

GC 编程手册



2023

Version 1.0

目录

目录	2
版权声明	6
联系我们	6
文档版本	7
前言	8
第一章 指令汇总	9
1 EtherCAT 指令	9
1.1 EtherCAT 主站操作	9
1.2 EtherCAT 状态检测	9
1.3 EtherCAT 回零功能	9
1.4 EtherCAT 从站 IO 指令	9
1.5 EtherCAT 高级功能	10
2 标准运动指令	11
2.1 控制器的基本操作	11
2.2 控制器的基本配置	11
2.3 控制器的状态检测	12
2.4 点位和 JOG 运动	13
2.5 回零运动	14
2.6 坐标系(插补)运动	14
2.7 IO 资源	18
2.8 手脉功能	20
2.9 龙门功能	21
2.10 位置比较输出	21
2.11 激光功能(选配)	22
2.12 PT 运动	24
2.13 PVT 运动	24
2.14 闭环控制(选配)	25
2.15 电子齿轮	25
2.16 电子凸轮	25
2.17 捕获功能	26
2.18 机械补偿	27
2.19 扩展资源	27
第二章 基本功能	31
1 EtherCAT 指令	31
1.1 功能介绍	31
1.2 指令说明	31
1.3 综合示例	35

2 控制器的基本操作.....	36
2.1 功能介绍.....	36
2.2 指令说明.....	36
2.3 综合示例.....	39
3 控制器的基本配置.....	41
3.1 功能介绍.....	41
3.2 指令说明.....	41
3.3 综合示例.....	50
4 控制器的状态检测.....	53
4.1 功能介绍.....	53
4.2 指令说明.....	54
4.3 综合示例.....	57
5 点位和 JOG 运动.....	59
5.1 功能介绍.....	59
5.2 指令说明.....	59
5.3 综合示例.....	67
6 回零运动.....	71
6.1 功能介绍.....	71
6.2 指令说明.....	74
6.3 综合示例.....	78
6.4 常见问题.....	79
7 坐标系(插补)运动.....	80
7.1 坐标系初始化.....	84
7.2 直线插补.....	91
7.3 平面圆弧插补.....	93
7.4 螺旋线插补.....	99
7.5 空间圆弧插补.....	101
7.6 多维插补.....	104
7.7 缓存区的操作(I/O, 延时等).....	106
7.8 坐标系的打包插补.....	117
7.9 坐标系的状态检测.....	120
7.10 坐标系的其他指令.....	124
7.11 综合示例.....	130
8 I/O 资源.....	133
8.1 功能介绍.....	133
8.2 指令说明.....	133
8.3 综合示例.....	151
9 手脉功能.....	153
9.1 功能介绍.....	153

9.2 指令说明.....	153
9.3 综合示例.....	154
10 龙门功能.....	155
10.1 功能介绍.....	155
10.2 指令说明.....	155
10.3 综合示例.....	156
第三章 高级功能.....	157
1 位置比较输出.....	157
1.1 多维位置比较输出	157
1.2 一维高速位置比较	161
1.3 二维高速位置比较	166
2 激光功能(选配).....	173
2.1 激光模式设置.....	173
2.2 基本控制模式.....	175
2.3 波形控制模式.....	184
2.4 位置比较控制模式	189
3 PT 运动.....	198
3.1 功能介绍.....	198
3.2 指令说明.....	199
3.3 综合示例.....	201
4 PVT 运动	204
4.1 功能介绍.....	204
4.2 指令说明.....	204
5 闭环控制(选配).....	207
5.1 功能介绍.....	207
5.2 指令说明.....	208
5.3 综合示例.....	210
6 电子齿轮	212
6.1 功能介绍.....	212
6.2 指令说明.....	213
6.3 综合示例.....	215
7 电子凸轮	216
7.1 功能介绍.....	216
7.2 指令说明.....	217
7.3 综合示例.....	221
8 捕获功能	223
8.1 功能介绍.....	223
8.2 指令说明.....	223
8.3 综合示例.....	227

9 机械补偿	229
9.1 螺距补偿	229
9.2 反间隙补偿	230
9.3 2D 平面补偿	231
10 扩展资源	237
10.1 辅助编码器	237
10.2 位置系统操作	237
10.3 控制器资源	241
10.4 数据采集	251
10.5 坐标系变换	256
第四章 动态库的使用	263
1 C++	265
2 C#	266
3 VB.NET	267
4 LABVIEW	269
第五章 注意事项	277
1 关于 NMC 指令	277
2 网络型控制器	278
3 PCIE 控制卡	279

版权声明

本手册版权归深圳市高川自动化技术有限公司所有, 未经本公司书面许可, 任何人不得翻印、翻译和抄袭本手册中的任何内容。

本手册中的信息资料仅供参考。由于改进设计和功能等原因, 高川自动化保留对本资料的最终解释权, 内容如有更改, 不另行通知。



调试、运动中的机器有危险! 用户有责任在机器中设计有效的出错处理和安全保护机制, 高川自动化没有义务和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

深圳市高川自动化技术有限公司

电话: 0755-23502680

邮箱: sales@gcauto.com.cn

网址: www.gcauto.com.cn

Shenzhen Gaochuan Industrial Automation Co., Ltd.

Tel: 0755-23502680

Email: sales@gcauto.com.cn

Website: www.gcauto.com.cn

文档版本

版本号	修订日期	内容
V1.0	2023 年 11 月 29 日	-
V1.01	2024 年 2 月 21 日	-

前言

为了给用户提供更快捷，更方便的服务，提高用户的工作效率，本手册主要针对控制卡各项功能的常用指令进行讲解(更多指令和功能请关注头文件或引用)，包括五大章节内容“指令汇总”，“基本功能”，“高级功能”，“动态库的使用”和“注意事项”，使用本手册时，建议优先阅读第五章内容；每章节都会有简单的概述，方便用户更好的理解每章节的内容。

第一章 指令汇总

本章节主要介绍控制卡中各项功能的指令含义，高度集中的指令汇总方便帮助用户在编程过程中能够快速找到所需要的功能指令。

1 EtherCAT 指令

1.1 EtherCAT 主站操作

函数原形	函数说明
NMC_EcatLoadConfigFromFile	下载 EtherCAT 配置文件
NMC_EcatStart	启动 EtherCAT 主站
NMC_EcatStop	停止 EtherCAT 主站

1.2 EtherCAT 状态检测

函数原形	函数说明
NMC_EcatGetSts	读取主站状态
NMC_EcatGetResource	读取当前在线的从站数量

1.3 EtherCAT 回零功能

函数原形	函数说明
NMC_EcatStartHome	启动回零
NMC_EcatGetHomeSts	读取驱动器的回零状态
NMC_EcatStopHome	终止回零

1.4 EtherCAT 从站 IO 指令

函数原形	函数说明
NMC_SetDOGroup	按通道设置通用输出 (支持超过 32 位)
NMC_SetDOBit	按位设置通用输出
NMC_GetDOGroup	按通道读取通用输出 (支持超过 32 位)
NMC_GetDOBit	按位读取通用输出

NMC_GetDIGroup	按通道读取通用输入(支持超过 32 位)
NMC_GetDIBit	按位取读取通用输入
NMC_IOModuleWr16Bit	扩展 IO 模块通道设置输出, 16 位长度
NMC_IOModuleWr32Bit	扩展 IO 模块通道设置输出, 32 位长度
NMC_IOModuleRdOut16Bit	扩展 IO 模块通道读取输出, 16 位长度
NMC_IOModuleRdOut32Bit	扩展 IO 模块通道读取输出, 32 位长度
NMC_IOModuleRd16Bit	扩展 IO 模块通道读取输入, 16 位长度
NMC_IOModuleRd32Bit	扩展 IO 模块通道读取输入, 32 位长度

注意: 该 IO 指令与标准 IO 资源指令一致;

1.5 EtherCAT高级功能

函数原形	函数说明
NMC_EcatSdoRead	EtherCAT 的 SDO 读取
NMC_EcatSdoWrite	EtherCAT 的 SDO 写入
NMC_EcatAxisSetMode	设置从站驱动器的工作模式, 用户一般不需要调用
NMC_EcatAxisGetMode	读取从站驱动器的工作模式
NMC_EcatAxisGetErrorCode	读取从站驱动器的错误代码
NMC_EcatAxisGetDi	读取从站驱动器的数字量输入状态

2 标准运动指令

2.1 控制器的基本操作

函数原形	函数说明
NMC_DevSearch	控制器搜寻
NMC_DevOpen	板卡打开(根据序号)
NMC_DevReset	控制器复位
NMC_DevOpenByID	板卡打开(根据 ID)
NMC_DevOpenByIP	板卡打开(根据 IP 地址)
NMC_DevClose	控制器关闭
NMC_MtOpen	打开单轴
NMC_MtClose	关闭单轴
NMC_CrdOpen	打开坐标系组
NMC_CrdOpenEx	打开坐标系组(支持多坐标系)
NMC_CrdClose	关闭坐标系组
NMC_LoadConfigFromFile	加载配置文件
NMC_SaveMotionConfig	保存当前配置到控制器(断电重启或复位后自动加载)

2.2 控制器的基本配置

函数原形	函数说明
NMC_MtSetLmtCfg	限位配置(同时配置有效及电平)
NMC_MtGetLmtOnOff	读取限位是否有效
NMC_MtGetLmtSns	读取限位有效电平
NMC_MtSetAlarmCfg	驱动报警配置(同时配置有效及电平)
NMC_MtGetAlarmOnOff	读取伺服报警是否有效
NMC_MtGetAlarmSns	读取伺服报警有效电平
NMC_MtSwLmtOnOff	设置软件限位是否有效
NMC_MtGetSwLmtOnOff	读取软件限位是否有效
NMC_MtSwLmtValue	设置软件限位数值

NMC_MtGetSwLmtValue	读取软件限位数值
NMC_MtSetSafePara	设置轴安全参数
NMC_MtGetSafePara	读取轴安全参数
NMC_MtSetStepMode	设置脉冲输出模式
NMC_MtGetStepMode	读取脉冲输出模式
NMC_MtSetAxisVelFilter	设置轴速度滤波参数
NMC_MtGetAxisVelFilter	读取轴速度滤波参数
NMC_MtSetAxisDampCompParam	设置轴补偿参数
NMC_MtSetPosErrAction	设置越限报警的动作
NMC_MtGetPosErrAction	读取越限报警的动作
NMC_MtSetStepFilter	设置脉冲输出滤波
NMC_MtGetStepFilter	读取脉冲输出滤波
NMC_SetEncMode	设置编码器模式
NMC_GetEncMode	读取编码器模式
NMC_MtSetPosErrLmt	设置允许的位置误差
NMC_MtGetPosErrLmt	读取允许的位置误差设定值
NMC_MtSetEstopDI	设置单轴急停 DI
NMC_MtGetEstopDI	读取单轴急停 DI
NMC_GetEstopDIEx	读取 Ex 单轴急停 DI
NMC_NMC_SetEstopDIEx	设置 Ex 单轴急停 DI, 触发后会置起 BIT_AXIS_ESTOP 标志位
NMC_ClrEstopDIEx	清除高级单轴急停 DI

2.3 控制器的状态检测

函数原形	函数说明
NMC_MtGetSts	读取当前轴状态
NMC_MtGetStsEx	快速读取当前轴状态(适合 PCIE 卡)
NMC_MtClrError	清除轴错误状态
NMC_MtClrStsByBits	清除轴错误状态, 按位清除

NMC_MtGetPrfPos	读取规划的位置
NMC_MtGetPrfVel	读取当前轴规划速度
NMC_MtGetAxisPos	读取机械轴的位置
NMC_MtGetCmdPos	读取发送到执行器的位置
NMC_MtGetEncPos	读取当前轴编码器通道位置
NMC_GetEncVel	读取编码器速度(单位是：脉冲/ms)
NMC_MtGetStsPack12Ex	读取打包轴状态

2.4 点位和JOG运动

函数原形	函数说明
NMC_MtSetPrfMode	设置单轴规划模式
NMC_MtGetPrfMode	读取单轴规划模式
NMC_MtSetPtpPara	设置 PTP 多个参数，并更新
NMC_MtGetPtpPara	读取 PTP 参数
NMC_MtSetPtpTgtPos	设置目标运动位置，只针对 PTP
NMC_MtGetPtpTgtPos	读取目标运动位置，只针对 PTP
NMC_MtSetVel	设置目标运动速度，只针对 PTP 和 Jog：PTP 模式下只接受正数，Jog 模式下正负号标识运动方向
NMC_MtGetVel	读取目标运动速度
NMC_MtSetJogPara	设置 Jog 运动参数
NMC_MtGetJogPara	读取 Jog 运动参数
NMC_MtUpdate	参数更新，启动运动，只针对 PTP 和 Jog
NMC_MtUpdateMulti	多轴同时启动
NMC_MtStop	单轴运动停止
NMC_MtAbruptStop	单轴急停，不会置起急停标志位
NMC_MtEstp	单轴急停
NMC_MtStopMulti	多轴同时停止
NMC_MtMovePtpAbs	单轴点位运动(绝对运动, 目标位置的正负代表运动方向)包含

	NMC_MtSetPrfMode , NMC_MtSetPtpPara , NMC_MtSetVel , NMC_MtSetPtpTgtPos , NMC_MtUpdate
NMC_MtMovePtpRel	单轴点位运动(相对运动, 目标位置的正负代表运动方向)
NMC_MtMovePtpAbsPack8	单轴点位运动打包(绝对运动)
NMC_MtMovePtpRelPack8	单轴点位运动打包(相对运动)
NMC_MtMoveJog	单轴连续速度运动(速度的正负代表运动方向)
NMC_MtSetPtpComparePara	设置 PTP 位置比较输出
NMC_MtGetPtpCompareSts	读取 PTP 比较输出状态

2.5 回零运动

函数原形	函数说明
NMC_MtSetHomePara	设置回零参数
NMC_MtGetHomePara	读取回零参数
NMC_MtGetHomeSts	读取回零状态
NMC_MtHome	启动回零
NMC_MtTryHome	尝试性回零(测试回零误差, 不清位置)
NMC_MtHomeStop	终止回零
NMC_MtGetHomeError	读取新回零位置和历史回零位置的差值

2.6 坐标系(插补)运动

2.6.1 坐标系初始化

函数原形	函数说明
NMC_CrdConfig	建立插补坐标系系统
NMC_CrdGetConfig	读取插补坐标系系统配置信息
NMC_CrdDelete	删除坐标系
NMC_CrdSetPara	设置坐标系参数
NMC_CrdGetPara	读取坐标系参数
NMC_CrdSetExtPara	设置坐标系扩展参数

NMC_CrdGetExtPara	读取坐标系扩展参数
NMC_CrdSetArcSecPara	设置圆弧插补参数(高级指令)
NMC_CrdGetArcSecPara	读取圆弧插补参数(高级指令)
NMC_CrdSetLookAheadCentriAcc	设置向心加速度限制
NMC_CrdSetFourthAxisToTurnVel	设置 4 轴直线插补 A 轴最大拐弯速度
NMC_CrdSetLookaheadPara	设置前瞻参数
NMC_CrdGetLookaheadPara	读取前瞻参数
NMC_CrdGetBufLength	读取插补线段长度
NMC_CrdGetStsEx	读取坐标系状态(PCIE 卡使用)

2.6.2 直线插补

函数原形	函数说明
NMC_CrdLineXYZEx	直线插补(带前瞻开关)
NMC_CrdLineXYZA	4 轴直线插补

2.6.3 平面圆弧插补

函数原形	函数说明
NMC_CrdArcCenterEx	XY 平面圆弧插补: 终点位置、圆心、方向(带前瞻开关)
NMC_CrdArcCenterYZEx	YZ 平面圆弧插补: 终点位置、圆心、方向(带前瞻开关)
NMC_CrdArcCenterZXEx	ZX 平面圆弧插补: 终点位置、圆心、方向(带前瞻开关)
NMC_CrdArcRadiusEx	XY 平面圆弧插补: 终点位置、半径、方向(带前瞻开关)
NMC_CrdArcRadiusYZEx	YZ 平面圆弧插补: 终点位置、半径、方向(带前瞻开关)
NMC_CrdArcRadiusZXEx	ZX 平面圆弧插补: 终点位置、半径、方向(带前瞻开关)
NMC_CrdArcPPPEx	XY 平面圆弧插补: 起点、中点、终点(带前瞻开关)
NMC_CrdEllipse	椭圆插补, 默认不参与速度前瞻, 起始和终止速度为 0(注意, 椭圆为整圆)

2.6.4 螺旋线插补

函数原形	函数说明
NMC_CrdHelixCenterEx	螺旋线插补(带前瞻开关)

2.6.5 空间圆弧插补

函数原形	函数说明
NMC_CrdArc3DEx	3D 圆弧插补(带前瞻开关)
NMC_CrdCircle3DEx	3D 圆弧插补(整圆)
NMC_CrdCirclePPP	XY 平面整圆插补

2.6.6 多维插补

函数原形	函数说明
NMC_CrdLineXYZD8	多轴直线插补(最多支持 8 轴)
NMC_CrdLineXYZD8Pack	打包的多轴直线插补(8 轴)

2.6.7 缓存区的操作(I0, 延时等)

函数原形	函数说明
NMC_CrdBufDo	缓冲区 D0(按位输出)
NMC_CrdBufDoEx	缓冲区 D0(掩码输出)
NMC_CrdBufOut	缓冲区输出模拟量或 PWM
NMC_CrdBufWaitDI	缓冲区 DI 等待
NMC_CrdBufSetEstopDI	缓冲区设置急停 DI
NMC_CrdBufEstopDIExOnOff	缓冲区高级急停 I0 启动关闭
NMC_CrdBufDelay	缓冲区延时
NMC_CrdBufAxMoveEx	缓冲区单轴移动(绝对位移移动, 带速度加速度设置)
NMC_CrdBufAxMoveRel	缓冲区单轴移动(相对位移移动)
NMC_CrdBufSetPtpMovePara	缓冲区单轴移动参数设置
NMC_CrdBufBeforeAxSyncMove	设置跟随运动前的运动补偿量
NMC_CrdBufWaitEncInPosition	缓冲区等待电机运动到位

NMC_CrdBufWaitPos	缓冲区等待特定位置，满足条件才执行下一条指令
NMC_AdvBufIoSetParam	设置 BufIo 输出参数
NMC_AdvBufIoGetParam	读取 BufIo 输出参数
NMC_CrdAdvBufIoOnAfterLen	缓冲区设置高级 BufIo 输出有效(运动一段距离后输出特定状态)
NMC_CrdAdvBufIoOffBeforeLen	缓冲区设置高级 BufIo 关闭输出(缓冲区全部运动结束前提前一段距离输出特定状态)
NMC_AdvBufIoOut	立即设置高级 BufIo 输出
NMC_AdvBufIoGetPulseCnt	读取 BufIo 的输出数量
NMC_DoBitPulseEnable	DO 脉冲输出
NMC_DoBitPulseDisable	关闭 DO 脉冲输出
NMC_CrdBufDoBitPulseEnable	DO 脉冲输出 (缓冲区)
NMC_CrdBufDoBitPulseDisable	关闭 DO 脉冲输出(缓冲区)
NMC_CrdBufStopMtn	缓冲区停止坐标系运动

2.6.8 坐标系的打包插补

函数原形	函数说明
NMC_CrdBufDataPack	打包插补数据，包括直线、圆弧插补，IO 操作等

2.6.9 坐标系的状态检测

函数原形	函数说明
NMC_CrdGetSts	读取坐标系状态
NMC_CrdGePrftPos	读取规划位置 XYZ
NMC_CrdGetAxisPos	坐标系模式下，读取多个轴的机械坐标位置
NMC_CrdGetVel	读取坐标系合成速度
NMC_CrdGetEncPos	读取编码器位置
NMC_CrdGetStsPack4	坐标系运动模式下，打包读取控制器状态
NMC_CrdBufGetFree	读取指令缓冲区空闲长度

NMC_CrdBufGetUsed	读取指令缓冲区已用长度
NMC_CrdGetUserSegNo	读取段号
NMC_CrdGetBufAllCmdCnt	读取总共压了多少条指令
NMC_CrdGetInnerSts	读取内部坐标系状态
NMC_CrdGetBufLeftLength	读取插补缓冲区中尚未完成的总位移量

2.6.10 坐标系的其他指令

函数原形	函数说明
NMC_CrdStartMtn	坐标系缓冲运动启动
NMC_CrdEndMtn	结束(指令压入)缓冲区运动(等待运动完后才结束压入,并置空闲标志)
NMC_CrdStopMtn	立即平滑停止运动
NMC_CrdClrError	清坐标系运动错误状态,同时清除所包含轴的错误状态
NMC_CrdBufClr	指令缓冲区清空
NMC_CrdSetOverRide	设置坐标系速度倍率
NMC_CrdGetOverRide	读取坐标系速度倍率
NMC_CrdSetOffset	设置轴缓冲区运动偏移
NMC_CrdEstopMtn	急停(并不清空指令缓冲区)
NMC_CrdGotoBreak	返回缓冲区运动的断点
NMC_CrdStartExeTimeCalc	开始计算缓冲区执行时间
NMC_CrdGetExeTime	读取缓冲区指令的执行时间,并停止计算,单位:ms
NMC_CrdSetBufLengthFlag	设置是否计算所有线段长度
NMC_SetMvProtect	启动坐标系的干涉保护
NMC_DelMvProtect	取消坐标系的干涉保护
NMC_GetMvProtectSts	读取坐标系的干涉状态
NMC_SetMvProtectMode01Para	设置坐标系的干涉保护参数
NMC_GetMvProtectMode01Para	读取坐标系的干涉保护参数

2.7 10资源

函数原形	函数说明
NMC_GetDIGroup	按通道设置通用输出(支持超过 32 位)
NMC_SetDOBit	按位设置通用输出
NMC_GetDOGroup	按通道读取通用输出(支持超过 32 位)
NMC_GetDOEx	读取输出 DO
NMC_GetDOBit	按位读取通用输出
NMC_GetDIGroup	按通道读取通用输入(支持超过 32 位)
NMC_GetDIBit	按位读取通用输入
NMC_IOModuleWr16Bit	扩展 IO 模块通道设置输出, 16 位长度
NMC_IOModuleWr32Bit	扩展 IO 模块通道设置输出, 32 位长度
NMC_IOModuleRdOut16Bit	扩展 IO 模块通道读取输出, 16 位长度
NMC_IOModuleRdOut32Bit	扩展 IO 模块通道读取输出, 32 位长度
NMC_IOModuleRd16Bit	扩展 IO 模块通道读取输入, 16 位长度
NMC_IOModuleRd32Bit	扩展 IO 模块通道读取输入, 32 位长度
NMC_MtGetMotionIO	读取运动控制专用 IO
NMC_MtGetMotionIOLogical	读取运动控制专用 IO, 逻辑电平
NMC_MtPtStartMtnEx	DI 沿触发
NMC_SetDIBitRevs	按位设置扩展 IO 输入信号取反(不会改变灯的状态)
NMC_GetDIBitRevs	按位读取扩展 IO 输入信号取反的设置值
NMC_SetDOBitRevs	按位设置扩展 IO 输出信号取反(会改变灯的状态)
NMC_GetDOBitRevs	按位读取扩展 IO 输出信号取反的设置值
NMC_SetDOBitAutoReverse	DO 输出定时脉冲
NMC_GetIOModuleSts	读取扩展 IO 模块的状态
NMC_IOModuleSetEn	设置扩展 IO 模块有效(带模块类型)
NMC_IOModuleGetType	读取扩展 IO 模块类型
NMC_MtSetSvOn	设置伺服 ON, 使能驱动器
NMC_MtSetSvOff	设置伺服 OFF, 卸载使能
NMC_MtSetSvClr	设置伺服报警清除
NMC_SetDacMode	设置模拟量输出范围

NMC_GetDacMode	读取模拟量输出范围
NMC_SetAdcMode	设置模拟量输入范围
NMC_GetAdcMode	读取模拟量输入范围
NMC_SetDac	模拟量输出
NMC_GetDac	读取模拟量输出值
NMC_GetAdc	读取模拟量输入值
NMC_SetDioMapping	增加一组映射
NMC_GetAllDioMapping	读取所有的 DIO 映射数据
NMC_ClrAllDioMapping	清除所有的 DIO 映射关系
NMC_SetDacBias	设置模拟量输出的偏移值
NMC_GetDacBias	读取模拟量输出的偏移值
NMC_SetDacLmt	设置模拟量输出的极限值
NMC_GetDacLmt	读取模拟量输出的极限值
NMC_SetDIFilter	设置 DI 的滤波系数
NMC_GetDIFilter	读取 DI 的滤波系数
NMC_GetDIEx	读取数字量输入信号值
NMC_GetDIRaw	读取数字量输入信号值(原始值)
NMC_SetDiReverseCount	设置数字量输入信号的翻转计数
NMC_GetDiReverseCount	读取数字量输入信号的翻转计数
NMC_SetDOMask	输出 DO, 按照掩码
NMC_GetAdcVoltageMulti	读取模拟量输入电压值
NMC_SetDacVoltageMulti	设置模拟量输出电压值
NMC_GetDacVoltageMulti	读取模拟量输出电压值

2.8 手脉功能

函数原形	函数说明
NMC_SetHandWheel	配置启用手轮
NMC_SetHandWheelInput	选择手轮跟随的编码器通道, 默认 256, 扩展编码器 1

NMC_SetHandWheelRatio	设置手轮跟随的倍率
NMC_ClrHandWheel	退出手轮

2.9 龙门功能

函数原形	函数说明
NMC_SetGantryMaster	设置龙门主动轴
NMC_SetGantrySlave	设置龙门从动轴
NMC_DelGantryGroup	龙门功能关闭

2.10 位置比较输出

2.10.1 多维位置比较输出(软件比较)

函数原形	函数说明
NMC_CompXDimensSetParam	设置多维位置比较的参数，并清除二维位置比较位置数据
NMC_CompXDimensGetParam	读取多维位置比较的参数
NMC_CompXDimensSetCmpOutMode	设置多维位置比较输出的模式
NMC_CompXDimensSetData	设置多维位置比较的数据
NMC_CompXDimensOnoff	启动或停止多维位置比较输出功能
NMC_CompXDimensStatus	读取多维位置比较的输出状态

2.10.2 一维高速位置比较(硬件比较)

函数原形	函数说明
NMC_CompHs1DimensSetParam	设置高速一维位置比较的参数(比较编码器)
NMC_CompHs1DimensGetParam	读取高速一维位置比较的参数
NMC_CompHs1DimensSetData	设置高速一维比较数据
NMC_CompHs1DimensOnOff	高速一维位置比较使能
NMC_CompHs1DimensStatus	高速一维位置比较的状态

2.10.3 二维高速位置比较(硬件比较)

函数原形	函数说明
NMC_Comp2DimensSetParam	设置高速二维位置比较的参数(比较编码器)
NMC_Comp2DimensGetParam	读取高速二维位置比较的参数
NMC_Comp2DimensSetData	设置高速二维比较数据
NMC_Comp2DimensOnoff	高速二维位置比较使能
NMC_Comp2DimensStatus	读取高速二维位置比较的状态
NMC_Comp2DimensStatusEx	读取高速二维位置比较的状态
NMC_Comp2DimensSetParamEx	设置高速二维位置比较的参数
NMC_Comp2DimensGetParamEx	读取高速二维位置比较的参数

2.11 激光功能(选配)

2.11.1 激光模式设置

函数原形	函数说明
NMC_LaserSetMode	设置激光控制的模式

2.11.2 基本控制模式

函数原形	函数说明
NMC_LaserSetPowerEx	设置立即输出激光的能量
NMC_LaserOnOff	激光立即输出开关
NMC_CrdBufLaserOnOff	缓冲区激光开关
NMC_LaserGetOnOff	读取当前激光开关状态
NMC_CrdBufLaserOnOff	缓冲区激光开关
NMC_CrdBufLaserSetFollow	设置缓冲区激光能量跟随
NMC_LaserSetFollowParam	设置缓冲区激光能量跟随滤波
NMC_SetLaserPowerCmpTable	设置激光能量补偿表
NMC_StartLaserPowerComp	启动激光能量补偿

NMC_StopLaserPowerComp	停止激光能量补偿
NMC_LaserSetOutputTypeEx	激光物理信号输出类型配置
NMC_LaserGetPower	读取当前激光的能量
NMC_LaserGetOnOff	读取当前激光开关状态
NMC_LaserSetParam	设置激光参数
NMC_LaserGetParam	读取激光参数
NMC_CrdBufLaserSetParam	缓冲区设置激光参数

2.11.3 波形控制模式

函数原形	函数说明
NMC_LaserSetTimeArrayPara	设置时间数组输出参数
NMC_LaserSetTimeArrayPower	设置激光能量, 对于立即输出模式, group 无效, pLaserPower 为指定的单个能量信息
NMC_LaserTimeArrayExe	执行时间序列激光, 只在时间序列输出模式下有效
NMC_CrdBufLaserTimeArrayExe	缓冲区执行时间序列激光, 只在时间序列输出模式下有效

2.11.4 位置比较控制模式

函数原形	函数说明
NMC_SHIOConfigPara	配置 SHIO 功能的参数
NMC_SHIOEnableMinFrg	SHIO 输出的最小频率功能开
NMC_SHIODisableMinFrg	SHIO 输出的最小频率功能关闭
NMC_CrdBufSHIOSetMinFrg	缓冲区 SHIO 输出的最小频率功能开
NMC_CrdBufSHIOClrMinFrg	缓冲区 SHIO 输出的最小频率功能关闭
NMC_CrdBufSHIOSetParam	缓冲区配置 SHIO 功能的参数
NMC_SHIOChangeAxisMask	切换轴(允许和坐标系不完全一致)
NMC_CrdBufSHIOChangeAxisMask	切换轴(允许和坐标系不完全一致), 缓冲区指令
NMC_SHIOGateOn	允许 GATE 输出
NMC_SHIOGateOff	禁止 GATE 输出

NMC_SHIOTriggerOn	设置 Trigger 输出
NMC_SHIOTriggerOff	禁止 Trigger 输出
NMC_BufSHIOGateOn	缓冲区允许 GATE 输出
NMC_BufSHIOGateOff	缓冲区禁止允许 GATE 输出
NMC_SetPwmToGate	PWM 映射到 GATE 引脚输出
NMC_CrdBufSHIOGatePulse	缓冲区 SHIO 点动出光, 输出一段 gate 脉冲, 注: 必须在激光 SHIO 控制模式下使用
NMC_SHIOGatePulse	SHIO 点动出光, 输出一段 gate 脉冲, 注: 必须在激光 SHIO 控制模式下使用
NMC_PwmPulseOut	输出一段 PWM 脉冲, 注: 必须在激光 SHIO 控制模式下使用
NMC_SetPWMPort	设置 PWM 的输出通道
NMC_GetPWMPort	读取 PWM 的输出通道

2.12 PT运动

函数原形	函数说明
NMC_MtPtSetStatic	设置 PT 运动的数据模式和循环次数
NMC_MtPtGetStatic	读取 PT 运动的数据模式和循环次数
NMC_MtPtGetSpace	查询 PT 数据剩余空间大小
NMC_MtPtPush	向 Pt 运动缓存区中压运动数据段
NMC_MtPtBufClr	清空 PT 数据
NMC_MtPtStartMtn	启动 Pt 运动
NMC_MtPtSetPara	设置 PT 运动的相关参数

2.13 PVT运动

函数原形	函数说明
NMC_MtPvtSetPara	设置 PVT 运动的相关参数
NMC_MtPvtGetPara	读取 PVT 运动的相关参数
NMC_MtPvtData	向 PVT 运动缓存区中压运动数据段

NMC_MtPvtBufGetSpace	查询 PVT 数据剩余空间大小
NMC_MtPvtBufClr	清空 PVT 数据
NMC_MtPvtStartMtn	启动 PVT 运动

2.14 闭环控制(选配)

函数原形	函数说明
NMC_MtSetCloseLoopDac	设置单轴闭环控制的 DA 参数, 轴与 DA 通道的对应
NMC_MtGetCloseLoopDac	读取单轴闭环控制的 DA 参数
NMC_MtSetCtrlMode	设置单轴的控制模式: 默认使用对应轴的编码器作为输入反馈, 对应序号的 DAC 作为输出
NMC_MtGetCtrlMode	读取单轴的控制模式
NMC_MtSetPIDPara	设置对应组号的 PID 参数
NMC_MtGetPIDPara	读取对应组号的 PID 参数
NMC_MtSetPIDIndex	设置使用哪组 PID
NMC_MtGetPIDIndex	读取正使用的 PID 组号

2.15 电子齿轮

函数原形	函数说明
NMC_MtGearSetDir	设置 Gear 跟随方向
NMC_MtGearGetDir	读取 Gear 跟随方向
NMC_MtGearSetMaster	设置 Gear 主轴参数
NMC_MtGearGetMaster	读取 Gear 主轴参数
NMC_MtGearSetRatio	设置 Gear 跟随倍率
NMC_MtGearGetRatio	读取 Gear 跟随倍率
NMC_MtGearStartMtn	启动 Gear 运动

2.16 电子凸轮

函数原形	函数说明
NMC_MtFollowSetDir	设置 FOLLOW 跟随方向
NMC_MtFollowGetDir	读取 FOLLOW 跟随方向
NMC_MtFollowSetMaster	设置 FOLLOW 主轴参数
NMC_MtFollowGetMaster	读取 FOLLOW 主轴参数
NMC_MtFollowSetLoopCount	设置 FOLLOW 的循环执行次数
NMC_MtFollowGetLoopCount	读取 FOLLOW 的循环执行次数
NMC_MtFollowSetEvent	设置 FOLLOW 的启动事件
NMC_MtFollowGetEvent	读取 FOLLOW 的启动事件
NMC_MtFollowGetSpace	读取 FOLLOW 的 FIFO 剩余空间
NMC_MtFollowPushData	设置 FOLLOW 的数据
NMC_MtFollowClear	清除 FOLLOW 对应 FIFO 号的数据
NMC_MtFollowStart	启动 Follow 运动
NMC_MtFollowSwitch	切换 Follow 运动的 FIFO 号

2.17 捕获功能

函数原形	函数说明
NMC_MtSetCaptSns	设置捕获有效电平
NMC_MtSetCapt	启动捕获
NMC_MtClrCaptSts	清除轴的捕获状态
NMC_MtGetCaptPos	读取捕获位置
NMC_MtSetAdvCaptParam	设置高级捕获参数, 并启动捕获
NMC_MtClrAdvCaptSts	清除高级捕获状态, 并取消该通道的捕获
NMC_MtGetAdvCaptPos	读取高级捕获状态
NMC_MtSetCaptRepeat	设置重复捕获次数
NMC_MtGetCaptRepeatStatus	读取重复捕获计数
NMC_MtGetCaptRepeatPosMulti	读取重复捕获位置值

2.18 机械补偿

2.18.1 螺距补偿

函数原形	函数说明
NMC_MtSetLeadScrewCompPara	设置螺距误差补偿参数
NMC_MtEnableLeadScrew	使能或禁止螺距位置补偿

2.18.2 反间隙补偿

函数原形	函数说明
NMC_MtSetBacklash	设置反向间隙补偿
NMC_MtGetBacklash	读取反向间隙补偿

2.18.3 XY平面补偿

函数原形	函数说明
NMC_Set2DCompensationTable	设置补偿表：以 XY 为基准参考，标定 XYZ 方向的偏差数据，并进行设置
NMC_Get2DCompensationTable	读取补偿表
NMC_Set2DCompensation	设置并启动 XY 平面误差补偿
NMC_Get2DCompensation	读取 XY 平面的误差补偿参数

2.19 扩展资源

2.19.1 辅助编码器

函数原形	函数说明
NMC_GetEncPos	读取编码器通道值
NMC_SetEncPos	设置编码器通道值

2.19.2 位置系统操作

函数原形	函数说明
NMC_MtZeroPos	单轴位置系统清零，规划以及编码器
NMC_MtSetAxisPos	设定轴位置
NMC_MtSetEncPos	设定编码器位置
NMC_MtSetPrfCoe	设置单轴比例系数
NMC_MtGetPrfCoe	读取单轴比例系数
NMC_MtSetEncCoe	设置轴通道编码器的系数, 默认为 1
NMC_MtGetEncCoe	读取轴通道编码器的系数
NMC_MtSetAxisArrivalPara	设置轴的到位误差参数
NMC_MtGetAxisArrivalPara	读取轴的到位误差参数
NMC_MtPrfConfig	设置单轴规划高级参数
NMC_MtGetPrfConfig	读取单轴规划高级参数
NMC_MtGetStsMulti	读取多轴状态

2.19.3 控制器资源

函数原形	函数说明
NMC_GetTime	读取时间
NMC_SetTime	设置时间
NMC_GetUID	读取控制器唯一序列号
NMC_UserParaWr	设置用户参数
NMC_UserParaRd	读取用户参数
NMC_SetCommPara	设置通讯参数
NMC_DevWriteID	修改板卡 ID 号
NMC_DevReadID	读取板卡 ID 号
NMC_GetDllVersion	读取库的版本
NMC_GetMtLibVersion	当前运动控制器固件的版本等信息
NMC_GetCardInfo	读取当前运动控制器信息
NMC_SetCmdDebug	启动指令调试

NMC_GetErrDesc	读取错误代码信息
NMC_SaveConfigToFile	保存为配置文件
NMC_GetBackedVarGroup2	读取当前备份的变量数值(断电自动保存)
NMC_SetBackedVarOnOff	开始或关闭变量的自动备份(断电自动保存，默认关闭状态)
NMC_GetBackedVarOnOff	读取当前自动备份的开启状态
NMC_DevGetPara	读取系统参数(long 型) IP 地址
NMC_DevSetPara	设置系统参数(long 型) IP 地址
NMC_UserSetPassword	设置用户密码
NMC_UserLogin	用户登陆
NMC_UserLogout	用户退出登陆
NMC_SetProfilePeriod	设置控制器的规划周期
NMC_GetProfilePeriod	读取控制器的规划周期
NMC_GetLastErr	读取最后一的错误代码
NMC_SetErrCodeMode	设置指令错误返回值模式
NMC_SetWatchDog	设置指令通讯看门狗
NMC_UserParaWrEx	设置扩展用户参数
NMC_UserParaRdEx	读取扩展用户参数
NMC_UserPrint	打印信息
NMC_GetClock	读取计时时钟，控制器上电开始

2.19.4 数据采集

函数原形	函数说明
NMC_GetCollectDataAddr	读取需要采集数据变量的地址
NMC_ConfigCollect	配置采集数据通道, 需要配置对应结构体参数
NMC_CollectOnOff	启动或停止数据采集
NMC_GetCollectSts	读取采集状态
NMC_GetCollectData	读取采集数据
NMC_ClearCollectSts	清除采集状态

2.19.5 坐标系变换

函数原形	函数说明
<u>NMC_CrdSetTransRotate</u>	使能旋转转换处理
<u>NMC_CrdDelTransRotate</u>	关闭旋转转换处理
<u>NMC_CrdSetTransPolar</u>	使能极坐标转换处理
<u>NMC_CrdRunToPolarPos</u>	运行至设定的极坐标位置并且进行圈数清零处理(利用单轴 PTP 运行到指定位置)
<u>NMC_CrdRunToPolarTheta</u>	运行至设定的极坐标角度位置(利用单轴 PTP 运行到指定位置)
<u>NMC_CrdDelTransPolar</u>	销毁极坐标机型(只恢复直角坐标系)
<u>NMC_CrdSetTransXYZA</u>	XYZA 机型设置接口
<u>NMC_CrdDelTransXYZA</u>	销毁 XYZA 机型, 回归 XYZ 结构
<u>NMC_CrdSetXYZAToolCalc</u>	求工具参数
<u>NMC_CrdSetXYZAPara</u>	设置 XYZA 机型参数
<u>NMC_CrdGetXYZAPara</u>	设置 XYZA 机型参数

第二章 基本功能

本章节主要针对运动控制卡的基本功能的指令，指令参数的含义和综合应用示例的讲解，方便帮助用户更好的了解控制器。

1 EtherCAT指令

1.1 功能介绍

GCE 系列控制卡在打开卡后必须先进行 EtherCAT 配置下载，启动 EtherCAT 主站等操作，才能进行后续操作，EtherCAT 配置文件 gml 的生成参考《GCS 用户手册》->功能->EtherCcat 测试，完整的介绍了其使用过程。EtherCAT 轴固定从序号 16 开始；EtherCAT 的 IO 访问与本地 IO 访问指令一样，其中 groupID 为从站序号(2 开始为从站)，每个从站的 IO 索引 index 从从站序号 groupID*32 开始；

1.2 指令说明

(1)EtherCAT配置下载

[NMC_EcatLoadConfigFromFile\(HAND devHandle, char *gmlFileName, short masterIdx\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄，通过 NMC_DevOpen 等接口读取
gmlFileName	char *	输入	总线配置文件.gml 路径(该文件通过高川 EtherCAT 配置工具生成(具体步骤请看《GCS 用户手册》第一章 2.4 章节)

(2)启动EtherCAT主站通讯

[NMC_EcatStart\(HAND devHandle\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

(3)停止EtherCAT主站通讯

[NMC_EcatStop\(HAND devHandle \);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
-----------	------	----	-------

(4) 读取EtherCAT主站状态

[NMC_EcatGetSts\(HAND devHandle, unsigned short *pSts\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pSts	unsigned short *	输出	0 为主站已经启动，其他值为尚未启动

(5) 读取当前在线的从站数量

[NMC_EcatGetResource\(HAND devHandle, short *pSlaveCnt, short *pDriverCnt, short *pIoCnt\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pSlaveCnt	short*	输出	返回从站数量
pDriverCnt	short*	输出	返回驱动器从站数量
pIoCnt	short*	输出	返回 IO 从站数量

(6) 启动回零

[NMC_EcatStartHome\(HAND axisHandle, int method, int velSwitch, int velZero, int acc, int prmEx\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄，通过 NMC_MtOpen 接口读取
method	int	输入	回零方式，根据标准 402 协议，具体请参考驱动器相关的说明
velSwitch	int	输入	找原点速度，脉冲/ms
velZero	int	输入	找 index 脉冲速度，脉冲/ms
acc	int	输入	加速度，脉冲/ms ²
prmEx	int	输入	保留

注意: EtherCAT 控制卡可以调用 2 种回零指令实现回零功能：[NMC_EcatStartHome](#) 和 [NMC_MtHome](#);

(7) 读取驱动器的回零状态

[NMC_EcatGetHomeSts\(HAND axisHandle, unsigned char *pStatus\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pStatus	unsigned char *	输出	驱动器的回零状态

注意：参考 [NMC_MtGetHomeSts;](#)

(8) 终止回零

[NMC_EcatStopHome\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄，通过 NMC_MtOpen 接口读取

(9) EtherCAT的SDO读取

[NMC_EcatSdoRead\(HAND devHandle, short devId, unsigned short idx, unsigned char sub_idx, unsigned int len, unsigned char *val, unsigned int *abort_code\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
devId	short	输入	从站号，取值[0, N]
idx	unsigned short	输入	SDO 索引
sub_idx	unsigned char	输入	SDO 子索引
len	unsigned int	输入	读取长度，单位字节
val	unsigned char *	输出	返回数据
abort_code	unsigned int *	输出	返回终止代码，指令出错时返回详细错误代码

(10) EtherCAT的SDO写入

[NMC_EcatSdoWrite\(HAND devHandle, short devId, unsigned short idx, unsigned char sub_idx, unsigned int len, unsigned char *val, unsigned int *abort_code\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
devId	short	输入	从站号, 取值[0, N]
idx	unsigned short	输入	SD0 索引
sub_idx	unsigned char	输入	SD0 子索引
len	unsigned int	输入	写入长度, 单位字节
val	unsigned char *	输出	写入数据
abort_code	unsigned int *	输出	返回终止代码, 指令出错时返回详细错误代码

(11) 设置从站驱动器的工作模式, 用户一般不需要调用

[NMC_EcatAxisSetMode\(HAND axisHandle, unsigned char mode\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mode	unsigned char	输入	Control Mode : CSP:8 CSV:9 CST:10

(12) 读取从站驱动器的工作模式

[NMC_EcatAxisGetMode\(HAND axisHandle, unsigned char *pMode\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pMode	unsigned char*	输出	返回工作模式

(13) 读取从站驱动器的错误代码

[NMC_EcatAxisGetErrorCode\(HAND axisHandle, unsigned short *pError\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pError	unsigned short *	输出	返回错误代码

(14) 读取从站驱动器的数字量输入状态

[NMC_EcatAxisGetDi\(HAND axisHandle, long *pDiVal\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

axisHandle	HAND	输入	轴句柄
pDiVal	long*	输出	返回数子量输入

1.3 综合示例

```
//ECAT 主站启动操作
short gc_example(void)
{
    /****** 已省略打开控制器部分，统一描述在第五章1 *****/

    int timeOut = 5000; // 等待时间ms
    unsigned short mstSts; // 获得Ecat状态
    rtn = NMC_EcatLoadConfigFromFile(devHandle, "test.gml", 0); // 下载配置文件(路径为当前工程的根目录)
    gc_rtn_error(rtn);
    rtn = NMC_EcatStart(devHandle); // 启动主站
    gc_rtn_error(rtn);
    while (1) // 等待主站,启动完成
    {
        rtn = NMC_EcatGetSts(devHandle, &mstSts); // 获得Ecat状态
        gc_rtn_error(rtn);
        if (mstSts == 0) // 主站启动OK
        {
            break;
        }
        _sleep(1); // 等待1ms
        timeOut--;
        if (timeOut < 0) // 启动主站超时
        {
            return 1;
        }
    }
    rtn = NMC_MtClrError(axisHandle); // 清除轴错误状态
    gc_rtn_error(rtn);
    return rtn;
}
```

2 控制器的基本操作

2.1 功能介绍

控制器的打开分为三种方式：

- (1) 调用函数 [NMC DevOpen](#) 按照序号打开, 序号从 0 开始;
- (2) 按照控制器的名称打开 [NMC DevOpenByID](#), 控制器的名称默认为 CARD1, 可通过 GCS 工具设置控制卡的名称《GCS 用户手册》->功能->控制器信息;
- (3) 按照网络型控制器设置的 IP 地址开打 [NMC DevOpenByIP](#);

2.2 指令说明

- (1) 搜索可用的, 已连接计算机的控制器, 返回控制器的数量、名称、以及描述信息

[NMC DevSearch\(TSearchMode mode, unsigned short *pDevNo, TDevInfo *pInfoList \);](#)

名称	数据类型	输入/输出	描述
mode	TSearchMode	输入	搜索模式, 枚举类型: USB、Ethernet、RS485 DLL 内部处理, 任意一个值均可
pDevNo	unsigned short*	输出	返回已连接控制器的数目。
pInfoList	TDevInfo *	输出	返回控制器信息列表 typedef struct { unsigned short address; // 在上 位机系统中的控制器序号, char idStr[16]; // 识别字符串 char description[64]; // 描述符 unsigned short ID; // 板上 ID(未用) } TDevInfo;

- (2) 根据序号打开控制器

[NMC DevOpen\(unsigned short devNo, PHAND pDevHandle \);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devNo	unsigned short	输入	序号, 取值范围 0~N
pDevHandle	PHAND	输出	控制器句柄指针

注意: 一台电脑上有多张 PCIE/GCE 卡时, 需要驱动安装完成后重启电脑, 可确保控制卡排序序号 (devNo) 固定;

(3) 复位控制器

[NMC DevReset \(HAND devHandle\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

(4) 根据 ID 打开控制器, 获得操作句柄

[NMC DevOpenByID \(char *idStr, PHAND pDevHandle \);](#)

名称	数据类型	输入/输出	描述
idStr	char *	输入	ID 字符串 (默认 CARD1) C 语言格式字符串, 以 \0 结尾。最长 16 字节 (包括结尾)
pDevHandle	PHAND	输出	控制器句柄指针

注意: ID 字符串可使用 GCS 工具修改;

(5) 根据控制器 IP 地址打开控制器, 获得操作句柄

[NMC DevOpenByIP \(unsigned char *pIPv4Array, PHAND pDevHandle \);](#)

名称	数据类型	输入/输出	描述
pIPv4Array	unsigned char *	输入	板卡 IP 地址, 四个字节, 例如: unsinged char ipv4[4] = {192, 168, 1, 110};
pDevHandle	PHAND	输出	控制器句柄指针

注意: PCIE/GCE 控制卡不支持 [NMC DevOpenByIP](#) 打开, 不需要设置 IP;

(6) 关闭控制器

[NMC_DevClose\(PHAND pDevHandle \);](#)

名称	数据类型	输入/输出	描述
pDevHandle	PHAND	输入	控制器句柄指针

(7) 打开单轴，读取轴句柄

[NMC_MtOpen\(HAND devHandle, short itemNo, PHAND pAxisHandle \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
itemNo	short	输入	轴号, 取值范围[0, n]
pAxisHandle	PHAND	输出	轴句柄指针

(8) 关闭单轴

[NMC_MtClose\(PHAND pAxisHandle \);](#)

名称	数据类型	输入/输出	描述
pAxisHandle	PHAND	输入	轴句柄指针

(9) 打开坐标系组

[NMC_CrdOpen\(HAND devHandle, PHAND pCrdHandle \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pCrdHandle	PHAND	输出	坐标系句柄指针

注意：启用坐标系运动时，此指令执行一次即可，直到执行 [NMC_CrdClose](#) 才需要再次打开；

(10) 打开坐标系组 (支持多坐标系)

[NMC_CrdOpenEx\(HAND devHandle, short itemNo, PHAND pCrdHandle \)](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
itemNo	short	输入	坐标系号, 取值范围[0, 1]
pCrdHandle	PHAND	输出	坐标系句柄指针

(11) 关闭坐标系组

[NMC_CrdClose\(PHAND pCrdHandle \);](#)

名称	数据类型	输入/输出	描述
pCrdHandle	PHAND	输入	坐标系句柄指针

(12) 加载配置文件，配置文件从 GCS 工具配置得到

[NMC_LoadConfigFromFile\(HAND devHandle, char *pFilePath\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pFilePath	char *	输入	配置文件路径(看示例)

(13) 保存基本的配置信息, 将这些信息保存到系统参数区, 控制器重启或 [NMC_DevReset](#) 后, 系统将使用这些参数。保存的参数包括报警、限位、脉冲方式、编码器方式、安全参数、滤波参数

[NMC_SaveMotionConfig\(HAND devHandle, short enable\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
enable	short	输入	1: 启用保存功能, 0: 关闭保存功能

2.3 综合示例

```
// 控制器基本操作
short gc_example(void)
{
    short rtn;                // 指令返回值
    short devNum;             // 设备序号, 从0开始
    short axisNum;            // 轴号, 从0开始
    HAND devHandle;           // 设备句柄
    HAND axisHandle[4];       // 轴句柄
    // 一个控制器时不必使用搜索控制器指令, 直接打开控制器即可
    rtn = NMC_DevOpen(devNum, &devHandle);           //打开控制器
    gc_rtn_error(rtn);
    rtn = NMC_MtOpen(devHandle, axisNum, &axisHandle[0]); //打开轴, 并获得轴句柄
    gc_rtn_error(rtn);
    rtn = NMC_MtSetSvOn(axisHandle[0]);               //打开轴使能(根据实际情况使用)
    gc_rtn_error(rtn);
}
```

```
rtn = NMC_LoadConfigFromFile(devHandle, "GCN800.cfg");    //加载控制器配置，文件在当  
前工程目录下  
gc_rtn_error(rtn);  
/*  
for (int i = 0; i < 4; i++) {  
    rtn = NMC_MtOpen(devHandle, i, &axisHandle[i]); //可以连续打开轴，并读取多个轴操  
作句柄  
    gc_rtn_error(rtn);  
}  
*/  
return rtn;  
}
```

3 控制器的基本配置

3.1 功能介绍

- (1) 在控制器能正常使用之前，除了打开控制器外，我们还需要对轴的正负限位是否有效、限位触发电平高或低、原点触发电平高或低、脉冲形式是正负脉冲还是脉冲加方向、脉冲方向的正负调整、编码器反馈、IO输入输出等配置；
- (2) 以上配置除了利用各个函数经行配置外，我们还提供了另外一种更为简捷的方法，即利用GCS配置工具生成配置文件，然后调用函数[NMC_LoadConfigFromFile](#)加载到控制器里面；
- (3) 关于配置文件的生成，请查看《GCS用户手册》->参数->参数配置器；

3.2 指令说明

- (1) 伺服报警配置，可同时配置伺服报警是否有效及其触发电平，默认报警为无效

[NMC_MtSetAlarmCfg\(HAND axisHandle, short alarmEnable, short alarmSns\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
alarmEnable	short	输入	伺服报警触是否有效，1 为有效，0 为无效
alarmSns	short	输入	伺服报警触发电平，1 为高电平触发，0 为低电平触发

- (2) 读取伺服报警触发是否有效的配置

[NMC_MtGetAlarmOnOff\(HAND axisHandle, short *enable\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
enable	short*	输出	伺服报警是否有效，1 为有效，0 为无效

- (3) 读取伺服报警触发电平

[NMC_MtGetAlarmSns\(HAND axisHandle, short *swt\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

swt	short*	输出	伺服报警触发电平设置，1 为高电平触发，0 为低电平触发
-----	--------	----	------------------------------

(4) 硬件限位配置，同时配置正负限位是否有效及其触发电平

[NMC MtSetLmtCfg\(HAND axisHandle, short posLmtEnable, short negLmtEnable, short posLmtSns, short negLmtSns \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posLmtEnable	short	输入	正向限位是否有效，1 为有效，0 为无效
negLmtEnable	short	输入	负向限位是否有效，1 为有效，0 为无效
posLmtSns	short	输入	正向限位触发电平，1 为高电平触发，0 为低电平触发
negLmtSns	short	输入	负向限位触发电平，1 为高电平触发，0 为低电平触发

注意：限位被触发，轴反向运动可解除，同向不可运动，需要执行 [NMC MtClrError](#) 指令；

(5) 读取限位触发是否有效

[NMC MtGetLmtOnOff\(HAND axisHandle, short*posEn, short*negEn\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posEn	short*	输出	正向限位是否有效，1 为有效，0 为无效
negEn	short*	输出	负向限位是否有效，1 为有效，0 为无效

(6) 读取限位触发电平

[NMC MtGetLmtSns\(HAND axisHandle, short*posSwt, short*negSwt\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posSwt	short*	输出	正向限位触发电平设置，1 为高电平触发，0 为低电平触发

negSwT	short*	输出	负向限位触发电平设置，1 为高电平触发，0 为低电平触发
--------	--------	----	------------------------------

(7) 软限位配置

[NMC MtSwLmtOnOff\(HAND axisHandle, short enable\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
enable	short	输入	软件限位是否有效，1 为有效，0 为无效

(8) 读取软件限位触发是否有效，默认有效，值为-2147483647~+2147483647，单位脉冲

[NMC MtGetSwLmtOnOff\(HAND axisHandle, short*enable\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
enable	short*	输出	软件限位是否有效，1 为有效，0 为无效

(9) 设置软件限位的数值

[NMC MtSwLmtValue\(HAND axisHandle, long posLmt, long negLmt\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posLmt	long	输入	正向软件限位设置值，单位为脉冲数
negLmt	long	输入	负向软件限位设置值，单位为脉冲数

(10) 读取软件限位的数值

[NMC MtGetSwLmtValue\(HAND axisHandle, long *posLmt, long *negLmt\);](#)

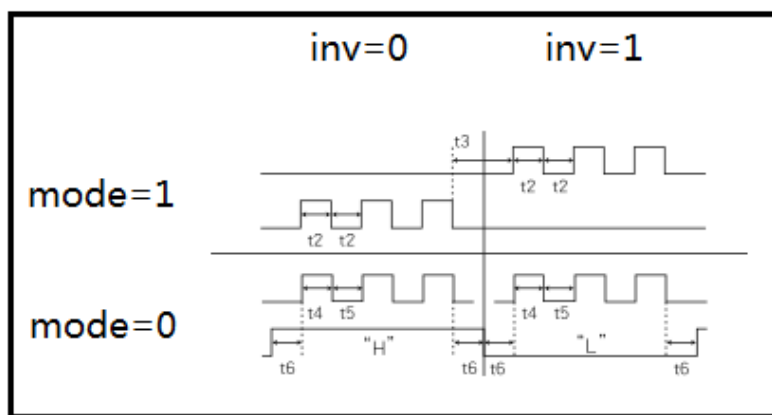
名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posLmt	long*	输出	正向软件限位设置值，单位为脉冲数
negLmt	long*	输出	负向软件限位设置值，单位为脉冲数

(11) 设置脉冲输出模式，默认不取反、输出模式为脉冲+方向

[NMC_MtSetStepMode\(HAND axisHandle, short inv, short mode\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
inv	short	输入	输出取反，0 为不取反，1 为取反
mode	short	输入	输出模式，0 为脉冲+方向，1 为正负脉冲

如下图所示：



(12) 读取脉冲输出模式

[NMC_MtGetStepMode\(HAND axisHandle, short * pInv, short*pMode\);](#)

参考：NMC_MtSetStepMode;

(13) 编码器模式配置

[NMC_SetEncMode\(HAND devHandle, unsigned char encId, unsigned short encMode\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
encId	unsigned char	输入	编码器通道序号，对应轴号, 取值 0~N;
encMode	unsigned short	输入	内部脉冲计数：encMode = 0x0100; 外部编码器：encMode = 0x0000; 外部编码器取反：encMode = 0x1000; 其余模式参考以下说明： 16 位编码器的模式字；

			<p>Bit7:0 分频系数, 数值为 $1 \sim 255$。对应分频数值 $1 \sim 256$;</p> <p>Bit9:8 信号源选择;</p> <p>00: 外部信号输入;</p> <p>01: 轴脉冲输入;</p> <p>10: 自动产生信号(正脉冲);</p> <p>11: 自动产生信号(负脉冲);</p> <p>Bit11:10 信号类型(外部);</p> <p>00: AB 相, 度差;</p> <p>01: 脉冲+方向;</p> <p>10: 正脉冲+负脉冲;</p> <p>11: 保留;</p> <p>Bit12 输入 A、B 交换(外部);</p> <p>0: 不交换, 1: 交换;</p> <p>Bit13 输入 A 取反(外部);</p> <p>0: 不取反, 1: 取反;</p> <p>Bit14 输入 B 取反(外部);</p> <p>0: 不取反, 1: 取反;</p> <p>Bit15 编码器饱和;</p> <p>0: 最大最小值翻转, 1: 不翻转;</p>
--	--	--	---

注意: 通常使用内部计数模式 $encMode = 0x0100$, 或者外部计数模式 $encMode = 0x0000$, 外部计数取反时 $encMode = 0x1000$, 可查看[编码器模式设置示例](#);

(14) 读取编码器模式

[NMC_GetEncMode\(HAND devHandle, unsigned char encId, unsigned short *encMode\);](#)

参考: [NMC_SetEncMode;](#)

(15) 轴运动安全参数配置

[NMC_MtSetSafePara\(HAND axisHandle, TSafePara*pPara\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPara	TSafePara*	输入	<p>安全相关的参数。</p> <pre>typedef struct { double estpDec; double maxVel; double maxAcc; }TSafePara;</pre> <p>estpDec: 急停减速度, 单位是脉冲/ms*ms, 不设置默认为 32767</p> <p>maxVel: 最大允许速度, 单位是脉冲/ms, 超过此速度的设置, 内部使用此速度, 不设置时默认为 32767</p> <p>maxAcc: 最大允许加速度, 单位是脉冲/ms*ms, 超过此加速度的设置, 内部使用此加速度, 不设置时默认为 32767</p>

(16) 读取轴安全参数

[NMC_MtGetSafePara\(HAND axisHandle, TSafePara*pPara\);](#)

参考: [NMC_MtSetSafePara;](#)

(17) 设置限位允许的位置误差

[NMC_MtSetPosErrLmt\(HAND axisHandle, long posErr\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posErr	long	输入	<p>允许的位置误差</p> <p>当位置误差超过设定值时, 轴状态置起 BIT_AXIS_POSERR 位置超限, 闭环模式下会</p>

			停止轴运动 posErr 为 0 表示不检查
--	--	--	------------------------

(18) 读取限位允许的位置误差

[NMC MtGetPosErrLmt\(HAND axisHandle, long *posErr\);](#)

参考: [NMC MtSetPosErrLmt](#)

(19) 设置轴速度的滤波参数, 滤波, 增加速度平滑, 减少抖动

[NMC MtSetAxisVelFilter\(HAND axisHandle, short filterCoef\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
filterCoef	short	输入	设置轴速度的滤波参数 filterCoef 系数范围[0,5], 单位 ms, 值越大, 速度越平滑

(20) 读取轴速度的滤波参数

[NMC MtGetAxisVelFilter\(HAND axisHandle, short *filterCoef\);](#)

参考: [NMC MtSetAxisVelFilter](#)

(21) 设置脉冲输出滤波

[NMC MtSetStepFilter\(HAND axisHandle, unsigned short coe\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
coe	unsigned short	输入	coe 系数: 范围 0~65535, 0 不滤波, 数值越大滤波效果越明显, 默认值为 0

注意: 脉冲输出受到干扰时可以尝试使用该指令;

(22) 读取脉冲输出滤波

[NMC MtGetStepFilter\(HAND axisHandle, unsigned short *pCoe\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

axisHandle	HAND	输入	轴句柄
coe	unsigned short*	输出	coe 系数: 范围 0~65535, 0 不滤波, 数值越大滤波效果越明显, 默认值为 0

(23) 设置单轴急停 DI

[NMC MtSetEstopDI \(HAND axisHandle, short gpiIndex, short sense\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
gpiIndex	short	输入	通用输入序号, 取值范围[0, n], 设置为 -1, 则表示取消急停 DI
sense	short	输入	触发电平, 0: 低电平, 1: 高电平

(24) 读取单轴急停 DI

[NMC MtGetEstopDI \(HAND axisHandle, short *gpiIndex, short* sense\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
gpiIndex	Short*	输出	gpiIndex: 通用输入序号[0, N]
sense	short*	输出	触发电平, 0: 低电平, 1: 高电平

(25) 读取 Ex 单轴急停 DI

[NMC GetEstopDIEx \(HAND devHandle, TEstopExParam *pParam, short group\);](#)

参考: [NMC SetEstopDIEx](#)

(26) 设置 Ex 单轴急停 DI, 触发后会置起 BIT_AXIS_ESTP 标志位

[NMC SetEstopDIEx \(HAND devHandle, TEstopExParam *pParam, short group\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pParam	TEstopExParam *	输入	typedef struct { short axMask; // 轴掩码

			<pre> short diType; // DI 类型, 0-通用输入, 1-负限位, 2-正限位, 3-原点, 4-Z 相 short diIndex; //通道序号 [0,] short filter; // 滤波系数, 取值范围[0, 255] long diSense; //触发电平, 0: 低电平, 1: 高电平 long reserved[3]; // 保留 }TEstopExParam; </pre>
group	short	输入	通道号, 取值[0, ESTOP_DI_EX_CH_NUM-1]

注意：单轴高级急停触发后，规划位置与实际位置可能存在差值；

(27)清除高级单轴急停 DI

[NMC_ClrEstopDIEx \(HAND devHandle, short group\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	通道号, 取值[0, ESTOP_DI_EX_CH_NUM-1]

(28)设置轴补偿参数

[NMC_MtSetAxisDampCompParam\(HAND axisHandle, short enable, short param1, short param2\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	控制器句柄
enable	short	输入	0: 取消 1: 使能
param1	short	输入	数值范围在(10~500), 建议先从大到小进行设置测试
param2	short	输入	数值为控制频率 HZ

(29)设置超限报警的动作

[NMC_MtSetPosErrAction\(HAND axisHandle,short actionsBits\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	控制器句柄
actionsBits	short	输入	轴超限报警后的处理方法，按位表示，默认为 0，即只置起轴超限标志位 //bit0: 该位为 1 时，关闭使能 //bit1: 该位为 1 时，停止轴运动 //其他保留

(30) 读取超限报警的动作

[NMC_MtGetPosErrAction\(HAND axisHandle,short *pActionsBits\);](#)

参考: [NMC_MtSetPosErrAction;](#)

3.3 综合示例

```
// 专用IO，脉冲滤波和编码器的配置等使用
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    rtn = NMC_MtSetLmtCfg(axisHandle, 1, 1, 0, 0); //设置轴限位有效，低电平触发
    gc_rtn_error(rtn);
    rtn = NMC_MtSetAlarmCfg(axisHandle, 1, 0); //设置轴1的轴报警有效，低电平触发
    gc_rtn_error(rtn);
    rtn = NMC_MtSetStepMode(axisHandle, 0, 0); //设置轴1的脉冲方式为脉冲加方向，不取反
    gc_rtn_error(rtn);
    rtn = NMC_SetEncMode(axisHandle, 0, 0x0000); //设置编码器读取数为外部计数，如果没接编码器
    信号到控制器，此处应该为内部计数
    gc_rtn_error(rtn);
    rtn = NMC_MtSetAxisVelFilter(axisHandle, 1); //有轴运行抖动，可稍微加一些轴速度滤波或者
    脉冲输出滤波
    gc_rtn_error(rtn);
    rtn = NMC_MtClrError(axisHandle);
    gc_rtn_error(rtn);
    return rtn;
}
```

//编码器模式设置

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    short source;    // 0:内部脉冲计数    1:外部编码器输入
    short dir;       // 0:AB相90度差    1:脉冲+方向    2:正脉冲+负脉冲
                    // 3:AB相90度差(负逻辑) 4:脉冲+方向(负逻辑) 5:正脉冲+负脉冲(负逻辑)
    short encMode;   // 编码器模式
    switch (source)
    {
    case 0:
        switch (dir)
        {
        case 0:
            encMode = 256; break;
        case 1:
            encMode = 1280; break;
        case 2:
            encMode = 2304; break;
        case 3:
            encMode = 4352; break;
        case 4:
            encMode = 17664; break;
        case 5:
            encMode = 6400; break;
        default: break;
        } break;
    case 1:
        switch (dir)
        {
        case 0:
            encMode = 0; break;
        case 1:
            encMode = 1024; break;
        case 2:
            encMode = 2048; break;
        case 3:
            encMode = 4096; break;
        case 4:
            encMode = 17408; break;
        case 5:
            encMode = 6144; break;
        default: break;
        } break;
    }
```

```
default:break;  
}  
rtn = NMC_SetEncMode(devHandle, axisNum, encMode); // 设置编码器模式  
gc_rtn_error(rtn);  
return rtn;  
}
```


4 控制器的状态检测

4.1 功能介绍

控制器的状态检测方便我们实时读取控制器的各种状态，包括为控制器的轴状态，轴位置，轴速度等；

轴状态字的定义表		
位	宏定义	说明
Bit0	<code>BIT_AXIS_BUSY = 0x00000001</code>	该位为 1 表示：运动；0 表示：静止
Bit1	<code>BIT_AXIS_POSREC = 0x00000002</code>	该位为 1 表示：伺服位置到达(实际位置与规划位置相等)，该位与到位误差设置有关 NMC MtSetAxisArrivalPara (误差值默认为 0)，步进电机请选择内部编码器
Bit2	<code>BIT_AXIS_MVERR = 0x00000004</code>	该位为 1 表示：上次运动出错，或当前无法启动运动，需要执行 NMC MtClrError
Bit3	<code>BIT_AXIS_SVON = 0x00000008</code>	该位为 1 表示：伺服使能
Bit4	<code>BIT_AXIS_CRD = 0x00000010</code>	该位为 1 表示：坐标系模式
Bit5	<code>BIT_AXIS_STEP = 0x00000020</code>	该位为 1 表示：步进；0 表示：伺服
Bit6	<code>BIT_AXIS_LMTP = 0x00000040</code>	该位为 1 表示：正向限位触发
Bit7	<code>BIT_AXIS_LMTN = 0x00000080</code>	该位为 1 表示：负向限位触发
Bit8	<code>BIT_AXIS_SOFTPOSLMT = 0x00000100</code>	该位为 1 表示：正向软限位触发
Bit9	<code>BIT_AXIS_SOFTNEGLMT = 0x00000200</code>	该位为 1 表示：负向软限位触发
Bit10	<code>BIT_AXIS_ALM = 0x00000400</code>	该位为 1 表示：伺服报警，需要执行 NMC MtClrError 才能运动
Bit11	<code>BIT_AXIS_POSERR = 0x00000800</code>	该位为 1 表示：位置超限，需要执行 NMC MtClrError 才能运动
Bit12	<code>BIT_AXIS_ESTP = 0x00001000</code>	该位为 1 表示：急停触发，需要执行 NMC MtClrError 才能运动
Bit13	<code>BIT_AXIS_HWERR = 0x00002000</code>	该位为 1 表示：硬件错误
Bit14	<code>BIT_AXIS_CAPTSET = 0x00004000</code>	该位为 1 表示： 捕获触发

4.2 指令说明

(1) 读取轴状态

[NMC_MtGetSts\(HAND axisHandle, short *pStsWord\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pStsWord	short*	输出	返回轴状态字，具体位定义见 轴状态字定义表

(2) 快速读取轴状态 (适合 PCIE 卡)

[NMC_MtGetStsEx\(HAND axisHandle, short *pStsWord\);](#)

参考: [NMC_MtGetSts](#);

注: [NMC_MtGetStsEx](#) 比 [NMC_MtGetSts](#) 执行时间快 7 倍左右;

(3) 清除轴的错误状态

[NMC_MtClrError\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(4) 清除轴的错误状态, 按位清除

[NMC_MtClrStsByBits\(HAND axisHandle, short stsMask\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
stsMask	Short	输入	对应位为 1 表明需要清除对应位的错误状态

(5) 读取轴的规划位置

[NMC_MtGetPrfPos\(HAND axisHandle, long *pos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pos	long *	输出	轴规划位置，单位是脉冲

(6) 读取轴的规划速度

[NMC MtGetPrfVel \(HAND axisHandle, double *pVel\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pVel	double *	输出	轴规划速度，单位是脉冲/ms

(7) 读取轴的机械位置

[NMC MtGetAxisPos \(HAND axisHandle, long *pos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pos	long *	输出	轴机械位置，单位是脉冲

(8) 读取轴的命令位置

[NMC MtGetCmdPos \(HAND axisHandle, long *pos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pos	long *	输出	轴命令位置，单位是脉冲 命令位置指控制器发送到执行器的命令脉冲数

(9) 读取当前轴编码器通道位置

[NMC MtGetEncPos \(HAND axisHandle, long *pPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	控制器句柄
pPos	long *	输出	返回编码器数值

(10) 按序号读取编码器通道的速度，当读取第一个辅助编码器速度时 encId=256

[NMC GetEncVel \(HAND devHandle, unsigned char encId, double *vel\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

encId	unsigned char	输入	编码器通道 ID, 取值 0~N
Vel	double *	输出	编码器速度(单位是: 脉冲/ms)

(11) 打包读取 12 个轴的状态

[NMC_MtGetStsPack12Ex\(HAND axisFirstHandle, TAxisStsPack12Ex *pPackSts \);](#)

名称	数据类型	输入/输出	描述
axisFirstHandle	HAND	输入	第一个轴句柄
pPackSts	TAxisStsPack12Ex*	输出	<p>打包的状态数据, 参考结构体定义</p> <p>该指令一次性读取 12 个轴的常见状态, 对于 PC-Base 的开发, 可以大大提高通讯效率。</p> <pre>typedef struct{ short mtSts[12]; // 单轴状态, 位定义参考 NMC_MtGetSts short mtMio[12]; // 单轴专用 IO, 位定义参考 NMC_MtGetMotionIO short mtMioLog[12]; // 单轴专用 IO: 逻辑电平, 位定义参 NMC_MtGetMotionIOLogical long mtPrfPos[12]; // 单轴规划位置 long mtEncPos[12]; // 单轴实际位置 float mtPrfVel[12]; // 单轴规划速度 short crdSts[2]; // 坐标系状态 float crdPrfVel[2]; // 坐标系运动速度 long crdUserSeg[2]; // 坐标系运行的缓冲区段号 long crdFreeSpace[2]; // 坐标系缓冲区剩余空间 long crdUsedSpace[2]; // 坐标系缓冲区使用空间 long gpi; // 通用输入</pre>

			<pre> long gpo; // 通用输出 long crdLeftLen[2]; // 坐标系缓冲区剩余段 长, 单位:脉冲 long crdAllCmdCnt[2]; // 坐标系缓冲区总 共压入的指令数目 long extDi[6]; // 扩展模块输入, 2 ~8 号站 long extDo[6]; // 扩展模块输出, 2 ~8 号站 short adc[4]; // 模拟量输入值 0~4 通道 short adcAux[2]; // 扩展模拟量输入值 0~1 通道 short dac[4]; // 模拟量输出值 0~4 通道 short dacAux[2]; // 扩展模拟量输出值 0~1 通道 long reserved[10]; // 预留 }TAxisStsPack12Ex; </pre>
--	--	--	--

注意：打包指令数据获取比较多，耗时长，若非必须，请用单轴获取 [NMC MtGetSts](#);

4.3 综合示例

```

//读取轴状态，位操作，轴位置，轴速度等
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    short axsiSts = 0;
    rtn = NMC_MtGetSts(axisHandle, &axsiSts); //读取轴状态
    gc_rtn_error(rtn);
    bool IsRuning = (axsiSts & (1 << 0)) == 0 ? false : true; //取bit0, 0静止, 1运动
    bool IsArrive = (axsiSts & (1 << 1)) == 0 ? false : true; //取bit1, 位置到达, 0, 未到达, 1到达
    bool IsError = (axsiSts & (1 << 2)) == 0 ? false : true; //取bit2, 运动是否出错, 0, 未出错, 1, 出错
    bool IsAxisOn = (axsiSts & (1 << 3)) == 0 ? false : true; //取bit3, 是否使能中
    bool PosArrived = (axsiSts & (1 << 6)) == 0 ? false : true; //取bit6, 正向限位是否触发
    bool NegArrived = (axsiSts & (1 << 7)) == 0 ? false : true; //取bit7, 负向限位是否触发
    bool PosSoftArrived = (axsiSts & (1 << 8)) == 0 ? false : true; //取bit8, 正向软限位是

```

否触发

```
bool NegSoftArrived = (axsiSts & (1 << 9)) == 0 ? false : true; //取bit9, 负向软限位是
```

否触发

```
bool IsAlarming = (axsiSts & (1 << 10)) == 0 ? false : true; //取bit10, 驱动器是否报警
```

```
bool IsPosErr = (axsiSts & (1 << 11)) == 0 ? false : true; //取bit11, 位置是否超过误差
```

极限

```
rtn = NMC_MtClrError(axisHandle); // 轴运动之前, 或者轴报错, 调用清除轴状态的指令
```

```
gc_rtn_error(rtn);
```

```
/******分割线*****
```

```
/*****下面读取为单轴的状态, 要同时读取多个轴的状态, 可使用for循环操作*****/
```

```
//读取轴1规划位置和实际位置, 规划速度和实际速度
```

```
long prfpos = 0;
```

```
long encpos = 0;
```

```
double prfvel = 0;
```

```
double encvel = 0;
```

```
unsigned short encId = 256;
```

```
rtn = NMC_MtGetPrfPos(axisHandle, &prfpos); //规划位置
```

```
gc_rtn_error(rtn);
```

```
rtn = NMC_MtGetEncPos(axisHandle, &encpos); //实际位置
```

```
gc_rtn_error(rtn);
```

```
rtn = NMC_MtGetPrfVel(axisHandle, &prfvel); //规划速度
```

```
gc_rtn_error(rtn);
```

```
rtn = NMC_GetEncVel(devHandle, encId, &encvel); //实际速度
```

```
gc_rtn_error(rtn);
```

```
/******分割线*****
```

```
TAxisStsPack12Ex stspack; // 控制器所有轴信息打包结构体
```

```
rtn = NMC_MtGetStsPack12Ex(axisHandle, &stspack); //一次性读取所有轴信息, 需要的时间会比较长
```

```
gc_rtn_error(rtn);
```

```
return rtn;
```

```
}
```

5 点位和JOG运动

5.1 功能介绍

- (1)控制器的每个轴都可以设置为单轴运动模式;
- (2)单轴运动的速度规划类型可以设置为点位运动(PTP, 位置的正负决定运动方向), 恒速运动(JOG, 速度的正负决定运动方向);
- (3)在单轴运动模式下, 各轴可以独立设置目标位置(JOG 运动不需要目标位置)、目标速度、加速度、减速度、起跳速度、停止速度等运动参数, 能够独立运动或停止;
- (4)PTP 运动或者 JOG 运动必要时可增加平滑系数, 使得加减速过程过渡更为平滑, 以减小机台振动, 如下图(v-t 图);

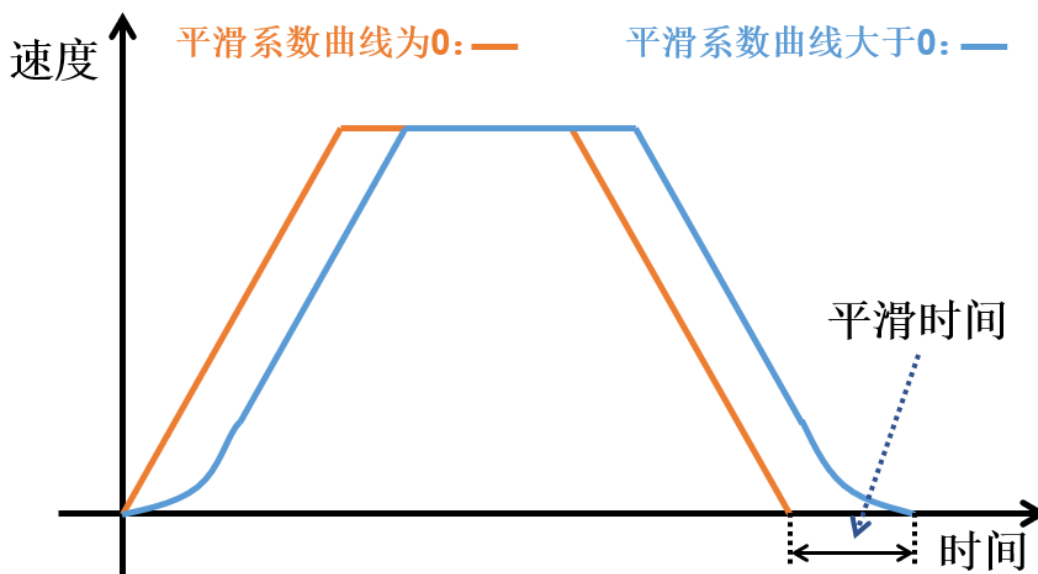


图 5.1.1 平滑系数对比曲线图

5.2 指令说明

- (1)设置单轴运动的规划模式

[NMC_MtSetPrfMode\(HAND axisHandle, short mode\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mode	short	输入	规划模式 // 各轴的规划模式 <code>#define MT_NONE_PRF_MODE (-1) // 无效</code> <code>#define MT_PTP_PRF_MODE (0) // 梯形规划</code>

			<pre> #define MT_JOG_PRF_MODE (1) //连续速度模式 #define MT_CRD_PRF_MODE (3) // 坐标系 #define MT_GANTRY_MODE (4) //龙门跟随模式 #define MT_PT_PRF_MODE (5) // PT 模式 #define MT_MULTI_LINE_MODE (6) // 多轴直线插补 #define MT_GEAR_PRF_MODE (7) // 电子齿轮模式 #define MT_FOLLOW_PRF_MODE (8) //Follow 跟随模式 </pre>
--	--	--	---

注意：坐标系模式不需要通过 `NMC_MtSetPrfMode` 设置；

(2) 读取单轴运动的规划模式

`NMC_MtGetPrfMode(HAND axisHandle, short*mode);`

参考：`NMC_MtSetPrfMode`

(3) 设置单轴 PTP 运动参数

`NMC_MtSetPtpPara(HAND axisHandle, TPtpPara *pAxPara);`

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pAxPara	TPtpPara *	输入	设置单轴点位运动参数，结构体定义如下 <pre> Typedef struct{ double acc; double dec; double startVel; double endVel; short smoothCoef; short reserved[3]; }TPtpPara; </pre> acc:加速度, 脉冲/ms ² dec:减速度, 脉冲/ms ² startVel: 起跳速度, 脉冲/ms

			endVel: 终止速度, 脉冲/ms smoothCoef: 平滑系数, [0, 199], 单位 ms
--	--	--	--

注意: PTP 指单轴点到点的运动;

(4) 获得单轴 PTP 运动参数

[NMC_MtGetPtpPara\(HAND axisHandle, TPtpPara *pAxPara\);](#)

参考: [NMC_MtGetPtpPara](#)

(5) 设置点到点运动的目标位置

[NMC_MtSetPtpTgtPos\(HAND axisHandle, long pos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pos	long	输入	设置运动目标位置, 单位脉冲 仅用于 PTP 模式

(6) 读取点到点运动的目标位置

[NMC_MtGetPtpTgtPos\(HAND axisHandle, long*pos\);](#)

参考: [NMC_MtSetPtpTgtPos](#)

(7) 设置目标速度

[NMC_MtSetVel\(HAND axisHandle, double vel\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
vel	double	输入	设置运动目标速度, 单位脉冲/ms 只针对 PTP 和 Jog: PTP 模式下只接受正数, Jog 模式下正负号标识运动方向

(8) 读取目标速度

[NMC_MtGetVel\(HAND axisHandle, double *vel\);](#)

参考: [NMC_MtSetVel;](#)

(9) 设置单轴 JOG(连续速度模式) 运动参数

[NMC_MtSetJogPara\(HAND axisHandle, TJogPara *pAxPara\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pAxPara	TJogPara *	输入	设置单轴 Jog 参数，结构体定义如下 <pre>typedef struct{ double acc; double dec; double smoothCoef; }TJogPara;</pre> acc: 加速度, 脉冲/ms ² dec: 减速度, 脉冲/ms ² smoothCoef: 平滑系数, [0, 199], 单位 ms

(10) 读取单轴 JOG(连续速度模式) 运动参数

[NMC_MtGetJogPara\(HAND axisHandle, TJogPara *pAxPara\);](#)

参考: [NMC_MtSetJogPara](#)

(11) 更新运动参数，只针对 PTP 和 Jog

[NMC_MtUpdate\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(12) 多轴同时更新运动参数，只针对 PTP 和 Jog

[NMC_MtUpdateMulti\(HAND axisHandle, long mask\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

mask	long	输入	按 bit 对应相应轴号, bit 为 1 表示启动, bit 为 0 表示不启动
------	------	----	---

(13) 立即停止运动, 根据用户设置的减速度进行停止, 如果用户没有设置则默认为最大值

[NMC MtStop\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(14) 单轴急停, 不会置起急停标志位, 根据用户设置的急停减速度进行停止, 如果用户没有设置则默认为最大值

[NMC MtAbruptStop\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(15) 多轴立即停止运动, 根据用户设置的减速度进行停止, 如果用户没有设置则默认为最大值

[NMC MtStopMulti\(HAND axisHandle, long mask\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mask	long	输入	按 bit 对应相应轴号, bit 为 1 表示需要停止, bit 为 0 表示不需要停止

(16) 急停, 根据用户设置的急停加速度进行停止, 如果用户没有设置则默认为最大值

[NMC MtEstop\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

注意: 调用急停后会将轴的[急停状态字](#)置起, 需调用[NMC MtClrError](#)清除状态才能继续运动;

(17) 单轴点位运动 (绝对位置, 目标位置的正负代表运动方向)

[NMC MtMovePtpAbs\(HAND axisHandle, double acc, double dec, double startVel, double](#)

[endVel, double maxVel, short smoothCoef, long tgtPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
acc	double	输入	加速度, 脉冲/ ms^2
dec	double	输入	减速度, 脉冲/ ms^2
startVel	double	输入	起跳速度, 脉冲/ms
endVel	double	输入	终止速度, 脉冲/ms
maxVel	double	输入	最大速度, 脉冲/ms
smoothCoef	short	输入	平滑系数, [0, 199], 单位 ms
tgtPos	long	输入	目标位置, 单位: 脉冲

(18) 单轴点位运动 (相对位置, 目标位置的正负代表运动方向)

[NMC_MtMovePtpRel \(HAND axisHandle, double acc, double dec, double startVel, double endVel, double maxVel, short smoothCoef, long relPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
acc	double	输入	加速度, 脉冲/ ms^2
acc	double	输入	减速度, 脉冲/ ms^2
startVel	double	输入	起跳速度, 脉冲/ms
endVel	double	输入	终止速度, 脉冲/ms
maxVel	double	输入	最大速度, 脉冲/ms
smoothCoef	short	输入	平滑系数, [0, 199], 单位 ms
relPos	long	输入	目标位置, 单位: 脉冲

(19) 单轴点位运动打包 (绝对运动)

[NMC_MtMovePtpAbsPack8 \(HAND axisFirstHandle, TMovePtpPack8 * pPackData\);](#)

名称	数据类型	输入/输出	描述
axisFirstHandle	HAND	输入	轴句柄
pPackData	double	输入	打包参数

			<pre>typedef struct{ short axisMask; // 轴掩码，对应 bit 为 1 表示该轴参与运动 short clrStsFlag; // 是否运动前先 清除轴状态，0：不清除，1：清除 short reserved[2]; // 保留 double acc[8]; // 加速度 double dec[8]; // 减加速度 double startVel[8]; // 起跳速度 double endVel[8]; // 终止速度 double maxVel[8]; // 最大速度 short smoothCoef[8]; // 平滑系数 long tgtPos[8]; // 位置，单位：脉冲 } TMovePtpPack8;</pre>
--	--	--	--

注意：指令适用于连续 8 个轴中的任意轴；

(20) 单轴点位运动打包(相对运动)

[NMC MtMovePtpRelPack8\(HAND axisFirstHandle, TMovePtpPack8 * pPackData\);](#)

名称	数据类型	输入/输出	描述
axisFirstHandle	HAND	输入	轴句柄
pPackData	double	输入	打包参数 <pre>typedef struct{ short axisMask; // 轴掩码，对应 bit 为 1 表示该轴参与运动 short clrStsFlag; // 是否运动前 先清除轴状态，0：不清除，1：清除 short reserved[2]; // 保留 double acc[8]; // 加速度 double dec[8]; // 减加速度</pre>

			<pre>double startVel[8]; // 起跳速度 double endVel[8]; // 终止速度 double maxVel[8]; // 最大速度 short smoothCoef[8]; // 平滑系数 long tgtPos[8]; // 位置, 单位: 脉冲 }TMovePtpPack8;</pre>
--	--	--	---

注意：指令适用于连续 8 个轴中的任意轴；

(21) 单轴连续速度运动(速度的正负代表运动方向)

[NMC MtMoveJog\(HAND axisHandle, double acc, double dec, double maxVel, short smoothCoef, short clrStsFlag\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
acc	double	输入	加速度, 脉冲/ms ²
dec	double	输入	减速度, 脉冲/ms ²
maxVel	double	输入	最大速度, 脉冲/ms
smoothCoef	short	输入	平滑系数, [0, 199], 单位 ms
clrStsFlag	short	输入	是否运动前先清除轴状态, 0: 不清除, 1: 清除

(22) 设置 PTP 位置比较输出

[NMC MtSetPtpComparePara\(HAND axisHandle, short dir, long comparaPos, double upDateVel\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
dir	short	输入	0: 负向, 1: 正向
comparaPos	long	输入	目标位置, 单位是脉冲
upDateVel	double	输入	比较后的更新速度 pulse/ms

(23) 读取 PTP 比较输出状态

[NMC_MtGetPtpCompareSts\(HAND axisHandle, short *pIsCompareActive, short *pIsCompareOut\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pIsCompareActive	short *	输出	目标位置，单位是 脉冲
pIsCompareOut	short *	输出	比较后的更新速度 pulse/ms

5.3 综合示例

5.3.1 点位运动

```
// 单轴点位PTP运动
short gc_example_ptp(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    rtn = NMC_MtSetPrfMode(axisHandle, MT_PTP_PRF_MODE); // 模式配置：将指定轴配置为点到点的运动模式
    gc_rtn_error(rtn);
    TtpPara ptpPrm;
    ptpPrm.acc = 0.5; //加速度
    ptpPrm.dec = 0.5; //减加速度
    ptpPrm.endVel = 0; //结束速度
    ptpPrm.startVel = 0; //起跳速度
    ptpPrm.smoothCoef = 0; //平滑系数
    rtn = NMC_MtSetPtpPara(axisHandle, &ptpPrm); // 运动参数配置
    gc_rtn_error(rtn);
    double maxVel = 1;
    rtn = NMC_MtSetVel(axisHandle, maxVel); // 设置规划的最高速度
    gc_rtn_error(rtn);
    long targetPos = 10000;
    rtn = NMC_MtSetPtpTgtPos(axisHandle, targetPos); // 设置目标位置
    gc_rtn_error(rtn);
    rtn = NMC_MtUpdate(axisHandle); // 启动运动
    gc_rtn_error(rtn);
    /***** 运动过程控制 *****/
    short axisSts;
    long axisPos;
    while (true)
    {
        rtn = NMC_MtGetSts(axisHandle, &axisSts); //读取轴状态标志位
    }
}
```

```
gc_rtn_error(rtn);
if ((axisSts & BIT_AXIS_BUSY) == 0)
{
    break;    // 运动完成
}

rtn = NMC_MtGetAxisPos(axisHandle, &axisPos); // 读取电机当前的理论位置
gc_rtn_error(rtn);
if (axisPos > 2000)                          // 位置大于2000时，更新速度
{
    maxVel = 5;
    rtn = NMC_MtSetVel(axisHandle, maxVel);
    gc_rtn_error(rtn);
    rtn = NMC_MtUpdate(axisHandle); // 更新速度
    gc_rtn_error(rtn);
}
}
return rtn;
}
```

5.3.2 点位运动(绝对位置)

```
// 点位运动(绝对位置)，只使用一条指令就可以完成点位运动(绝对位置)功能
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    double acc = 0.5;                //加速度
    double dec = 0.5;                //减加速度
    double endVel = 0;                //结束速度
    double startVel = 0;              //起跳速度
    double maxVel = 10;               //最大速度
    double smoothCoef = 0;            //平滑系数
    long tgtPos = 10000;              //目标位置
    rtn = NMC_MtMovePtpAbs(axisHandle, acc, dec, startVel, endVel, maxVel, smoothCoef, tgtPos);
    gc_rtn_error(rtn);
    return rtn;
}
```

5.3.3 点位运动(相对位置)

```
// 点位运动(相对运动)，只使用一条指令就可以完成点位运动(相对运动)功能
short gc_example(void)
{
```



```
/****** 已省略打开控制器部分，统一描述在第五章1 ******/  
double acc = 0.5; //加速度  
double dec = 0.5; //减加速度  
double endVel = 0; //结束速度  
double startVel = 0; //起跳速度  
double maxVel = 10; // 最大速度  
double smoothCoef = 0; //平滑系数  
long relPos = 10010; //相对目标位置  
rtn = NMC_MtMovePtpRel(axisHandle, acc, dec, startVel, endVel, maxVel, smoothCoef, relPos);  
gc_rtn_error(rtn);  
return rtn;  
}
```

5.3.4 JOG运动1

```
// JOG运动运行过程  
short gc_example(void)  
{  
    /****** 已省略打开控制器部分，统一描述在第五章1 ******/  
    rtn = NMC_MtSetPrfMode(axisHandle, MT_JOG_PRF_MODE); //模式配置：将指定轴配置为JOG模式  
    gc_rtn_error(rtn);  
    TJogPara jogPrm;  
    jogPrm.acc = 0.5; //加速度  
    jogPrm.dec = 0.5; //减加速度  
    jogPrm.smoothCoef = 0; //平滑系数  
    rtn = NMC_MtSetJogPara(axisHandle, &jogPrm); // 运动参数配置  
    gc_rtn_error(rtn);  
    double maxVel = 5;  
    rtn = NMC_MtSetVel(axisHandle, maxVel); // 设置规划的最高速度  
    gc_rtn_error(rtn);  
  
    rtn = NMC_MtUpdate(axisHandle); // 启动运动  
    gc_rtn_error(rtn);  
    /** 运动过程控制 **/  
    short velUpdateFlag = 0;  
    long axisPos;  
    while (true)  
    {  
        rtn = NMC_MtGetAxisPos(axisHandle, &axisPos); // 读取电机当前的理论位置  
        gc_rtn_error(rtn);  
        if ((axisPos > 10000) && (velUpdateFlag == 0)) // 位置大于10000时，更新速度  
        {  
            velUpdateFlag = 1;  
        }  
    }  
}
```

```
        maxVel = 1;
        rtn = NMC_MtSetVel(axisHandle, maxVel);    // 设置规划的最高速度
        gc_rtn_error(rtn);
        rtn = NMC_MtUpdate(axisHandle);           // 更新速度
        gc_rtn_error(rtn);
    }
    if (axisPos > 20000)                            // 位置大于20000时，停止运动
    {
        rtn = NMC_MtStop(axisHandle);              // 停止运动
        gc_rtn_error(rtn);
        break;
    }
}
return rtn;
}
```

5.3.5 JOG运动2

```
// JOG运动之一条指令完成
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    double acc = 0.5;
    double dec = 0.5;
    double maxVel = 5;
    short smoothCoef = 0;
    short clrStsFlag = 0;
    rtn = NMC_MtMoveJog(axisHandle, acc, dec, maxVel, smoothCoef, clrStsFlag);
    gc_rtn_error(rtn);
    /*** 运动过程控制 省略 ***/
    return rtn;
}
```

6 回零运动

6.1 功能介绍

- (1) 以下整合了各种运动的回零方式，只需要调用 [NMC_MtSetHomePara](#) 和 [NMC_MtHome](#) 即可实现回零功能；[NMC_MtSetHomePara](#) 用于设置回零参数，[NMC_MtHome](#) 用于启动回零；
- (2) 对于回零参数的设置，不熟悉的用户可先使用我们提供的 GCS 工具先回零，然后将回零界面的参数赋值给 [NMC_MtSetHomePara](#)，回零过程中可以用 [NMC_MtGetHomeSts](#) 查询回零状态，用 [NMC_MtHomeStop](#) 停止回零；
- (3) 回零前，确认轴的方向正确性，即发正脉冲，轴会朝着正限位方向运动；假如正负限位为低电平触发，则对应的回零结构体 THomeSetting 的参数 lmtEdge 应为下降沿触发，反之则为上升沿触发，原点信号用法相同，如下图；



- (4) 如果回零的方向不对，参数设置让其往负向回零，轴却往正向寻找原点，原因可能是[上升沿](#)或者[下降沿](#)触发参数给错导致的，如上图红框；
- (5) 回零应保证轴的规划(命令)位置和实际位置大小相等，方向相同，即配置轴参数时，如果选择编码器是外部触发时，在“轴测试”页面中的规划(命令)位置和实际位置配置为一致，见下图：

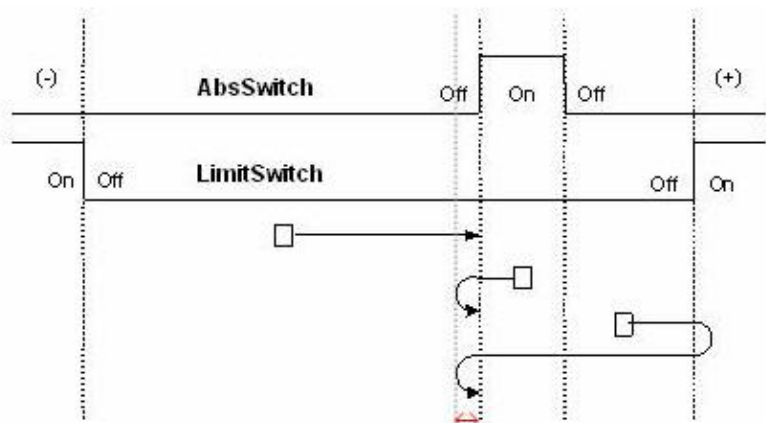


(6) 控制器默认是以脉冲计数作为位置参考，如果使用外部编码器，则需要在回零之前正确设置编码器的工作模式，并执行位置清零；

(7) 回零的功能是不对伺服使能信号进行控制的，如果驱动器需要伺服使能信号，则应在回零前，打开伺服；

回零图示

(1) 以原点开关为参考(MODE1)，一次搜寻原点

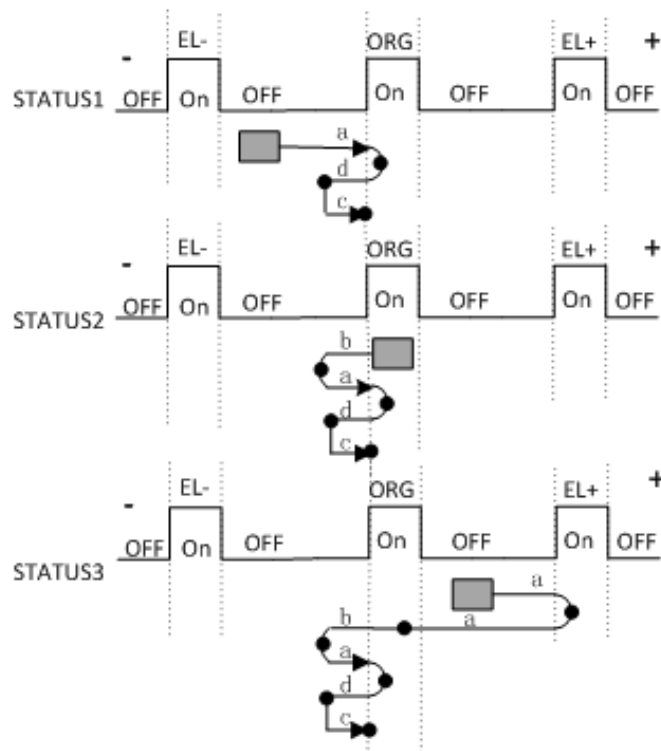


图中方块表示初始位置，曲线表示回零的过程。分别表示了3种不同初始位置时回零的过程；返回时离开触发开关的距离可设置，对应为 `retSwOffset`。如果不设置则是离开触发开关时平滑停止的距离。

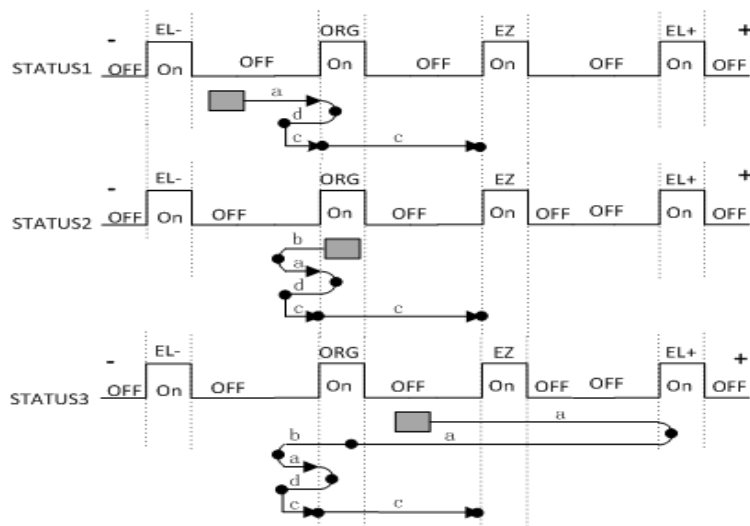
以限位为参考 MODE2 和 MODE1 类似(图略)；

以 Z 相为参考 MODE3 和 MODE1 类似，不支持限位回退(图略)；

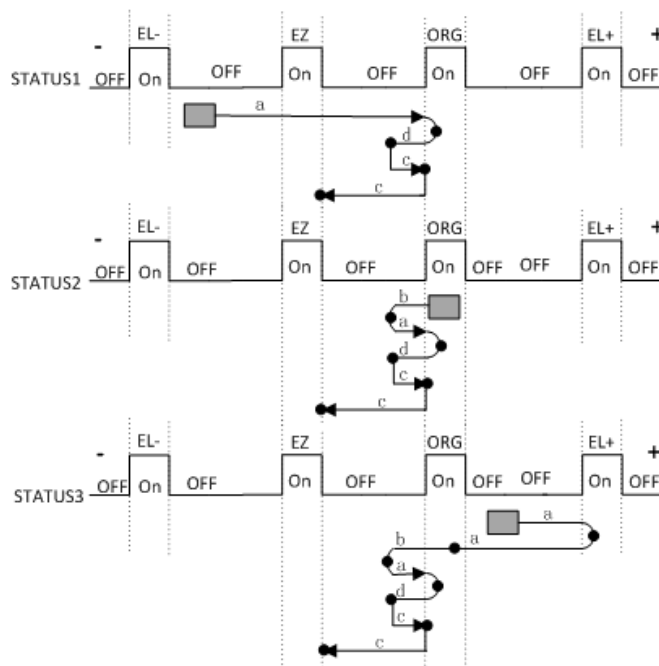
(2) 以原点开关为参考(MODE1)，两次搜寻原点



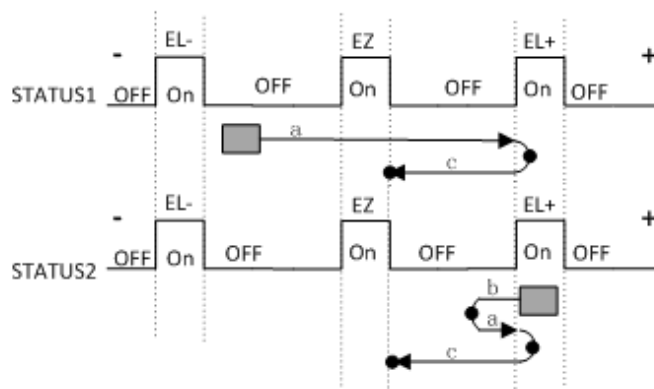
(3) 以原点开关和+Z 相信号为参考(MODE4)，两次搜寻原点



(4) 以原点开关和-Z 相信号为参考(MODE5)，两次搜寻原点



(5) 以限位开关和 Z 相为参考 **MODE6**，一次搜寻原点



注意：选择 MODE1，MODE4，MODE5 回零时，如果先碰到限位会反向继续寻找原点；

6.2 指令说明

(1) 设置回零参数

[NMC MtSetHomePara\(HAND axisHandle, THomeSetting *pHomePara\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pHomePara	THomeSetting *	输入	回零参数结构 ，参见后面的说明

(2) 读取回零参数

[NMC_MtGetHomePara\(HAND axisHandle, THomeSetting *pHomePara\);](#)

参考: [NMC_MtSetHomePara](#)

(3) 启动回零动作，回零完成后轴位置将自动清零

[NMC_MtHome\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

注意: 回零功能为捕获实际编码器位置，若不接 AB 相，[编码器模式](#)选择内部脉冲计数；

(4) 尝试性回零(测试回零误差，不清位置)

[NMC_MtTryHome\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(5) 终止回零动作

[NMC_MtHomeStop\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(6) 读取回零状态

[NMC_MtGetHomeSts\(HAND axisHandle, unsigned short *pStsWord\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pStsWord	unsigned short*	输出	返回状态字。 状态位定义如下： bit0: 该位为 1: 回零中； bit1: 该位为 1: 回零成功； bit2: 该位为 1: 回零失败；

			<p>bit3: 该位为 1: 回零错误, 运动参数出错导致不动;</p> <p>bit4: 该位为 1: 回零错误, 搜寻过程中开关没触发;</p>
--	--	--	---

(7) 读取新回零位置和历史回零位置的差值

[NMC MtGetHomeError\(HAND axisHandle, long *cmdPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
cmdPos	long *	输出	返回位置的差值

回零参数说明

回零参数结构定义:

```
typedef struct {
    short mode;           // 模式, HM_MODE1 ~ HM_MODE6 (必须)
    short dir;            // 搜寻零点方向(必须), 0: 负向, 1: 正向, 其它值无意义
    long offset;          // 原点偏移(必须)
    double scan1stVel;    // 基本搜寻速度(必须)
    double scan2ndVel;    // 低速(回零偏差大时考虑降速), 二次搜寻时需要 (*+Z相回零)
    double acc;           // 加速度
    unsigned char reScanEn; // 是否二次搜寻零点(可选, 不用时设为0)
    unsigned char homeEdge; // 原点触发沿(默认0下降沿触发), 与原点触发电平有关
    unsigned char lmtEdge;  // 限位触发沿(默认0下降沿触发), 与限位触发电平有关
    unsigned char zEdge;    // 限位触发沿(默认0下降沿触发), 与Z向触发电平有关
    unsigned long iniRetPos; // 起始时反向离开当前位置的运动距离(可选, 不用时设为0)
    unsigned long retSwOffset; // 反向运动时离开开关距离(可选, 不用时设为0)
    unsigned long safeLen;  // 安全距离, 回零时最远搜寻距离(可选, 0表示不限制距离)
    unsigned char usePreSetPtpPara; // 1表示需要在回零前, 自己设置回零运动(点到点)的参数
}
```



```

unsigned char reserved[3];    // 保留

long reserved2;              // 保留

} THomeSetting;

```

(1) 基本模式 mode (必须)

模式	硬件	说明	沿/电平
MODE1	单原点	原点开关触发	沿触发
MODE2	单限位	限位开关触发	沿触发
MODE3	单 Z 相	编码器的 Index 信号触发	沿触发
MODE4	原点+ Z 相	原点触发后, 正向寻找到 Index 信号触发	沿触发
MODE5	原点-Z 相	原点触发后, 反向寻找到 Index 信号触发	沿触发
MODE6	限位-Z 相	限位触发后, 反向寻找到 Index 信号触发	沿触发

(2) 搜寻零点方向 dir (必须);

(3) 零点偏移 offset, 回零动作成功后, 坐标原点和硬件触发位置的偏移;

(4) 第一次搜寻零点速度 scan1stVel (必须, 应设置为非零值);

(5) 第二次搜寻零点速度 scan2ndVel (可选, 两次搜寻零点时必须设置为非零值);

(6) 是否两次搜寻零点 reScanEn (可选, 默认一次搜寻零点), 两次回零是指先以设定第一次搜寻速度(scan1stVel)寻找零点信号, 再返回用设定第二次搜寻速度(scan2ndVel)寻找一次;

(7) 初始固定回退距离 iniRetPos (可选, 默认为 0), 32 位无符号数;

(8) 零点开关回退距离 retSwOffset (可选, 两次搜寻零点时必须设置), 第二次回原点过程中碰原点或是碰限位返回时, 离开回零信号触发位置的距离。图中 c 段距离;

(9) acc: 回零运动的加速度;

(10) 原点, 触发沿 homeEdge (默认下降沿, 值为 0, 输入信号为低电平触发)

(11) 限位, 触发沿 lmtEdge (默认下降沿, 值为 0, 输入信号为低电平触发);

(12) Z 相, 触发沿 zEdge (默认下降沿, 值为 0, 输入信号为低电平触发);

(13) 扫描安全距离 safeLen (可选, 默认为无穷大);

(14) usePreSetPtpPara=0 时, 回零运动的减加速度默认等于 acc, 起跳速度、终点速度、平滑系数默认为 0;

6.3 综合示例

下面示例代码实现了通过原点开关进行回零，在进行回零前，需要正确设置限位开关电平，并确认速度范围和安全距离，在确认安全的情况下才能执行回零动作。

```
//回零运动例程(单轴回零)
short gc_example(void)
{
    /******* 已省略打开控制器部分，统一描述在第五章1 *****/
    THomeSetting homeSetup;    // 回零参数设置(根据自身机构特点配置)
    homeSetup.mode = HM_MODE1; // 回零模式(HM_MODE1单原点回零)
    homeSetup.dir = 0;          // 搜寻零点方向(必须)，0:负向,1: 正向,其它值无意义
    homeSetup.offset = 0;       // 原点偏移
    homeSetup.scan1stVel = 5;    // 基本搜寻速度
    homeSetup.scan2ndVel = 0;    // 二次回零时使用，低速(建议小于10p/ms)，与参数reScanEn一
    起使用
    homeSetup.acc = 0.5;        // 加速度
    homeSetup.reScanEn = 0;      // 二次搜寻零点，与参数scan2ndVel一起使用
    homeSetup.homeEdge = 0;      // 原点，触发沿,下降沿
    homeSetup.lmtEdge = 0;       // 限位,触发沿(默认下降沿)
    homeSetup.zEdge = 0;         // Z相位,触发沿(默认下降沿)
    homeSetup.iniRetPos = 0;      // 起始反向运动距离(可选,不用时设为0)
    homeSetup.retSwOffset = 0;    // 反向运动时离开开关距离(可选,不用时设为0)
    homeSetup.safeLen = 0;        //安全距离,回零时最远搜寻距离(可选,不用时设为0,不限制距离)
    homeSetup.usePreSetPtpPara = 0; //当usePreSetPtpPara=0时，回零运动的减加速度默认等于
    acc,起跳速度、终点速度、平滑系数默认为0
    rtn = NMC_MtSetHomePara(axisHandle, &homeSetup); // 设置回零参数
    gc_rtn_error(rtn);
    rtn = NMC_MtHome(axisHandle); // 启动回零
    gc_rtn_error(rtn);
    /******* 查询回零过程 *****/
    short homeSts;
    while (true)
    {
        rtn = NMC_MtGetHomeSts(axisHandle, &homeSts);    //读取回零状态
        gc_rtn_error(rtn);
        if ((homeSts & BIT_AXHOME_OK) != 0) { break; /*对应轴原点信号被触发，回零完成*/ }
        if ((homeSts & BIT_AXHOME_FAIL) != 0)
            || ((homeSts & BIT_AXHOME_ERR_MV) != 0)
            || ((homeSts & BIT_AXHOME_ERR_SWT) != 0))
        {
            return 1;                // 回零过程出现错误
        }
    }
}
```

```
return rtn;  
}
```

6.4 常见问题

正常回零的情况下，出现回零位置偏差，总结以下几点作为参考(优先使用GCS工具测试回零功能)：

(1) Z相信号可能在硬限位之外，无法触发(机械结构)

使用GCS工具的菜单栏 “功能” -> “位置捕获”，检查Z相信号位置，调整机械位置；

(2) 原点/限位与Z相距离太近，速度过快，传感器反应延迟(机械结构)

原点/限位点应该与Z相有一定的距离，把二次回零的速度参数设置改小；

(3) 控制器的脉冲地线未与驱动器连接(接线)

驱动器与控制器需要共地，使用万用表检查；

(4) 干扰导致

电机动力线上加磁环尝试，整理导线位置；

(5) 联轴器锁不紧导致(机械结构)

加固机械结构；

(6) 步进电机的编码器源需要选择内部脉冲计数, 伺服电机选择外部编码器输入(参数设置)

回零功能时根据捕获功能处理的，捕获位置是根据实际位置触发的；

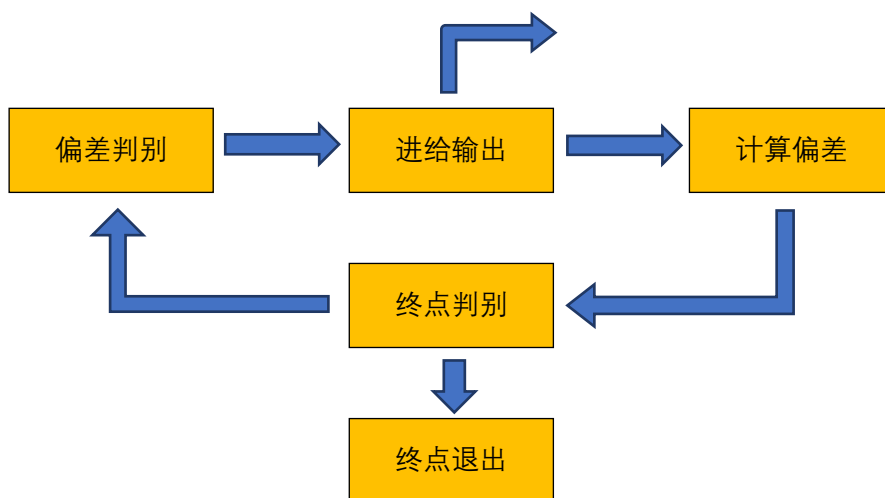
(7) 回零前，规划位置与实际位置不相等(参数设置)

确保启动回零时，规划位置与实际位置必须相等，启动前(包括上使能)进行位置清零，若每次位置清零后回零仍然出现位置不等，检查驱动器，电机编码器问题；

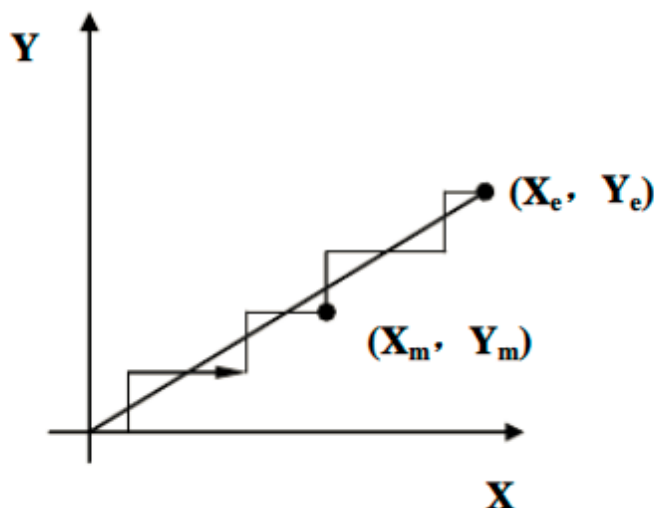
7 坐标系(插补)运动

(1)插补：即机床多轴协调运动，使刀具按照一定的轨迹进行运动的过程，常用的插补运动有平面直线插补，平面圆弧插补，空间直线插补，空间圆弧插补，螺旋线插补等，同时高川运动控制卡还提供的多轴插补运动；

逐点比较法是很典型的插补算法，它是一种最早的插补算法，该法的原理是：运动系统在控制过程中，能逐点的计算和运动轨迹和给定轨迹的偏差，并根据偏差控制进给轴向给定的轨迹靠拢，缩小偏差，使加工轨迹逼近给定轨迹。



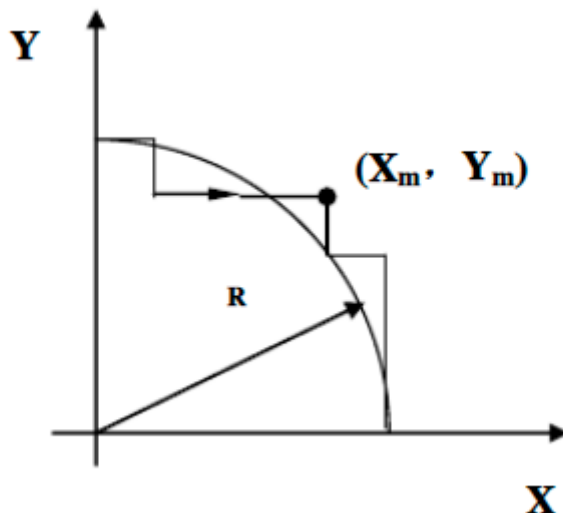
直线插补逼近示意图：



在此方式中，两点间的插补沿着直线的点群来逼近，沿此直线控制刀具的运动。所谓直线插补就是只能用于实际轮廓是直线的插补方式(如果不是直线，也可以用逼近的方式把曲线用一段线段去逼近，从而每一段线段就可以用直线插补了)。首先假设在实际轮廓起始点处沿 x 方向走一小段(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，实际轮廓就由一段段的折线拼接而成，虽然是折线，但是如果我们每一段走刀线段都非常小(在精度允许范围内)，那么此段折线和实际轮廓还是可以近似地看成相同的曲线的。

圆弧插补逼近示意图：



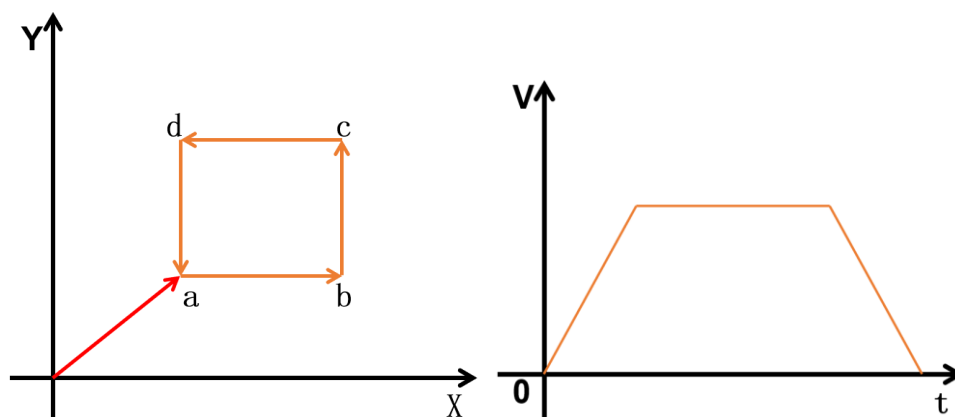
此插补方式中，根据两端点间的插补数字信息，计算出逼近实际圆弧的点群，控制刀具沿这些点运动，加工出圆弧曲线。

在机床的实际加工中，被加工工件的轮廓形状千差万别，各式各样。严格说来，为了满足几何尺寸精度的要求，刀具中心轨迹应该准确地依照工件的轮廓形状来生成。然而，对于简单的曲线，数控装置易于实现，但对于较复杂的形状，若直接生成，势必会使算法变得很复杂，计算机的工作量也相应地大大增加。因此，在实际应用中，常常采用一小段直线或圆弧去进行逼近，有些场合也可以用抛物线、椭圆、双曲线和其他高次曲线去逼近(或称为拟合)。所谓插补是指数据密化的过程。在对数控系统输入有限坐标点(例如起点、终点)的情况下，计算机根据线段的特征(直线、圆弧、椭圆等)，运用一定的算法，自动地在有限坐标点之间生成一系列的坐标数据，即所谓数据密化，从而自动地对各坐标轴进行脉冲分配，完成整个线段的轨迹运行，以满足加工精度的要求。

(2) 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；

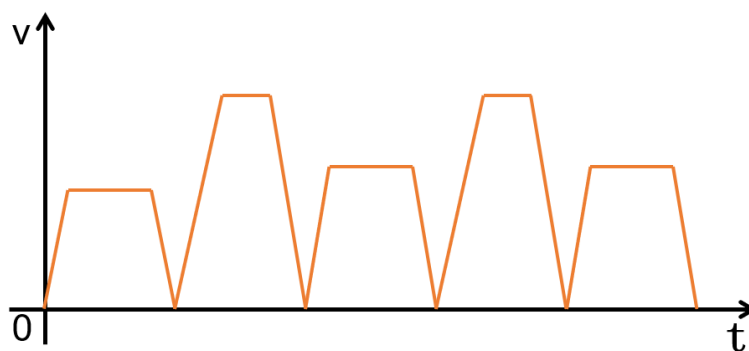
下面的例子说明前瞻预处理的机制优势。假设机床要加工一个长方形的零件，刀具所走的轨迹如图下图 7.1(a)所示，假设 a 点到 b 点距离 3000 个单位长度，有 30 段规划； b 点到 c 点距

离 2000 个单位长度，有 20 段规划，每段规划 100 个单位长度。

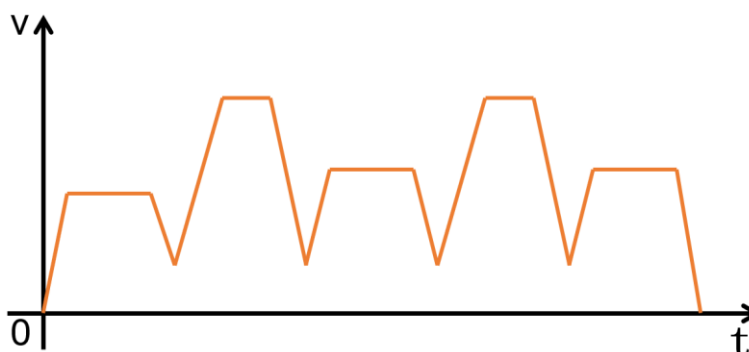


7.1(a)

7.1(b)



7.1(c)



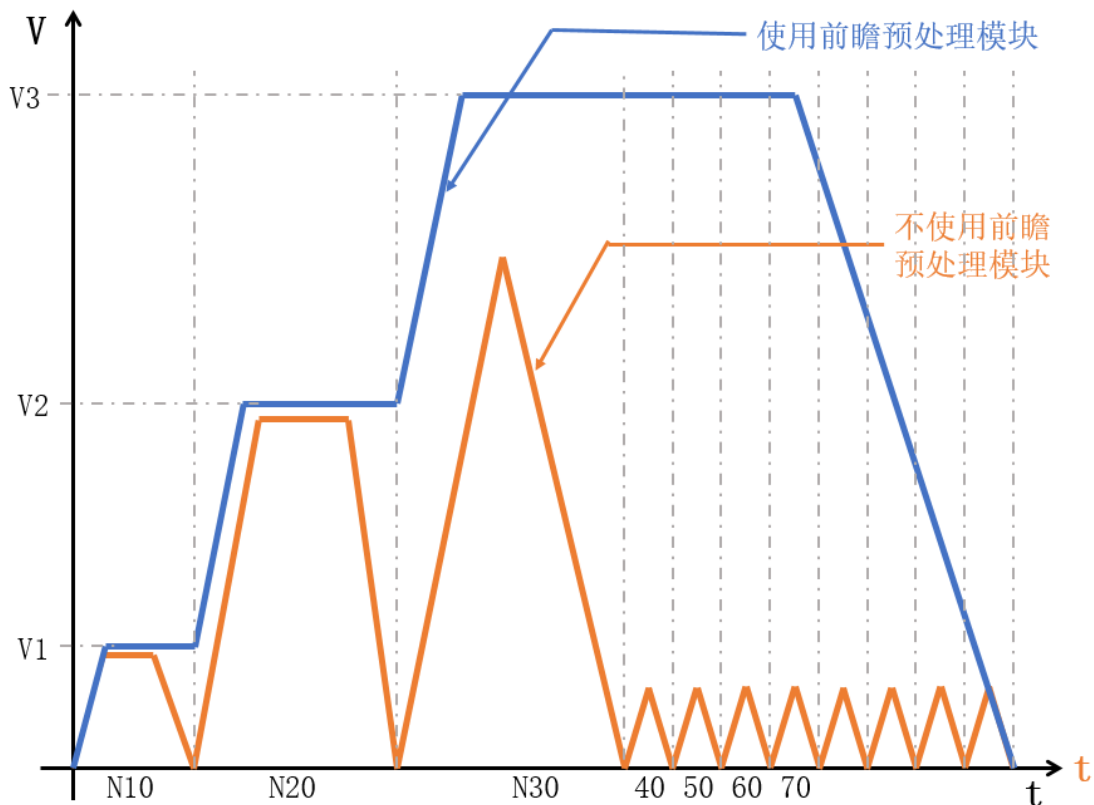
7.1(d)

图 7.1 有前瞻与没有前瞻的速度规划区别示意图 ($t \neq t$)

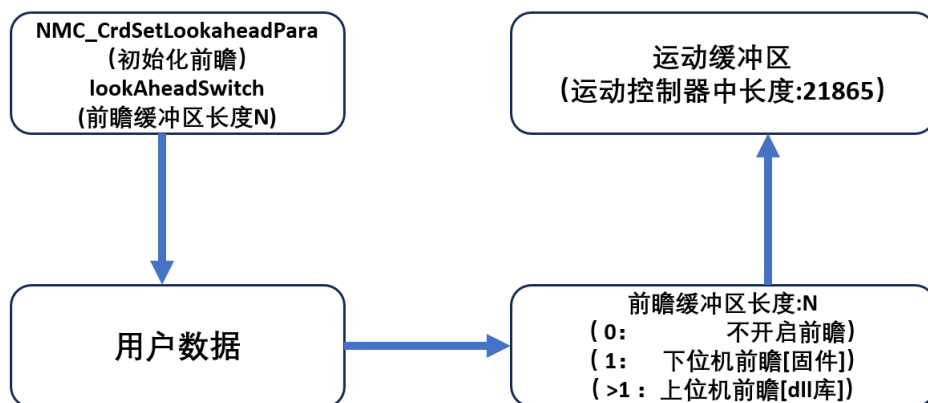
如果按照图 7.1(b)所示的速度规划，即在拐角处不减速，则加工精度一定会较低，而且可能在拐弯时对刀具和零件造成较大冲击。如果按照图 7.1(c)所示的速度规划，即在拐角处减速为 0，可以最大限度保证加工精度，但加工速度就会慢下来。如果按照图 7.1(d)所示的速度规划，在拐角处将速度减到一个合理值，既可以满足加工精度又能提高加工速度，就是一个好的速度规划。为了实现类似图 7.1(d)所示的速度规划，前瞻预处理模块不仅要知道当

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 7.1 (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 7.1.1 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，速度上有显著的提升，加工更流畅。前瞻预处理流程图如图 7.1.2 所示：



7.1.1 使用和不使用前瞻预处理模块的速度曲线对比图 ($t \neq t$)



7.1.2 前瞻预处理流程图

7.1 坐标系初始化

(1) 坐标系(插补)运动的基本使用步骤是：打开坐标系→初始化坐标系→压入插补指令→结束压入→启动插补运动→查询坐标系状态→等待运动完成；

(2) 每个轴可以配置成单轴或坐标系(插补)方式，相同轴仅存在于一种运动模式；

(3) 坐标系初始化之前需要调用 [NMC_CrdOpen](#) 获得坐标系句柄，整个运动过程只执行一次；

7.1.1 指令说明

(1) 建立插补坐标系系统(必须)

[NMC_CrdConfig\(HAND crdHandle, TCrdConfig *pConfig\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pConfig	TCrdConfig *	输入	坐标系配置，结构体定义如下： <pre>typedef struct { short axCnts; //轴数 short reserved[3]; //保留 short pAxArray[4]; //轴映射表 short port[4]; //端口映射表 } TCrdConfig;</pre>

注意：建立插补坐标系时，目前坐标系最多支持 4 个轴，轴映射表中轴号取值范围为[0, n]，端口均设为 0；

(2) 读取插补坐标系系统的配置

[NMC_CrdGetConfig\(HAND crdHandle, TCrdConfig *pConfig\);](#)

参考：[NMC_CrdConfig](#)

(3) 解除组(坐标系)，删除坐标系的轴映射和参数配置，返回到单轴模式

[NMC_CrdDelete\(HAND crdHandle \);](#)

参考：略

(4) 设置坐标系参数

[NMC_CrdSetPara\(HAND crdHandle, TCrdPara *pCrdPara\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pCrdPara	TCrdPara *	输入	坐标系参数，结构体定义如下： <pre>typedef struct { short orgFlag; // 是否自定义原点 short reserved[3]; // 保留 long offset[4]; // 自定义坐标系原点偏置(基于机械原点), 单位: 脉冲 double synAccMax; // 最大合成加速度, 单位: 脉冲/ms^2 double synVelMax; // 最大合成速度, 单位: 脉冲/ms } TCrdPara;</pre>

(5) 读取坐标系参数

[NMC_CrdGetPara\(HAND crdHandle, TCrdPara *pCrdPara\);](#)

参考: [NMC_CrdSetPara;](#)

(6) 设置坐标系高级参数

[NMC_CrdSetExtPara\(HAND crdHandle, TExtCrdPara *extCrdPara\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
extCrdPara	TExtCrdPara *	输入	坐标系高级参数，结构体定义如下： <pre>typedef struct { double startVel; // (默认0) double T; // 前瞻时间常数, 越大则速度变化越小 (默认1) double smoothDec; // 停止减速度(默认accMax)</pre>

			<pre>double abruptDec;//停止减速度(默认无穷大) short lookAheadSwitch; //0:不需要前瞻, 1:使用下位机前瞻, [2,n]: 使用上位机前瞻, 数值为前瞻段数缓存段数(默认有前瞻,为1) short eventTime;// 最小的匀速段时间, 单位 ms(默认10ms) short reserved[2]; }TExtCrdPara;</pre>
--	--	--	--

(7) 读取坐标系参数

[NMC_CrdGetExtPara\(HAND_crdHandle, TExtCrdPara * extCrdPara\);](#)

参考: [NMC_CrdGetExtPara;](#)

(8) 设置圆弧插补参数(高级指令)

[NMC_CrdSetArcSecPara\(HAND_crdHandle, TArcSecSetting *pSetting\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pSetting	TArcSecSetting *	输入	<pre>typedef struct{ double minSectionLen;// 分解的最小段长, 默认 1 脉冲, 范围[1,10000] double maxArcDiff; // 最大的圆弧有效性误差, 单位: // 圆弧插补误差配置表 // 注意: MTN 通过限制圆弧插补速度, 从而保证 插补误差。r 全部为 0, 则表示关闭这个功能 double r[MAX_ERR_TABLE_SECTION];// 半径 double err[MAX_ERR_TABLE_SECTION+1]; // 半径对应的插补误差, 半径 [0, r0], 对应 err0; 半径[r0, r1], 对应 err1; 半径[r1, +max], 对</pre>

			<pre> 应 err2 }TArcSecSetting; </pre>
--	--	--	--------------------------------------

(9) 读取圆弧插补参数(高级指令)

[NMC_CrdGetArcSecPara\(HAND crdHandle, TArcSecSetting *pSetting\);](#)

参考: [NMC_CrdSetArcSecPara](#)

(10) 设置向心加速度限制

[NMC_CrdSetLookAheadCentriAcc\(HAND crdHandle, short isUsingSetAcc, double centriAcc\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
isUsingSetAcc	short	输入	1:启用设置向心加速度 0:不启用
centriAcc	double	输入	向心加速度, 单位脉冲/ms ²

注: 圆弧受向心加速度影响, 值根据机台最大承受能力, 圆弧不限速下, 尽量设置大;

(11) 设置 4 维插补下, A 轴的最大容忍转弯速度, 若大于该速度, 则需要降速处理

[NMC_CrdSetFourthAxisTolTurnVel\(HAND crdHandle, double tolVel\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
tolVel	double	输入	最大容忍转弯速度

(12) 设置前瞻参数

[NMC_CrdSetLookaheadPara\(HAND crdHandle, TLookaheadPara *lkhPara\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
lkhPara	TLookaheadPara *	输入	<pre> typedef struct{ short lookAheadSwitch; // 前 瞻功能控制, 0:不需要前瞻, 1:使用下 位机前瞻, [2, n]: 使用上位机前瞻, 数 </pre>

			<p>值为前瞻段数，缓冲区段数</p> <p><code>short centAccEn;</code> // 是否使用指定的向心加速度，默认为 0，即不使用</p> <p><code>short crossEn;</code> // 短线段合并开关, 0:关闭合并; 1: 启用合并</p> <p><code>short eventTime;</code> // 最小的匀速段时间, 单位 ms, (默认为 10)</p> <p><code>double accMax;</code> // 机台允许的最大加速度, 默认 1000</p> <p><code>double T;</code> // 前瞻时间常数, 越大则速度变化越小(默认 1), 拐角速度与之相关;</p> <p><code>double slowAng;</code> // 减速角度, 小于设定角度, 则不减速</p> <p><code>double stopAng;</code> // 停止角度, 大于设定角度, 则减速为 0</p> <p><code>double crossProp;</code> // 短线段合并粒度, 取值范围 (0, n], 数值越大, 则合并越夸张</p> <p><code>double centAcc;</code> // 向心加速度</p> <p>// 单轴允许的速度变化量</p> <p><code>double dvMax[CRD_MAX_DIMENSION];</code></p> <p>// 单轴允许的最大加速度</p> <p><code>double daMax[CRD_MAX_DIMENSION];</code></p> <p><code>double radiusCoef;</code> // 曲率系数, 系数越大, 则同曲率曲线内的速度越大, 默认值 1.5, 为 0 表示取消曲率计算</p> <p><code>double radiusErr;</code> // 曲率误差,</p>
--	--	--	--

			默认值 200, 请根据实际误差要求配置 <pre>double radiusAng; // 曲率角</pre> 度, 默认值 7 <pre>short preHandleEn; // 轨迹预</pre> 处理功能开关 <pre>short reserved; // 保留参数</pre> <pre>float preHandleDisAngMin; //</pre> 轨迹预处理最小角度, 保留尖角 <pre>float preHandleDisAngMax; //</pre> 轨迹预处理最大角度 <pre>float preHandleTol; // 预处理</pre> 误差, 单位: 脉冲 <pre>} TLookaheadPara;</pre>
--	--	--	--

(13) 读取前瞻参数

[NMC_CrdGetLookaheadPara \(HAND crdHandle, TLookaheadPara *lkhPara, short defaultFlag\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
lkhPara	TLookaheadPara *	输入	前瞻相关的参数
defaultFlag	short	输出	1 表示读取默认值, 0 表示读取当前值

参考: [NMC_CrdSetLookaheadPara](#)

(14) 读取插补线段长度

[NMC_CrdGetBufLength \(HAND crdHandle, double *pLen\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pLen	double *	输出	线段长度

(15) 读取坐标系状态

[NMC_CrdGetStsEx\(HAND crdHandle, short *pSts, long *pUserSeg \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pSts	short *	输出	crd 状态, 按位表示
pUserSeg	long *	输入	当前的用户段号

7.1.2 代码示例

```
// 坐标系(插补)运动初始化
short gc_example(void)
{
    /***** 已省略打开控制器部分, 统一描述在第五章1 *****/
    rtn = NMC_CrdOpen(devHandle, &crdHandle); //打开坐标系
    gc_rtn_error(rtn);
    TCrdConfig crd;
    crd.axCnts = 3; // 设置3轴坐标系
    crd.pAxArray[0] = 0; // 轴端口0对应X轴
    crd.pAxArray[1] = 1; // 轴端口1对应Y轴
    crd.pAxArray[2] = 2; // 轴端口2对应Z轴
    crd.pAxArray[3] = -1; // 轴端口3对应A轴
    crd.port[0] = 0; // 端口映射表均设为0
    crd.port[1] = 0; // 端口映射表均设为0
    crd.port[2] = 0; // 端口映射表均设为0
    crd.port[3] = 0; // 端口映射表均设为0
    rtn = NMC_CrdConfig(crdHandle, &crd); //配置坐标系
    gc_rtn_error(rtn);
    TCrdPara crdPara;
    crdPara.orgFlag = 1; // 默认原点(0, 0, 0, 0)
    crdPara.offset[0] = 0; // 坐标原点配置设为0
    crdPara.offset[1] = 0; // 坐标原点配置设为0
    crdPara.offset[2] = 0; // 坐标原点配置设为0
    crdPara.offset[3] = 0; // 坐标原点配置设为0
    crdPara.synAccMax = 50; //最大合成加速度p/ms^2, 50已经比较大
    crdPara.synVelMax = 800; //最大合成速度, 允许范围内, 两个参数尽可能大
    rtn = NMC_CrdSetPara(crdHandle, &crdPara); // 设置运动参数
    gc_rtn_error(rtn);
    /***** 以下配置可选 *****/
    TCrdSafePara crdSafePara;
    crdSafePara.estpDec = 1000; //急停加速度
    crdSafePara.maxAcc = 50; //设置坐标系最大加速度50p/ms^2
}
```

```

crdSafePara.maxVel = 800; //设置坐标系最大速度800p/ms
rtn = NMC_CrdSetSafePara(crdHandle, &crdSafePara); //设置安全参数
gc_rtn_error(rtn);
/***** 以上配置可选 *****/
rtn = NMC_CrdClrError(crdHandle); //清除坐标系错误状态
gc_rtn_error(rtn);
rtn = NMC_CrdBufClr(crdHandle); //压入指令之前需要清空缓存区
gc_rtn_error(rtn);
/*****
* 以上为坐标系初始化完成,坐标系(插补)运动基本流程:
* 打开坐标系->初始化坐标系->压入插补指令->结束压入->启动插补运动->查询坐标系状态->等
* 待运动完成
*****/
return rtn;
}

```

7.2 直线插补

7.2.1 指令说明

(1) 直线插补(带前瞻开关)

NMC_CrdLineXYZEx(HAND crdHandle, long segNo, short crdAxMask, long
*pTgPosArray, double endVel, double vel, double synAcc, short lookaheadDis);

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
crdAxMask	short	输入	参与的轴,按位表示
pTgPosArray	long *	输入	目标位置数组,最大长度为 3
endVel	double	输入	终点速度,单位 pulse/ms
vel	double	输入	运行速度,单位 pulse/ms
synAcc	double	输入	合成加速度,单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻,0: 使用前瞻,则控制器自动计算终点速度,1: 禁用前瞻,使用设定的终点速度(endVel)

(2) 四轴直线插补

[NMC_CrdLineXYZA\(HAND crdHandle, long segNo, short crdAxMask, long *pTgPosArray, double endVel, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
crdAxMask	short	输入	参与的轴, 按位表示
pTgPosArray	long *	输入	目标位置数组, 最大长度为 4
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

7.2.2 代码示例

```
//直线插补, 本例默认使用前瞻, lookaheadDis = 0;
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /****** 已省略坐标系初始化部分, 见上节7.1.2 *****/
    long tgPos[3];
    tgPos[0] = 10000;
    tgPos[1] = 10000;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 0, 7, tgPos, 0, 10, 0.5, 0);    // 压入直线
    gc_rtn_error(rtn);
    // 压入直线
    tgPos[0] = 0;    tgPos[1] = 0;    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 0, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    /****** 省略启动插补部分 *****/
    return rtn;
}
```


7.3 平面圆弧插补

7.3.1 指令说明

(1)XY 平面圆弧插补：终点位置、圆心、方向

NMC CrdArcCenterEx(HAND crdHandle, long segNo, long *pTgPosArray, long *pCenterPosArray, short circleDir, double endVel, double vel, double synAcc, short lookaheadDis);

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表示 XY 轴的目标位置)
pCenterPosArray	long *	输入	圆心坐标(二维数组, 分别表示 XY 轴相对于起点的圆心位置), 注意 : 圆心坐标为相对于起点的相对位置
circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

(2)YZ 平面圆弧插补：终点位置、圆心、方向

NMC CrdArcCenterYZEx(HAND crdHandle, long segNo, long *pTgPos, long *pCenterPos, short circleDir, double velEnd, double vel, double synAcc, short lookaheadDis);

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表示 YZ 轴的目标位置)
pCenterPosArray	long *	输入	圆心坐标(二维数组, 分别表示 YZ 轴相对于起点的圆心位置), 注意: 圆心坐标为相对于起点的相对位置
circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

(3) ZX 平面圆弧插补: 终点位置、圆心、方向

NMC CrdArcCenterZXEx(HAND crdHandle, long segNo, long *pTgPos, long *pCenterPos, short circleDir, double velEnd, double vel, double synAcc, short lookaheadDis);

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表 ZX 轴的目标位置)
pCenterPosArray	long *	输入	圆心坐标(二维数组, 分别表示 ZX 轴相对于起点的圆心位置), 注意: 圆心坐标为相对于起点的相对位置

circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算 终点速度, 1: 禁用前瞻, 使用设定的终点速度 (endVel)

(4) XY 平面圆弧插补: 终点位置、半径、方向

[NMC CrdArcRadiusEx\(HAND crdHandle, long segNo, long *pTgPosArray, double radius, short circleDir, double endVel, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表示 XY 轴的目标位置)
radius	double	输入	圆弧半径, 大于 0 表示劣弧, 小于 0 表示优弧(起点位置和终点位置都在圆上)
circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算 终点速度, 1: 禁用前瞻, 使用设定的 终点速度(endVel)

(5) YZ 平面圆弧插补：终点位置、半径、方向

[NMC CrdArcRadiusYZEx\(HAND crdHandle, long segNo, long *pTgPos, double radius, short circleDir, double velEnd, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表示 YZ 轴的目标位置)
radius	double	输入	圆弧半径, 大于 0 表示劣弧, 小于 0 表示优弧(起点位置和终点位置都在圆上)
circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

(6) ZX 平面圆弧插补：终点位置、半径、方向

[NMC CrdArcRadiusZXEx\(HAND crdHandle, long segNo, long *pTgPos, double radius, short circleDir, double velEnd, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(二维数组, 分别表示 ZX 轴的目标位置)
radius	double	输入	圆弧半径, 大于 0 表示劣弧, 小于 0 表示优弧(起点位置和终点位置都在圆上)

circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向 圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度 (endVel)

(7) XY 平面圆弧插补: 起点(当前点)、中点、终点

[NMC CrdArcPPPEx\(HAND crdHandle, long segNo, long *pMidPosArray, long *pTgPosArray, double endVel, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
pMidPosArray	long *	输入	中间位置点坐标(二维数组, 分别表示中间点的 XY 轴的位置)
pTgPosArray	long *	输入	终点位置坐标(二维数组, 分别表示终点的 XY 轴的位置)
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度 (endVel)

(8) 椭圆插补, 默认不参与速度前瞻, 起始和终止速度为 0(注意! 椭圆为整圆)

[NMC CrdEllipse\(HAND crdHandle, long segNo, double abRatio, long *pCenterPos, short ellipseDir, double vel, double synAcc\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
abRatio	double	输入	表示椭圆 AB 轴的长度比例值，值范围： [0.05, 1]
pCenterPos	long *	输入	椭圆圆心点位置(二维数组, 分别表示中点的 XY 轴的位置)。注意：起点位置到圆心位置为长轴（椭圆圆心是相对位置的）
vel	double	输入	运行速度，单位 pulse/ms
synAcc	double	输入	合成加速度，单位 pulse/ms ²

7.3.2 代码示例

```
//直线插补，平面圆弧插补，本例默认使用前瞻，lookaheadDis = 0;
short gc_example(void)
{
    /****** 已省略打开控制器部分，统一描述在第五章1 *****/
    /****** 已省略坐标系初始化部分，见上节7.1.2 *****/
    // 压入直线
    long tgPos[3];
    tgPos[0] = 10000;
    tgPos[1] = 10000;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 0, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    // 压入直线
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 0, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    //压入XY平面圆弧插补
    long tgpos[2];
    tgpos[0] = 0;
    tgpos[1] = 0;
    long cenpos[2];
    cenpos[0] = 5000;
    cenpos[1] = 5000;
```

```
//终点圆心方向, 起点为(0, 0), 圆心(5000, 5000)的整圆, 顺时针  
rtn = NMC_CrdArcCenterEx(crdHandle, 1, tgpos, cenpos, 0, 0, 10, 1, 0);  
//终点半径方向, 起点为(0, 0), 终点(3000, 4000)的半圆, 顺时针  
tgpos[0] = 3000;  
tgpos[1] = 4000;  
rtn = NMC_CrdArcRadiusEx(crdHandle, 2, tgpos, 2500, 1, 0, 10, 1, 0);  
//起点、中点、终点, 起点(10000, 0), 中点(5000, -5000), 终点(0, 0)  
tgpos[0] = 0;  
tgpos[1] = 0;  
cenpos[0] = 5000;  
cenpos[1] = -5000;  
rtn = NMC_CrdArcPPPEX(crdHandle, 3, cenpos, tgpos, 0, 10, 1, 0);  
/***** 省略启动插补部分 *****/  
return rtn;  
}
```

7.4 螺旋线插补

7.4.1 功能介绍

通过 X 轴, Y 轴, Z 轴各自的进给量驱动 3 个轴的电机同时运动, 合成运动轨迹即是螺旋线轨迹, 通过插补周期和插补算法的调用, 最终达到螺旋线的终点。(当水平圆弧运动和垂直直线运动叠加的并且并列执行, 所走的轨迹为一螺旋线, 既水平做圆弧运动在工作平面确定的轴(X 轴和 Y 轴)进行, 直线运动的轴(Z 轴)在垂直方向进行, 通过这种方式合成加工出理想的螺旋体结构称为螺旋线插补)如图 7.4.1 螺旋线插补示意图。

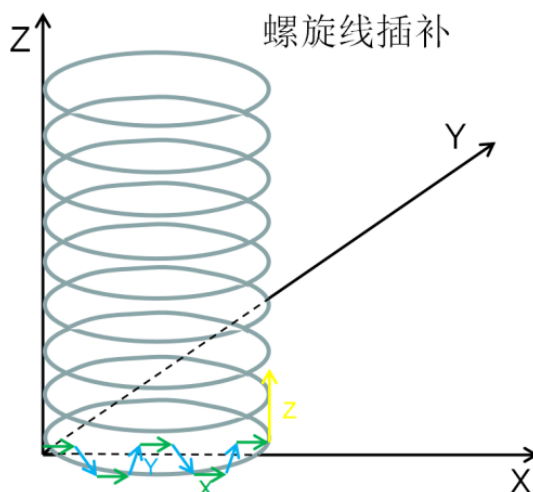


图 7.4.1 螺旋线插补示意图

7.4.2 指令说明

(1) 螺旋线插补

`NMC_CrdHelixCenterEx(HAND crdHandle, long segNo, long *pTgPosArray, long *pCenterPosArray, short circleDir, double rounds, double endVel, double vel, double synAcc, short lookaheadDis);`

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
pTgPosArray	long *	输入	目标位置(三维数组, 分别表示终点的XYZ轴的位置)
pCenterPosArray	long *	输入	圆心位置(二维数组, 分别表示XY轴相对于起点的圆心位置), 注意 : 圆心坐标为相对于起点的相对位置
circleDir	short	输入	圆弧方向, 0 表示顺时针方向, 1 表示逆时针方向
rounds	double	输入	Z方向圈数
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

7.4.3 代码示例

```
//螺旋线插补
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /****** 已省略坐标系初始化部分, 见上节7.1.2 *****/
    long tgPos[3];
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 0;
```



```

//压入直线运动到(0, 0, 0)
rtn = NMC_CrdLineXYZEx(crdHandle, 15, 7, tgPos, 0, 10, 0.5, 0);
gc_rtn_error(rtn);
long tgGenpos[2];
tgGenpos[0] = 5000;
tgGenpos[1] = 0;
tgPos[0] = 0;
tgPos[1] = 0;
tgPos[2] = 20000;
//圆心(5000, 0)高度20000, 总共旋转了5圈
rtn = NMC_CrdHelixCenterEx(crdHandle, 16, tgPos, tgGenpos, 0, 5, 0, 10, 1, 0);
gc_rtn_error(rtn);
/***** 省略启动插补部分 *****/
return rtn;
}

```

7.5 空间圆弧插补

7.5.1 指令说明

(1) 3D 圆弧插补

[NMC_CrdArc3DEx\(HAND crdHandle, long segNo, long *pMidPos, long *pTgPos, double velEnd, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
pMidPos	long *	输入	中点位置(三维数组, 分别表示中点的XYZ轴的位置)
pTgPos	long *	输入	终点位置(三维数组, 分别表示终点的XYZ轴的位置)
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定

			的终点速度(endVel)
--	--	--	---------------

(2) 3D 圆弧插补(整圆)

[NMC CrdCircle3DEx\(HAND crdHandle, long segNo, long *pMidPos, long *pTgPos, double velEnd, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
pMidPos	long *	输入	中点位置(三维数组, 分别表示中点的 XYZ 轴的位置)
pTgPos	long *	输入	终点位置(三维数组, 分别表示终点的 XYZ 轴的位置)
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

(3) XY平面整圆插补

[NMC CrdCirclePPP\(HAND crdHandle, long segNo, short planeIdx, long *pStPos, long *pMidPosArray, long *pTgPosArray, double endVel, double vel, double synAcc, short lookaheadDis \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
planeIdx	short	输入	平面类型, 0:XY 1:YZ, 2:ZX, 目前只支持 XY 平面
pStPos	long *	输入	起点位置点坐标(二维数组, 分别表示起点的 XY 轴的位置)

pMidPosArray	long *	输入	中间位置点坐标（二维数组, 分别表示中间点的 XY 轴的位置）
pTgPosArray	long *	输入	终点位置坐标（二维数组, 分别表示终点的 XY 轴的位置）
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度 (endVel)

7.5.2 代码示例

```
//空间圆弧插补
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /****** 已省略坐标系初始化部分, 见上节7.1.2 *****/
    //压入直线运动到(0, 0, 0)
    long tgPos[3];
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 17, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 10000;
    long midpos[3];
    midpos[0] = 5000;
    midpos[1] = 5000;
    midpos[2] = 5000;
    //压入一个过(5000, 5000, 5000), 终点(0, 0, 10000)的3D圆弧
    rtn = NMC_CrdArc3D(crdHandle, 18, midpos, tgPos, 0, 10, 1);
    gc_rtn_error(rtn);
    /****** 省略启动插补部分 *****/
    return rtn;
}
```

7.6 多维插补

7.6.1 指令说明

(1) 多轴直线插补(最多支持 8 轴)

[NMC CrdLineXYZD8\(HAND crdHandle, long segNo, long crdAxmask, long extAxMask, long *pTgPosArray, double endVel, double vel, double synAcc, short lookaheadDis\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号 (0~n 用户自定义)
crdAxmask	long	输入	坐标系参与的轴, 按位表示 (指代坐标系内的 XYZA, 与轴号无关)
extAxMask	long	输入	其他参与轴, 按位表示, 不能包括坐标系中 (与轴号有关)
pTgPosArray	long *	输入	目标位置数组, 长度为所有参与运动的轴的数量, 索引小的是坐标系内的轴的坐标 (按 crdAxMask 位排列)。索引大的是其它联动轴的坐标 (按 extAxMask 位排列)
endVel	double	输入	终点速度, 单位 pulse/ms
vel	double	输入	运行速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms ²
lookaheadDis	short	输入	是否使用前瞻, 0: 使用前瞻, 则控制器自动计算终点速度, 1: 禁用前瞻, 使用设定的终点速度(endVel)

(2) 打包的多轴直线插补

[NMC CrdLineXYZD8Pack\(HAND crdHandle, short count, TCrdLineXYZD8Unit *pCmdArray\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

count	short	输入	打包指令数, 取值范围[1, 18]
pCmdArray	TCrdLineXYZD8Unit *	输入	指令列表 // 8 轴插补打包单元结构体 <pre>typedef struct{ long segNo; // 用户自定义段号 short crdAxMask; short extAxMask; long tgPos[8]; // 目标位置 float vel; // 运行速度 float endVel; // 终点速度 float acc; // 插补加速度 short lookaheadDis; // 是否使用前瞻 short reserved; // 保留 }TCrdLineXYZD8Unit;</pre>

7.6.2 代码示例

```
//多维插补 X,Y,Z,A,B,C运动
short gc_example(void)
{
    /***** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /***** 已省略坐标系初始化部分, 见上节7.1.2 *****/
    long tgPos[6];
    tgPos[0] = 10000;
    tgPos[1] = 10000;
    tgPos[2] = 10000;
    tgPos[3] = 10000;
    tgPos[4] = 10000;
    tgPos[5] = 10000;
    //X, Y, Z, A, B, C轴一起跑到(10000, 10000, 10000, 10000, 10000, 10000)
    //参与轴为 X Y Z 轴, 其他参与轴为 A B C(4 5 6 轴)
    rtn = NMC_CrdLineXYZD8(crdHandle, 4, 0x7, 0x38, tgPos, 0, 10, 1, 0);
    gc_rtn_error(rtn);
    //X, Y, Z, A, B, C轴一起跑到(0, 10000, 0, 0, 10000, 0)
    tgPos[0] = 0;
    tgPos[1] = 10000;
    tgPos[2] = 0;
```

```

    tgPos[3] = 0;
    tgPos[4] = 10000;
    tgPos[5] = 0;
    rtn = NMC_CrdLineXYZD8(crdHandle, 5, 0x7, 0x38, tgPos, 0, 10, 1, 0);
    gc_rtn_error(rtn);
    TCrdLineXYZD8Unit pack[10];           //下面介绍打包指令,一次性压入10条指令
    for (int i = 0; i < 10; i++)
    {
        pack[i].acc = 1;
        pack[i].crdAxMask = 0x7;
        pack[i].endVel = 0;
        pack[i].extAxMask = 0x38;
        pack[i].lookaheadDis = 0;
        pack[i].reserved = 0;
        pack[i].segNo = 5 + i;           //前面的例程已经压到了,该处给任意值都不影响实际运动
        pack[i].tgPos[0] = 1000 * i;
        pack[i].tgPos[1] = -1000 * i;
        pack[i].tgPos[2] = 1000 * i;
        pack[i].tgPos[3] = -1000 * i;
        pack[i].tgPos[4] = 1000 * i;
        pack[i].tgPos[5] = -1000 * i;
        pack[i].tgPos[6] = 0;
        pack[i].tgPos[7] = 0;
    }
    rtn = NMC_CrdLineXYZD8Pack(crdHandle, 10, pack);
    gc_rtn_error(rtn); //可以预测到轴最终跑到(9000, -9000, 9000, -9000, 9000, -9000)
    /***** 省略启动插补部分 *****/
    return rtn;
}

```

7.7 缓存区的操作(IO, 延时等)

7.7.1 指令说明

(1)缓冲区 D0

[NMC_CrdBufDo\(HAND crdHandle, long segNo, short doType, long ch, long value\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
doType	short	输入	//D0 类型, 见宏定义 // 缓冲区输出 D0 组定义

			<pre> #define CRD_BUFF_DO_MOTOR_ENABLE 1 //电机使能 #define CRD_BUFF_DO_MOTOR_CLEAR 2 //电机报警清除 #define CRD_BUFF_DO_GPD01 3 // 通用输出 1 #define CRD_BUFF_DO_GPD02 4 // 通用输出 2 #define CRD_BUFF_DO_EXTD01 5 // 扩展模块 1 #define CRD_BUFF_DO_EXTD02 6 // 扩展模块 2 #define CRD_BUFF_DO_EXTD03 7 // 扩展模块 3 #define CRD_BUFF_DO_EXTD04 8 // 扩展模块 4 #define CRD_BUFF_DO_EXTD05 9 // 扩展模块 5 #define CRD_BUFF_DO_EXTD06 10 // 扩展模块 6 </pre>
ch	long	输入	位序号, 取值范围[0, 31]
value	long	输入	输出值, 取值范围[0, 1]

(2) 缓冲区 DO

[NMC_CrdBufDoEx\(HAND crdHandle, long segNo, short group, long doMask, long doValue\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
group	short	输入	<pre> //DO 类型, 见宏定义 // 缓冲区输出 DO 组定义 #define CRD_BUFF_DO_MOTOR_ENABLE 1 //电机使能 #define CRD_BUFF_DO_MOTOR_CLEAR 2 // 电机报警清除 #define CRD_BUFF_DO_GPD01 3 // 通用输出 1 #define CRD_BUFF_DO_GPD02 4 // 通用输出 2 #define CRD_BUFF_DO_EXTD01 5 // 扩展模块 1 #define CRD_BUFF_DO_EXTD02 6 // 扩展模块 2 #define CRD_BUFF_DO_EXTD03 7 // 扩展模块 3 </pre>

			<pre>#define CRD_BUFF_DO_EXTD04 8 // 扩展模块 4 #define CRD_BUFF_DO_EXTD05 9 // 扩展模块 5 #define CRD_BUFF_DO_EXTD06 10 // 扩展模块 6</pre>
doMask	long	输入	位掩码
doValue	long	输入	输出值

(3) 缓冲区输出(模拟量和 PWM)

[NMC CrdBufOut\(HAND crdHandle, long segNo, short group, short ch, long value\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
group	short	输入	组别, 见宏定义 <pre>// 通用输出宏类型 #define BUF_OUT_GROUP_DA 0 // 模拟量输出 #define BUF_OUT_GROUP_PWM 1 // PWM 输出</pre>
ch	long	输入	位掩码
value	long	输入	通道号, 取值范围[0, n]

(4) 缓冲区 DI 等待

[NMC CrdBufWaitDI\(HAND crdHandle, long segNo, short index, short diValue, long waitLastTime\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
index	short	输入	通道号, 取值范围[0, 127], 前 64 通道代表通用 DI, 后 64 通道代表扩展 IO
diValue	short	输入	等待值

waitLastTime	long	输入	超时, 单位: 毫秒
--------------	------	----	------------

(5) 缓存区 DI 急停

[NMC_CrdBufSetEstopDI \(HAND crdHandle, long segNo, short axis, short gpiIndex, short sense\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axis	short	输入	轴号, 取值范围[0, n)
gpiIndex	short	输入	通用输入序号[0, n]
sense	short	输入	触发电平[0, 1]

(6) 缓存区延时

[NMC_CrdBufDelay \(HAND crdHandle, long segNo, short scale, long count\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
scale	short	输入	延时单位, 0 表示单位为毫秒, 1 表示单位为秒
count	long	输入	延时时长

(7) 缓冲区单轴移动(带参数)

[NMC_CrdBufAxMoveEx \(HAND crdHandle, long segNo, short axisMask, long *pTgPos, double vel, double acc, short blockEn, short synEn\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axisMask	short	输入	参与的轴
pTgPos	long *	输入	目标位置

vel	double	输入	速度 pulse/ms
acc	double	输入	加速度 pulse/ms ²
blockEn	short	输入	是否为阻塞模式, 0, 不阻塞, 1, 阻塞
synEn	short	输入	是否为同步模式, 如果 synEn 和 blockEn 同时为 1, 则同步优先

注意: 阻塞模式为 [NMC_CrdBufAxMoveEx](#) 指令运行完毕才运行下一条插补指令; 同步模式为: 是否和下一条插补指令同起同停, 同步模式下一条插补指令必需有位移;

(8) 缓冲区单轴移动(相对位移移动)

[NMC_CrdBufAxMoveRel\(HAND crdHandle, long segNo, short axisMask, long *pRelPos, short blockEn, short synEn\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axisMask	short	输入	参与的轴
pRelPos	long *	输入	相对移动位置
vel	double	输入	速度 pulse/ms
acc	double	输入	加速度 pulse/ms ²
blockEn	short	输入	是否为阻塞模式, 0, 不阻塞, 1, 阻塞
synEn	short	输入	是否为同步模式, 如果 synEn 和 blockEn 同时为 1, 则同步优先

(9) 缓冲区单轴移动参数设置

[NMC_CrdBufSetPtpMovePara\(HAND crdHandle, long segNo, short axisNo, double vel, double acc, short soomthCoef\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axisNo	short	输入	轴号, [0, n]

vel	double	输入	速度 pulse/ms
acc	double	输入	加速度 pulse/ms ²
soomthCoef	short	输入	平滑系数

(10) 设置跟随运动前的运动补偿量

[NMC_CrdBufBeforeAxSyncMove\(HAND crdHandle, long segNo, short axis, long relDistance, double vel\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axisNo	short	输入	轴号, [0, n]
relDistance	long	输入	相对补偿位移量
vel	double	输入	补偿速度 pulse/ms

(11) 缓冲区等待电机运动到位

[NMC_CrdBufWaitEncInPosition\(HAND crdHandle, long segNo, long axisMask, long overTime\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)
axisMask	short	输入	需要等待到位的轴掩码(按位对应轴号, 不能超出控制器的最大轴数)
overTime	long	输入	等待到位超时的时间, 单位: ms

(12) 缓冲区等待特定位置, 满足条件才执行下一条指令

[NMC_CrdBufWaitPos\(HAND crdHandle, long segNo, short axisNo, short condition, long pos, short posSrc, long overTime\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	short	输入	段号(用户自定义)

axisNo	short	输入	需要等待到位的轴号
condition	short	输入	到位条件, 0: 小于等于设定位置 1: 大于等于设定位置
pos	long	输入	设定位置
posSrc	short	输入	等待规划还是编码器, 0: 编码器 1: 内部规划
overTime	long	输入	等待到位超时的时间, 单位: ms

(13) 缓冲区高级急停 IO 启动关闭

[NMC_CrdBufEstopDIExOnOff\(HAND crdHandle, long segNo, short onOff, short group\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
onOff	short	输入	1-启用高级急停 IO, 0-关闭高级急停 IO
group	short	输入	通道号, 取值[0, ESTOP_DI_EX_CH_NUM-1]

(14) 设置 BufIO 输出参数

[NMC_AdvBufIoSetParam\(HAND devHandle, TAdvBufIoParam *pPrm, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPrm	TAdvBufIoParam *	输入	<pre>typedef struct{ short outType; // 输出类型, 0: 通用输出; 1: Gate 信号; 其他保留 short outGroup; // 输出组, 取值范围[0, n] short outIndex; // 输出序号, 取值范围[0, n] short outSns; // 信号, 有效电平, 0: 低电平有效, 1: 高电平</pre>

			<pre> short pulseMode;// 0: 电平输出, 1: 脉冲输出 short reserved1; //保留 long pulseOnTime;//有效电平时间(脉冲输出方式下有效), 单位: 微秒 long pulseOffTime; //无效电平时间(脉冲输出方式下有效), 单位: 微秒 // 保留 unsigned char reserved[32]; }TAdvBufIoParam; </pre>
ch	short	输入	输出组, 取值范围: [0, MAX_ADV_BUFIO_GROUP) MAX_ADV_BUFIO_GROUP = 2

(15) 读取 BufIo 输出参数

[NMC AdvBufIoGetParam\(HAND devHandle, TAdvBufIoParam *pPrm, short ch\);](#)

参考: [NMC AdvBufIoSetParam](#)

(16) 缓冲区设置高级 BufIo 输出有效(运动一段距离后输出特定状态)

[NMC CrdAdvBufIoOnAfterLen\(HAND crdHandle, long segNo, long outLength, long value, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
outLength	long	输入	距离, 单位: 脉冲
value	long	输入	脉冲模式下输出脉冲个数, 电平方式下无意义
ch	short	输入	输出组, 取值范围: [0, MAX_ADV_BUFIO_GROUP)

			MAX_ADV_BUFIO_GROUP = 2
--	--	--	-------------------------

(17) 缓冲区设置高级 BufIo 关闭输出 (缓冲区全部运动结束前提前一段距离输出特定状态)

[NMC_CrdAdvBufIoOffBeforeLen\(HAND crdHandle, long segNo, long outLength, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
outLength	long	输入	距离, 单位: 脉冲
ch	short	输入	输出组, 取值范围: [0, MAX_ADV_BUFIO_GROUP) MAX_ADV_BUFIO_GROUP = 2

(18) 立即设置高级 BufIo 输出

[NMC_AdvBufIoOut\(HAND devHandle, long value, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
value	long	输入	脉冲模式下输出脉冲个数, 电平方式下 0:关闭, 1:打开。
ch	short	输入	输出组, 取值范围: [0, MAX_ADV_BUFIO_GROUP) MAX_ADV_BUFIO_GROUP = 2

(19) 读取 BufIo 的输出数量

[NMC_AdvBufIoGetPulseCnt\(HAND devHandle, long *pOutCnt, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pOutCnt	long *	输出	输出脉冲个数
ch	short	输出	输出组, 取值范围: [0, MAX_ADV_BUFIO_GROUP)

			MAX_ADV_BUFIO_GROUP = 2
--	--	--	-------------------------

(20) D0 脉冲输出

[NMC DoBitPulseEnable \(HAND devHandle, short doType, short doIndex, long highLevelTime, long lowLevelTime, long outCount, short initialLevel, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
doType	short	输入	D0 类型, 0: 通用输出; 其他保留
doIndex	short	输入	D0 序号, 取值范围[0, n]
highLevelTime	long	输入	高电平宽度, 单位:us
lowLevelTime	long	输入	低电平宽度, 单位:us
outCount	long	输入	输出脉冲个数
initialLevel	short	输入	0: 先输出低电平, 1: 先输出高电平
ch	short	输入	输出脉冲通道号, 取值范围: [0, MAX_NUM_DOBIT_PULSE-1]

(21) 关闭 D0 脉冲输出

[NMC DoBitPulseDisable \(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	关闭脉冲通道号, 取值范围: [0, MAX_NUM_DOBIT_PULSE-1]

(22) D0 脉冲输出 (缓冲区)

[NMC CrdBufDoBitPulseEnable \(HAND crdHandle, long segNo, short doType, short doIndex, long highLevelTime, long lowLevelTime, long outCount, short initialLevel, short ch\);](#)

参考: [NMC DoBitPulseEnable](#)

(23) 关闭 D0 脉冲输出 (缓冲区)

[NMC_CrdBufDoBitPulseDisable\(HAND crdHandle, long segNo, short ch\);](#)

参考: [NMC_DoBitPulseEnable](#)

(24)缓冲区停止坐标系运动

[NMC_CrdBufStopMtn\(HAND crdHandle, long segNo, short crdIdx\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
crdIdx	short	输入	停止坐标系序号, -1 表示停止自身

7.7.2 代码示例

```
//缓冲区直线插补
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /****** 已省略坐标系初始化部分, 见上节7.1.2 *****/
    //压入直线运动到(0, 0, 0)
    long tgPos[3];
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 19, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    //运行到(10000, 10000, 0)后输出一个D0
    tgPos[0] = 10000;
    tgPos[1] = 10000;
    tgPos[2] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 20, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    rtn = NMC_CrdBufDo(crdHandle, 21, 3, 0, 0);           // 缓冲区控制D0输出
    gc_rtn_error(rtn);
    tgPos[0] = 0;
    tgPos[1] = 0;
    tgPos[2] = 0;
    //直线运动到(0, 0, 0)后延时ms后关闭刚才的输出
    rtn = NMC_CrdLineXYZEx(crdHandle, 21, 7, tgPos, 0, 10, 0.5, 0);
    gc_rtn_error(rtn);
    rtn = NMC_CrdBufDelay(crdHandle, 22, 0, 1000);       // 缓冲区延时
}
```



```
gc_rtn_error(rtn);
rtn = NMC_CrdBufDo(crdHandle, 23, 3, 0, 1);
gc_rtn_error(rtn);
//直线运动到(10000, 10000, 0)
tgPos[0] = 10000;
tgPos[1] = 10000;
tgPos[2] = 0;
rtn = NMC_CrdLineXYZEx(crdHandle, 24, 7, tgPos, 0, 10, 0.5, 0);
gc_rtn_error(rtn);
//直线运动到(0, 0, 0), 同时让轴同步运动到的位置10000
tgPos[0] = 0;
tgPos[1] = 0;
tgPos[2] = 0;
long posex[1];
posex[0] = 10000;
rtn = NMC_CrdBufAxMoveEx(crdHandle, 25, 0x8, posex, 10, 1, 0, 1);
gc_rtn_error(rtn);
rtn = NMC_CrdLineXYZEx(crdHandle, 26, 7, tgPos, 0, 10, 0.5, 0);
gc_rtn_error(rtn);
/***** 省略启动插补部分 *****/
return rtn;
}
```

7.8 坐标系的打包插补

当插入的指令非常多时，如果一条一条的插入，占用时间会增加，会拖慢效率，我们提供了便于一次性插入多条指令的方法，可以很好的提高效率，常用于大量小线段的插补。

7.8.1 指令说明

(1) 打包插补数据

[NMC_CrdBufDataPack\(HAND crdHandle, unsigned char *pBufData, short dataLen\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pBufData	unsigned char *	输入	插补数据结构存储数组 // 缓冲区数据打包 #define BUF_LINE 0//缓冲区直线插补 #define BUF_DO 1 //缓冲区 DO 输出 #define BUF_OUT 2 //缓冲区 OUT 输出

			<pre> #define BUF_DELAY 3//缓冲区延时 #define BUF_AXMOVE 4 //缓冲区单轴运动 #define BUF_DOEX 5 //缓冲区 D0 输出 (根据掩码输出) #define BUF_ARC_R 6 //平面圆弧插 补: 终点位置、半径、方向 #define BUF_ARC_C 7 //平面圆弧插 补: 终点位置、圆心、方向 #define BUF_LASER_SETPOWER 8 // 激光: 设置能量 #define BUF_LASER_ONOFF 9 // 激光: 缓冲区开关光 #define BUF_LASER_SETFOLLOW 10 // 激光: 设置跟随 #define BUF_LASER_SETPARAM 11 // 激光: 设置参数 #define BUF_LINEXYZA 12 // 缓冲区四轴直线插补 #define BUF_SHIO_GATEPULSE 13 // 缓冲区输出 Gate 脉冲 #define BUF_DOBITPULSE 14 // DoBitPulse 功能 #define BUF_XYZD8 15 // LineXYZD8, 数据结构体为: TCrdLineXYZD8Unit #define BUF_SHIOMINFRQ 16 // 设置或清除 SHIO 最小频率 #define BUF_SHIOSETPARAM 17 // 设置 SHIO 参数 </pre>
--	--	--	---

			<pre>#define BUF_SHIOGATEONOFF 18 // 设置 Gate 开关 #define BUF_WAITENCINPOS 19 // 等待电机到位 #define BUF_WAITDI 20 // 等待 DI</pre>
dataLen	short	输入	数据长度

注意：压入数据时，先压入指令字，然后再压入指令字对应的工作数据，总的的数据长度不超过1000个字节；

7.8.2 代码示例

```
//打包插补
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    /***** 已省略坐标系初始化部分，见上节7.1.2 *****/
    short offset = 0;
    short cmdType = 0;
    unsigned char bufData[1024];
    memset(bufData, 0, sizeof(bufData));
    for (int i = 10; i >= 0; i--) {
        cmdType = BUF_LINE;
        TCrdBufLine crdBufLine;
        memset(&crdBufLine, 0, sizeof(crdBufLine));
        crdBufLine.segNo = i + 1;
        crdBufLine.tgPos[0] = (i + 1) * 2000;
        crdBufLine.tgPos[1] = (i + 1) * 1000;
        crdBufLine.tgPos[2] = 0;
        crdBufLine.endVel = 4;
        crdBufLine.vel = 10;
        crdBufLine.synAcc = 1;
        crdBufLine.mask = 0x3;
        crdBufLine.lookaheadDis = 0;
        memcpy(&bufData[offset], &cmdType, sizeof(short));
        offset += sizeof(short);
        memcpy(&bufData[offset], &crdBufLine, sizeof(TCrdBufLine));
        offset += sizeof(TCrdBufLine);
        if (offset >= 0)
```

```

    {
        rtn = NMC_CrdBufDataPack(crdHandle, bufData, offset);
        gc_rtn_error(rtn);
        memset(bufData, 0, sizeof(bufData));
        offset = 0;
    }
}
if (offset > 0)
{
    rtn = NMC_CrdBufDataPack(crdHandle, bufData, offset); // 打包插补数据
    gc_rtn_error(rtn);
}
/***** 省略启动插补部分 *****/
return rtn;
}

```

7.9 坐标系的状态检测

7.9.1 指令说明

(1) 读取坐标系状态

[NMC_CrdGetSts\(HAND crdHandle, short *pStsWord\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pStsWord	short *	输入	返回状态字, 参考宏定义 <pre> #define BIT_CORD_BUSY 0x00000001 // bit 0 , 运动:1, 静止 0, 立即运动下 运动停止, 完成 #define BIT_CORD_MVERR 0x00000002 // bit 1 , 运动出错, 或当前运动指令无 法启动, 需要软件复位 #define BIT_CORD_EMPTY 0x00000004 // bit 2 , 缓冲区空 #define BIT_CORD_FULL 0x00000008 // bit 3 , 缓冲区满 </pre>

			<pre>#define BIT_CORD_NODATASTOP 0x00000010 // bit 4, 缓冲区空异常 停止或者急停 #define BIT_CORD_SDRAM_HWERR 0x00000020 // bit 5, 插补缓冲区硬件 或者其他错误</pre>
--	--	--	---

(2) 读取规划位置 XYZ

[NMC_CrdGetPrfPos\(HAND crdHandle, short cnts, long *pPosArray\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
cnts	short	输入	读取个数, 1~N
pPosArray	long *	输出	返回坐标数组

(3) 坐标系模式下, 读取多个轴的机械坐标位置

[NMC_CrdGetAxisPos\(HAND crdHandle, short cnts, long *pPosArray\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
cnts	short	输入	读取个数, 1~N
pPosArray	long *	输出	返回坐标数组

(4) 读取坐标系合成速度

[NMC_CrdGetVel\(HAND crdHandle, double *pVel\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pVel	double *	输出	坐标系合成速度

(5) 读取编码器位置

[NMC_CrdGetEncPos\(HAND crdHandle, short cnts, long *pPosArray\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
cnts	short	输出	读取个数, 1~N
pPosArray	long *	输出	返回坐标数组

(6) 坐标系运动模式下, 打包读取控制器状态

[NMC_CrdGetStsPack4\(HAND crdHandle, TPackedCrdSts4 *pPackSts\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pPackSts	TPackedCrdSts4 *	输出	<pre>typedef struct{ short crdSts; // 坐标系状态 short axSts[4]; // 坐标系里各轴状态 long prfPos[4]; // 用户坐标系下的规划位置 long axisPos[4]; // 机械坐标系下的规划位置 long encPos[4]; // 编码器位置 long userSeg; // 运行的缓冲区段号 double prfVel; // 运动速度 long gpDi; // 通用输入 0~31 long gpDo; // 通用输出 0~31 short motDi[4]; // 限位、原点、报警。请参考专用 IO 位定义(搜索 BIT_AXMTIO_LMTN) short reserved; // 保留 long crdFreeSpace; // 缓冲区剩余空间 long crdUsedSpace; }TPackedCrdSts4;</pre>

(7) 读取指令缓冲区空闲长度

[NMC_CrdBufGetFree\(HAND crdHandle, long *pRes \);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

crdHandle	HAND	输入	坐标系句柄
pRes	long *	输出	缓冲区中还未执行的指令个数

(8) 读取指令缓冲区已用长度

[NMC_CrdBufGetUsed\(HAND crdHandle, long *pLen \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pLen	long *	输出	读取指令缓冲区已用长度

(9) 读取段号

[NMC_CrdGetUserSegNo\(HAND crdHandle, long *pSegNo\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pSegNo	long *	输出	返回的当前段号

(10) 读取总共压了多少条指令

[NMC_CrdGetBufAllCmdCnt\(HAND crdHandle, long *pCnt\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pCnt	long *	输出	返回的指令数

(11) 读取内部坐标系状态

[NMC_CrdGetInnerSts\(HAND crdHandle, long *pStsWord\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pStsWord	long *	输出	// 坐标系状态字位定义:内部扩展 #define BIT_CORD_POSREC 0x00000040 // bit 6 , 伺服位置 到达, 步进模式时位置到达, 伺服模式

		<p>时实际位置到达误差限</p> <pre>#define BIT_CORD_AUXAXIS_BUSY 0x00000080 // bit 7 , 坐标系运 动中的关联轴启动前处于运动状态错 误 #define BIT_CORD_AUXAXIS_ERR 0x00000100 // bit 8 , 插补辅助 轴错误 #define BIT_CORD_AXIS_ERR 0x00000200 // bit 9 , 插补轴存 在报警错误(如限位、驱动报警) #define BIT_CORD_SDRAM_CALC_ERR 0x00000400 // bit 10 , SDRAM 缓冲区计算错误 #define BIT_CORD_SCARA_CALC_ERR 0x00000800 // bit 11 , SCARA 计 算数据错误</pre>
--	--	--

(12) 读取插补缓冲区中尚未完成的总位移量

[NMC_CrdGetBufLeftLength\(HAND crdHandle, double *pLen\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pLen	double *	输出	长度, 单位 脉冲

7.9.2 代码示例

坐标系状态检测常用于读取坐标系的运动状态, 运动是否完成, 运行中是否出错等, 展示例程请看下章节 7.10.2;

7.10 坐标系的其他指令

7.10.1 指令说明

(1) 坐标系缓冲运动启动

[NMC_CrdStartMtn\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(2) 结束(指令压入)缓冲区运动（等待运动完后才结束压入，并置空闲标志）

[NMC_CrdEndMtn\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(3) 立即平滑停止运动

[NMC_CrdStopMtn\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

注意：不清空指令缓冲区，再次启动能继续运行缓冲区剩余指令；

(4) 清除坐标系运动错误状态, 同时清除所包含轴的错误状态

[NMC_CrdClrError\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(5) 指令缓冲区清空

[NMC_CrdBufClr\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(6) 设置坐标系速度倍率

[NMC_CrdSetOverRide\(HAND_crdHandle, double_overRide\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
overRide	double	输入	坐标系速度倍率, 取值范围 (0, 10)

(7) 读取坐标系速度倍率

[NMC_CrdGetOverRide\(HAND crdHandle, double *pOverRide\);](#)

参考：略

(8) 设置轴缓冲区运动偏移

[NMC_CrdSetOffset\(HAND crdHandle, short count, long *pOffsetArray \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
count	short	输入	设置偏移的轴数
pOffsetArray	long *	输出	缓冲区运动偏移, long 数组, 会同时修改坐标系内相关轴的运动偏移

(9) 急停, 不清空指令缓冲区

[NMC_CrdEstopMtn\(HAND crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(10) 返回缓冲区运动的断点

[NMC_CrdGotoBreak\(HAND crdHandle, double acc, double vel\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
acc	double	输入	返回时使用的加速度, 单位 pulse/ms ²
vel	double	输入	返回时使用的速度, 单位 pulse/ms

(11) 开始计算缓冲区执行时间

[NMC_CrdStartExeTimeCalc\(void\);](#)

参考：略

注意：如果需要计算缓冲区执行时间，需要使用上位机前瞻，开始后，所有的缓冲区指令，都不会压入控制器；

(12) 读取缓冲区指令的执行时间，并停止计算，单位:ms

[NMC_CrdGetExeTime\(double *pTime\);](#)

名称	数据类型	输入/输出	描述
pTime	double *	输出	缓冲区指令的执行时间，单位 ms

(13) 设置是否计算所有线段长度

[NMC_CrdSetBufLengthFlag\(HAND crdHandle, short flag\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
flag	short	输入	是否计算所有线段长度，1，是，0，否

(14) 启动坐标系的干涉保护

[NMC_SetMvProtect\(HAND devHandle, short groupNo, short mode\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输出	控制器句柄
groupNo	short	输入	干涉坐标系组号(目前只有两个坐标系，所以该值为 0)
mode	short	输入	0:基本模式，出现干涉，两坐标系均停止 1:激光模式，出现干涉，其中一个停止并离开

(15) 取消坐标系的干涉保护

[NMC_DelMvProtect\(HAND devHandle, short groupNo\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
groupNo	short	输入	干涉坐标系组号(目前只有两个坐标系, 所以该值为 0)

(16) 读取坐标系的干涉状态

[NMC_GetMvProtectSts\(HAND devHandle, short groupNo, short *pSts\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
groupNo	short	输入	干涉坐标系组号(目前只有两个坐标系, 所以该值为 0)
pSts	short *	输出	0 没有启动干涉功能 1 没有出现干涉 2 干涉发生

(17) 设置坐标系的干涉保护参数

[NMC_SetMvProtectMode01Para\(HAND devHandle, short groupNo, TMvProtectPara *pMvProtectPara\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
groupNo	short	输入	干涉坐标系组号(目前只有两个坐标系, 所以该值为 0)
pMvProtectPara	TMvProtectPara *	输入	<pre>typedef struct{ short crdNo[2]; // 需要干涉保护的 两坐标系号 short crdAxNo[2]; // 坐标系的 干涉轴号, 指 XYZ 中哪一个(X:0 Y:1 Z:2) short moveDir[2]; // 两干涉轴分 别往距离减小时的运动方向, 1: 正向 -1: 负向</pre>

			<pre> long orgDis; // 两坐标系的 crdAxNo 轴在 origin 时的距离 long mvProtectDis; // 干涉的保护距离 long safeWaitPos; // mode 为激光模式时，干涉轴需要停止到的等待位置(mode 为 0 时无效) double mvToSafeAcc; // 运动到等待位置的移动加速度 double mvToSafeVel; // 运动到等待位置的移动速度 short stopCrdNo; // mode 为激光模式时，需要停止的坐标系号(mode 为 0 时无效) short reserved[3]; // 保留参数 }TmvProtectPara; </pre>
--	--	--	---

(18) 读取坐标系的干涉保护参数

[NMC_GetMvProtectMode01Para\(HAND devHandle, short groupNo, TmvProtectPara *pMvProtectPara\);](#)

参考: [NMC_SetMvProtectMode01Para](#)

7.10.2 代码示例

```

//启动插补
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    /***** 已省略坐标系初始化部分，见上节7.1.2 *****/
    /***** 省略压入插补部分，见上几节 *****/
    /***** 启动插补部分 *****/

    short crdsts = 0;
    double vel = 0;

```

```

long segnum = 0;
long fre = 0;
rtn = NMC_CrdEndMtn(crdHandle);           //结束压入
gc_rtn_error(rtn);
rtn = NMC_CrdStartMtn(crdHandle);         //启动坐标系
gc_rtn_error(rtn);
while (true)
{
    rtn = NMC_CrdGetVel(crdHandle, &vel);   //读取插补速度
    gc_rtn_error(rtn);
    rtn = NMC_CrdGetUserSegNo(crdHandle, &segnum); //读取正在运行的段号
    gc_rtn_error(rtn);
    rtn = NMC_CrdBufGetFree(crdHandle, &fre); //读取缓存区剩余空间
    gc_rtn_error(rtn);
    rtn = NMC_CrdGetSts(crdHandle, &crdsts); //读取坐标系状态
    gc_rtn_error(rtn);
    if ((crdsts & BIT_CORD_BUSY) == 0) { break; /*运动完成*/}
    if ((crdsts & BIT_CORD_MVERR) == 1) { /*运动出错，做处理，此处略*/}
}
return rtn;
}

```

7.11 综合示例

/*坐标系运动完整案例，建立一个三轴坐标系，以(0, 0, 0)为起点，(10000, 0, 0)为终点，(5000, 0, 0)为圆心坐标，逆时针绘制一个半圆。(注意，圆弧插补时，圆心参数实际为圆心相对起点偏移量，而不是圆心坐标)*/

```

short gc_example(void)
{
    /****** 已省略打开控制器部分，统一描述在第五章1 *****/
    // 坐标系初始化
    rtn = NMC_CrdOpen(devHandle, &crdHandle); //打开坐标系
    gc_rtn_error(rtn);
    TCrdConfig crd;
    crd.axCnts = 3; // 设置3轴坐标系
    crd.pAxArray[0] = 0; // 轴端口0对应X轴
    crd.pAxArray[1] = 1; // 轴端口1对应Y轴
    crd.pAxArray[2] = 2; // 轴端口2对应Z轴
    crd.pAxArray[3] = -1; // 轴端口3对应A轴，-1为不使用
    crd.port[0] = 0; // 端口设为0
    crd.port[1] = 0; // 端口设为0
    crd.port[2] = 0; // 端口设为0
    crd.port[3] = 0; // 端口设为0
    rtn = NMC_CrdConfig(crdHandle, &crd); //配置坐标系
}

```

```
gc_rtn_error(rtn);
TCrdPara crdPara;
crdPara.orgFlag = 1; // 默认原点(0,0,0,0)
crdPara.offset[0] = 0; // 坐标原点设为0
crdPara.offset[1] = 0; // 坐标原点设为0
crdPara.offset[2] = 0; // 坐标原点设为0
crdPara.offset[3] = 0; // 坐标原点设为0
crdPara.synAccMax = 10; //最大合成加速度p/ms2
crdPara.synVelMax = 50; //最大合成速度,允许范围内,两个参数尽可能大
rtn = NMC_CrdSetPara(crdHandle, &crdPara); // 设置运动参数
gc_rtn_error(rtn);
/***** 以下配置可选 *****/
TCrdSafePara crdSafePara;
crdSafePara.estpDec = 1000; //急停加速度
crdSafePara.maxAcc = 50; //设置坐标系最大加速度50p/ms2
crdSafePara.maxVel = 800; //设置坐标系最大速度800p/ms
rtn = NMC_CrdSetSafePara(crdHandle, &crdSafePara); //设置安全参数
gc_rtn_error(rtn);
/***** 以上配置可选 *****/
rtn = NMC_CrdClrError(crdHandle); //清除坐标系错误状态
gc_rtn_error(rtn);
rtn = NMC_CrdBufClr(crdHandle); //压入指令之前需要清空缓存区
gc_rtn_error(rtn);
// 往缓冲区压入数据,其他缓冲区指令请参考编程手册和头文件
//直线插补,运动到(0,0,0)
long segNo = 1;
long targetPos[3] = { 0,0,0 };
long centerPos[3] = { 5000,0,0 };
double vel = 2;
double acc = 0.5;
rtn = NMC_CrdLineXYZ(crdHandle, segNo++, 0x7, targetPos, 0, vel, acc);
gc_rtn_error(rtn);
//圆弧插补
targetPos[0] = 10000;
targetPos[1] = 0;
rtn = NMC_CrdArcCenter(crdHandle, segNo++, targetPos, centerPos, 1, 0, vel, acc);
gc_rtn_error(rtn);
rtn = NMC_CrdBufDelay(crdHandle, segNo++, CRD_BUFF_DELAY_SCALE_MS, 100); // 延时毫秒
gc_rtn_error(rtn);
rtn = NMC_CrdBufDo(crdHandle, segNo++, CRD_BUFF_DO_GPD01, 1, 0); // 输出DO
gc_rtn_error(rtn);
// 结束数据压入
rtn = NMC_CrdEndMtn(crdHandle);
gc_rtn_error(rtn);
```

```
// 启动缓冲区运动
rtn = NMC_CrdStartMtn(crdHandle);
gc_rtn_error(rtn);
short crdSts;
// 等待运动结束
while (true)
{
    rtn = NMC_CrdGetSts(crdHandle, &crdSts);
    gc_rtn_error(rtn);
    if ((crdSts & BIT_CORD_BUSY) == 0) { break; /*运动完成*/ }
}
return 0;
}
```


8 IO资源

8.1 功能介绍

控制器包括以下几类硬件资源：

数字量输入：通用输入、专用输入(正负限位、驱动报警、原点)；

数字量输出：通用输出、专用输出(报警清除、伺服使能)；

模拟量输入；

模拟量输出；

8.2 指令说明

(1)按通道设置通用输出(支持超过 32 位)，与 [NMC_SetDO](#) 指令功能相同

[NMC_SetDOGroup\(HAND devHandle, long value, short groupID\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
value	long	输入	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
groupID	short	输入	DO 组, 取值范围[0, n], 0: 本地 D00~D031, 1: 本地 D032~D063, 其他指扩展 IO 模块

(2)按位设置通用输出

[NMC_SetDOBit\(HAND devHandle, short bitIndex, short value\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 DO
value	short	输入	设置输出电平, 默认情况下, 1 表示高电平, 0 表示低电平

(3)按通道读取通用输出(支持超过 32 位)，与 [NMC_GetDO](#) 指令功能相同

[NMC_GetDOGroup\(HAND devHandle, long *pDoValue, short groupID\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
groupID	short	输入	D0 组, 取值范围[0, n], 0: 本地 D00~D031, 1: 本地 D032~D063, 其他指扩展 IO 模块
value	long*	输出	返回输出电平, 默认情况下, 1 表示高电平, 0 表示低电平。

(4) 按位读取通用输出

[NMC_GetDOBit\(HAND devHandle, short bitIndex, short *bitValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	位序号 0~n
bitValue	short*	输出	返回通用数字量输出状态, 默认情况下, 1 表示高电平, 0 表示低电平

注: 网络控制器不适用;

(5) 按通道读取通用输入(支持超过 32 位), 与 [NMC_GetDI](#) 指令功能相同

[NMC_GetDIGroup\(HAND devHandle, long *pInValue, short groupID\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
groupID	short	输入	DI 组, 取值范围[0, n], 0: 本地 DI0~DI31, 1: 本地 DI32~DI63, 其他指扩展 IO 模块
inValue	long *	输出	按位指示返回通用数字量输入状态 1, 表示高电平/开路, 0, 表示低电平/闭合

(6) 按位读取通用输入

[NMC_GetDIBit\(HAND devHandle, short bitIndex, short *bitValue\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	位序号 0~n
bitValue	short*	输出	返回通用数字量输入状态, 1 表示高电平/开路, 0 表示低电平/闭合

(7) 扩展 IO 模块通道设置, 16 位长度

[NMC_IOModuleWr16Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned short data\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n
offset	unsigned char	输入	地址偏移, 单位字节
data	unsigned short	输入	写入数据

(8) 扩展 IO 模块通道设置, 32 位长度

[NMC_IOModuleWr32Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned long data\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n
offset	unsigned char	输入	地址偏移, 单位字节
data	unsigned long	输入	写入数据

(9) 扩展 IO 模块通道读取输出, 16 位长度

[NMC_IOModuleRdOut16Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned short *pData\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n

offset	unsigned char	输入	地址偏移，单位字节
pData	unsigned short*	输出	返回数据

(10) 扩展 IO 模块通道读取输出，32 位长度

[NMC_IOModuleRdOut32Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned long *pData\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n
offset	unsigned char	输入	地址偏移，单位字节
pData	unsigned long*	输出	返回数据

(11) 扩展 IO 模块通道读取输入，16 位长度

[NMC_IOModuleRd16Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned short *pData\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n
offset	unsigned char	输入	地址偏移，单位字节
pData	unsigned short*	输出	返回数据

(12) 扩展 IO 模块通道读取输入，32 位长度

[NMC_IOModuleRd32Bit\(HAND devHandle, unsigned char chDevId, unsigned char offset, unsigned long *pData\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	从站序号 2~n
offset	unsigned char	输入	地址偏移，单位字节
pData	unsigned long*	输出	返回数据

(13) 读取轴专用 IO

[NMC_MtGetMotionIO\(HAND axisHandle, long *inValue\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
inValue	long *	输出	返回专用数字量输入状态，位定义如下： bit 0：负向限位 bit 1：正向限位 bit 2：原点 bit 3：驱动报警 bit 4：电机到位 bit 5：捕获源信号

(14) 读取运动控制专用 IO, 逻辑电平

[NMC_MtGetMotionIOLogical\(HAND axisHandle, long *pIoValue\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pIoValue	long *	输出	返回专用 IO 的状态，原点，限位，报警。 参考位定义。对应位 0 为低电平，1 为高电平

(15) 按位设置通用输入信号取反(不会改变灯的状态)

[NMC_SetDIBitRevs\(HAND devHandle, short bitIndex, short revs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输入, 大于 64 为扩展 DI
revs	short	输入	是否取反, 1: 取反, 0: 不取反

(16) 按位读取通用输入信号是否取反

[NMC_GetDIBitRevs\(HAND devHandle, short bitIndex, short *pRevs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输入, 大于 64 为扩展 Di
pRevs	short *	输出	是否取反, 1: 取反, 0: 不取反

(17) 按位设置通用输出信号取反(会改变灯的状态)

[NMC_SetDOBitRevs\(HAND devHandle, short bitIndex, short revs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 Do
revs	short	输入	是否取反, 1: 取反, 0: 不取反

(18) 按位读取通用输出信号取反的设置值

[NMC_GetDOBitRevs\(HAND devHandle, short bitIndex, short *pRevs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 Do
pRevs	short *	输出	是否取反, 1: 取反, 0: 不取反

(19) 按位设置输出信号取反的设置值

[NMC_SetDOBitRevsEx\(HAND devHandle, short doType, short group, short bitIndex, short revs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
doType	short	输入	#define DO_TYPE_MOTOR_ENABLE 1//电机

			使能 #define DO_TYPE_MOTOR_CLEA 2 //电机报警清除 #define DO_TYPE_GPDO 3 // 通用输出
group	short	输入	组号，暂时只对 GP0 有效，指 DO 的组号
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 Do
revs	short	输入	是否取反, 1: 取反, 0: 不取反

(20) 按位读取输出信号取反的设置值

[NMC_GetDOBitRevsEx\(HAND devHandle, short doType, short group, short bitIndex, short *pRevs\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
doType	short	输入	#define DO_TYPE_MOTOR_ENABLE 1//电机使能 #define DO_TYPE_MOTOR_CLEA 2 //电机报警清除 #define DO_TYPE_GPDO 3 // 通用输出
group	short	输入	组号，暂时只对 GP0 有效，指 DO 的组号
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 Do
pRevs	short *	输出	是否取反, 1: 取反, 0: 不取反

(21) DO 输出持续设定时间后翻转

[NMC_SetDOBitAutoReverse\(HAND devHandle, short bitIndex, short value, short reverseTime\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
bitIndex	short	输入	取值范围[0, n], 位序号, 前 64 位为本地的通用输出, 大于 64 为扩展 Do
value	short	输入	设置通用数字量输出。1, 输出高电

			平, 0, 输出低电平
reverseTime	short	输入	持续的电平, 单位: 毫秒

(22) 读取扩展 I/O 模块的状态

[NMC_GetIOModuleSts\(HAND devHandle, unsigned long *sts\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
sts	unsigned long *	输出	I/O 模块状态

(23) 设置扩展 I/O 模块有效(带模块类型)

[NMC_IOModuleSetEn\(HAND devHandle, unsigned char chDevId, short chDevType\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	控制器 ID
chDevType	short	输入	模块类型, 见宏定义 // 扩展模块类型定义 <pre>#define IOMODULE_TYPE_I064 1 // 32DI 32DO 模块(包括 16DI16DO 模块) #define IOMODULE_TYPE_I032_DA 2 // 4AD4DA 模块</pre>

(24) 读取扩展 I/O 模块类型

[NMC_IOModuleGetType\(HAND devHandle, unsigned char chDevId, short *pChDevType\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
chDevId	unsigned char	输入	设备 ID
chDevType	short	输出	返回的模块类型, 见宏定义 // 扩展模块类型定义

			<pre>#define IOMODULE_TYPE_I064 1 // 32DI32DO 模块(包括 16DI16DO 模块) #define IOMODULE_TYPE_I032_DA 2 // 4AD4DA 模块</pre>
--	--	--	---

(25) 设置伺服 ON，轴静止时执行，如果后面是运动指令，需要延时一个周期

[NMC_MtSetSvOn\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(26) 设置伺服 OFF，轴静止时执行

[NMC_MtSetSvOff\(HAND axisHandle \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(27) 设置伺服报警清除输出

[NMC_MtSetSvClr\(HAND axisHandle, short swt \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
swt	short	输入	设置开关有效状态。0 有效(低电平), 1, 无效(高电平)

(28) 设置 DAC(模拟量输出)通道的模式

[NMC_SetDacMode\(HAND devHandle, short ch, short mode\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输出通道号, 取值范围[0, n]
mode	short	输入	模拟量输出范围, 0:0~5V, 1:0~10V,

			2: 0~10.8V, 3: +/-5V, 4: +/-10V, 5: +/-10.8V, 其他无效, 默认为 4
--	--	--	--

(29) 读取 DAC (模拟量输出) 通道的模式

[NMC_GetDacMode\(HAND devHandle, short ch, short *pMode\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输出通道号, 取值范围[0, n]
pMode	short *	输出	模拟量输出范围, 0: 0~5V, 1: 0~10V, 2: 0~10.8V, 3: +/-5V, 4: +/-10V, 5: +/-10.8V

(30) 设置 ADC 参数

[NMC_SetAdcMode\(HAND devHandle, short ch, short range, short filterCoe\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输入通道号, 0~7 表示轴通道上的 AD, 256: 表示扩展 AD
range	short	输入	模拟量范围 0: 0~5V, 1: 0~10V, 2: 0~10.8V, 3: +/-5V, 4: +/-10V, 5: +/-10.8V, 默认为 4, 目前只支持 +/-10V
filterCoe	short	输入	滤波系数, 取值范围[0, 64], 0 表示取消 ADC 滤波, 单位: ms, 默认值为 0

(31) 读取 ADC 参数

[NMC_GetAdcMode\(HAND devHandle, short ch, short *pRange, short *pFilterCoe\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量通道号, 0~7 表示轴通道上的

			AD, 256: 表示扩展 AD
pRange	short *	输出	模拟量范围 0:0~5V, 1:0~10V, 2:0~10.8V, 3:+/-5V, 4:+/-10V, 5:+/-10.8V, 默认为 4, 目前只支持 +/-10V
pFilterCoe	short *	输出	滤波系数, 取值范围[0, 64], 0 表示取消 ADC 滤波, 单位:ms, 默认值为 0

(32) 设置 DAC(模拟量输出) 输出值

[NMC_SetDac\(HAND devHandle, short ch, short dacValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输出通道号, 取值范围[0, n]
dacValue	short	输出	模拟量输出值, 取值范围[-32768, 32767], 对应 DAC 输出范围

(33) 读取 DAC(模拟量输出) 输出值

[NMC_GetDac\(HAND devHandle, short ch, short *pDacValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输出通道号, 取值范围[0, n]
pDacValue	short *	输出	模拟量输出值, 取值范围[-32768, 32767], 对应 DAC 输出范围

(34) 读取 ADC(模拟量输入) 输入值

[NMC_GetAdc\(HAND devHandle, short ch, short *pAdcValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量输入通道号, 取值范围[0, n]
pAdcValue	short *	输出	模拟量输入值, 取值范围[-

			32768, 32767], 对应 ADC 输出范围
--	--	--	--------------------------------

(35) 增加一组 DIO 映射，映射关系目前最多存在 8 组

[NMC_SetDioMapping\(HAND devHandle, TDioMappingCfg *pDioCfg\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pDioCfg	TDioMappingCfg *	输入	// DIO 映射参数配置 typedef struct { unsigned char enable; // 1: 启用, 0: 禁用 unsigned char pinGrp; // 映射的信号类型 unsigned char pinIndex; // 映射的信号序号 unsigned char outEnable; // 输出允许 unsigned char newGrp; // 映射到的信号类型 unsigned char newIndex; // 映射到的信号序号 }TDioMappingCfg;

(36) 读取所有的 DIO 映射数据

[NMC_GetAllDioMapping\(HAND devHandle, TDioMappingCfg *pDioCfg\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pDioCfg	TDioMappingCfg *	输出	// DIO 映射参数配置 typedef struct {

			<pre> unsigned char enable; // 1: 启用, 0: 禁用 unsigned char pinGrp; // 映射的信号类型 unsigned char pinIndex; // 映射的信号序号 unsigned char outEnable; // 输出允许 unsigned char newGrp; // 映射到的信号类型 unsigned char newIndex; // 映射到的信号序号 }TDioMappingCfg; </pre>
--	--	--	--

(37)清除所有的 DIO 映射关系

[NMC_ClrAllDioMapping\(HAND devHandle\);](#)

参考：略

(38)设置模拟量输出的偏移值

[NMC_SetDacBias\(HAND devHandle, short ch, short bias\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量通道号, [0, n]表示本地, [256, n]表示扩展模拟量通道
bias	short	输入	DAC 输出偏移值

(39)读取模拟量输出的偏移值

[NMC_GetDacBias\(HAND devHandle, short ch, short *pBias\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量通道号, [0, n]表示本地, [256, n]表示扩展模拟量通道
pBias	short *	输出	DAC 输出偏移值

(40) 设置模拟量输出的极限值

[NMC_SetDacLmt\(HAND devHandle, short ch, short maxVal, short minVal\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量通道号, [0, n]表示本地, [256, n]表示扩展模拟量通道
maxVal	short	输入	输出最大值
minVal	short	输入	输出最小值

(41) 读取模拟量输出的极限值

[NMC_GetDacLmt\(HAND devHandle, short ch, short *pMaxVal, short *pMinVal\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	模拟量通道号, [0, n]表示本地, [256, n]表示扩展模拟量通道
pMaxVal	short *	输出	输出最大值
pMinVal	short *	输出	输出最小值

(42) 设置 DI 的滤波系数

[NMC_SetDIFilter\(HAND devHandle, short diType, short diGroup, short diIndex, short filtTime\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输入	DI 类型, 见 DI_type 宏定义

diGroup	short	输入	DI 组，取值范围 [0,n]，对于 DI_TYPE_GPI, 则 0: 本地 DI31~DI0, 1: 本地 DI63~DI32, 其他指扩展 IO 模块；对于其他 DI 类型，暂时保留
diIndex	short	输入	DI 通道号，取值范围 [0, n]
filtTime	short	输入	滤波时间，单位 ms，默认为 3, 取值范围 [0, 32]

(43) 读取 DI 的滤波系数

[NMC GetDIFilter\(HAND devHandle, short diType, short diGroup, short diIndex, short *pFiltTime\);](#)

参考: [NMC SetDIFilter](#)

(44) 读取数字量输入信号值

[NMC GetDIEx\(HAND devHandle, short diType, short groupID, long *pDiValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输入	DI 类型, 见 DI type 宏定义
groupID	short	输入	DI 组，取值范围 [0,n]，对于 DI_TYPE_GPI, 则 0: 本地 DI31~DI0, 1: 本地 DI63~DI32, 其他指扩展 IO 模块；对于其他 DI 类型，暂时保留
pDiValue	long *	输出	返回的输入值

(45) 读取数字量输入信号值(原始值)

[NMC GetDIRaw\(HAND devHandle, short diType, short groupID, long *pDiValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输出	DI 类型, 见 DI type 宏定义

groupId	short	输入	DI 组，取值范围 [0,n]，对于 DI_TYPE_GPI，则 0：本地 DI31~DI0，1：本地 DI63~DI32，其他指扩展 IO 模块；对于其他 DI 类型，暂时保留
pDiValue	long *	输出	返回的输入值

(46) 设置数字量输入信号的翻转计数

[NMC_SetDiReverseCount\(HAND devHandle, short diType, short diIndex, unsigned long *pReverseCountArray, short count\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输入	DI 类型, 见 DI_type 宏定义
diIndex	short	输入	DI 序号, 取值范围 [0, n]，对于通用输入，只支持 0~15，对于其他只支持 0~7
pReverseCountArray	unsigned long *	输出	返回的翻转计数数组
count	short	输出	同时读取的数量，取值范围 [1, 16]

(47) 读取数字量输入信号的翻转计数

[NMC_GetDiReverseCount\(HAND devHandle, short diType, short diIndex, unsigned long *pReverseCountArray, short count\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
diType	short	输入	DI 类型, 见 DI_type 宏定义
diIndex	short	输入	DI 序号, 取值范围 [0, n]
pReverseCountArray	unsigned long *	输出	翻转计数数组
count	short	输出	同时读取的数量，取值范围 [1, 16]

(48) 输出 DO, 按照掩码

[NMC_SetDOMask\(HAND devHandle, long outMask, long outValue ,short doType, short groupID\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
outMask	long	输出	按位输出掩码
outValue	long	输出	按位输出数值
doType	short	输入	//D0 类型, 见宏定义 // 缓冲区输出 D0 组定义 #define CRD_BUFF_DO_MOTOR_ENABLE 1 // 电机使能 #define CRD_BUFF_DO_MOTOR_CLEAR 2 // 电机报警清除 #define CRD_BUFF_DO_GPD01 3 // 通用输出 1 #define CRD_BUFF_DO_GPD02 4 // 通用输出 2 #define CRD_BUFF_DO_EXTD01 5 // 扩展模块 1 #define CRD_BUFF_DO_EXTD02 6 // 扩展模块 2 #define CRD_BUFF_DO_EXTD03 7 // 扩展模块 3 #define CRD_BUFF_DO_EXTD04 8 // 扩展模块 4 #define CRD_BUFF_DO_EXTD05 9 // 扩展模块 5 #define CRD_BUFF_DO_EXTD06 10 // 扩展模块 6
groupID	short	输入	组号

(49) 读取模拟量输入电压值

[NMC_GetAdcVoltageMulti\(HAND devHandle, short ch, short count, double *pAdcVolArray\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号, [0, n]表示本地, 256 表示扩展模拟量通道
count	short	输入	读取的通道数量, 取值范围[1, 8]

pAdcVolArray	double *	输出	读取的电压值数组
--------------	----------	----	----------

(50) 设置模拟量输出电压值

[NMC_SetDacVoltageMulti\(HAND devHandle, short ch, short count, double *pDacVolArray \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号, [0, n]表示本地, 256 表示扩展模拟量通道
count	short	输出	读取的通道数量, 取值范围[1, 8]
pDacVolArray	double *	输入	设置的电压值数组

(51) 读取模拟量输出电压值

[NMC_GetDacVoltageMulti\(HAND devHandle, short ch, short count, double *pDacVolArray \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号, [0, n]表示本地, 256 表示扩展模拟量通道
count	short	输出	读取的通道数量, 取值范围[1, 8]
pDacVolArray	double *	输出	读取的电压值数组

(52) 读取输出 DO

[NMC_GetDOEx\(HAND devHandle, short doType, short groupID, long *pOutValue \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
doType	short	输入	#define DO_TYPE_MOTOR_ENABLE 1 // 电机使能 #define DO_TYPE_MOTOR_CLEAR 2 // 电机报警清除 #define DO_TYPE_GPDO 3 // 通用输出 1
groupID	short	输入	DO 组, 取值范围 [0, n], 0: 本地

			DI0~DI31, 1: 本地 DI32~DI63, 其他 指扩展 IO 模块
pOutValue	long *	输出	读取的数值

(53)DI 沿触发

[NMC_MtPtStartMtnEx\(HAND axisHandle, short diIndex, short diSns\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
diIndex	short	输入	通用输入序号, 只支持通用输入信号, 取值范围[0, 32]
diSns	short	输入	触发沿, 0 为下降沿, 1 为上升沿

8.3 综合示例

```
//数字量输入输出, 模拟量读取/设置
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    long diVal = 0;
    long diValEx = 0;
    long doVal = 0;
    long doValEx = 0;
    short di0 = 0;
    short di64 = 0;
    long axisIO = 0;
    bool IsExmoNeed = false;
    //如果需要读取扩展模块
    if (IsExmoNeed == true) {
        //设备ID需要从2开始, 多个扩展模块, 依次增加, 拨码开关也要拨码成对应ID
        rtn = NMC_IOModuleSetEn(devHandle, 2, 1);
    }
    rtn = NMC_GetDIGroup(devHandle, &diVal, 0); //读取全部本地通用数字量输入
    gc_rtn_error(rtn);
    rtn = NMC_GetDIBit(devHandle, 0, &di0); //按位读取(单个读取), 这里示范读取第0个输入
    gc_rtn_error(rtn);
    rtn = NMC_GetDIGroup(devHandle, &diValEx, 2); //全部读取扩展模, 这里读取一个模块的
    gc_rtn_error(rtn);
}
```

```
rtn = NMC_GetDIBit(devHandle, 64, &di64); //按位读取(单个读取), 读取扩展模块第0个输入
gc_rtn_error(rtn);
rtn = NMC_GetDOGroup(devHandle, &doVal, 0); //读取本地全部通用数字量输出
gc_rtn_error(rtn);
rtn = NMC_GetDOGroup(devHandle, &doValEx, 2); //读取扩展模块
gc_rtn_error(rtn);
rtn = NMC_SetDOGroup(devHandle, 0, 0); //设置通用数字量全部输出
gc_rtn_error(rtn);
rtn = NMC_SetDOGroup(devHandle, 0xFFFD, 0); //设置通用数字量第1个输出
gc_rtn_error(rtn);
rtn = NMC_SetDOBit(devHandle, 0, 0); //设置通用数字量第0个输出, 按位输出
gc_rtn_error(rtn);
rtn = NMC_MtGetMotionIO(axisHandle, &axisIO); //专用IO, 如限位原点的电平读取
if ((axisIO & (1 << 0)) != 0) {} //负限位为高电平
if ((axisIO & (1 << 1)) != 0) {} //正限位为高电平
if ((axisIO & (1 << 2)) != 0) {} //原点为高电平
rtn = NMC_SetDacMode(devHandle, 0, 4); //模拟量设置, 默认, +-10V, 此函数可省略
gc_rtn_error(rtn);
rtn = NMC_SetDac(devHandle, 0, 32767); //模拟量设置输出满值
short dac = 0;
rtn = NMC_GetDac(devHandle, 0, &dac); //读取模拟输出
gc_rtn_error(rtn);
short adc = 0;
rtn = NMC_SetAdcMode(devHandle, 0, 4, 0); //设置ADC, 默认, +-10V, [-32767, 32767]
gc_rtn_error(rtn);
rtn = NMC_GetAdc(devHandle, 0, &adc); // 读取ADC
gc_rtn_error(rtn);
return 0;
}
```

9 手脉功能

9.1 功能介绍

通过手摇脉冲发生器(电子手轮)如图 9.1.1，把脉冲信号(外部)发送给运动控制器，运动控制器再下发指令给伺服电机，控制着伺服电机的运动；手轮功能可以设置任意一个轴跟随编码器运动；编码器信号源为控制器上的“手脉输入/辅助编码器”。



9.1.1 电子手轮

9.2 指令说明

(1) 启动手轮

[NMC_SetHandWheel\(HAND devHandle,short axis,double ratio,double acc,double vel\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
axis	short	输入	轴号, 取值范围[0, n]
ratio	short	输入	跟随倍率, 取值范围(0, ..], 数值越大, 则同样的输入, 跟随轴运动距离越长
acc	double	输入	跟随的加速度
vel	double	输入	跟随的速度

(2) 选择手轮跟随的编码器通道

[NMC_SetHandWheelInput\(HAND devHandle,short index\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
index	short	输入	编码器通道编号, 如果是轴通道, 取值范围[0, n], 如果是扩展编码器通道, 则256 表示第一个扩展编码器通道, 257 表示第二个, 以此类推。

(3) 设置手轮跟随的倍率（无需退出手轮即可更改）

[NMC_SetHandWheelRatio\(HAND devHandle, double ratio\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ratio	double	输入	跟随倍率, 取值范围(0, ...]

(4) 退出手轮

[NMC_ClrHandWheel\(HAND devHandle\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

9.3 综合示例

```
//轴1跟随手脉运动
short gc_example(void)
{
    /****** 已省略打开控制器部分, 统一描述在第五章1 *****/
    rtn = NMC_ClrHandWheel(devHandle);          //先关闭所有手脉
    gc_rtn_error(rtn);
    rtn = NMC_SetHandWheelInput(devHandle, 256); //选择主轴为辅助编码器通道
    gc_rtn_error(rtn);
    //启动手脉, 跟随倍率为, 速度pulse/ms, 加速度pulse/ms^2
    rtn = NMC_SetHandWheel(devHandle, 0, 5, 1, 10);
    gc_rtn_error(rtn);
    return 0;
}
```

10 龙门功能

10.1 功能介绍

一个轴需要两个伺服电机同时带动起来，其中一个主动轴，一个是从动轴(从动轴可以有多个)，我司的控制器实现的龙门功能是开环龙门，可以保证主动轴和从动轴发送相同的脉冲数，在设定的允许误差范围内可以正常工作，可以减少因单轴拖动而出现的较大误差，龙门功能始终保持轴的运动同步如图 10.1.1；

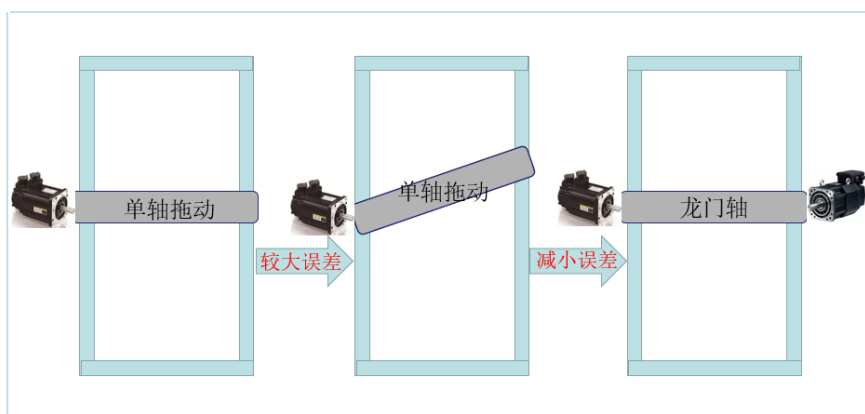


图 10.1.1 龙门功能示意图

10.2 指令说明

(1) 设置龙门主动轴

[NMC_SetGantryMaster\(HAND axisHandle, short group \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	龙门主动轴句柄
group	short	输入	龙门组号, 取值范围[0, n]

注意：龙门主轴、从轴的正方向应该一致，即发正脉冲，运动方向一致；龙门主轴、从轴的编码器反馈应该一致，即编码器反馈值方向相同，大小相等；

(2) 设置龙门从动轴

[NMC_SetGantrySlave\(HAND axisHandle , short group, long gantryErrLmt \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	龙门从动轴句柄

group	short	输入	龙门组号, 取值范围[0, n]
gantryErrLmt	long	输入	龙门保护误差, 取值范围 (0, 32767]

注意: 若有 Z 相信号回零, 注意电机安装位置:

(3)关闭龙门

[NMC_DelGantryGroup\(HAND axisHandle, short group \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	龙门主动轴句柄
group	short	输入	龙门组号, 取值范围[0, n]

10.3 综合示例

```
//龙门功能配置
short gc_example(void)
{
    short rtn;                // 指令返回值
    short devNum;             // 设备序号, 从0开始
    short axisNum;            // 轴号, 从0开始
    HAND devHandle;           // 设备句柄
    HAND axisHandle[2];       // 轴句柄
    HAND crdHandle;           // 坐标系句柄
    rtn = NMC_DevOpen(devNum, &devHandle);           //打开控制器
    gc_rtn_error(rtn);
    for (short i = 0; i < 2; i++) {
        rtn = NMC_MtOpen(devHandle, axisNum, &axisHandle[i]); //打开轴, 并获得轴句柄
        gc_rtn_error(rtn);
        rtn = NMC_MtSetSvOn(axisHandle[i]);           //打开轴使能(根据实际情况使用)
        gc_rtn_error(rtn);
    }
    rtn = NMC_SetGantryMaster(axisHandle[0], 0); //设置龙门主轴, 组号为0
    gc_rtn_error(rtn);
    rtn = NMC_SetGantrySlave(axisHandle[1], 0, 2000); //设置龙门组的从轴, 允许误差为2000脉冲
    gc_rtn_error(rtn);
    return 0;
}
```


第三章 高级功能

本章节主要描述运动控制卡高级功能的指令，指令参数的含义和综合应用示例的讲解，方便帮助用户更好的了解控制器。

1 位置比较输出

1.1 多维位置比较输出

1.1.1 功能介绍

多维位置比较输出功能可以支持4组。每组最多支持同时触发3路输出(也可以支持轮流触发3路输出)。每路输出可以指定不同的IO类型和输出方式。

相关的参数说明如下，通过[NMC_CompXDimensSetParam](#)接口配置：

参数	说明
dimens	维度，可设为 1 或者 2
axMask	位置比较轴掩码
Src	轴位置类型：0：规划位置；1：编码器类型
outCnts	输出 IO 数量，最多三组
outType	速出方式，0：脉冲，1：电平
outChnType	输出通道类型：0：GP0，1：Gate 信号
outIndex	输出通道序号，对于 GP0 范围[0, 63]，对于 Gate 只能为 0
outStLevel	初始电平(0 或者 1)
outGateTime	脉冲模式下的脉冲时间，单位毫秒(ms)
errZone	容差半径，单位 pulse

设置好参数后，通过[NMC_CompXDimensSetData](#)下载位置比较数组，

对于二维位置比较，位置数组的数据组织如下示意(n个位置比较点)：

X	Y	X	Y	...	X	Y
1	1	2	2		n	n

对于一维位置比较，位置数组的数据组织如下示意(n个位置比较点)：



1.1.2 指令说明

(1) 设置多维位置比较参数

[NMC_CompXDimensSetParam\(HAND devHandle, TCompXDimensParam *param, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
param	TCompXDimensParam *	输入	多维位置比较参数结构体 <pre> typedef struct{ short dims; //维度 short axMask; //使用的轴, 按位 short src; //轴位置类型: 0: 规划, 1: 编码器 short outCnts; // 输出数量[1, 3] short outType[CMP_OUTPUT_CHN_MAX]; //输出方式: 0:脉冲, 1:电平 short outChnType[CMP_OUTPUT_CHN_MAX]; //通道类型: 0: GPO, 1:GATE通道 short outIndex[CMP_OUTPUT_CHN_MAX]; //GPO: 0~63, GATE: 0 short outStLevel[CMP_OUTPUT_CHN_MAX]; //电平模式起始电平, 0:低电平, 1:高电平 short outGateTime[CMP_OUTPUT_CHN_MAX]; //脉冲模式下的脉冲时间:单位ms short errZone; //进入比较点容差半径范围(pulse) } TCompXDimensParam; </pre>
chn	short	输入	通道号, 范围[0, 3]

注意：通道类型的GPO为普通DO输出，GATE通道为扩展接口信号定义的GATE引脚，当选GPO时，GATE引脚也同时输出；

(2) 读取多维位置比较参数

[NMC_CompXDimensGetParam\(HAND devHandle, TCompXDimensParam *param, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
param	TCompXDimensParam *	输出	参考： TCompXDimensParam
chn	short	输入	通道号，范围[0, 3]

(3) 设置多维位置比较的输出模式

[NMC_CompXDimensSetCmpOutMode\(HAND devHandle, short outMode, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
outMode	short	输入	输出模式 0：同时输出模式 1： 轮循输出模式
chn	short	输入	通道号，范围[0, 3]

(4) 设置多维位置比较的数据点

[NMC_CompXDimensSetData\(HAND devHandle, long *pPosArray, short count, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPosArray	long *	输入	比较数组地址， 注意： 若是一维比较，则pPosArray传入一维数组地址，若是二维比较，则pPosArray应传入二维数组地址
count	short	输入	比较的数据点数， 注意： 若为二维数组比较时，每两个数据为一个点数
chn	short	输入	通道号，范围[0, 3]

(5) 多维位置比较开关

[NMC_CompXDimensOnoff\(HAND devHandle, short onOff, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
onOff	short	输入	开关 0: 关闭比较 1: 打开比较
chn	short	输入	通道号, 范围[0, 3]

(6) 查询多维位置比较状态

[NMC_CompXDimensStatus\(HAND devHandle, short *pStatus, short *pOutCount, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pStatus	short *	输出	0 未启动比较 1 比较输出中
pOutCount	short *	输出	已经输出的个数
chn	short	输入	通道号, 范围[0, 3]

1.1.3 综合示例

这里配置了一维位置比较, 把 a, b, c, d 按顺序结合在一起即可。

a. 配置参数

```
//参数配置
TCompXDimensParam cmdXDPrm;
memset(&cmdXDPrm, 0, sizeof(TCompXDimensParam));
cmdXDPrm.dimens = 1;           // 设置为一维位置比较
cmdXDPrm.axMask = 0x1;        // 第一轴为比较轴
cmdXDPrm.src = 0;             //位置源为规划位置
cmdXDPrm.outCnts = 2;         //输出两路输出
cmdXDPrm.outType[0] = 0;       // 通道的输出类型为脉冲方式
cmdXDPrm.outType[1] = 0;       // 通道的输出类型为脉冲方式
cmdXDPrm.outChnType[0] = 0;    //通道的输出类型为GPO
cmdXDPrm.outChnType[1] = 1;    //通道的输出类型为GATE
cmdXDPrm.outIndex[0] = 0;      //通道的输出通道为D00
cmdXDPrm.outIndex[1] = 0;      //通道的输出通道为Gate
cmdXDPrm.outStLevel[0] = 0;    //通道的初始电平
cmdXDPrm.outStLevel[1] = 0;    //通道的初始电平
cmdXDPrm.outGateTime[0] = 100; // 脉冲宽度为100毫秒
```

```
cmdXDPrm.outGateTime[1] = 50;          // 脉冲宽度为50毫秒
cmdXDPrm.errZone = 100;                // 容差半径为100个脉冲
rtn = NMC_CompXDimensSetParam(devHandle, &cmdXDPrm, 0); // 设置二维比较输出group0的参数
gc_rtn_error(rtn);
rtn = NMC_CompXDimensSetCmpOutMode(devHandle, 0, 0); // 设置为同时输出模式
gc_rtn_error(rtn);
```

b. 下载位置比较数组

```
//下载位置比较数组
int xPos[1024];
for (int i = 0; i < 10; i++)          // 设置位置比较数组10个点
{
    xPos[i] = (i + 1) * 10000;        // X轴位置, X轴运动到此位置, 对应的D0输出信号
}
rtn = NMC_CompXDimensSetData(devHandle, (long*)xPos, 10, 0);
gc_rtn_error(rtn);
rtn = NMC_CompXDimensOnoff(devHandle, 1, 0); // 启动位置比较输出
gc_rtn_error(rtn);
```

c. 开始比较输出

启动输出后, 第一轴运动到 10000, 20000, 30000...90000, 100000 等 10 个位置时, D00 和 Gate 都会输出一个脉冲信号;

```
//开始比较输出
rtn = NMC_CompXDimensOnoff(devHandle, 1, 0); // 启动位置比较输出
gc_rtn_error(rtn);
```

d. 读取位置比较输出状态

```
//读取位置比较输出状态
short cmpSts, cmdOutput;
rtn = NMC_CompXDimensStatus(devHandle, &cmpSts, &cmdOutput, 0);
gc_rtn_error(rtn);
```

1.2 一维高速位置比较

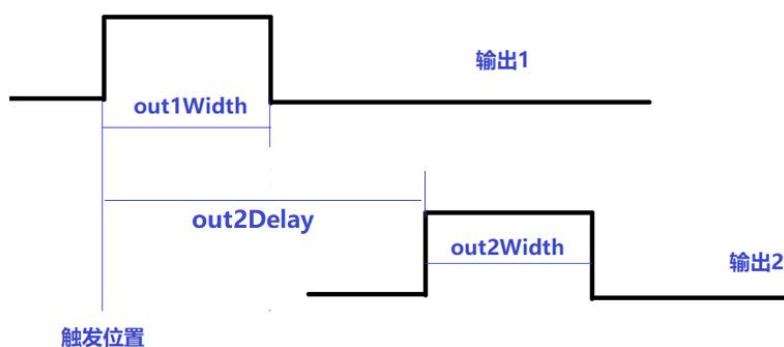
1.2.1 功能介绍

一维高速位置比较输出功能可以支持4路, 通过[NMC_CompHs1DimensSetParam](#)接口配置, 相关的参数说明如下:

参数	说明
DirNo	编码器源(0~7)对应 0~7 轴的编码器
Out1StLevel	输出 1 的起始电平
Out2StLevel	输出 2 的起始电平
Out1Width	输出 1 的脉冲宽度, 0~65535, 单位微秒
Out2Width	输出 2 的脉冲宽度, 0~65535, 单位微秒
Out2Delay	输出 2 相对输出 1 的延时, 0~65535, 单位微秒

设置好参数后, 通过 [NMC_CompHs1DimensSetData](#) 下载位置比较数组, 然后通过 [NMC_CompHs1DimensOnOff](#) 启动输出, 输出过程可以通过 [NMC_CompHs1DimensStatus](#) 查询输出状态。

高速高精度的比较输出, 通常用于飞拍等场合, Out1, out2 可以分别用于控制光源开灯及相机的启动拍照, 输出时序如下图所示:



1.2.2 指令说明

(1) 设置高速位置比较的参数

[NMC_CompHs1DimensSetParam](#)(HAND devHandle, TCompHs1DimensParam *param, short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
param	TCompHs1DimensParam *	输入	一维高速位置比较参数结构 <pre>typedef struct { short dirNo; // 编码器源 0~7, 对应轴 0~7 的编码器 short out1StLevel; // out1 起始电平 (0 或 1)</pre>

			<pre> short out2StLevel; //out2起始电平(0或1) short reserved; //保留 long out1Width; //out1脉冲宽度, 0~65535, 单位us long out2Width; //out2脉冲宽度, 0~65535, 单位us long out2Delay; //out2延时, 0~65535, 单位us } TCompHs1DimensParam; </pre>
chn	short	输入	通道号, 范围[0, 3]

(2) 读取高速位置比较参数

[NMC_CompHs1DimensGetParam\(HAND devHandle, TCompHs1DimensParam *param, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
param	TCompHs1DimensParam *	输出	参考: TCompHs1DimensParam
chn	short	输入	通道号, 范围[0, 3]

(3) 设置高速位置比较的数据点

[NMC_CompHs1DimensSetData\(HAND devHandle, long *pArrayPos, short count, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pArrayPos	long *	输入	比较数组地址
count	short	输入	比较的数据点数
chn	short	输入	通道号, 范围[0, 3]

(4) 高速位置比较开关

[NMC_CompHs1DimensOnOff\(HAND devHandle, short onOff, short chn\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
onOff	short	输入	开关 0: 关闭比较 1: 打开比较 2: 手动输出 (必须在空闲状态下, 也需要提前设置参数)
chn	short	输入	通道号, 范围[0, 3]

(5) 查询高速位置比较状态

[NMC CompHs1DimensStatus\(HAND devHandle, short * pBusy, short *pOutCount, short *pWaitCnts, short *pFreeCnts, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pBusy	short *	输出	0 未启动比较 1 比较输出中
pOutCount	short *	输出	已经输出的个数
pWaitCnts	short *	输出	等待比较的个数
pFreeCnts	short *	输出	缓冲区空闲的个数
chn	short	输入	通道号, 范围[0, 3]

1.2.3 综合示例

这里配置了一维高速位置比较, 把 a, b, c, d 按顺序结合在一起即可。

a. 配置参数

```
// 配置参数
TCompHs1DimensParam cmpPrm;
memset(&cmpPrm, 0, sizeof(TCompHs1DimensParam));
cmpPrm.dirNo = 0; // 第一轴
cmpPrm.out1StLevel = 1; // out1起始电平为高电平
cmpPrm.out2StLevel = 1; // out2起始电平为高电平
cmpPrm.out1Width = 100; // out1脉冲宽度为us
cmpPrm.out2Width = 100; // out2脉冲宽度为us
cmpPrm.out2Delay = 100; // out1与out2之间的延时为us
rtn = NMC_CompHs1DimensSetParam(devHandle, &cmpPrm, 0);
gc_rtn_error(rtn);
```

b. 下载位置比较数组


```
// 下载位置比较数组
int hsldPos[8];
for (int i = 0; i < 8; i++)
{
    hsldPos[i] = (i + 1) * 10000;    // X轴位置
}
rtn = NMC_CompHsldDimensSetData(devHandle, (long*)hsldPos, 8, 0);
gc_rtn_error(rtn);
```

c. 开始比较输出

启动输出后，第一轴运动到 10000，20000，30000...等 8 个位置时，对应的输出通道会有脉冲输出。

```
// 启动位置比较输出
rtn = NMC_CompHsldDimensOnOff(devHandle, 1, 0);
gc_rtn_error(rtn);
```

d. 读取位置比较输出状态

```
//读取比较状态
short cmpIsBusy, cmpOutCount, cmpWaitCount, cmpFreeSpace;
rtn = NMC_CompHsldDimensStatus(devHandle, &cmpIsBusy, &cmpOutCount, &cmpWaitCount,
&cmpFreeSpace, 0);
gc_rtn_error(rtn);
```

手动输出

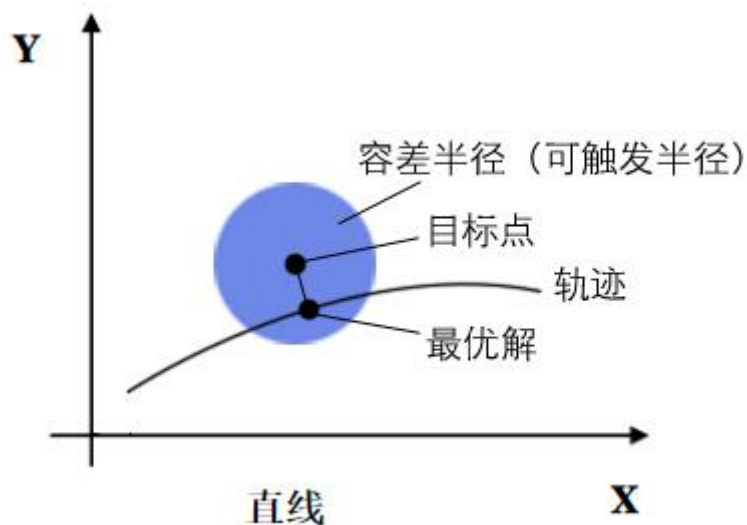
```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    TCompHsldDimensParam cmpPrm;
    memset(&cmpPrm, 0, sizeof(TCompHsldDimensParam));
    cmpPrm.dirNo = 0;    // 第一轴
    cmpPrm.out1StLevel = 1;    // out1起始电平为高电平
    cmpPrm.out2StLevel = 1;    // out2起始电平为高电平
    cmpPrm.out1Width = 10000;    // out1脉冲宽度为us
    cmpPrm.out2Width = 10000;    // out2脉冲宽度为us
    cmpPrm.out2Delay = 100;    // out1与out2之间的延时为us
    rtn = NMC_CompHsldDimensSetParam(devHandle, &cmpPrm, 0);
    gc_rtn_error(rtn);
    rtn = NMC_CompHsldDimensOnOff(devHandle, 2, 0); // 启动位置比较输出
    gc_rtn_error(rtn);
}
```

```
return 0; }
```

1.3 二维高速位置比较

1.3.1 功能介绍

当系统轨迹运行到图示蓝色区域内时，控制器会通过寻找最优解算法得出与目标点最近的点，然后比较输出口立即输出信号。



1.3.2 指令说明

(1) 设置高速二维位置比较的参数

[NMC_Comp2DimensSetParam\(HAND devHandle, short group, TComp2DimensParam *param, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
param	TComp2DimensParam *	输入	<pre>#define CMP_OUTPUT_CHN_MAX (3) //二维位置点比较结构体 typedef struct{ // 比较输出的通道:-1:不处理, 范围 0~1 short outputchn[CMP_OUTPUT_CHN_MAX];</pre>

			<pre> // 输出方式 0: 脉冲 1: 电平 short outputType[COMP_OUTPUT_CHN_MAX]; // 通道类型: 0 GPO, 1 GATE 通道 short chnType[COMP_OUTPUT_CHN_MAX]; // 方向 1 的位置源轴号(0~11) short dir1No; // 方向 2 的位置源轴号(0~11) short dir2No; // 轴位置类型 : 0 规划 1: 编码器 short posSrc; // 电平模式下的起始电平(0 或 1) short stLevel[COMP_OUTPUT_CHN_MAX]; // 脉冲方式脉冲时间:单位 ms short gateTime[COMP_OUTPUT_CHN_MAX]; // 进入比较点容差半径范围(pulse) short errZone; } TComp2DimensParam; </pre>
chn	short	输入	保留, 设为 0

(2) 读取高速二维位置比较的参数

[NMC_Comp2DimensGetParam\(HAND devHandle, short group, TComp2DimensParam *param, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
param	TComp2DimensParam *	输出	参数结构体
chn	short	输入	保留, 设为 0

(3) 设置高速二维比较数据

[NMC_Comp2DimensSetData\(HAND devHandle, short group, long *pArrayPos, short count, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
pArrayPos	long*	输入	比较数组地址
count	short	输入	比较数量
chn	short	输入	保留, 设为 0

(4) 高速二维位置比较使能

[NMC_Comp2DimensOnoff\(HAND devHandle, short group, short onOff, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
onOff	short	输入	0 停止, 1 输出 2 手动
chn	short	输入	保留, 设为 0
chn	short	输入	保留, 设为 0

(5) 读取高速二维位置比较输出状态

[NMC_Comp2DimensStatus\(HAND devHandle, short group, short *pStatus, short *pOutCount, short chn\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
pStatus	short*	输出	0 未启动比较 1 比较输出中
pOutCount	short*	输出	输出的个数
chn	short	输入	保留, 设为 0

(6) 读取高速二维位置比较输出状态 Ex

NMC_Comp2DimensStatusEx (HAND devHandle, short group, TComp2DimensSts *pStatus, short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
pStatus	TComp2DimensSts*	输出	<pre>typedef struct{ short sts; // 运行状态, 空闲 1 忙 short reserved1; // 保留 long freeSpace; // 控制器剩余空间 long usedSpace; // 剩余位置比较点 long outCount; // 已经输出的个数 long reserved2[4]; // 保留 } TComp2DimensSts;</pre>
chn	short	输入	保留, 设为 0

(7) 设置高速二维位置比较的参数

NMC_Comp2DimensSetParamEx (HAND devHandle, short group, TComp2DimensParamEx *param, short chn);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号, 0 或者 1
param	TComp2DimensParamEx *	输入	<pre>typedef struct{ short outputChn; // 比较 输出的通道号, 取值[0, n] short outputType; // 输出 方式 0: 脉冲 1: 电平 short chnType; // 通道类 型: 0 GPO, 1 GATE 通道 short dir1No; // 方向 1</pre>

			的位置源轴号(0~11) <code>short dir2No; // 方向 2</code> 的位置源轴号(0~11) <code>short posSrc; // 轴位置</code> 类型 : 0 规划 1: 编码器 <code>short stLevel; // 电平模</code> 式下的起始电平(0 或 1) <code>short errZone; // 进入</code> 比较点容差半径范围(pulse) <code>short directOutZone; //</code> 近距离直接触发范围 <code>short vibrateRange; //</code> 抖动滤波范围 <code>long gateTime; // 脉冲</code> 方式脉冲时间, 单位 us <code>long minIntervalTime; //</code> 最小触发时间间隔, 单位 us <code>long reserved[8]; // 保</code> 留, 默认值 0(不使用) <code>} TComp2DimensParamEx;</code>
chn	short	输入	保留, 设为 0

(8) 读取高速二维位置比较的参数

[NMC_Comp2DimensGetParamEx\(HAND devHandle, short group, TComp2DimensParamEx
*param, short chn\);](#)

参考: [NMC_Comp2DimensSetParamEx](#)

1.3.3 综合示例

```
//二维高速位置比较
short gc_example(void)
```

```
{
    /******* 已省略打开控制器部分，统一描述在第五章1 *****/
    TComp2DimensParam compara_2d;
    memset(&compara_2d, 0, sizeof(compara_2d));
    rtn = NMC_Comp2DimensGetParam(devHandle, 0, &compara_2d, 0); //读取第0组二维位置比较参数
    gc_rtn_error(rtn);
    compara_2d.dir1No = 0; //轴1，-1不使用该轴
    compara_2d.dir2No = 1; //轴2，-1不使用该轴
    compara_2d.chnType[0] = 1; //通道0的硬件接口为控制器扩展口上的gate(hsio)引脚
    compara_2d.chnType[1] = 0; //通道1的硬件接口为gpo
    compara_2d.chnType[2] = 0; //通道2的硬件接口为gpo
    compara_2d.outputType[0] = 0; //通道0输出脉冲信号
    compara_2d.outputType[1] = 1; //通道1输出电平信号
    compara_2d.outputType[2] = 1; //通道2输出电平信号
    compara_2d.outputchn[0] = 0; //只在通道输出
    compara_2d.outputchn[1] = -1; //通道不处理
    compara_2d.outputchn[2] = -1; //通道不处理
    compara_2d.errZone = 10; //容差半径脉冲
    compara_2d.gateTime[0] = 10; //通道0脉冲输出时间ms
    compara_2d.gateTime[1] = 10; //通道1(输出为GPO时该值无效)
    compara_2d.gateTime[2] = 10; //通道2(输出为GPO时该值无效)
    compara_2d.posSrc = 1; //比较源为编码器
    compara_2d.stLevel[0] = 0; //通道初始电平为低电平
    rtn = NMC_Comp2DimensSetParam(devHandle, 0, &compara_2d, 0); //设置第0组二维位置比较参数
    gc_rtn_error(rtn);
    int count = 100;
    long* compos = new long[2 * count];
    for (int i = 0; i < count; i++) //设置第0组比较的数据
    {
        *(compos + i * 2) = 100 * i + 100; //X坐标
        *(compos + i * 2 + 1) = 100 * i + 100; //Y坐标
    }
    rtn = NMC_Comp2DimensSetData(devHandle, 0, compos, 0, 0); //先清空
    gc_rtn_error(rtn);
    rtn = NMC_Comp2DimensSetData(devHandle, 0, compos, count, 0); //再设置
    gc_rtn_error(rtn);
    rtn = NMC_Comp2DimensOnoff(devHandle, 0, 1, 0); //开始比较输出
    gc_rtn_error(rtn);
    //开启后可以通过NMC_Comp2DimensStatus读取比较的状态
    return 0;
}
```

//二维高速位置比较，手动输出一个信号

```
short gc_example(void)
```

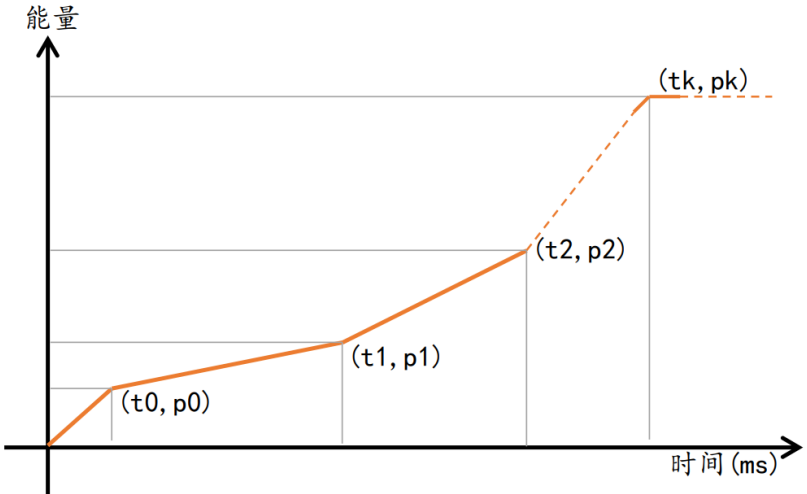
```
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    TComp2DimensParam compara_2d;
    memset(&compara_2d, 0, sizeof(compara_2d));
    rtn = NMC_Comp2DimensGetParam(devHandle, 0, &compara_2d, 0); //读取第0组二维位置比较参数
    gc_rtn_error(rtn);
    compara_2d.dir1No = 0; //轴1，-1不使用该轴
    compara_2d.dir2No = 1; //轴2，-1不使用该轴
    compara_2d.chnType[0] = 1; //通道0的硬件接口为控制器扩展口上的gate(hsio)引脚
    compara_2d.chnType[1] = 0; //通道1的硬件接口为gpo
    compara_2d.chnType[2] = 0; //通道2的硬件接口为gpo
    compara_2d.outputType[0] = 0; //通道0输出脉冲信号
    compara_2d.outputType[1] = 1; //通道1输出电平信号
    compara_2d.outputType[2] = 1; //通道2输出电平信号
    compara_2d.outputchn[0] = 0; //只在通道输出
    compara_2d.outputchn[1] = -1; //通道不处理
    compara_2d.outputchn[2] = -1; //通道不处理
    compara_2d.errZone = 10; //容差半径脉冲
    compara_2d.gateTime[0] = 10; //通道0脉冲输出时间ms
    compara_2d.gateTime[1] = 10; //通道1(输出为GPO时该值无效)
    compara_2d.gateTime[2] = 10; //通道2(输出为GPO时该值无效)
    compara_2d.posSrc = 1; //比较源为编码器
    compara_2d.stLevel[0] = 0; //通道初始电平为低电平
    rtn = NMC_Comp2DimensSetParam(devHandle, 0, &compara_2d, 0); //设置第0组二维位置比较参数
    gc_rtn_error(rtn);
    rtn = NMC_Comp2DimensOnoff(devHandle, 0, 2, 0); //开启二维比较
    gc_rtn_error(rtn);
    //开启后可以通过NMC_Comp2DimensStatus()读取比较的状态
    return 0;
}
```

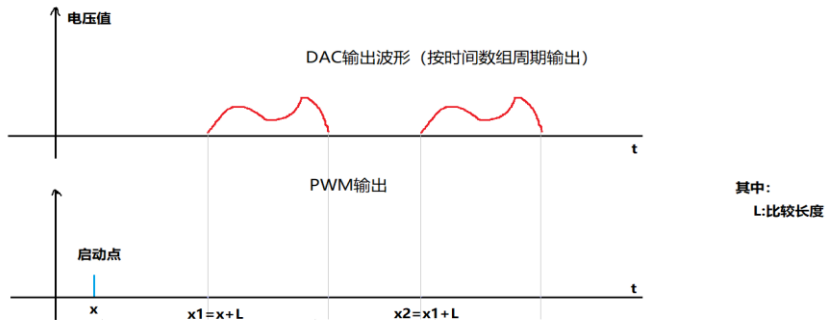


2 激光功能(选配)

2.1 激光模式设置

2.1.1 功能介绍

激光控制包括三种控制模式(通过 [NMC LaserSetModes](#) 配置)如下:

模式	说明
基本控制模式	用户通过指令(立即或者缓冲区)直接控制激光的开关和能量 也可以设定为自动速度能量跟随
波形控制模式	<p>用户下载‘能量—时间’数组到控制器,控制器在激光输出时,自动按照数组设定的关系控制激光,如下示意(激光开):</p>  <p>时间单位最小为 100us。</p> <p>分为两种模式:</p> <p>单点激光能量时间数组控制</p> <p>用户在调用 NMC LaserTimeArrayExe/ NMC CrdBufLaserTimeArrayExe 后,控制器按照设定的能量-时间数组控制激光能量,直到最后一个点,然后关闭激光输出。</p> <p>位置比较配合焊接,用户设定位置比较的轴,比较长度等,并下载能量-时间数组,并调用 NMC LaserTimeArrayExe/ NMC CrdBufLaserTimeArrayExe 启动位置比较。则控制器自动计算位置差值,当满足条件时,输出一次能量波形,示意图如下。</p>

	 <p>电压值</p> <p>DAC输出波形 (按时间数组周期输出)</p> <p>PWM输出</p> <p>其中: L:比较长度</p> <p>启动点</p> <p>x</p> <p>$x1=x+L$</p> <p>$x2=x1+L$</p> <p>t</p> <p>t</p> <p>波形控制模块下，可以通过 NMC_LaserOnOff/NMC_CrdBufLaserOnOff 强制中断激光能量控制输出。</p>
位置比较控制模式	<p>位置比较控制模式 (SHIO)，用于对门控信号进行控制，通常应用是自动通过位置变化量 (规划或者编码器) 对门控信号进行打开或关闭的控制方式，如下图所示：</p> <p>dt:gate信号打开时间 dL:位移间距</p>  <p>gate信号</p> <p>单轴或多轴合成位置</p>

2.1.2 指令说明

(1) 设置激光的控制模式

[NMC_LaserSetMode\(HAND devHandle, short mode, short ch \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
mode	short	输入	激光控制模式 LASER_DISABLE_MODE (0) // 禁用激光功能 BASIC_OUTPUT_MODE (1) // 基本控制模式 TIME_ARRAY_OUTPUT_MODE (2) // 波形控制模式 SHIO_OUTPUT_MODE (3) // 位置比较控制模式
ch	short	输入	通道号，范围[0,1]

2.1.3 综合示例

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    rtn = NMC_LaserSetMode(devHandle, BASIC_OUTPUT_MODE, 0); //设置激光通道为立即模式
    gc_rtn_error(rtn);
    return 0;
}
```

2.2 基本控制模式

2.2.1 功能介绍

用户通过设置频率和占空比调节激光器能量，即通过指令(立即或者缓冲区)直接控制激光的开关和能量。

基本控制模式下，每个激光通道可以配置为不同的物理信号输出类型(通过 [NMC_LaserSetOutputTypeEx](#) 配置)如下：

模式	说明
LASER_NONE	关闭激光输出
LASER_DA	模拟量输出
LASER_PWM_DUTY	占空比输出 此模式下， NMC_LaserSetOutputTypeEx 函数的参数optionVal用来设置PWM的频率
LASER_PWM_FRQ	频率输出，占空比固定 此模式下， NMC_LaserSetOutputTypeEx 函数的参数 optionVal 用来设置 PWM 的占空比
LASER_PWM_FRQ_EXT	频率输出，脉宽固定 此模式下， NMC_LaserSetOutputTypeEx 函数的参数 optionVal 用来设置 PWM 的脉宽时间

2.2.2 指令说明

(1) 设置激光物理信号输出类型

[NMC_LaserSetOutputTypeEx](#)(HAND devHandle, short outputType, short index, double optionVal, short ch);

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
outputType	short	输入	激光物理信号类型 LASER_NONE (0) // 关闭激光输出模式 LASER_DA (1) // DA输出 LASER_PWM_DUTY (2) // 占空比输出 LASER_PWM_FRQ (3) // 频率输出 LASER_PWM_FRQ_EXT (4) // 频率输出, 脉宽固定
index	short	输入	输出通道序号, 取值范围[0, n]
optionVal	double	输入	LASER_DA: 无意义 LASER_PWM_DUTY: 该值为 PWM 的频率, 单位 HZ LASER_PWM_FRQ: 该值作为占空比值, (0~100) LASER_PWM_FRQ_EXT: 该值为脉宽, 单位为微秒, 取值范围(0, ~)
ch	short	输入	通道号, 范围[0, 1]

(2) 设置激光参数(最大最小能量待机能量)

[NMC LaserSetParam\(HAND devHandle, long onDelay, long offDelay, long minValue, long maxValue, long standbyPower, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
onDelay	long	输入	开光延时, 单位 us, 取值范围[0, 65535]
offDelay	long	输入	关光延时, 单位 us, 取值范围[0, 65535]
minValue	long	输入	最小输出值, DA 输出时, 范围[0, 32767], 占空比输出时, 范围[0, 100], 频率输出时, 范围[0, 2000000]
maxValue	long	输入	最大输出值, DA 输出时, 范围[0, 32767], 占空比输出时, 范围[0, 100], 频率输出时, 范围[0, 2000000]

standbyPower	long	输入	待机能量，DA 输出时，范围[0, 32767], 占空比输出时，范围 [0,100], 频率输出时，范围 [0, 2000000]，为表示取消待机能量输出功能
ch	short	输入	通道号，范围[0, 1]

(3) 读取激光参数(最大最小能量待机能量)

[NMC LaserGetParam\(HAND devHandle, long *pOnDelay, long *pOffDelay, long *pMinValue, long *pMaxValue, long *pStandbyPower, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pOnDelay	long*	输出	开光延时, 单位 us, 取值范围[0, 65535]
pOffDelay	long*	输出	关光延时, 单位 us, 取值范围[0, 65535]
pMinValue	long*	输出	最小输出值，DA 输出时，范围[0, 32767], 占空比输出时，范围 [0, 100], 频率输出时，范围 [0, 2000000]
pMaxValue	long*	输出	最大输出值，DA 输出时，范围[0, 32767], 占空比输出时，范围 [0, 100], 频率输出时，范围 [0, 2000000]
pStandbyPower	long*	输出	待机能量，DA 输出时，范围[0, 32767], 占空比输出时，范围 [0, 100], 频率输出时，范围 [0, 2000000]，0 为表示取消待机能量输出功能
ch	short	输入	通道号，范围[0, 1]

(4) 设置立即输出激光能量

[NMC LaserSetPowerEx\(HAND devHandle, double outVal, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
outVal	double	输入	激光能量，DA 输出时，范围[0, 32767], 占空比输出时，范围 [0, 100], 频率输出时，范围 [0, 2000000]

			[0, 2000000]
ch	short	输入	通道号, 范围[0, 1]

(5) 设置激光立即输出开关

[NMC_LaserOnOff\(HAND devHandle, short onOff, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
onOff	short	输入	开关, 0: 关光, 1: 开光
ch	short	输入	通道号, 范围[0, 1]

(6) 读取当前激光能量

[NMC_LaserGetPower\(HAND devHandle, double *pVal, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pVal	double*	输出	当前激光能量
ch	short	输入	通道号, 范围[0, 1]

(7) 读取当前激光开关状态

[NMC_LaserGetOnOff\(HAND devHandle, short *pOnOffState, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pOnOffState	short*	输出	激光开关状态, 1: 激光处于打开状态, 0: 激光处于关闭状态
ch	short	输入	通道号, 范围[0, 1]

(8) 缓冲区设置激光参数(最大最小能量待机能量)

[NMC_CrdBufLaserSetParam\(HAND crdHandle, long segment, long onDelay, long offDelay, long minValue, long maxValue, long standbyPower, short ch\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
onDelay	long	输入	开光延时, 单位 us, 取值范围[0, 65535]
offDelay	long	输入	关光延时, 单位 us, 取值范围[0, 65535]
minValue	long	输入	最小输出值, DA 输出时, 范围[0, 32767], 占空比输出时, 范围 [0, 100], 频率输出时, 范围 [0, 2000000]
maxValue	long	输入	最大输出值, DA 输出时, 范围[0, 32767], 占空比输出时, 范围 [0, 100], 频率输出时, 范围 [0, 2000000]
standbyPower	long	输入	待机能量, DA 输出时, 范围[0, 32767], 占空比输出时, 范围 [0, 100], 频率输出时, 范围 [0, 2000000], 0 为表示取消待机能量输出功能
ch	short	输入	通道号, 范围[0, 1]

(9) 缓冲区设置输出激光能量

[NMC_CrdBufLaserPower \(HAND crdHandle, long segNo, long power, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
power	long	输入	激光能量, DA 输出时, 范围[0, 32767], 占空比输出时, 范围 [0, 100], 频率输出时, 范围 [0, 2000000]
ch	short	输入	通道号, 范围[0, 1]

(10) 缓冲区设置激光输出开关

[NMC_CrdBufLaserOnOff \(HAND crdHandle, long segNo, short onOff, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

segNo	long	输入	段号(用户自定义)
onOff	short	输入	开关, 0: 关光, 1: 开光
ch	short	输入	通道号, 范围[0, 1]

(11) 设置激光能量跟随滤波及输出方式

[NMC LaserSetFollowParam\(HAND devHandle, short powerFilter, short followAdvMode, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
powerFilter	long	输入	能量输出滤波系数, 取值范围[0, 32], 0: 不开启(默认);
followAdvMode	short	输入	能量输出计算方法, 0 最小值截取模式 1 最小值起点模式
ch	short	输入	通道号, 范围[0, 1]

(12) 设置缓冲区激光能量跟随

[NMC CrdBufLaserSetFollow\(HAND crdHandle, long segNo, double overRide, short followType, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
overRide	double	输入	跟随倍率, 为 0 表示取消激光能量跟随
followType	short	输入	跟随类型 0: 跟随规划速度; 1: 跟随实际速度
ch	short	输入	通道号, 范围[0, 1]

最小值截取法: $X(\text{激光能量}) = \text{跟随倍率} * V_{el}(\text{合成速度})$;



最小值起点法: $X(\text{激光能量}) = \text{跟随倍率} * V_{el}(\text{合成速度}) + \text{Min}$;



(13) 设置激光能量补偿表

[NMC_SetLaserPowerCmpTable\(HAND devHandle, short tableNo, long *pXCmpPos, long *pYCmpPos, short xCount, short yCount, unsigned long powerMin, unsigned long powerMax, unsigned long *pLaserCmpPower, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
tableNo	short	输入	补偿表号：支持 20 张表
pXCmpPos	long *	输入	X 方向轴的比较位置数组地址, 长度为 xCount
pYCmpPos	long *	输入	Y 方向轴的比较位置数组地址, 长度为 yCount
xCount	short	输入	表 X 方向的长度, 取值范围[2, 10]
yCount	short	输入	表 Y 方向的长度, 取值范围[2, 10]
powerMin	unsigned long	输入	大于该最小能量才补偿
powerMax	unsigned long	输入	小于该最大能量才补偿
pLaserCmpPower	Unsigned long *	输入	补偿表的值, 该参数为二维数组的首地址
ch	short	输入	通道号, 范围[0, 1]

(14) 启动激光能量补偿

[NMC_StartLaserPowerComp\(HAND devHandle, short *pAxisNo, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pAxisNo	short*	输入	pAxisNo[0]表 X 方向的位置比较轴号 pAxisNo[1]表 Y 方向的位置比较轴号
ch	short	输入	通道号, 范围[0, 1]

(15) 停止激光能量补偿

[NMC_StopLaserPowerComp\(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号，范围[0, 1]

2.2.3 综合示例

一般控制流程：

a. 配置激光控制模式及信号输出模式

```
rtn = NMC_LaserSetMode(devHandle, BASIC_OUTPUT_MODE, 0); //设置激光通道为立即模式
gc_rtn_error(rtn);
// 激光通道配置为占空比，使用PWM输出通道，频率为1000HZ
rtn = NMC_LaserSetOutputType(devHandle, LASER_PWM_DUTY, 0, 1000, 0);
gc_rtn_error(rtn);
```

b. 直接设置激光能量/开关激光

```
rtn = NMC_LaserSetPowerEx(devHandle, 10, 0); // 直接设置激光能量
gc_rtn_error(rtn);
rtn = NMC_LaserOnOff(devHandle, 1, 0); // 开激光
gc_rtn_error(rtn);
```

c. 插补缓冲区中对激光进行控制

```
/****** 已省略打开控制器部分，统一描述在第五章1 *****/
/****** 坐标系建立过程略 *****/
rtn = NMC_LaserSetMode(devHandle, BASIC_OUTPUT_MODE, 0); //设置激光通道为立即模式
gc_rtn_error(rtn);
// 激光通道配置为占空比，使用PWM输出通道，频率为1000HZ
rtn = NMC_LaserSetOutputTypeEx(devHandle, LASER_PWM_DUTY, 0, 1000, 0);
gc_rtn_error(rtn);
rtn = NMC_CrdBufClr(devHandle); // 清除缓冲区，压入缓冲区指令
gc_rtn_error(rtn);
rtn = NMC_CrdBufLaserPower(crdHandle, 100, 10, 0); //设置激光通道的能量为10%
gc_rtn_error(rtn);
rtn = NMC_CrdBufLaserOnOff(crdHandle, 101, 1, 0); // 开激光
gc_rtn_error(rtn);
// 启动直线插补
long tgtPos[2];
tgtPos[0] = 10000;
```

```
    tgtPos[1] = 0;
    rtn = NMC_CrdLineXYZEx(crdHandle, 154, 0x3, tgtPos, 0, 10, 1, 0);
    gc_rtn_error(rtn);
    rtn = NMC_CrdBufLaserOnOff(crdHandle, 101, 0, 0);    //插补完成后关闭激光输出
    gc_rtn_error(rtn);
    rtn = NMC_CrdEndMtn(crdHandle);                      //指令压入结束
    gc_rtn_error(rtn);
    rtn = NMC_CrdStartMtn(crdHandle);                    //启动缓冲区运动
    gc_rtn_error(rtn);
```

d. 速度能量跟随输出

```
/****** 已省略打开控制器部分，统一描述在第五章1 *****/
/****** 坐标系建立过程略 *****/
// 设置激光速度能量跟随滤波为ms，输出计算方式为最小值截取模式
rtn = NMC_LaserSetFollowParam(devHandle, 16, 0, 0);
gc_rtn_error(rtn);
// 打开激光速度能量跟随，跟随倍率为5，跟随规划位置
rtn = NMC_CrdBufLaserSetFollow(crdHandle, 1, 5, 0, 0);
gc_rtn_error(rtn);
rtn = NMC_CrdBufLaserOnOff(crdHandle, 101, 0, 0);    // 关闭激光
gc_rtn_error(rtn);
double vel = 10;
double acc = 1;
long tgPos[2];
// 运动到起始点
tgPos[0] = 20000;
tgPos[1] = 0;
rtn = NMC_CrdLineXYZEx(crdHandle, 154, 0x3, tgPos, 0, vel, acc, 0);
gc_rtn_error(rtn);
rtn = NMC_CrdBufLaserOnOff(crdHandle, 101, 1, 0);    // 开光
gc_rtn_error(rtn);
// 加工(插补过程中，激光自动根据速度变化)
tgPos[0] = 100000;
tgPos[1] = 0;
rtn = NMC_CrdLineXYZEx(crdHandle, 154, 0x3, tgPos, 0, vel, 0.01, 0);
gc_rtn_error(rtn);
rtn = NMC_CrdBufLaserOnOff(crdHandle, 101, 0, 0);    // 加工完成，关闭激光
gc_rtn_error(rtn);
rtn = NMC_CrdEndMtn(crdHandle);                      //指令压入结束
gc_rtn_error(rtn);
rtn = NMC_CrdStartMtn(crdHandle);                    //启动缓冲区运动
gc_rtn_error(rtn);
```

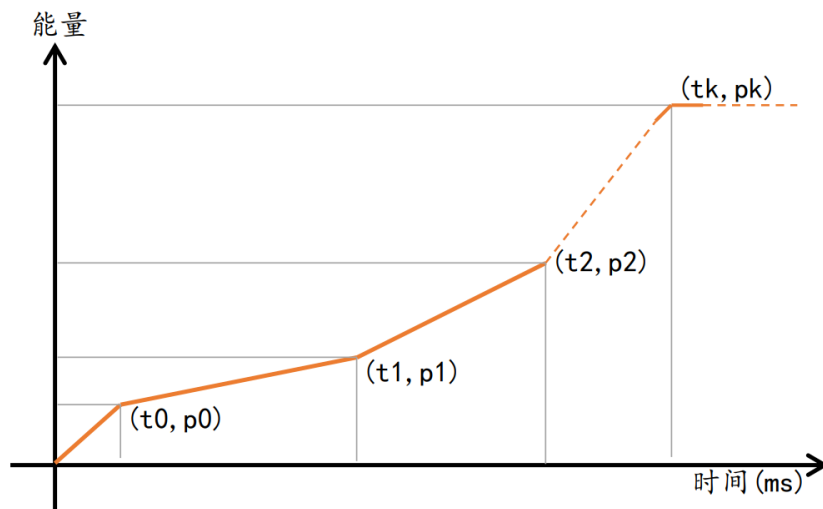
e. 使用激光能量补偿表

```
//补偿指令在压入插补指令之前调用
short tableNo = 0;
//长度的数组，定义个区间，总计*10的区间
long xCmpPos[11] = {0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000 };
long yCmpPos[11] = {0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000 };
unsigned long pLaserCmpPower[10][10]; //总计*10的补偿能量值
for (tableNo = 0; tableNo < 1; tableNo++)
{
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            pLaserCmpPower[i][j] = tableNo * 100 + i * 10;
        }
    }
    // 设置通道激光能量补偿表
    rtn = NMC_SetLaserPowerCmpTable(devHandle, tableNo, xCmpPos, yCmpPos, 10, 10,
    tableNo * 1000, (tableNo + 1) * 1000, &pLaserCmpPower[0][0], 0);
    gc_rtn_error(rtn);
}
short pAxisNo[2] = { 0, 1 }; //X方向为轴，Y方向为轴
rtn = NMC_StartLaserPowerComp(devHandle, pAxisNo, 0); // 启用激光能量补偿表
gc_rtn_error(rtn);
```

2.3 波形控制模式

2.3.1 功能介绍

模式	说明
波形控制模式	用户下载‘能量—时间’数组到控制器，控制器在激光输出时，自动按照数组设定的关系控制激光，如下示意(激光开)：



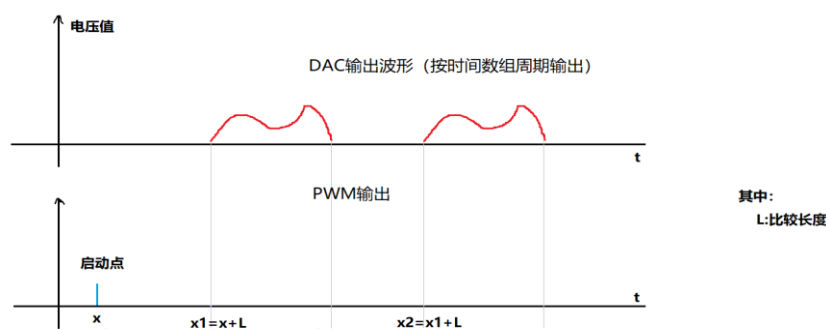
时间单位最小为 100us。

分为两种模式：

单点激光能量时间数组控制

用户在调用 [NMC_LaserTimeArrayExe](#)/[NMC_CrdBufLaserTimeArrayExe](#) 后，控制器按照设定的能量-时间数组控制激光能量，直到最后一个点，然后关闭激光输出。

位置比较配合焊接：用户设定位置比较的轴，比较长度等，并下载能量-时间数组，并调用 [NMC_LaserTimeArrayExe](#)/[NMC_CrdBufLaserTimeArrayExe](#) 启动位置比较。则控制器自动计算位置差值，当满足条件时，输出一能量波形，示意图如下。



波形控制模块下，可以通过 [NMC_LaserOnOff](#)/[NMC_CrdBufLaserOnOff](#) 强制中断激光能量控制输出。

2.3.2 指令说明

(1) 设置时间数组输出参数

[NMC_LaserSetTimeArrayPara\(HAND devHandle, TTimeArrayPara *pPara, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPara	TTimeArrayPara *	输入	<pre>typedef struct{ short pwmEnable; // 是否需要输出PWM, 0: 立即输出一个波形, 1: 根据位移周期输出 short outputType; // 开关信号输出类型: 0: gate, 1: GP0 short outputCh; // 开关信号输出通道 short stLevelRevs; // SHIO输出电平取反, 默认为0, 不取反 long pwmPeriod; // 保留, PWM周期, 单位us, 不能小于时间数组的总周期 long pwmWidth; // 保留, PWM脉宽, 单位us, 此参数保留, 脉宽等于时间数组的总周期 long gateDistance; // 固定模式下的位置间隔单位: pulse 默认0, 模式2~4 下会进行有效性检查 long minFrqFrq; // 保留, SHIO输出最低频率, 单位HZ short posSrc; // 比较模式, 外部编码器还是内部规划值 0: 外部编码器(推荐), 1: 内部规划值。 short axisMask; // 轴号, 按bit 位对应。(一般两个轴)。 short minFrqEn; // 保留, 是否启用SHIO输出最低频率, 默认0, 不启用 short reserved2; // 保留 }TTimeArrayPara;</pre>
ch	short	输入	通道号, 范围[0, 1]

(2) 设置激光能量时间数组

[NMC_LaserSetTimeArrayPower\(HAND devHandle, short group, TLaserPower *pLaserPower\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号，最大支持 12 组，取值范围[0, 11]
pLaserPower	TLaserPower*	输入	能量时间数组地址 <pre>typedef struct{ unsigned short time[LASET_POINT]; // 每个 点之间的间隔时间, 单位: 100 微秒 short power[LASET_POINT]; // 各点的能量大小 short count; // 实际压入点数 } TLaserPower;</pre>

(3) 执行激光能量时间数组

[NMC_LaserTimeArrayExe\(HAND devHandle, short group, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
group	short	输入	组号，最大支持 12 组，取值范围[0, 11]
ch	short	输入	通道号，范围[0, 1]

(3) 缓冲区执行激光能量时间数组

[NMC_CrdBufLaserTimeArrayExe\(HAND crdHandle, long segNo, short group, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
group	short	输入	组号，最大支持 12 组，取值范围[0, 11]
ch	short	输入	通道号，范围[0, 1]

2.3.3 综合示例

a. 配置激光控制模式及信号输出模式

```
//设置激光输出模式为波形模式
rtn = NMC_LaserSetMode(devHandle, TIME_ARRAY_OUTPUT_MODE, 0);
gc_rtn_error(rtn);
// 激光通道配置为DA模拟量输出, 使用扩展模拟量通道256
rtn = NMC_LaserSetOutputType(devHandle, LASER_DA, 256, 0, 0);
gc_rtn_error(rtn);
```

b. 单点激光能量时间数组控制

```
TLaserPower lserPw;
memset(&lserPw, 0, sizeof(TLaserPower));
lserPw.count = 40;
for (int i = 0; i < 20; i++) // 配置时间能量数组
{
    lserPw.time[i] = 100;
    lserPw.power[i] = i * 2;
}
for (int i = 0; i < 20; i++)
{
    lserPw.time[i + 20] = 100;
    lserPw.power[i + 20] = (19 - i) * 2;
}
rtn = NMC_LaserSetTimeArrayPower(devHandle, 0, &lserPw);
gc_rtn_error(rtn);
// 开始执行, 执行调用就开光了。最后一个能量值为0, 控制器在运行完数组后, 将自动关闭激光
rtn = NMC_LaserTimeArrayExe(devHandle, 0, 0);
gc_rtn_error(rtn);
```

c. 位置比较配合焊接

```
//设置激光输出模式为波形模式
rtn = NMC_LaserSetMode(devHandle, TIME_ARRAY_OUTPUT_MODE, 0);
gc_rtn_error(rtn);
// 激光通道配置为DA模拟量输出, 使用扩展模拟量通道256
rtn = NMC_LaserSetOutputType(devHandle, LASER_DA, 256, 0, 0);
gc_rtn_error(rtn);
// 能量控制数组
TLaserPower lserPw;
memset(&lserPw, 0, sizeof(TLaserPower));
lserPw.count = 60;
for (int i = 0; i < 20; i++)
{
    lserPw.time[i] = 100;
```



```

}

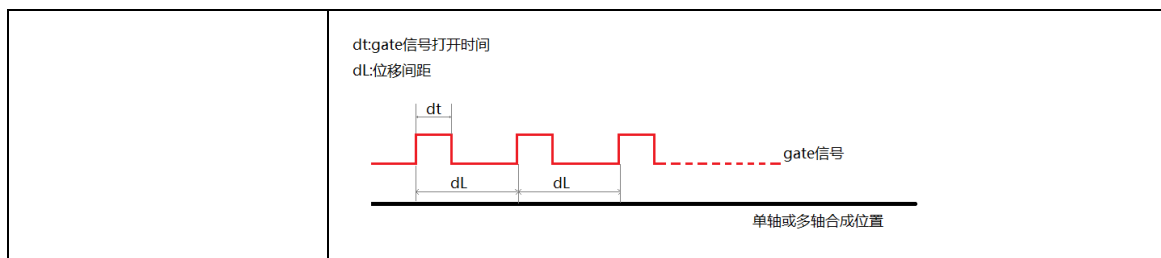
lserPw.power[0] = 2000;
lserPw.power[1] = 4000;
lserPw.power[2] = 6000;
lserPw.power[3] = 4000;
lserPw.power[4] = 2000;
lserPw.power[5] = 0;
rtn = NMC_LaserSetTimeArrayPower(devHandle, 0, &lserPw);
gc_rtn_error(rtn);
// 配置参数
TTimeArrayPara timeAr;
timeAr.stLevelRevs = 0;
timeAr.axisMask = 0x3;           // 参与电机掩码: x3表示第一二轴合成
timeAr.minFrqEn = 0;           // 保留
timeAr.gateDistance = 20;       // 位置比较段长
timeAr.pwmEnable = 1;          // 启用位置比较控制
timeAr.pwmWidth = 1;           // 保留
timeAr.outputType = 0;         // 保留
timeAr.outputCh = 1;           // 保留
timeAr.posSrc = 0;              // 位置源: :encoder, 1:prf
timeAr.pwmPeriod = 500000;     // 不小于时间数组长度
rtn = NMC_LaserSetTimeArrayPara(devHandle, &timeAr, 0);
gc_rtn_error(rtn);
// 开始执行
// 缓冲区请使用NMC_CrdBufLaserTimeArrayExe
rtn = NMC_LaserTimeArrayExe(devHandle, 0, 0);
gc_rtn_error(rtn);
//关闭激光能量输出(缓冲区请使用NMC_CrdBufLaserOnOff)
rtn = NMC_LaserOnOff(devHandle, 0, 0);
gc_rtn_error(rtn);

```

2.4 位置比较控制模式

2.4.1 功能介绍

模式	说明
位置比较控制模式	位置比较控制模式(SHIO)，用于对门控信号进行控制，通常应用是自动通过位置变化量(规划或者编码器)对门控信号进行打开或关闭的控制方式，如下图示意：



2.4.2 指令说明

(1) 配置 SHIO 功能的参数

[NMC_SHIOConfigPara\(HAND devHandle, TSHIOPara *pSHIOPara, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pSHIOPara	TSHIOPara *	输入	<pre>typedef struct{ short isArray; // 是否固定间距还是数组。 0: 固定间距(仅支持), 默认 0 short outMode; // 输出模式。默认 1 // 1: 只输出 gate 立即开或关, // 2: 根据位移输出 gate // 3: 根据位移输出 gate, gate 和同部 trigger 信号同步 // 4: 根据位移输出 gate, gate 和信号输入同步 short posSrc; // 比较模式, 外部编码器还是 内部规划值。 0: 外部编码器(推荐), 1: 内部规 划值。 默认 0。 short axisMask; // 轴号, 按 bit 位对应。(一 般两个轴)。 默认 0。 double delay; // 延时开关光时间(暂不 用), 单位: s。默认 0。 double gateTime; // 设置 gate 打开时间, 单 位: s(内部最小值: 1/36us), 取值范围</pre>

			<p>(0, 0.0009)</p> <p>long gateDistance; // 固定模式下的位置间隔 单位: pulse。默认 0, 模式 2~4 下会进行有效性检查。</p> <p>long k; // 低频过渡系数</p> <p>long startDot; // 启动时出光打点: 0-不打点, 1-打点。</p> <p>long lowFrqRange; // 低频范围: 0:1K, 1:10K, 2:90K。</p> <p>long reserved4; // 保留参数, 应设为 0;</p> <p>long reserved5; // 保留参数, 应设为 0;</p> <p>}TSHIOPara;</p>
ch	short	输入	通道号, 目前只为 0

(2) 启用SHIO最小频率

[NMC_SHIOEnableMinFrq\(HAND devHandle, long minFrq, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
minFrq	long	输入	最小频率, 单位 Hz
ch	short	输入	通道号, 目前只为 0

(3) 关闭SHIO最小频率

[NMC_SHIODisableMinFrq\(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号, 目前只为 0

(4) 缓冲区启用SHIO最小频率

[NMC_CrdBufSHIOSetMinFrq\(HAND crdHandle, long segment, long minFrq, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segment	long	输入	段号(用户自定义)
minFrq	long	输入	最小频率, 单位 Hz
ch	short	输入	通道号, 目前只为 0

(5) 缓冲区关闭SHIO最小频率

[NMC_CrdBufSHIOClrMinFrq\(HAND crdHandle, long segment, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segment	long	输入	段号(用户自定义)
ch	short	输入	通道号, 目前只为 0

(6) 缓冲区配置SHIO参数

[NMC_CrdBufSHIOSetParam\(HAND crdHandle, long segment, double delay, double gateTime, double gateDistance, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segment	long	输入	段号(用户自定义)
delay	double	输入	延时, 单位毫秒
gateTime	double	输入	gate 打开时间, 单位: s(内部最小值: 1/36us), 取值范围(0, 0.0009)
gateDistance	double	输入	位置间隔 单位: pulse
ch	short	输入	通道号, 目前只为 0

(7) 切换SHIO比较轴

[NMC_SHIOChangeAxisMask\(HAND devHandle, short axisMask, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

axisMask	short	输入	轴号掩码，按 bit 位对应，例如比较 0x3 为比较最前面两个轴
ch	short	输入	通道号，目前只为 0

(8) 缓冲区切换SHIO比较轴

[NMC_CrdBufSHIOChangeAxisMask\(HAND crdHandle, long segment, short axisMask, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segment	long	输入	段号(用户自定义)
axisMask	short	输入	轴号掩码，按 bit 位对应，例如比较 0x3 为比较最前面两个轴
ch	short	输入	通道号，目前只为 0

(9) 打开Gate输出

[NMC_SHIOGateOn\(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号，目前只为 0

(10) 关闭Gate输出

[NMC_SHIOGateOff\(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号，目前只为 0

(11) 打开Trigger脉冲输出

[NMC_SHIOTriggerOn\(HAND devHandle, double freq, double width, short ch\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
freq	double	输入	trigger 脉冲频率单位: HZ
width	double	输入	trigger 脉冲宽度, 单位: s
ch	short	输入	通道号, 目前只为 0

(12) 关闭Trigger脉冲输出

[NMC_SHIOTriggerOff\(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
ch	short	输入	通道号, 目前只为 0

(13) 缓冲区打开Gate输出

[NMC_BufSHIOGateOn\(HAND crdHandle, long segNo, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
ch	short	输入	通道号, 目前只为 0

(14) 缓冲区关闭Gate输出

[NMC_BufSHIOGateOff\(HAND crdHandle, long segNo, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
ch	short	输入	通道号, 目前只为 0

(15) PWM映射到GATE引脚输出 (使基本模式和位置比较模式下的激光接口引脚一致)

[NMC_SetPwmToGate\(HAND devHandle, short pwmCh, short gateCh, short onOff\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

pwmCh	short	输入	PWM 通道号, 目前只为 0
gateCh	short	输入	GATE 通道号, 目前只为 0
onOff	short	输入	0 -- 不映射, 1 -- 映射 (默认为 0)

(16)缓冲区SHIO点动出光, 输出一段gate脉冲

[NMC_CrdBufSHIOGatePulse\(HAND crdHandle, long segNo, double gateTime, double gateFrq, long outCount, short ch\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long	输入	段号(用户自定义)
gateTime	double	输入	gate 输出的脉宽, 单位: 微秒
gateFrq	double	输入	gate 输出的频率, 单位: HZ
outCount	long	输入	输出脉冲个数, 为 0 表示连续输出
ch	short	输入	通道号, 目前只为 0

(17)SHIO点动出光, 输出一段gate脉冲

[NMC_SHIOGatePulse\(HAND devHandle, double gateTime, double gateFrq, long outCount, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
gateTime	double	输入	gate 输出的脉宽, 单位: 微秒
gateFrq	double	输入	gate 输出的频率, 单位: HZ
outCount	long	输入	输出脉冲个数, 为 0 表示连续输出
ch	short	输入	通道号, 目前只为 0

(18)输出一段PWM脉冲

[NMC_PwmPulseOut\(HAND devHandle, double onTime, double pwmFrq, long outCount, short ch\);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
onTime	double	输入	PWM 输出的脉宽，单位：微秒
pwmFrq	double	输入	PWM 输出的频率，单位：HZ
outCount	long	输入	输出脉冲个数，为 0 表示连续输出
ch	short	输入	通道号，目前只为 0

(19) 设置 PWM 的输出通道

[NMC_SetPWMPort \(HAND devHandle, short portType, short portIdx, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
portType	short	输入	0:默认通道, 1: GPDO
portIdx	short	输入	映射通道号
ch	short	输入	PWM 通道

(20) 读取 PWM 的输出通道

[NMC_GetPWMPort \(HAND devHandle, short *pPortType, short *pGpoIdx, short ch\);](#)

参考: [SetPWMPort](#)

2.4.3 综合示例

a. 立即输出 Gate 信号，NMC_SHIOGateOn 后立即输出 Gate

```

/***** 立即输出一段GATE信号 *****/
rtn = NMC_LaserSetMode(devHandle, SHIO_OUTPUT_MODE, 0);
gc_rtn_error(rtn);
TSHIOPara shioPrm;
shioPrm.isArray = 0;
shioPrm.outMode = 1; // 立即输出
shioPrm.posSrc = 1;
shioPrm.axisMask = 0x3;
shioPrm.delay = 0;
shioPrm.gateTime = 0.0008;
shioPrm.gateDistance = 20;
rtn = NMC_SHIOConfigPara(devHandle, &shioPrm, 0); //位置比较控制模式参数配置
gc_rtn_error(rtn);

```



```
rtn = NMC_BufSHIOGateOn(devHandle, 1, 0); // 启动gate信号输出,缓冲区中控制gate信号
gc_rtn_error(rtn);
rtn = NMC_BufSHIOGateOff(devHandle, 1, 0); // 关闭gate信号输出
gc_rtn_error(rtn);
// 也可以在非缓冲区中,调用 NMC_SHIOGateOn/NMC_SHIOGateOff 对 gate 信号进行控制
```

b. 根据位移间距输出 gate

```
//位置比较控制模式
TSHIOPara shioPrm;
rtn = NMC_LaserSetMode(devHandle, SHIO_OUTPUT_MODE, 0);
gc_rtn_error(rtn);
shioPrm.isArray = 0;
shioPrm.outMode = 2; // 根据位移间隔输出gate
shioPrm.posSrc = 0; // 位置源为外部编码器
shioPrm.axisMask = 0x3; // XY合成位置
shioPrm.delay = 0; // 无延时
shioPrm.gateTime = 0.00001; // gate打开时间为微秒
shioPrm.gateDistance = 20; // 位移间距为pulse
rtn = NMC_SHIOConfigPara(devHandle, &shioPrm, 0); //参数配置
gc_rtn_error(rtn);
rtn = NMC_BufSHIOGateOn(crdHandle, 1, 0); // 缓冲区中控制gate信号,启动gate信号输出
gc_rtn_error(rtn);
// 运动到起始点
long tgtPos[2];
tgtPos[0] = 20000;
tgtPos[1] = 0;
rtn = NMC_CrdLineXYZ(crdHandle, 154, 0x3, tgtPos, 0, 50, 1);
gc_rtn_error(rtn);
rtn = NMC_BufSHIOGateOff(crdHandle, 1, 0); // 关闭gate信号控制
gc_rtn_error(rtn);
// 加工过程中, gate信号按照设定自动输出
tgtPos[0] = 100000;
tgtPos[1] = 0;
rtn = NMC_CrdLineXYZ(crdHandle, 154, 0x3, tgtPos, 0, 50, 0.01);
gc_rtn_error(rtn);
```

3 PT运动

3.1 功能介绍

PT 模式使用一系列“位置、时间”数据点描述速度规划，用户需要将曲线分割成若干段如下图所示 3.1 所示：

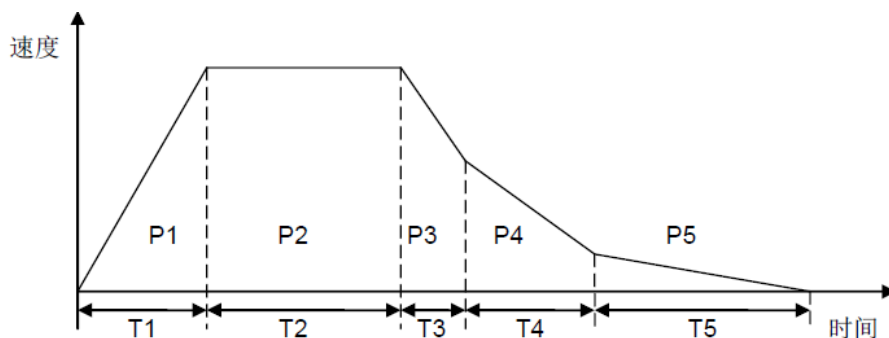


图 3.1 PT 运动速度曲线

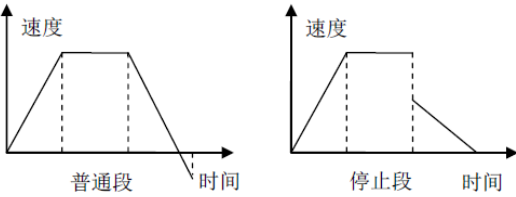
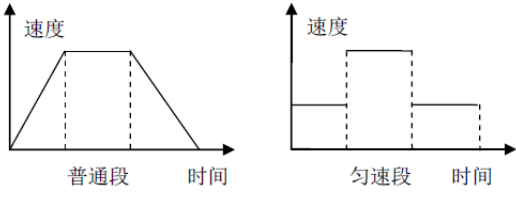
整个速度曲线被分割成 5 段，第 1 段起点速度为 0，经过时间 T_1 运动位移 P_1 ，因第 1 段的终点速度为 $v_1 = \frac{2P_1}{T_1}$ ；第 2 段起点速度为 v_1 ，经过时间 T_2 运动位移 P_2 ，因此第 2 段的终点速度为 $v_2 = \frac{2P_2}{T_2} - v_1$ ；第 3、4、5 段依此类推。PT 模式的数据段要求用户输入每段所需时间和位置点。

PT 模式有两种数据模式：数据驻存模式、数据刷新模式，如下

模式	说明
数据驻存模式	此模式下 PT 运动缓存区运动完后不会清除缓冲区数据，用户可以设置循环次数，循环进行运动；缓冲区大小为 32 段。
数据刷新模式	此模式下 PT 运动缓存区会在两个 FIFO 切换运行，一个 FIFO 运行完后空间会切换到另一个 FIFO，可以继续压入数据；缓冲区大小两个 FIFO 各 32 段，总共 64 段

PT 模式的数据段有 3 种类型，如下：

数据类型	说明
MT_PT_NORMAL	普通段，第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。
MT_PT_STOP	停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。

	
MT_PT_EVEN	<p>匀速段，该段的段内速度保持不变，段内速度=段内位移/段内时间</p> 

PT 是将一系列的点与时间写入到控制器中，在每个采样创建一个实时的位置；

PT 算法：在用户定义的“位置和时间”点之间，PT 算法计算出一个合适的速度曲线。PT 算法保证控制器的轨迹计算符合每一个已知的点位置和时间。分段速度简单的由位置和时间的差分计算出来。

3.2 指令说明

(1) 设置 PT 运动的数据模式和循环次数

[`NMC_MtPtSetStatic\(HAND axisHandle, short onOff, long loopCount\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
onOff	short	输入	1: 数据驻存模式 0: 数据刷新模式(默认模式)
loopCount	long	输入	循环次数：仅数据驻存模式下有效

(2) 读取 PT 运动的数据模式和循环次数

[`NMC_MtPtGetStatic\(HAND axisHandle, short *pOnOff, long *pLoopCount\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pOnOff	short *	输出	返回数据模式
pLoopCount	long *	输出	返回循环次数

(3) 读取 PT 运动缓存区剩余空间大小

[NMC_MtPtGetSpace\(HAND axisHandle, short *pSpace, short *pUsed\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pSpace	short *	输出	返回的剩余空间大小
pUsed	short *	输出	已使用的空间段数

(4) 向 PT 运动缓存区中压运动数据段

[NMC_MtPtPush\(HAND axisHandle, short count, double*pPosArray, long *pTimeArray, short *pTypeArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
count	short	输入	压入的数据段数，取值范围[1, 32]
pPosArray	double *	输入	段运行距离，相对于 PT 起点的绝对位置，单位：脉冲
pTimeArray	long *	输入	段运行时间，相对于 PT 起点的绝对时间，单位：毫秒
pTypeArray	short *	输入	段类型 MT_PT_NORMAL (0) 普通段 MT_PT_STOP (1) 停止段 MT_PT_EVEN (2) 匀速段

(5) 清空 PT 缓冲区

[NMC_MtPtBufClr\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(6) 启动 PT 运动

[NMC_MtPtStartMtn\(HAND axisHandle, short otherSynAxCnts, short *pOtherSynAxArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
otherSynAxCnts	short	输入	不包括 axisHandle 的其他同步启动轴数量
pOtherSynAxArray	short *	输入	其他同步启动轴的序号：0~N

(7) 设置 PT 运动的相关参数

[NMC_MtPtSetPara\(HAND axisHandle, TPtPara *pPara \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPara	TPtPara *	输入	<pre>typedef struct{ double smoothDec; //停止减速度 double abruptDec; //急停减速度，暂时保留 double reserved1[2]; // 保留 long reserved2[4]; // 保留 }TPtPara;</pre>

3.3 综合示例

a. 数据驻存模式(静态 FIFO 模式)

```

/***** 已省略打开控制器部分，统一描述在第五章1 *****/
rtn = NMC_MtSetPrfMode(axisHandle, MT_PT_PRF_MODE); // 设置轴为PT运动模式
gc_rtn_error(rtn);
/*****PT数据驻存模式*****/
rtn = NMC_MtPtSetStatic(axisHandle, 1, 5); //设置为PT数据驻存模式, 循环次
gc_rtn_error(rtn);
rtn = NMC_MtPtBufClr(axisHandle); //清空PT数据缓冲区
gc_rtn_error(rtn);
//查询PT缓冲区空间
short space = 0;
short used = 0;
rtn = NMC_MtPtGetSpace(axisHandle, &space, &used);
gc_rtn_error(rtn);
double posArray[3] = { 10000, 30000, 40000 }; //相对于PT起点的绝对位置
long timeArray[3] = { 100, 200, 300 }; //相对于PT起点的绝对时间

```

```

short typeArray[3] = { MT_PT_NORMAL, MT_PT_NORMAL, MT_PT_NORMAL };
rtn = NMC_MtPtPush(axisHandle, 3, posArray, timeArray, typeArray); //往PT缓冲区压数据
gc_rtn_error(rtn);
short otherSynAxArray[] = { 0 };
rtn = NMC_MtPtStartMtn(axisHandle, 0, otherSynAxArray); //启动PT运动
gc_rtn_error(rtn);
while (1)
{
    short sts = 0;
    rtn = NMC_MtGetSts(axisHandle, &sts);
    gc_rtn_error(rtn);
    if ((sts & BIT_AXIS_BUSY) != BIT_AXIS_BUSY) { break; } //等待运动结束
}

```

b. 数据刷新模式(动态 FIFO)

```

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

/***** 已省略打开控制器部分，统一描述在第五章1 *****/
rtn = NMC_MtSetPrfMode(axisHandle, MT_PT_PRF_MODE); // 设置轴为PT运动模式
gc_rtn_error(rtn);
/***** PT数据刷新模式 *****/
rtn = NMC_MtZeroPos(axisHandle); // 位置清零
gc_rtn_error(rtn);
rtn = NMC_MtPtSetStatic(axisHandle, 0, 0); //设置为PT数据刷新模式
gc_rtn_error(rtn);
rtn = NMC_MtPtBufClr(axisHandle); //清空PT数据缓冲区
gc_rtn_error(rtn);
double pos = 0;
double vel = 0;
double velPre = 0;
double time = 0;
short start = 0;
short loop = 1;
long ltime = 0;
short type = 0;
short otherSynAxArray = 1;
while (1)
{
    short space = 0;
    short used = 0;

```

```
rtn = NMC_MtPtGetSpace(axisHandle, &space, &used); //查询PT缓冲区空间
gc_rtn_error(rtn);
if (space > 0)
{
    time += DELTA;
    vel = A * sin((2 * PI) / T * time); // 计算段末速度
    pos += 1000 * (vel + velPre) * DELTA / 2; // 计算段内位移
    velPre = vel;
    if (time < loop * T)
    {
        ltime = (long)(time * 1000);
        type = MT_PT_NORMAL;
        //发送新数据, 往PT缓冲区压数据
        rtn = NMC_MtPtPush(axisHandle, 1, &pos, &ltime, &type);
        gc_rtn_error(rtn);
    }
    else
    {
        pos = 0;
        ltime = (long)(loop * T * 1000);
        type = MT_PT_STOP;
        //发送终点数据, 往PT缓冲区压数据
        rtn = NMC_MtPtPush(axisHandle, 1, &pos, &ltime, &type);
        gc_rtn_error(rtn);
        pos = 0;
        time = loop * T;
        velPre = 0;
        ++loop;
        if (loop > LOOP) { break; }
    }
}
else if (0 == start) {
    rtn = NMC_MtPtStartMtn(axisHandle, 0, &otherSynAxArray); // 启动PT运动
    gc_rtn_error(rtn);
    start = 1;
}
}

while (1) {
    _sleep(50);
    short sts = 0;
    rtn = NMC_MtGetSts(axisHandle, &sts); // 获得轴状态
    gc_rtn_error(rtn);
    if ((sts & BIT_AXIS_BUSY) != BIT_AXIS_BUSY) { break; } //等待运动结束
}
```

4 PVT运动

4.1 功能介绍

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

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

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

如果给定相邻 2 个数据点的“位置、速度、时间”参数，可以得到如下方程组：

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

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

4.2 指令说明

(1) 设置 PVT 运动的相关参数

[NMC MtPvtSetPara\(HAND axisHandle, TPvtPara *pPara \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPara	TPvtPara *	输入	<pre>typedef struct{ double smoothDec;//停止减速度 double abruptDec;//急停减速度，暂时保留 double reserved1[2];//保留 long dataMode;//工作模式,1:数据驻存模式 0:数据刷新模式(默认) long loopCount;//循环次数：</pre>

			仅数据驻存模式下有效 long reserved2[4]; // 保留 } TPvtPara;
--	--	--	---

(2) 读取 PVT 运动的相关参数

[NMC MtPvtGetPara\(HAND axisHandle, TPvtPara *pPara \);](#)

参考: [NMC MtPvtSetPara](#)

(3) 向 PVT 运动缓存区中压运动数据段

[NMC MtPvtData\(HAND axisHandle, short count, double *pPosArray, double *pTimeArray, double *pVelArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
count	short	输入	压入的数据段数: 1~32
pPosArray	double *	输入	段运行距离
pTimeArray	double *	输入	段运行时间
pVelArray	double *	输入	段运行速度

(4) 查询 PVT 数据剩余空间大小

[NMC MtPvtBufGetSpace\(HAND axisHandle, short *pFreeSpace, short *pUsedSpace\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pFreeSpace	short *	输出	返回的剩余空间大小
pUsedSpace	short *	输出	已使用的空间段数

(5) 清空 PVT 数据

[NMC MtPvtBufClr\(HAND axisHandle\);](#)

(6) 启动 PVT 运动

[NMC_MtPvtStartMtn\(HAND axisHandle, short otherSynAxCnts, short *pOtherSynAxArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
otherSynAxCnts	short	输入	不包括 axisHandle 的其他同步启动轴数量
pOtherSynAxArray	short *	输入	其他同步启动轴的序号：0~N

注意：同步的轴必须属于同一个控制器

5 闭环控制(选配)

5.1 功能介绍

5.1.1 开环控制模式(脉冲控制)

开环控制是不将控制的结果反馈回来影响当前控制的系统。在使用步进电机，或者伺服电机的位置控制模式时，上位机程序发出运动指令，控制器接收到命令之后以脉冲的形式发给驱动器，驱动器收到脉冲开始运动，开环控制控制框图如图 5.1.1；在这个过程中，编码器位置信息的采集并不参与控制调整，但是它可以用来判断运动是否真正到位，仅此而已；在高川运动控制器默认为脉冲控制模式，即开环控制模式(对应驱动器的位置控制模式)。

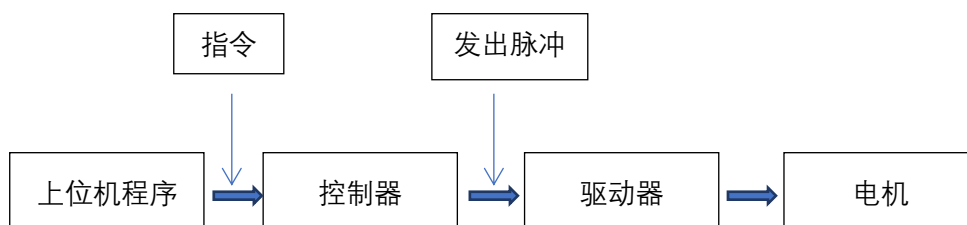


图 5.1.1 开环控制框图

5.1.2 闭环控制模式(模拟量控制)

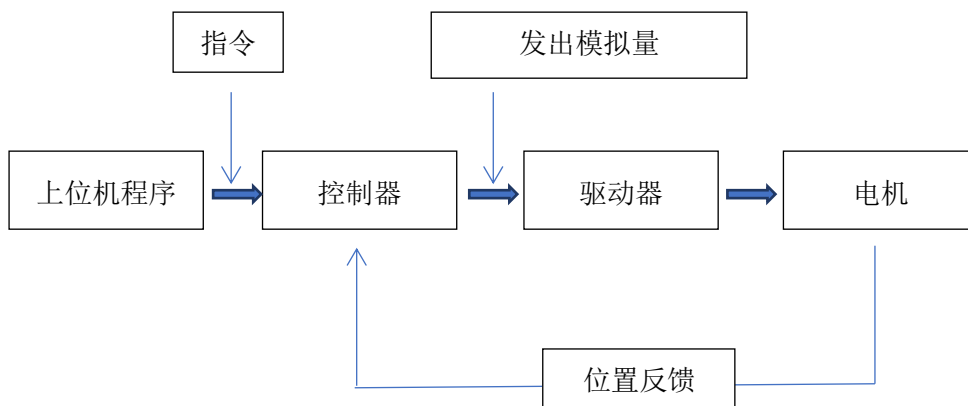
闭环控制是根据控制对象输出反馈来进行校正的控制方式，它是在测量出实际与计划发生偏差时，对输出进行影响。在使用伺服电机的速度控制模式时，上位机程序发出运动指令，控制器不断搜集编码器或者光栅尺反馈的位置与规划器位置的差值，即跟随误差，通过一定的控制算法得到实时的控制量(电压)来控制电机的运动。

电机的位置反馈信息可通过编码器或者光栅尺直接接入到控制器，也可以先接入驱动器，再由驱动器传给控制器。

在高川运动控制器中需要把轴运动模式通过调用函数[NMC_MtSetCtrlMode](#)设置为闭环模式，在调试过程之前需要确认好轴运动方向和编码器反馈是否一致，以免发生撞击事件，可先将驱动器设置成位置控制模式，发脉冲来确认轴的规划位置和编码器位置是否方向一致，大小相同，如果不一致，需要调整驱动器编码器反馈方向和反馈比例。

不论是开环控制器模式还是闭环控制器模式，除了轴的配置不同外，其余比如运动指令调用、使能等都是同样的使用方法。

支持模拟量控制的控制器型号(选配)，就可以把轴运动控制设置为模拟量控制的闭环模式。



5.2 指令说明

(1) 设置单轴闭环控制的 DA 参数, 轴与 DA 通道的对应

[`NMC_MtSetCloseLoopDac\(HAND axisHandle, TDacMotor *pDacPara\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pDacPara	TDacMotor *	输入	<pre>typedef struct{ short inverse; // 电压是否取反 short bias; // Dac零漂 short dacLmt; // Dac输出极限值 }TDacMotor;</pre>

(2) 读取单轴闭环控制的 DA 参数

[`NMC_MtGetCloseLoopDac\(HAND axisHandle, TDacMotor *pDacPara\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pDacPara	TDacMotor *	输出	返回 Dac 参数结构

(3) 设置单轴的控制模式；默认使用对应轴的编码器作为输入反馈, 对应序号的 DAC 作为输出；

闭环模式下, 先调用 NMC_SetCloseLoopDac 指令

[`NMC_MtSetCtrlMode\(HAND axisHandle, short mode\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mode	short	输入	控制模式:0 脉冲控制, 1 DA 闭环控制

(4) 读取单轴的控制模式

[NMC_MtGetCtrlMode\(HAND axisHandle, short *pMode\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pMode	short *	输出	返回控制模式

(5) 设置对应组号的 PID 参数

[NMC_MtSetPIDPara\(HAND axisHandle, short index, TPidPara *pidPara\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
index	short	输入	组号, 范围[0, 2]
pidPara	TPidPara *	输入	<pre>typedef struct{ float kp; // 增益系数 float ki; // 积分系数 float kd; // 微分系数 float kvff; // 速度前馈系数 long integralLimit; // 积分饱和极限 long derivativeLimit; // 微分饱和极限 short outLimit; // 输出饱和极限 } TPidPara;</pre>

(6) 读取对应组号的 PID 参数

[NMC_MtGetPIDPara\(HAND axisHandle, short index, TPidPara *pPidPara\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
index	short	输入	组号，范围[0, 2]
pPidPara	TPidPara *	输出	返回 pid 参数结构

(7) 设置使用哪组 PID

[NMC_MtSetPIDIndex\(HAND axisHandle, short index\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
index	short	输入	组号，范围[0, 2]

(8) 读取正使用的 PID 组号

[NMC_MtGetPIDIndex\(HAND axisHandle, short *pIndex\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pIndex	short *	输出	返回组号，范围[0, 2]

(9) 设置允许的位置误差, 当位置误差超过设定值时, 电机停止运动, 提示位置误差超限

[NMC_MtSetPosErrLmt\(HAND axisHandle, long posErr \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posErr	long	输入	误差脉冲数，闭环模式下 posErr 不能为 0

5.3 综合示例

编程控制过程如下(闭环轴的编码器必需设置为外部编码器；建议初次使用闭环将

[NMC_MtSetCloseLoopDac](#) 参数里的 dacLmt 设一个较小值如 500, 避免 inverse 方向不对导致高速飞车)：

```
short gc_example(void)
```

```
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    short axisIndex = 0;
    //闭环设置
    TDacMotor dacMotor;
    //更改运动方向，这个值和编码器取反要一同修改
    dacMotor.inverse = 0;
    dacMotor.bias = 0;
    //32767对应模拟量最大输出值V，初次调试建议设一个较小值如，避免inverse和编码器不匹配导致高速飞车
    dacMotor.dacLmt = 500;
    rtn = NMC_MtSetCloseLoopDac(axisHandle, &dacMotor);
    gc_rtn_error(rtn);
    rtn = NMC_SetEncMode(devHandle, axisIndex, 0x0000); //设置编码器为外部编码器
    gc_rtn_error(rtn);
    rtn = NMC_MtSetCtrlMode(axisHandle, 1); //设置为闭环控制模式
    gc_rtn_error(rtn);
    rtn = NMC_MtSetPosErrLmt(axisHandle, 32767); //设置闭环最大位置误差必需设置
    gc_rtn_error(rtn);
    TPidPara pidPara;
    rtn = NMC_MtGetPIDPara(axisHandle, 0, &pidPara); //读取闭环PID
    gc_rtn_error(rtn);
    pidPara.kp = 5;
    pidPara.ki = 1;
    rtn = NMC_MtSetPIDPara(axisHandle, 0, &pidPara); //设置闭环PID 必需设置
    gc_rtn_error(rtn);
    rtn = NMC_MtSetPIDIndex(axisHandle, 0); //选择使用的闭环PID 必需设置
    gc_rtn_error(rtn);
    rtn = NMC_MtSetSvOn(axisHandle);
    gc_rtn_error(rtn);
    /***** 下面就可以直接调用轴运动指令了 *****/
    return 0;
}
```

6 电子齿轮

6.1 功能介绍

Gear 运动模式，即电子齿轮运动模式，能够实现两轴或者多轴运动的速度按指定比例同步，从而替代常见的机械齿轮连接机构。其主要特点如下：

- (1) 每组 Gear 运动关联的两个轴，被跟随轴称为主轴，跟随轴称为从轴，多个从轴可以随同一个主轴，从轴又可以作为其他轴的主轴；
- (2) 从轴可以跟随主轴的规划位置或者实际位置，从轴也可以直接跟随外部的编码器输入；
- (3) 从轴可以指跟随主轴的某个运动方向(正向或者负向)或者双向跟随；
- (4) 主轴和从轴之间按照固定的速度比例运动，即传动比，可以在运动过程随时修改；
- (5) 当启动电子齿轮或者改变传动比时，可以设置离合区，从而让跟随过程更加平滑。离合区越大，则同步过程越平滑，其速度-时间曲线示意图如 5.1.1 下(红色线为主轴速度，绿色线为从轴速度)：

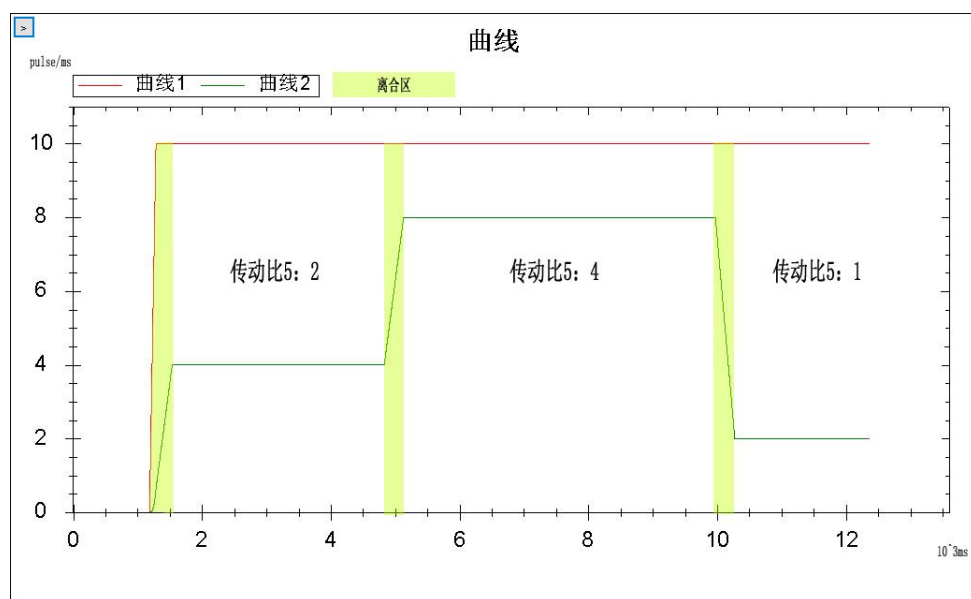


图 6.1.1 速度-时间曲线

- (6) 根据传动比，离合区的不同，主轴带动从轴运动。上位机向伺服系统(运动控制器)发出位置指令，位置指令脉冲有 2 种形式，如图 6.1.2 为位置指令脉冲示意图。

- ① 脉冲+方向
- ② CW 脉冲+CCW 脉冲(正脉冲+负脉冲)

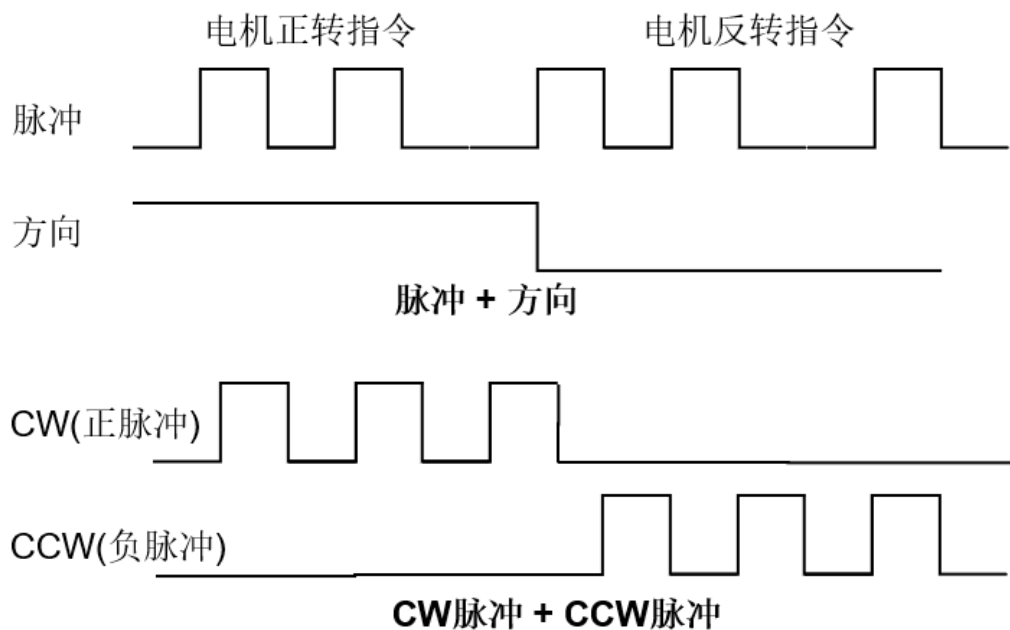


图 6.1.2 位置指令脉冲示意图

位置指令脉冲包含电机的位移和方向两个方面，伺服系统的位置反馈脉冲当量(控制器输出一个定位控制脉冲时，所产生的定位控制移动的位移)由检测器(如光电脉冲编码器)的分辨率，以及电机每转对应的机械位移量等决定。当指令脉冲单位与位置反馈脉冲当量不一致时，就可以使用电子齿轮使二者完全匹配：

6.2 指令说明

(1) 设置从轴跟随方向

[`NMC_MtGearSetDir\(HAND axisHandle, short dir\);`](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
dir	short	输入	=0: 双向跟随 <0: 负向跟随 >0: 正向跟随

(2) 读取从轴跟随方向

[`NMC_MtGearGetDir\(HAND axisHandle, short* pdir\);`](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

axisHandle	HAND	输入	从轴句柄
pdir	short*	输出	=0: 双向跟随 <0: 负向跟随 >0: 正向跟随

(3) 设置 Gear 主轴参数

[NMC MtGearSetMaster\(HAND axisHandle, short masterNo, short masterType\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
masterNo	short	输入	主轴序列号 (0~N)
masterType	short	输入	主轴类型 1: AXIS 规划值 2: AXIS 反馈值 3: 编码器值

(4) 读取 Gear 主轴参数

[NMC MtGearGetMaster\(HAND axisHandle, short * pmasterNo, short * pmasterType\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pmasterNo	short*	输出	主轴序列号 (0~N)
pmasterType	short*	输出	主轴类型 1: AXIS 规划值 2: AXIS 反馈值 3: 编码器值

(5) 设置 Gear 跟随倍率

[NMC MtGearSetRatio\(HAND axisHandle, long masterEven, long slaveEven, long masterSlope\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
masterEven	long	输入	传动比系数, 主轴位移
slaveEven	long	输入	传动比系数, 从轴位移
masterSlope	long	输入	离合器位移: 必须大于 0, 同时不能等于 1

			启动跟随倍率改变后，到倍率更新完成过程中，主轴的位移值越大，则离合区越长；倍率变换期间主轴的移动距离，不按照加速度的方式，而是一个位置的对应关系；主轴是恒速的，从轴则匀加速跟随；0 表示没有离合区；
--	--	--	---

(6) 读取 Gear 跟随倍率

[NMC MtGearGetRatio\(HAND axisHandle, long *pMasterEven, long *pSlaveEven, long *pMasterSlope\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pMasterEven	long *	输出	传动比系数，主轴位移
pSlaveEven	long *	输出	传动比系数，从轴位移
pMasterSlope	long *	输出	离合区位移，必须大于 0，同时不能等于 1

(7) 启动 Gear 运动

[NMC MtGearStartMtn\(HAND axisHandle, short syncAxCnts, short *pSyncAxArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
syncAxCnts	short	输入	不包括 axisHandle 的其他同步启动轴数量
pSyncAxArray	short *	输入	其他同步启动轴的序号：0~N

6.3 综合示例

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    //设置轴2的运动模式为电子齿轮模式(轴2为主轴，轴1为从轴)
    rtn = NMC_MtSetPrfMode(axisHandle[0], MT_GEAR_PRF_MODE);
    gc_rtn_error(rtn);
}
```

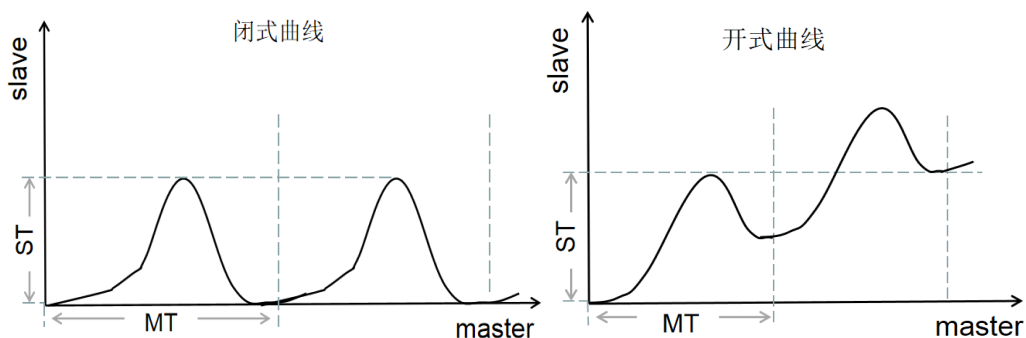
```
rtn = NMC_MtGearSetDir(axisHandle[0], 0);           //设置Gear跟随方向
gc_rtn_error(rtn);
rtn = NMC_MtGearSetMaster(axisHandle[0], 1, 1);     //设置主轴2规划器
gc_rtn_error(rtn);
rtn = NMC_MtGearSetRatio(axisHandle[0], 5, 4, 2000); //设置传动比5:4和离合区
gc_rtn_error(rtn);
//启动电子齿轮运动，当主轴运动时，从轴将按照设置的传动比和离合区运动,运动过程中调用
//NMC_MtGearSetRatio可更新传动比和离合区参数
rtn = NMC_MtGearStartMtn(axisHandle[0], 0, 0);
gc_rtn_error(rtn);
return 0;
}
```

7 电子凸轮

7.1 功能介绍

Follow 运动模式，即电子凸轮运动模式，能够实现两轴或者多轴运动的速度和位置同步，其主要特点如下：

- (1) 电子凸轮属于多轴同步运动(速度位置同步)，这种运动(也叫 Follow 运动)是基于主轴和一个或者多个从轴系统。主轴可以是物理轴，也可以是虚拟轴(一种算法，没有实际的输出)；
- (2) 从轴可以跟随主轴的规划位置或者实际位置，从轴也可以直接跟随外部的编码器输入；
- (3) 从轴可以指跟随主轴的某个运动方向(正向或者负向)或者双向跟随；
- (4) 与从轴之间的同步，通过用户设定的多个数据段自动规划，每个数据段包含主轴位移，从轴位移，速度规划类型三个参数，即主轴在完成设定位移的过程中，从轴也自动完成设定位移，这个过程的速度曲线由速度规划类型决定；
- (5) 每个从轴有两个 FIFO 用于缓存同步数据段，每个 FIFO 最多可以存储 32 个数据段，通过手动切换，其中一个 FIFO 的数据用完后会自动切换到另外一个 FIFO；
- (6) 通过配置 FIFO 的循环次数，可实现从轴周期性的跟随；
- (7) 同步的启动可以配置为立即启动，也可以配置为主轴穿越某个位置时自动启动；
- (8) 启动位置和终点位置是否一致可以将电子凸轮曲线分为闭式曲线和开式曲线，如图 7.1.1；



7.2 指令说明

(1) 设置 FOLLOW 跟随方向

[NMC_MtFollowSetDir\(HAND axisHandle, short dir\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
dir	short	输入	= 0: 双向跟随 < 0: 负向跟随 > 0: 正向跟随

(2) 读取 FOLLOW 跟随方向

[NMC_MtFollowGetDir\(HAND axisHandle, short *pDir\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pDir	short *	输出	= 0: 双向跟随 < 0: 负向跟随 > 0: 正向跟随

(3) 设置 FOLLOW 主轴参数

[NMC_MtFollowSetMaster\(HAND axisHandle, short masterNo, short masterType\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄

masterNo	short	输入	主轴序列号 (0~N)
masterType	short	输入	主轴类型 <pre>#define PROFILE_FOLLOW_MASTER_NONE</pre> (0) // 0:无效 <pre>#define PROFILE_FOLLOW_MASTER_AXIS_PRF</pre> (1) // 1:AXIS 规划值 <pre>#define PROFILE_FOLLOW_MASTER_AXIS_ENC</pre> (2) // 2: AXIS 反馈值 <pre>#define PROFILE_FOLLOW_MASTER_ENC</pre> (3) // 3: 编码器值

(4) 读取 FOLLOW 主轴参数

[NMC MtFollowGetMaster\(HAND axisHandle, short *pMasterNo, short *pMasterType\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pMasterNo	short*	输出	主轴序列号 (0~N)
pMasterType	short*	输出	主轴类型，见宏定义

(5) 设置 FOLLOW 的循环执行次数

[NMC MtFollowSetLoopCount\(HAND axisHandle, long loopCnt\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
loopCnt	long	输入	循环次数

(6) 读取 FOLLOW 的循环执行次数

[NMC MtFollowGetLoopCount\(HAND axisHandle, long *pLoopCnt\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
ploopCnt	long *	输出	循环次数

(7) 设置 FOLLOW 的启动事件

[NMC MtFollowSetEvent\(HAND axisHandle, short eventType, short masterDir, long pos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
eventType	short	输入	1 表示调用启动指令以后立即启动 2 表示主轴穿越设定位置以后启动跟随
masterDir	short	输入	穿越启动时, 主轴的运动方向: 1 主轴正向运动, -1 主轴负向运动
pos	long	输入	穿越位置

(8) 读取 FOLLOW 的启动事件

[NMC MtFollowGetEvent\(HAND axisHandle, short *pEventType, short *pMasterDir, long *pPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pEventType	short *	输出	1 表示调用启动指令以后立即启动 2 表示主轴穿越设定位置以后启动跟随
pMasterDir	short *	输出	穿越启动时, 主轴的运动方向: 1 主轴正向运动, -1 主轴负向运动
pPos	long *	输出	穿越位置

(9) 读取 FOLLOW 的 FIFO 剩余空间

[NMC MtFollowGetSpace\(HAND axisHandle, short *pSpace, short fifoNo\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
pSpace	short*	输出	空间大小
fifoNo	short	输入	FIFO 号, 0 或 1

(10) 设置 FOLLOW 的数据

[NMC MtFollowPushData\(HAND axisHandle, long masterPos, double slavePos, short type, short](#)

[fifoNo;](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
masterPos	double	输入	主轴位移
slavePos	double	输入	从轴位移
type	short	输入	数据段类型：0 普通段，默认；1 匀速段；2 减速到 0 段；3 保持 FIFO 之间速度连续
fifoNo	short	输入	FIFO 号，0 或 1

(11)清除 FOLLOW 对应 FIFO 号的数据

[NMC MtFollowClear\(HAND axisHandle, short fifoNo\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
fifoNo	short	输入	FIFO 号，0 或 1

(12)启动 Follow 运动

[NMC MtFollowStart\(HAND axisHandle, short syncAxCnts, short *pSyncAxArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
syncAxCnts	short	输入	不包括 axisHandle 的其他同步启动轴数量
pSyncAxArray	short *	输入	其他同步启动轴的序号：0~N

(13)切换 Follow 运动的 FIFO 号

[NMC MtFollowSwitch\(HAND axisHandle, short syncAxCnts, short *pSyncAxArray\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	从轴句柄
syncAxCnts	short	输入	不包括 axisHandle 的其他同步进行 FIFO

			切换的轴数量
pSyncAxArray	short *	输入	其他同步进行 FIFO 切换的轴的序号：0~N

7.3 综合示例

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

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

```

short gc_example(void)
{
    short rtn;                // 指令返回值
    short devNum;             // 设备序号，从0开始
    short axisNum;            // 轴号，从0开始
    HAND devHandle;           // 设备句柄
    HAND axisHandle[2];       // 轴句柄
    HAND crdHandle;           // 坐标系句柄

    rtn = NMC_DevOpen(devNum, &devHandle);           //打开控制器
    gc_rtn_error(rtn);
    for (short i = 0; i < 2; i++) {
        rtn = NMC_MtOpen(devHandle, i, &axisHandle[i]); //打开轴, 并获得轴句柄
        gc_rtn_error(rtn);
        rtn = NMC_MtSetSvOn(axisHandle[i]);           //打开轴使能(根据实际情况使用)
        gc_rtn_error(rtn);
    }

    //启动主轴JOG运动, 速度脉冲/ms, 加、减速度为脉冲/ms^2
    rtn = NMC_MtMoveJog(axisHandle[0], 1, 1, 50, 0, 1);
    gc_rtn_error(rtn);
    //设置轴二的运动模式为Follow模式
    rtn = NMC_MtSetPrfMode(axisHandle[1], MT_FOLLOW_PRF_MODE);
    gc_rtn_error(rtn);
    rtn = NMC_MtFollowClear(axisHandle[1], 0);        //清空从轴FIFO数据
    gc_rtn_error(rtn);
    //设置跟随主轴的规划位置

```

```
rtn = NMC_MtFollowSetMaster(axisHandle[1], 0, PROFILE_FOLLOW_MASTER_AXIS_PRF);
gc_rtn_error(rtn);
double masterpos = 20000;
double slavepos = 10000;
rtn = NMC_MtFollowPushData(axisHandle[1], masterpos, slavepos, 0, 0); //设置跟随数据
gc_rtn_error(rtn);
masterpos += 20000;
slavepos += 20000;
rtn = NMC_MtFollowPushData(axisHandle[1], masterpos, slavepos, 0, 0);
gc_rtn_error(rtn);
masterpos += 20000;
slavepos += 10000;
rtn = NMC_MtFollowPushData(axisHandle[1], masterpos, slavepos, 0, 0);
gc_rtn_error(rtn);
rtn = NMC_MtFollowSetLoopCount(axisHandle[1], 0); //设置为无限循环
gc_rtn_error(rtn);
//设置跟随条件, 主轴正向运动到位置时启动
rtn = NMC_MtFollowSetEvent(axisHandle[1], 2, 1, 50000);
gc_rtn_error(rtn);
rtn = NMC_MtFollowStart(axisHandle[1], 0, 0); //启动Follow运动
gc_rtn_error(rtn);
//跟随运动过程中可调用NMC_MtFollowGetLoopCount 查询循环执行次数
return 0;
}
```

8 捕获功能

8.1 功能介绍

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，IO 捕获，编码器 Z 相捕获，IO+Z 相捕获以及先 IO 触发再 Z 相触发捕获。函数 [NMC MtSetCaptSns](#) 可以设置捕获的方式，捕获的 IO 输入源以及捕获触发沿；高级捕获功能还可以支持对一个信号的前沿和后沿同时进行捕获。捕获是否触发可以从 [NMC MtGetSts](#) 判断；

在运动控制器上，轴 1 对应原点 (HM0)、正向限位 (LM0+)、负向限位 (LM0-) 和数字 IO (DI0~DI15) 输入的引脚丝印。

硬件捕获 IO 源定义表	
宏定义	说明
<code>CAPT_IO_SRC_HOME = 0</code>	原点输入作为捕获 IO
<code>CAPT_IO_SRC_LMTN = 1</code>	负向限位输入作为捕获 IO
<code>CAPT_IO_SRC_LMTP = 2</code>	正向限位输入作为捕获 IO
<code>CAPT_IO_SRC_DI0 = 3</code>	通用数字输入 0 作为捕获 IO
<code>CAPT_IO_SRC_DI1 = 4</code>	通用数字输入 1 作为捕获 IO
<code>CAPT_IO_SRC_DI2 = 5</code>	通用数字输入 2 作为捕获 IO
<code>CAPT_IO_SRC_DI3 = 6</code>	通用数字输入 3 作为捕获 IO
<code>CAPT_IO_SRC_DI4 = 7</code>	通用数字输入 4 作为捕获 IO
<code>CAPT_IO_SRC_DI5 = 8</code>	通用数字输入 5 作为捕获 IO
<code>CAPT_IO_SRC_DI6 = 9</code>	通用数字输入 6 作为捕获 IO
<code>CAPT_IO_SRC_DI7 = 10</code>	通用数字输入 7 作为捕获 IO
<code>CAPT_IO_SRC_DI8 = 11</code>	通用数字输入 8 作为捕获 IO
<code>CAPT_IO_SRC_DI9 = 12</code>	通用数字输入 9 作为捕获 IO
<code>CAPT_IO_SRC_DI10 = 13</code>	通用数字输入 10 作为捕获 IO
<code>CAPT_IO_SRC_DI11 = 14</code>	通用数字输入 11 作为捕获 IO
<code>CAPT_IO_SRC_DI12 = 15</code>	通用数字输入 12 作为捕获 IO

8.2 指令说明

通用捕获

(1) 设置硬件捕获参数

[NMC_MtSetCaptSns\(HAND axisHandle, short mode, short ioSrc, short level \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mode	short	输入	编码器硬件捕获模式 CAPT_MODE_Z(宏定义 0) 编码器 Z 相捕获 CAPT_MODE_IO(宏定义 1) IO 捕获 CAPT_MODE_Z_AND_IO(宏定义 2) IO+Z 相捕获 CAPT_MODE_Z_AFT_IO(宏定义 3) 先IO触发再Z相触发捕获
ioSrc	short	输入	硬件捕获 IO 源选择, 见上表
level	short	输入	触发边沿, bit0:index 电平 (0 为下降沿, 1 为上升沿), bit1:IO 电平

(2) 读取硬件捕获参数

[NMC_MtGetCaptSns\(HAND axisHandle, short *pMode, short *pIoSrc, short *pLevel \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pMode	short *	输出	返回硬件捕获模式
pIoSrc	short *	输出	返回硬件捕获 IO 源
pLevel	short *	输出	返回触发边沿, bit0:index 电平 (0 为下降沿, 1 为上升沿), bit1:IO 电平

(3) 启动捕获功能

[NMC_MtSetCapt\(HAND axisHandle \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

注意: 捕获是否触发可以从 [NMC_MtGetSts](#) 判断。

(4) 读取捕获触发时编码器位置

[NMC_MtGetCaptPos\(HAND axisHandle, long *pPos \);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPos	long *	输出	返回捕获位置

(5) 清除轴的捕获状态

[NMC_MtClrCaptSts\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

高级捕获

```
// 捕获源
#define CAPT_EX_SRC_GPI 0          // 通用输入
#define CAPT_EX_SRC_NEGLMT = 1;    // 负向限位
#define CAPT_EX_SRC_POSLMT = 2;    // 正向限位
#define CAPT_EX_SRC_HOME = 3;      // 原点
#define CAPT_EX_SRC_Z = 4;         // Z向信号
#define CAPT_EX_SRC_PRFPPOS = 5;   // 规划位置（当规划位置达到设定值时触发）
#define CAPT_EX_SRC_ENCPOS = 6;    // 编码器位置
```

(6) 设置高级捕获参数, 并启动捕获

[NMC_MtSetAdvCaptParam\(HAND devHandle, TAdvCaptureParam *pParam, short ch\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	轴句柄
pParam	TAdvCaptureParam *	输入	<pre>typedef struct{ short capPosIndex; //捕获的位置源序号, 0~N: //轴 1~N+1。(默认 0)注: 捕获的编码器位置根据用户 //设置的编码器模式决定 short trigSrc; //触发源, 见上面的定义, (默认: CAPT_EX_SRC_GPI) short trigIndex; // 触发源序号。(默认 0)</pre>

			<pre>short filter; // 滤波时间常数, 单位 0.1 毫秒, 取值范围[0, 255] long trigValue; // 触发值, 对于触发源为 I0, 表示信号触发的有效电平; 对于触发源为位置, 则表示触发捕获的位置。(默认 0) } TAdvCaptureParam;</pre>
ch	short	输入	捕获通道号, 取值范围[0, 3]

(7) 清除高级捕获状态, 并取消该通道的捕获

[NMC_MtClrAdvCaptSts \(HAND devHandle, short ch\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
ch	short	输入	捕获通道号, 取值范围[0, 3]

(8) 读取高级捕获状态

[NMC_MtGetAdvCaptPos \(HAND devHandle, short *captSts, long *pPosArray, short ch\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
captSts	short *	输出	捕获状态, bit0 表示设置的信号前沿触发到, bit1 表示信号的后沿触发到。
pPosArray	long *	输出	返回位置, 在触发到时, 位置为捕获到的位置值 pPosArray[0]: 信号前沿触发位置, pPosArray[1]: 信号后沿触发位置, 单位: 脉冲
ch	short	输入	捕获通道号, 取值范围[0, 3]

(9) 设置重复捕获次数

[NMC_MtSetCaptRepeat \(HAND axisHandle, short count\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

count	short	输入	重复捕获次数，取值范围[0, REPEAT_CAPT_MAX_NUM], 0表示取消重复捕获 REPEAT_CAPT_MAX_NUM = 64
-------	-------	----	---

注意：设置后将清除捕获数据；

(10) 读取重复捕获计数

[NMC_MtGetCaptRepeatStatus\(HAND axisHandle, short *pCount\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pCount	short *	输出	已经捕获次数

(11) 读取重复捕获位置值

[NMC_MtGetCaptRepeatPosMulti\(HAND axisHandle, long *pPosArray, short startNum, short count\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPosArray	long *	输出	返回的位置值数组
startNum	short	输入	起始计数
count	short	输入	重复捕获次数，取值范围[1, 32], 一次最多读取 32 个

8.3 综合示例

a. 捕获功能

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    //设置捕获参数, DIO为捕获源，下降沿触发
    rtn = NMC_MtSetCaptSns(axisHandle, CAPT_MODE_IO, CAPT_IO_SRC_DIO, 0);
    gc_rtn_error(rtn);
    rtn = NMC_MtSetCapt(axisHandle); //启动捕获
    gc_rtn_error(rtn);
    /***** 此处省略轴的运动过程 *****/
    short sts;
    long pos;
```

```
while (1)
{
    rtn = NMC_MtGetSts(axisHandle, &sts);
    gc_rtn_error(rtn);
    if ((sts & BIT_AXIS_CAPTSET) == BIT_AXIS_CAPTSET)    //已经捕获到
    {
        rtn = NMC_MtGetCaptPos(axisHandle, &pos);        //读取捕获位置
        gc_rtn_error(rtn);
        break;
    }
}
return 0;
}
```

b. 高级捕获功能

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    TAdvCaptureParam advCaptureParam;
    advCaptureParam.capPosIndex = 0;
    advCaptureParam.filter = 100;
    advCaptureParam.trigIndex = 0;
    advCaptureParam.trigSrc = CAPT_EX_SRC_GPI;
    advCaptureParam.trigValue = 0;
    rtn = NMC_MtSetAdvCaptParam(devHandle, &advCaptureParam, 0); //设置高级捕获功能
    gc_rtn_error(rtn);
    /***** 此处省略轴的运动过程 *****/
    short sts;
    long pos[2];
    while (1)
    {
        rtn = NMC_MtGetAdvCaptPos(devHandle, &sts, pos, 0);
        gc_rtn_error(rtn);
        if (sts == 0x3)    //前后沿均已捕获到
        {
            break;
        }
    }
    return 0;
}
```


9 机械补偿

9.1 螺距补偿

9.1.1 功能介绍

将数控机床某轴的指令坐标位置与高精度测量系统所得的实际坐标位置相比较,计算出在全程上的误差,并分别绘制出其曲线(描点),再将该误差曲线数值化并以表格的形式输入到数控系统中,以提高机床的定位精度。螺距补偿只对机床补偿段起作用,在数控系统允许的范围区间起到补偿作用。

9.1.2 指令说明

(1) 设置螺距误差补偿参数

[NMC MtSetLeadScrewCompPara\(HAND axisHandle, short num, long startPos, long cmpLen, short *pCompPos, short *pCompNeg\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
num	short	输入	补偿的段数, 取值范围[2, 128]
startPos	long	输入	补偿起始位置, 终止位置为 startPos + cmpLen
cmpLen	long	输入	补偿区间总长度
pCompPos	short *	输入	正向补偿值数组
pCompNeg	short *	输入	负向补偿值数组

注意: num 从起点为第 1 段, 如 num=5, 则 cmpLen/4=区间长度; 补偿值会平均补偿到该区间长度;

(2) 启动或者关闭螺距误差补偿

[NMC MtEnableLeadScrew\(HAND axisHandle, short enable\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
enable	short	输入	开关, 1: 启动, 0: 关闭

9.1.3 综合示例

螺距误差补偿

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    short comPos[] = { 10, 20, -10 };
    short comNeg[] = { 5, -5, 5 };
    //设置螺距误差补偿参数, 3个点(2段区间长度), 起点, 终点
    rtn = NMC_MtSetLeadScrewCompPara(axisHandle, 3, 10000, 20000, comPos, comNeg);
    gc_rtn_error(rtn);
    rtn = NMC_MtEnableLeadScrew(axisHandle, 1); //启动补偿
    gc_rtn_error(rtn);
    return 0;
}
```

9.2 反间隙补偿

9.2.1 功能介绍

在数控机床上，由于各坐标轴进给传动链上驱动部件的反向死区、各机械运动传动的反向间隙等误差的存在，造成各坐标轴在由正向运动转为反向运动时形成反向偏差，通常也称反向间隙或失动量；

9.2.2 指令说明

(1) 设置反向间隙补偿参数

[NMC_MtSetBacklash\(HAND axisHandle, long compValue, double compDelta, long compDir\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
compValue	long	输入	补偿量，单位脉冲
compDelta	double	输入	周期补偿量，比如补偿量为 100，周期补偿量为 20，则反向补偿会在 5 个规划周期完成 100 个脉冲的补偿
compDir	long	输入	补偿方向, 0: 正向补偿, 1: 负向补偿

(2) 读取反向间隙补偿参数

[NMC_MtGetBacklash\(HAND axisHandle, long *pCompValue, double *pCompDelta, long *pCompDir\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pCompValue	long *	输出	返回补偿量
pCompDelta	double *	输出	返回周期补偿量
pCompDir	long *	输出	返回补偿方向

9.2.3 综合示例

反向间隙补偿(负向补偿应该在回零之后,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,反之正向补偿需要回零后负向预紧)

```
//设置反向间隙补偿参数,负向补偿脉冲1000,周期补偿量为200
rtn = NMC_MtSetBacklash(axisHandle, 1000, 200, 1);
gc_rtn_error(rtn);
```

9.3 2D平面补偿

9.3.1 功能介绍

在精密的运动控制领域中,如果传动组件精准度不够,会导致生产时的机械手臂或者导轨上的装置在移动时产生位置偏移,从而达不到精度要求。常见的 XY 滑台(如图 9.3.1)只有移动轴的自由度,若导轨加工不良,组装不良,就会造成角度的误差,从而机台在组装时造成空间误差,因此,必须测量出角度误差,知道误差后进行补偿,从而降低误差。

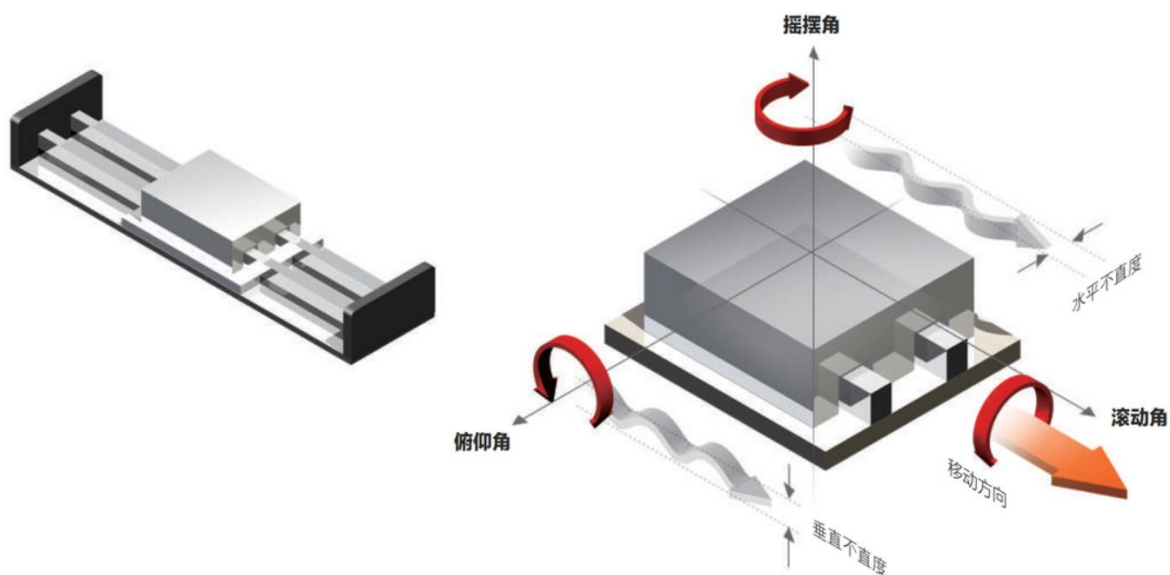


图 9.3.1 XY 滑台示意图

根据以上分析，我们可以通过 2D 误差补偿技术来提高加工精度，下图 9.3.2 为 2D 误差补偿示意图，如果用户想要依据测量计算的结果补偿 X 与 Y 方向的位置，需要先建立在对应的坐标上，对 X 轴向与 Y 轴向需补偿的位置值，并提供控制器相关的设定值，包含距离间距与补偿。开启误差补偿功能后，控制器的轴命令中，输出的位置命令会叠加需要补偿的位移量，以消除测量出已知的误差量。

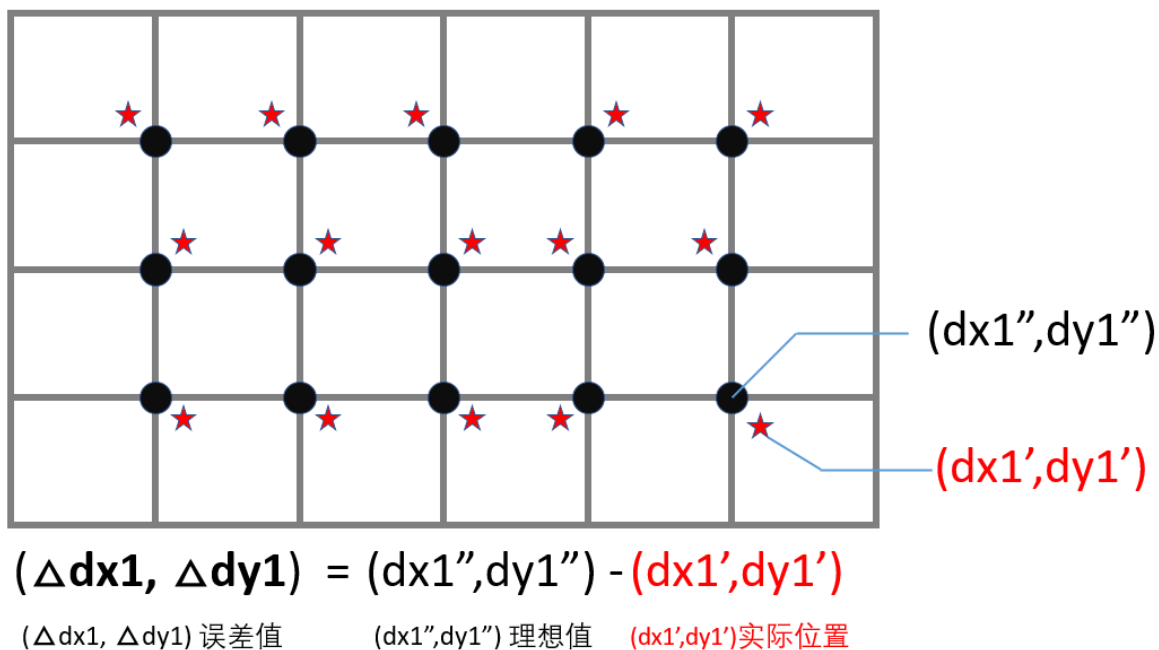


图 9.3.2 2D 误差补偿示意图

9.3.2 指令说明

(1) 设置补偿表：以 XY 为基准参考，标定 XYZ 方向的偏差数据，并进行设置

`NMC_Set2DCompensationTable(HAND devHandle, short tableIndex, T2DCompensationTable *pTable, TCompData *pData);`

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
tableIndex	short	输入	取值 0(为后续扩展准备)
pTable	T2DCompensationTable *	输入	<pre> #define XY2DCOMP_MAX_TABLE_NUM (4) //平面补偿的最大表格数量 typedef struct{ short count[2]; // count[0]和count[1]分别为X, Y方向的数据点数, 每个方向最小2个 short reserved[2]; // 保留 long posBegin[2]; // posBegin[0]和posBegin[1]分别为X, Y方向的起始补偿位置 long step[2]; // step[0]和step[1]分别为XY方向的补偿步长 double angle; // 标定坐标系与机械坐标系的夹角, 单位: 度 } T2DCompensationTable; </pre>
pData	TCompData *	输入	<pre> typedef struct{ long xDirComp; long yDirComp; long zDirComp; } TCompData; </pre> <p>//数据包含 X*Y 个点, 点数组总长度 X*Y*3<=3000, 每个点包含 XYZ 3 个方向的补偿值, 数组按照 x 方向排列</p>

(2) 读取补偿表

[NMC_Get2DCompensationTable\(HAND devHandle, short tableIndex, T2DCompensationTable *pTable, TCompData *pData\);](#)

参考: [NMC_Set2DCompensationTable](#)

(3) 设置并启动 XY 平面误差补偿

[NMC_Set2DCompensation\(HAND devHandle, T2DCompensation *pComp\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pComp	T2DCompensation *	输入	<pre>typedef struct{ short enable;// 启动 (1) 或停止 (0) 补偿 short tableIndex; // 使用的目录表: 取值0 short axisType[2]; // 2D补偿表使用的轴类型:规划:0, 编码器:1; axisType[0]对应X方向, axisType[1]对应Y方向; short axisIndex[3];// axisIndex[0]:X方向补偿的轴号, axisIndex[1]:Y方向补偿的轴号; axisIndex[2]:Z方向补偿的轴号 } T2DCompensation;</pre>

(4) 读取 XY 平面的误差补偿参数

[NMC_Get2DCompensation\(HAND devHandle, T2DCompensation *pComp\);](#)

参考: [NMC_Set2DCompensation](#)

9.3.3 综合示例

```
//二维位置补偿示例
short gc_example(void)
{
    /***** 已省略打开控制器部分, 统一描述在第五章1 *****/
    /***** 设置补偿参数, 并启动补偿 *****/
}
```

```

T2DCompensationTable tTable;
tTable.angle = 0;           //坐标系夹角，可取X方向第一个点与最后一个点的Y方向上偏差值
                             //计算角度，单位：度
tTable.count[0] = 3;       //X方向的数据点数3个
tTable.count[1] = 4;       //Y方向的数据点数4个
tTable.posBegin[0] = 0;    //X方向的起始补偿位置0
tTable.posBegin[1] = 0;    //Y方向的起始补偿位置0
tTable.step[0] = 3000;     // X方向的补偿步长，即为补偿间距，每3000脉冲补偿一个点
tTable.step[1] = 4000;     // Y方向的补偿步长，即为补偿间距，每4000脉冲补偿一个点
tTable.reserved[0] = 0;    // 保留值
TCompData tCompData[3*4]; // 补偿量数组
for(int i=0; i<3*4;i++)
{
    tCompData[i].xDirComp = i%3;      // x方向上的补偿量
    tCompData[i].yDirComp = i/4;      // y方向上的补偿量
    tCompData[i].zDirComp = 0;        // z方向上的补偿量
}
rtn = NMC_Set2DCompensationTable(devHandle, 0, &tTable,&tCompData[0]);
gc_rtn_error(rtn);
T2DCompensation t2DCompensation;
t2DCompensation.tableIndex = 0;      // 使用的目录表：取值0
t2DCompensation.axisType[0] = 1;     // 0表示规划，1表示编码器；X方向类型
t2DCompensation.axisType[1] = 1;     // 0表示规划，1表示编码器；Y方向类型
t2DCompensation.axisIndex[0] = 0;    // X方向的轴号
t2DCompensation.axisIndex[1] = 1;    // Y方向的轴号
t2DCompensation.axisIndex[2] = -1;   // Z方向的轴号
t2DCompensation.enable = 1;          // 启动(1)或停止(0)补偿
rtn = NMC_Set2DCompensation(devHandle, &t2DCompensation); // 设置并启动XY平面误差补偿
gc_rtn_error(rtn);
return rtn;
}

/***** 取消补偿 *****/
short gc_example(void)
{
    T2DCompensation comp;
    comp.enable = 0;      // 取消补偿
    comp.tableIndex = 0;  // 使用0号补偿表
    comp.axisType[0] = 0; // X轴根据规划位置补偿
    comp.axisType[1] = 0; // Y轴根据规划位置补偿
    comp.axisIndex[0] = 0; // X轴轴号0
    comp.axisIndex[1] = 1; // Y轴轴号0
    comp.axisIndex[2] = -1;
    short rtn = NMC_Set2DCompensation(devHandle, &comp);
    gc_rtn_error(rtn);
}

```

```
return rtn;  
}
```


10 扩展资源

10.1 辅助编码器

指令说明

(1) 读取编码器通道值

[NMC GetEncPos\(HAND devHandle, short encId, long *pPos \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
encId	short	输入	对于轴编码器通道，取值范围[0, n] 对于扩展编码器通道，256 表示第一个扩展编码器通道，257 表示第二个，以此类推
pPos	long *	输出	返回编码器通道数值

(2) 设置编码器通道值

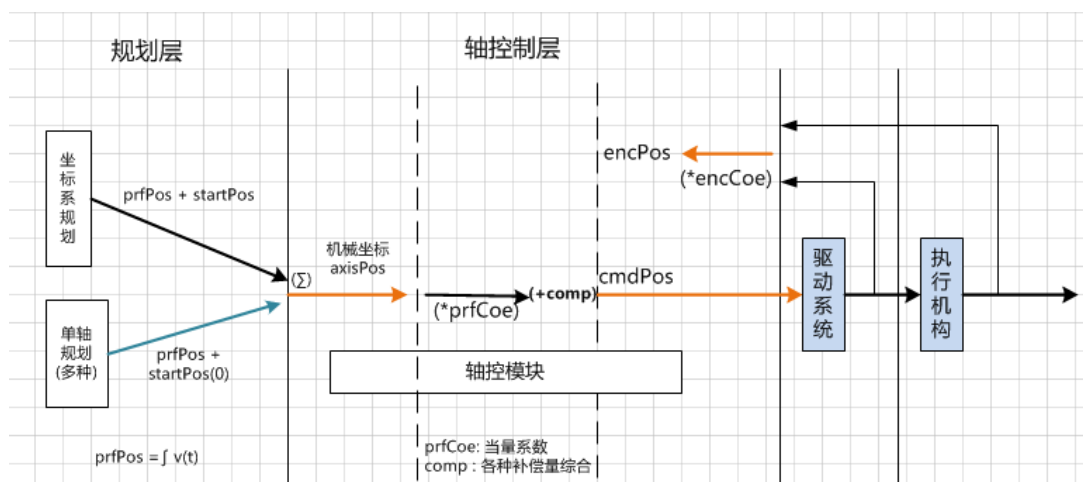
[NMC SetEncPos\(HAND devHandle, short encId, long encPos \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
encId	short	输入	对于轴编码器通道，取值范围[0, n] 对于扩展编码器通道，256 表示第一个扩展编码器通道，257 表示第二个，以此类推
encPos	long	输入	编码器通道设定数值

10.2 位置系统操作

10.2.1 功能介绍

控制系统的位置系统框图如下：



变量名	说明	对应的指令
prfPos	规划位置，规划器产生的位置	NMC_MtGetPrfPos NMC_CrdGetVel
StartPos	用户设定的位置偏差： 单轴通过 NMC_MtPrfConfig 设置 坐标系通过 NMC_CrdSetPara 设置	NMC_MtPrfConfig NMC_MtGetPrfConfig NMC_CrdSetPara NMC_CrdGetPara
PrfCoe	规划的比例系数	NMC_MtSetPrfCoe NMC_MtGetPrfCoe
comp	补偿量(反向间隙补偿、螺距补偿等)	
axisPos	机械位置， $axisPos = prfPos + startPos$	NMC_MtGetAxisPos NMC_CrdGetAxisPos
cmdPos	命令位置 ($axisPos * prfCoe$, 经过滤波、补偿等处理后得到)，指发送到驱动系统的指令位置(脉冲数)	NMC_MtGetCmdPos
encPos	编码器位置，机构的反馈值	NMC_MtGetEncPos NMC_GetEncPos
encCoe	编码器反馈的比例系数 注意： $encCoe = (\text{命令脉冲} / \text{命令脉冲对应的编码器反馈值})$ 。例如，往驱动器发 1000 个脉冲，驱动器反馈 250，则 encCoe 设置为 4	NMC_MtSetEncCoe NMC_MtGetEncCoe

10.2.2 指令说明

(1) 单轴位置系统清零, 规划以及编码器

[NMC MtZeroPos\(HAND axisHandle\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄

(2) 设定机械位置, 轴静止时执行, 如果后面是运动指令, 需要延时一个周期

[NMC MtSetAxisPos\(HAND axisHandle, long axisPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
axisPos	long	输出	设定机械位置值 注意: 只能在轴静止时使用

(3) 设定编码器位置, 轴静止时执行, 如果后面是运动指令, 需要延时一个周期

[NMC MtSetEncPos\(HAND axisHandle, long encPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
axisPos	long	输出	设定编码器位置值 注意: 只能在轴静止时使用

(4) 设置单轴比例系数

[NMC MtSetPrfCoe\(HAND axisHandle, double inCoe\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
inCoe	double	输入	单轴比例系数, 取值范围 (0, 1]

(5) 读取单轴比例系数

[NMC MtGetPrfCoe\(HAND axisHandle, double *inCoe\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
inCoe	Double*	输出	单轴比例系数, 取值范围 (0, 1]

(6) 设置轴通道编码器的系数, 默认为 1

[NMC_MtSetEncCoe\(HAND axisHandle, double encCoe\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
encCoe	double	输入	单轴比例系数, 取值范围 (0, 1]

(7) 读取轴通道编码器的系数

[NMC_MtGetEncCoe\(HAND axisHandle, double *pEncCoe\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pEncCoe	Double*	输出	单轴比例系数, 取值范围 (0, 1]

(8) 设置轴的到位误差参数

[NMC_MtSetAxisArrivalPara\(HAND axisHandle, long arrivalBand, long stableTime\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
arrivalBand	double	输入	到位误差值, 单位 Pulse 取值大于 0
stableTime	long	输入	到位保持时间, 单位 ms 取值大于 0

(9) 读取轴的到位误差参数

[NMC_MtGetAxisArrivalPara\(HAND axisHandle, long *pArrivalBand, long *pStableTime\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pArrivalBand	long *	输出	到位误差值, 单位 Pulse
pStableTime	long *	输出	到位保持时间, 单位 ms

(10) 设置单轴规划高级参数

[NMC MtPrfConfig\(HAND axisHandle, short mapAxisNo, short port, long startPos\);](#)

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
mapAxisNo	long *	输入	轴号, 取值范围[0, n]
port	long *	输入	端口号, 取值范围[0, 1], 默认为 0
startPos	long	输入	偏置, 默认为 0

(11) 读取单轴规划高级参数

[NMC MtGetPrfConfig\(HAND axisHandle, short *mapAxisNo, short *port, long *startPos\);](#)

参考: [NMC MtPrfConfig](#)

(12) 读取多轴状态

[NMC MtGetStsMulti\(HAND firstAxisHandle, short *pStsArray, short count, unsigned long *pClock\);](#)


名称	数据类型	输入/输出	描述
firstAxisHandle	HAND	输入	第一个轴句柄
pStsArray	short *	输出	返回的状态数组
count	short	输入	轴数量
pClock	unsigned long *	输出	返回当前控制器时钟, 单位 125us

注意: PCIE卡适用或者网络卡特定固件适用;

10.3 控制器资源

10.3.1 功能介绍

控制器内部为用户保留2K大小的参数区, 方便用户根据自己的需求使用, 如给控制器加密; 还可以修改控制器时间, 用户信息等。

	<p>特别提示: 控制器存储电池寿命按出厂时间计算预计 3 到 5 年, 电池失效后相关功能运作将会不正常, 请合理使用, 建议只存储加密、收款方面数据, 不建议存储应用程序</p>
---	---

的参数。本节会在电池寿命相关的功能处作标注说明，请留意。

10.3.2 指令说明

(1) 读取时间

[NMC_GetTime\(HAND devHandle, TNMCTime *pTime \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pTime	TNMCTime *	输出	返回时间结构，参考TNMCTime 结构定义 <pre>typedef struct{ short ear; // 真实年份[2000,2099] short mon; // 月,取值范围[1,12] short day; // 日,取值范围[1,31] short hour; // 时,取值范围[0,23] short min; // 分,取值范围[0,59] short second; // 秒,取值范围[0,59] } TNMCTime;</pre>
	电池寿命预计 3 到 5 年，电池失效后读取和设置时间会不正确，请合理使用；		

(2) 设置时间

[NMC_SetTime\(HAND devHandle, TNMCTime *pTime,char *pPassword\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pTime	TNMCTime *	输入	时间结构，参考 TNMCTime 结构定义：
pPassword	char *	输入	控制器系统密码，长度最多为16
	电池寿命预计 3 到 5 年，电池失效后读取和设置时间会不正确，请合理使用；		

(3) 读取设备唯一序列号

[NMC_GetUID\(HAND devHandle, unsigned long *pUID \);](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

h	HAND	输入	控制器句柄
pUID	unsigned long *	输出	返回设备唯一序列号

(4) 设置用户参数

[NMC UserParaWr\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pWrData \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
add	unsigned long	输入	参数区的偏移地址(字节地址)
len	unsigned long	输入	写入长度, 单位: 字节
pWrData	unsigned char *	输入	要写入的数据 注意: 1) 写入的数据掉电不丢失。 2) 一次最多写256字节 3) 参数区总长度为2048字节。 4) 此指令比其它指令操作时间会长

注意: 写入用户参数的次数是有限制的, 理论上不低于 5000 次, 需要合理安排策略;

(5) 读取用户参数

[NMC UserParaRd\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pRdData \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
add	unsigned long	输入	参数区的偏移地址(字节地址)
len	unsigned long	输入	写入长度, 单位: 字节
pRdData	unsigned char *	输出	读取出数据存储的数组

(6) 设置通讯参数

[NMC SetCommPara\(HAND devHandle, unsigned long waitTimeInUs, unsigned long retryTimes\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
waitTimeInUs	unsigned long	输入	等待时间, 微秒
retryTimes	unsigned long	输入	通讯重试次数

(7) 修改板卡 ID 号

[NMC_DevWriteID\(HAND devHandle, char *pIdStr \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pIdStr	char *	输入	要写入的板卡 ID 字符串, 最长 16 字节, 以\0 结尾; 修改 ID 号完成后, 板卡要掉电重启, 新的 ID 才有效。

(8) 读取板卡 ID 号

[NMC_DevReadID\(HAND devHandle, char *pIdStr \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pIdStr	char *	输入	存储字符串的数组, 数组长度大于 16 字节

(9) 读取库的版本

[NMC_GetDllVersion\(char *pVersion\);](#)

名称	数据类型	输入/输出	描述
pVersion	char *	输出	接收版本信息

(10) 读取当前运动控制器固件的版本等信息

[NMC_GetMtLibVersion\(HAND devHandle, TMtLibVersion *pVersion \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pVersion	TMtLibVersion *	输出	接收版本信息

(11) 读取当前运动控制器信息

[NMC_GetCardInfo\(HAND devHandle, TCardInfo *pInfo \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pInfo	TCardInfo *	输出	返回信息结构，参考 TCardInfo 结构定义 <pre>typedef struct{ char strVer[16]; // 控制器版本号 char strOemVer[16]; // OEM 版本号 (定制控制器才有)，默认为 0 short time[6]; // 时间，年月日，时分秒 short axisNum; // 支持的轴数 short encNum; // 支持的编码器数 short diNum; // 数字输入数量 short doNum; // 数字输出数量 short daNum; // 模拟量通道 short adNum; // 模拟量输入通道 short shioNum; // 同步高速 IO 通道 short reserved; // 保留 char ipv4[4]; // IP 地址 char idStr[16]; // 板卡名称，多卡可用名称 打开参考：NMC_DevOpenByID unsigned long uid[4]; // 唯一序列号 }TCardInfo;</pre>

(12) 启动指令调试

[NMC_SetCmdDebug\(short enable, char *debugOutputFile\);](#)

名称	数据类型	输入/输出	描述
enable	short	输入	0 关闭调试信息，默认关闭，1: 打开通讯的 debug 输出；2: 打印到文件；3: 打印输出

			到 GCS(管理员方式打开); 4 打印到 debugView(管理员方式打开)
debugOutputFile	char *	输出	调试输出文件

Ex: ①NMC_SetCmdDebug(0, "") ②NMC_SetCmdDebug(2, "d://test.txt") ③NMC_SetCmdDebug(3, "")

④NMC_SetCmdDebug(4, "")

(13) 读取错误代码信息

[NMC_GetErrDesc\(short errCode, char *errDesc\);](#)

名称	数据类型	输入/输出	描述
errCode	short	输入	错误代码
errDesc	char *	输出	返回的错误代码描述

(14) 保存为配置文件


[NMC_SaveConfigToFile\(HAND devHandle, char *pFilePath\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pFilePath	char *	输入	文件路径

(15) 读取当前备份的变量数值(断电自动保存)


[NMC_GetBackedVarGroup2\(HAND devHandle, TBackGroup2 *pBackVar\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pBackVar	TBackGroup2 *	输出	备份的变量值, 参考结构体定义 <pre>typedef struct{ long crdSegNo[2]; //坐标系运动的段号 long crdPrfPos[2][3]; // 坐标系运动的规划位置 long axPrfPos[12]; // 规划位置 long axisPos[12]; // 机械位置 }</pre>

			<code>long encPos[12]; // 编码器位置</code> <code>}TBackGroup2;</code>
	电池寿命预计 3 到 5 年，电池失效后该功能会受到影响，请合理使用；		


(16) 开启或关闭变量自动备份功能(断电自动保存)

[`NMC_SetBackedVarOnOff\(HAND devHandle, short en\);`](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
en	short	输入	是否开启，1: 开启变量的自动备份，0: 停止变量的自动备份
	电池寿命预计 3 到 5 年，电池失效后该功能会受到影响，请合理使用；		

(17) 读取当前自动备份的开启状态

[`NMC_GetBackedVarOnOff\(HAND devHandle, short *pEn\);`](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pEn	short *	输出	当前的开启状态
	电池寿命预计 3 到 5 年，电池失效后该功能会受到影响，请合理使用；		

(18) 读取系统参数(long 型)

[`NMC_DevGetPara\(HAND devHandle, unsigned long paraID, long *pValue \);`](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
paraID	unsigned long	输入	系统参数 ID
pValue	long *	输入	返回系统参数值

(19) 设置系统参数(long 型)

[`NMC_DevSetPara\(HAND devHandle, unsigned long paraID, long value \);`](#)

名称	数据类型	输入/输出	描述
----	------	-------	----

devHandle	HAND	输入	控制器句柄
paraID	unsigned long	输入	系统参数 ID
value	long	输入	设置参数值

注意：IP 地址等参数写入成功后，需要调用 `NMC_DevSetPara(devHandle, PARA_WRITE_EN, 1)` 才能使写入的参数保存，控制器重新启动后生效；

(20) 修改控制器系统密码, 密码用于修改系统时钟等

[NMC_ChangePassword\(HAND devHandle, char *pPasswordOld, char *pPasswordNew\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPasswordOld	char *	输入	控制器系统当前密码, 长度最多为 15 个字符
pPasswordNew	char *	输入	新的控制器系统密码, 长度最多为 15 个字符

(21) 验证系统密码

[NMC_VerifyPassword\(HAND devHandle, char *pPassword\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPassword	char *	输入	控制器系统当前密码, 长度最多为 15 个字符

(22) 设置用户密码

[NMC_UserSetPassword\(HAND devHandle, char *pUserName, char *pPasswordOld, char *pPasswordNew\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pUserName	char *	输入	指定的用户名, 目前只支持"USER1"
pPasswordOld	char *	输入	该用户当前密码, 长度最多为 15 个字符
pPasswordNew	char *	输入	新的该用户密码, 长度最多为 15 个字符

(23) 用户登陆

[NMC_UserLogin\(HAND devHandle, char *pUserName, char *pPassword\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pUserName	char *	输入	指定的用户名, 目前只支持"USER1"
pPassword	char *	输入	该用户当前密码, 长度最多为 15 个字符

(24) 用户退出登陆

[NMC_UserLogout\(HAND devHandle, char *pUserName\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pUserName	char *	输入	指定的用户名, 目前只支持"USER1"

(25) 设置控制器的规划周期

[NMC_SetProfilePeriod\(HAND devHandle, short period\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
period	short	输入	规划周期, 单位 us, 只能为 250 500 1000

(26) 读取控制器的规划周期

[NMC_GetProfilePeriod\(HAND devHandle, short *pPeriod\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pPeriod	short *	输入	返回规划周期

(27) 读取最后一次的错误代码

[NMC_GetLastError\(void\);](#)

参考: 略

(28) 设置指令错误返回值模式

[NMC_SetErrCodeMode\(short_mode\);](#)

名称	数据类型	输入/输出	描述
mode	short	输入	0-标准模式, 将返回详细的错误代码(默认); 1-简洁模式, 只返回错误代码类别

(29) 设置指令通讯看门狗

[NMC_SetWatchDog\(HAND devHandle, long timeout, short stopMode, short groupID, long doValue\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
timeout	long	输入	看门狗超时时间, 单位毫秒, 小于等于 0 代表关闭看门狗功能
stopMode	short	输入	超时停止模式, 1: 马上停止, 0: 缓冲区执行完毕后停止
groupID	short	输入	超时输出 do 的组号
doValue	long	输入	超时输出 do 状态

(30) 设置扩展用户参数

[NMC_UserParaWrEx\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pWrData \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
add	unsigned long	输入	参数区的偏移地址(字节地址)
len	unsigned long	输入	写入长度, 单位: byte
pWrData	unsigned char *	输入	要写入的数据

注意: ① 写入的数据掉电不丢失;

② 一次最多写 1000 字节, 参数区总长度为 32K 字节;

③ 此指令比其它指令操作时间会长, 如果出现通讯错误(返回-1), 则需要将通过

[NMC_SetCommPara](#) 延长指令通讯时间;

(31) 读取扩展用户参数

[NMC_UserParaRdEx\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pRdData \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
add	unsigned long	输入	参数区的偏移地址(字节地址)
len	unsigned long	输入	写入长度, 单位: byte
pRdData	unsigned char *	输出	要读取的数据存储

注意: 一次最多读取 1000 字节。参数区总长度为 32K 字节。(参考函数 : NMC_UserParaWr)

(32) 打印信息

[NMC_UserPrint\(char *msg\);](#)

名称	数据类型	输入/输出	描述
msg	char *	输入	信息内容

(33) 读取计时时钟, 控制器上电开始

[NMC_GetClock\(HAND devHandle, unsigned long *pClock, unsigned long *pLoop\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	轴句柄
pClock	unsigned long *	输出	计数值, 单位: 125us
pLoop	unsigned long *	输出	循环周期计数

10.4 数据采集

10.4.1 功能介绍

数据采集模块, 用于实时采集(最小数据间隔 1 个规划周期)控制器工作数据, 用于分析和验证加工过程中轨迹, 速度等是否满足要求。数据采集模块提供给用户二次开发的 API, 用户可以通过这些 API 实现自己的采集过程。用户也可以通过调试工具 GCS 来完成数据采集;

10.4.2 指令说明

(1) 读取需要采集数据变量的地址

`NMC_GetCollectDataAddr(HAND devHandle, short index, short dataType, unsigned long *pAddr, short *pDataLen);`

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
index	short	输入	变量的序号(从 0 开始)
dataType	short	输入	变量的类型, 参数‘Collect 模块: 变量类型’宏定义
pAddr	unsigned long *	输出	返回的数据地址
pDataLen	short *	输出	返回的该数据长度

注意:

<code>#define COLLECT_ADDRESS_PRF_POS</code>	(0) // 规划位置
<code>#define COLLECT_ADDRESS_AXIS_POS</code>	(1) // 机械位置
<code>#define COLLECT_ADDRESS_ENC_POS</code>	(2) // 编码器位置
<code>#define COLLECT_ADDRESS_CMD_POS</code>	(3) // 命令位置
<code>#define COLLECT_ADDRESS_AXIS_VEL</code>	(4) // 电机速度
<code>#define COLLECT_ADDRESS_CRD_POS</code>	(5) // 坐标系位置
<code>#define COLLECT_ADDRESS_CRD_VEL</code>	(6) // 坐标系速度
<code>#define COLLECT_ADDRESS_CRD_POS0</code>	(5) // 坐标系位置
<code>#define COLLECT_ADDRESS_CRD_VELO</code>	(6) // 坐标系速度
<code>#define COLLECT_ADDRESS_ENC_VEL</code>	(7) // 编码器速度
<code>#define COLLECT_ADDRESS_CMD_VEL</code>	(8) // 命令速度
<code>#define COLLECT_ADDRESS_CRD_POS1</code>	(9) // 坐标系位置
<code>#define COLLECT_ADDRESS_CRD_VEL1</code>	(10) // 坐标系速度
<code>#define COLLECT_ADDRESS_LASER_OUTPUT</code>	(11) // 激光输出(补偿前)
<code>#define COLLECT_ADDRESS_LASER_GATE</code>	(12) // 激光gate信号状态
<code>#define COLLECT_ADDRESS_LASER_POWER</code>	(13) // 激光能量当前输出值


```
#define COLLECT_ADDRESS_DI          (14) // 数字量输入状态

#define COLLECT_ADDRESS_DO          (15) // 数字量输出状态
```

(2) 配置采集数据通道, 需要配置对应结构体参数

[NMC ConfigCollect\(HAND devHandle, TCollectCfg *pCollect, TCollectTrig *pTrig \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pCollect	TCollectCfg *	输入	<pre>typedef struct{ short count; // 需要采集的变量个数 short interval; // 采集的间隔时间, 0表示每隔毫米采集一次数据, 1表示每隔ms... unsigned long address[COLLECT_LIST_MAX]; // 变量的地址 short length[COLLECT_LIST_MAX]; // 每个变量的长度(单位: char) }TCollectCfg;</pre>
pTrig	TCollectTrig *	输入	<pre>typedef struct{ short mode; // 触发模式, short source1; // 触发源 short source2; // 触发源 short startDelay; // 触发启动的延时 double value; // 触发比较值 }TCollectTrig;</pre>

(3) 启动或停止数据采集

[NMC CollectOnOff\(HAND devHandle, short en\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
en	short	输入	1 启动 0 停止

(4) 读取采集状态

[NMC_GetCollectSts\(HAND devHandle, short *pSts, long *pDataLen \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
pSts	short *	输出	采集状态, 按位表示各自状态
pDataLen	long *	输出	采集的数据量

(5) 读取采集数据

[NMC_GetCollectData\(HAND devHandle, short len, unsigned char *pData \);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄
len	short	输出	采集数据长度(单位: char, 一次最多读取 1440 字节)
pData	unsigned char *	输出	采集的数据(均以 char 为单元存储)

(6) 清除采集状态

[NMC_ClearCollectSts\(HAND devHandle\);](#)

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	控制器句柄

10.4.3 综合示例

```

short gc_example(void)
{
    /***** 已省略打开控制器部分, 统一描述在第五章1 *****/
    unsigned long addr[8];
    short len[8];
    rtn = NMC_CollectOnOff(devHandle, 0);           //关闭采集
    gc_rtn_error(rtn);
    rtn = NMC_ClearCollectSts(devHandle);          //清除采集状态
    gc_rtn_error(rtn);
    //读取采集变量地址, 采集轴规划位置和命令速度
    rtn = NMC_GetCollectDataAddr(devHandle, 0, COLLECT_ADDRESS_PRF_POS, &addr[0], &len[0]);
}

```

```
gc_rtn_error(rtn);
rtn = NMC_GetCollectDataAddr(devHandle, 0, COLLECT_ADDRESS_CMD_VEL, &addr[1], &len[1]);
gc_rtn_error(rtn);
TCollectCfg g_cltCfg;
memset(&g_cltCfg, 0, sizeof(TCollectCfg));
//采集两个数据, 间隔ms
g_cltCfg.count = 2;
g_cltCfg.interval = 0;
g_cltCfg.address[0] = addr[0];
g_cltCfg.length[0] = len[0];
g_cltCfg.address[1] = addr[1];
g_cltCfg.length[1] = len[1];
TCollectTrig cltTrig;
memset(&cltTrig, 0, sizeof(TCollectTrig));
cltTrig.mode = COLLECT_MODE_NONE;
rtn = NMC_ConfigCollect(devHandle, &g_cltCfg, &cltTrig); //采集模式无条件采集
gc_rtn_error(rtn);
rtn = NMC_CollectOnOff(devHandle, 1); // 开始采集
gc_rtn_error(rtn);
rtn = NMC_MtMovePtpAbs(axisHandle, 1, 1, 0, 0, 10, 0, 10000); //启用一个PTP运动
gc_rtn_error(rtn);
short cltSts;
long dataLen;
short axisSts;
short getLen;
double data[8192];
while (1)
{
    rtn = NMC_MtGetSts(axisHandle, &axisSts);
    gc_rtn_error(rtn);
    if ((axisSts & BIT_AXIS_BUSY) != BIT_AXIS_BUSY) { break; }
    _sleep(10);
}
rtn = NMC_CollectOnOff(devHandle, 0); // 运动停止关闭采集
gc_rtn_error(rtn);
getLen = 0;
rtn = NMC_GetCollectSts(devHandle, &cltSts, &dataLen); //读取采集的数据量
gc_rtn_error(rtn);
short count = dataLen / 1440;
//一次最多读取个字节超过需要分次读取
for (int i = 0; i < count; i++)
{
    rtn = NMC_GetCollectData(devHandle, 1440, (unsigned char*)(data + i * 180));
    gc_rtn_error(rtn);
}
```

```

}

rtn = NMC_GetCollectData(devHandle, dataLen%1440, (unsigned char*)(data+ count * 180));
gc_rtn_error(rtn);
return 0;
}

```

10.5 坐标系变换

10.5.1 功能介绍

目前控制器支持以下三种坐标系变换功能：

坐标系变换模式	说明
旋转转换	用户数据是按照直角坐标描述, 实际加工在一个旋转面上加工, 可以用此功能
极坐标转换	用户数据是按照直角坐标描述, 实际机械机构是一个旋转轴和一个进给轴
XYZA 机型转换	XYZ 为正常直角坐标系, 工具末端与 A 旋转中心不一致

10.5.2 指令说明

(1) 使能旋转转换处理

[NMC_CrdSetTransRotate\(HAND crdHandle, short rotAxisNo, double angleRadEqual, long firstAxisInitPos, long secAxisInitPos\)](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
rotAxisNo	short	输入	旋转轴轴号, 取值[0, N]
angleRadEqual	double	输入	旋转轴脉冲转弧度系数, 单位每个脉冲对应的弧度值
firstAxisInitPos	long	输入	旋转中心, X 轴的位置
secAxisInitPos	long	输入	旋转中心, Y 轴的位置

(2) 关闭旋转转换处理

[NMC_CrdDelTransRotate\(HAND crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(3) 使能极坐标转换处理

[NMC_CrdSetTransPolar\(HAND crdHandle, short rotAxNo, short transAxNo, double rotEquiv\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
rotAxNo	short	输入	旋转轴轴号, 取值[0, N]
transAxNo	short	输入	平移轴轴号, 取值[0, N]
rotEquiv	double	输入	旋转轴当量, 2PI/电机一圈脉冲数

(4) 运行至设定的极坐标位置并且进行圈数清零处理(利用单轴 PTP 运行到指定位置)

[NMC_CrdRunToPolarPos\(HAND crdHandle, double xPos, double yPos, double rotVel, double transVel\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
xPos	double	输入	极坐标系下 X 方向位置
yPos	double	输入	极坐标系下 Y 方向位置, 比如运动到 (5000, 5000) 的位置, 则相当于旋转轴转到 45 度角, 进给轴运动到根号 2*5000 的位置
rotVel	double	输入	旋转速度
transVel	double	输入	进给速度

(5) 运行至设定的极坐标角度位置(利用单轴 PTP 运行到指定位置)

[NMC_CrdRunToPolarTheta\(HAND crdHandle, double theta, double vel, short clrRoundFlag\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

theta	double	输入	旋转轴目标角度, 单位弧度
vel	double	输入	旋转速度
clrRoundFlag	short	输入	是否清除圈数

(6) 销毁极坐标机型 (只恢复直角坐标系)

[NMC_CrdDelTransPolar\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(7) XYZA 机型设置接口

[NMC_CrdSetTransXYZA\(HAND_crdHandle, short *pAxisMapArray \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pAxisMapArray	short *	输入	pAxisMapArray[0]~[3] 分别对应 X、Y、Z 和 A

(8) 销毁 XYZA 机型, 回归 XYZ 结构

[NMC_CrdDelTransXYZA\(HAND_crdHandle\);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄

(9) 求工具参数

[NMC_CrdSetXYZAToolCalc\(HAND_crdHandle, double_deltaTheta, long_deltaX, long_deltaY, long *pToolX, long *pToolY \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
deltaTheta	double	输入	从 0° 往正方向旋转过的角度值, 单位弧度
deltaX	long	输入	旋转 deltaTheta 角度后, 重新校正到同一点位置后, X 轴的相对移动量
deltaY	long	输入	旋转 deltaTheta 角度后, 重新校正到同一

			点位置后，Y 轴的相对移动量
pToolX	long *	输出	返回工具的 X 值
pToolY	long *	输出	返回工具的 Y 值

(10) 设置 XYZA 机型参数

[NMC_CrdSetXYZAPara\(HAND crdHandle, double pulse2Rad, long toolX, long toolY \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pulse2Rad	double	输入	A 轴的每个脉冲对应的弧度值
toolX	long	输入	工具的 X 值
toolY	long	输入	工具的 Y 值

(11) 读取 XYZA 机型参数

[NMC_CrdGetXYZAPara\(HAND crdHandle, double *pPulse2Rad, long* ToolX, long *pToolY \);](#)

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
pPulse2Rad	double *	输出	返回 A 轴的每个脉冲对应的弧度值
pToolX	long *	输出	返回工具的 X 值
pToolY	long *	输出	返回工具的 Y 值

10.5.3 综合示例

a. 旋转变换例程

```

short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    /***** 已省略建立坐标系(插补)过程 *****/
    short rtn = 0;
    double mmToPluse = 1000;
    double endVel, vel, acc, rotVel;
    endVel = 0;
    vel = 10;
    acc = 1;
}

```

```
rotVel = 3;
double m_l = 100; //圆角矩形的长为mm
double m_w = 60; //圆角矩形的宽为mm
double m_r = 5; //圆角矩形的倒角半径为mm
//旋转轴为轴, 旋转一圈脉冲数为, 旋转中心为(, 15000)
rtn = NMC_CrdSetTransRotate(crdHandle, 3, (2 * 3.1415926) / 10000, 0, 15000);
gc_rtn_error(rtn);
rtn = NMC_CrdBufClr(crdHandle); //指令缓冲区清空
gc_rtn_error(rtn);
long rotPos = -2500;
long posArray[4] = { 0, 0, 0, 0 };
long segNo = 0;
//插补轨迹第一条得指定旋转变换的轴, 这条跟随位移是
rtn = NMC_CrdBufAxMoveRel(crdHandle, segNo++, 8, posArray, 0, 1);
gc_rtn_error(rtn);
//运动到原点
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 15, posArray, endVel, vel, acc, 0);
gc_rtn_error(rtn);
//缓冲区单轴移动参数设置
rtn = NMC_CrdBufSetPtpMovePara(crdHandle, segNo++, 3, rotVel, 1, 0);
gc_rtn_error(rtn);
posArray[0] = (0.5 * m_l - m_r) * mmToPluse;
// 直线插补
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 3, posArray, endVel, vel, acc, 0);
gc_rtn_error(rtn);
//缓冲区单轴移动(相对位移移动)
rtn = NMC_CrdBufAxMoveRel(crdHandle, segNo++, 8, &rotPos, 0, 1);
gc_rtn_error(rtn);
posArray[0] = 10000;
posArray[1] = 0;
// XY平面圆弧插补: 终点位置、半径、方向
rtn=NMC_CrdArcRadiusEx(crdHandle, segNo++, posArray, 5000, 1, endVel, vel, acc, 0);
gc_rtn_error(rtn);
posArray[0] = (0.5 * m_l) * mmToPluse;
posArray[1] = (m_w - m_r) * mmToPluse;
// 直线插补
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 3, posArray, endVel, vel, acc, 0);
gc_rtn_error(rtn);
//缓冲区单轴移动(相对位移移动)
rtn = NMC_CrdBufAxMoveRel(crdHandle, segNo++, 8, &rotPos, 0, 1);
gc_rtn_error(rtn);
posArray[0] = (0.5 * m_l - m_r) * mmToPluse;
posArray[1] = (m_w)*mmToPluse;
// XY平面圆弧插补: 终点位置、半径、方向
```



```

rtn=NMC_CrdArcRadiusEx(crdHandle, segNo++, posArray, (m_r)*mmToPluse, 1, endVel, vel, acc, 0);
gc_rtn_error(rtn);
posArray[0] = (-0.5 * m_l + m_r) * mmToPluse;
posArray[1] = (m_w)*mmToPluse;
// 直线插补
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 3, posArray, endVel, vel, acc, 0);
gc_rtn_error(rtn);
// 缓冲区单轴移动(相对位移移动)
rtn = NMC_CrdBufAxMoveRel(crdHandle, segNo++, 8, &rotPos, 0, 1);
gc_rtn_error(rtn);
posArray[0] = (-0.5 * m_l) * mmToPluse;
posArray[1] = (m_w - m_r) * mmToPluse;
//XY平面圆弧插补: 终点位置、半径、方向
rtn=NMC_CrdArcRadiusEx(crdHandle, segNo++, posArray, (m_r)*mmToPluse, 1, endVel, vel, acc, 0);
gc_rtn_error(rtn);
posArray[0] = (-0.5 * m_l) * mmToPluse;
posArray[1] = (m_r)*mmToPluse;
// 直线插补
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 3, posArray, endVel, vel, acc, 0);
gc_rtn_error(rtn);
//缓冲区单轴移动(相对位移移动)
rtn = NMC_CrdBufAxMoveRel(crdHandle, segNo++, 8, &rotPos, 0, 1);
gc_rtn_error(rtn);
posArray[0] = (-0.5 * m_l + m_r) * mmToPluse;
posArray[1] = 0;
// XY平面圆弧插补: 终点位置、半径、方向
rtn=NMC_CrdArcRadiusEx(crdHandle, segNo++, posArray, (m_r)*mmToPluse, 1, endVel, vel, acc, 0);
gc_rtn_error(rtn);
posArray[0] = 0;
posArray[1] = 0;
rtn = NMC_CrdLineXYZEx(crdHandle, segNo++, 3, posArray, endVel, vel, acc, 0); //直线插补
gc_rtn_error(rtn);
rtn = NMC_CrdEndMtn(crdHandle); //结束缓冲区
gc_rtn_error(rtn);
rtn = NMC_CrdStartMtn(crdHandle); //启动缓冲区插补
gc_rtn_error(rtn);
return 0;
}

```

b. 极坐标变换例程(略)

c. XYZA 机型变换例程

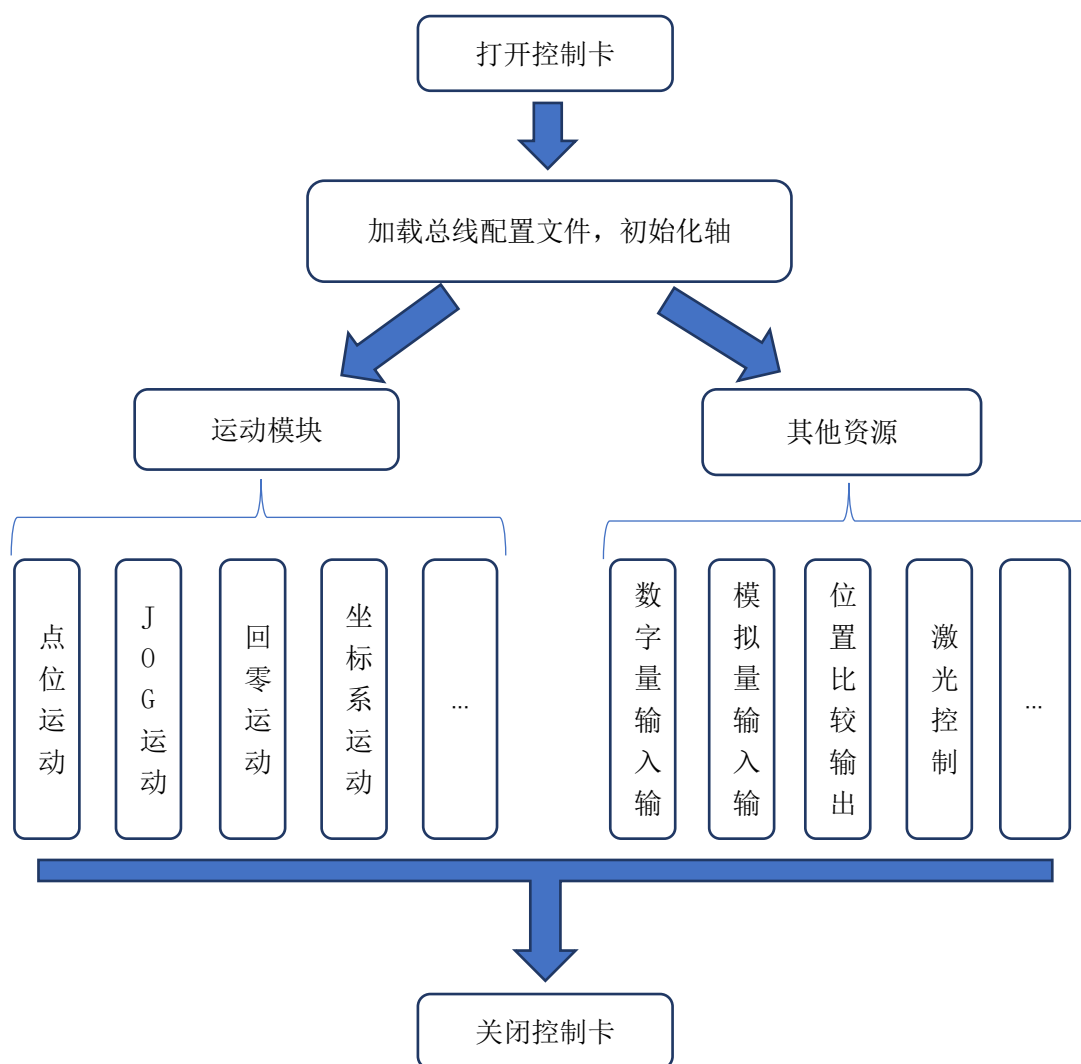
```
#define PI 3.14159
```

```
short gc_example(void)
{
    /***** 已省略打开控制器部分，统一描述在第五章1 *****/
    /***** 先建立坐标系(插补), 过程略 *****/
    double pulse2Rad = 2 * PI / 288000;          //A轴旋转一圈脉冲为
    long deltaX = -3687;                          //0转到PI/3后重新校正到一点是的X偏移量
    long deltaY = -9494;                          //0转到PI/3后重新校正到一点是的Y偏移量
    long toolX = 0;
    long toolY = 0;
    //计算工具偏移
    rtn = NMC_CrdSetXYZAToolCalc(crdHandle, PI / 3, deltaX, deltaY, &toolX, &toolY);
    gc_rtn_error(rtn);
    rtn = NMC_CrdSetXYZAPara(crdHandle, pulse2Rad, toolX, toolY); //设置工具参数
    gc_rtn_error(rtn);
    short pMapAxisArray[4] = { 0, 1, 2, 3 };
    rtn = NMC_CrdSetTransXYZA(crdHandle, pMapAxisArray);          //设置XYZA机型
    gc_rtn_error(rtn);
    return 0;
}
```

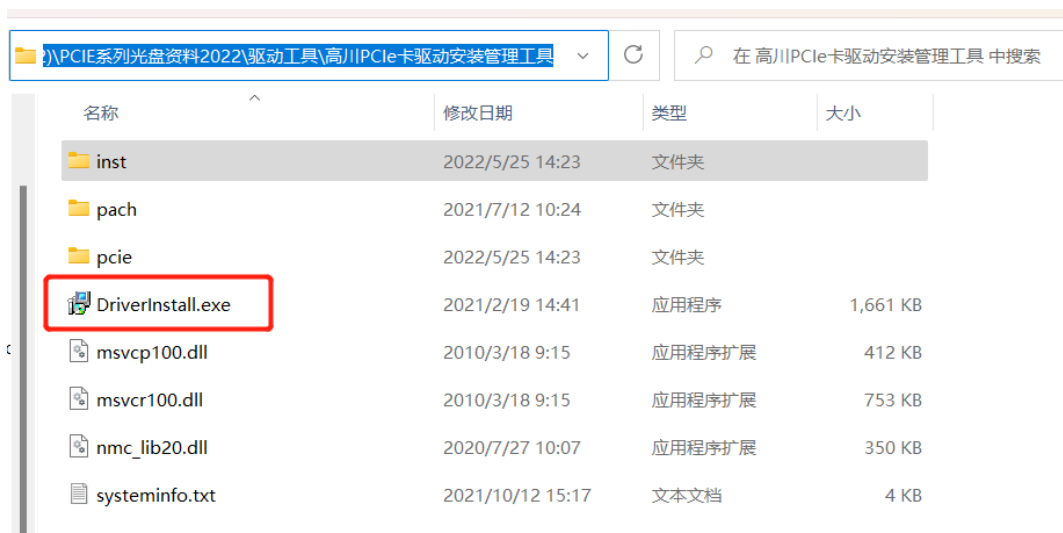
第四章 动态库的使用

GCN/GC/GCS/GCE 系列控制器的应用程序开发可使用多种编程语言，如 C++、C#、VB.NET、LabView 等，我们提供了丰富的函数接口和功能模块，建议选择熟悉的语言开发。

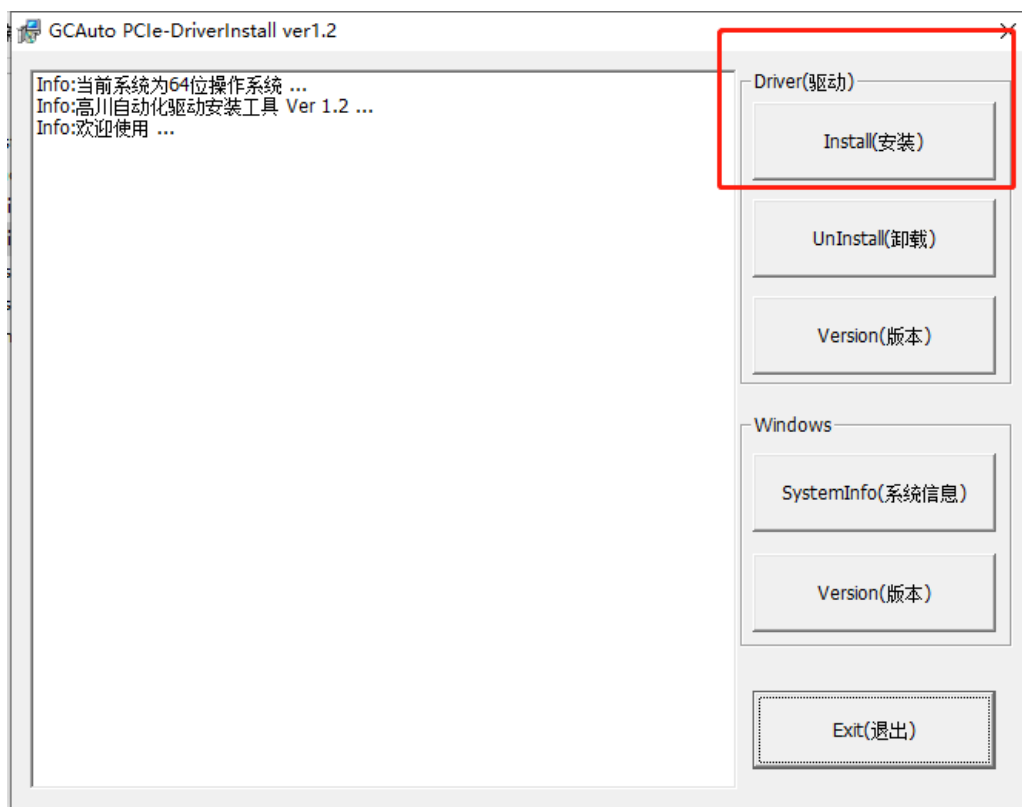
GCN/GC/GCS/GCE 系列控制器的使用，参考以下思路，将有助于用户对我们运动控制卡的整体把握和深入了解。



GCS/GC/GCE 运动控制卡需要安装驱动，安装工具在光盘资料；如果已经安装过早期的驱动文件，最好在设备管理器里面找到 PICE 卡的驱动卸载并删除，然后用下图所示工具 DriverInstall.exe 安装。


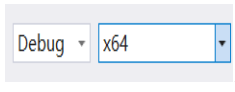


点击 **Install(安装)**，即可完成安装，若失败，请查看“产品手册”中《GC 常用功能手册》；



1 C++

(1) 将对应的控制器文件：光盘资料\动态库\C++和 dll 拷贝到工

程目录下； 选择**_x86 文件， 选择**_x64 文件；

(2) 在源文件中包含头文件；

(3) 设置 nmc_lib20.lib 为引用库；

(4) 调用 NMC 指令，开始应用开发；

请参考下面的例子：

```
// 包含头文件(请根据实际路径修改)
#include "mtn_lib20.h"
#include "mtn_lib20_ext.h"
#include "mtn_lib20_err.h"
#pragma comment(lib, "nmc_lib20.lib") // 引用库
short gc_example(void)
{
    short rtn;           // 指令返回值
    short devNum;        // 设备序号, 从0开始
    short axisNum;       // 轴号, 从0开始
    HAND devHandle;      // 设备句柄
    HAND axisHandle;     // 轴句柄
    HAND crdHandle;      // 坐标系句柄
    TDevInfo devList[4]; // 设备信息
    rtn = NMC_DevSearch(Ethernet, (unsigned short*)&devNum, devList); // 搜索控制器
    gc_rtn_error(rtn);
    if (devNum < 0) { return 1; } // 没有找到控制器
    rtn = NMC_DevOpen(devNum, &devHandle); // 打开控制器
    gc_rtn_error(rtn);
    rtn = NMC_MtOpen(devHandle, axisNum, &axisHandle); // 打开轴, 并获得轴句柄
    gc_rtn_error(rtn);
    rtn = NMC_MtSetSvOn(axisHandle); // 打开轴使能(根据实际情况使用)
    gc_rtn_error(rtn);
    //rtn = NMC_DevReset(devHandle); // 复位控制器
    //gc_rtn_error(rtn);
    return 0;
}
```

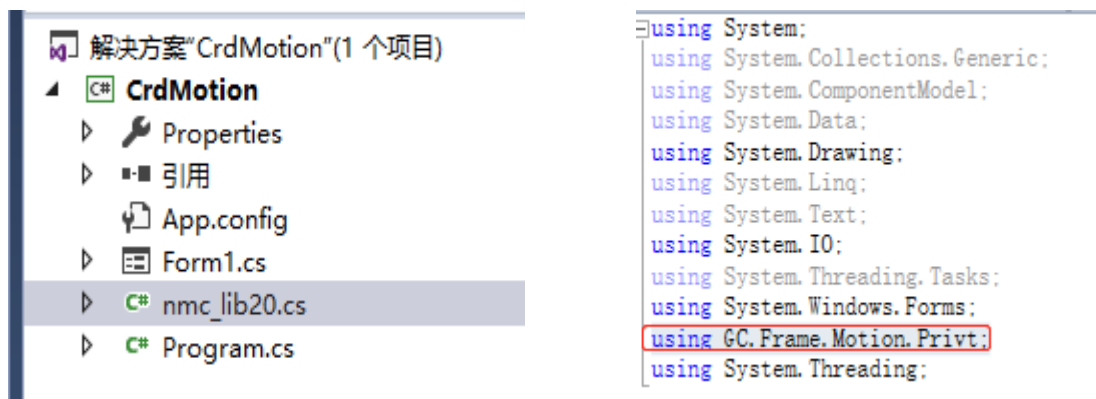
注意：详细源码请看 光盘资料\例程\C++；

2 C#

(1) 将对应的控制器文件：光盘资料\动态库\C#和 dll 拷贝到工

程目录下；
Debug x86 选择**_x86 文件， Debug x64 选择**_x64 文件；

(2) 在项目解决方案管理器中选中项目，右键->添加现有项，将 nmc_lib20.cs 添加进来，同时在.cs 文件中 using 命令空间，如下



(3) 调用 NMC_指令即可开发(c#示例可以参考光盘目录里面的 c#例程)

```
private void button1_Click(object sender, EventArgs e)
{
    short rtn = 0; // 指令返回值
    ushort devHandle = 0; // 控制器句柄
    ushort[] axisHandle = new ushort[4]; // 轴句柄
    //打开控制器，根据控制器名称打开，控制器名称可用我司测试软件GCS读取
    rtn = CNMCLib20.NMC_DevOpenByID("CARD1", ref devHandle);
    rtn = CNMCLib20.NMC_LoadConfigFromFile(devHandle,
    System.Text.ASCIIEncoding.Default.GetBytes("GCN400.CFG")); //加载配置文件
    for (short i = 0; i < 4; i++)
    {
        rtn = CNMCLib20.NMC_MtOpen(devHandle, i, ref axisHandle[i]); //打开各轴
        rtn = CNMCLib20.NMC_MtClrError(axisHandle[i]); //清除各轴状态
        rtn = CNMCLib20.NMC_MtZeroPos(axisHandle[i]); //位置清零
        rtn = CNMCLib20.NMC_MtSetSvOn(axisHandle[i]); //伺服使能
    }
    //rtn = CNMCLib20.NMC_DevReset(devHandle); //复位一下
    //rtn 指令返回值请自行判断
}
```

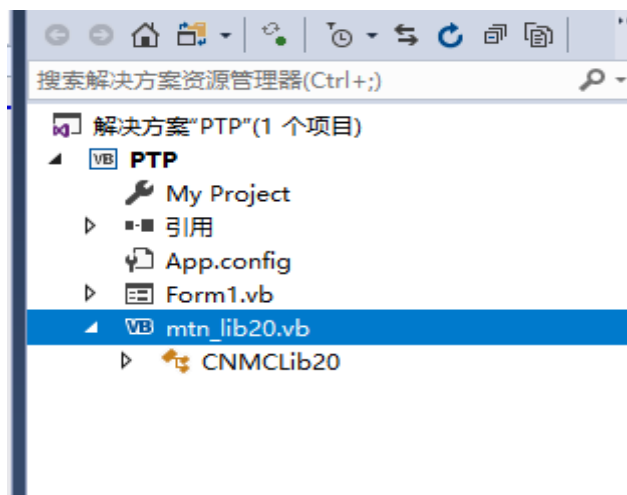
注意：详细源码请看 光盘资料\例程\C#；

3 VB.NET

(1) 将对应的控制器文件：光盘资料\动态库\VB.NET 和 d11 拷贝到工

程目录下；
Debug x86 选择**_x86 文件， Debug x64 选择**_x64 文件；

(2) 在项目解决方案管理器中选中项目，右键->添加现有项，将 nmc_lib20.vb 添加进来。



(3) 调用 NMC_指令即可开发，例如



```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles  
Button1.Click  
Dim rtn As Short = 0  
' 创建控制器句柄  
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_DevOpenByID("CARD1", devHandle)  
SendErrorMsg("NMC_DevOpenByID", rtn)  
' 加载控制器配置  
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_LoadConfigFromFile(devHandle,  
System.Text.ASCIIEncoding.Default.GetBytes("GCN400.cfg"))  
SendErrorMsg("NMC_LoadConfigFromFile", rtn)
```

```
' 使能轴 1
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtOpen(devHandle, 0, axishandle) ' 打开
轴 1
SendErrorMsg("NMC_MtOpen", rtn)
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtClrError(axishandle) ' 清除轴状态
SendErrorMsg("NMC_MtClrError", rtn)
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtZeroPos(axishandle) ' 位置清零
SendErrorMsg("NMC_MtZeroPos", rtn)
' Timer1.Start()
End Sub
```

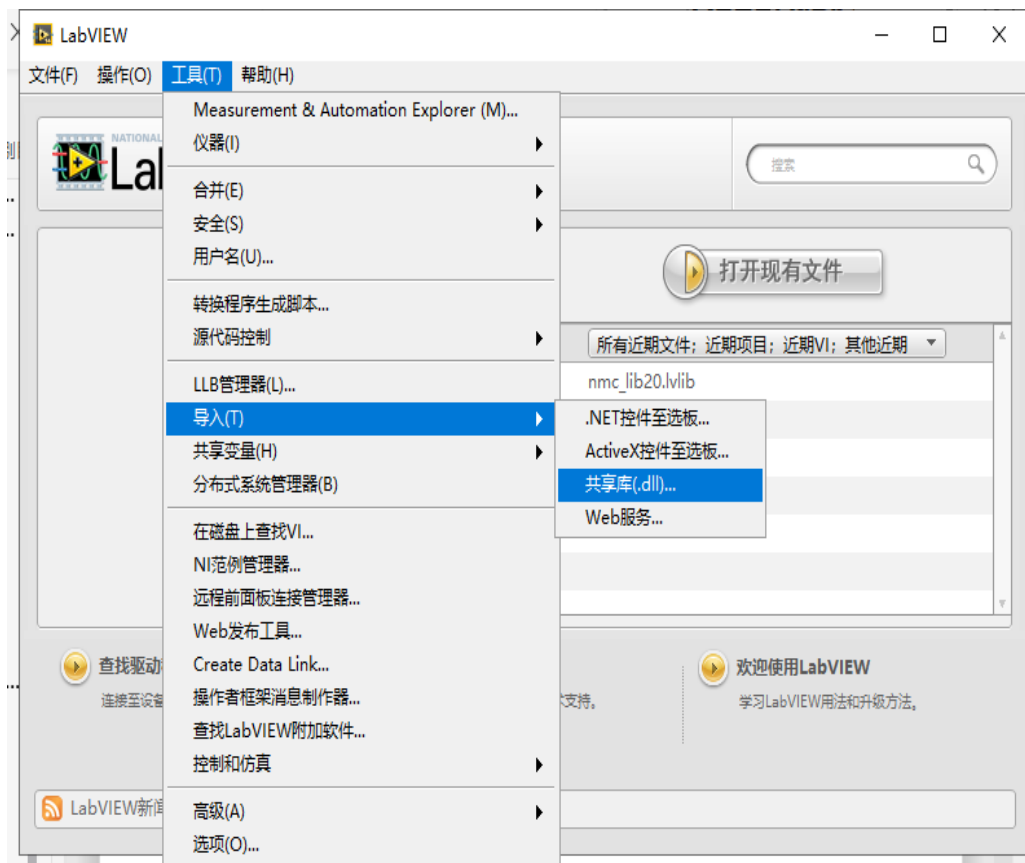
注意：详细源码请看 光盘资料\例程\VB.NET；GCN400.cfg 配置文件由 GCS 工具生成；

4 LABVIEW

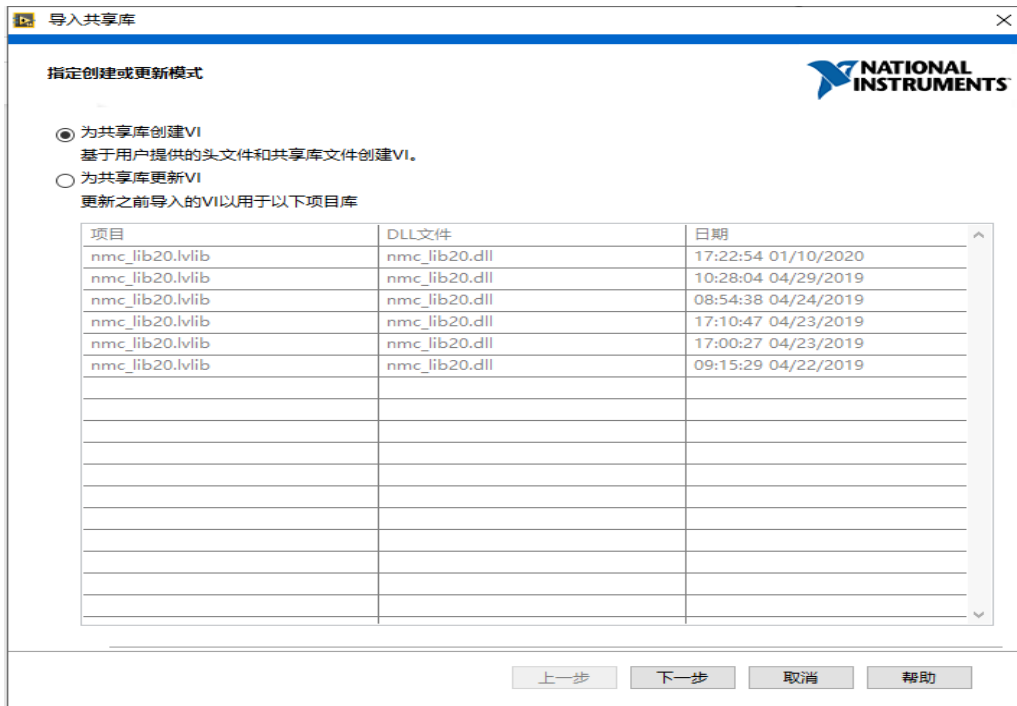
LABVIEW 调用库函数编程有多种方式：

- (1) 直接利用我司提供的 VIs 进行编程，具体可参考我司提供的例程：光盘资料\例程\LABVIEW；
- (2) 利用 LABVIEW 的导入工具，将我们提供的 “.h” 文件和 “.dll” 文件导入，自己生成 VIs, 示例如下：

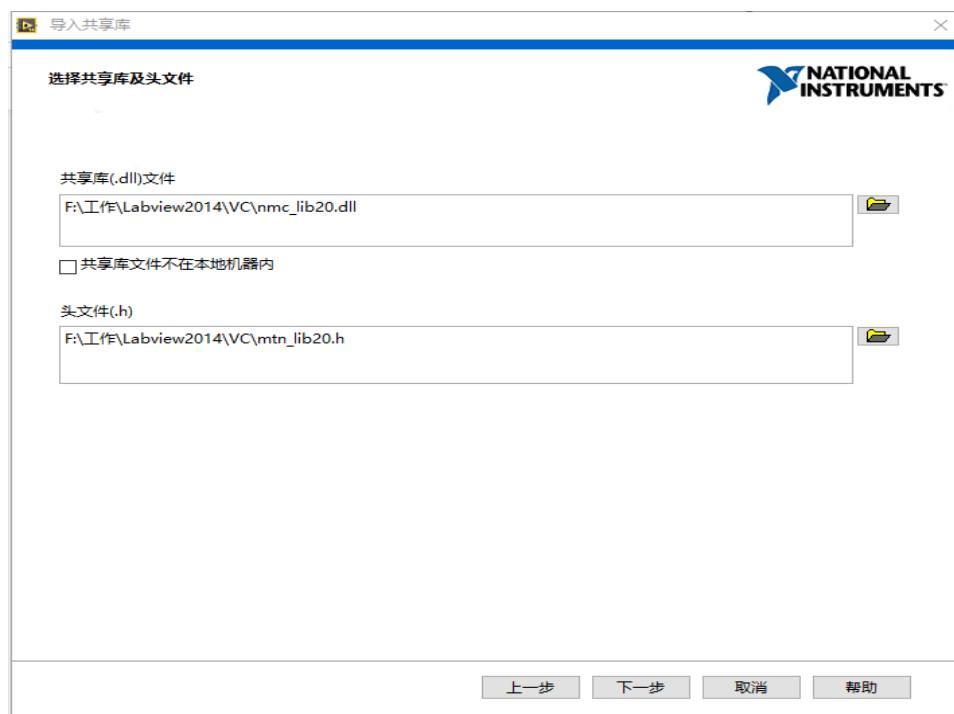
- ① 打开 LABVIEW，选择 工具->导入->dll 共享库；



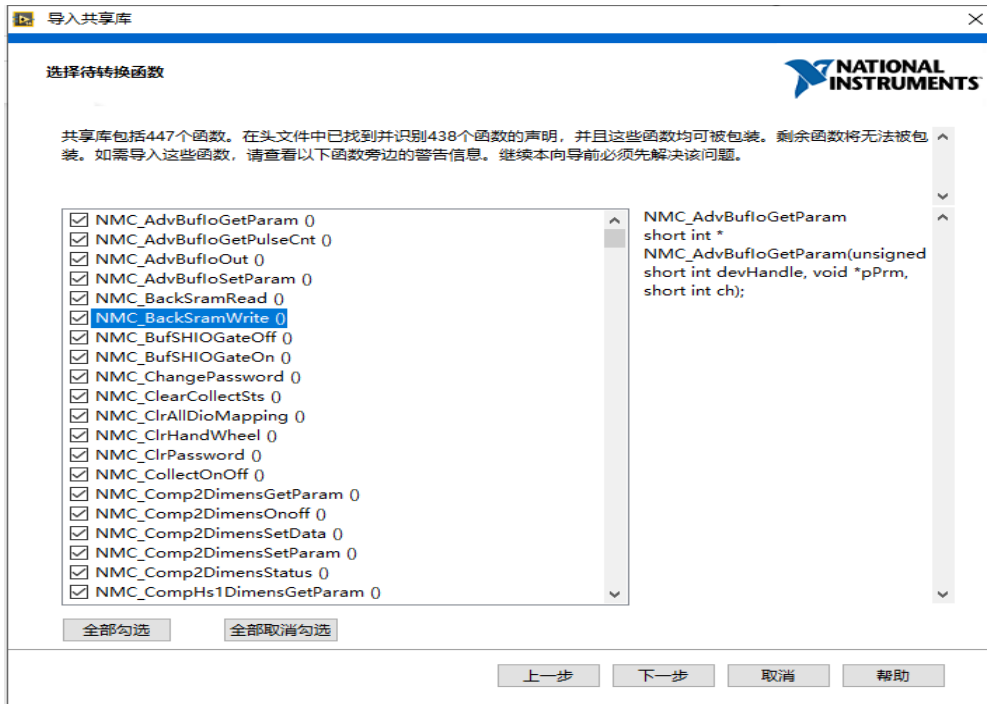
- ② 为共享库创建 VI->下一步。(如果之前导入过，也可以选择 为共享库更新 VI)



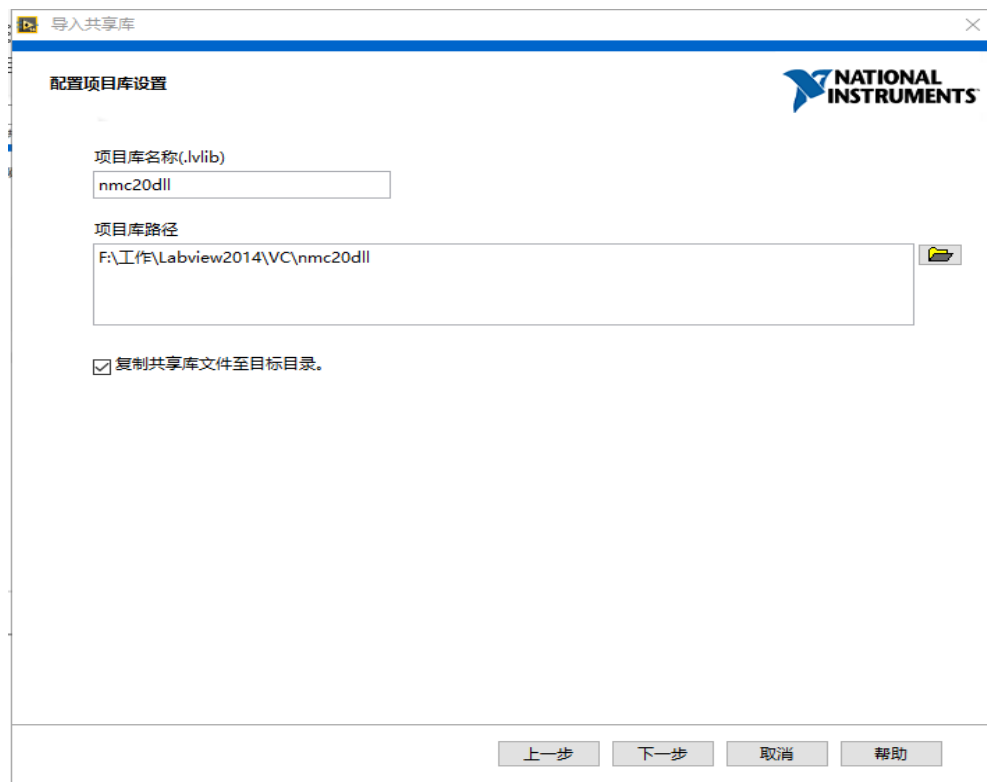
③ 分别选择我司提供的 dll 库和.h 头文件(注意：头文件已经整合了 mtn_lib20.h、mtn_lib20_oem.h、mtn_lib20err.h 和 mtn_lib20ext.h，此头文件在 LABVIEW 例程里面可以找到)；



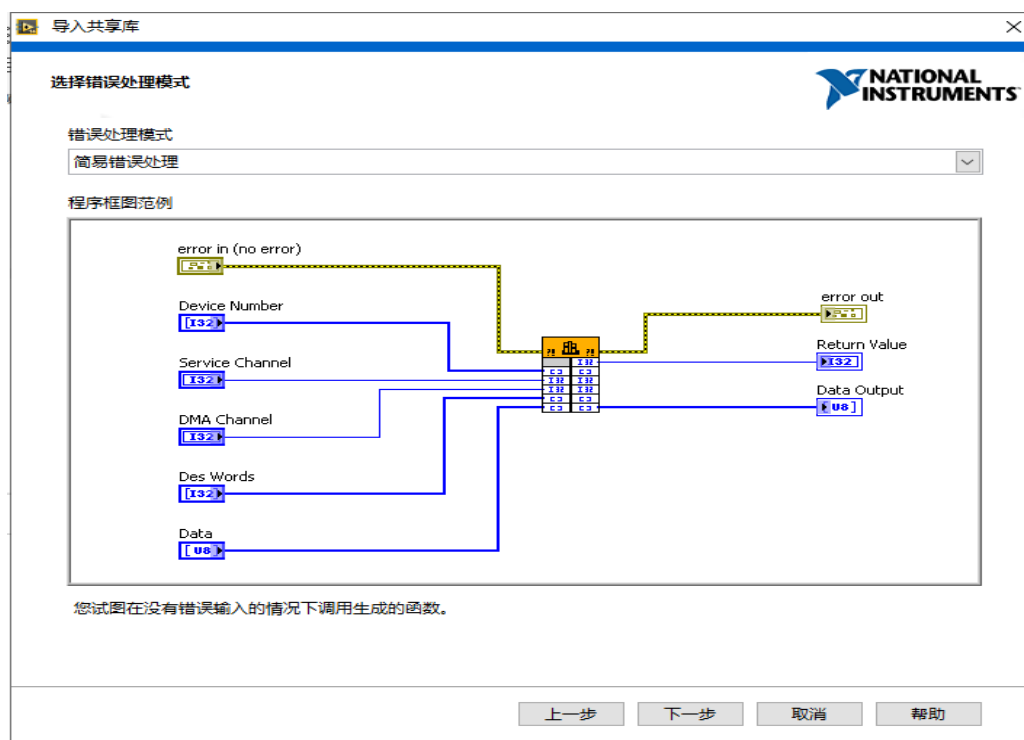
④ 经过一段时间的解析，可以从 DLL 中读取得到各个模块，见下表中排列的函数，继续下一步



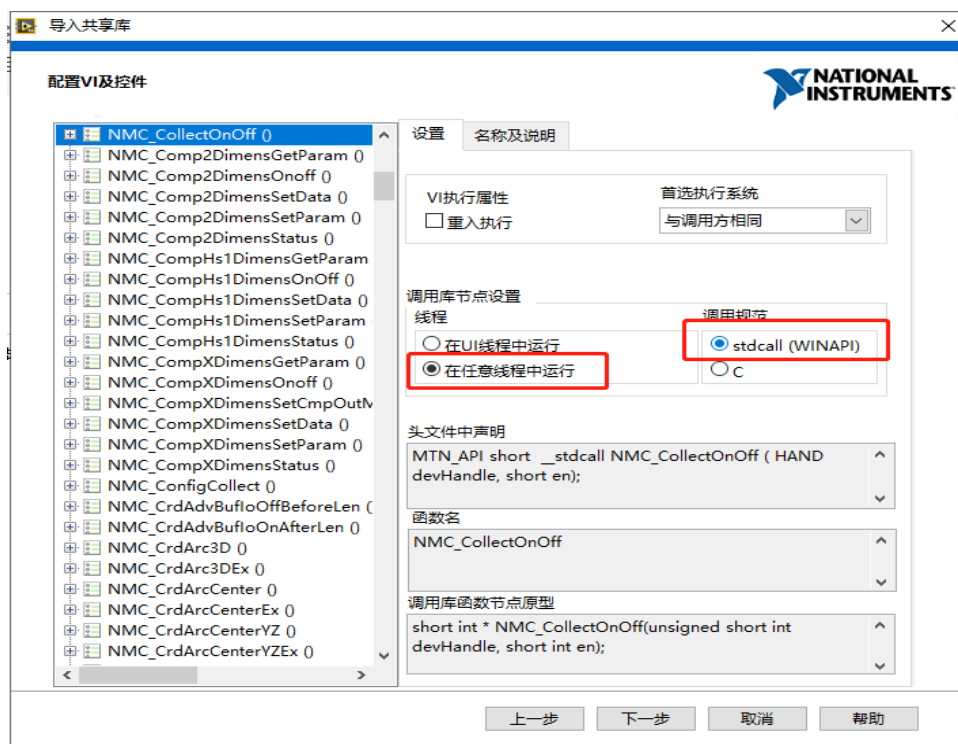
⑤ 填写导出名称和选择导出路径，下一步



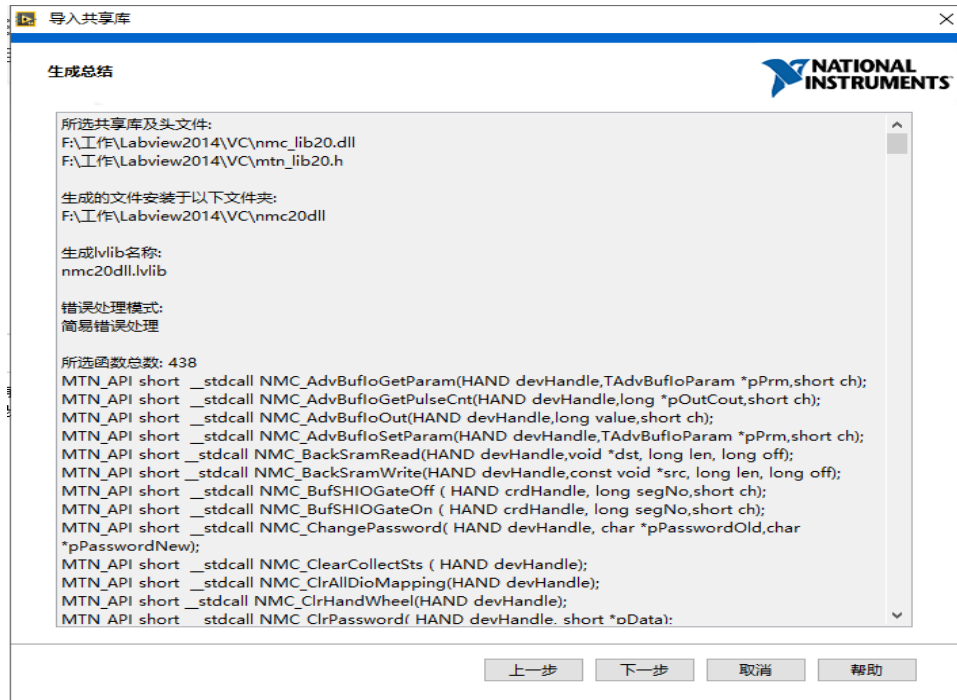
下一步；



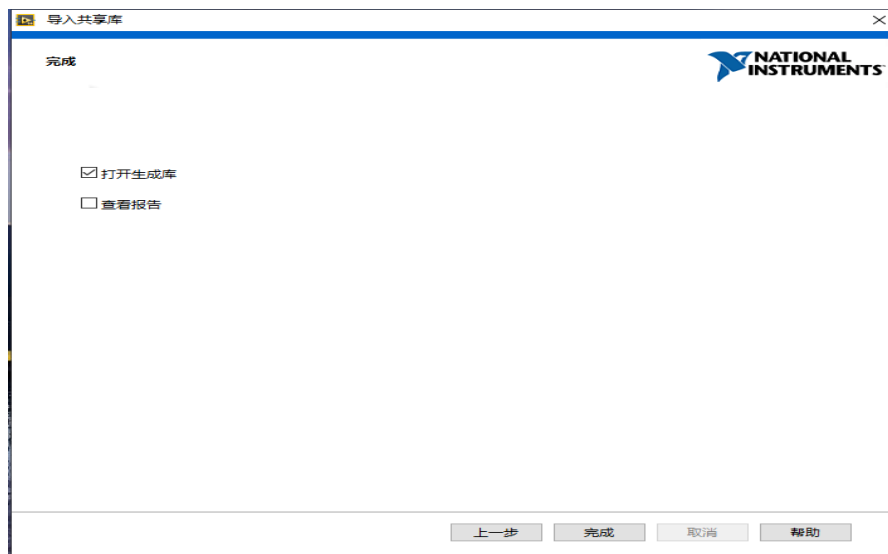
⑥ 为所有函数选择在任意线程调用和 stdcall (WINAPI)，然后下一步。(请不要只选择一个就直接下一步，否则未选择的函数会在调用时出错)

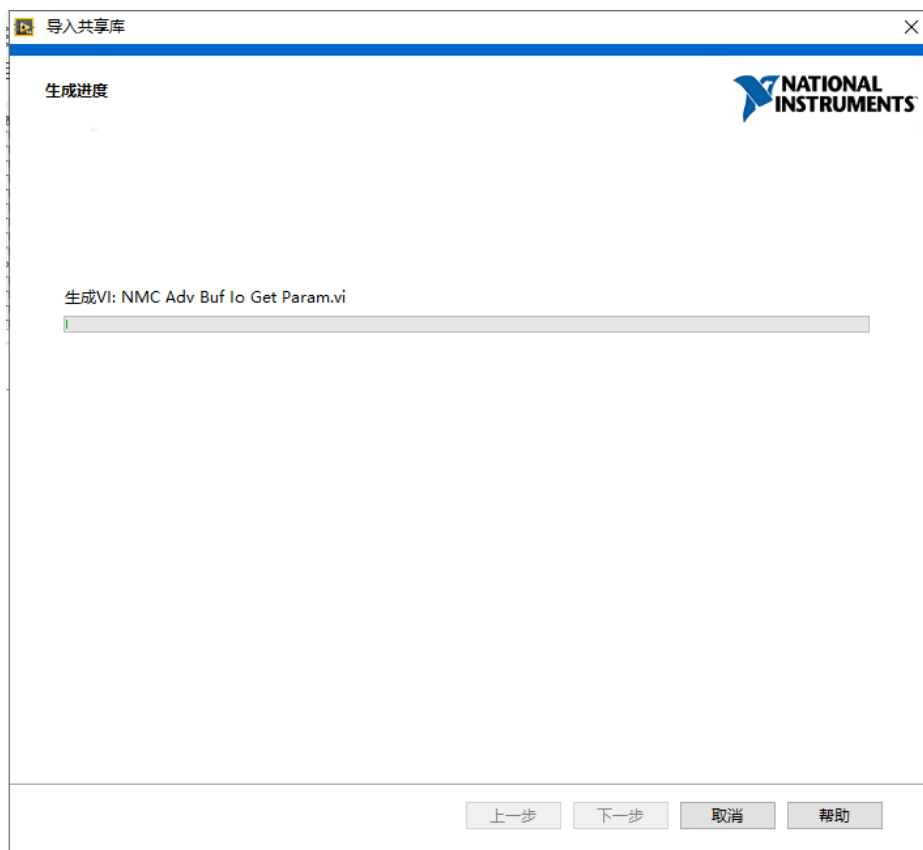


⑦ 下一步

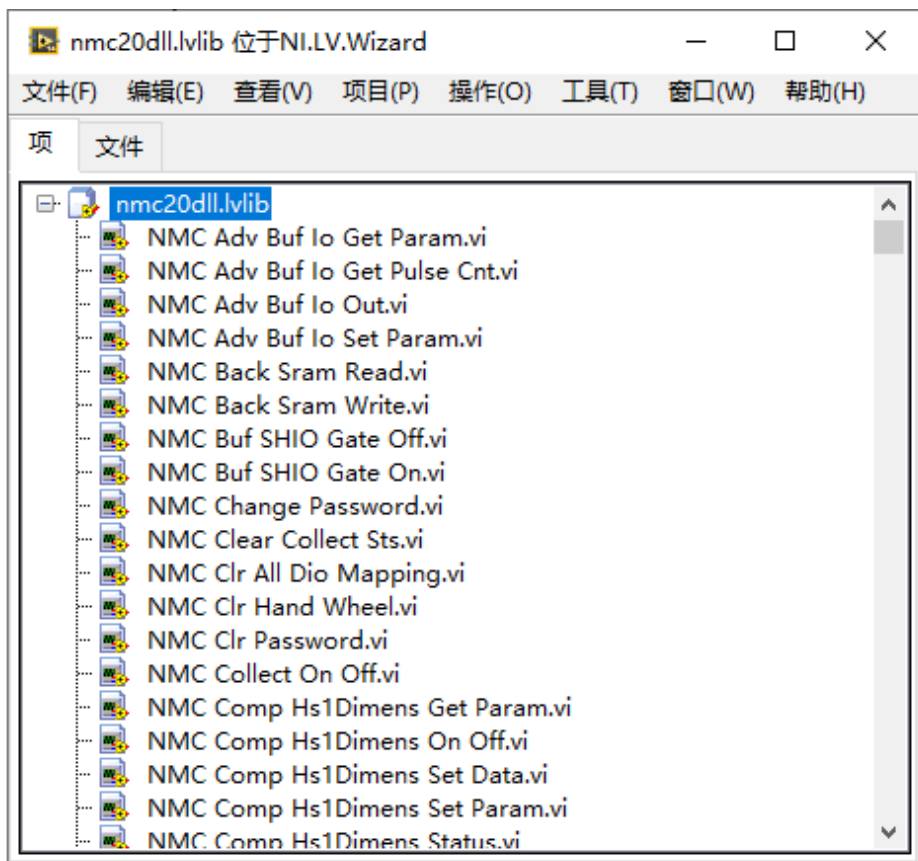


⑧ 等待生成完成，过程会占用时间，请耐心等待；

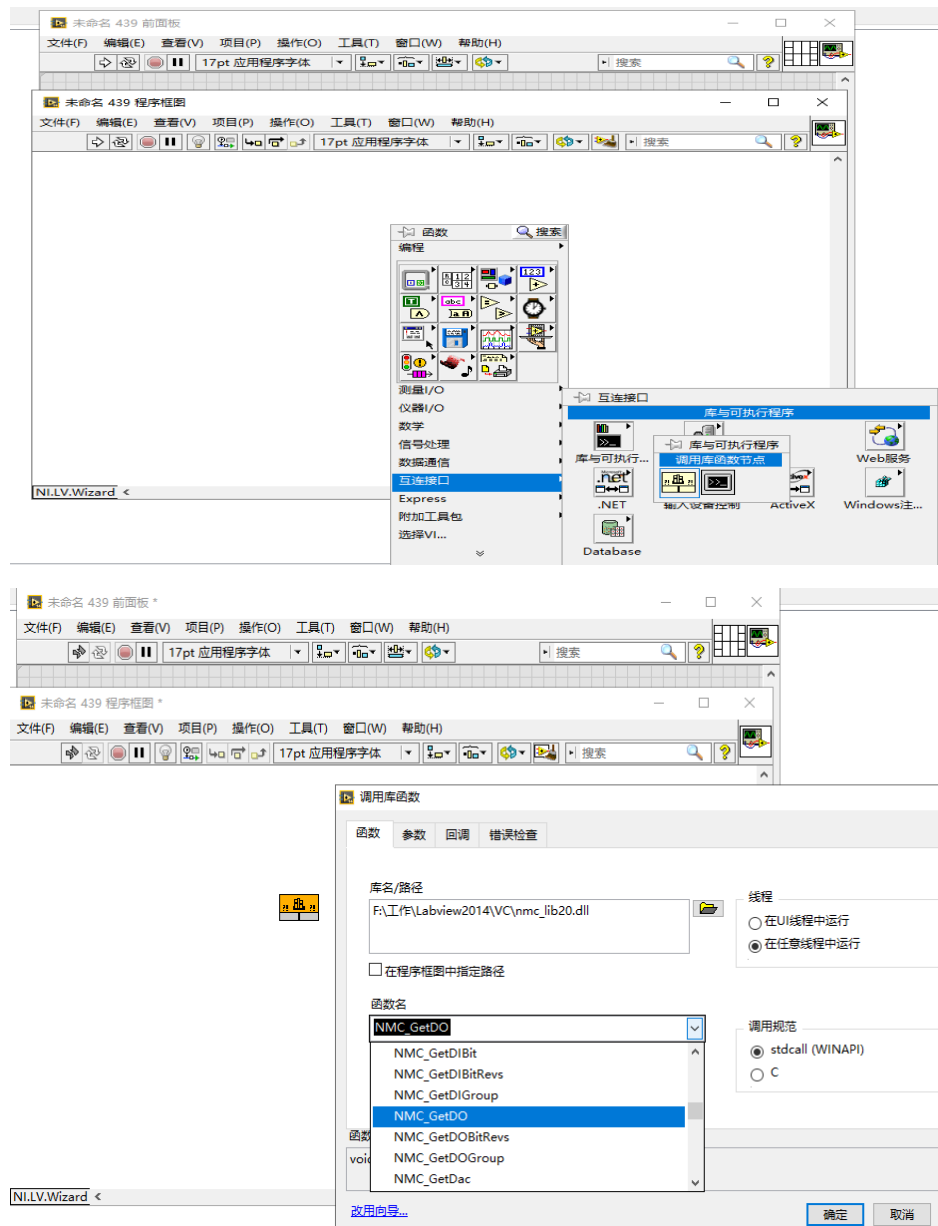


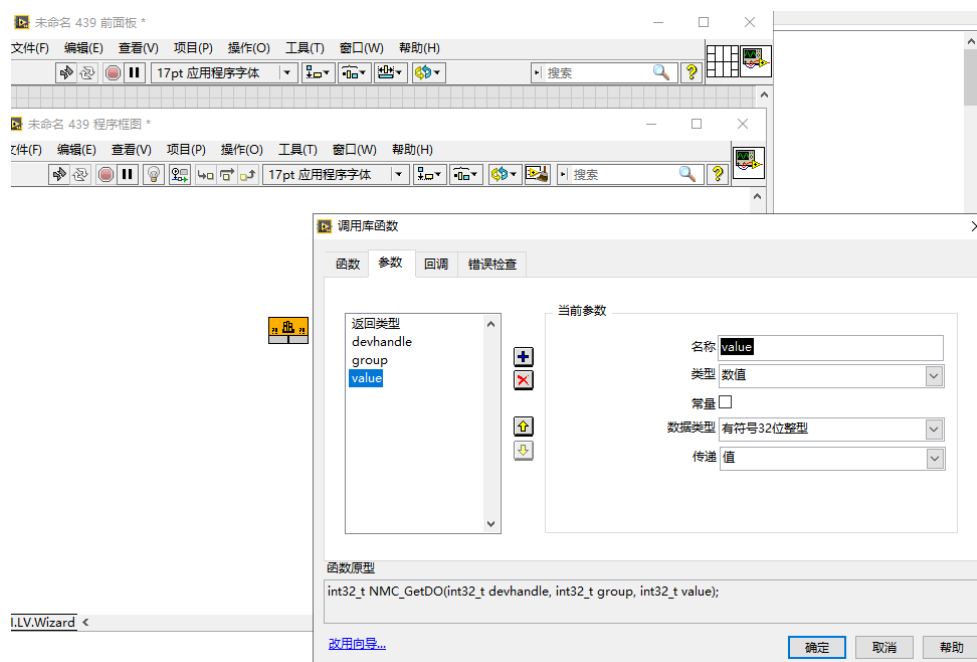


⑨ 至此就可以在项目中拖入使用了



(3)通过库函数调用节点调用，需要参照.h 文件为调用的函数添加参数，**数据类型**对照表可在网上进行搜索。





注意：详细源码请看 光盘资料\例程\LABVIEW；

第五章 注意事项

本章节主要是针对用户在使用控制器和指令时需要注意事项的总结，在编程控制器之前，可以优先阅读取本章节内容，以便于提前了解和认识我们的运动控制器。

1 关于NMC指令

- (1) 若无特别说明，API中的返回值RTN_CMD_SUCCESS (即0值)表示指令执行成功，其他表示错误，参考 mtn_lib20_err.h;
- (2) 若无特别说明，axisHandle表示轴句柄，devHandle表示控制器句柄，crdHandle表示坐标系句柄;
- (3) 若无特别说明，有关序号的则从0开始，如NMC_MtOpen中的itemNo参数，0表示第一个轴 (EtherCAT轴序号从16开始);
- (4) 若无特别说明，有关速度的单位为脉冲每毫秒 (pulse/ms)，加速度单位为脉冲每平方毫秒 (pulse/ms²)，位置单位为脉冲 (pulse);
- (5) 指令接口函数中，有三种编程对象，以 NMC_Mt 开头的单轴控制指令，以 NMC_Crd 开头的坐标系指令，其余是对控制器的指令;
- (6) 在API中，结构体定义中的所有reservedx成员都是保留参数，请不要修改他们;
- (7) 本手册的综合示例中，打开控制器部分，变量定义，指令返回值，设备句柄，轴句柄，坐标系句柄，设备序号，轴序号和指令返回值统一为以下定义，用户阅读取时请熟知;

```

short rtn;                // 指令返回值
short devNum;             // 设备序号，从0开始
short axisNum;            // 轴号，从0开始
HAND devHandle;           // 设备句柄
HAND axisHandle;          // 轴句柄
HAND crdHandle;           // 坐标系句柄

rtn = NMC_DevOpen(devNum, &devHandle);           //打开控制器
gc_rtn_error(rtn);
rtn = NMC_MtOpen(devHandle, axisNum, &axisHandle); //打开轴, 并获得轴句柄
gc_rtn_error(rtn);
rtn = NMC_MtSetSvOn(axisHandle);                 //打开轴使能(根据实际情况使用)
gc_rtn_error(rtn);

short gc_rtn_error(short rtn)
{
    if (rtn != RTN_CMD_SUCCESS) {
        return rtn;                // 指令执行错误
    }
    return rtn;
}

```

2 网络型控制器

- (1) 网络型控制器的 IP 地址不能设置成 169.254.x.x, 同时设置电脑端 IP 也不要包含有该地址;
- (2) 网络型控制器应与其他网络设备在不同网段, 如相机、激光器等;
- (3) 当使用多个网络型控制器时, 推荐使用后两种方式(NMC_DevOpenByID, NMC_DevOpenByIP)打开控制器, 此时不同的控制器需要设置不同的名称和不同网段的 IP;
- (4) 网络型控制器默认 IP 为 192.168.1.110, 使用多个控制器时, 应设为不同网段的 IP, 如 192.168.2.110, 192.168.3.110, 192.168.4.111 等, 设置完成后, 请重新接通电源; 当多个设备接入同一台工控机时, 尽量确保网络控制器使用唯一网段;
- (5) NMC 函数执行成功时, 函数返回值为 0, 不成功时返回值不等于 0, 此时可通过 GCS 工具里面的错误代码查询功能进行查询: GCS 工具->菜单栏->工具->错误代码查询;
- (6) 凡是对控制器信息做修改, 请重新上电控制器;

3 PCIE控制卡

安装驱动时请退出杀毒软件，全盘杀毒可能导致驱动被误删。两张或两张以上的控制卡在同一台工控机上运行时，驱动安装完成后，请电脑关机后再打开，以完成控制卡序号(0~N)的正确排序，避免指令(NMC_DevOpen)打开控制卡时序号错乱，靠近CPU的卡为第一张，建议使用ID号打开多张卡。凡是对控制卡信息做修改，请电脑关机后再开启。