	动态模式，插补运动完成后缓冲区内的数据会自动被清除。 CRD_BUFFER_MODE_STATIC_INPUT 11: 静态模式输入数据阶段 CRD_BUFFER_MODE_STATIC_READY 12: 静态模式输入数据完成 静态模式下，插补运动完成控制器不会自动清除缓冲区数据，再次调用 GT_CrdStart 可以重新启动运动。
fifo	插补缓存区号。取值范围： [0, 1], 默认值为： 0。
返回值	GT_SetCrdBufferMode
相关指令	
指令示例	例程 6-12 静态插补缓冲区模式

指令 63 GT_GetCrdPos

指令原型	short GT_GetCrdPos (short crd, double *pPos)		
指令说明	查询该坐标系的当前坐标位置值。获取的坐标值可能和规划位置不一致，取决于建立坐标系的原点是否为零。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围： [1, 2]。		
pPos	读取的坐标系的坐标值。单位： pulse。该参数应该为一个数组首元素的指针，数组的元素个数取决于该坐标系的维数。		
指令返回值	请参照指令返回值列表		
相关指令	无。		
指令示例	例程 6-7 插补 FIFO 管理		

指令 64 GT_GetCrdPrm

指令原型	short GT_GetCrdPrm (short crd, TCrdPrm *pCrdPrm)		
指令说明	查询坐标系参数。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围： [1, 2]。		
pCrdPrm	读取坐标系的相关参数 结构体的成员含义参照 GT_SetCrdPrm 指令说明。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetCrdPrm		
指令示例	无。		

指令 65 GT_GetCrdSegmentTime

指令原型	short GT_GetCrdSegmentTime(short crd, long segmentIndex, double *pSegmentTime, long *pSegmentNumber, short fifo);		
指令说明	获取插补段的执行时间。		
指令类型	立即指令。	章节页码	74
指令参数	该指令共有 5 个参数，参数的详细信息如下。		

crd	坐标系号。正整数，取值范围：[1, 2]。
segmentIndex	插补段的索引号，为数据压入控制器缓冲区的顺序，取值范围：[1,N]。
pSegmentTime	指定的插补段所需的执行时间。当未启动插补运动，则所计算的时间为预估时间；执行完插补运动再调用该指令，得到的为实际执行时间；如果使用前瞻，则需要等到把前瞻的指令压入控制器后才能获取到该段的执行时间。单位：ms。
pSegmentNumber	插补段的用户段号，如果没有使用 GT_LnXYWN 等指令，则该值为 0。
fifo	插补缓存区号。取值范围：[0, 1]，默认值为：0。
返回值	请参照指令返回值列表。
相关指令	GT_GetCrdTime
指令示例	无。

指令 66 GT_GetCrdStopDec

指令原型	short GT_GetCrdStopDec (short crd, double *pDecSmoothStop, double *pDecAbruptStop)		
指令说明	查询插补运动平滑停止、急停合成加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pDecSmoothStop	查询坐标系合成平滑停止加速度，单位：pulse/ms ² 。		
pDecAbruptStop	查询坐标系合成急停加速度，单位：pulse/ms ² 。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 67 GT_GetCrdTime

指令原型	short GT_GetCrdTime(short crd, TCrdTime ,short fifo);		
指令说明	获取插补运动的执行时间。		
指令类型	立即指令。	章节页码	74
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pTime	<pre> TCrdTime { double time; //插补运动总时间，单位：ms long segmentUsed; //缓冲区位置指针，不需要了解 long segmentHead; long segmentTail; } </pre>		
fifo	插补缓存区号。取值范围：[0, 1]，默认值为：0。		
返回值	请参照指令返回值列表。		
相关指令	指令 65 GT_GetCrdSegmentTime		
指令示例	无。		

指令 68 GT_GetCrdVel

指令原型	short GT_GetCrdVel(short crd, double *pSynVel)		
指令说明	查询该坐标系的当前坐标速度值。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSynVel	读取的坐标系的合成速度值，单位：pulse/ms。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 69 GT_GetDac

指令原型	short GT_GetDac(short dac, short *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 dac 输出电压。		
指令类型	立即指令，调用后立即生效。	章节页码	109
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
dac	dac 起始轴号。		
pValue	输出电压。		
count	读取的通道数。默认为 1。 1 次最多可以读取 8 个 dac 轴。		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetDac		
指令示例	例程 7-3 访问 DAC		

指令 70 GT_GetDi

指令原型	short GT_GetDi(short diType, long *pValue)		
指令说明	读取数字 IO 输入状态。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。 MC_MPG(该宏定义为 6)：手轮 MPG 轴选和倍率信号（24V 电平输入）。		
pValue	数字 IO 输入状态，按位指示 IO 输入电平(根据配置工具 di 的 reverse 值不同而不同)。 当 reverse=0 时，1 表示高电平，0 表示低电平。 当 reverse=1 时，1 表示低电平，0 表示高电平。		
指令返回值	请参照指令返回值列表。		

相关指令	无。
指令示例	例程 7-1 访问数字 IO

指令 71 GT_GetDiRaw

指令原型	short GT_GetDiRaw(short diType, long *pValue)		
指令说明	读取数字 IO 输入状态的原始值。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
diType	指定数字 IO 类型。		
	MC_LIMIT_POSITIVE(该宏定义为 0): 正限位。		
	MC_LIMIT_NEGATIVE(该宏定义为 1): 负限位。		
	MC_ALARM(该宏定义为 2): 驱动报警。		
	MC_HOME(该宏定义为 3): 原点开关。		
	MC_GPI(该宏定义为 4): 通用输入。		
pValue	MC_ARRIVE(该宏定义为 5): 电机到位信号。		
	数字 IO 输入状态的原始值，按位指示 IO 输入电平。		
	1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 72 GT_GetDiReverseCount

指令原型	short GT_GetDiReverseCount (short diType, short diIndex, unsigned long *pReverseCount, short count)		
指令说明	取数字量输入信号的变化次数。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
diType	指定数字 IO 类型。		
	MC_LIMIT_POSITIVE(该宏定义为 0): 正限位。		
	MC_LIMIT_NEGATIVE(该宏定义为 1): 负限位。		
	MC_ALARM(该宏定义为 2): 驱动报警。		
	MC_HOME(该宏定义为 3): 原点开关。		
	MC_GPI(该宏定义为 4): 通用输入。		
diIndex	MC_ARRIVE(该宏定义为 5): 电机到位信号。		
	数字量输入的索引。		
	取值范围:		
	diType= MC_LIMIT_POSITIVE 时: [1, 8]。		
	diType= MC_LIMIT_NEGATIVE 时: [1, 8]。		
	diType= MC_ALARM 时: [1, 8]。		
	diType= MC_HOME 时: [1, 8]。		
pReverseCount	diType= MC_GPI 时: [1, 16]。		
	diType= MC_ARRIVE 时: [1, 4]。		
	读取的数字量输入的变化次数。		
count	读取变化次数的数字量输入的个数，默认为 1。		

指令返回值	1 次最多可以读取 4 个数字量输入的变化次数。
相关指令	无。
指令示例	例程 7-1 访问数字 IO

指令 73 GT_GetDo

指令原型	short GT_GetDo (short doType, long *pValue)		
指令说明	读取数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10): 驱动器使能。 MC_CLEAR(该宏定义为 11): 报警清除。 MC_GPO(该宏定义为 12): 通用输出。		
pValue	数字 IO 输出状态，按位指示 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 74 GT_GetEncPos

指令原型	short GT_GetEncPos (short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	108
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
encoder	编码器起始轴号。正整数。 对于 4 轴控制器，取值范围：[1, 4]和[9, 11]。第 9, 10 轴为两路辅助编码器，第 11 轴为手轮接口专用编码器。 对于 8 轴控制器，取值范围：[1, 11]。第 9,10 轴为 2 路辅助编码器。第 11 轴为手轮接口专用编码器。		
pValue	编码器位置。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 7-2 读取 8 个轴编码器和辅助编码器位置值		

指令 75 GT_GetEncVel

指令原型	short GT_GetEncVel (short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取编码器速度		

指令类型	立即指令，调用后立即生效。	章节页码	108
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
encoder	编码器起始轴号。正整数。 对于 4 轴控制器，取值范围：[1, 4]和[9, 11]。第 9, 10 轴为两路辅助编码器，第 11 轴为手轮接口专用编码器。 对于 8 轴控制器，取值范围：[1, 11]。第 9,10 轴为 2 路辅助编码器。第 11 轴为手轮接口专用编码器。		
pValue	编码器速度。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 7-2 读取 8 个轴编码器和辅助编码器位置值		

指令 76 GT_GetFunId

指令原型	short GT_GetFunId (char *pFunName, short *pFunId)		
指令说明	读取运动程序中函数的标识。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
pFunName	运动程序函数名称。		
pFunId	根据运动程序函数名称查询函数标识。		
指令返回值	若返回值为 1，2007 或者 2008： 请检查重新检查 GT_Download 是否调用成功。 若失败，请根据 GT_Download 返回值提示操作，直至成功。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 10-1 运动程序单线程累加求和		

指令 77 GT_GetGearMaster

指令原型	short GT_GetGearMaster (short profile, short *pMasterIndex, short *pMasterType, short *pMasterItem)		
指令说明	读取电子齿轮运动跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	72
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。正整数。		
pMasterIndex	读取主轴索引。正整数。		
pMasterType	读取主轴类型。 GEAR_MASTER_PROFILE（该宏定义为）表示跟随规划轴(profile)的输出值。 GEAR_MASTER_ENCODER（该宏定义为）表示跟随编码器(encoder)的输出值。 GEAR_MASTER_AXIS（该宏定义为）表示跟随轴(axis)的输出值。		
pMasterItem	读取输出位置类型。当 masterType=GEAR_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值，默认为该值。 1 表示 axis 的编码器位置输出值。		

指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_Prfgear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。
相关指令	GT_SetGearMaster
指令示例	无。

指令 78 GT_GetGearRatio

指令原型	short GT_GetGearRatio(short profile, long *pMasterEven, long *pSlaveEven, long *pMasterSlope)		
指令说明	读取电子齿轮比。		
指令类型	立即指令，调用后立即生效。	章节页码	72
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号。正整数。		
pMasterEven	读取传动比系数，主轴位移。单位：pulse。		
pSlaveEven	读取传动比系数，从轴位移。单位：pulse。		
pMasterSlope	读取主轴离合器区位移。单位：pulse。		
指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_Prfgear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetGearRatio		
指令示例	无。		

指令 79 GT_GetHomePrm

指令原型	short GT_GetHomePrm(short axis,THomePrm *pHomePrm)		
指令说明	读取设置到控制器的Smart Home回原点参数。		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	进行回原点的轴号，对于 4 轴卡，取值范围：[1, 4]，对于 8 轴卡，取值范围[1,8]。		
pHomePrm	获取Smart Home回原点的参数，该参数为一结构体，详细参数定义及说明请参照结构体THomePrm。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例			

指令 80 GT_GetHomeStatus

指令原型	GT_GetHomeStatus(short axis,THomeStatus *pHomeStatus)		
指令说明	获取Smart Home回原点的状态。		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	进行回原点的轴号，对于 4 轴卡，取值范围：[1, 4]，对于 8 轴卡，取值范围[1,8]。		
pHomeStatus	获取Smart Home回原点的状态参数，该参数为一结构体，详细参数定义及说明请参照结构体THomeStatus。 typedef struct		

```

{
    short run;      // 是正在进行回原点，0—已停止运动，1-正在回
                    原点
    short stage;    // 回原点运动的阶段
    short error;    // 回原点过程的发生的错误
    short pad1;     // 保留（无具体含义）
    long capturePos; // 捕获到Home或Index时刻的编码器位置
    long targetPos;  // 需要运动到的目标位置（原点位置或者原点
                    位置+偏移量），在搜索Limit时或者搜索Home
                    或Index时，设置的搜索距离为0，那么该值显示
                    为805306368
} THomeStatus;

```

回原点运动的阶段宏定义：

HOME_STAGE_IDLE	(0): 未启动Smart Home回 原点
HOME_STAGE_START	(1): 启动Smart Home回原点
HOME_STAGE_ON_HOME_LIMIT_ESCAPE	(2): 正在从原点或限位上回退
HOME_STAGE_SEARCH_LIMIT	(10): 正在寻找限位
HOME_STAGE_SEARCH_LIMIT_STOP	(11): 触发限位停止
HOME_STAGE_SEARCH_LIMIT_ESCAPE	(13): 反方向运动脱离限位
HOME_STAGE_SEARCH_LIMIT_RETURN	(15): 重新回到限位
HOME_STAGE_SEARCH_LIMIT_RETURN_STOP	(16): 重新回到限位停止
HOME_STAGE_SEARCH_HOME	(20): 正在搜索Home
HOME_STAGE_SEARCH_HOME_STOP	(22): HOME_MODE_FORCED_HOME 和 HOME_MODE_FORCED_HOME_ INDEX模式下, 搜索Home过程中遇 到限位停止, 准备反向搜索
HOME_STAGE_SEARCH_HOME_RETURN	(25): 搜索到Home后运动到 捕获的Home位置
HOME_STAGE_SEARCH_INDEX	(30): 正在搜索Index
HOME_STAGE_GO_HOME	(80): 正在执行回原点过程
HOME_STAGE_END	(100): 回原点结束

回原点过程的发生的错误宏定义：

HOME_ERROR_NONE	(0): 未发生错误
HOME_ERROR_NOT_TRAP_MODE	(1): 执行Smart Home回原点的 轴不是处于点位运动模式
HOME_ERROR_DISABLE	(2): 执行Smart Home回原点的 轴未使能
HOME_ERROR_ALARM	(3): 执行Smart Home回原点的 轴驱动报警
HOME_ERROR_STOP	(4): 未完成回原点，轴停止运

	HOME_ERROR_STAGE	动（例如搜索距离太短） (5): 回原点阶段错误
	HOME_ERROR_HOME_MODE	(6): 模式错误（例，轴已经启动Smart Home，再重复调用回原点指令，则报错）
	HOME_ERROR_SET_CAPTURE_HOME	(7): 设置Home捕获模式失败
	HOME_ERROR_NO_HOME	(8): 未找到Home
	HOME_ERROR_SET_CAPTURE_INDEX	(9): 设置Index捕获模式失败
	HOME_ERROR_NO_INDEX	(10): 位找到Index
指令返回值	请参照指令返回值列表。	
相关指令	无。	
指令示例		

指令 81 GT_GetJogPrm

指令原型	short GT_GetJogPrm(short profile, TJogPrm *pPrm)		
指令说明	读取 Jog 运动模式下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	69
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号。正整数。		
pPrm	设置 Jog 模式运动参数。该参数为一个结构体，包含三个参数，详细的参数定义及说明请参照 GT_SetJogPrm 指令说明。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为 Jog 模式，若不是，请先调用 GT_PrJog 将当前轴设置为 Jog 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetJogPrm		
指令示例	例程 6-2 Jog 运动		

指令 82 GT_GetMtrBias

指令原型	short GT_GetMtrBias(short dac, short *pBias)		
指令说明	读取模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
dac	模拟量输出通道号，正整数。		
pBias	读取的零漂补偿值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetMtrBias		
指令示例	无。		

指令 83 GT_GetMtrLmt

指令原型	short GT_GetMtrLmt(short dac, short *pLimit)		
指令说明	读取模拟量输出通道的输出电压饱和极限值。		

指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
dac	模拟量输出通道号。正整数。		
pLimit	读取的输出电压饱和极限值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetMtrLmt		
指令示例	无。		

指令 84 GT_GetPid

指令原型	short GT_GetPid(short control, short index, TPid *pPid)		
指令说明	读取 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	152
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
control	伺服控制器编号。正整数。		
index	伺服控制参数的索引号。取值范围：[1, 3]。		
pPid	读取 PID 参数。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetPid		
指令示例	例程 4-4 设置第 1 轴为闭环控制方式		

指令 85 GT_GetPos

指令原型	short GT_GetPos(short profile, long *pPos)		
指令说明	读取目标位置。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
profile	规划轴号。正整数。		
pos	读取目标位置，单位：pulse。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrflTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetPos		
指令示例	无。		

指令 86 GT_GetPosErr

指令原型	short GT_GetPosErr(short control, long *pError)		
指令说明	读取跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号。正整数。		
pError	读取的跟随误差极限值。单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetPosErr		
指令示例	无。		

指令 87 GT_GetPrfAcc

指令原型	short GT_GetPrfAcc(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号。正整数。		
pValue	规划加速度。单位：pulse/ms ² 。		
count	读取的轴数，默认为 1，正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度		

指令 88 GT_GetPrfMode

指令原型	short GT_GetPrfMode(short profile, long *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取轴运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号，正整数。		
pValue	轴运动模式。 0: 点位运动，控制器上电后默认为该模式； 1: Jog 模式； 2: PT 模式； 3: 电子齿轮模式； 4: Follow 模式； 5: 插补模式； 6: Pvt 模式。		
count	读取的轴数，默认为 1，正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度		

指令 89 GT_GetPrfPos

指令原型	short GT_GetPrfPos(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划位置。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号，正整数。		

pValue	规划位置。单位：pulse。
count	读取的轴数，默认为 1，正整数。
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度

指令 90 GT_GetPrfVel

指令原型	short GT_GetPrfVel(short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划速度。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	起始规划轴号，正整数。		
pValue	规划速度。单位：pulse/ms。		
count	读取的轴数，默认为 1，正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度		

指令 91 GT_GetRemainderSegNum

指令原型	short GT_GetRemainderSegNum(short crd, long *pSegment, short fifo=0)		
指令说明	读取未完成的插补段段数。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSegment	读取的剩余插补段的段数。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 92 GT_GetSoftLimit

指令原型	short GT_GetSoftLimit (short axis, long *pPositive, long *pNegative)		
指令说明	读取轴正向软限位和负向软限位。		
指令类型	立即指令，调用后立即生效。	章节页码	128
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号，正整数。		
pPositive	读取正向软限位。		
pNegative	读取负向软限位。		
指令返回值	请参照指令返回值列表。		

相关指令	GT_SetSoftLimit
指令示例	无。

指令 93 GT_GetStopDec

指令原型	short GT_GetStopDec(short profile, double *pDecSmoothStop, double *pDecAbruptStop)		
指令说明	读取平滑停止减速度和急停减速度。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划器的编号，正整数。		
pDecSmoothStop	读取的平滑停止减速度。单位：pulse/ms ² 。		
pDecAbruptStop	读取的急停减速度。单位：pulse/ms ² 。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetStopDec		
指令示例	无。		

指令 94 GT_GetSts

指令原型	short GT_GetSts(short axis, long *pSts, short count=1, unsigned long *pClock=NULL)		
指令说明	读取轴状态。		
指令类型	立即指令，调用后立即生效。	章节页码	58
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	起始轴号，正整数。		
pSts	32 位轴状态字，详细定义参见表 5-2 轴状态定义。		
count	读取的轴数，默认为 1，正整数。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_ClrSts		
指令示例	例程 5-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度		

指令 95 GT_GetThreadSts

指令原型	short GT_GetThreadSts(short thread, TThreadSts *pThreadSts)		
指令说明	读取线程的状态		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
thread	线程编号。取值范围：[0, 31]。		
pThreadSts	读取线程状态 typedef struct ThreadSts { short run; // 运行状态 short error; // 当前执行的指令返回值 double result; // 函数返回值 short line; // 当前执行的指令行号 } TThreadSts;		

指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	例程 10-1 运动程序单线程累加求和

指令 96 GT_GetTrapPrm

指令原型	short GT_GetTrapPrm(short profile, TTrapPrm *pPrm)		
指令说明	读取点位运动模式下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
pPrm	读取点位运动模式运动参数，该参数为一个 结构体 ，包含四个参数，详细的参数定义及说明请参照 GT_SetTrapPrm 指令说明。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrftTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetTrapPrm		
指令示例	例程 6-1 点位运动		

指令 97 GT_GetUserSegNum

指令原型	short GT_GetUserSegNum (short crd, long *pSegment, short fifo=0)		
指令说明	读取自定义插补段段号。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
pSegment	读取的用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetUserSegNum		
指令示例	无。		

指令 98 GT_GetVarId

指令原型	short GT_GetVarId(char *pFunName, char *pVarName, TVarInfo *pVarInfo)		
指令说明	读取运动程序中变量的标识。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
pFunName	全局变量输入 NULL。 局部变量所在函数的名称。		
pVarName	运动程序变量名称。		
pVarInfo	根据运动程序函数名称和变量名称查询变量标识。		
指令返回值	若返回值为 1，2007 或者 2008： 请检查重新检查 GT_Download 是否调用成功。 若失败，请根据 GT_Download 返回值提示操作，直至成功。		

其他返回值：请参照指令返回值列表。
相关指令
指令示例

指令 99 GT_GetVarValue

指令原型	short GT_GetVarValue(short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
指令说明	读取运动程序中变量的值。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
page	数据页编号。		
	全局变量为-1。		
	局部变量取值范围：[0, 31]。		
pVarInfo	需要访问的变量标识。		
pValue	需要读取的变量值。		
count	需要读取的变量值的数量，取值范围：[1, 8]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetVarValue		
指令示例	例程 10-1 运动程序单线程累加求和		

指令 100 GT_GetVel

指令原型	short GT_GetVel(short profile, double *pVel)		
指令说明	读取目标速度。		
指令类型	立即指令，调用后立即生效。	章节页码	65, 69
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
pVel	读取目标速度。单位：pulse/ms。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrTrp 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_SetVel		
指令示例	无。		

指令 101 GT_GetVersion

指令原型	short GT_GetVersion(char **pVersion)		
指令说明	读取运动控制器固件的版本号。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pVersion	读取的运动控制器的固件版本号字符串。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 11-1 读取运动控制器版本号		

指令 102 GT_GoHome

指令原型	short GT_GoHome(short axis, THomePrm *pHomePrm)		
指令说明	启动 Smart Home 实现各种方式回原点		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	进行回原点的轴号，对于 4 轴卡，取值范围：[1, 4]，对于 8 轴卡，取值范围[1,8]。		
pHomePrm	设置 Smart Home 回原点的参数，该参数为一结构体，详细参数定义及说明请参照结构体 THomePrm: <pre>typedef struct { short mode; // 回原点模式 short moveDir; // 设置启动搜索原点时的运动方向（如果回原点运动包含搜索 Limit 则为搜索 Limit 的运动方向）： // -1-负方向，1-正方向 short indexDir; // 设置搜索 Index 的运动方向：-1-负方向， // 1-正方向，在限位+Index 回原点模式下 moveDir 与 // indexDir 应该相异 short edge; // 设置捕获沿：0-下降沿，1-上升沿 short triggerIndex; // 默认与回零轴号一致，不需要设置 short pad1[3]; // 保留，其中 pad1[0]表示捕获到 Home 后运动到最终位置 // （捕获位置 + homeOffset）所使用的速度，0 或其它值- // 使用 velLow（默认），1-使用 velHigh。 double velHigh; // 回原点运动的高速速度（单位：pulse/ms） double velLow; // 回原点运动的低速速度（单位：pulse/ms） double acc; // 回原点运动的加速度（单位：pulse/ms^2） double dec; // 回原点运动的减速度（单位：pulse/ms^2） short smoothTime; // 回原点运动的平滑时间：取值[0,50]，单位：ms， // 具体含义与 GTS 系列控制器点位运动相似 short pad2[3]; // pad2[1]表示在电机启动回零时是否检测机械处于限位或 // 原点位置，0 或其它值-不检测（默认），1-检测。如果 // 不启用检测则当机械刚好处于限位或原点位置时将无法 // 回零；如果启用检测则当机械处于限位或原点时电机 // 会先按照 escapeStep 参数设置的回退距离进行回退， // 在根据具体回零模式进行回零，因此如果启用检测功能， // 那么 escapeStep 的值不能为 0，否则无法执行回退动作。 // 对于压到限位或原点的电平判断：控制器默认高电平触发， // 即限位或原点处于高电平状态则控制器认为当前压在 // 限位或原点上面，若用户的限位开关或原点开关不是此种模 // 式接法，可以通过控制器配置里面的 Di 选择项选择限位 // 或原点并设置其的“输入反转”为“取反”。 long homeOffset; // 最终停止的位置相对于原点的偏移量 long searchHomeDistance; // 设定的搜索 Home 的搜索范围，0 表示 // 搜索距离为 805306368 long searchIndexDistance; // 设定的搜索 Index 的搜索范围，0 表示 // 搜索距离为 805306368 long escapeStep; // 采用“限位回原点”方式时，反方向离开</pre>		

	限位的脱离步长 <pre>long pad3[2]; // 保留（不需要设置）</pre> <pre>} THomePrm;</pre> 回原点模式宏定义： HOME_MODE_LIMIT (10): 限位回原点 HOME_MODE_LIMIT_HOME (11): 限位+Home回原点 HOME_MODE_LIMIT_INDEX (12): 限位+Index回原点 HOME_MODE_LIMIT_HOME_INDEX (13): 限位+Home+Index回原点 HOME_MODE_HOME (20): Home回原点 HOME_MODE_HOME_INDEX (22): Home+Index回原点 HOME_MODE_INDEX (30): Index回原点 HOME_MODE_FORCED_HOME (40): 强制Home回原点 HOME_MODE_FORCED_HOME_INDEX (41): 强制Home+Index回原点
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	

指令 103 GT_GpiSns

指令原型	short GT_GpiSns(unsigned short sense)		
指令说明	设置运动控制器数字量输入的电平逻辑。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下：		
sense	按位表示各数量输入的电平逻辑，从 bit0~bit15，分别对应数字量输入 1 到 16。影响 GT_GetDi 指令读取电平的结果。 0: 输入电平不取反，通过 GT_GetDi 指令读取到 0 表示输入低电平，通过 GT_GetDi 指令读取到 1 表示输入高电平； 1: 输入电平取反，通过 GT_GetDi 指令读取到 0 表示输入高电平，通过 GT_GetDi 指令读取到 1 表示输入低电平；		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 104 GT_Index

指令原型	short GT_Index(short axis, long pos, long offset)		
指令说明	设置自动回原点功能为 home+index 模式。不能在运动程序运行中调用。		
指令类型	立即指令，调用后立即生效。	章节页码	117
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	需要进行原点急停操作的轴号，正整数。		
pos	搜索距离。以当前位置为起点，搜索距离为正时向正方向搜索，搜索距离为负时向负方向搜索。单位：pulse。		
offset	原点偏移量，单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 8-1 Home/Index 捕获		

指令 105 GT_InitLookAhead

指令原型	short GT_InitLookAhead (short crd, short fifo, double T, double accMax, short n, TCrdData *pLookAheadBuf)		
指令说明	初始化插补前瞻缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
T	拐弯时间。单位：ms。		
accMax	最大加速度。单位：pulse/ms ² 。		
n	前瞻缓存区大小。取值范围：[0, 32767]。		
pLookAheadBuf	前瞻缓存区内存区指针。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-8 前瞻预处理例程		

指令 106 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触发电平。																
指令类型	立即指令，调用后立即生效。											章节页码		51			
指令参数	该指令共有 1 个参数，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。参数的详细信息如下。																
sense	按位标识轴的限位的触发电平状态。下表中“+”代表正限位，“-”代表负限位。																
	对于 4 轴控制器，状态位与限位开关对应关系如下：																
	状态位	7	6	5	4	3	2	1	0								
	限 位	轴 4		轴 3		轴 2		轴 1									
	开关	-	+	-	+	-	+	-	+								
	对于 8 轴控制器，状态位与限位开关对应关系如下：																
	状态位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	限 位	轴 8		轴 7		轴 6		轴 5		轴 4		轴 3		轴 2		轴 1	
	开关	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+
指令返回值	请参照指令返回值列表。																
相关指令	无。																
指令示例	例程 4-2 修改限位开关触发电平																

指令 107 GT_LmtsOff

指令原型	short GT_LmtsOff(short axis, short limitType=-1)		
指令说明	控制轴限位信号无效。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	控制轴号，正整数。		

limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为0): 将该轴的正限位无效。 MC_LIMIT_NEGATIVE(该宏定义为1): 将该轴的负限位无效。 -1: 将该轴的正限位和负限位都无效, 默认为该值。
指令返回值	若返回值为 1: 请检查相应轴限位报警, 配置文件是否已经配置了限位无效。 其他返回值: 请参照指令返回值列表。
相关指令	GT_LmtsOn
指令示例	例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式

指令 108 GT_LmtsOn

指令原型	short GT_LmtsOn(short axis, short limitType=-1)		
指令说明	控制轴限位信号有效。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	控制轴号，正整数。		
limitType	需要有效的限位类型。		
	MC_LIMIT_POSITIVE(该宏定义为0)：将该轴的正限位有效。		
	MC_LIMIT_NEGATIVE(该宏定义为1)：将该轴的负限位有效。		
	-1：将该轴的正限位和负限位都有效，默认为该值。		
指令返回值	若返回值为 1：请检查相应轴限位报警，配置文件是否已经配置了限位无效。 其他返回值：请参照指令返回值列表。		
相关指令	GT_LmtsOff		
指令示例	无。		

指令 109 GT_LnXY

指令原型	short GT_LnXY (short crd, long x, long y, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面二维直线插补。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1：		
	(1) 检查当前坐标系是否映射了相关轴。		
	(2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。		
	(3) 检查相应的 fifo 是否已满。		
	其他返回值：请参照指令返回值列表。		

相关指令	无。
指令示例	例程 6-5 直线插补例程

指令 110 GT_LnXYG0

指令原型	short GT_LnXYG0 (short crd, long x, long y, double synVel, double synAcc, short fifo=0)		
指令说明	二维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 111 GT_LnXYZ

指令原型	short GT_LnXYZ (short crd, long x, long y, long z, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	三维直线插补。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。		

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

指令 112 GT_LnXYZA

指令原型	short GT_LnXYZA (short crd, long x, long y, long z, long a, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	四维直线插补。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
velEnd	插补段的终点速度。取值范围：[0, 32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： - 检查当前坐标系是否映射了相关轴。 - 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 - 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 113 GT_LnXYZAG0

指令原型	short GT_LnXYZAG0 (short crd, long x, long y, long z, long a, double synVel, double synAcc, short fifo=0)		
指令说明	四维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		

fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 114 GT_LnXYZG0

指令原型	short GT_LnXYZG0 (short crd, long x, long y, long z, double synVel, double synAcc, short fifo=0)		
指令说明	三维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 32767)，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767)，单位：pulse/ms ² 。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 115 GT_LoadConfig

指令原型	short GT_LoadConfig(char *pFile)		
指令说明	下载配置信息到运动控制器，调用该指令后需再调用GT_ClrSts才能使该指令生效。		
指令类型	立即指令，调用后立即生效。	章节页码	35
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
pFile	配置文件的文件名。文件名格式：*.cfg 或*.CFG。用户可根据自己的需求，使用运动控制器管理软件 MCT2008 生成此配置文件。		
指令返回值	若返回值为 1：请停止各轴规划运动后再设置。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-1 点位运动		

指令 116 GT_Open

指令原型	short GT_Open(short channel=0, short param=1)		
指令说明	打开运动控制器。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
channel	打开运动控制器的方式。默认值为：0。		
	0：正常打开运动控制器。 1：内部调试方式，用户不能使用。		
param	当 channel=1 时，该参数有效。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_Close		
指令示例	例程 11-1		

指令 117 GT_PauseThread

指令原型	short GT_PauseThread(short thread)		
指令说明	暂停正在运行的线程。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
thread	线程编号。取值范围：[0, 31]。		
指令返回值	若返回值为 1：		
	(1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。		
	其他返回值：请参照指令返回值列表。		
相关指令	GT_RunThread;GT_StopThread		
指令示例	无。		

指令 118 GT_Prfgear

指令原型	short GT_Prfgear (short profile, short dir)		
指令说明	设置指定轴为电子齿轮运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	65, 72
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
dir	设置跟随方式。 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随。		
指令返回值	若返回值为 1：		
	(1) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2) 当前已经是电子齿轮模式，但再次设置的 dir 与当前的 dir 不一致。		
	其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-3 电子齿轮跟随		

指令 119 GT_Prfgog

指令原型	short GT_PrJog(short profile)		
指令说明	设置指定轴为 Jog 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	65, 69
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-2 Jog 运动		

指令 120 GT_PrTrap

指令原型	short GT_PrTrap(short profile)		
指令说明	设置指定轴为点位运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 6-1 点位运动		

指令 121 GT_ProfileScale

指令原型	short GT_ProfileScale(short axis, short alpha, short beta)		
指令说明	设置控制轴的规划器当量变换值。注意：alpha 的绝对值要大于 beta 的绝对值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	控制轴号，正整数。		
alpha	规划器当量的 alpha 值，取值范围：(-32767, 0) 和 (0, 32767)。		
beta	规划器当量的 beta 值，取值范围：(-32767, 0) 和 (0, 32767)。		
指令返回值	若返回值为 1：若当前轴再规划运动，请调用 GT_Stop 停止运动在调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GT_EncScale		
指令示例	无。		

指令 122 GT_Reset

指令原型	short GT_Reset()		
指令说明	复位运动控制器。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令无输入参数。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式		

指令 123 GT_RunThread

指令原型	short GT_RunThread(short thread)		
指令说明	启动线程。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
thread	线程编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1： (1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。 其他返回值：请参照指令返回值列表。		
相关指令			
指令示例	例程 10-1 运动程序单线程累加求和		

指令 124 GT_SetAdcFilter

指令原型	short GT_SetAdcFilter(short adc, short filterTime)		
指令说明	设置模拟量输入的滤波器时间参数。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
adc	模拟量输入通道的编号，取值范围：[1, 8]。		
filterTime	模拟量输入信号的滤波器时间参数，取值范围：[0, 50]。无单位。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 125 GT_SetAxisBand

指令原型	short GT_SetAxisBand(short axis, long band, long time)		
指令说明	设置轴到位误差带。 规划器静止，规划位置 and 实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号，正整数。		
band	误差带大小。单位：脉冲。		
time	误差带保持时间。单位：250 微秒。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetAxisBand		
指令示例	例程 11-2 电机到位检测功能		

指令 126 GT_SetBacklash

指令原型	short GT_SetBacklash (short axis, long compValue, double compChangeValue, long compDir)		
指令说明	设置反向间隙补偿的相关参数。		

指令类型	立即指令，调用后立即生效。	章节页码	153
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	需要进行反向间隙补偿的轴的编号，正整数。		
compValue	反向间隙补偿值，当为 0 时表示没有使能反向间隙补偿功能。取值范围：[0, 1073741824]，单位：脉冲。		
compChangeValue	反向间隙补偿的变化量。取值范围：[0, 1073741824]，单位：pulse/ms。当该参数的值为 0 或者大于等于 compValue 时，则反向间隙的补偿量将瞬间叠加在规划位置上，没有渐变的过程。		
compDir	反向间隙补偿方向。 0：只补偿负方向，当电机向负方向运动时，将施加补偿量，当电机向正方向运动时，不施加补偿量。 1：只补偿正方向，当电机向正方向运动时，将施加补偿量，当电机向负方向运动时，不施加补偿量。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetBacklash		
指令示例	无。		

指令 127 GT_SetCaptureMode

指令原型	short GT_SetCaptureMode(short encoder, short mode)		
指令说明	设置编码器捕获方式，并启动捕获。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
encoder	编码器轴号，正整数。		
mode	编码器捕获模式 CAPTURE_HOME(该宏定义为 1) Home 捕获 CAPTURE_INDEX(该宏定义为 2) Index 捕获 CAPTURE_PROBE(该宏定义为 3) 探针捕获 CAPTURE_HSIO0(该宏定义为 6) HSIO0 口捕获 CAPTURE_HSIO1(该宏定义为 7) HSIO1 口捕获		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetCaptureMode		
指令示例	例程 8-1 Home/Index 捕获		

指令 128 GT_SetCaptureRepeat

指令原型	short GT_SetCaptureRepeat(short encoder, short count)		
指令说明	设置重复捕获。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
encoder	重复捕获的编码器轴号，正整数。		
count	最大捕获次数，取值范围：[1, 256]		
指令返回值	请参照指令返回值列表。		
相关指令	无。		

指令示例 例程 8-6 重复捕获使用说明

指令 129 GT_SetCaptureSense

指令原型	short GT_SetCaptureSense(short encoder, short mode, short sense)		
指令说明	设置捕获电平。		
指令类型	立即指令，调用后立即生效。	章节页码	112
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
encoder	编码器轴号，正整数。		
mode	编码器捕获模式 CAPTURE_HOME（该宏定义为 1）：Home 捕获。 CAPTURE_INDEX（该宏定义为 2）：Index 捕获。 CAPTURE_PROBE（该宏定义为 3）：探针捕获。 HSIO 捕获不能设置捕获电平。		
sense	捕获电平，可设置 0 或者 1。 0：下降沿触发。 1：上升沿触发。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 130 GT_SetCardNo

指令原型	short GT_SetCardNo(short index)		
指令说明	切换当前运动控制器卡号。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
index	将被设置为当前运动控制器的卡号。取值范围：[0, 15]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetCardNo		
指令示例	无。		

指令 131 GT_SetControlFilter

指令原型	short GT_SetControlFilter(short control, short index)		
指令说明	设定 PID 索引，支持 3 组 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	152
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
control	伺服控制器编号，正整数。		
index	伺服控制参数的索引号。取值范围：[1, 3]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetControlFilter		
指令示例	无。		

指令 132 GT_SetCrdBufferMode

指令原型	short GT_SetCrdBufferMode(short crd, short bufferMode, short fifo);
------	---

指令说明	设置插补缓冲区的模式（静态模式或动态模式）。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围： [1 , 2]。		
bufferMode	插补缓冲区模式： CRD_BUFFER_MODE_DYNAMIC_DEFAULT 0： 动态模式，插补运动完成后缓冲区内的数据会自动被清除。 CRD_BUFFER_MODE_STATIC_INPUT 11： 静态模式输入数据阶段。 CRD_BUFFER_MODE_STATIC_READY 12： 静态模式输入数据完成。 静态模式下，需要先调用该指令指示“静态模式数据输入阶段”，接着压入插补数据，完成后需要再次调用该指令指示“静态模式输入数据完成”，目前静态模式能够。插补运动完成控制器不会自动清除缓冲区数据，再次调用 GT_CrdStart 可以重新启动运动。 动态模式和静态模式之间可以相互切换。		
	fifo	插补缓存区号。取值范围： [0, 1]， 默认值为： 0。	
	返回值		
	相关指令	GT_GetCrdBufferMode	
	指令示例	例程 6-12 静态插补缓冲区模式	

指令 133 GT_SetCrdPrm

指令原型	short GT_SetCrdPrm(short crd, TCrdPrm *pCrdPrm)		
指令说明	设置坐标系参数，确立坐标系映射，建立坐标系。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号，取值范围： [1, 2]。		
pCrdPrm	设置坐标系的相关参数。 <pre>typedef struct CrdPrm { short dimension; short profile[8]; double synVelMax; double synAccMax; short evenTime; short setOriginFlag; long originPos[8]; }TCrdPrm;</pre> dimension: 坐标系的维数。取值范围： [1, 4]。 Profile[8]: 坐标系与规划器的映射关系。Profile[0..7]对应规划轴 1~8，如果规划轴没有对应到该坐标系，则 profile[x]的值为 0；如果对应到了 X 轴，则 profile[x]为 1，Y 轴对应为 2，Z 轴对应为 3，A 轴对应为 4。不允许多个规划轴映射到相同坐标系的相同坐标轴，也不允许把相同规划轴对应到不同的坐标系，否则该指令将会返回错误值。每个元素的取值范围： [0, 4]。		

指令返回值	synVelMax: 该坐标系的 <big>最大合成速度</big> 。如果用户在输入插补段的时候所设置的目标速度大于了该速度,则将会被限制为该速度。取值范围: (0, 32767)。单位: pulse/ms。
	synAccMax: 该坐标系的 <big>最大合成加速度</big> 。如果用户在输入插补段的时候所设置的加速度大于了该加速度,则将会被限制为该加速度。取值范围: (0, 32767)。单位: pulse/ms ² 。
	evenTime: 每个插补段的 <big>最小匀速段时间</big> 。取值范围: [0, 32767)。单位: ms。
	setOriginFlag: 表示是否需要指定坐标系的原点坐标的规划位置,该参数可以方便用户建立区别于机床坐标系的加工坐标系。0: 不需要指定原点坐标值,则坐标系的原点在当前规划位置上。1: 需要指定原点坐标值,坐标系的原点在 originPos 指定的规划位置上。
相关指令	originPos[8]: 指定的坐标系原点的规划位置值。
	若返回值为 1: (1) 若坐标系下各轴在规划运动,请调用 GT_Stop 停止运动再调用该指令。 (2) 请检查映射到 Profile 有没被激活,若无,则返回错误。 (3) 请见检查相应轴是否在坐标系下。
指令示例	其他返回值: 请参照指令返回值列表。
	GT_GetCrdPrm
指令示例	GT_GetCrdPrm
	例程 6-4 建立坐标系

指令 134 GT_SetCrdStopDec

指令原型	short GT_SetCrdStopDec (short crd, double decSmoothStop, double decAbruptStop)		
指令说明	设置插补运动平滑停止、急停合成加速度		
指令类型	立即指令,调用后立即生效。	章节页码	76
指令参数	该指令共有 3 个参数,参数的详细信息如下。		
crd	坐标系号。正整数,取值范围: [1, 2]。		
decSmoothStop	设置的坐标系合成平滑停止加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。		
decAbruptStop	设置的坐标系合成急停加速度。取值范围: (0, 32767), 单位: pulse/ms ² 。		
指令返回值	若返回值为 1: 检查当前坐标系是否映射了相关轴。 其他返回值: 请参照指令返回值列表。		
相关指令	GT_GetCrdStopDec		
指令示例	无。		

指令 135 GT_SetDac

指令原型	short GT_SetDac(short dac, short *pValue, short count)		
指令说明	设置 dac 输出电压。当闭环模式下, da 输出通道与轴挂接时,用户不能调用该指令直接输出电压。		
指令类型	立即指令,调用后立即生效。	章节页码	109
指令参数	该指令共有 3 个参数,参数的详细信息如下。		
dac	dac 起始轴号。		
pValue	输出电压。8 路轴控接口 -32768 对应-10V, 32767 对应+10V。 4 路非轴接口 0 对应 0V, 32767 对应+10V。		
count	设置的通道数。默认为 1。 1 次最多可以设置 8 路 dac 输出。		
指令返回值	请参照指令返回值列表。		

相关指令	GT_GetDac
指令示例	例程 7-3 访问 DAC

指令 136 GT_SetDiReverseCount

指令原型	short GT_SetDiReverseCount(short diType, short diIndex, unsigned long *pReverseCount, short count)		
指令说明	设置数字量输入信号的变化次数的初值。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
diIndex	数字量输入的索引。 取值范围： diType= MC_LIMIT_POSITIVE 时：[1, 8]。 diType= MC_LIMIT_NEGATIVE 时：[1, 8]。 diType= MC_ALARM 时：[1, 8]。 diType= MC_HOME 时：[1, 8]。 diType= MC_GPI 时：[1, 16]。 diType= MC_ARRIVE 时：[1, 8]。		
pReverseCount	设置的数字量输入的变化次数的初值。		
count	设置变化次数初值的数字量输入的个数。默认为 1。 1 次最多可以设置 4 个数字量输入的变化次数的初值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetDiReverseCount		
指令示例	例程 7-1 访问数字 IO		

指令 137 GT_SetDo

指令原型	short GT_SetDo(short doType, long value)		
指令说明	设置数字 IO 输出状态。若 do 有挂接轴，则对应的不能直接输出。默认驱动器使能与轴挂接，所以用户不能调用该指令设置驱动器使能输出的电平。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
value	按位指示数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		

相关指令	无。
指令示例	例程 7-1 访问数字 IO

指令 138 GT_SetDoBit

指令原型	short GT_SetDoBit(short doType, short doIndex, short value)		
指令说明	按位设置数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型		
	MC_ENABLE(该宏定义为 10)：驱动器使能。		
	MC_CLEAR(该宏定义为 11)：报警清除。		
	MC_GPO(该宏定义为 12)：通用输出。		
doIndex	输出 IO 的索引。		
	取值范围：		
	doType=MC_ENABLE 时：[1, 8]。		
	doType=MC_CLEAR 时：[1, 8]。		
value	doType=MC_GPO 时：[1, 16]。		
	设置数字 IO 输出电平。		
	默认情况下，1 表示高电平，0 表示低电平。		
	若返回值为 1：检查设置输出的 bit 是否挂接轴，若挂接，则不能直接输出。		
指令返回值	其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 7-1 访问数字 IO		

指令 139 GT_SetDoBitReverse

指令原型	short GT_SetDoBitReverse (short doType, short doIndex, short value, short reverseTime)		
指令说明	使数字量输出信号输出定时脉冲信号。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
doType	指定数字 IO 类型。		
	MC_ENABLE(该宏定义为 10)：驱动器使能。		
	MC_CLEAR(该宏定义为 11)：报警清除。		
	MC_GPO(该宏定义为 12)：通用输出。		
doIndex	输出 IO 的索引。		
	取值范围：		
	doType=MC_ENABLE 时：[1, 8]。		
	doType=MC_CLEAR 时：[1, 8]。		
value	doType=MC_GPO 时：[1, 16]。		
	设置数字 IO 输出电平。		
	默认情况下，1 表示高电平，0 表示低电平。		
	维持 value 所设置电平的时间，取值范围：(0, 32767]，单位：250μs。		
reverseTime	若返回值为 1：检查设置输出的 bit 是否挂接轴，若挂接，则不能直接输出。		
指令返回值	其他返回值：请参照指令返回值列表。		
相关指令	无。		

指令示例	无。
------	----

指令 140 GT_SetEncPos

指令原型	short GT_SetEncPos (short encoder, long encPos)		
指令说明	修改编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	108
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
encoder	编码器起始轴号，正整数。		
encPos	编码器位置。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 141 GT_SetGearMaster

指令原型	short GT_SetGearMaster(short profile, short masterIndex, short masterType, short masterItem)		
指令说明	设置电子齿轮运动跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	72
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
masterIndex	主轴索引，正整数。		
	主轴索引不能与规划轴号相同，最好主轴索引号小于规划轴号，如主轴索引为 1 轴，规划轴号为 2 轴。		
masterType	主轴类型。 GEAR_MASTER_PROFILE（该宏定义为 2）表示跟随规划轴(profile)的输出值。默认为该类型。 GEAR_MASTER_ENCODER（该宏定义为 1）表示跟随编码器(encoder)的输出值。 GEAR_MASTER_AXIS（该宏定义为 3）表示跟随轴(axis)的输出值。		
masterItem	轴类型，当 masterType=GEAR_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值。默认为该值。 1 表示 axis 的编码器位置输出值。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_PrflGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetGearMaster		
指令示例	例程 6-3 电子齿轮跟随		

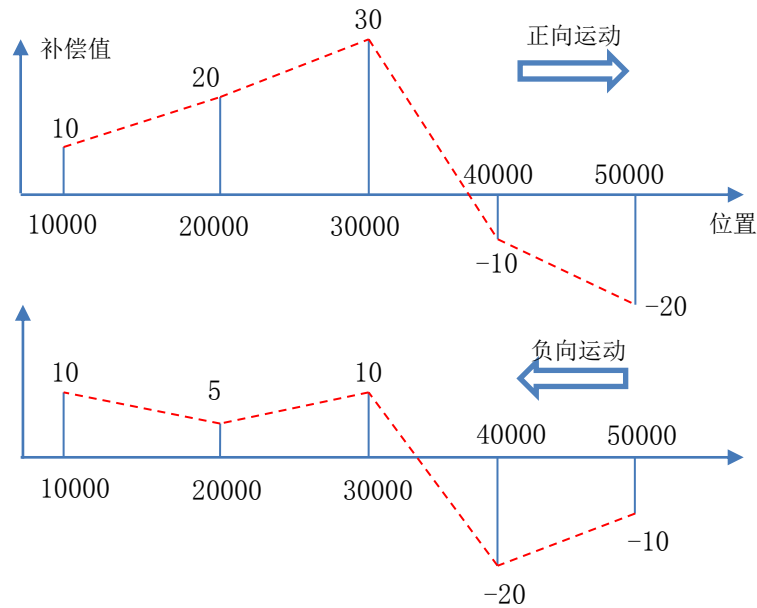
指令 142 GT_SetGearRatio

指令原型	short GT_SetGearRatio(short profile, long masterEven, long slaveEven, long masterSlope)
指令说明	设置电子齿轮比。

指令类型	立即指令，调用后立即生效。	章节页码	72
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
masterEven	传动比系数，主轴位移。 正整数，单位：pulse。		
slaveEven	传动比系数，从轴位移。 单位：pulse。		
masterSlope	主轴离合区位移。 单位：pulse。取值范围：不能小于 0 或者等于 1。		
指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GT_PrfrGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetGearRatio		
指令示例	例程 6-3 电子齿轮跟随		

指令 143 GT_SetLeadScrewComp

指令原型	short GT_SetLeadScrewComp(short axis,short n, long startPos, long lenPos, long *pCompPos, long *pCompNeg)		
指令说明	设置螺距误差补偿参数。		
指令类型	立即指令，调用后立即生效。	章节页码	69
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
axis	轴号，正整数。		
n	补偿点个数，取值范围：[1,256]。		
startPos	为补偿起始点的规划位置，单位：pulse。补偿起始点不能为负。		
lenPos	补偿区域总长，单位：pulse。补偿起始点和补偿总长一起构成了补偿区域，超出该规划位置范围，补偿无效。		
pCompPos	为对应为正方向运动时，各点位置需要补偿的脉冲数。		
pCompNeg	为对应为负方向运动时，各点位置需要补偿的脉冲数。		
指令返回值			
相关指令	GT_EnableLeadScrewComp		
指令示例	如下图所示，补偿起始位置为 10000，补偿总长为 40000，设置补偿节点数为 5，因此补偿节点位置为{10000,20000,30000,40000,50000}，正向运动时从 10000 开始运动 40000 个脉冲，测量到达上述补偿节点位置时，规划位置 and 实际位置的差值依次为 10->20->30->(-10)->(-20)；负向运动时从 50000 开始运动 40000 个脉冲，测量达到上述补偿节点位置时，规划位置 and 实际位置的差值依次为(-10)->(-20)->10->5->10。则{10,20,30,-10,-20}作为正向补偿值，{10,5,10,-20,-10}作为负向补偿值（注意，负向测量和正向测量不同，负向测量的第一个值是负向补偿数组的最后一个元素）。 （注意：每段的长度必须保持等长进行测量，测量点之间的补偿值是两个测量点补偿值的线性插值计算出来的补偿值）		



```
long comPos[] = {10,20,30,-10,-20};
long comNeg[] = {10,5,10,-20,-10};
rtn=GT_SetLeadScrewComp(1,5,10000,40000,comPos,comNeg);
rtn=GT_EnableLeadScrewComp(1,1);
```

指令 144 GT_SetJogPrm

指令原型

short GT_SetJogPrm(short profile, TJogPrm *pPrm)

指令说明

设置 Jog 运动模式下的运动参数。

指令类型

立即指令，调用后立即生效。

章节页码

69

指令参数

该指令共有 2 个参数，参数的详细信息如下。

profile

规划轴号，正整数。

pPrm

设置 Jog 模式运动参数。该参数为一个结构体，包含三个参数，详细的参数定义及说明如下：

typedef struct JogPrm

```
{
    double acc;
    double dec;
    double smooth;
} TJogPrm;
```

acc: 点位运动的加速度。正数，单位：pulse/ms²。**dec:** 点位运动的减速度。正数，单位：pulse/ms²。未设置减速度时，默认减速度和加速度相同。**smooth:** 平滑系数。取值范围：[0, 1)。平滑系数的数值越大，加减速过程越平稳。

指令返回值

若返回值为 1：

- (1) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。
- (2) 请检查当前轴是否为 Jog 模式，若不是，请先调用 GT_PrflJog 将当前轴设置为 Jog 模式。

其他返回值：请参照指令返回值列表。

相关指令	GT_GetJogPrm
指令示例	例程 6-2 Jog 运动

指令 145 GT_SetMtrBias

指令原型	short GT_SetMtrBias(short dac, short bias)		
指令说明	设置模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
dac	模拟量输出通道号，正整数。		
bias	零漂补偿值，取值范围：[-32768, 32767]。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetMtrBias		
指令示例	无。		

指令 146 GT_SetMtrLmt

指令原型	short GT_SetMtrLmt(short dac, short limit)		
指令说明	设置模拟量输出通道的输出电压饱和极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
dac	模拟量输出通道号，正整数。		
limit	输出电压饱和极限值，取值范围：(0, 32767]。若设置为 32767，则限制允许输出的电压范围为：-10V~+10V；若设置为 16384，则限制允许输出的电压范围为：-5V~+5V。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetMtrLmt		
指令示例	无。		

指令 147 GT_SetOverride

指令原型	short GT_SetOverride (short crd, double synVelRatio)		
指令说明	设置插补运动目标合成速度倍率。		
指令类型	立即指令，调用后立即生效。	章节页码	76
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
synVelRatio	设置的插补目标速度倍率，取值范围：(0, 1]，系统默认该值为：1。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 148 GT_SetPid

指令原型	short GT_SetPid(short control, short index, TPid *pPid)		
指令说明	设置 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	152

指令参数	该指令共有 3 个参数，参数的详细信息如下。
control	伺服控制器编号，正整数。
index	伺服控制参数的索引号，取值范围：[1, 3]。
pPid	设置 PID 参数 <pre>typedef struct Pid { double kp; double ki; double kd; double kvff; double kaff; long integralLimit; long derivativeLimit; short limit; }TPid;</pre> kp: 比例增益; ki: 积分增益; kd: 微分增益; kvff: 速度前馈系数; kaff: 加速度前馈系数; integralLimit: 积分饱和极限; derivativeLimit: 微分饱和极限; limit: 控制量输出饱和极限;
指令返回值	请参照指令返回值列表。
相关指令	GT_GetPid
指令示例	例程 4-4 设置第 1 轴为闭环控制方式

指令 149 GT_SetPos

指令原型	short GT_SetPos(short profile, long pos)		
指令说明	设置目标位置。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
pos	设置目标位置，单位：pulse。取值范围：[-1073741823, 1073741823]。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrflTrap 将当前轴设置为 Trap 模式。		
	其他返回值：请参照指令返回值列表。		
相关指令	GT_GetPos		
指令示例	例程 6-1 点位运动		

指令 150 GT_SetPosErr

指令原型	short GT_SetPosErr(short control, long error)		
指令说明	设置跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	51

指令参数	该指令共有 2 个参数，参数的详细信息如下。
control	伺服控制器编号，正整数。
error	跟随误差极限值，取值范围：(0, 2147483648)。单位：pulse。
指令返回值	请参照指令返回值列表。
相关指令	GT_GetPosErr
指令示例	无。

指令 151 GT_SetPrfPos

指令原型	short GT_SetPrfPos(short profile, long prfPos)		
指令说明	修改指定轴的规划位置。禁止在运动状态下修改规划位置。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
profile	规划轴编号，正整数。		
prfPos	设置的规划位置的值。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetPrfPos		
指令示例	例程 6-1 点位运动		

指令 152 GT_SetSoftLimit

指令原型	short GT_SetSoftLimit (short axis, long positive, long negative)		
指令说明	设置轴正向软限位和负向软限位。		
指令类型	立即指令，调用后立即生效。	章节页码	128
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
axis	轴号，正整数。		
positive	正向软限位，当规划位置大于该值时，正限位触发。 默认值为：0x7fffffff，表示正向软限位无效。		
negative	负向软限位，当规划位置小于该值时，负限位触发。 默认值为：0x80000000，表示负向软限位无效。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_GetSoftLimit		
指令示例	例程 9-1 软限位使用		

指令 153 GT_SetStopDec

指令原型	short GT_SetStopDec(short profile, double decSmoothStop, double decAbruptStop)		
指令说明	设置平滑停止减速度和急停减速度。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令类型	立即指令，调用后立即生效。		
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
profile	规划器的编号，正整数。		
decSmoothStop	平滑停止减速度。单位：pulse/ms ² 。		
decAbruptStop	急停减速度。单位：pulse/ms ² 。		
指令返回值	请参照指令返回值列表。		

相关指令	GT_GetStopDec
指令示例	无。

指令 154 GT_SetStopIo

指令原型	short GT_SetStopIo(short axis, short stopType, short inputType, short inputIndex)		
指令说明	设置平滑停止和紧急停止数字量输入的信息。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
axis	需要设置停止IO信息的轴的编号，正整数。		
stopType	需要设置停止 IO 信息的停止类型。 0: 紧急停止类型。 1: 平滑停止类型。		
inputType	设置的数字量输入的类型。 MC_LIMIT_POSITIVE(该宏定义为 0)，正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)，负限位。 MC_ALARM(该宏定义为 2)，驱动报警。 MC_HOME(该宏定义为 3)，原点开关。 MC_GPI(该宏定义为 4)，通用输入。 MC_ARRIVE(该宏定义为 5)，电机到位信号。		
inputIndex	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定。 对于 8 轴控制器： 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1, 8]； 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1, 8]； 当 inputType= MC_ALARM 时，取值范围：[1, 8]； 当 inputType= MC_HOME 时，取值范围：[1, 8]； 当 inputType= MC_GPI 时，取值范围：[1, 16]。 对于 4 轴控制器： 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1, 4]； 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1, 4]； 当 inputType= MC_ALARM 时，取值范围：[1, 4]； 当 inputType= MC_HOME 时，取值范围：[1, 4]； 当 inputType= MC_GPI 时，取值范围：[1, 16]； 当 inputType= MC_ARRIVE 时，取值范围：[1, 4]。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 155 GT_SetTrapPrm

指令原型	short GT_SetTrapPrm(short profile, TTrapPrm *pPrm)		
指令说明	设置点位模式运动下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		

pPrm	设置点位运动模式运动参数，该参数为一个结构体，包含四个参数，详细的参数定义及说明如下： <pre>typedef struct TrapPrm { double acc; double dec; double velStart; short smoothTime; } TTrapPrm;</pre> acc: 点位运动的加速度。正数，单位：pulse/ms²。 dec: 点位运动的减速度。正数，单位：pulse/ms²。未设置减速度时，默认减速度和加速度相同。 velStart: 起跳速度。正数，单位：pulse/ms。默认值为 0。 smoothTime: 平滑时间。正整数，取值范围：[0, 50]，单位 ms。平滑时间的数值越大，加减速过程越平稳。
	若返回值为 1： (1) 若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrftTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。
	GT_GetTrapPrm
	例程 6-1 点位运动

指令 156 GT_SetUserSegNum

指令原型	short GT_SetUserSegNum(short crd, long segNum, short fifo=0)		
指令说明	设置自定义插补段段号。		
指令类型	缓存区指令。	章节页码	76
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
crd	坐标系号。正整数，取值范围：[1, 2]。		
segNum	设置用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GT_GetUserSegNum		
指令示例	无。		

指令 157 GT_SetVarValue

指令原型	short GT_SetVarValue (short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
指令说明	设置运动程序中变量的值。		
指令类型	立即指令，调用后立即生效。	章节页码	134

指令参数	该指令共有 4 个参数，参数的详细信息如下。
page	数据页编号。 全局变量为-1。 局部变量取值范围：[0, 31]。
pVarInfo	需要访问的变量标识。
pValue	需要写入的变量值。
count	需要写入的变量值的数量，取值范围：[1, 8]。
指令返回值	请参照指令返回值列表。
相关指令	GT_GetVarValue
指令示例	例程 10-1 运动程序单线程累加求和

指令 158 GT_SetVel

指令原型	short GT_SetVel(short profile, double vel)		
指令说明	设置目标速度。		
指令类型	立即指令，调用后立即生效。	章节页码	65, 69
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
profile	规划轴号，正整数。		
vel	设置目标速度。单位：pulse/ms。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GT_PrTrp 将当前轴设置为 Trap 模式。		
	其他返回值：请参照指令返回值列表。		
相关指令	GT_GetVel		
指令示例	例程 6-1 点位运动		

指令 159 GT_StepDir

指令原型	short GT_StepDir(short step)		
指令说明	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
step	脉冲输出通道号，正整数。		
指令返回值	若返回值为 1： (1)若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令；。 (2)若当前轴伺服使能，请调用 GT_AxisOff 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 4-3 设置第 1 轴为脉冲控制“脉冲+方向”方式		

指令 160 GT_StepPulse

指令原型	short GT_StepPulse(short step)		
指令说明	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
step	脉冲输出通道号，正整数。		

指令返回值	若返回值为 1： (1)若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2)若当前轴伺服使能，请调用 GT_AxisOff 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 161 GT_StepOrthogonal

指令原型	short GT_StepOrthogonal(short step)		
指令说明	将脉冲输出通道的脉冲输出模式设置为正交脉冲。		
指令类型	立即指令，调用后立即生效。	章节页码	51
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
step	脉冲输出通道号，正整数。		
指令返回值	若返回值为 1： (1)若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 (2)若当前轴伺服使能，请调用 GT_AxisOff 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 162 GT_Stop

指令原型	short GT_Stop(long mask, long option)										
指令说明	停止一个或多个轴的规划运动，停止坐标系运动。										
指令类型	立即指令，调用后立即生效。								章节页码	58	
指令参数	该指令共有 2 个参数，参数的详细信息如下。										
mask	按位指示需要停止运动的轴号或者坐标系号。当 bit 位为 1 时表示停止对应的轴或者坐标系。										
	对于 4 轴控制器：										
	Bit	9	8	7	6	5	4	3	2	1	0
	对应轴或坐标系	坐标系 2	坐标系 1	/	/	/	/	4 轴	3 轴	2 轴	1 轴
	对于 8 轴控制器：										
	Bit	9	8	7	6	5	4	3	2	1	0
option	对应轴或坐标系	坐标系 2	坐标系 1	8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
	按位指示停止方式。当 bit 位为 0 时表示平滑停止对应的轴或坐标系，当 bit 位为 1 时表示急停对应的轴或坐标系。										
	对于 4 轴控制器：										
	Bit	9	8	7	6	5	4	3	2	1	0
	对应轴或坐标系	坐标系 2	坐标系 1	/	/	/	/	4 轴	3 轴	2 轴	1 轴
	对于 8 轴控制器：										
指令返回值	Bit	9	8	7	6	5	4	3	2	1	0
	对应轴或坐标系	坐标系 2	坐标系 1	8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
	请参照指令返回值列表。										

相关指令	无。
指令示例	无。

指令 163 GT_StopCapture

指令原型	short GT_StopCapture (short encoder)		
指令说明	停止捕获。需要调用 GT_SetCaptureMode 才能重新启动捕获。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
encoder	编码器轴号，正整数。		
指令返回值	请参照指令返回值列表。		
相关指令	GT_SetCaptureMode		
指令示例	无。		

指令 164 GT_StopThread

指令原型	short GT_StopThread(short thread)		
指令说明	停止正在运行的线程。		
指令类型	立即指令，调用后立即生效。	章节页码	134
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
thread	线程编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1： (1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。 其他返回值：请参照指令返回值列表。		
相关指令	GT_RunThread ; GT_PauseThread		
指令示例	无。		

指令 165 GT_SynchAxisPos

指令原型	short GT_SynchAxisPos(long mask)									
指令说明	axis 合成规划位置和所关联的 profile 同步。 axis 合成编码器位置和所关联的 encoder 同步。									
指令类型	立即指令，调用后立即生效。							章节页码	147	
指令参数	该指令共有 1 个参数，参数的详细信息如下。									
mask	按位标识需要进行位置同步的轴号。0：表示不需要进行位置同步，1：需要进行位置同步。									
	对于 4 轴控制器：									
	Bit	3	2	1	0					
	对应轴	4 轴	3 轴	2 轴	1 轴					
	对于 8 轴控制器：									
	Bit	7	6	5	4	3	2	1	0	
	对应轴	8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴	
指令返回值	若返回值为 1：请检查参数 mask 是否设置为 0。 其他返回值：请参照指令返回值列表。									
相关指令	无。									

指令示例 无。

指令 166 GT_Update

指令原型	short GT_Update(long mask)									
指令说明	启动点位运动或 Jog 运动。									
指令类型	立即指令，调用后立即生效。							章节页码	65, 69	
指令参数	该指令共有 1 个参数，参数的详细信息如下。									
mask	按位指示需要启动点位运动或 Jog 运动的轴号。当 bit 位为 1 时表示启动对应的轴。									
	对于 4 轴控制器：									
	Bit	3	2	1	0					
	对应轴	4 轴	3 轴	2 轴	1 轴					
	对于 8 轴控制器：									
	Bit	7	6	5	4	3	2	1	0	
	对应轴	8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴	
指令返回值	请参照指令返回值列表。									
相关指令	无。									
指令示例	例程 6-1 点位运动									

指令 167 GT_ZeroPos

指令原型	short GT_ZeroPos(short axis, short count)		
指令说明	清零规划位置 and 实际位置，并进行零漂补偿。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
axis	需要位置清零的起始轴号，正整数。		
count	需要位置清零的轴数。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GT_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	例程 8-2 Home 回原点		

第14章 索引

14.1 指令索引

指令 1	GT_AlarmOff.....	163
指令 2	GT_AlarmOn.....	163
指令 3	GT_ArcXYC.....	163
指令 4	GT_ArcXYR.....	164
指令 5	GT_ArcYZC.....	165
指令 6	GT_ArcYZR.....	165
指令 7	GT_ArcZXC.....	166
指令 8	GT_ArcZXR.....	166
指令 9	GT_ArcXYZ.....	167
指令 10	GT_AxisOff.....	168
指令 11	GT_AxisOn.....	168
指令 12	GT_Bind.....	169
指令 13	GT_BufDA.....	169
指令 14	GT_BufDelay.....	169
指令 15	GT_BufGear.....	170
指令 16	GT_BufIO.....	170
指令 17	GT_BufLmtsOff.....	171
指令 18	GT_BufLmtsOn.....	171
指令 19	GT_BufMove.....	172
指令 20	GT_BufSetStopIo.....	172
指令 21	GT_ClearCaptureStatus.....	173
指令 22	GT_Close.....	173
指令 23	GT_ClrSts.....	173
指令 24	GT_CompareContinuePulseMode.....	174
指令 25	GT_CompareData.....	174
指令 26	GT_CompareLinear.....	175
指令 27	GT_ComparePulse.....	176
指令 28	GT_CompareStatus.....	176
指令 29	GT_CompareStop.....	177
指令 30	GT_CrdClear.....	177
指令 31	GT_CrdData.....	177
指令 32	GT_CrdSpace.....	178
指令 33	GT_CrdStart.....	178
指令 34	GT_CrdStatus.....	178
指令 35	GT_CtrlMode.....	179
指令 36	GT_Download.....	179
指令 37	GT_EnableLeadScrewComp.....	179
指令 38	GT_EncOff.....	180
指令 39	GT_EncOn.....	180

指令 40	GT_EncScale	180
指令 41	GT_EncSns	181
指令 42	GT_GearStart	181
指令 43	GT_GetAdc	181
指令 44	GT_GetAdcValue	182
指令 45	GT_GetAxisBand	182
指令 46	GT_GetAxisEncAcc	182
指令 47	GT_GetAxisEncPos	183
指令 48	GT_GetAxisEncVel	183
指令 49	GT_GetAxisError	183
指令 50	GT_GetAxisPrfAcc	184
指令 51	GT_GetAxisPrfPos	184
指令 52	GT_GetAxisPrfVel	184
指令 53	GT_GetBacklash	185
指令 54	GT_GetCaptureRepeatPos	185
指令 55	GT_GetCaptureRepeatStatus	185
指令 56	GT_GetCaptureMode	186
指令 57	GT_GetCaptureStatus	186
指令 58	GT_GetCardNo	186
指令 59	GT_GetClock	187
指令 60	GT_GetClockHighPrecision	187
指令 61	GT_GetControlFilter	187
指令 62	GT_GetCrdBufferMode	187
指令 63	GT_GetCrdPos	188
指令 64	GT_GetCrdPrm	188
指令 65	GT_GetCrdSegmentTime	188
指令 66	GT_GetCrdStopDec	189
指令 67	GT_GetCrdTime	189
指令 68	GT_GetCrdVel	190
指令 69	GT_GetDac	190
指令 70	GT_GetDi	190
指令 71	GT_GetDiRaw	191
指令 72	GT_GetDiReverseCount	191
指令 73	GT_GetDo	192
指令 74	GT_GetEncPos	192
指令 75	GT_GetEncVel	192
指令 76	GT_GetFunId	193
指令 77	GT_GetGearMaster	193
指令 78	GT_GetGearRatio	194
指令 79	GT_GetHomePrm	194
指令 80	GT_GetHomeStatus	194
指令 81	GT_GetJogPrm	196
指令 82	GT_GetMtrBias	196
指令 83	GT_GetMtrLmt	196
指令 84	GT_GetPid	197
指令 85	GT_GetPos	197

指令 86	GT_GetPosErr.....	197
指令 87	GT_GetPrfAcc.....	198
指令 88	GT_GetPrfMode.....	198
指令 89	GT_GetPrfPos.....	198
指令 90	GT_GetPrfVel.....	199
指令 91	GT_GetRemainderSegNum.....	199
指令 92	GT_GetSoftLimit.....	199
指令 93	GT_GetStopDec.....	200
指令 94	GT_GetSts.....	200
指令 95	GT_GetThreadSts.....	200
指令 96	GT_GetTrapPrm.....	201
指令 97	GT_GetUserSegNum.....	201
指令 98	GT_GetVarId.....	201
指令 99	GT_GetVarValue.....	202
指令 100	GT_GetVel.....	202
指令 101	GT_GetVersion.....	202
指令 102	GT_GoHome.....	202
指令 103	GT_GpiSns.....	204
指令 104	GT_Index.....	204
指令 105	GT_InitLookAhead.....	205
指令 106	GT_LmtSns.....	205
指令 107	GT_LmtsOff.....	205
指令 108	GT_LmtsOn.....	206
指令 109	GT_LnXY.....	206
指令 110	GT_LnXYG0.....	207
指令 111	GT_LnXYZ.....	207
指令 112	GT_LnXYZA.....	208
指令 113	GT_LnXYZAG0.....	208
指令 114	GT_LnXYZG0.....	209
指令 115	GT_LoadConfig.....	209
指令 116	GT_Open.....	210
指令 117	GT_PauseThread.....	210
指令 118	GT_Prfgear.....	210
指令 119	GT_Prfgog.....	210
指令 120	GT_Prftap.....	211
指令 121	GT_ProfileScale.....	211
指令 122	GT_Reset.....	211
指令 123	GT_RunThread.....	212
指令 124	GT_SetAdcFilter.....	212
指令 125	GT_SetAxisBand.....	212
指令 126	GT_SetBacklash.....	212
指令 127	GT_SetCaptureMode.....	213
指令 128	GT_SetCaptureRepeat.....	213
指令 129	GT_SetCaptureSense.....	214
指令 130	GT_SetCardNo.....	214
指令 131	GT_SetControlFilter.....	214

指令 132	GT_SetCrdBufferMode.....	214
指令 133	GT_SetCrdPrm	215
指令 134	GT_SetCrdStopDec	216
指令 135	GT_SetDac.....	216
指令 136	GT_SetDiReverseCount.....	217
指令 137	GT_SetDo	217
指令 138	GT_SetDoBit	218
指令 139	GT_SetDoBitReverse.....	218
指令 140	GT_SetEncPos	219
指令 141	GT_SetGearMaster	219
指令 142	GT_SetGearRatio.....	219
指令 143	GT_SetLeadScrewComp	220
指令 144	GT_SetJogPrm.....	221
指令 145	GT_SetMtrBias	222
指令 146	GT_SetMtrLmt	222
指令 147	GT_SetOverride.....	222
指令 148	GT_SetPid.....	222
指令 149	GT_SetPos	223
指令 150	GT_SetPosErr	223
指令 151	GT_SetPrfPos	224
指令 152	GT_SetSoftLimit.....	224
指令 153	GT_SetStopDec	224
指令 154	GT_SetStopIo.....	225
指令 155	GT_SetTrapPrm.....	225
指令 156	GT_SetUserSegNum.....	226
指令 157	GT_SetVarValue	226
指令 158	GT_SetVel.....	227
指令 159	GT_StepDir.....	227
指令 160	GT_StepPulse.....	227
指令 161	GT_StepOrthogonal	228
指令 162	GT_Stop.....	228
指令 163	GT_StopCapture	229
指令 164	GT_StopThread.....	229
指令 165	GT_SynchAxisPos	229
指令 166	GT_Update.....	230
指令 167	GT_ZeroPos.....	230

14.2 例程索引

例程 3-1	检测 GT 指令是否正常执行	17
例程 4-1	修改编码器计数方向	52
例程 4-2	修改限位开关触发电平	53
例程 4-3	设置第 1 轴为脉冲控制“脉冲+方向”方式.....	54
例程 4-4	设置第 1 轴为闭环控制方式	55
例程 5-1	获取轴 1 的轴状态、运动模式、位置、速度和加速度.....	60

例程 6-1	点位运动	66
例程 6-2	Jog 运动	69
例程 6-3	电子齿轮跟随	74
例程 6-4	建立坐标系	79
例程 6-5	直线插补例程	80
例程 6-6	圆弧插补例程	84
例程 6-7	插补 FIFO 管理	87
例程 6-8	前瞻预处理例程	92
例程 6-9	刀向跟随功能 GT_BufMove	95
例程 6-10	刀向跟随功能 GT_BufGear	98
例程 6-11	刀向跟随功能——实际工件加工	101
例程 7-1	访问数字 IO	106
例程 7-2	读取 8 个轴编码器和辅助编码器位置值	109
例程 7-3	访问 DAC	110
例程 7-4	访问 ADC	111
例程 8-1	Home/Index 捕获	113
例程 8-2	Home 回原点	115
例程 8-3	Home+Index 回原点	120
例程 8-4	探针捕获	124
例程 8-5	HSIO 捕获用法示例	126
例程 8-6	重复捕获使用说明	127
例程 9-1	软限位使用	129
例程 10-1	运动程序单线程累加求和	136
例程 10-2	运动程序多线程累加求和	138
例程 11-1	读取运动控制器版本号	146
例程 11-2	电机到位检测功能	148
例程 11-3	位置比较输出指令详细的用法	156

14.3 表格索引

表 1-1	指令列表	8
表 3-1	运动控制器指令返回值定义	17
表 4-1	下载配置文件指令	34
表 4-2	配置信息修改指令列表	50
表 4-3	GT_EncSns 指令参数状态位定义	51
表 4-4	例程 4-1 中的编码器计数方向设定表	51
表 4-5	GT_LmtSns 指令参数状态位定义	52
表 4-6	例程 3-2 限位开关设定表	52
表 4-7	控制器配置初始化状态	55
表 5-1	运动状态检测指令列表	57
表 5-2	轴状态定义	58
表 6-1	设置运动模式指令列表	63
表 6-2	点位运动模式指令列表	63
表 6-3	Jog 运动模式指令列表	66
表 6-4	电子齿轮运动模式指令列表	70
表 6-5	插补运动模式指令列表	74

表 7-1 运动控制器硬件资源	103
表 7-2 访问数字 IO 指令列表	103
表 7-3 访问编码器指令列表	107
表 7-4 访问 DAC 指令列表	109
表 7-5 轴控模拟电压值与指令读取数值对应关系	109
表 7-6 非轴模拟电压值与指令读取数值对应关系	109
表 7-7 访问模拟量输入指令列表	110
表 7-8 模拟电压值与指令读取数值对应关系	110
表 8-1 高速硬件捕获指令列表	111
表 9-1 软限位指令列表	127
表 10-1 运动程序指令列表	133
表 10-2 可在运动程序中使用的指令其它指令	142
表 10-3 打开/关闭运动控制器指令列表	144
表 10-4 读取固件版本号指令列表	144
表 10-5 固件版本号的定义格式	145
表 10-6 读取系统时钟指令列表	145
表 10-7 打开/关闭电机使能信号指令列表	145
表 10-8 维护位置值指令列表	145
表 10-9 电机到位检测指令列表	146
表 10-10 设置 PID 参数指令列表	150
表 10-11 反向间隙补偿指令列表	152
表 10-12 螺距误差补偿指令	152
表 10-13 位置比较输出指令列表	153
表 11-1 Smart Home 指令列表	157

14.4 图片索引

图 4-1 开环运动控制系统的配置	20
图 4-2 闭环运动控制系统的配置	21
图 4-3 MCT2008 运动控制器管理软件界面	22
图 4-4 打开控制器配置	22
图 4-5 axis 配置界面	23
图 4-6 axis 配置对控制系统的影响	23
图 4-7 step 配置界面	26
图 4-8 dac 配置界面	27
图 4-9 encoder 配置界面	28
图 4-10 encoder 配置对控制系统的影响	29
图 4-11 encoder 输入脉冲反转项的影响	29
图 4-12 control 配置界面	30
图 4-13 跟随误差解释图	30
图 4-14 profile 配置界面	31
图 4-15 di 配置界面	32
图 4-16 do 配置界面	33
图 4-17 生成配置文件界面	35
图 4-18 开环模式下 axis 配置流程	36

图 4-19 开环模式下 step 配置流程.....	36
图 4-20 开环模式下 profile 配置流程.....	36
图 4-21 开环模式下 do 配置流程	37
图 4-22 开环配置示例之 axis 配置界面	38
图 4-23 开环配置示例之 encoder 配置界面.....	39
图 4-24 开环配置示例之 di 配置界面 1	40
图 4-25 开环配置示例之 di 配置界面 2	40
图 4-26 写入控制器状态	41
图 4-27 点击进入轴状态界面	42
图 4-28 轴状态界面	42
图 4-29 Jog 模块界面.....	43
图 4-30 闭环模式下 axis 配置流程.....	44
图 4-31 闭环模式下 dac 配置流程.....	44
图 4-32 闭环模式下 encoder 配置流程.....	44
图 4-33 闭环模式下 control 配置流程.....	45
图 4-34 闭环模式下 profile 配置流程.....	45
图 4-35 闭环模式下 do 配置流程	46
图 4-36 闭环配置示例之 dac 配置界面.....	47
图 4-37 闭环配置示例之 encoder 配置界面.....	48
图 4-38 闭环配置示例之 control 配置界面.....	49
图 4-39 PID 设置界面.....	50
图 6-1 点位运动速度曲线	64
图 6-2 点位运动速度规划	64
图 6-3 Jog 模式速度曲线.....	67
图 6-4 Jog 模式动态改变目标速度.....	67
图 6-5 电子齿轮模式速度曲线	71
图 6-6 电子齿轮模式主轴速度规划	72
图 6-7 电子齿轮模式从轴速度规划	72
图 6-8 直线插补示意图	76
图 6-9 圆弧插补示意图	76
图 6-10 加工坐标系偏移量示意图	77
图 6-11 不同 evenTime 下的速度曲线.....	79
图 6-12 直线插补例程运动轨迹	79
图 6-13 圆弧插补逆时针方向	81
图 6-14 半径取正值/负值圆弧插补示意图	82
图 6-15 圆心坐标描述方法示意图	82
图 6-16 圆弧插补例程运动轨迹	83
图 6-17 插补主运动与辅助运动流程	86
图 6-18 插补 FIFO 管理例程之换刀轨迹.....	86
图 6-19 使用前瞻与不使用前瞻的速度规划区别	89
图 6-20 使用和不使用前瞻预处理功能模块的速度曲线对比图.....	90
图 6-21 前瞻预处理流程图	90
图 6-22 前瞻预处理例程之运动轨迹图	91
图 6-23 没有进行前瞻预处理的合成速度曲线	93
图 6-24 进行了前瞻预处理后的合成速度曲线	94
图 6-25 刀向跟随功能 GT_BufMove 的运动轨迹.....	95

图 6-26 插补缓存区内的点位运动速度图	97
图 6-27 刀向跟随功能 GT_BufGear 的运动轨迹	98
图 6-28 插补缓存区内的跟随运动速度图	99
图 6-29 刀向跟随功能之工件尺寸和刀运动轨迹	100
图 7-1 限位开关示意图	104
图 7-2 开环控制系统示意图	107
图 8-1 Home 回原点示意图 1.....	114
图 8-2 Home 回原点示意图 2.....	114
图 8-3 Home 回原点示意图 3.....	114
图 8-4 Home+Index 回原点示意图 1	118
图 8-5 Home+Index 回原点示意图 2	118
图 8-6 Home+Index 回原点示意图 3	118
图 8-7 Home+Index 回原点示意图 4	118
图 8-8 Home+Index 回原点示意图 5	118
图 9-1 轴运动范围	127
图 9-2 软限位触发	128
图 10-1 运动程序与应用程序的关系	131
图 10-2 MCT2008 运动程序编译说明界面.....	133
图 10-3 线程、函数和数据页的关系	134
图 10-4 电机到位检测功能	146
图 10-5 电机到位的运动状态检测	147
图 10-6 输出脉冲和输出电平比较	155
图 11-1 限位回原点	157
图 11-2 限位+Home 回原点	158
图 11-3 限位+Index 回原点	158
图 11-4 限位+Home+Index 回原点	158
图 11-5 Home 回原点.....	158
图 11-6 Home+Index 回原点	159
图 11-7 Index 回原点	159
图 11-8 强制 Home 回原点.....	159
图 11-9 强制 Home+Index 回原点	159
图 11-10 强制 Home+Index 回原点	160
图 11-11 强制 Home+Index 回原点.....	160
图 11-12 强制 Home+Index 回原点	160