

控制器编程手册



版权声明



© Copyright2016Shenzhen Gaochuan Industrial Automation Co., Ltd.
All Rights Reserved.

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，恕不另行通知。



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

联系方式

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: +86 755 23502680
Email: sales@gcauto.com.cn
Website: www.gcauto.com.cn

目录

版权声明.....	2
联系方式.....	2
第一章 指令汇总.....	9
1. 控制器的打开和基本操作.....	9
2. 控制器的基本配置(单轴).....	9
3. 控制器的状态检测.....	10
4. 点位和 JOG 运动.....	10
5. 回零运动.....	11
6. 坐标系的运动.....	11
6.1 坐标系的初始化.....	11
6.2 直线插补和平面圆弧插补.....	12
6.3 多维(8)插补指令.....	12
6.3 螺旋线插补.....	12
6.4 空间圆弧插补.....	13
6.5 缓存区的操作(I/O、延时、单轴跟随等).....	13
6.6 插补打包指令介绍.....	13
6.7 坐标系的状态检测.....	13
6.8 其他指令.....	14
7. I/O 资源访问.....	14
8. 手脉功能.....	15
9. 龙门功能.....	15
10. 位置比较输出.....	16
10.1 多维位置比较输出(软件比较).....	16
10.2 一维高速位置比较(硬件比较).....	16
11. 激光功能.....	17
11.1 激光模式设置.....	17
11.2 基本控制模式.....	17
11.3 波形控制模式.....	17
11.4 位置比较控制模式.....	18
12. PT 运动.....	18

13. 闭环控制.....	18
14. 电子齿轮.....	19
15. 电子凸轮.....	19
16. 拓展资源及其他指令.....	20
16.1 辅助编码器.....	20
16.2 捕获功能.....	20
16.3 位置系统操作.....	20
16.4 螺距补偿和间隙补偿.....	21
16.5 控制器资源(参数、版本信息等).....	21
16.6 数据采集.....	22
16.7 坐标系变换功能.....	22
第二章 基本功能.....	23
一、库函数的使用方法.....	23
1. C++.....	27
2. C#.....	29
3. VB.NET.....	30
4. LABVIEW.....	31
二、控制器的打开和基本操作.....	39
1、功能原理.....	39
2、指令说明.....	39
3、综合示例.....	42
4、常见问题和注意事项.....	43
三、控制器的基本配置.....	44
1、功能原理.....	44
2、指令说明.....	45
3、综合示例.....	51
4、开环和闭环.....	51
四、控制器的状态检测.....	52
1、功能原理.....	52
2、指令说明.....	53

3、综合示例.....	56
五、点位和 JOG 运动.....	58
1、功能原理.....	58
2、指令说明.....	58
3、综合示例.....	64
六、回零运动.....	68
1、功能原理.....	68
2、指令说明.....	69
3、回零参数详细说明.....	70
4、回零图示.....	71
5、综合示例.....	73
6、注意事项.....	75
七、插补运动.....	76
1、功能介绍.....	76
2、插补概念及原理介绍.....	76
3、坐标系初始化.....	78
4、直线插补和平面圆弧插补.....	82
5、多维(8)插补指令.....	87
6、螺旋线插补.....	89
7、空间圆弧插补.....	91
8、缓存区的操作（IO、延时、单轴操作等）.....	92
9、打包指令介绍.....	97
9、坐标系状态检测.....	100
10、启动坐标系运动等其他指令.....	103
11、综合示例.....	106
八、IO 资源访问.....	110
1、功能原理.....	110
2、指令说明.....	110
3、综合示例.....	118
九、手脉功能.....	119

1、指令说明.....	119
2、综合示例.....	120
十、龙门功能.....	121
1、指令说明.....	121
2、综合示例.....	121
3、注意事项.....	122
第三章 高级功能.....	123
一、位置比较输出功能.....	123
1、多维位置比较输出.....	123
2、一维高速位置比较.....	127
3、二维高速位置比较.....	131
二、激光功能.....	136
1、激光模式设置.....	136
2、基本控制模式（DA、PWM 占空比、PWM 频率）.....	137
3、波形控制模式.....	145
4、位置比较控制模式.....	150
三、PT 运动功能.....	157
四、闭环控制功能.....	164
1、功能说明.....	164
五、电子齿轮.....	168
1、功能原理.....	168
2、指令说明.....	168
3、综合示例.....	170
六、电子凸轮.....	171
1、功能原理.....	171
2、指令说明.....	171
3、综合示例.....	174
七、拓展资源及其他功能.....	177
1、辅助编码器.....	177
2、捕获功能.....	177

3、位置系统操作.....	181
4、螺距补偿和间隙补偿.....	184
5、控制器资源(参数、版本信息等).....	186
6、数据采集功能.....	193
7、坐标系变换功能.....	197
第四章 GCS.exe 使用简介.....	204
一、运行 GCS.exe.....	204
二、测试单轴运动.....	205
1、测试.....	205
2、配置.....	206
3、回零——单轴回零测试.....	206
三、IO 测试.....	207
四、参数配置器.....	207
五、坐标系运动.....	210
1、指令——可以插入以下八种指令.....	210
2、运动——可以测试指令列表中的指令.....	210
六、激光控制测试.....	211
七、高级 IO 功能.....	214
八、位置时间 (PT) 运动.....	215
九、位置比较输出.....	216
十、其他资源 (辅助编码器等)	216
十一、通用 IO 扩展模块.....	218
十二、手轮测试。.....	219
十三、龙门测试.....	219
十四、IO 扩展模块.....	220
十五、位置捕获.....	221
十六、数据采集.....	222
十七、控制器信息.....	223
十八、系统时间及密码.....	223
十九、电机往复测试.....	224

二十、错误代码查询.....	225
二十一、指令调用监听器.....	225
二十二、多轴轨迹图.....	227
二十三、示波器.....	228
二十四、NMC 指令测试.....	230

第一章 指令汇总

关于NMC指令的统一说明

- 1.结构体定义中所有的'reservedxxx'的成员都是保留参数，请不要修改他们。
- 2.无特别说明，所有API返回RTN_CMD_SUCCESS（即0值）表示执行成功，其他则表示错误代码,见mtn_lib20_err.h。
- 3.所有的API参数中，无特别说明，[axisHandle](#)表示操作轴的句柄，[devHandle](#)表示目标控制器的句柄，[crdHandle](#)表示目标坐标系组句柄。
- 4.若无特别说明，所有API的参数中，牵涉到序号则从0开始，比如NMC_MtOpen中的itemNo参数，0表示第一个轴。
- 5.若无特殊说明，所有API的参数中，涉及到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）。
- 6.本手册只挑选了常用功能和函数，更多函数和功能请关注头文件或引用，我们在里面做了充分详细的注释。

1. 控制器的打开和基本操作

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（根据控制器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. 控制器的基本配置(单轴)

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_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

3. 控制器的状态检测

函数原形	函数说明
NMC_MtGetSts	读当前轴状态
NMC_MtClrError	清除轴错误状态
NMC_MtClrStsByBits	清除轴错误状态，按位清除
NMC_MtGetPrfPos	读取规划坐标系的位置
NMC_MtGetPrfVel	读当前轴规划速度
NMC_MtGetAxisPos	读取机械坐标的位置
NMC_MtGetCmdPos	读发送到执行器的位置
NMC_MtGetEncPos	读当前轴编码器通道
NMC_GetEncVel	读编码器速度(单位是：脉冲/ms)
NMC_MtGetStsPack12Ex	读取打包轴状态

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	获取 PTP 参数
NMC_MtSetPtpTgtPos	设置目标运动位置，只针对 PTP
NMC_MtGetPtpTgtPos	获取目标运动位置，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	获取目标运动速度
NMC_MtSetJogPara	设置 Jog 运动参数
NMC_MtGetJogPara	获取 Jog 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），包含 NMC_MtSetPrfMode , NMC_MtSetPtpPara , NMC_MtSetVel , NMC_MtSetPtpTgtPos , NMC_MtUpdate
NMC_MtMovePtpRel	单轴点位运动（相对运动）
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

注：PTP 指单轴点到点运动

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	读新回零位置和历史回零位置的差值

6. 坐标系的运动

6.1 坐标系的初始化

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	获取坐标系参数
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	读圆弧插补参数（高级指令）
NMC_CrdSetLookAheadCentriAcc	设置向心加速度限制
NMC_CrdSetFourthAxisTolTurnVel	设置 4 轴直线插补 A 轴最大拐弯速度

6.2 直线插补和平面圆弧插补

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
NMC_CrdArcCenterEx	XY 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 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 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）

6.3 多维(8)插补指令

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）

6.3 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）
--------------------------------------	--------------

6.4 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)

6.5 缓存区的操作 (IO、延时、单轴跟随等)

函数原形	函数说明
NMC_CrdBufDo	缓冲区 DO（按位输出）
NMC_CrdBufDoEx	缓冲区 DO（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_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

6.6 插补打包指令介绍

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

6.7 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	获取坐标系合成速度
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	获取插补缓冲区中尚未完成的总位移量

6.8 其他指令

函数原形	函数说明
NMC_CrdStartMtn	坐标系缓冲运动启动
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	获取坐标系速度倍率
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

7. IO 资源访问

函数原形	函数说明
NMC_SetDOGroup	设置通用输出(按通道,支持超过 32 位),带默认 group
NMC_SetDOBit	按位设置通用输出
NMC_GetDOGroup	读取通用输出(按通道,支持超过 32 位),带默认 group
NMC_GetDIGroup	读通用输入(按通道,支持超过 32 位),带默认 group
NMC_GetDIBit	按位读通用输入
NMC_MtGetMotionIO	读运动控制专用 IO
NMC_MtGetMotionIOLogical	读运动控制专用 IO,逻辑电平
NMC_SetDIBitRevs	按位设置通用输入信号取反
NMC_GetDIBitRevs	按位读取通用输入信号取反
NMC_SetDOBitRevs	按位设置通用输出信号取反

NMC_GetDOBitRevs	按位读取通用输出信号取反
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	获取所有的 DIO 映射数据
NMC_ClrAllDioMapping	清除所有的 DIO 映射关系

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

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

10. 位置比较输出

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	获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	获取多维位置比较的输出状态

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 1 维位置比较的参数（比较编码器）
NMC_CompHs1DimensGetParam	读取高速 1 维位置比较的参数
NMC_CompHs1DimensSetData	设置高速 1 维比较数据
NMC_CompHs1DimensOnOff	高速 1 维位置比较使能
NMC_CompHs1DimensStatus	高速 1 维位置比较的状态

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 2 维位置比较的参数（比较编码器）
NMC_Comp2DimensGetParam	读取高速 2 维位置比较的参数
NMC_Comp2DimensSetData	设置高速 2 维比较数据
NMC_Comp2DimensOnoff	高速 2 维位置比较使能
NMC_Comp2DimensStatus	获取高速 2 维位置比较的状态
NMC_Comp2DimensStatusEx	获取高速 2 维位置比较的状态

11. 激光功能

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

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

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	设置激光的能量,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

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_SHIOEnableMinFrq	SHIO 输出的最小频率功能开
NMC_SHIODisableMinFrq	SHIO 输出的最小频率功能关闭
NMC_CrdBufSHIOSetMinFrq	缓冲区 SHIO 输出的最小频率功能开
NMC_CrdBufSHIOClrMinFrq	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,注意: 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,注意: 必须在激光 SHIO 控制模式下使用
NMC_PwmPulseOut	输出一段 PWM 脉冲,注意: 必须在激光 SHIO 控制模式下使用

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	获取 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

13. 闭环控制

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	获取单轴闭环控制的 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	获取对应组号的 PID 参数
NMC_MtSetPIDIndex	设置使用哪组 PID
NMC_MtGetPIDIndex	获取正使用的 PID 组号

14. 电子齿轮

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

15. 电子凸轮

函数原形	函数说明
NMC_MtFollowSetDir	设置 FOLLOW 跟随方向
NMC_MtFollowGetDir	获取 FOLLOW 跟随方向
NMC_MtFollowSetMaster	设置 FOLLOW 主轴参数
NMC_MtFollowGetMaster	获取 FOLLOW 主轴参数
NMC_MtFollowSetLoopCount	设置 FOLLOW 的循环执行次数
NMC_MtFollowGetLoopCount	获取 FOLLOW 的循环执行次数
NMC_MtFollowSetEvent	设置 FOLLOW 的启动事件
NMC_MtFollowGetEvent	获取 FOLLOW 的启动事件
NMC_MtFollowGetSpace	获取 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
------------------------------------	----------------------

16. 拓展资源及其他指令

16.1 辅助编码器

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

16.2 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

16.3 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	获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

16.4 螺距补偿和间隙补偿

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
NMC_MtSetBacklash	设置反向间隙补偿
NMC_MtGetBacklash	获取反向间隙补偿

16.5 控制器资源(参数、版本信息等)

函数原形	函数说明
NMC_GetTime	读时间
NMC_SetTime	写时间
NMC_GetUID	读设备唯一序列号
NMC_UserParaWr	写用户参数
NMC_UserParaRd	读用户参数
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	获取错误代码信息
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	开始或者关闭变量的自动备份（断电自动保存，默认为关闭状态）
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	获取控制器的规划周期
NMC_GetLastError	读取最后一次的错误代码
NMC_SetErrCodeMode	设置指令错误返回值模式
NMC_SetWatchDog	设置指令通讯看门狗

16.6 数据采集

函数原形	函数说明
NMC_GetCollectDataAddr	获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	获取采集状态
NMC_GetCollectData	获取采集数据
NMC_ClearCollectSts	清除采集状态

16.7 坐标系变换功能

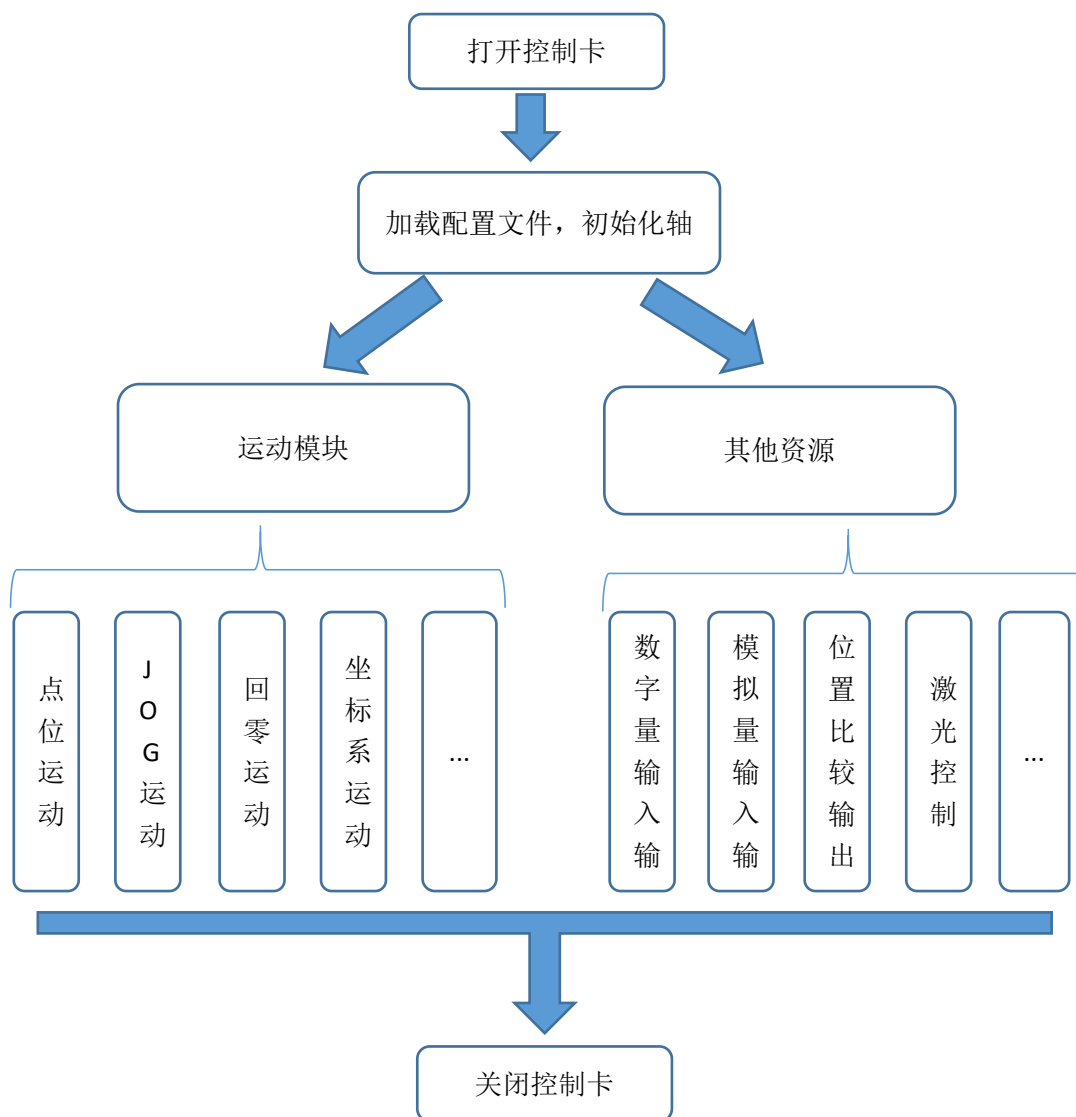
函数原形	函数说明
NMC_CrdSetTransRotate	使能旋转转换处理
NMC_CrdDelTransRotate	关闭旋转转换处理
NMC_CrdSetTransPolar	使能极坐标转换处理
NMC_CrdRunToPolarPos	运行至设定的极坐标位置并且进行圈数清零处理(利用单轴 PTP 运行到指定位置)
NMC_CrdRunToPolarTheta	运行至设定的极坐标角度位置(利用单轴 PTP 运行到指定位置)
NMC_CrdDelTransPolar	销毁极坐标机型(只恢复直角坐标系)
NMC_CrdSetTransXYZA	XYZA 机型设置接口
NMC_CrdDelTransXYZA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

第二章 基本功能

一、库函数的使用方法

GCN 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或功能模块。建议您使用您熟悉的语言进行开发。

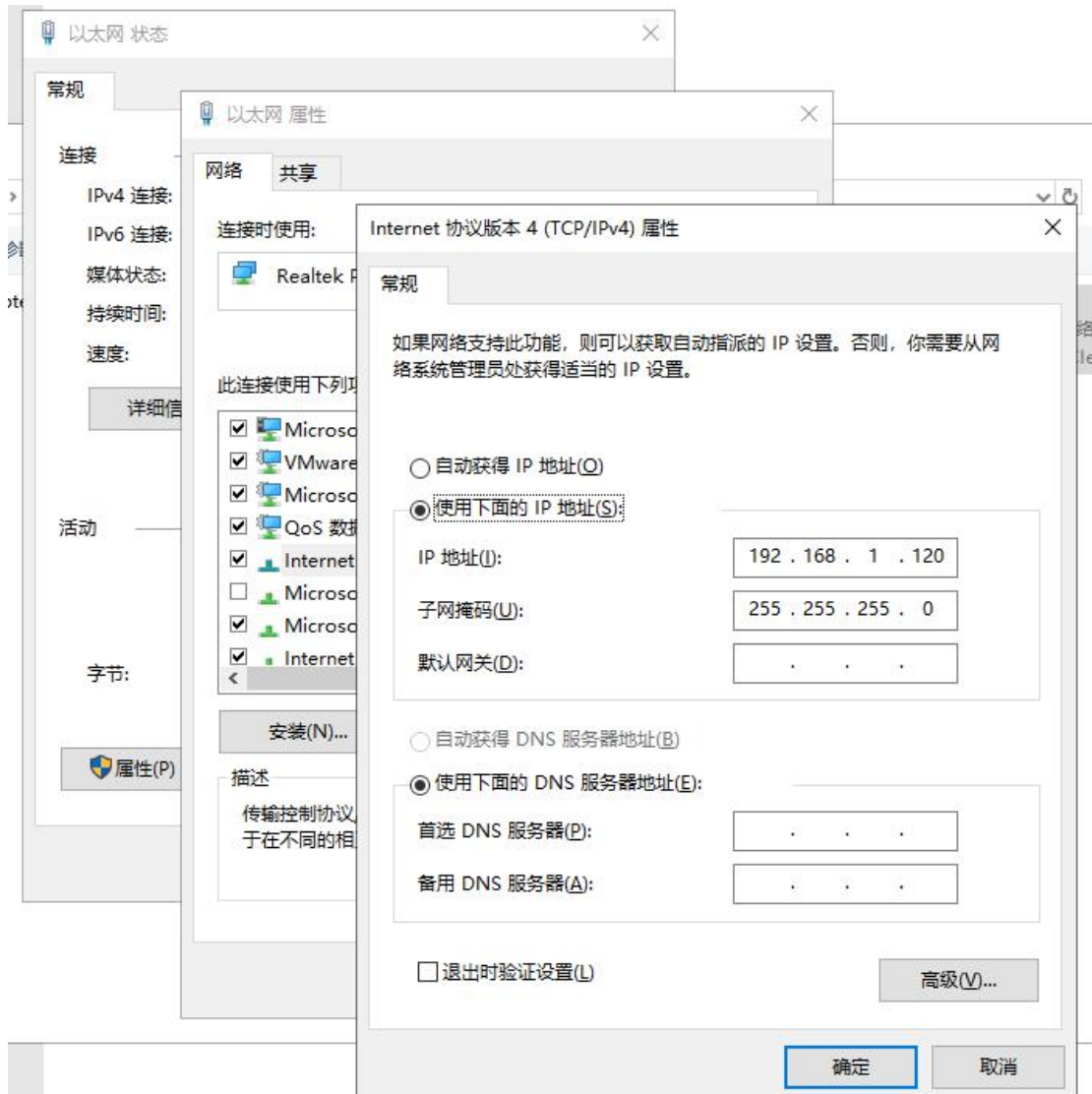
高川 GCN 系列运动控制卡的使用，参考以下思路，将有助于您对我们运动控制卡进行整体把握和深入了解。



GCN 系列控制器支持多种语言开发，网络版控制器不需要安装驱动，仅需要一根网线，即可实现客户端与控制卡的通讯，GCN 系列控制器默认 IP 为 192.168.1.110，只需将客户端设备的 IP 设置为相同网

段，即可通过函数实现链接。

当同时使用多个运动控制器时，需要客户端设备具备多个网口，分别与控制卡通过网线进行连接，这时，建议将控制卡 IP 设置为不同网段，例如 192.168.2.110 等等，对应的客户端网口 IP 也需要设置



为对应的网段，例如 192.168.2.120 等等，以免控制卡连接时发生混乱，或者搞不清楚控制的是哪张控制卡。修改控制卡 IP 的操作我们将放在本手册第四章“GCS.EXE 的使用”中进行描述。

PCIE 版本驱动安装

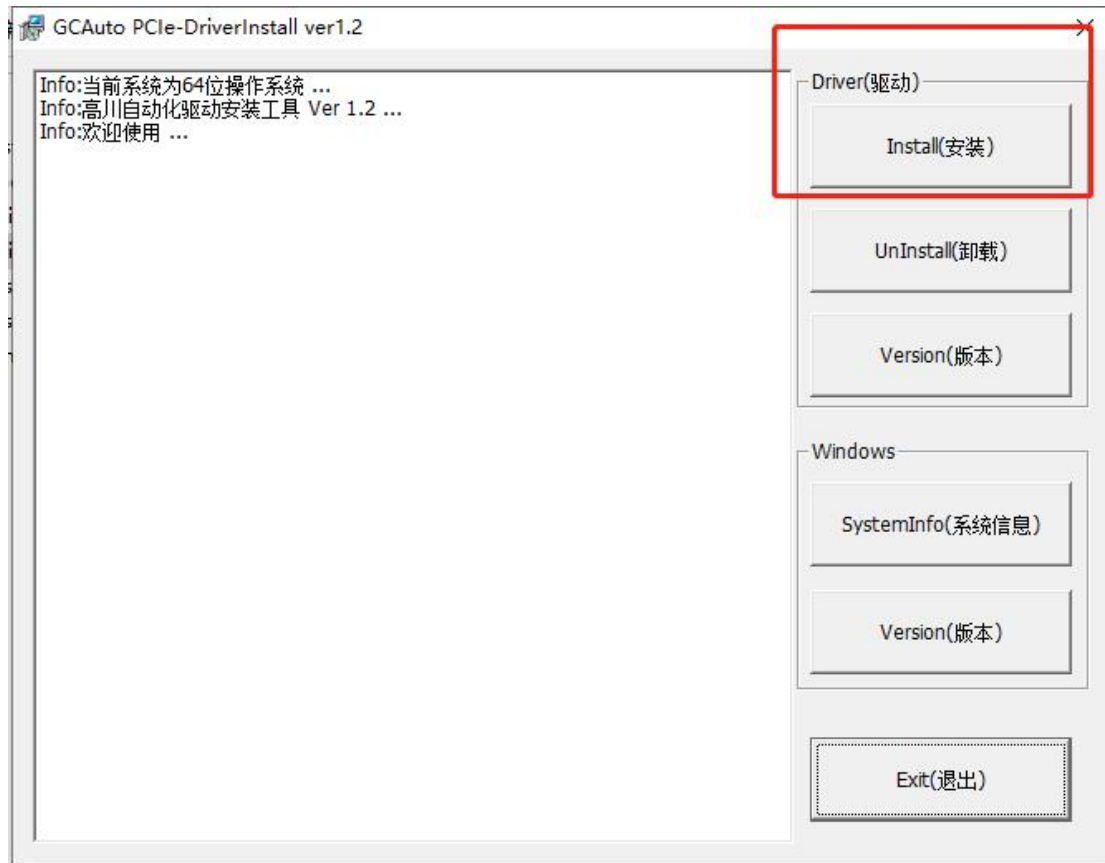
对于 PCIE 插卡式控制器需要安装驱动，安装工具在光盘资料：...\PC Base 开发资料\2. PCIE 驱动\高川 PCIe 卡驱动安装管理工具 1.2 下。

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。

GCN系列光盘资料2020新版 > PC Base开发资料 > 2. PCIE驱动 > 高川PCIe卡驱动安装管理工具1.2

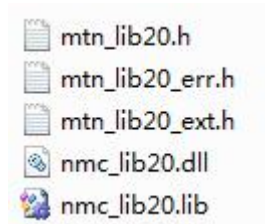
名称	修改日期	类型	大小
inst	2020/12/3 17:20	文件夹	
pach	2020/12/3 15:41	文件夹	
pcie	2020/12/3 17:20	文件夹	
DriverInstall.exe	2020/12/3 17:21	应用程序	1,658 KB
msvcp100.dll	2010/3/18 9:15	应用程序扩展	412 KB
msvcr100.dll	2010/3/18 9:15	应用程序扩展	753 KB
nmc_lib20.dll	2020/7/27 10:07	应用程序扩展	350 KB

点击按钮 **Install(安装)** 安装。



1. C++

1.将如下文件拷贝到工程目录



2.在源文件中包含头文件

3.设置 nmc_lib20.lib 为调用库

4.调用 NMC 指令，开始应用开发

下面是一个简单的例子，请参考

```
// 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"

// 引用库
#pragma comment(lib, "nmc_lib20.lib")

// 控制器初始化
short DevInit(void)
{
    short devNo = 0;           //设备序号，从零开始
    HAND devHandle=0;         //定义一个控制器句柄
    short rtn=0;               //用于接收函数返回值
    unsigned short devNum=0;   //设备数量
    TDevInfo devList[4];

    // 搜索控制器
    rtn = NMC_DevSearch(Ethernet, &devNum, devList);
    if(rtn != 0)
    {
        return rtn;           // 控制器搜索出错
    }
    if(devNum<= 0)
    {
        return 1;             // 没有找到控制器
    }

    // 打开第一个控制器
    rtn = NMC_DevOpen(devNo, &devHandle);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return rtn;           // 控制器打开出错
    }

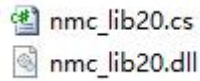
    // 复位控制器：控制器复位到初始状态（可不复位，默认加载保存到控制器的配置）
    rtn = NMC_DevReset(devHandle);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return rtn;           // 复位控制器出错
    }

    return 0;
}
```

注：详细源码请查看高川控制器光盘资料中的 C++ 例程。

2. C#

1.将如下文件拷贝至工程目录



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



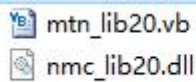
3.调用 NMC_指令即可开发（本手册主要阐述 c++代码，c#示例可参阅光盘目录里面的 c#例程）

```
ushort devhandle = 0;    //控制器句柄
ushort[] axishandle = new ushort[4]; //轴句柄
short rtn = 0;
//打开控制器，根据控制器名称打开，控制器名称可用我公司测试软件 GCS 获取到
rtn = CNMCLib20.NMC_DevOpenByID("CARD1", ref devhandle);
//复位一下
//rtn = CNMCLib20.NMC_DevReset(devhandle);
//加载配置文件
rtn = CNMCLib20.NMC_LoadConfigFromFile(devhandle,
System.Text.ASCIIEncoding.Default.GetBytes("GCN400.CFG"));
//打开各轴
for (short i = 0; i < 4; i++)
{
    rtn = CNMCLib20.NMC_MtOpen(devhandle, i, ref axishandle[i]);
    //清除各轴状态
    rtn = CNMCLib20.NMC_MtClrError(axishandle[i]);
    //位置清零
    //rtn = CNMCLib20.NMC_MtZeroPos(axishandle[i]);
    //伺服使能
    rtn = CNMCLib20.NMC_MtSetSvOn(axishandle[i]);
}
```

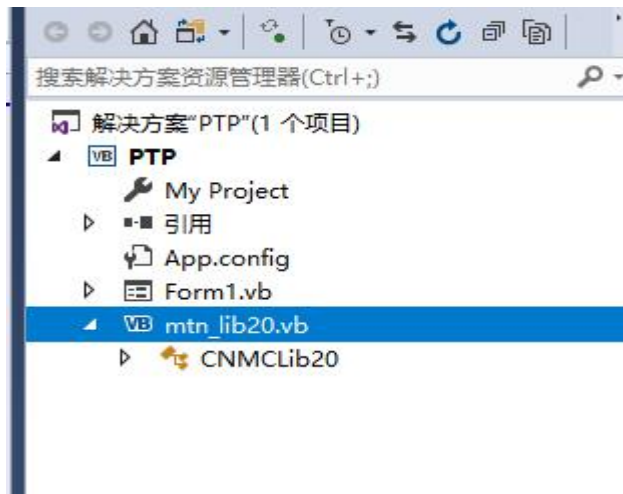
注：详细源码请查看高川控制器光盘资料中的 c#例程。

3. VB. NET

1.将如下文件拷贝到工程目录下：



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
    ' 打开轴 1
```

```
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtOpen(devhandle, 0, axishandle)
SendErrorMsg("NMC_MtOpen", rtn)
'清除轴状态
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtClrError(axishandle)
SendErrorMsg("NMC_MtClrError", rtn)
'位置清零
rtn = GC.Frame.Motion.Privt.CNMCLib20.NMC_MtZeroPos(axishandle)
SendErrorMsg("NMC_MtZeroPos", rtn)
'Timer1.Start()
End Sub
```

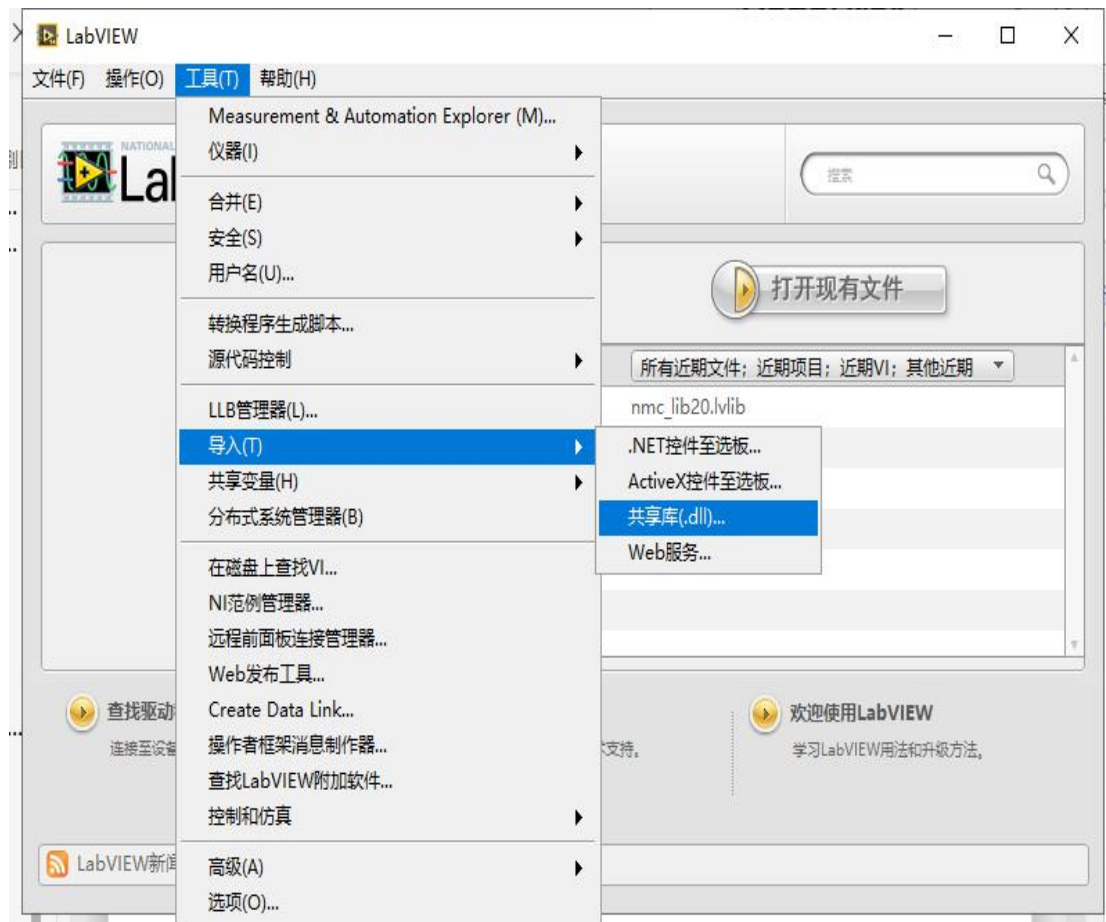
注：详细源码请查看高川控制器光盘资料中的 VB.NET 例程。

4. LABVIEW

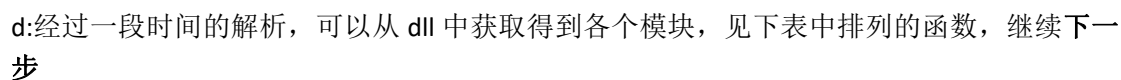
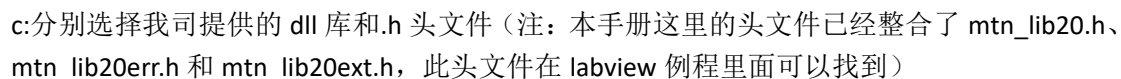
Labview 调用库函数编程有多种方式。

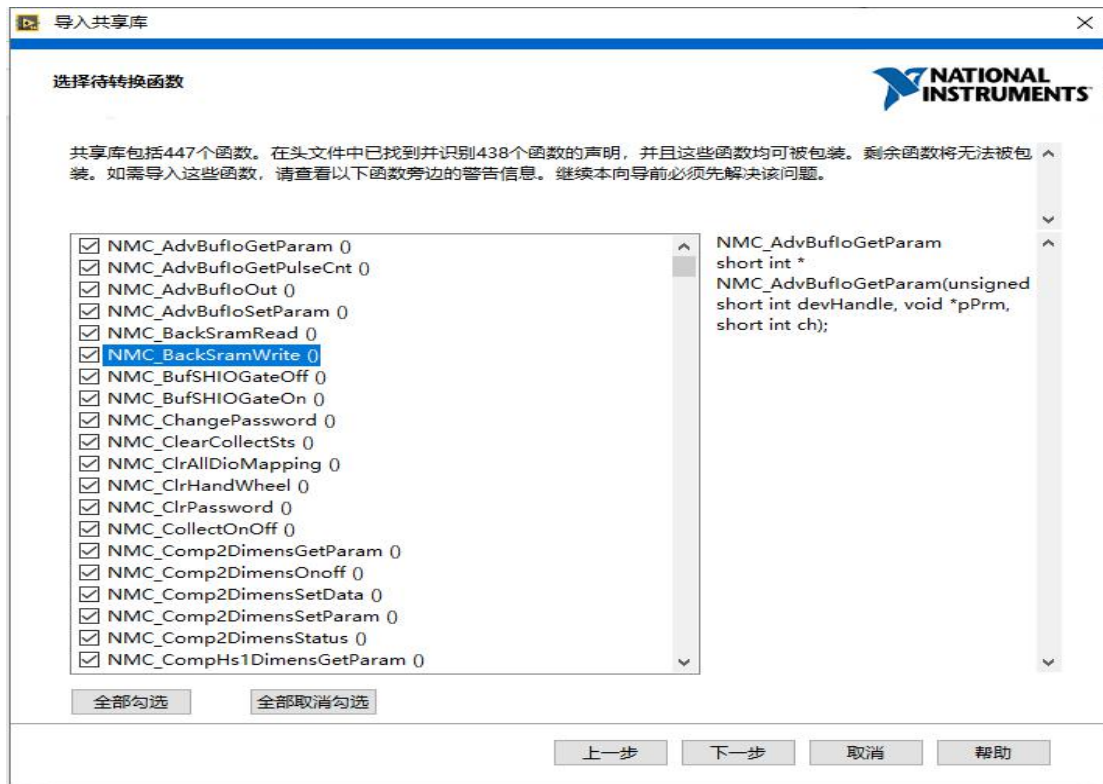
- 1、直接利用我司提供的 VIS 进行编程，具体可参照我司提供的 labview 例程。
- 2、利用 labview 的导入工具，将我们提供的“.h”文件和“.dll”文件导入，自己生成 VIs，示例如下：

a: 打开 labview，选择 工具->导入->dll 共享库

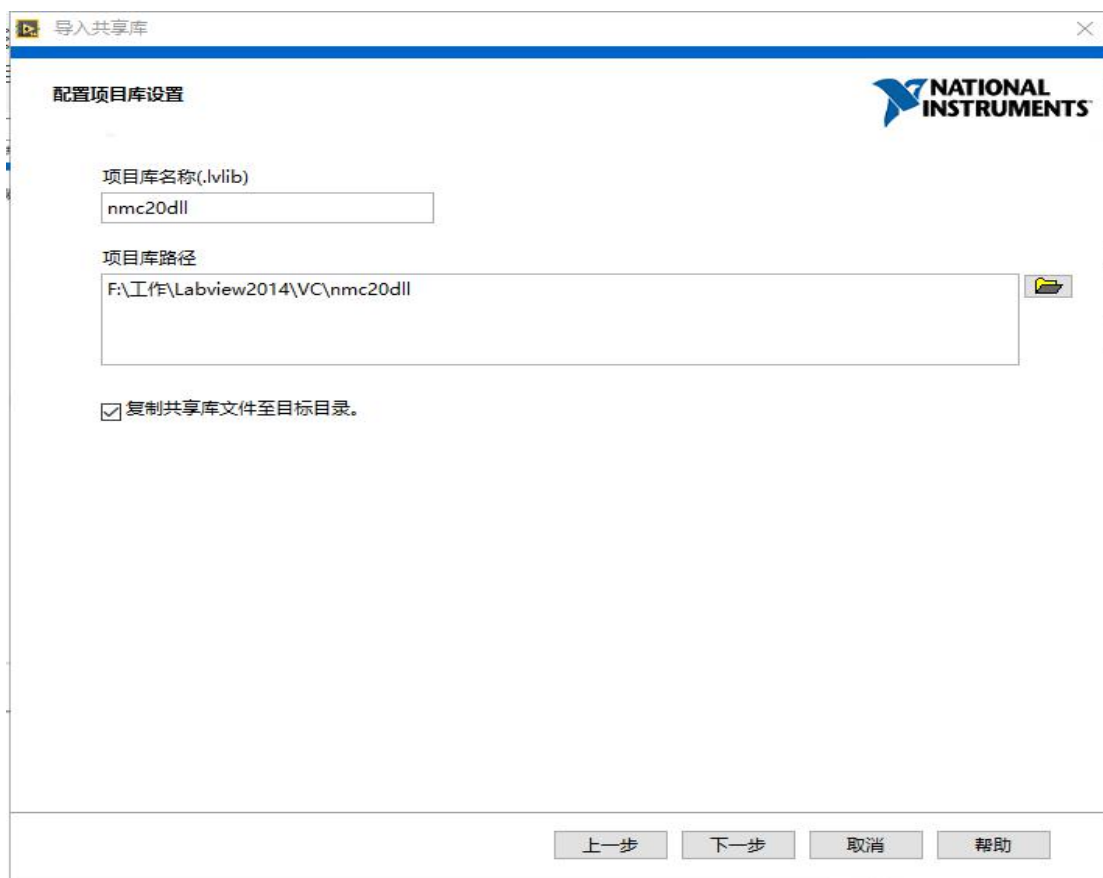


b: 为共享库创建 VI->下一步。（如果之前导入过，也可以选择 为共享库更新 VI）

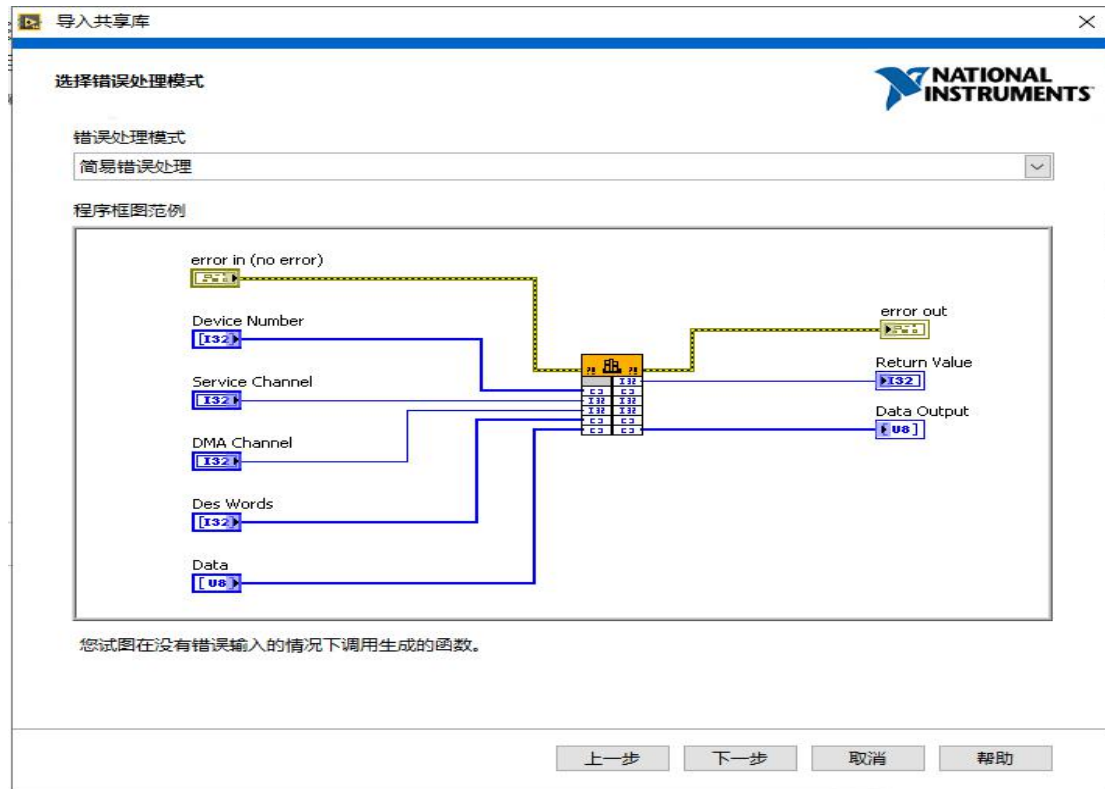




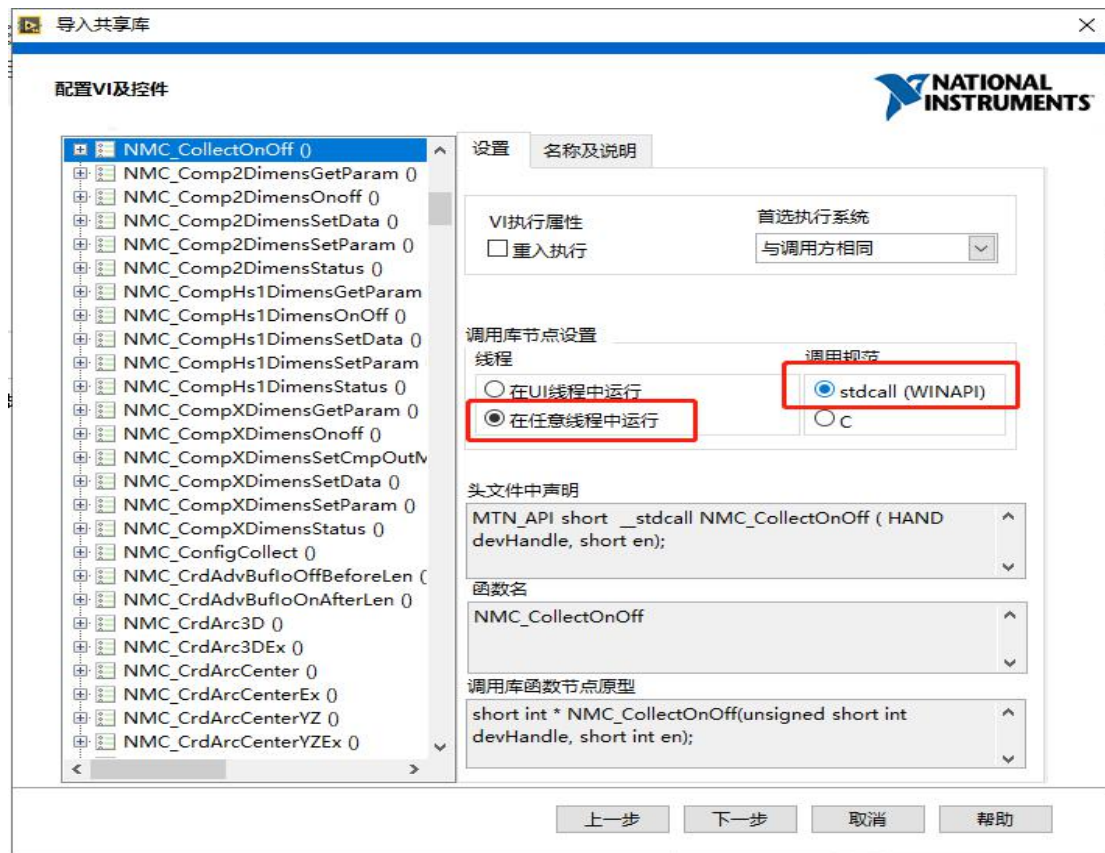
e:填写导出名称和选择导出路径，下一步



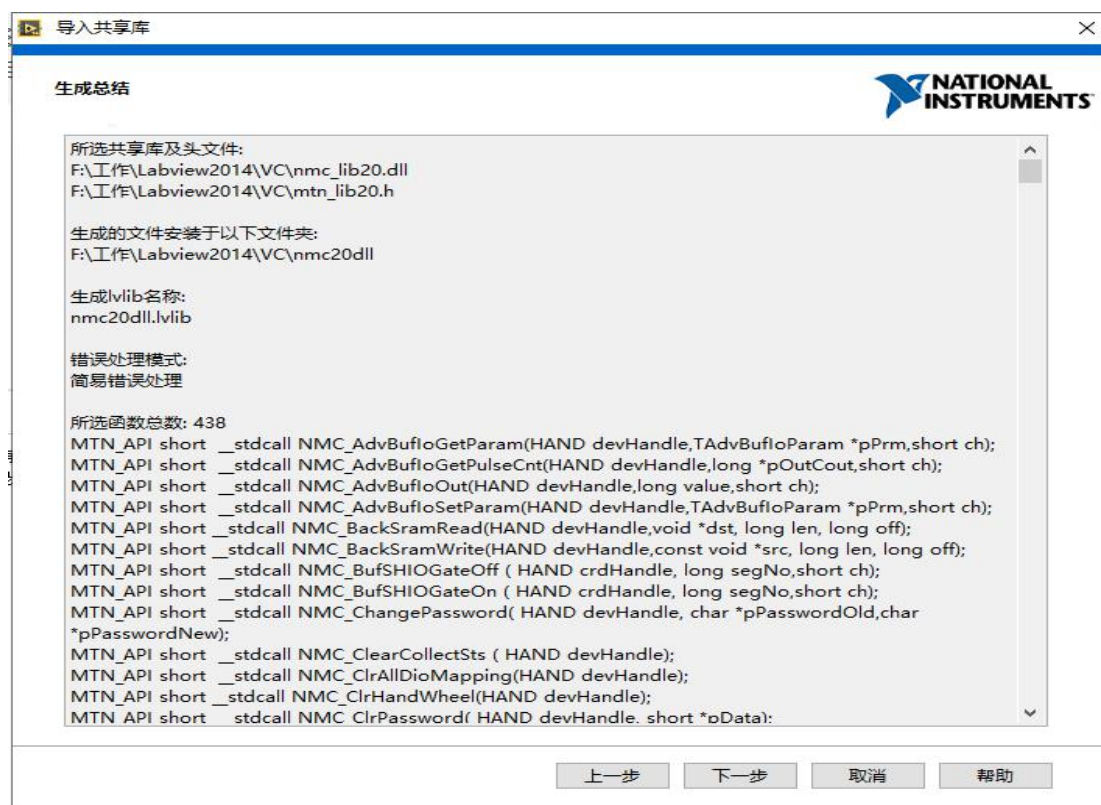
下一步



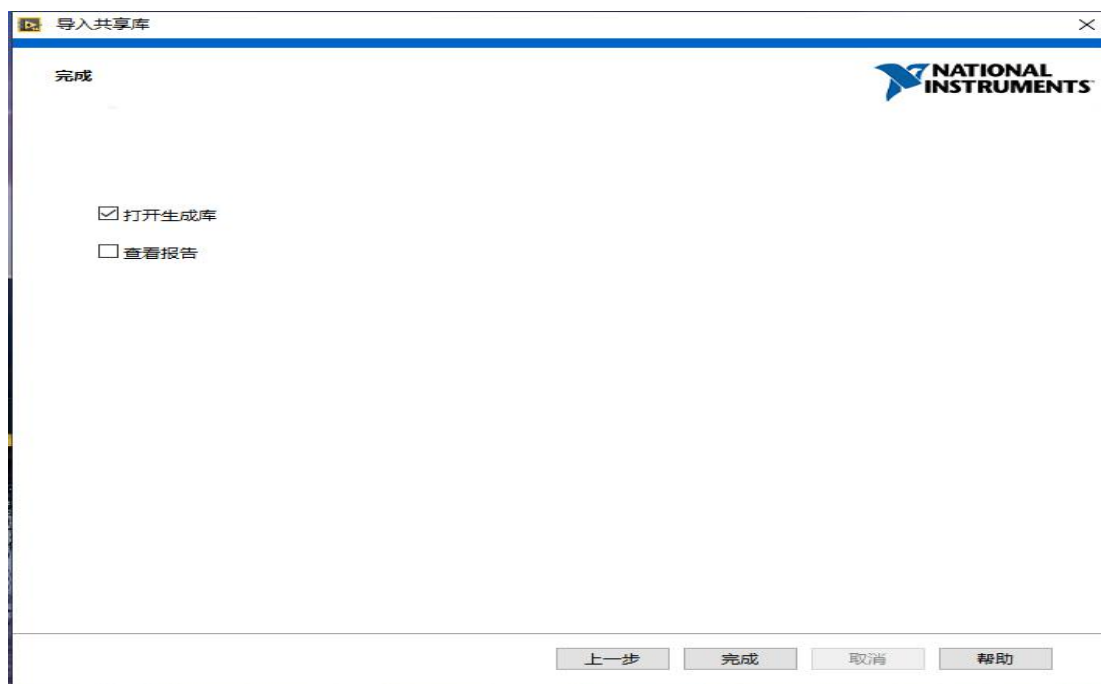
f:为所有函数选择在任意线程调用和 **stdcall(WINAPI)**，然后下一步。（请不要只选择一个就直接下一步，否则未选择的函数会在调用时出错）

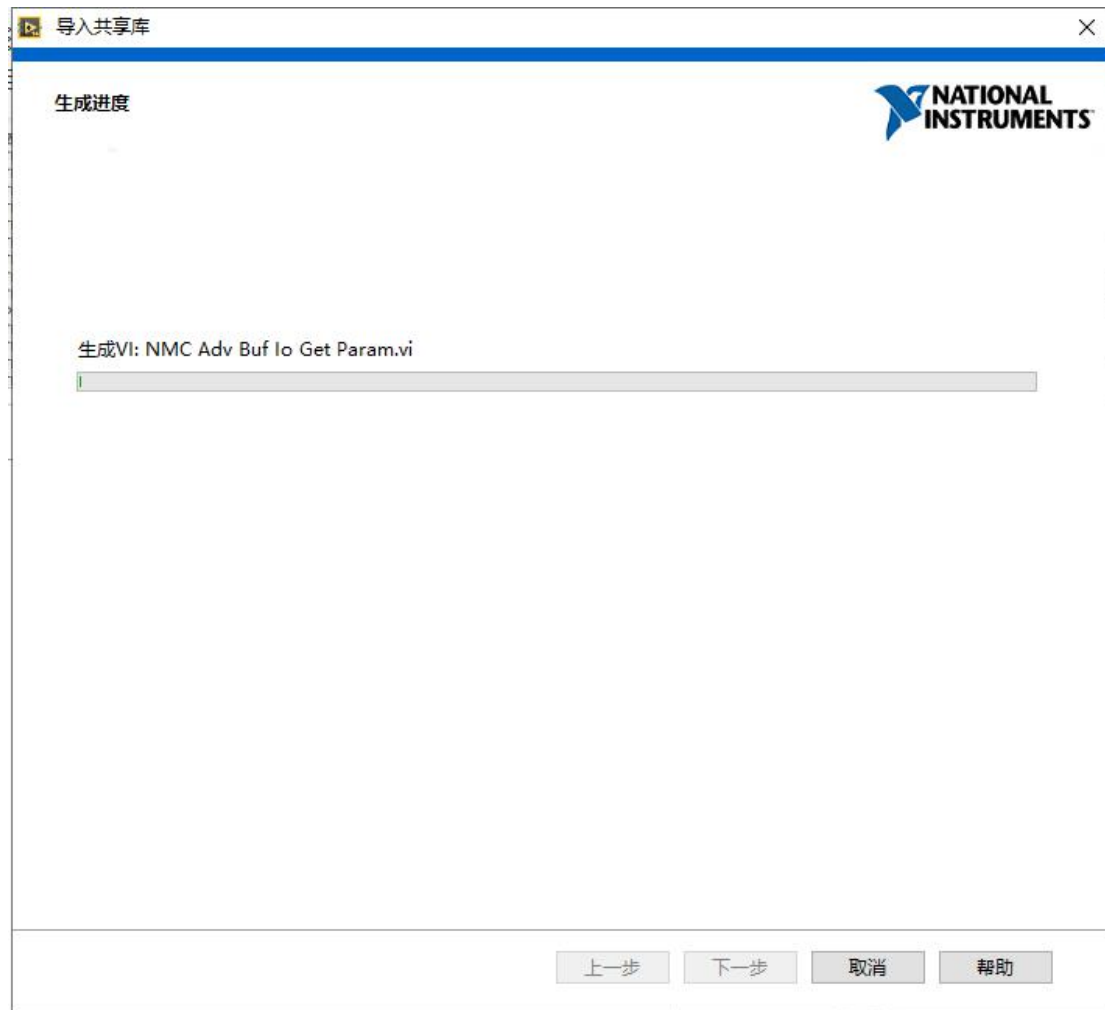


g:下一步

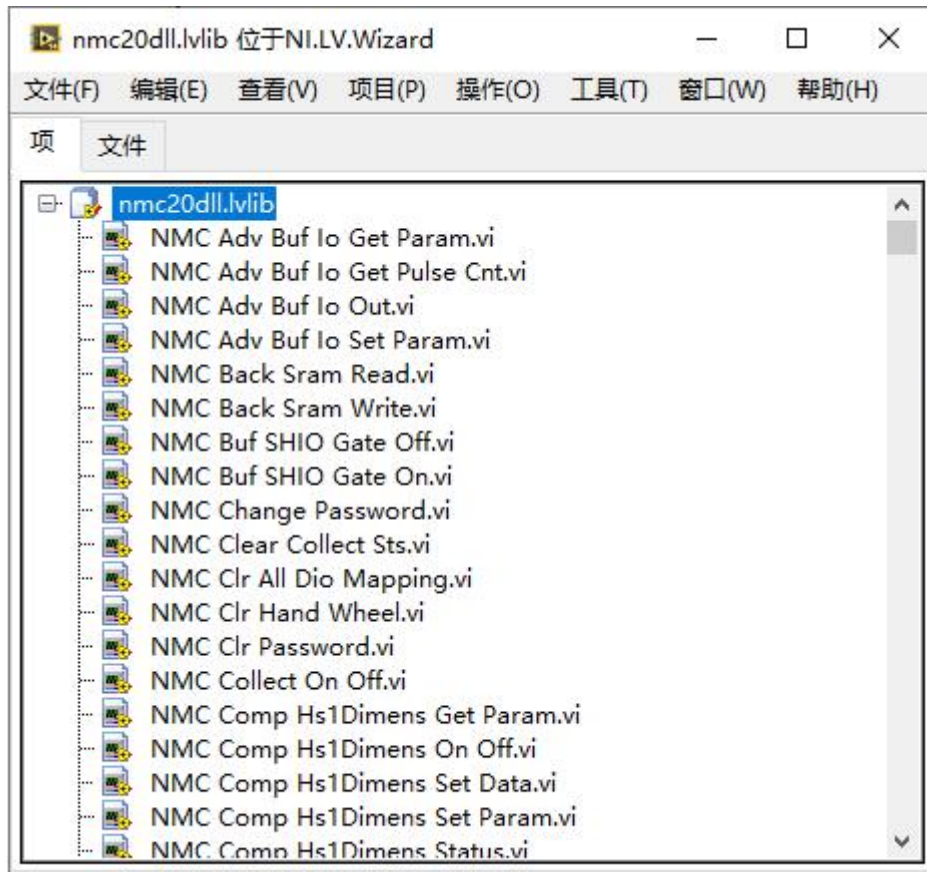


h:等待生成完成，这个过程会比较占用时间，耐心等待即可。

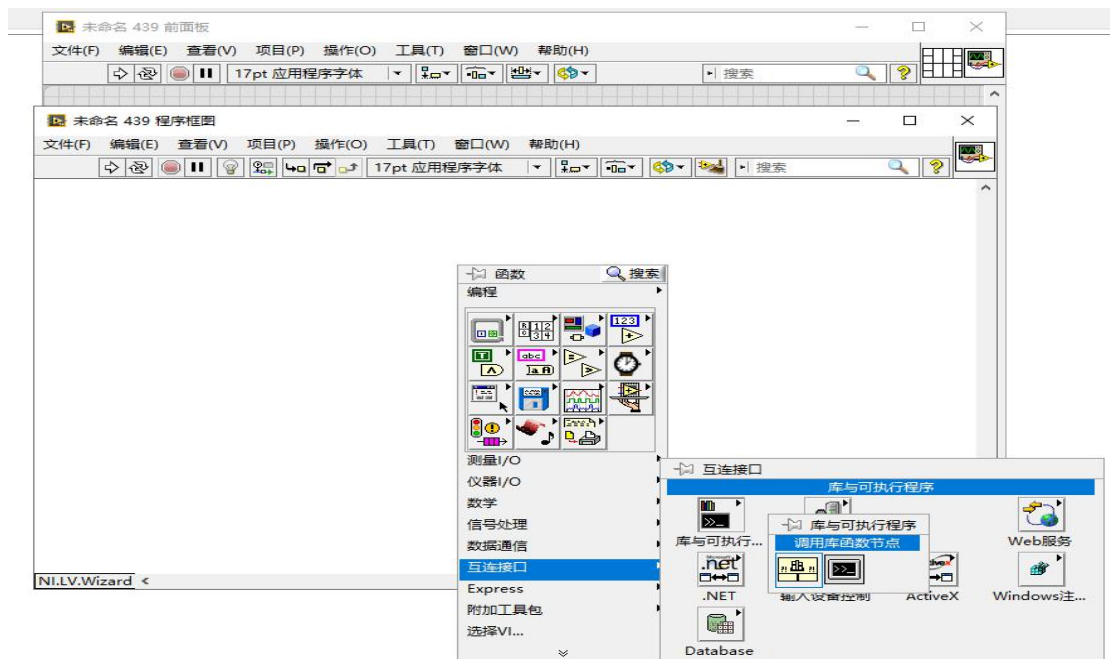


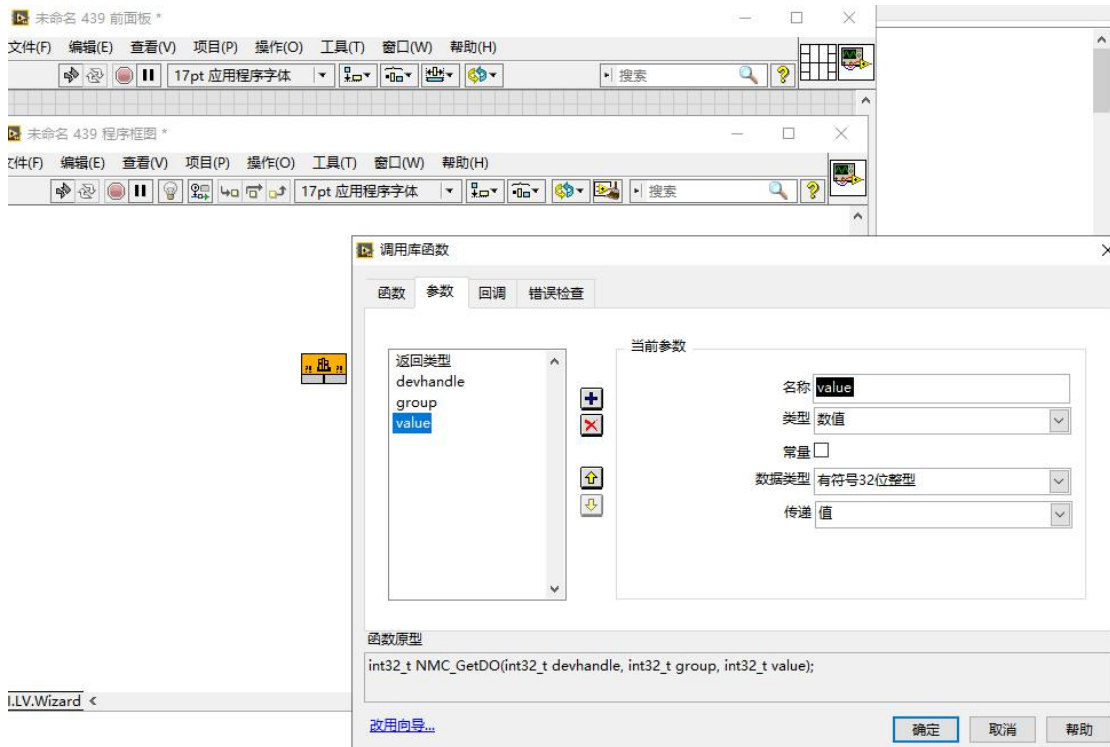
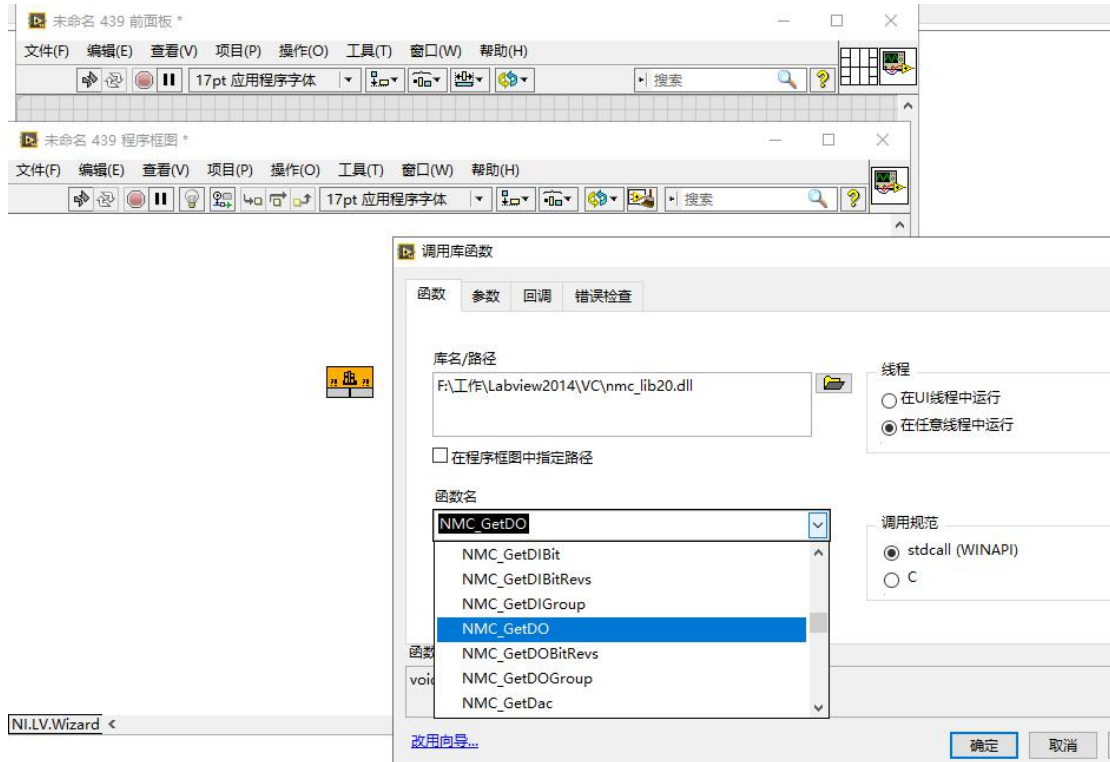


i:至此就可以在项目中拖入使用了



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。





二、控制器的打开和基本操作

1、功能原理

※控制器的打开基本分为三种方式。

※调用函数 [NMC_DevOpen\(\)](#) 按照序号打开, 序号从 0 开始。

※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.EXE 设置控制卡的名称。

※按照网络型控制器设置的 IP 地址开打, [NMC_DevOpenByIP\(\)](#), 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 IP, 如 192.168.2.110, 192.168.3.110 等。

※网络型控制器的 IP 地址不能设置成 169.254.x.x。

※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。

※当使用多网络型控制器时, 推荐使用后两种方式打开控制器, 此时不同的控制器最好设置不同的名称和不同网段的 IP。

※插卡式控制器 (PCIE 版本) 需要安装驱动才能打开, 不支持 [NMC_DevOpenByIP](#) 打开。

2、指令说明

1) 搜索已连接的控制器

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

功能: 搜索可用的、已连接计算机的控制器。不包括已经打开的设备。返回控制器的数量、名称、以及描述信息。

参数:

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 根据实际使用的连接选择。
pDevNo	unsigned short*	输出	返回已连接设备的数目。
pInfoList	TDevInfo *	输出	返回设备信息列表 typedef struct { unsigned short address; // 在上位机系统中的设备序号, char idStr[16]; // 识别字符串 char description[64]; // 描述符 unsigned short ID; // 板上的 ID(未用) } TDevInfo;

2) 根据序号打开控制器

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

功能：根据序号打开板卡。

参数：

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

3) 复位控制器

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

功能：复位控制器

参数：

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

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

功能：根据 ID 打开控制器，取得操作句柄。使用 ID 打开，在多卡时可以不受硬件端口变化的影响，能可靠地打开预先设定好的控制器。用户可以通过软件工具或指令修改此名称。

参数：

名称	数据类型	输入/输出	描述
idStr	char *	输入	ID 字符串。 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

5) 根据 IP 地址打开控制器

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

功能：根据控制器 IP 地址打开控制器，取得操作句柄。控制器默认 IP 是 192.168.1.110, 用户可以通过软件工具或指令修改 IP。

参数：

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: unsigned char ipv4[4] = {192,168,1,110};
pDevHandle	PHAND	输出	控制器句柄指针

注意：控制卡的 IP 地址不能设置成 169.254.x.x。

6) 关闭控制器

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

功能：关闭控制器

参数：

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

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

功能：打开单轴，获取轴句柄

参数：

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\(PHAND pAxisHandle \);](#)

功能：关闭单轴。

参数：

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

功能：打开坐标系。

参数：

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，该指针所指值一般是257

10) 打开坐标系组（支持多坐标系）

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

功能：打开坐标系组（支持多坐标系）。

参数：

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，该指针所指值一般是257

11) 关闭坐标系组

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

功能：关闭坐标系组

参数：

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

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

功能：加载配置文件，配置文件从 GCS.EXE 经过配置得到。

参数：

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

13) 保存配置信息到硬件内存

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

功能：保存基本的配置信息,将这些信息保存到系统参数区,控制器重启或者 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。

参数：

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 表示启用保存功能，0 表示关闭

3、综合示例

（本实例在 VS2008MFC 下完成，后续不做特别说明，均指该编程环境，源码在光盘资料的“运动控制编程说明”文件夹中）

```
#include "mtn_lib20.h"
#include "mtn_lib20_err.h"
#include "mtn_lib20_ext.h"

#pragma comment(lib,"nmc_lib20.lib")

void CheckReturn(int rtn,CString funcname)
{
    if (rtn!=0)
    {
        CString str=L"";
        str.Format(L"error=%d",rtn);
        MessageBox(NULL,str,funcname,NULL);
    }
}

//控制器句柄
HAND devhandle;

//轴句柄
HAND axishandle[4];

void CExampleDlg::OnBnClickedButton1()
{
    //搜索控制器可省略，直接打开控制器
    short rtn=0;
    //打开控制器
```

```

rtn=NMC_DevOpen(0,&devhandle);
CheckReturn(rtn,L"NMC_DevOpen");
//复位控制器
rtn=NMC_DevReset(devhandle);
CheckReturn(rtn,L"NMC_DevReset");
//加载控制器配置
rtn=NMC_LoadConfigFromFile(devhandle,"GCN800.cfg");
CheckReturn(rtn,L"NMC_LoadConfigFromFile");
//获取各轴操作句柄
for(int i=0;i<4;i++)
{
    rtn=NMC_MtOpen(devhandle,i,&axishandle[i]);
    CheckReturn(rtn,L"NMC_MtOpen");
}
}

```

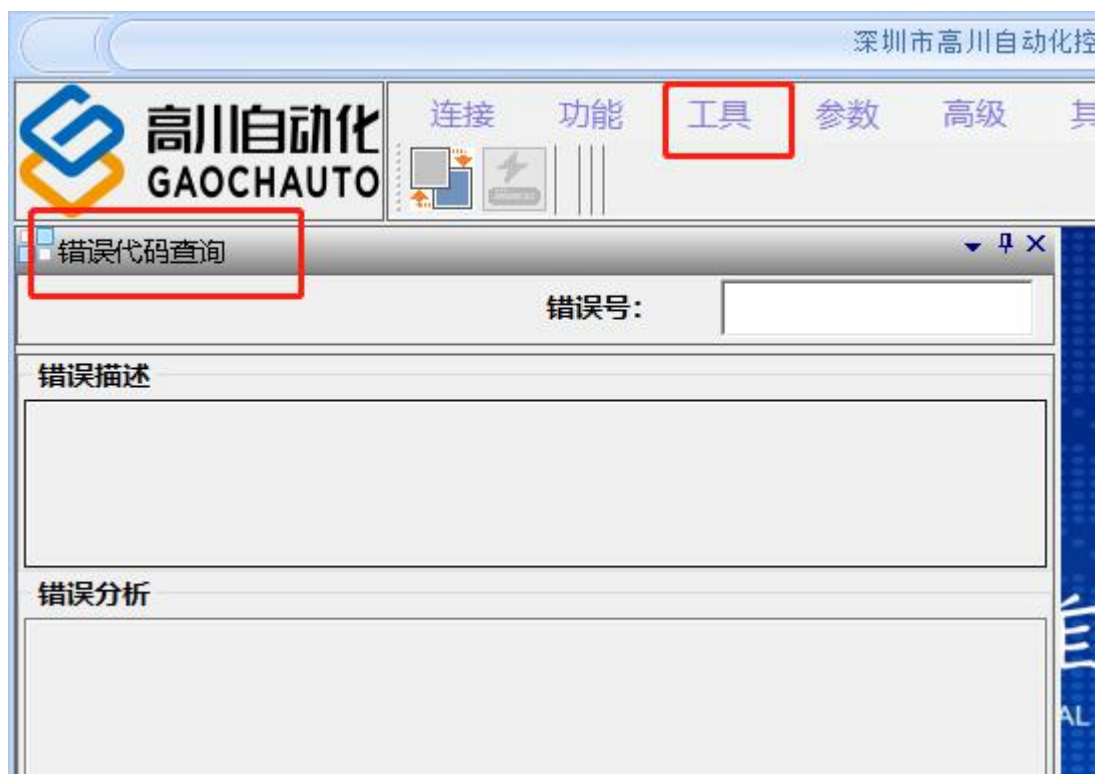
4、常见问题和注意事项

1) 多种网络设备时，IP 地址该如何分配？

当遇到多种网络设备时，比如有相机和控制器同时使用，应将控制器和相机的 IP 地址设置成不同的网段。

2) 调用 API 函数返回值报错（不等于 0）应该怎么处理？

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 软件工具里面的错误代码查询功能进行查询。或者也可以通过 vc 引用的头文件进行搜索。





三、控制器的基本配置

1、功能原理

※在控制器能正常使用之前,除了打开控制器外,我们还需要对其进行一些配置。

※包括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输入输出等等。

※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.EXE 配置工具生成配置文件,然后调用函数 [NMC_LoadConfigFromFile\(\)](#) 加载到控制器去。

※关于配置文件的生成我们将在本手册第四章给您描述。



2、指令说明

1) 伺服报警配置

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

功能：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

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

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

功能：读取伺服报警触发是否有效的配置

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

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

功能：读取伺服报警触发电平

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

2)限位配置

[NMC_MtSetLmtCfg\(HAND axisHandle, short posLmtEnable,short negLmtEnable, short posLmtSns,short negLmtSns \)](#)

功能：硬件限位配置，同时配置正负限位是否有效及其触发电平

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

[NMC_MtGetLmtOnOff\(HAND axisHandle,short*posEn,short*negEn\)](#)

功能：读取限位触发是否有效

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

[NMC_MtGetLmtSns\(HAND axisHandle, short*posSwt, short*negSwt\)](#)

功能：读取限位触发电平

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

3)软限位配置

[NMC_MtSwLmtOnOff \(HAND axisHandle, short enable \)](#)

功能：设置软件限位触发是否有效

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

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

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

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

[NMC_MtSwLmtValue \(HAND axisHandle, long posLmt,long negLmt \)](#)

功能：设置软件限位的数值

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

[NMC_MtGetSwLmtValue \(HAND axisHandle, long *posLmt,long *negLmt \)](#)

功能：读取软件限位的数值

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

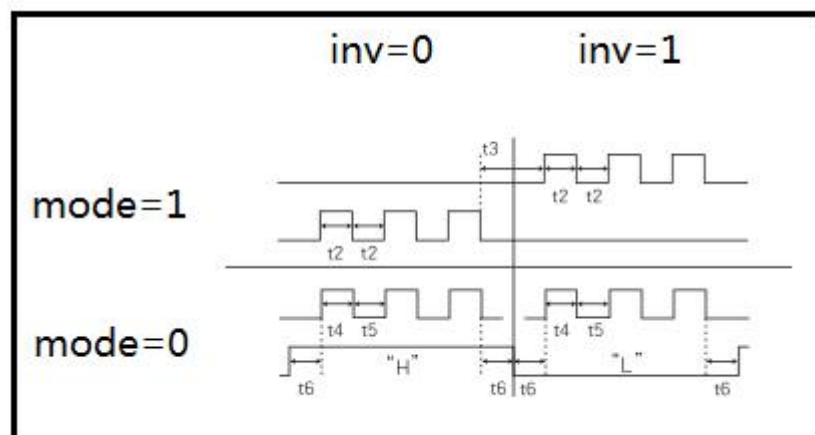
4)脉冲输出配置

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

功能：设置脉冲输出模式，默认不取反、输出模式为脉冲+方向

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：



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

功能：读取脉冲输出模式

参数：参考 NMC_MtSetStepMode

5)编码器模式配置

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

功能：设置编码器模式

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，取值 0~N
encMode	unsigned short	输入	<p>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：</p> <p>16 位编码器的模式字</p> <p>Bit7:0 分频系数，数值为~255。对应分频数值 1~256</p> <p>Bit9:8 信号号源选择 00: 外部信号输入 01: 轴脉冲输入 10: 自动产生信号（正脉冲） 11: 自动产生信号（负脉冲）</p> <p>Bit11:10 信号类型（外部） 00: AB 相，度差 01: 脉冲+方向 10: 正脉冲+负脉冲 11: 保留</p> <p>Bit12 输入 A、B 交换（外部） 0: 不交换，: 交换</p> <p>Bit13 输入 A 取反（外部） 0: 不取反，: 取反</p> <p>Bit14 输入 B 取反（外部） 0: 不取反，: 取反</p> <p>Bti15 编码器饱和 0: 最大最小值翻转，1: 不翻转</p>

※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.

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

功能：读取编码器模式

参考 [NMC_SetEncMode](#)

6)轴运动安全参数配置

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

功能：设置轴安全参数

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pPara	TSafePara*	输入	安全相关的参数。 <pre>typedef struct { double estpDec; double maxVel; double maxAcc; }TSafePara;</pre> <p>estpDec：急停加速度，单位是脉冲/ms*ms，不设置默认为无穷大 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。不设置时默认为无穷大 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。不设置时默认为无穷大</p>

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

功能：读取轴安全参数

参考 [NMC_MtSetSafePara](#)

[NMC_MtSetPosErrLmt\(HAND axisHandle, long _posErr \)](#)

功能：设置允许的位置误差

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
posErr	long	输入	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 posErr 为 0 表示不检查

[NMC_MtGetPosErrLmt\(HAND axisHandle, long *posErr \)](#)

功能：读取允许的位置误差

参考 [NMC_MtSetPosErrLmt.](#)

7)滤波，增加速度平滑，减少抖动

[NMC_MtSetAxisVelFilter\(HAND axisHandle, short filterCoef \)](#)

功能：设置轴速度的滤波参数

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

[NMC_MtGetAxisVelFilter\(HAND axisHandle, short *filterCoef \)](#)

功能：读取轴速度的滤波参数

参考 [NMC_MtSetAxisVelFilter](#)

[NMC_MtSetStepFilter \(HAND axisHandle, unsigned short coe\)](#)

功能：设置脉冲输出滤波

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, 单位 ms, 0 不滤波, 数值越大滤波效果越明显, 默认值为 0

[NMC_MtGetStepFilter \(HAND axisHandle, unsigned short *pCoe \)](#)

功能：读取脉冲输出滤波

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, 单位 ms, 0 不滤波, 数值越大滤波效果越明显, 默认值为 0

8) 急停 IO

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

功能：设置单轴急停 DI

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

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

功能：读取单轴急停 DI

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
gpiIndex	Short*	输出	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*	输出	触发电平,0: 低电平,1: 高电平

3、综合示例

本示例对轴一进行了简单的配置，配置完成之后清除一下轴状态 `NMC_MtClrErr`，便可以使能轴 `NMC_MtSetSvOn`。

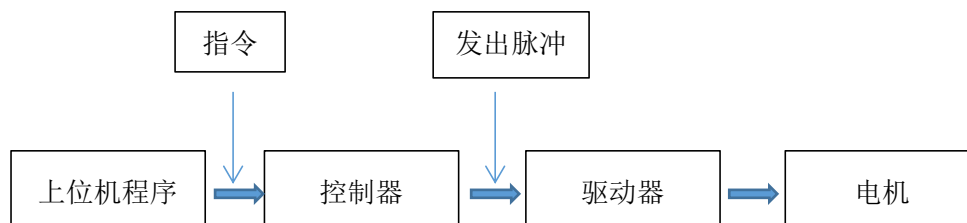
```
//设置轴限位有效，低电平触发
//axishandle[0]为上文中获取的一号轴的句柄
short rtn=0;
rtn=NMC_MtSetLmtCfg(axishandle[0],1,1,0,0);
//设置一轴的轴报警有效，低电平触发，如果控制器没接报警信号，此处应该将轴报警无效
rtn=NMC_MtSetAlarmCfg(axishandle[0],1,0);
//设置一号轴的脉冲方式为脉冲加方向，不取反
rtn=NMC_MtSetStepMode(axishandle[0],0,0);
//设置编码器读数为外部计数,如果没接编码器信号到控制器，此处应该为内部计数
rtn=NMC_SetEncMode(axishandle[0],0,0x0000);
//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
/*rtn=NMC_MtSetAxisVelFilter(axishandle[0],1);*/
```

4、开环和闭环

我们经常听到控制器开环控制或者闭环控制，那么开环、闭环是什么意思呢？在高川运动控制器中如何操作呢？接下来给大家做一个简要的描述。

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。

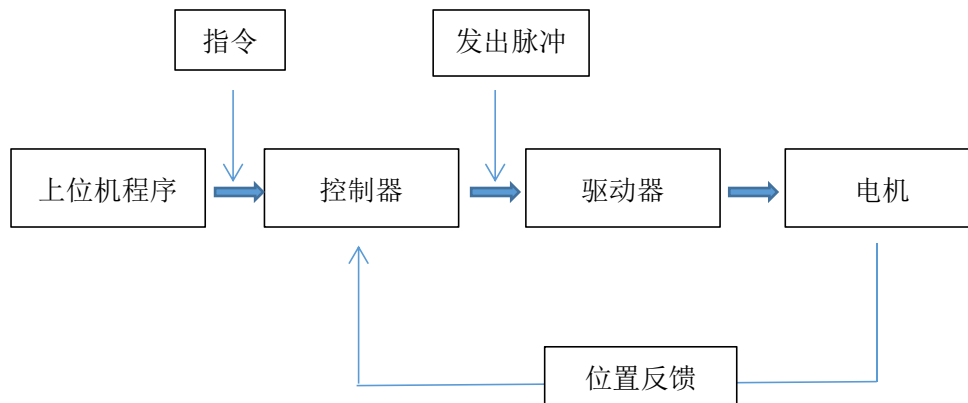


在这个过程中编码器位置信息的采集并不参与控制调整，但是它可以用来判断运动是否真正到位，仅此而已。

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）。

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 \(HAND axisHandle,short mode\)](#) 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。

四、控制器的状态检测

1、功能原理

※控制器的状态检测方便我们实时获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。

轴状态字定义表	
位	说明
Bit0	该位为 1 表示：运动，0：静止
Bit1	该位为 1 表示：伺服位置到达，步进模式时位置到达，伺服模式时实际位置到达误差限
Bit2	该位为 1 表示：上次运动出错，或当前无法启动运动，需要软件复位
Bit3	该位为 1 表示：伺服使能
Bit4	该位为 1 表示：坐标系模式
Bit5	该位为 1 表示：步进，0：伺服
Bit6	该位为 1 表示：正向限位触发
Bit7	该位为 1 表示：负向限位触发
Bit8	该位为 1 表示：正向软限位触发
Bit9	该位为 1 表示：负向软限位触发
Bit10	该位为 1 表示：伺服报警，需要软件复位
Bit11	该位为 1 表示：位置超限，需要软件复位
Bit12	bit 12,该位为 1 表示：急停，需要软件复位
Bit13	bit 13,该位为 1 表示：硬件错误
Bit14	bit 14,该位为 1 表示：捕获触发

2、指令说明

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

功能：读取轴状态

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

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

功能：清除轴的错误状态

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

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

功能：清除轴的错误状态,按位清除

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

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

功能：读取轴规划位置

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

[NMC_MtGetPrfVel\(HAND axisHandle, double *pVel \)](#)

功能：读取轴规划速度

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

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

功能：读取轴机械位置

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

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

功能：读取轴命令位置

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

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

功能：读当前轴编码器通道

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

[NMC_GetEncVel\(HAND devHandle, unsigned char encId, double *vel \)](#)

功能：按序号读取编码器通道的速度，当获取第一个辅助编码器速度时 encId=256

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)
-----	----------	----	------------------

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

功能：打包读取 12 个轴的状态

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 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, 位定义参考 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 long gpo; // 通用输出 long crdLeftLen[2];</pre>

			<pre> // 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号站~8号站 long extDo[6]; // 扩展模块输出, 2号站~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>
--	--	--	---

3、综合示例

```

short rtn=0;

short axsists=0;
//获取轴状态=====
rtn = NMC_MtGetSts(axishandle[0], &axsists);
//取 bit0,0 静止,1 运动
bool IsRuning = (axsists & (1 << 0)) == 0 ? false : true;
//取 bit1,位置到达,0,未到达,1 到达
bool IsArrive = (axsists & (1 << 1)) == 0 ? false : true;
//取 bit2,运动是否出错,0,未出错,1,出错
bool IsError = (axsists & (1 << 2)) == 0 ? false : true;
//取 bit3,是否使能中
bool IsAxisOn = (axsists & (1 << 3)) == 0 ? false : true;
//取 bit6,正向限位是否触发
bool PosArrived = (axsists & (1 << 6)) == 0 ? false : true;
//取 bit7,负向限位是否触发
bool NegArrived = (axsists & (1 << 7)) == 0 ? false : true;
//取 bit8,正向软限位是否触发
bool PosSoftArrived = (axsists & (1 << 8)) == 0 ? false : true;
//取 bit9,负向软限位是否触发

```



```
bool NegSoftArrived = (axsists & (1 << 9)) == 0 ? false : true;
//取 bit10,驱动器是否报警
bool IsAlarming = (axsists & (1 << 10)) == 0 ? false : true;
//取 bit11,位置是否超过误差极限
bool IsPosErr = (axsists & (1 << 11)) == 0 ? false : true;
//=====分割线=====

//轴初始化完成之后，或者轴运动之前，或者轴报错，都可以调用清除轴状态的指令
rtn=NMC_MtClrError(axishandle[0]);
//=====分割线=====

//获取轴一规划位置 and 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;
rtn=NMC_MtGetPrfPos(axishandle[0],&prfpos);
rtn=NMC_MtGetEncPos(axishandle[0],&encpos);
rtn=NMC_MtGetPrfVel(axishandle[0],&prfvel);
rtn=NMC_MtGetEncPos(axishandle[0],&encpos);

//=====分割线=====
//以上为获取单轴的状态，你会发现，假如要同时获取多个轴的状态，那不是得 for 循环多个轴，
//其实不必如此麻烦，下面我们演示打包指令，一次性获取所有轴信息，包括 IO 等信息。
TAxisStsPack12Ex stspack;
rtn=NMC_MtGetStsPack12Ex(axishandle[0],&stspack);
//参考 TAxisStsPack12Ex 定义，我们可以看到控制器所有信息都读取上来了，只需解析 stspack 即可
```

五、点位和 JOG 运动

1、功能原理

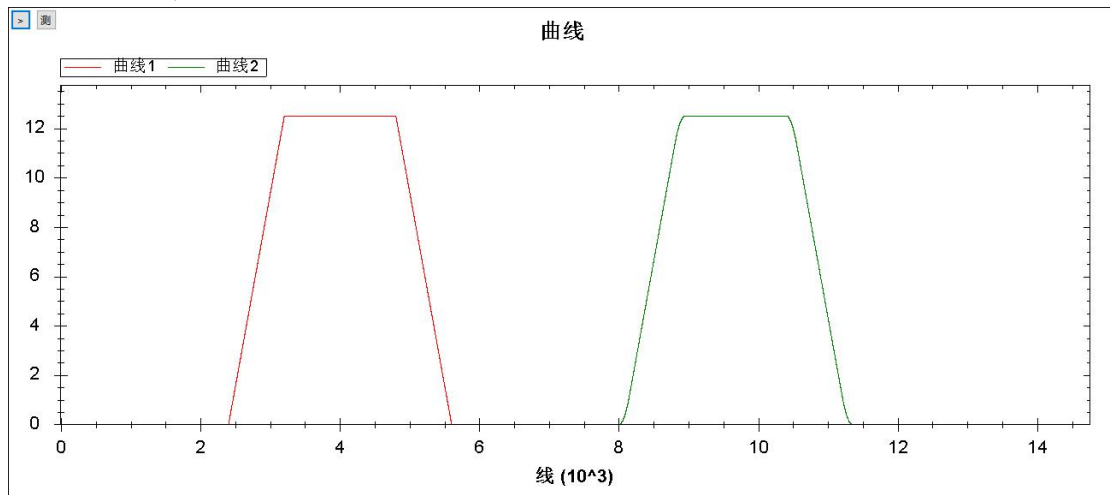
※本章节教会我们如何使轴运动起来。

※控制器的每个轴都可以设置为单独运动模式。

※单轴运动的速度曲线类型可以设置为 T 型曲线（PTP），恒速运动（JOG）。

※在点位运动模式下，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。

※点位运动或者 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）。



红色曲线平滑系数 0，绿色曲线平滑系数 150

2、指令说明

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

功能：设置单轴运动速度曲线类型

参数：

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 #define MT_NONE_PRF_MODE (-1) // 无效 #define MT_PTP_PRF_MODE (0) // 梯形规划 #define MT_JOG_PRF_MODE (1) // 连续速度模式 #define MT_CRD_PRF_MODE (3) // 坐标系

			<pre>#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> <p>注意：坐标系模式不需要通过 NMC_MtSetPrfMode 设置</p>
--	--	--	---

NMC_MtGetPrfMode (HAND axisHandle, short*mode)

功能：读取单轴运动速度曲线类型

参数参考 NMC_MtSetPrfMode

NMC_MtSetPtpPara(HAND axisHandle, TPtpPara *pAxPara)

功能：设置单轴 PTP 运动参数

参数：

名称	数据类型	输入/输出	描述
axisHandle	HAND	输入	轴句柄
pAxPara	TPtpPara *	输入	<p>设置单轴点到点运动参数，结构体定义如下</p> <pre>typedef struct { double acc; double dec; double startVel; double endVel; short smoothCoef; short reserved[3]; } TPtpPara;</pre> <p>acc:加速度, 脉冲/ms² dec:减速度, 脉冲/ms² 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</p>

NMC_MtGetPtpPara(HAND axisHandle, TPtpPara *pAxPara)

功能：设置单轴 PTP 运动参数

参数参考 NMC_MtGetPtpPara

NMC_MtSetPtpTgtPos(HAND axisHandle, long pos)

功能：设置点到点运动的目标位置

参数：

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
--	--	--	------------

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

功能：读取点到点运动的目标位置

参数参考 NMC_MtSetPtpTgtPos

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

功能：设置目标速度（最大速度）

参数：

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

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

功能：读取目标速度（最大速度）

参数参考 NMC_MtSetVel

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

功能：设置单轴 JOG（连续速度模式）运动参数

参数：

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

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

功能：设置单轴 JOG（连续速度模式）运动参数

参数参考 NMC_MtSetJogPara

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

功能：更新运动参数，只针对 PTP 和 Jog

参数：

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

axisHandle	HAND	输入	轴句柄
------------	------	----	-----

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

功能：多轴同时更新运动参数，只针对 PTP 和 Jog

参数：

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

[NMC_MtStop\(HAND axisHandle \)](#)

功能：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

参数：

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

[NMC_MtAbruptStop\(HAND axisHandle \)](#)

功能：单轴急停，不会置起急停标志位，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

参数：

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

[NMC_MtStopMulti\(HAND axisHandle \)](#)

功能：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

参数：

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

[NMC_MtEstp\(HAND axisHandle \)](#)

功能：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

参数：

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

注：调用急停后会将轴的急停状态字置起，需调用 [NMC_MtClrError\(HAND axisHandle \)](#) 清除状态才能继续运动。

[NMC_MtMovePtpAbs\(HAND axisHandle,double acc,double acc,double startVel,double endVel,double maxVel,short smoothCoef,long tgtPos\);](#)

功能：单轴点位运动（绝对位置）

参数:

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
tgtPos	long	输入	目标位置,单位: 脉冲

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

功能: 单轴点位运动 (相对位置)

参数:

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

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

功能: 点位运动打包 (绝对位置)

参数:

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：1.如果某轴存在无法运动的错误状态，则所有轴不会启动运动

[NMC_MtMovePtpRelPack8\(HAND axisFirstHandle,TMovePtpPack8 * pPackData\);](#)

功能：点位运动打包（相对位置）

参数：

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>

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

功能：单轴 JOG 运动

参数：

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

3、综合示例

3.1 点位运动

```

short rtn;

// 模式配置：将指定轴配置为点到点运动模式
rtn = NMC_MtSetPrfMode(axishandle[0],MT_PTP_PRF_MODE);

if (rtn!=0)
{
    return;
}

// 运动参数配置
TPtpPara 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[0],&ptpPrm);
if (rtn!=0)
{
    return;
}

// 设置规划的最高速度
double maxVel = 10;
rtn = NMC_MtSetVel(axishandle[0],maxVel);
if (rtn!=0)
{
    return;
}

// 设置目标位置
long targetPos = 10000;
rtn = NMC_MtSetPtpTgtPos(axishandle[0],targetPos);
if (rtn!=0)
{
    return;
}

```



```
}  
  
// 启动运动  
rtn = NMC_MtUpdate(axishandle[0]);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return;          // 启动运动出错  
}  
  
// 运动过程控制  
short velUpdateFlag = 0;  
short axisSts;  
while(true)  
{  
    rtn = NMC_MtGetSts(axishandle[0],&axisSts);  
    if((axisSts & BIT_AXIS_BUSY) == 0)  
    {  
        break;      // 运动完成  
    }  
}  
  
}
```

3.2 点位运动（绝对位置）

```
//动态库提供了一条函数，可以完成以上运动操作  
rtn=NMC_MtMovePtpAbs(axishandle[0],0.5,0.5,0,0,10,0,10000);  
//等待运动完成过程略。
```

3.3 点位运动（相对位置）

```
//在当前基础上往正方形移动个脉冲  
rtn=NMC_MtMovePtpRel(axishandle[0],0.5,0.5,0,0,10,0,10000);  
//等待运动完成过程略。
```

3.4 JOG 运动

```
// TODO: 在此添加控件通知处理程序代码  
short rtn;  
  
// 模式配置：将指定轴配置为 JOG 模式  
rtn = NMC_MtSetPrfMode(axishandle[0],MT_JOG_PRF_MODE);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return;          // 模式配置出错  
}  
  
// 运动参数配置  
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[0],&jogPrm);
if(rtn != RTN_CMD_SUCCESS)
{
    return;                // 运动参数配置出错
}

// 设置速度
double maxVel = 5;
rtn = NMC_MtSetVel(axishandle[0],maxVel);
if(rtn != RTN_CMD_SUCCESS)
{
    return;                // 设置规划的最高速度出错
}

// 启动运动
rtn = NMC_MtUpdate(axishandle[0]);
if(rtn != RTN_CMD_SUCCESS)
{
    return;                // 启动运动出错
}

// 运动过程控制
short velUpdateFlag = 0;
long axisPos;
while(true)
{
    // 读取电机当前的理论位置
    rtn = NMC_MtGetAxisPos(axishandle[0],&axisPos);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return;            // 出错
    }

    // 位置大于时，停止运动
    if(axisPos> 20000)
    {
        rtn = NMC_MtStop(axishandle[0]);
        if(rtn != RTN_CMD_SUCCESS)
        {
            return;        // 停止运动出错
        }
        break;
    }
}
```

```
}  
}
```

3.5 JOG 运动

```
//同样，JOG 运动也可以一条指令到达目的  
rtn=NMC_MtMoveJog(axishandle[0],0.5,0.5,5,0,0);  
// 运动过程控制省略
```

六、回零运动

1、功能原理

※我们整合了控制的各种回零方式，客户只需要调用 [NMC_MtSetHomePara](#) 和 [NMC_MtHome](#) 即可实现回零。

※[NMC_MtSetHomePara](#) 用于设置回零参数，[NMC_MtHome](#) 用于启动回零。

※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.EXE 软件先回零，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。

※需要提醒的是，回零前请先确认好轴的方向的正确的，即发正脉冲，轴会朝着正限位方向运动。此外，假如正负限位为低电平触发，则对应的回零结构体 THomeSetting 参数 lmtEdge 应为下降沿触发，反之则为上升沿触发。原点信息和电机 Z 向信号亦然。

※一些用户发现回零方向不对，明显让其往负向回零，轴却往正方形查找原点，原因很可能就是上升沿或者下降沿触发参数给错导致的。

※回零应保证轴的规划位置 and 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，轴应使得轴测试页面的实际位置和命令位置一致，见下图。



2、指令说明

1) 设置回零参数

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

功能：设置回零参数。

参数：

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

返回值：略

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

功能：读取回零参数。

2) 启动回零

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

功能：启动回零动作。回零完成后轴位置将自动清零。

参数：

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

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

功能：尝试性回零（测试回零误差，不清位置）

参数：

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

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

功能：终止回零动作。

参数：

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

3) 读回零状态

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

功能：读取回零状态。

参数：

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。参数宏定义 状态位定义如下： bit 0 ，该位为 1：回零中 bit 1 ，该位为 1：回零成功

			bit 2 , 该位为 1: 回零失败 bit 3 , 该位为 1: 回零错误, 运动参数出错导致不动 bit 4 , 该位为 1: 回零错误, 搜寻过程中开关没触发
--	--	--	---

4) 读新回零位置和历史回零位置的差值

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

功能：读取回零误差。

参数：

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

返回值：略

注：回零成功时才有意义。

3、回零参数详细说明

回零参数结构定义如下：

```
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 (两次搜寻时需要)
    double acc;           // 加速度
    unsigned char reScanEn; // 是否两次搜寻零点 (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 (必须)。

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)

两次回原点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)

11) 限位，触发沿 lmtEdge (默认下降沿，值为 0)

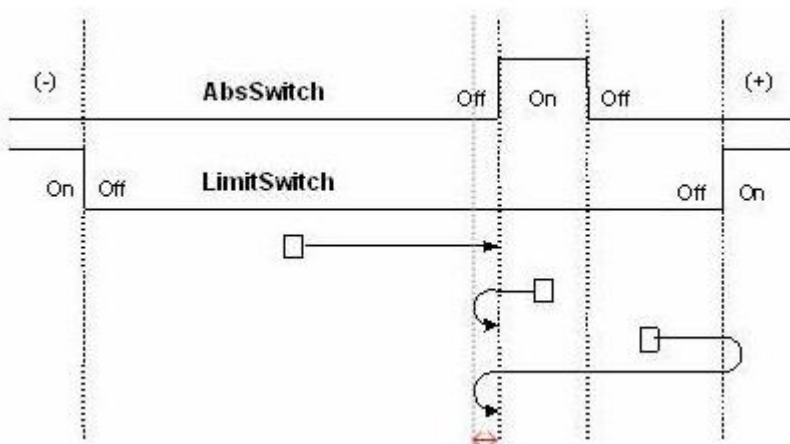
12) Z 相，触发沿 zEdge (默认下降沿，值为 0)

13) 扫描安全距离 safLen (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

4、回零图示

1) 以原点开关为参考 (MODE1)，一次搜寻原点



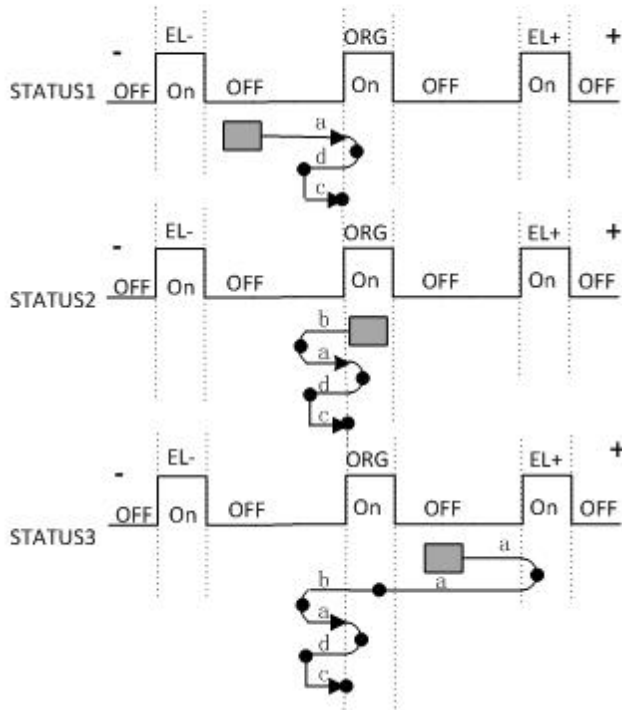
图中方块表示初始位置，曲线表示回零的过程。分别表示了 3 种不同初始位置时回零的过程。

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。

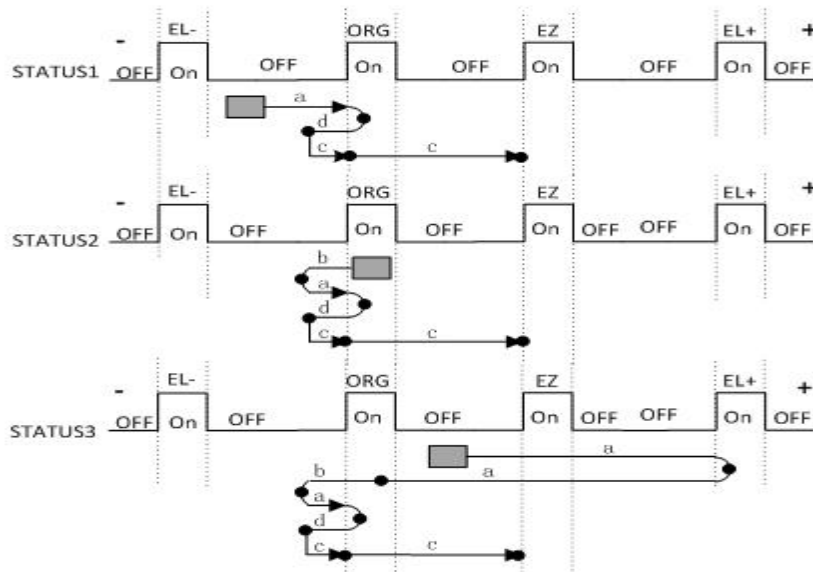
以限位为参考 `MODE2` 和 `MODE1` 类似（图略）。

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

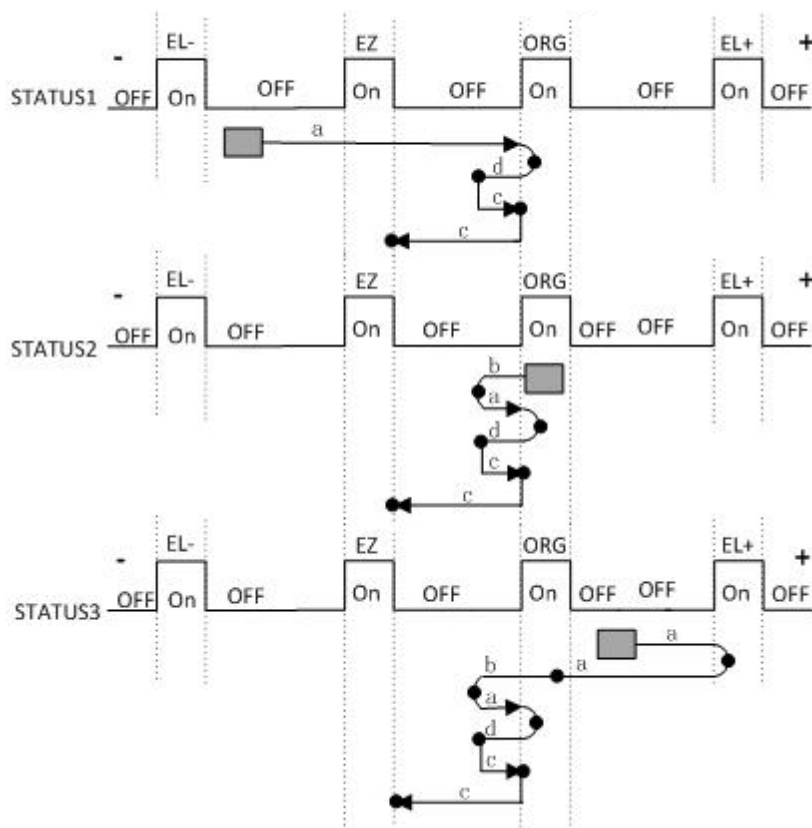
2) 以原点开关为参考（MODE1），两次搜寻原点



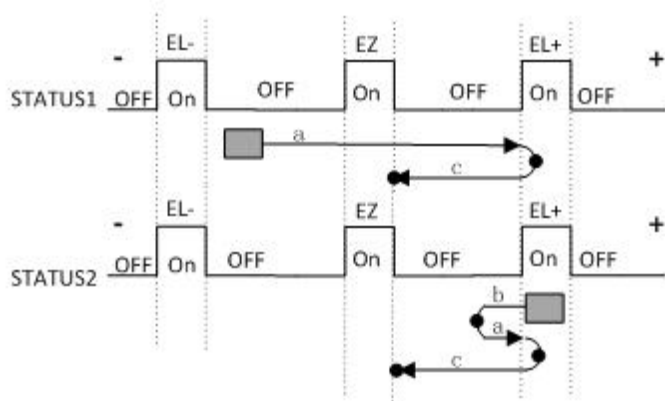
3) 以原点开关和+Z 相信号为参考（MODE4），两次搜寻原点



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



5、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。

```
short DevAxisHoming(HAND axisHandle)
{
    short rtn;

    // 回零参数设置（请根据自身机构特点配置）
    THomeSetting homeSetup;
```

```
homeSetup.mode = HM_MODE1;    // 回零模式
homeSetup.dir = 0;             // 回零初始方向:负向
homeSetup.offset = 0;         // 原点偏移
homeSetup.scan1stVel = 5;     // 基本搜寻速度
homeSetup.scan2ndVel = 1;     // 低速
homeSetup.acc = 0.5;          // 加速度
homeSetup.reScanEn = 1;       // 两次搜寻零点
homeSetup.homeEdge = 0;       // 原点，触发沿,下降沿
homeSetup.lmtEdge = 0;        //
homeSetup.zEdge = 0;          //
homeSetup.iniRetPos = 0;       // 起始反向运动距离
homeSetup.retSwOffset = 0;     // 反向运动时离开开关距离
homeSetup.safeLen = 0;        // 安全距离
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
rtn = NMC_MtSetHomePara(axisHandle,&homeSetup);
if(rtn != RTN_CMD_SUCCESS)
{
    return rtn;    // 参数配置出错
}
// 启动回零
rtn = NMC_MtHome(axisHandle);
if(rtn != RTN_CMD_SUCCESS)
{
    return rtn;    // 启动出错
}

// 回零过程查询
short homeSts;
while(true)
{
    rtn = NMC_MtGetHomeSts(axisHandle,&homeSts);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return rtn;
    }
    if((homeSts & BIT_AXHOME_OK) != 0)
    {
        break;    // 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 0;
}
```

6、注意事项

- 1) 以限位为回零信号时，可设置是否允许限位急停，以及急停的加速度。
- 2) 对于脉冲输出的控制器，默认是以脉冲计数作为位置参考。如果使用外部编码器，则需要在回零前正确设置编码器的工作模式，并执行位置清零。
- 3) 回零功能不对伺服允许信号进行控制。如果驱动器需要伺服允许信号，则应在回零前，打开伺服。
- 4) 选择 MODE1，MODE4，MODE5 回零时，如果先碰到限位会反向继续寻找原点。

七、插补运动

1、功能介绍

※控制器提供单轴运动和多轴联动（坐标系）功能。

※每个轴可以配置成单轴方式或坐标系（插补）方式。同一个轴同一时刻只能属于其中一种运动模式。

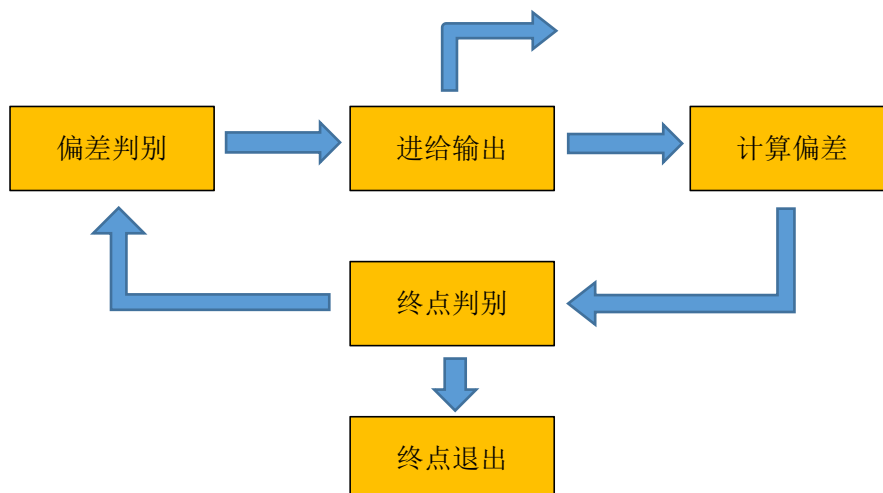
※在接口函数里，有三种编程对象，分别是控制器，单轴和坐标系。对于单轴控制的指令 是以 NMC_Mt 开头，对坐标系的是以 NMC_Crd 开头，其余是对控制器的指令。

※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-->启动插补运动-->查询坐标系状态等待运动完成。

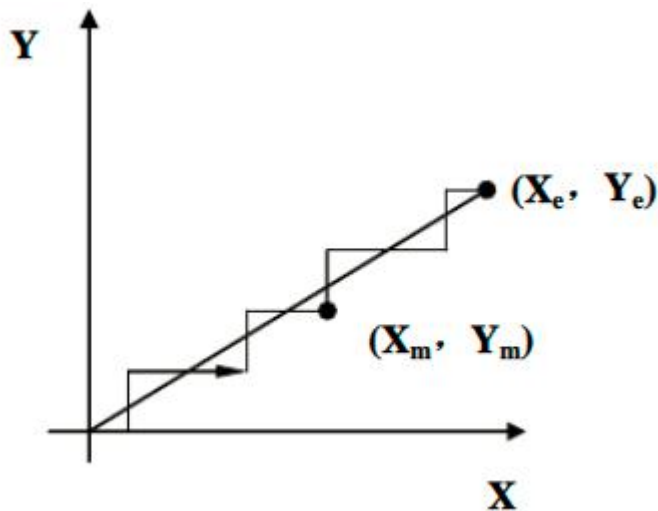
2、插补概念及原理介绍

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。

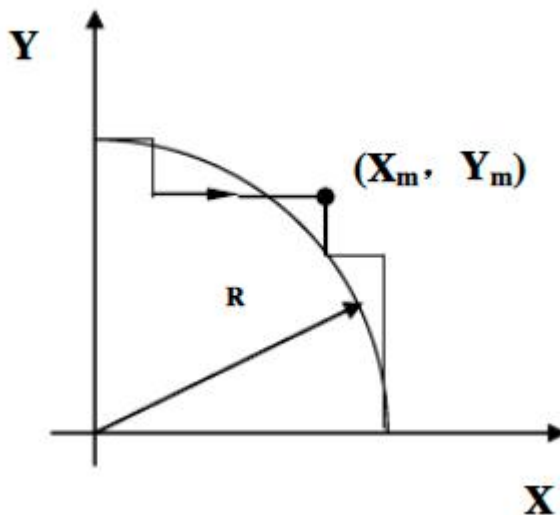


直线插补逼近示意图：



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。

3、坐标系初始化

坐标系初始化之前需要调用 [NMC_CrdOpen](#) 获取坐标系句柄。

3.1、指令说明

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

功能：建立插补坐标系系统(必须)

参数：

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： 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;

注：1.目前坐标系最多支持 4 个轴

2.轴映射表中轴号取值范围为[0,n]

3.端口均设为 0

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

功能：读取插补坐标系系统的配置

参数参考 NMC_CrdConfig

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

功能：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

参数：略

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

功能：设置坐标系参数

参数：

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： 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;

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

功能：读取坐标系参数

参数参考 [NMC_CrdSetPara](#)

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

功能：设置坐标系高级参数

参数：

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： 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; //（默认accMax） double abruptDec; //（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;

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

功能：读取坐标系参数

参数参考 [NMC_CrdGetExtPara](#)

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

功能：设置圆弧插补参数（高级指令）

参数：

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], 对应 err2 }TArcSecSetting;</pre>

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

功能：读圆弧插补参数（高级指令）

参数略。

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

功能：设置向心加速度限制

参数：

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

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

功能：设置 4 维插补下，A 轴的最大容忍转弯速度，若大于该速度，则需要进行降速处理

参数：

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

3.2、代码示例

```
short rtn=0;
//打开坐标系
//坐标系的建立
rtn = NMC_CrdOpen(devhandle, & crdhandle);
//配置坐标系
TCrdConfig crd;
TCrdPara crdPara;//运动参数
crd.axCnts = 3; //3 轴坐标系
crd.pAxArray[0]=0; //端口 0 对应 X 轴
crd.pAxArray[1] = 1; //端口 1 对应 Y 轴
crd.pAxArray[2] = 2; //端口 2 对应 Z 轴
crd.pAxArray[3] = -1;
crd.port[0] = 0;
crd.port[1] = 0;
crd.port[2] = 0;
crd.port[3] = 0;
rtn =NMC_CrdConfig(crdhandle, & crd);
crdPara.orgFlag = 1;//默认原点(0,0,0,0)
crdPara.offset[0] = 0;
crdPara.offset[1] = 0;
crdPara.offset[2] = 0;
crdPara.offset[3] = 0;
crdPara.synAccMax = 50;//最大合成加速度 p/ms^2,加速度为已经是比较大的加速度了
crdPara.synVelMax = 800;//最大合成速度,在轴允许范围内, 这两个参数可以尽量大一些
rtn =NMC_CrdSetPara(crdhandle, & crdPara);
//////////以下配置如需必要, 可以不配置
=====//////////
//设置安全参数
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);
//清除坐标系错误状态
rtn =NMC_CrdClrError(crdhandle);
//压入指令之前需要清空缓存区
rtn=NMC_CrdBufClr(crdhandle);
```

4、直线插补和平面圆弧插补

4.1、指令说明

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

功能：直线插补（带前瞻开关）

参数：

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
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	最大速度, 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)

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

功能：四轴直线插补

参数：

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
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	最大速度, 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)

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

功能：XY 平面圆弧插补：终点位置、圆心、方向

参数：

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
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		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	最大速度, 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）

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

功能：YZ 平面圆弧插补：终点位置、圆心、方向

参数：

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
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		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	最大速度, 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）

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

功能：ZX 平面圆弧插补：终点位置、圆心、方向

参数:

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
pTgPosArray	long *	输入	目标位置 (二维数组,分别表示 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		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	最大速度, 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)

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

功能: XY 平面圆弧插补: 终点位置、半径、方向

参数:

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
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
circleDir	short		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	最大速度, 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)

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

功能: YZ 平面圆弧插补: 终点位置、半径、方向

参数:

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
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
circleDir	short		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	最大速度, 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)

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

功能: ZX 平面圆弧插补: 终点位置、半径、方向

参数:

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
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
circleDir	short		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	最大速度, 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)

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

功能: XY 平面圆弧插补: 起点 (当前点)、中点、终点

参数:

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
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	最大速度, 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）

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

功能：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）

参数：

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
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的
vel	double	输入	最大速度, 单位 pulse/ms
synAcc	double	输入	合成加速度, 单位 pulse/ms^2

4.2、代码示例

```

short rtn=0;
long pos1[3];
pos1[0]=10000;
pos1[1]=10000;
pos1[2]=0;
//本例程默认使用前瞻，lookaheadDis=0;
//压入直线
rtn=NMC_CrdLineXYZEx(crdhandle,0,7,pos1,0,10,0.5,0);
//压入直线
pos1[0]=0;

```

```
pos1[1]=0;
pos1[2]=0;
rtn=NMC_CrdLineXYZEx(crdhandle,0,7,pos1,0,10,0.5,0);
//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（10000，0）的半圆,顺时针
tgpos[0]=10000;
tgpos[1]=0;
rtn=NMC_CrdArcRadiusEx(crdhandle,2,tgpos,0,1,0,10,1,0);
//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);
```

5、多维(8)插补指令

5.1、指令说明

[NMC_CrdLineXYZD8\(HAND crdHandle, long segNo, long crdAxmask, long extAxMask, long *pTgPosArray,double endVel, double vel,double synAcc,short lookaheadDis \);](#)

功能：多轴直线插补（最多支持 8 轴）

参数：

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
crdAxmask	double	输入	坐标系中参与的轴,按位表示
extAxMask	long *	输入	其他参与轴,按位表示,不能包括坐标系中
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	最大速度, 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)

[NMC_CrdLineXYZD8Pack\(HAND crdHandle, short count, TCrdLineXYZD8Unit *pCmdArray\);](#)

功能: 打包的多轴直线插补

参数:

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 typedef struct { long segNo; // 用户自定义段号 short crdAxMask; short extAxMask; long tgPos[8]; // 目标位置 float vel; // 最大速度 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;

5.2、代码示例

这里给出一个 XYZ 轴运动, 567 轴也参与的例程。

```
short rtn=0;
long tgpos2[6];
tgpos2[0]=10000;
tgpos2[1]=10000;
tgpos2[2]=10000;
tgpos2[3]=10000;
tgpos2[4]=10000;
tgpos2[5]=10000;
//XYZ,456 轴一起跑到 (10000,10000,10000,10000,10000,10000) ,
```



```
rtn=NMC_CrdLineXYZD8(crdhandle,4,0x7,0x38,tgpos2,0,10,1,0);
//XYZ,456 轴一起跑到 (0,10000,0,0,10000,0)
tgpos2[0]=0;
tgpos2[1]=10000;
tgpos2[2]=0;
tgpos2[3]=0;
tgpos2[4]=10000;
tgpos2[5]=0;
rtn=NMC_CrdLineXYZD8(crdhandle,5,0x7,0x38,tgpos2,0,10,1,0);
//下面介绍打包指令,我们一次性压入条指令
TCrdLineXYZD8Unit pack[10];
//这里只做一个简单的赋值示例
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);
//可以预测到轴最终跑到 (9000, -9000, 9000, -9000, 9000, -9000)
```

6、螺旋线插补

6.1、指令说明

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

功能：螺旋线插补

参数：

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
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	最大速度，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）

6.2、代码示例

```

short rtn=0;

long pos4[3];
pos4[0]=0;
pos4[1]=0;
pos4[2]=0;
//压入直线运动到（0，0，0）
rtn=NMC_CrdLineXYZEx(crdhandle,15,7,pos4,0,10,0.5,0);

long cenpos4[2];
cenpos4[0]=5000;
cenpos4[1]=0;

pos4[0]=0;
pos4[1]=0;
pos4[2]=20000;
//圆心（5000，0）高度20000，总共旋转了5圈
rtn=NMC_CrdHelixCenterEx(crdhandle,16,pos4,cenpos4,0,5,0,10,1,0);

```

7、空间圆弧插补

7.1、指令说明

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

功能：3D 圆弧插补

参数：

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
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	最大速度，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）

[NMC_CrdCircle3DEx\(HAND crdHandle,long segNo, long *pMidPos, long *pTgPos,double velEnd, double vel,double synAcc ,short lookaheadDis\);](#)

功能：3D 圆弧插补(整圆)

参数：

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
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	最大速度，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、代码示例

```
short rtn=0;

long pos5[3];
pos5[0]=0;
pos5[1]=0;
pos5[2]=0;
//压入直线运动到（0，0，0）
rtn=NMC_CrdLineXYZEx(crdhandle,17,7,pos5,0,10,0.5,0);
pos5[0]=0;
pos5[1]=0;
pos5[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,pos5,0,10,1);
```

8、缓存区的操作（IO、延时、单轴操作等）

8.1、指令说明

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

功能：缓冲区 DO

参数：

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
doType	short	输入	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>#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]

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

功能：缓冲区 DO

参数：

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
group	short	输入	Do 类型, 见宏定义 // 缓冲区输出 DO 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>
doMask	long	输入	位掩码
doValue	long	输入	输出值

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

功能：缓冲区 DO(模拟量和 PWM)

参数：

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
group	short	输入	组别, 见宏定义 // 通用输出宏类型 <pre>#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]
-------	------	----	-----------------

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

功能：缓冲区 DI 等待

参数：

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
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

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

功能：缓存区 DI 急停

参数：

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
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]

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

功能：缓存区延时

参数：

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
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

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

功能：缓冲区单轴移动(带参数)

参数：

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
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：阻塞模式为，本条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时开始运行。同步模式下一条插补指令必需有位移。

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

功能：缓冲区单轴移动(相对位移移动)

参数：

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
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

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

功能：缓冲区单轴移动参数设置

参数：

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
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

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

功能：设置跟随运动前的运动补偿量

参数：

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
axisNo	short	输入	轴号, [0, n]
relDistance			相对补偿位移量
vel	double	输入	补偿速度 pulse/ms

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

功能：缓冲区等待电机运动到位

参数：

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
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			等待到位超时的时间，单位：ms

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

功能：缓冲区等待电机运动到位

参数：

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
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

8.2、代码示例

```
short rtn=0;
long pos6[3];
pos6[0]=0;
pos6[1]=0;
pos6[2]=0;
```



```
//压入直线运动到 (0, 0, 0)
rtn=NMC_CrdLineXYZEx(crdhandle,19,7,pos6,0,10,0.5,0);
//运行到 (10000, 10000, 0) 后输出一个 DO
pos6[0]=10000;
pos6[1]=10000;
pos6[2]=0;
rtn=NMC_CrdLineXYZEx(crdhandle,20,7,pos6,0,10,0.5,0);
rtn=NMC_CrdBufDo(crdhandle,21,3,0,0);
pos6[0]=0;
pos6[1]=0;
pos6[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,pos6,0,10,0.5,0);
rtn=NMC_CrdBufDelay(crdhandle,22,0,1000);
rtn=NMC_CrdBufDo(crdhandle,23,3,0,1);
pos6[0]=10000;
pos6[1]=10000;
pos6[2]=0;
//直线运动到 (10000, 10000, 0)
rtn=NMC_CrdLineXYZEx(crdhandle,24,7,pos6,0,10,0.5,0);
//直线运动到 (0, 0, 0),同时让轴同步运动到的位置 10000
pos6[0]=0;
pos6[1]=0;
pos6[2]=0;
long posex[1];
posex[0]=10000;
rtn=NMC_CrdBufAxMoveEx(crdhandle,25,0x8,posex,10,1,0,1);
rtn=NMC_CrdLineXYZEx(crdhandle,26,7,pos6,0,10,0.5,0);
```

9、打包指令介绍

※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。

9.1、指令说明

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

功能：打包插补数据

参数：

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 #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 DO 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 #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
dataLen	short	输入	数据长度

注：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，总的长度不超过 1000 个字节。

8.2、代码示例

```
short rtn=0;

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 crdBuLine;
    memset(&crdBuLine, 0, sizeof(crdBuLine));
    crdBuLine.segNo = i + 1;
    crdBuLine.tgPos[0] = (i + 1) * 2000;
    crdBuLine.tgPos[1] = (i + 1) * 1000;
    crdBuLine.tgPos[2] = 0;
    crdBuLine.endVel = 4;
    crdBuLine.vel = 10;
    crdBuLine.synAcc = 1;
    crdBuLine.mask = 0x3;
    crdBuLine.lookaheadDis = 0;
    memcpy(&bufData[offset], &cmdType, sizeof(short));
    offset += sizeof(short);
    memcpy(&bufData[offset], &crdBuLine, sizeof(TCrdBufLine));
    offset += sizeof(TCrdBufLine);
    if (offset >=0)
    {
        rtn = NMC_CrdBufDataPack(crdhandle, bufData, offset);
        if (rtn != 0)
        {
            //指令压入错误
            return;
        }
        memset(bufData, 0, sizeof(bufData));
        offset = 0;
    }
}

if (offset>0)
{
    rtn = NMC_CrdBufDataPack(crdhandle, bufData, offset);
}
```

9、坐标系状态检测

9.1、指令说明

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

功能：读取坐标系状态

参数：

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 #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>

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

功能：读取规划位置 XYZ

参数：

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

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

功能：坐标系模式下, 读取多个轴的机械坐标位置

参数：

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

NMC_CrdGetVel(HAND crdHandle,double *pVel);

功能：获取坐标系合成速度

参数：

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

NMC_CrdGetEncPos (HAND crdHandle, short cnts, long *pPosArray);

功能：获取坐标系合成速度

参数：

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

NMC_CrdGetStsPack4(HAND crdHandle, TPackedCrdSts4 *pPackSts);

功能：坐标系运动模式下,打包读取控制器状态

参数：

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>

NMC_CrdBufGetFree (HAND crdHandle, long *pRes);

功能：读取指令缓冲区已用长度

参数：

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

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

功能：读取指令缓冲区已用长度

参数：

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

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

功能：读段号

参数：

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

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

功能：读取总共压了多少条指令

参数：

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

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

功能：读取内部坐标系状态

参数：

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, 伺服模式时实际位置到达误差限 #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)

			<pre> #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>
--	--	--	--

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

功能：获取插补缓冲区中尚未完成的总位移量

参数：

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

9.2、代码示例

坐标系状态检测常用于获取坐标系得运动状态，运动是否完成，运行中是否出错等，这个例程我们将联合下一节一起向大家展示。

10、启动坐标系运动等其他指令

10.1、指令说明

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

功能：坐标系缓冲运动启动

参数：

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

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

功能：立即平滑停止运动

参数：

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

注：立即停止运动。并不清空指令缓冲区。需要再次启动才能继续运行缓冲区指令。

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

功能：清坐标系运动错误状态,同时清除所包含轴的错误状态

参数:

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

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

功能: 指令缓冲区清空

参数:

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

[NMC_CrdSetOverRide\(HAND crdHandle, double overRide \);](#)

功能: 设置坐标系速度倍率

参数:

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)

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

功能: 设置坐标系速度倍率

参数略。

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

功能: 设置坐标系速度倍率

参数:

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

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

功能: 急停,不清空指令缓冲区

参数:

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

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

功能: 返回缓冲区运动的断点

参数:

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

NMC_CrdStartExeTimeCalc(void);

功能：开始计算缓冲区执行时间

参数：略

注：1.如果需要计算执行时间，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。

NMC_CrdGetExeTime(double *pTime);

功能：读取缓冲区指令的执行时间，并停止计算，单位:ms

参数：

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

NMC_CrdSetBufLengthFlag(HAND crdHandle, short flag);

功能：设置是否计算所有线段长度

参数：

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

10.2、代码示例

```
//前面的介绍中我们已经建立并初始化了坐标系，并且压入了一些插补指令
```

```
//接下来我们将启动插补运动，并获取插补运动的状态
```

```
short rtn=0;
```

```
short crdsts=0;
```

```
double vel=0;
```

```
long segnum=0;
```

```
long fre=0;
```

```
rtn=NMC_CrdEndMtn(crdhandle);
```

```
rtn=NMC_CrdStartMtn(crdhandle);
```

```
if (rtn!=0)
```

```
{
```

```
    //坐标系运动启动失败
```

```
    return;
```

```
}
```

```
while (true)
```

```
{
```

```
    //获取插补速度
```

```
    rtn=NMC_CrdGetVel(crdhandle,&vel);
```

```
    //获取正在运行的段号
```

```
    rtn=NMC_CrdGetUserSegNo(crdhandle,&segnum);
```

```
//获取缓存区剩余空间
rtn=NMC_CrdBufGetFree(crdhandle,&fre);
//读坐标系状态
rtn=NMC_CrdGetSts(crdhandle,&crdsts);
if(rtn!=0)
{
    //指令报错
    return;
}
if ((crdsts&BIT_CORD_BUSY)==0)
{
    //运动完成
    break;
}
if ((crdsts&BIT_CORD_MVERR) ==1)
{
    //运动出错，做一些处理，此处略
}
}
```

11、综合示例

前面是一些详细的叙述，下面我们来介绍一个完整的使用例程。

测试时请注意速度和位置在安全的范围内，以免发生意外。

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 DevCrdMove(HAND devHandle)
{
    short rtn;
    HAND crdHandle;
    // 创建 CRD 操作句柄
    rtn = NMC_CrdOpen(devHandle,&crdHandle);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return rtn;    // 指令出错
    }

    // Crd 系统配置
    TCrdConfig crdCfg;
    crdCfg.axCnts = 3;    // 三轴
    crdCfg.pAxArray[0] = 0;    // 坐标系运动 X 轴为第一轴
}
```

```
crdCfg.pAxArray[1] = 1;           // 坐标系运动 Y 轴为第二轴
crdCfg.pAxArray[2] = 2;           // 坐标系运动 Z 轴为第三轴
crdCfg.pAxArray[3] = -1;         //
crdCfg.port[0] = 0;              // 端口为
crdCfg.port[1] = 0;              // 端口为
crdCfg.port[2] = 0;              // 端口为
crdCfg.port[3] = 0;              // 端口为
rtn = NMC_CrdConfig(crdHandle,&crdCfg);
if(rtn != RTN_CMD_SUCCESS)
{
    return rtn;           // 指令出错
}

// Crd 系统参数配置
TCrdPara crdPrm;
crdPrm.orgFlag = 1;           // 设置坐标系原点
crdPrm.synAccMax = 10;        // 插补最大合成加速度
crdPrm.synVelMax = 50;        // 插补最大合成速度
crdPrm.offset[0] = 0;         // 偏移量为
crdPrm.offset[1] = 0;         // 偏移量为
crdPrm.offset[2] = 0;         // 偏移量为
crdPrm.offset[3] = 0;         // 偏移量为

rtn = NMC_CrdSetPara(crdHandle,&crdPrm);
if(rtn != RTN_CMD_SUCCESS)
{
    return rtn;           // 指令出错
}

// 清空缓冲区数据
rtn = NMC_CrdBufClr(crdHandle);
if(rtn != RTN_CMD_SUCCESS)
{
    return rtn;           // 指令出错
}

// 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
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); //运动到(0,0,0)
if(rtn != RTN_CMD_SUCCESS)
```

```
{  
    return rtn;    // 指令出错  
}  
targetPos[0] = 10000;  
targetPos[1] = 0;  
rtn = NMC_CrdArcCenter(crdHandle,segNo++,targetPos,centerPos,1,0,vel,acc); //圆弧插补  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return rtn;    // 指令出错  
}  
rtn = NMC_CrdBufDelay(crdHandle,segNo++,CRD_BUFF_DELAY_SCALE_MS,100);// 延时毫秒  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return rtn;    // 指令出错  
}  
rtn = NMC_CrdBufDo(crdHandle,segNo++,CRD_BUFF_DO_GPDO1,1,0); // 输出 DO  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return rtn;    // 指令出错  
}  
  
// 结束数据压入  
rtn = NMC_CrdEndMtn(crdHandle);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return rtn;    // 指令出错  
}  
  
// 启动缓冲区运动  
rtn = NMC_CrdStartMtn(crdHandle);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return rtn;    // 指令出错  
}  
  
// 等待运动结束  
short crdSts;  
while(true)  
{  
    rtn = NMC_CrdGetSts(crdHandle,&crdSts);  
    if(rtn != RTN_CMD_SUCCESS)  
    {  
        return rtn;  
    }  
}
```

```
        if((crdSts&BIT_CORD_BUSY) == 0)
        {
            break;    // 运动完成
        }
    }

    return 0;
}
```

八、IO 资源访问

1、功能原理

※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；

※本章介绍如何在应用程序中使用这些硬件资源。

2、指令说明

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

功能：设置通用输出(按通道,支持超过 32 位)，此函数与 [NMC_SetDO](#) 功能相同，可任选一个来设置控制器通用输出状态。

参数：

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	DO 组，取值范围[0,n],0: 本地 DI31~DI0, 1: 本地 DI63~DI32，其他指扩展 IO 模块
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。

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

功能：按位设置通用输出

参数：

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	unsigned long	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。

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

功能：读取通用输出(按通道,支持超过 32 位)，此函数与 [NMC_GetDO](#) 功能相同。

参数:

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	DO 组, 取值范围[0,n],0: 本地 DI31~DI0, 1: 本地 DI63~DI32, 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。

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

功能: 读通用输入(按通道,支持超过 32 位) , 此函数与 [NMC_GetDI](#) 功能相同。

参数:

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 DI31~DI0, 1: 本地 DI63~DI32, 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

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

功能: 按位读取通用输入

参数:

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

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

功能: 读取轴专用 IO

参数:

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

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

功能：读运动控制专用 IO,逻辑电平

参数：

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。对应位为 0 为低电平,1 为高电平

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

功能：按位设置通用输入信号取反

参数：

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

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

功能：按位读取通用输入信号是否取反

参数：

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

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

功能：按位设置通用输出信号取反

参数：

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

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

功能：按位读取通用输出信号取反

参数：

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
-------	---------	----	-------------------

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

功能：按位设置输出信号取反，

参数：

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	<pre>#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</pre>
group	short	输入	组号，暂时只对 GPO 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

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

功能：按位读取输出信号取反

参数：

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	<pre>#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</pre>
group	short	输入	组号，暂时只对 GPO 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

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

功能：DO 输出定时脉冲

参数：

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

NMC_GetIOModuleSts(HAND devHandle, unsigned long *sts);

功能：读取扩展 IO 模块的状态

参数：

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	io 模块状态

NMC_IOModuleSetEn(HAND devHandle, unsigned char chDevId,short chDevType);

功能：设置扩展 IO 模块有效(带模块类型)

参数：

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	输入	模块类型，见宏定义 // 扩展模块类型定义 #define IOMODULE_TYPE_I064 1 // 32DI32DO 模块（包括 16DI16DO 模块） #define IOMODULE_TYPE_I032_DA 2 // 4AD4DA 模块

NMC_IOModuleGetType(HAND devHandle, unsigned char chDevId,short *pChDevType);

功能：读取扩展 IO 模块类型

参数：

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

NMC_MtSetSvOn(HAND axisHandle);

功能：设置伺服 ON, 轴静止时执行,如果后面是 update 指令,需要延时一个周期

参数:

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

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

功能: 设置伺服 OFF, 轴静止时执行, 如果后面是 update 指令, 需要延时一个周期

参数:

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

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

功能: 设置伺服报警清除输出的状态

参数:

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 (输出低电平), 1, 无效 (输出高电平)

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

功能: 设置 DAC (模拟量输出) 通道的模式

参数:

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

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

功能: 获取 DAC (模拟量输出) 通道的模式

参数:

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

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

功能: 设置 Adc 参数

参数:

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

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

功能: 读取 Adc 参数

参数:

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

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

功能: 设置 DAC (模拟量输出) 输出值

参数:

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

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

功能: 获取 DAC (模拟量输出) 输出值

参数:

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

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

功能: 获取 ADC (模拟量输入) 输入值

参数:

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

NMC_SetDioMapping(HAND devHandle,TDioMappingCfg *pDioCfg)

功能：增加一组映射，映射关系目前最多存在 8 组

参数：

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;

NMC_GetAllDioMapping(HAND devHandle,TDioMappingCfg *pDioCfg)

功能：获取所有的 DIO 映射数据

参数：

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 { 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;

			}TDioMappingCfg;
--	--	--	------------------

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

功能：清除所有的DIO映射关系

参数略。

3、综合示例

```
//数字量输入输出
//获取本地通用数字量输入
short rtn=0;
long divaule=0;
long divalueex=0;
long dovalue=0;
long dovalueex=0;
short di0=0;
short di64=0;
long axis1=0;
bool IsExmoNeed=false;
//如果需要读拓展模块
if (IsExmoNeed==true)
{
    //设备ID 需要从2开始，多个拓展模块，依次增加，拨码开关也要拨码成对应ID
    rtn=NMC_IOModuleSetEn(devhandle,2,1);
}
//读取全部本地通用数字量输入
rtn=NMC_GetDIGroup(devhandle,&divaule,0);
//按位读取（单个读取），这里示范读第0个输入
rtn=NMC_GetDIBit(devhandle,0,&di0);
//全部读取拓展模，这里读一个模块的
rtn=NMC_GetDIGroup(devhandle,&divalueex,2);
//按位读取（单个读取），读拓展模块第0个输入
rtn=NMC_GetDIBit(devhandle,64,&di64);

//读取本地全部通用数字量输出
rtn=NMC_GetDOGroup(devhandle,&dovalue,0);
//读拓展模块
rtn=NMC_GetDOGroup(devhandle,&dovalueex,2);
//设置通用数字量输出
//全部输出
rtn=NMC_SetDOGroup(devhandle,0,0);
//第1个输出
rtn=NMC_SetDOGroup(devhandle,0xFFFF,0);
```

```
//第 0 个输出,按位输出
rtn=NMC_SetDOBit(devhandle,0,0);

//专用 IO, 如限位原点的电平读取
rtn=NMC_MtGetMotionIO(axishandle[0],&axis1);
if ((axis1&(1<<0))!=0)
{
    //负限位为高电平
}
if ((axis1&(1<<1))!=0)
{
    //正限位为高电平
}
if ((axis1&(1<<2))!=0)
{
    //原点为高电平
}

//模拟量
//设置
rtn=NMC_SetDacMode(devhandle,0,4);//默认, +-10V, 此函数可省略
//输出满值
rtn=NMC_SetDac(devhandle,0,32767);
//读取模拟输出
short dac=0;
rtn=NMC_GetDac(devhandle,0,&dac);
//读取
short adc=0;
rtn=NMC_SetAdcMode(devhandle,0,4,0);//默认, +-10V, 此函数可省略
rtn=NMC_GetAdc(devhandle,0,&adc);
```

九、手脉功能

※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手脉端口

1、指令说明

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

功能：启动手轮

参数：

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

NMC_SetHandWheelInput(HAND devHandle,short index);

功能：选择手轮跟随的编码器通道

参数：

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。

NMC_SetHandWheelRatio(HAND devHandle,double ratio);

功能：设置手轮跟随的倍率

参数：

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,..]

NMC_ClrHandWheel(HAND devHandle);

功能：设置手轮跟随的倍率

参数：

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

2、综合示例

这里演示轴 1 跟随手脉运动。

```
short rtn=0;
//为了方便切换，这里先关闭所有手脉
rtn=NMC_ClrHandWheel(devhandle);
//选择主轴为辅助编码器通道
rtn=NMC_SetHandWheelInput(devhandle,256);
if (rtn!=0)
{
```



```

//指令报错
}
//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);
if(rtn!=0)
{
//指令报错
}

```

十、龙门功能

1、指令说明

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

功能：设置龙门主动轴

参数：

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]

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

功能：设置龙门从动轴

参数：

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,...)

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

功能：关闭龙门

参数：

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]

2、综合示例

```

//设置龙门主轴,组号为

```

```
short rtn=NMC_SetGantryMaster(axishandle[0],0);  
//设置龙门组的从轴为轴，允许误差为 2000 脉冲  
rtn=NMC_SetGantrySlave(axishandle[1],0,2000);
```

3、注意事项

※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，即编码器反馈至方向相同，大小相若。

第三章 高级功能

一、位置比较输出功能

1、多维位置比较输出

1.1、功能原理

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，可视为 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	输出方式，0：脉冲，1：电平
outChnType	输出通道类型：0：GPO，1：Gate 信号
outIndex	输出通道序号，对于 GPO 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）：



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）：



1.2、指令说明

(1) 设置多维位置比较参数

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

参数:

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 dimens; //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: 脉冲: 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]

(2) 获取多维位置比较参数

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

参数:

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

参数:

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

功能: 设置多维位置比较的数据点

参数:

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: 若是 1 维比较, 则 pPosArray 传入一维数组地址, 若是 2 维比较, 则 pPosArray 应传入 2 维数组地址
count	short	输入	比较的数据点数, 注: 若为 2 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\(HAND devHandle, short onOff, short chn\);](#)

参数:

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

参数:

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]

1.3、综合示例

a. 配置参数

```
short rtn=0;

// devhandle 为控制器控制器句柄, 设置二维比较输出 group0 的参数
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 DOO
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);
if(rtn != 0){ return ;}
rtn = NMC_CompXDimensSetCmpOutMode(devhandle,0,0);
if(rtn != 0){ return ;} // 设置为同时输出模式
```

b.下载位置比较数组

```
//插入数据
// 写位置比较数组，个点
int xdPos[1024];
for (int i=0;i<10;i++)
{
    xdPos[i] = (i + 1)*10000; // X 轴位置
}
rtn = NMC_CompXDimensSetData(devhandle,(long *)xdPos,10,0);
if(rtn != 0){ return ;}

// 启动位置比较输出
rtn = NMC_CompXDimensOnoff(devhandle,1,0);
if(rtn != 0){ return ;}
```

c.开始比较输出

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

```
// 写位置比较数组，个点
int xdPos[1024];
for (int i=0;i<10;i++)
{
    xdPos[i] = (i + 1)*10000; // X 轴位置
}
rtn = NMC_CompXDimensSetData(devhandle,(long *)xdPos,10,0);
if(rtn != 0){ return ;}
```

```
// 启动位置比较输出
rtn = NMC_CompXDimensOnoff(devhandle,1,0);
if(rtn != 0){ return ;}
```

d. 读取位置比较输出状态

```
// 读取位置比较状态
short cmpSts,cmdOutput;
rtn = NMC_CompXDimensStatus(devhandle,&cmpSts,&cmdOutput,0);
if(rtn != 0){ return ;}
```

2、一维高速位置比较

2.1、功能原理

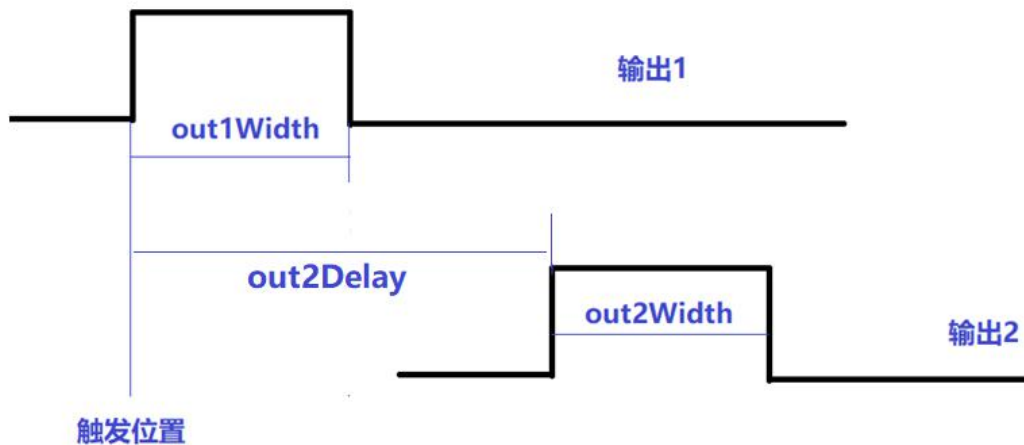
一维高速位置比较输出功能可以支持 4 路。相关的参数说明如下，通过 [NMC_CompHs1DimensSetParam](#) 接口配置：

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：



2.2、指令说明

(1) 设置高速位置比较的参数（比较编码器）

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

参数:

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） 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) 获取高速位置比较参数

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

参数:

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

参数:

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

参数:

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

参数:

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]

2.3、综合示例

a. 配置参数

```
short rtn=0;

// g_hDev 为控制器句柄，设置一维比较输出 group0 的参数
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);  
if(rtn != 0){ return ;}
```

b. 下载位置比较数组

```
// 写位置比较数组，8 个点  
int hs1dPos[8];  
for (int i=0;i<8;i++)  
{  
    hs1dPos[i] = (i + 1)*10000; // X 轴位置  
}  
rtn = NMC_CompHs1DimensSetData(devhandle,(long *)hs1dPos,8,0);  
if(rtn != 0){ return ;}
```

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_CompHs1DimensOnOff(devhandle,1,0);  
if(rtn != 0){ return ;}
```

d. 读取位置比较输出状态

```
//读比较状态  
short cmpIsBusy,cmpOutCount,cmpWaitCount,cmpFreeSpace;  
rtn = NMC_CompHs1DimensStatus(devhandle,&cmpIsBusy,&cmpOutCount,& cmpWaitCount,&  
cmpFreeSpace,0);  
if(rtn != 0){ return ;}
```

2.4、手动输出

```
short rtn=0;

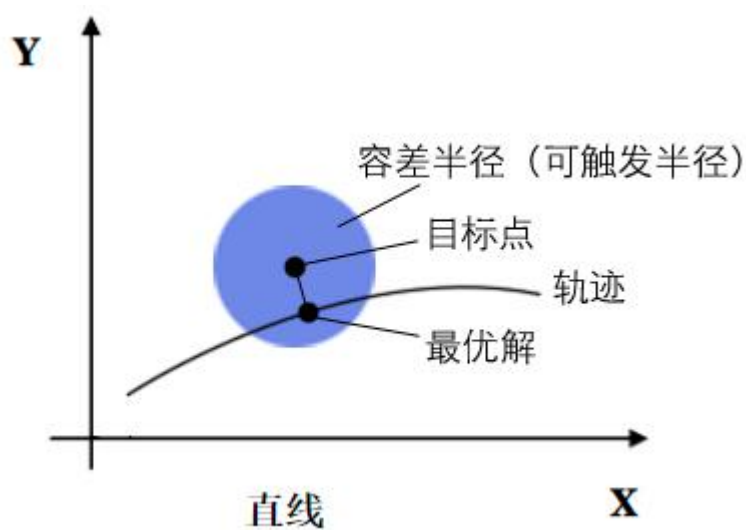
// g_hDev 为控制器控制器句柄，设置一维比较输出 group0 的参数
TCompHs1DimensParam cmpPrm;
memset(&cmpPrm,0,sizeof(TCompHs1DimensParam));
cmpPrm.dirNo= 0;           // 第一轴
cmpPrm.out1StLevel= 1;     // out1 起始电平为高电平
cmpPrm.out2StLevel = 1;    // out2 起始电平为高电平
cmpPrm.out1Width = 10000;  // out1 脉冲宽度为 us
cmpPrm.out2Width = 10000; // out2 脉冲宽度为 us
cmpPrm.out2Delay = 100;    // out1 与 out2 之间的延时为 us
rtn = NMC_CompHs1DimensSetParam(devhandle,& cmpPrm,0);
if(rtn != 0){ return ;}

// 启动位置比较输出
rtn = NMC_CompHs1DimensOnOff(devhandle,2,0);
if(rtn != 0){ return ;}
```

3、二维高速位置比较

3.1、功能原理

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。



3.2、指令说明

(1) 设置2维位置比较的参数

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

参数:

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

(2) 获取2维位置比较的参数

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

参数:

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

ch	short	输入	保留，设为 0
----	-------	----	---------

(3) 设置二维比较数据

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

参数:

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
ch	short	输入	保留，设为 0

(4) 2维位置比较使能

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

参数:

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
ch	short	输入	保留，设为 0

(5) 获取2维位置比较输出状态

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

参数:

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
ch	short	输入	保留，设为 0

(6) 获取2维位置比较输出状态

[NMC_Comp2DimensStatusEx\(HAND devHandle,short group,TComp2DimensSts *pStatus,short chn\);](#)

参数:

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	typedef struct {

			<pre> 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>
ch	short	输入	保留，设为 0

3.3、综合示例

```

short rtn=0;
TComp2DimensParam compara_2d;
memset(&compara_2d,0,sizeof(compara_2d));
//获取第组的二维位置比较参数
rtn=NMC_Comp2DimensGetParam(devhandle,0,&compara_2d,0);
compara_2d.dir1No=0; //轴，等于-1 就是二维比较的一维模式
compara_2d.dir2No=1; //轴，等于-1 就是二维比较的一维模式
compara_2d.chnType[0]=1; //通道的硬件接口为控制器拓展口上的 gate (hsio) 引脚
compara_2d.chnType[1]=0; //通道的硬件接口为 gpo
compara_2d.chnType[2]=0; //通道的硬件接口为 gpo
compara_2d.outputType[0]=0; //通道输出脉冲信号
compara_2d.outputType[1]=1; //通道输出电平信号
compara_2d.outputType[2]=2; //通道输出电平信号
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; //通道脉冲输出时间 ms
compara_2d.gateTime[1]=10; //通道（输出为 GPO 时该值无效）
compara_2d.gateTime[2]=10; //通道（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
//设置第组二维位置比较参数
rtn=NMC_Comp2DimensSetParam(devhandle,0,&compara_2d,0);
//设置第组比较的数据
int count=100;
long *compos=new long[2*count];
for (int i=0;i<count;i++)
{
    *(compos+i*2)=100*i+100; //X 坐标
    *(compos+i*2+1)=100*i+100; //Y 坐标
}

```

```
}  
//先清空  
rtn=NMC_Comp2DimensSetData(devhandle,0,compos,0,0);  
//再设置  
rtn=NMC_Comp2DimensSetData(devhandle,0,compos,count,0);  
//开始比较  
rtn=NMC_Comp2DimensOnoff(devhandle,0,1,0);  
//开启后可以通过 NMC_Comp2DimensStatus()读取比较的状态
```

3.4、手动输出

手动输出一个信号

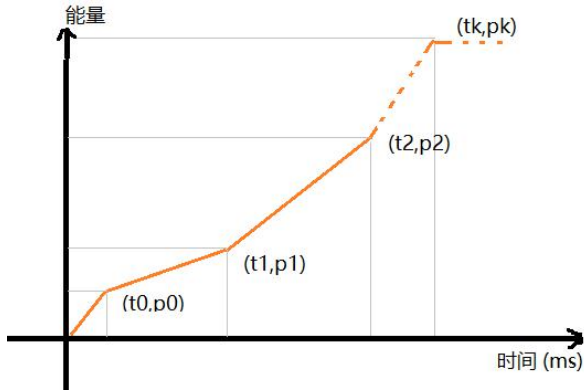
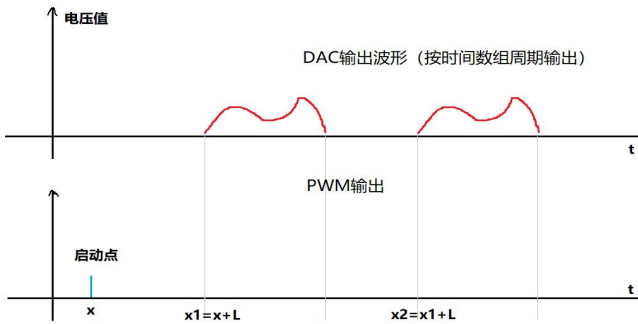
```
short rtn=0;  
  
TComp2DimensParam compara_2d;  
memset(&compara_2d,0,sizeof(compara_2d));  
//获取第组的二维位置比较参数  
rtn=NMC_Comp2DimensGetParam(devhandle,0,&compara_2d,0);  
compara_2d.dir1No=0; //轴，等于-1 就是二维比较的一维模式  
compara_2d.dir2No=1; //轴，等于-1 就是二维比较的一维模式  
compara_2d.chnType[0]=1; //通道的硬件接口为控制器拓展口上的 gate (hsio) 引脚  
compara_2d.chnType[1]=0; //通道的硬件接口为 gpo  
compara_2d.chnType[2]=0; //通道的硬件接口为 gpo  
compara_2d.outputType[0]=0; //通道输出脉冲信号  
compara_2d.outputType[1]=1; //通道输出电平信号  
compara_2d.outputType[2]=2; //通道输出电平信号  
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; //通道脉冲输出时间 ms  
compara_2d.gateTime[1]=10; //通道（输出为 GPO 时该值无效）  
compara_2d.gateTime[2]=10; //通道（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  
//设置第组二维位置比较参数  
rtn=NMC_Comp2DimensSetParam(devhandle,0,&compara_2d,0);  
rtn=NMC_Comp2DimensOnoff(devhandle,0,2,0);  
//开启后可以通过 NMC_Comp2DimensStatus()读取比较的状态
```

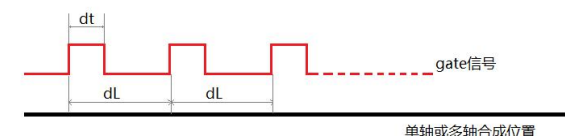
二、激光功能

1、激光模式设置

1.1、功能原理

激光控制包括三种控制模式（通过 [NMC_LaserSetMode](#)），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>其中： L:比较长度</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。
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，如下图示意：</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>

1.2、指令说明

设置激光的控制模式

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

参数：

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]

1.3、综合示例

```
short rtn=0;

// g_hDev 为控制器控制句柄，设置激光通道为立即模式
rtn = NMC_LaserSetMode(devhandle,BASIC_OUTPUT_MODE,0);
if(rtn != 0){ return ;}
```

2、基本控制模式（DA、PWM 占空比、PWM 频率）

2.1、功能原理

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](#)），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 <code>optionVal</code> 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 <code>optionVal</code> 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 <code>optionVal</code> 用来设置 PWM 的脉宽时间

2.2、指令说明

(1) 设置激光物理信号输出类型

[NMC_LaserSetOutputTypeEx \(HAND devHandle, short outputType, short index, double optionVal, short ch\);](#)

参数:

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 \(HAND devHandle, long onDelay ,long offDelay, long minVal ue ,long maxVal ue,long standbyPower,short ch\);](#)

参数:

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 \(HAND devHandle, long *pOnDelay, long *pOffDelay, long *pMinValue, long *pMaxValue, long *pStandbyPower, short ch\);](#)

参数:

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], 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 \(HAND devHandle, double outVal, short ch\);](#)

参数:

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]
ch	short	输入	通道号, 范围[0,1]

(5) 设置激光立即输出开关

[NMC_LaserOnOff \(HAND devHandle, short onOff, short ch\);](#)

参数:

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 \(HAND devHandle,double *pVal,short ch\);](#)

参数:

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 \(HAND devHandle,short *pOnOffState,short ch\);](#)

参数:

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 \(HAND crdHandle, long segment,long onDelay ,long offDelay, long minValue ,long maxValue,long standbyPower,short ch\);](#)

参数:

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
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]

(9) 缓冲区设置输出激光能量

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

参数:

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

参数:

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
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\(HAND devHandle,short powerFilter,short followAdvMode,short ch\);](#)

参数:

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\(HAND crdHandle,long segNo,double overRide,short followType,short ch\);](#)

参数:

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
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]

(13) 设置激光能量补偿表

[NMC_SetLaserPowerCmpTable \(HAND devHandle,short tableNo,long *pXCmpPos,long *pYCmpPos,short xCount,short yCount, unsigned long powerMin,unsigned long powerMax,unsigned long *pLaserCmpPower,short ch\);](#)

参数:

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 2 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 \(HAND devHandle,short *pAxisNo,short ch\);](#)

参数:

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]

(14) 停止激光能量补偿

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

参数:

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]

2.3、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

```
short rtn=0;

// g_hDev 为控制器控制器句柄, 设置激光通道为立即模式
rtn = NMC_LaserSetMode(devhandle,BASIC_OUTPUT_MODE,0);
if(rtn != 0){ return ;}

// 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);
if(rtn != 0){ return ;}
```

b.直接设置激光能量/开关激光

```
// 直接设置激光能量
rtn = NMC_LaserSetPowerEx(devhandle,10,0);
if(rtn != 0){ return ;}
```

```
// 开光
rtn = NMC_LaserOnOff(devhandle,1,0);
if(rtn != 0){ return ;}
```

c.插补缓冲区中对激光进行控制

```
//坐标系建立过程略
//设置激光通道为立即模式
rtn = NMC_LaserSetMode(devhandle,BASIC_OUTPUT_MODE,0);
if(rtn != 0){ return ;}
// 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);    if(rtn != 0){ return ;}
// 清除缓冲区，压入缓冲区指令
rtn = NMC_CrdBufClr(devhandle);
if(rtn != 0) { return ;}
//设置激光通道的能量为 10%
rtn = NMC_CrdBufLaserPower(crdhandle,100,10,0);
if(rtn != 0){ return ;}
// 开激光
rtn = NMC_CrdBufLaserOnOff(crdhandle,101,1,0);
if(rtn != 0){ return ;}
// 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);
if(rtn != 0){ return ;}
//插补完成后关闭激光输出
rtn = NMC_CrdBufLaserOnOff(crdhandle,101,0,0);
if(rtn != 0){ return ;}

//指令压入结束
rtn = NMC_CrdEndMtn(crdhandle);
if(rtn != 0){ return ;}

//启动缓冲区运动
rtn = NMC_CrdStartMtn(crdhandle);
if(rtn != 0){ return ;}
```

d.速度能量跟随输出

```
//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);
if(rtn != 0){ return ;}
```

```
// 打开激光速度能量跟随，跟随倍率为，跟随规划位置
rtn = NMC_CrdBufLaserSetFollow(crdhandle,1,5,0,0);
if(rtn != 0){ return ;}

// 关闭激光
rtn = NMC_CrdBufLaserOnOff(crdhandle,101,0,0);
if(rtn != 0){ return ;}

double vel=10;
double acc=1;
// 运动到起始点
tgtPos[0] = 20000;
tgtPos[1] = 0;
rtn = NMC_CrdLineXYZEx(crdhandle,154,0x3,tgtPos,0,vel,acc,0);
if(rtn != 0){ return ;}

// 开光
rtn = NMC_CrdBufLaserOnOff(crdhandle,101,1,0);
if(rtn != 0){ return ;}

// 加工（插补过程中，激光自动根据速度变化）
tgtPos[0] = 100000;
tgtPos[1] = 0;
rtn = NMC_CrdLineXYZEx(crdhandle,154,0x3,tgtPos,0,vel,0.01,0);
if(rtn != 0){ return ;}

// 加工完成，关闭激光
rtn = NMC_CrdBufLaserOnOff(crdhandle,101,0,0);
if(rtn != 0){ return ;}

//指令压入结束
rtn = NMC_CrdEndMtn(crdhandle);
if(rtn != 0){ return ;}

//启动缓冲区运动
rtn = NMC_CrdStartMtn(crdhandle);
if(rtn != 0){ return ;}
```

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++)
{
```



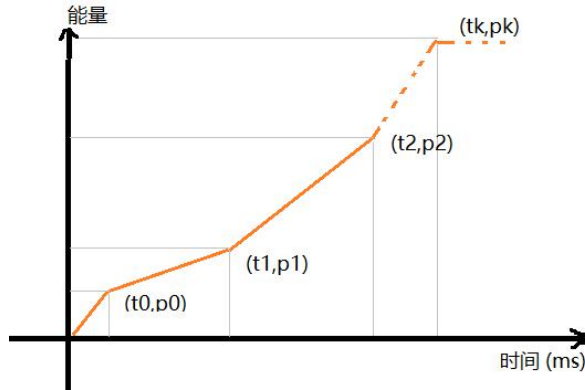
```

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);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
}
short pAxisNo[2]={0,1};//X 方向为轴，Y 方向为轴
// 启用激光能量补偿表
rtn = NMC_StartLaserPowerComp (devhandle,pAxisNo,0);          if(rtn != RTN_CMD_SUCCESS)
{
    return;
}

```

3、波形控制模式

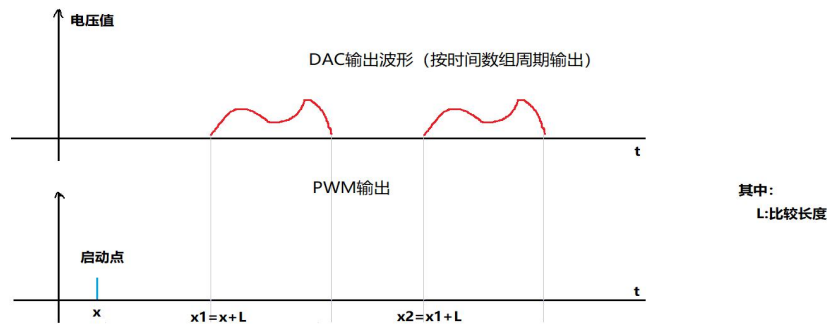
3.1、功能说明

模式	说明
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。 分为两种模式：</p>

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，输出一次能量波形，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。

3.2、指令说明

(1) 设置立即输出激光能量

[NMC_LaserSetTimeArrayPara\(HAND devHandle,TTimeArrayPara *pPara,short ch\);](#)

参数:

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</pre>

			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;
ch	short	输入	通道号, 范围[0,1]

(2) 设置激光能量时间数组

[NMC_LaserSetTimeArrayPower\(HAND devHandle,short group,TLaserPower *pLaserPower\);](#)

参数:

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 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;

(3) 执行激光能量时间数组

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

参数:

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

参数:

名称	数据类型	输入/输出	描述
crdHandle	HAND	输入	坐标系句柄
segNo	long		段号
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]

3.3、综合示例

a.配置激光控制模式及信号输出模式

```
short rtn=0;
//设置激光输出模式为波形模式
rtn = NMC_LaserSetMode(devhandle, TIME_ARRAY_OUTPUT_MODE,0);
if(rtn != 0){ return ;}
// 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);
if(rtn != 0){ return ;}
```

b.单点激光能量时间数组控制

```
// 配置时间能量数组
TLaserPower lserPw;
memset(&lserPw,0,sizeof(TLaserPower));
lserPw.count = 40;
for(int i=0;i<20;i++)
{
    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);
if(rtn != 0){ return ;}

// 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);
if(rtn != 0){ return ;}
```

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);
if(rtn != 0){ return ;}
// 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);
if(rtn != 0){ return ;}
// 能量控制数组
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);
if(rtn != 0){ return ;}

// 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);
if(rtn != 0){ return ;}

// 开始执行
// 缓冲区请使用 NMC_CrdBufLaserTimeArrayExe
rtn = NMC_LaserTimeArrayExe(devhandle,0,0);
if(rtn != 0){ return ;}

// .....

//关闭激光能量输出 (缓冲区请使用 NMC_CrdBufLaserOnOff)
rtn = NMC_LaserOnOff(devhandle,0, 0);
if(rtn != 0){ return ;}
```

4、位置比较控制模式

4.1、功能说明

模式	说明
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，如下图示意：</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>

4.2、指令说明

（1）配置 SHIO 功能的参数

[NMC_SHIOConfigPara\(HAND devHandle, TSHIOPara *pSHIOPara, short ch\);](#)

参数：

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</pre>

			<p>最小值: 1/36us),取值范围(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: 0:1:K,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\(HAND devHandle,long minFrq,short ch\);](#)

参数:

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\(HAND devHandle,short ch\);](#)

参数:

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\(HAND crdHandle,long segment, long minFrq,short ch\)](#)

参数:

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
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\(HAND crdHandle,long segment,short ch\);](#)

参数:

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
ch	short	输入	通道号, 目前只为 0

(6) 缓冲区配置SHIO参数

[NMC_CrdBufSHIOSetParam\(HAND crdHandle, long segment, double delay, double gateTime, double gateDistance, short ch\);](#)

参数:

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
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\(HAND devHandle, short axisMask, short ch\);](#)

参数:

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\(HAND crdHandle, long segment, short axisMask, short ch\);](#)

参数:

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
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\(HAND devHandle, short ch\);](#)

参数:

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\(HAND devHandle, short ch\);](#)

参数:

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\(HAND devHandle,double freq,double width ,short ch\);](#)

参数:

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	triger 脉冲频率单位: HZ
width	double	输入	triger 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\(HAND devHandle ,short ch\);](#)

参数:

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

参数:

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
ch	short	输入	通道号，目前只为 0

(14) 缓冲区关闭Gate输出

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

参数:

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
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\(HAND devHandle, short pwmCh,short gateCh,short onOff\);](#)

参数:

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\(HAND crdHandle,long segNo,double gateTime,double gateFrq,long outCount,short ch\);](#)

参数:

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
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\(HAND devHandle,double gateTime, double gateFrq,long outCount,short ch\);](#)

参数:

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\(HAND devHandle,double onTime, double pwmFrq,long outCount,short ch\);](#)

参数:

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

4.3、综合示例

a.立即输出 Gate 信号, NMC_SHIOGateOn 后立即输出 Gate

```
short rtn=0;

//=====立即输出一段 GATE 信号=====

rtn = NMC_LaserSetMode(devhandle,SHIO_OUTPUT_MODE,0);

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

```
//位置比较控制模式参数配置
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);
if(rtn != 0){ return; }

//.....

// 缓冲区中控制 gate 信号
// 启动 gate 信号输出
rtn = NMC_BufSHIOGateOn(devhandle,1,0);
if(rtn != 0){ return; }
// 关闭 gate 信号输出
rtn = NMC_BufSHIOGateOff(devhandle,1,0);
if(rtn != 0){ return; }
// 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

```
rtn = NMC_LaserSetMode(devhandle,SHIO_OUTPUT_MODE,0);
if(rtn != 0){ return; }
//位置比较控制模式参数配置
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);
if(rtn != 0){ return; }

//.....

// 缓冲区中控制 gate 信号
// 启动 gate 信号输出
rtn = NMC_BufSHIOGateOn(crdhandle,1,0);
if(rtn != 0){ return; }
```

```
// 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);
if(rtn != 0){ return; }
// 关闭 gate 信号控制
rtn = NMC_BufSHIOGateOff(crdhandle,1,0);
if(rtn != 0){ return; }
// 加工，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);
if(rtn != 0){ return; }
```

三、PT 运动功能

1、功能说明

PT 模式使用一系列“位置、时间”数据点描述速度规划，用户需要将曲线分割成若干段如下图所示：

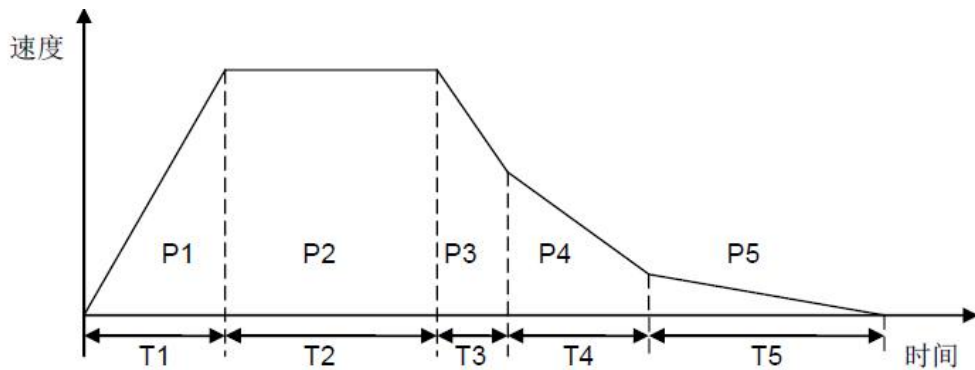


图 1 PT 运动速度曲线

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 ，因此第 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

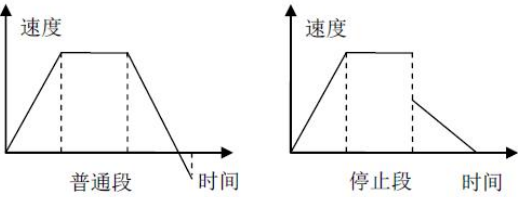
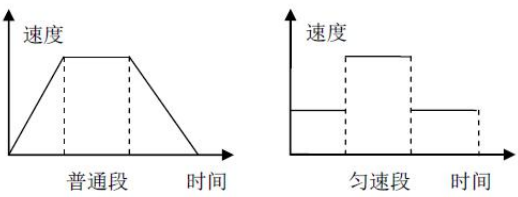
点速度为 $v_2 = \frac{2P_2}{T_2} - v_1$ ；第 3、4、5 段依此类推。PT 模式的数据段要求用户输入每段所需时间和位置点。

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> 

2、指令说明

(1) 设置 PT 运动的数据模式和循环次数

[NMC_MtPtSetStatic\(HAND axisHandle,short onOff,long loopCount\);](#)

参数:

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) 获取 PT 运动的数据模式和循环次数

[NMC_MtPtGetStatic\(HAND axisHandle,short *pOnOff,long *pLoopCount\);](#)

参数:

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) 获取 PT 运动缓存区剩余空间大小

[NMC_MtPtGetSpace\(HAND axisHandle,short *pSpace,short *pUsed\);](#)

参数:

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\);](#)

参数:

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\(HAND axisHandle\);](#)

参数:

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\(HAND axisHandle,short otherSynAxCnts,short *pOtherSynAxArray\);](#)

参数:

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

3、综合示例

a.数据驻存模式（静态 FIFO 模式）

```
short rtn=0;
// 设置轴为 PT 运动模式
rtn = NMC_MtSetPrfMode(axisHandle[0],MT_PT_PRF_MODE);
if(rtn != RTN_CMD_SUCCESS)
{
    return; // 设置轴为 PT 运动模式出错
}
//////////PT 数据驻存模式//////////
//设置为 PT 数据驻存模式,循环次
```

```
rtn = NMC_MtPtSetStatic(axishandle[0],1,5);
if(rtn != RTN_CMD_SUCCESS)
{
    return;          // 设置为 PT 数据驻存模式,循环次出错
}
//清空 PT 数据缓冲区
rtn = NMC_MtPtBufClr(axishandle[0]);
if(rtn != RTN_CMD_SUCCESS)
{
    return;          // 清空 PT 数据缓冲区出错
}
//查询 PT 缓冲区空间
short space=0;
short used=0;
rtn = NMC_MtPtGetSpace(axishandle[0],&space,&used);
if(rtn != RTN_CMD_SUCCESS)
{
    return;          // 查询 PT 缓冲区空间出错
}
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};
//往 PT 缓冲区压数据
rtn = NMC_MtPtPush(axishandle[0],3,posArray,timeArray,typeArray);
if(rtn != RTN_CMD_SUCCESS)
{
    return;          //往 PT 缓冲区压数据出错
}
short otherSynAxArray[]={0};
//启动 PT 运动
rtn = NMC_MtPtStartMtn(axishandle[0],0,otherSynAxArray);
if(rtn != RTN_CMD_SUCCESS)
{
    return;          // 启动 PT 运动出错
}
while(1)
{
    short sts=0;
    rtn = NMC_MtGetSts(axishandle[0],&sts);
    if ((sts&BIT_AXIS_BUSY)!=BIT_AXIS_BUSY)//等待运动结束
    {
        break;
    }
}
```


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

// 设置轴为 PT 运动模式
rtn = NMC_MtSetPrfMode(axishandle[0],MT_PT_PRF_MODE);
if(rtn != RTN_CMD_SUCCESS)
{
    return;           // 设置轴为 PT 运动模式出错
}

////////////////////PT 数据刷新模式////////////////////////////////////
rtn = NMC_MtZeroPos(axishandle[0]);
//设置为 PT 数据刷新模式
rtn = NMC_MtPtSetStatic(axishandle[0],0,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;           // 设置为 PT 数据刷新模式
}

//清空 PT 数据缓冲区
rtn = NMC_MtPtBufClr(axishandle[0]);
if(rtn != RTN_CMD_SUCCESS)
{
    return;           // 清空 PT 数据缓冲区出错
}

double pos = 0;
double vel = 0;
double velPre = 0;
double time = 0;
short start = 0;
short loop = 1;
long ltime=0;
short type=0;
while(1)
{
    //查询 PT 缓冲区空间
    short space=0;
    short used=0;
    rtn = NMC_MtPtGetSpace(axishandle[0],&space,&used);
    if(rtn != RTN_CMD_SUCCESS)
    {
        return;       // 查询 PT 缓冲区空间出错
    }
}
```

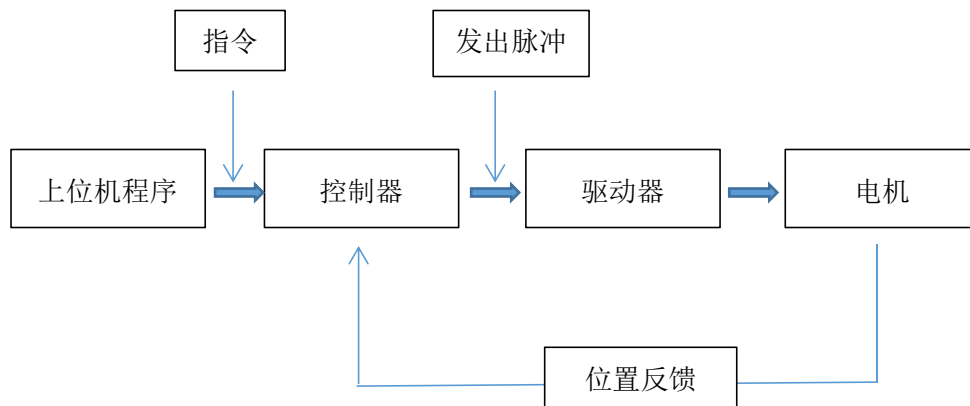
```
}  
if( space > 0 )  
{  
    time += DELTA;  
    // 计算段末速度  
    vel = A*sin((2*PI)/T*time);  
    // 计算段内位移  
    pos += 1000*(vel+velPre)*DELTA/2;  
    velPre = vel;  
    if(time < loop*T)  
    {  
        ltime = (long)(time*1000);  
        type = MT_PT_NORMAL;  
        // 发送新数据  
        rtn = NMC_MtPtPush(axishandle[0],1,&pos,&ltime,&type);  
        if(rtn != RTN_CMD_SUCCESS)  
        {  
            return;          //往 PT 缓冲区压数据出错  
        }  
    }  
    else  
    {  
        // 发送终点数据  
        pos = 0;  
        ltime = (long)(loop*T*1000);  
        type = MT_PT_STOP;  
        rtn = NMC_MtPtPush(axishandle[0],1,&pos,&ltime,&type);  
        if(rtn != RTN_CMD_SUCCESS)  
        {  
            return;          //往 PT 缓冲区压数据出错  
        }  
        pos = 0;  
        time = loop*T;  
        velPre = 0;  
        ++loop;  
        if( loop > LOOP )  
        {  
            break;  
        }  
    }  
}  
else if( 0 == start )  
{  
    // 启动 PT 运动
```

```
    rtn = NMC_MtPtStartMtn(axishandle[0],0,otherSynAxArray);  
    if(rtn != RTN_CMD_SUCCESS)  
    {  
        return;        // 启动 PT 运动出错  
    }  
    start = 1;  
}  
}  
while(1)  
{  
    Sleep(50);  
    short sts=0;  
    rtn = NMC_MtGetSts(axishandle[0],&sts);  
    if ((sts&BIT_AXIS_BUSY)!=BIT_AXIS_BUSY)//等待运动结束  
    {  
        break;  
    }  
}
```

四、闭环控制功能

1、功能说明

对应支持模拟量的控制器型号（选配），可以就轴运动控制设置为模拟量控制的闭环模式。



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 \(HAND axisHandle,TDacMotor *pDacPara\);](#)

参数：

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）获取单轴闭环控制的 DA 参数

[NMC_MtGetCloseLoopDac \(HAND axisHandle,TDacMotor *pDacPara\);](#)

参数：

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

参数：

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

参数：

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\(HAND axisHandle,short index, TPidPara *pidPara\);](#)

参数：

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\(HAND axisHandle,short index, TPidPara *pPidPara\);](#)

参数：

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\(HAND axisHandle,short index\);](#)

参数：

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) 获取正使用的 PID 组号

[NMC_MtGetPIDIndex\(HAND axisHandle,short *pIndex\);](#)

参数:

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

参数:

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

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）:

```
//此处省略控制器初始化部分
short rtn=0;
short axisIndex=0;
//闭环设置
TDacMotor dacMotor;
dacMotor.inverse=0; //更改运动方向，这个值和编码器取反要一同修改。
dacMotor.bias=0;
dacMotor.dacLmt = 500;//32767; //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
rtn = NMC_MtSetCloseLoopDac(axisHandle[0],&dacMotor);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
rtn = NMC_SetEncMode(devhandle,axisIndex,0x0000);//设置编码器为外部编码器
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
```

```
rtn = NMC_MtSetCtrlMode(axishandle[0],1);//设置为闭环控制模式
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
rtn = NMC_MtSetPosErrLmt(axishandle[0],32767);//设置闭环最大位置误差必需设置
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
TPidPara pidPara;
rtn = NMC_MtGetPIDPara(axishandle[0],0,&pidPara);//读取闭环 PID
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
pidPara.kp=5;
pidPara.ki=1;
rtn = NMC_MtSetPIDPara(axishandle[0],0,&pidPara);//设置闭环 PID 必需设置
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
rtn = NMC_MtSetPIDIndex(axishandle[0],0);//选择使用的闭环 PID 必需设置
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
rtn = NMC_MtSetSvOn(axishandle[0]);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//至此就可以直接调用轴运动指令驱动轴了
```

五、电子齿轮

1、功能原理

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：

※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。

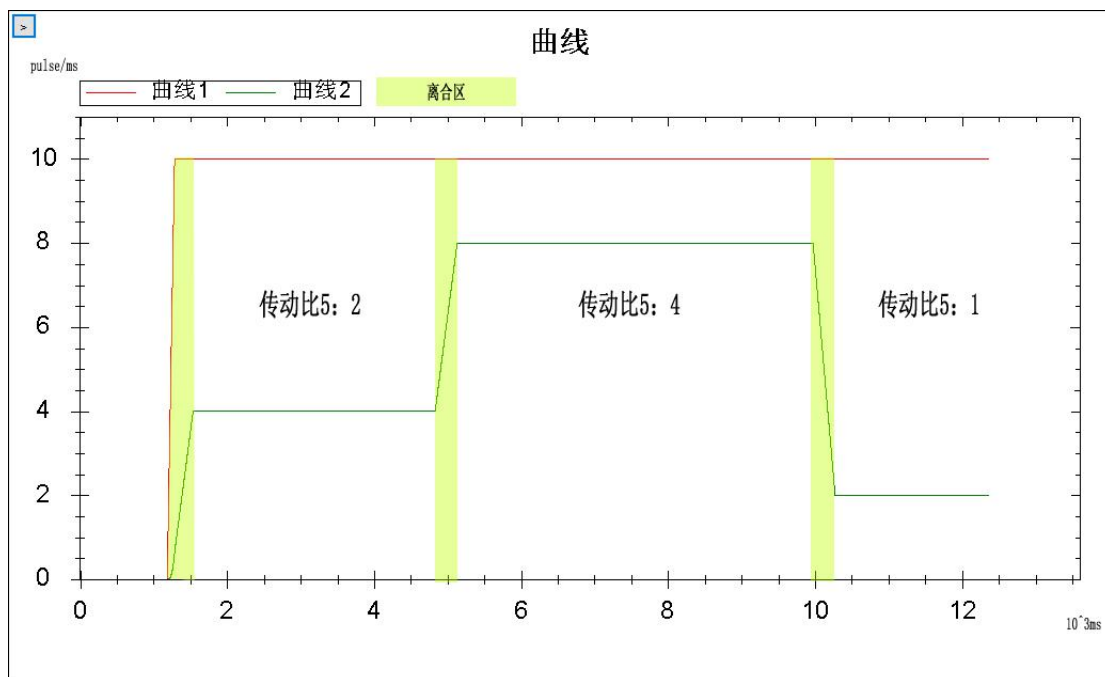
※从轴可以跟随主轴的规划位置或者实际位置，从轴也可以直接跟随外部的编码器输入。

※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。

※主轴和从轴之间按照固定的速度比例运动，即传动比，传动比可以在运动过程随时修改

※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-时间曲线示意图如下：



（图示红色线为主轴速度，绿色线为从轴速度）

2、指令说明

（1）设置从轴跟随方向

[NMC_MtGearSetDir\(HAND axisHandle,short dir\);](#)

参数：

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) 获取从轴跟随方向

[NMC_MtGearGetDir\(HAND axisHandle,short* pdir\);](#)

参数:

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\(HAND axisHandle,short masterNo,short masterType\);](#)

参数:

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) 获取 Gear 主轴参数

[NMC_MtGearGetMaster\(HAND axisHandle,short * pmasterNo,short * pmasterType\);](#)

参数:

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\(HAND axisHandle,long masterEven,long slaveEven,long masterSlope\);](#)

参数:

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	离合区位移, 必须大于 0, 同时不能等于 1

(6) 获取 Gear 跟随倍率

[NMC_MtGearGetRatio\(HAND axisHandle,long *pMasterEven,long *pSlaveEven,long *pMasterSlope\);](#)

参数:

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\(HAND axisHandle,short syncAxCnts,short *pSyncAxArray\);](#)

参数:

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

3、综合示例

```
//此处省略控制器初始化部分
short rtn=0;
//设置轴二的运动模式为电子齿轮模式
rtn=NMC_MtSetPrfMode(axishandle[1],MT_GEAR_PRF_MODE);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置为向跟随
rtn=NMC_MtGearSetDir(axishandle[1],0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置跟随轴一的规划器
rtn=NMC_MtGearSetMaster(axishandle[1],0,1);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置传动比 5:4 和离合区
```

```

rtn=NMC_MtGearSetRatio(axishandle[1],5,4,2000);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//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[1],0,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}

```

六、电子凸轮

1、功能原理

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：

※每组 Follow 运动关联的两个轴，被跟随轴称为主轴，跟随轴称为从轴。多个从轴可以跟随同一个主轴，从轴又可以作为其他轴的主轴。

※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

※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

※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

※通过配置 FIFO 的循环次数，可实现从轴周期性的跟随

※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

2、指令说明

（1）设置 FOLLOW 跟随方向

[NMC_MtFollowSetDir\(HAND axisHandle,short dir\);](#)

参数：

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) 获取 FOLLOW 跟随方向

[NMC_MtFollowGetDir\(HAND axisHandle,short *pDir\);](#)

参数:

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\(HAND axisHandle,short masterNo,short masterType\);](#)

参数:

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 #define PROFILE_FOLLOW_MASTER_NONE (0) // 0:无效 #define PROFILE_FOLLOW_MASTER_AXIS_PRF (1) // 1:AXIS 规划值 #define PROFILE_FOLLOW_MASTER_AXIS_ENC (2) // 2: AXIS 反馈值 #define PROFILE_FOLLOW_MASTER_ENC (3) // 3: 编码器值

(4) 获取 FOLLOW 主轴参数

[NMC_MtFollowGetMaster\(HAND axisHandle,short *pMasterNo,short *pMasterType\);](#)

参数:

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\(HAND axisHandle,long loopCnt\);](#)

参数:

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) 获取 FOLLOW 的循环执行次数

[NMC_MtFollowGetLoopCount\(HAND axisHandle,long *pLoopCnt\);](#)

参数:

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\(HAND axisHandle,short eventType,short masterDir,long pos\);](#)

参数:

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) 获取 FOLLOW 的启动事件

[NMC_MtFollowGetEvent\(HAND axisHandle,short *pEventType,short *pMasterDir,long *pPos\);](#)

参数:

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) 获取 FOLLOW 的 fifo 剩余空间

[NMC_MtFollowGetSpace\(HAND axisHandle,short *pSpace,short fifoNo\);](#)

参数:

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\(HAND axisHandle,long masterPos,double slavePos,short type,short fifoNo\);](#)

参数:

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 匀加速;

			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\(HAND axisHandle,short fifoNo\);](#)

参数:

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\(HAND axisHandle,short syncAxCnts,short *pSyncAxArray\);](#)

参数:

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\(HAND axisHandle,short syncAxCnts,short *pSyncAxArray\);](#)

参数:

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

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

//此处省略控制器初始化部分

short rtn=0;

//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);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置轴二的运动模式为 FOLLOW 模式
rtn=NMC_MtSetPrfMode(axishandle[1],MT_FOLLOW_PRF_MODE);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//清空从轴 FIFO 数据
rtn=NMC_MtFollowClear(axishandle[1],0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置跟随主轴的规划位置
rtn=NMC_MtFollowSetMaster(axishandle[1],0,PROFILE_FOLLOW_MASTER_AXIS_PRF);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
//设置跟随数据
double masterpos=20000;
double slavepos=10000;
rtn=NMC_MtFollowPushData(axishandle[1],masterpos,slavepos,0,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
masterpos+=20000;
slavepos+=20000;
rtn=NMC_MtFollowPushData(axishandle[1],masterpos,slavepos,0,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
masterpos+=20000;
slavepos+=10000;
rtn=NMC_MtFollowPushData(axishandle[1],masterpos,slavepos,0,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return;
}
```

```
}  
//设置为无限循环  
rtn=NMC_MtFollowSetLoopCount(axishandle[1],0);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return;  
}  
//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);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return;  
}  
//启动 Follow 运动  
rtn=NMC_MtFollowStart(axishandle[1],0,0);  
if(rtn != RTN_CMD_SUCCESS)  
{  
    return;  
}  
//跟随运动过程中可调用 NMC_MtFollowGetLoopCount 查询循环执行次数
```


七、拓展资源及其他功能

1、辅助编码器

1.1、指令说明

(1) 读取编码器通道值

[NMC_GetEncPos\(HAND devHandle, short encId, long *pPos \);](#)

参数:

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\(HAND devHandle, short encId, long encPos \);](#)

参数:

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

2、捕获功能

2.1、功能说明

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。

2.2、指令说明

2.2.1、通用捕获

(1) 设置硬件捕获参数

[NMC_MtSetCaptSns\(HAND axisHandle, short mode, short ioSrc, short level \);](#)

参数:

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
ioSrc	short	输入	硬件捕获 IO 源选择, 值参照宏定义, 见头文件
level	short	输入	触发边沿, 0 为下降沿, 1 为上升沿

(2) 读取硬件捕获参数

[NMC_MtGetCaptSns\(HAND axisHandle, short *pMode, short *pIoSrc, short *pLevel \);](#)

参数:

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, 0 为下降沿, 1 为上升沿

(3) 启动捕获功能

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

参数:

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

注: 捕获是否触发可以从轴状态字判断 ([NMC_MtGetSts\(\)](#))

(4) 读取捕获触发时编码器位置

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

参数:

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\(HAND axisHandle\);](#)

参数:

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

2.2.2、高级捕获

(1) 设置高级捕获参数,并启动捕获

[NMC_MtSetAdvCaptParam\(HAND devHandle, TAdvCaptureParam *pParam,short ch\);](#)

参数:

名称	数据类型	输入/输出	描述
devHandle	HAND	输入	轴句柄
pParam	TAdvCaptureParam *	输入	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) 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 IO,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;
ch	short	输入	捕获通道号,取值范围[0,3]

(2) 清除高级捕获状态,并取消该通道的捕获

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

参数:

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]

(3) 读取高级捕获状态

[NMC_MtGetAdvCaptPos\(HAND devHandle, short *captSts,long *pPosArray ,short ch\);](#)

参数:

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]

2.3、综合示例

a.捕获功能

```

short rtn=0;
//设置捕获参数,DIO 为捕获源,下降沿触发
rtn = NMC_MtSetCaptSns(axishandle[0],CAPT_MODE_IO,CAPT_IO_SRC_DIO,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return ;
}
//启动捕获
rtn = NMC_MtSetCapt(axishandle[0]);
if(rtn != RTN_CMD_SUCCESS)
{
    return ;
}

//此处省略轴的运动过程
short sts;
long pos;
while(1)
{
    rtn = NMC_MtGetSts(axishandle[0],&sts);
    if ((sts&BIT_AXIS_CAPTSET)==BIT_AXIS_CAPTSET)//已经捕获到
    {
        //读取捕获位置
        rtn = NMC_MtGetCaptPos(axishandle[0],&pos);
        break;
    }
}

```

b. 高级捕获功能

```

short rtn=0;
TAdvCaptureParam advCaptureParam;

//设置高级捕获功能

```

```

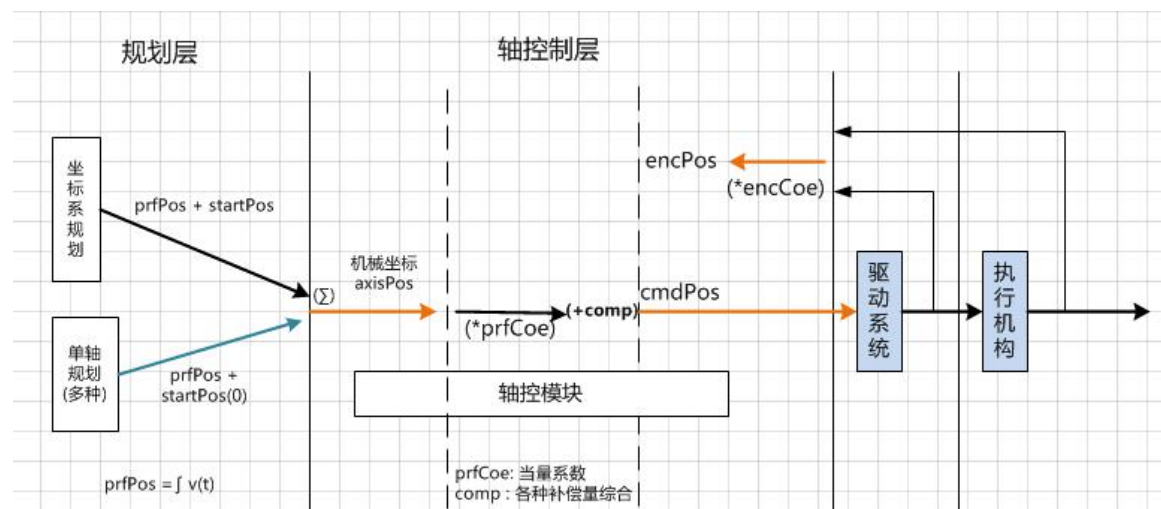
advCaptureParam.capPosIndex = 0;
advCaptureParam.filter = 100;
advCaptureParam.trigIndex = 0;
advCaptureParam.trigSrc = CAPT_EX_SRC_GPI;
advCaptureParam.trigValue = 0;
rtn = NMC_MtSetAdvCaptParam(devhandle,&advCaptureParam,0);
if(rtn != RTN_CMD_SUCCESS)
{
    return ;
}
////此处省略轴的运动过程
short sts;
long pos[2];
while(1)
{
    rtn = NMC_MtGetAdvCaptPos(devhandle,&sts,pos,0);
    if (sts==0x3)//前后沿均已捕获到
    {
        break;
    }
}

```

3、位置系统操作

3.1、功能说明

控制系统的位置系统框图如下：



说明：

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_CrdGetPrfPos
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=(命令脉冲/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

3.2、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 update 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：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 update 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：只能在轴静止时使用

3)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]

4)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]

5)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]

6)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]

7)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,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

8)获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,单位 Pulse
pStableTime	long *	输出	到位保持时间,单位 ms

9) 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

10) 读取单轴规划高级参数

[NMC_MtGetPrfConfig\(HAND axisHandle,short *mapAxisNo,short *port,long *startPos\);](#)

参数参考 [NMC_MtPrfConfig\(\)](#).

4、螺距补偿和间隙补偿

4.1、指令说明

(1) 设置螺距误差补偿参数

[NMC_MtSetLeadScrewCompPara\(HAND axisHandle,short num, long startPos,long cmpLen,short *pCompPos,short *pCompNeg\);](#)

参数:

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

(2) 启动或者关闭螺距误差补偿

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

参数:

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

(3) 设置反向间隙补偿参数

[NMC_MtSetBacklash\(HAND axisHandle, long compValue,double compDelta,long compDir\);](#)

参数:

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

(4) 获取反向间隙补偿参数

`NMC_MtGetBacklash(HAND axisHandle,long *pCompValue,double *pCompDelta,long *pCompDir);`

参数:

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

4.2、综合示例


a.螺距误差补偿

```
short rtn=0;
short comPos[]={10,20,-10};
short comNeg[]={5,-5,5};
//设置螺距误差补偿参数
rtn = NMC_MtSetLeadScrewCompPara(axishandle[0],3,10000,20000,comPos,comNeg); //3 个点段直线, 起点, 终点
if(rtn != RTN_CMD_SUCCESS)
{
    return ; //
}
rtn = NMC_MtEnableLeadScrew(axishandle[0],1); //启动补偿
if(rtn != RTN_CMD_SUCCESS)
{
    return ; //
}
```

b.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);
if(rtn != RTN_CMD_SUCCESS)
{
    return ;
}
```

5、控制器资源(参数、版本信息等)


	<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的参数。</p> <p>本节会在电池寿命相关的功能处作标注说明，请留意。</p>
---	--

5.1、指令说明

(1) 读时间

[NMC_GetTime\(HAND devHandle, TNMCTime *pTime \);](#)


参数：

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 year; // 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，电池失效后读写时间会不正确，请合理使用。		

(2) 写时间

[NMC_SetTime\(HAND devHandle, TNMCTime *pTime,char *pPassword\);](#)

参数：

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，电池失效后读写时间会不正确，请合理使用。		

(3) 读设备唯一序列号

[NMC_GetUID\(HAND devHandle, unsigned long *pUID \);](#)

参数：

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

devHandle	HAND	输入	轴句柄
pUID	unsigned long *	输出	返回设备唯一序列号

(4) 写用户参数

[NMC_UserParaWr\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pWrData \);](#)

参数:

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: 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: 写用户参数的次数是有限制的, 理论上不低于 5000 次, 您在使用该功能时需要合理安排策略。

(5) 读用户参数

[NMC_UserParaRd\(HAND devHandle, unsigned long add, unsigned long len, unsigned char *pRdData \);](#)

参数:

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\(HAND devHandle, unsigned long waitTimeInUs, unsigned long retryTimes\);](#)

参数:

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\(HAND devHandle, char *pIdStr \);](#)

参数:

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\(HAND devHandle, char *pIdStr\);](#)

参数:

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\(char *pVersion\);](#)

参数:

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 \(HAND devHandle, TMtLibVersion *pVersion \);](#)

参数:

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 \(HAND devHandle, TCardInfo *pInfo \);](#)

参数:

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 }</pre>

			<pre> 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 I/O 通道 short reserved; // 保留 char ipv4[4]; // IP 地址 char idStr[16]; // 板卡名称, 多 卡时可用名称打开参考 NMC_DevOpenByID unsigned long uid[4]; // 唯一序列 号 } TCardInfo; </pre>
--	--	--	--

(12) 启动指令调试

[NMC_SetCmdDebug\(short enable,char *debugOutputFile\);](#)

参数:

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
debugOutputFile	char *	输出	调试输出文件

(13) 获取错误代码信息

[NMC_GetErrDesc\(short errCode,char *errDesc\);](#)

参数:

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

参数:


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\(HAND devHandle, TBackGroup2 *pBackVar\);](#)

参数:


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 typedef struct {

			<pre> 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 long encPos[12]; // 编码器位置 }TBackGroup2; </pre>
	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\(HAND devHandle, short en\);](#)


参数：

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\(HAND devHandle, short *pEn\);](#)

参数：

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\(HAND devHandle, unsigned long paraID, long *pValue \);](#)

参数：

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\(HAND devHandle, unsigned long paraID, long value \);](#)

参数：

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：IP 地址等参数写成功后，需要调用 NMC_DevSetPara(devHandle, PARA_WRITE_EN, 1) 才能

使得写下去的参数保存，IP 地址等参数写成功后，将在控制器重新启动后生效

注：控制卡的 IP 地址不能设置成 169.254.x.x。

（20）修改控制器系统密码,密码用于修改系统时钟等

[NMC_ChangePassword\(HAND devHandle, char *pPasswordOld,char *pPasswordNew\);](#)

参数：

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\(HAND devHandle, char *pPassword\);](#)

参数：

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\(HAND devHandle, char *pUserName,char *pPasswordOld,char *pPasswordNew\);](#)

参数：

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\(HAND devHandle, char *pUserName,char *pPassword\);](#)

参数：

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\(HAND devHandle, char *pUserName\);](#)

参数：

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\(HAND devHandle,short period\);](#)

参数:

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) 获取控制器的规划周期

[NMC_GetProfilePeriod\(HAND devHandle,short *pPeriod\);](#)

参数:

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\(void\);](#)

参数: 无

(28) 设置指令错误返回值模式

[NMC_SetErrCodeMode\(short mode\);](#)

参数:

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\(HAND devHandle,long timeout,short stopMode,short groupID,long doValue\);](#)

参数:

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

6、数据采集功能

6.1、功能说明

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。

6.2、指令说明

（1）获取需要采集数据变量的地址

[NMC_GetCollectDataAddr\(HAND devHandle, short index,short dataType,unsigned long *pAddr, short *pDataLen\);](#)

参数：

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：

```

#define COLLECT_ADDRESS_PRF_POS           (0) // 规划位置
#define COLLECT_ADDRESS_AXIS_POS          (1) // 机械位置
#define COLLECT_ADDRESS_ENC_POS           (2) // 编码器位置
#define COLLECT_ADDRESS_CMD_POS           (3) // 命令位置
#define COLLECT_ADDRESS_AXIS_VEL          (4) // 电机速度
#define COLLECT_ADDRESS_CRD_POS           (5) // 坐标系位置
#define COLLECT_ADDRESS_CRD_VEL           (6) // 坐标系速度
#define COLLECT_ADDRESS_CRD_POS0          (5) // 坐标系位置
#define COLLECT_ADDRESS_CRD_VEL0          (6) // 坐标系速度
#define COLLECT_ADDRESS_CRD_POS1          (9) // 坐标系位置
#define COLLECT_ADDRESS_CRD_VEL1          (10) // 坐标系速度
#define COLLECT_ADDRESS_ENC_VEL           (7) // 编码器速度
#define COLLECT_ADDRESS_CMD_VEL           (8) // 命令速度
#define COLLECT_ADDRESS_LASER_OUTPUT      (11) // 激光输出（补偿前）
#define COLLECT_ADDRESS_LASER_GATE        (12) // 激光gate信号状态
#define COLLECT_ADDRESS_LASER_POWER       (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\(HAND devHandle, TCollectCfg *pCollect,TCollectTrig *pTrig \);](#)

参数:

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\(HAND devHandle, short en\);](#)

参数:

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) 获取采集状态

[NMC_GetCollectSts \(HAND devHandle, short *pSts, long *pDataLen \);](#)

参数:

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) 获取采集数据

[NMC_GetCollectData \(HAND devHandle, short len, unsigned char *pData \);](#)

参数:

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 \(HAND devHandle\);](#)

参数:

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

6.3、综合示例

```

short rtn=0;
    unsigned long addr[8];
    short len[8];
    //关闭采集
    rtn = NMC_CollectOnOff(devhandle,0);
    //清除采集状态
    rtn = NMC_ClearCollectSts(devhandle);
    //获取采集变量地址，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]);
    if(rtn != 0)
    {
        return;
    }
    rtn = NMC_GetCollectDataAddr(devhandle,0,COLLECT_ADDRESS_CMD_VEL,&addr[1],&len[1]);
    if(rtn != 0)
    {
        return;
    }

    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));
//采集模式无条件采集
cltTrig.mode = COLLECT_MODE_NONE;

rtn = NMC_ConfigCollect(devhandle, &g_cltCfg ,&cltTrig);
if(rtn != 0)
{
    return;
}

// 开始采集
rtn = NMC_CollectOnOff(devhandle,1);
if(rtn != 0)
{
    return ;
}

//启动一个运动
rtn = NMC_MtMovePtpAbs(axishandle[0],1,1,0,0,10,0,10000);
short cltSts;
long dataLen;
short axisSts;
short getLen;
double data[8192];
while(1)
{
    rtn= NMC_MtGetSts(axishandle[0],&axisSts);
    if ((axisSts&BIT_AXIS_BUSY)!=BIT_AXIS_BUSY)
    {
        break;
    }
    Sleep(10);
}

// 运动停止关闭采集
rtn = NMC_CollectOnOff(devhandle,0);
if(rtn != 0)
{
    return ;
}

//获取采集的数据量
```

```
getLen = 0;
rtn = NMC_GetCollectSts(devhandle,&cltSts,&dataLen);
if(rtn != 0)
{
    return;
}
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));
    if(rtn != 0)
    {
        return;
    }
}
rtn = NMC_GetCollectData (devhandle,dataLen%1440,(unsigned char *)(data+count*180));
if(rtn != 0)
{
    return;
}
```

7、坐标系变换功能

7.1、功能说明

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
旋转变换	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

7.2、指令说明

(1) 使能旋转变换处理

[NMC_CrdSetTransRotate\(HAND crdHandle,short rotAxisNo, double angleRadEqual,long firstAxisInitPos,long secAxisInitPos\)](#)

参数：

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\(HAND crdHandle\);](#)

参数:

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\(HAND crdHandle, short rotAxNo, short transAxNo, double rotEquiv\);](#)

参数:

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\(HAND crdHandle, double xPos, double yPos, double rotVel, double transVel\);](#)

参数:

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\(HAND crdHandle, double theta, double vel, short clrRoundFlag\);](#)

参数:

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\(HAND crdHandle\);](#)

参数:

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\(HAND crdHandle, short *pAxisMapArray \);](#)

参数:

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\(HAND crdHandle\);](#)

参数:

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\(HAND crdHandle, double _deltaTheta,long _deltaX,long _deltaY,long *pToolX,long *pToolY \);](#)

参数:

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\(HAND crdHandle, double _pulse2Rad,long toolX,long toolY \);](#)

参数:

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) 获取 XYZA 机型参数

[NMC_CrdGetXYZAPara\(HAND crdHandle, double *pPulse2Rad, long* ToolX, long *pToolY\);](#)

参数:

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

7.3、综合示例

a.旋转变换例程

```
//先建立插补坐标系,过程略

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);
if (rtn)
{
    return;
}

rtn = NMC_CrdBufClr(crdhandle);
if (rtn)
{
    return;
}

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);
//运动到原点
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,15,posArray,endVel,vel,acc,0);

rtn = NMC_CrdBufSetPtpMovePara(crdhandle,segNo++,3,rotVel,1,0);
if (rtn)
{
    return;
}

posArray[0] = (0.5*m_l - m_r)*mmToPluse;
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,3,posArray,endVel,vel,acc,0);

rtn = NMC_CrdBufAxMoveRel(crdhandle,segNo++,8,&rotPos,0,1);
posArray[0] = (0.5*m_l)*mmToPluse;
posArray[1] = (m_r)*mmToPluse;
rtn = NMC_CrdArcRadiusEx(crdhandle,segNo++,posArray,(m_r)*mmToPluse,1,endVel,vel,acc,0);

posArray[0] = (0.5*m_l)*mmToPluse;
posArray[1] = (m_w - m_r)*mmToPluse;
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,3,posArray,endVel,vel,acc,0);

rtn = NMC_CrdBufAxMoveRel(crdhandle,segNo++,8,&rotPos,0,1);
posArray[0] = (0.5*m_l - m_r)*mmToPluse;
posArray[1] = (m_w)*mmToPluse;
rtn = NMC_CrdArcRadiusEx(crdhandle,segNo++,posArray,(m_r)*mmToPluse,1,endVel,vel,acc,0);

posArray[0] = (-0.5*m_l + m_r)*mmToPluse;
posArray[1] =(m_w)*mmToPluse;
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,3,posArray,endVel,vel,acc,0);

rtn = NMC_CrdBufAxMoveRel(crdhandle,segNo++,8,&rotPos,0,1);
posArray[0] = (-0.5*m_l)*mmToPluse;
posArray[1] = (m_w - m_r)*mmToPluse;
rtn = NMC_CrdArcRadiusEx(crdhandle,segNo++,posArray,(m_r)*mmToPluse,1,endVel,vel,acc,0);

posArray[0] = (-0.5*m_l)*mmToPluse;
posArray[1] = (m_r)*mmToPluse;
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,3,posArray,endVel,vel,acc,0);
```

```
rtn = NMC_CrdBufAxMoveRel(crdhandle,segNo++,8,&rotPos,0,1);
posArray[0] = (-0.5*m_l + m_r)*mmToPluse;
posArray[1] = 0;
rtn = NMC_CrdArcRadiusEx(crdhandle,segNo++,posArray,(m_r)*mmToPluse,1,endVel,vel,acc,0);

posArray[0] = 0;
posArray[1] = 0;
rtn = NMC_CrdLineXYZEx(crdhandle,segNo++,3,posArray,endVel,vel,acc,0);

rtn = NMC_CrdEndMtn(crdhandle);//结束缓冲区
if (rtn)
{
    return;
}
rtn = NMC_CrdStartMtn(crdhandle);//启动缓冲区插补
if (rtn)
{
    return;
}
```

b.极坐标变换例程（略）

c.XYZA 机型变换例程

//先建立插补坐标系,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);
if (rtn)
{
    return;
}
//设置工具参数
rtn = NMC_CrdSetXYZAPara(crdhandle,pulse2Rad,toolX,toolY);
if (rtn)
{
    return;
}
short pMapAxisArray[4]={0,1,2,3};
//设置 XYZA 机型
```

```
rtn = NMC_CrdSetTransXYZA(crdhandle,pMapAxisArray);  
if (rtn)  
{  
    return;  
}
```

第四章 GCS.exe 使用简介

GCS.exe 是控制器的演示软件，通过 GCS.exe 可以测试控制器的各项运动功能以及 IO 功能。

一、运行 GCS.exe

运行 GCS.exe，启动窗口。在点击“扫描并连接”按钮之前，请确认控制器已经通过网口与 PC 机连接，确保控制器的 IP 已经配置在 PC 机的一个网段中（控制器默认网络地址为 192.168.1.110，可以在“功能—控制器信息”中设置修改）。

点击按钮，有设备连接显示如下：



连接上控制器后，可以在功能菜单中选择要测试的项目



二、测试单轴运动

轴测试窗口可以进行单轴的测试，配置，回零

1、测试



状态显示框显示指令位置，实际位置，电机速度（命令速度），单轴状态。

轴切换可以切换所要测试的轴，也可在“功能——轴测试”三角形区域打开多个轴测试窗口。

配置——可以设置报警激活，脉冲输出方式，直接生效。

运动设置：

默认 PTP 点位运动，“正向”运动到正数“运行位置”，“负向”运动到负数“运行位置”；

“速度模式”打钩，Jog 模式，按住“正向”“负向”按钮运动，松开停止；

“相对运动”打钩，PTP 模式；

2、配置

轴切换: Axis 1

测试 配置 回零

☒ 正向限位激活 ☒ 软限位激活

☒ 负向限位激活 正向最大行程: 2147483647

☐ 电机报警激活 负向最大行程: -2147483648

脉冲模式: 平滑系数: 0

脉冲+方向 正向限位电平: 急停减速度: 32767

低电平 最大速度: 32767

负向限位电平: 最大加速度: 32767

低电平 最大位置误差: 30000

伺服报警电平: 编码器源: 内部脉冲计数

低电平 编码器信号: 脉冲+方向

☐ 指令方向取反

更新

配置单轴运动的相关参数，点击“更新”生效。

3、回零——单轴回零测试

轴切换: Axis 1

测试 配置 回零

参数

回零模式: 单原点 规划位置: 0

最大行程: 0 p 实际位置: 0

回零搜索速度: 20.00 p/ms

回零返回速度: 5.00 p/ms

回零原点偏置: 0 p

回零加速度: 1.00 p/ms^2

☐ 负向回零 ☐ 原点上升沿触发

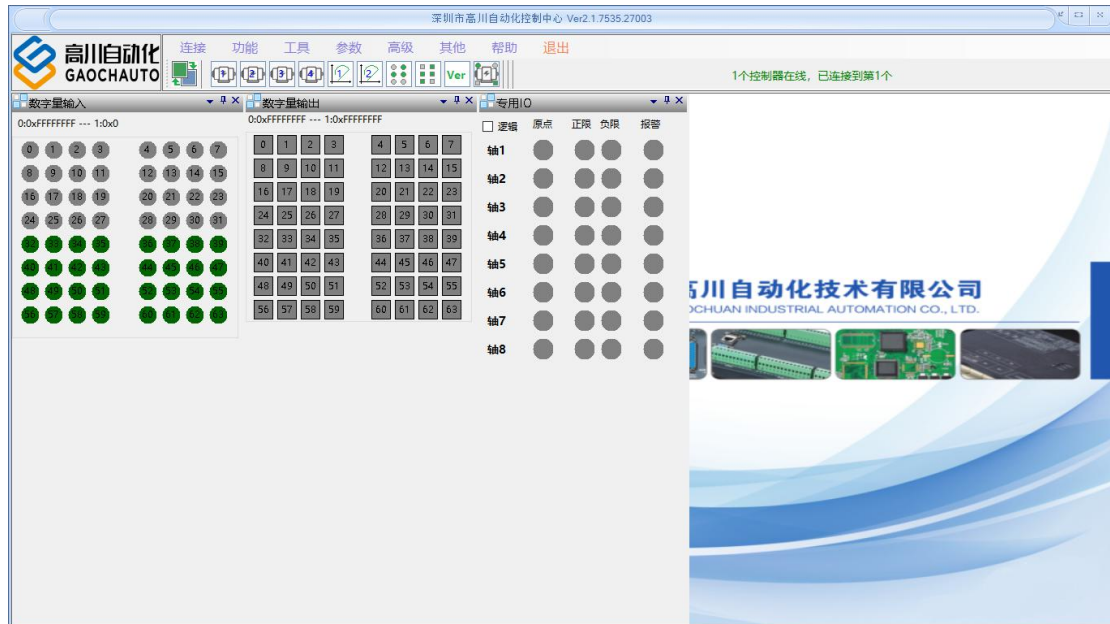
☐ 二次回零 ☐ 限位上升沿触发

☐ 不清位置 ☐ Z上升沿触发

开始回零

停止回零

三、IO 测试



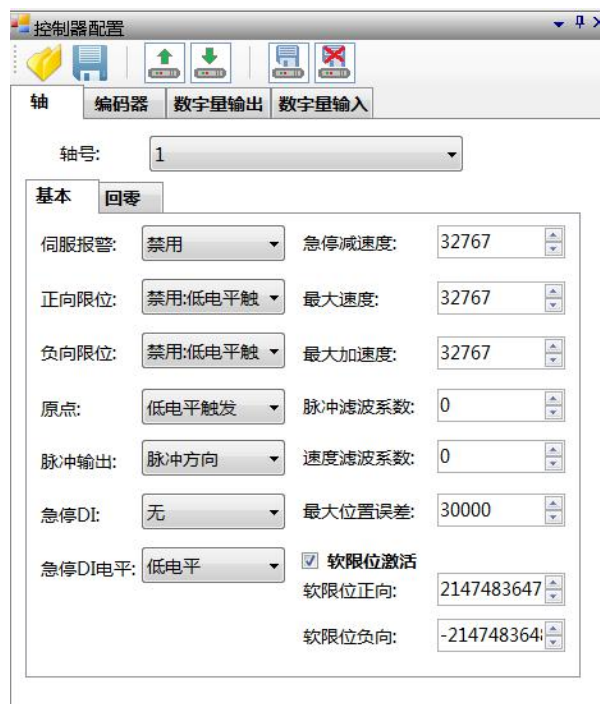
可以在“功能”或者快捷栏中点开窗口测试通用IO 和专用IO

四、参数配置器

打开菜单中“参数——参数配置器”。

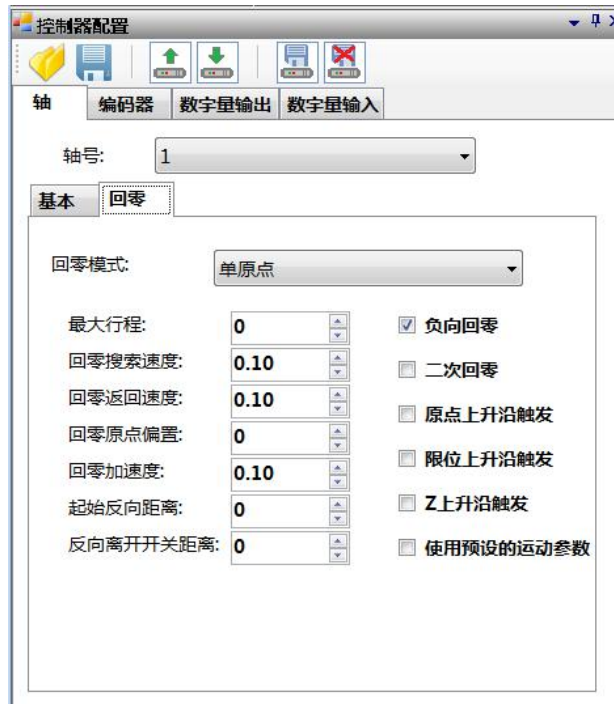
轴号：可切换轴进行参数配置。

配置由用户实际情况进行设置，然后下载配置，可在外部保存为文件（文件可以在另外控制卡打开进行同样的配置）。

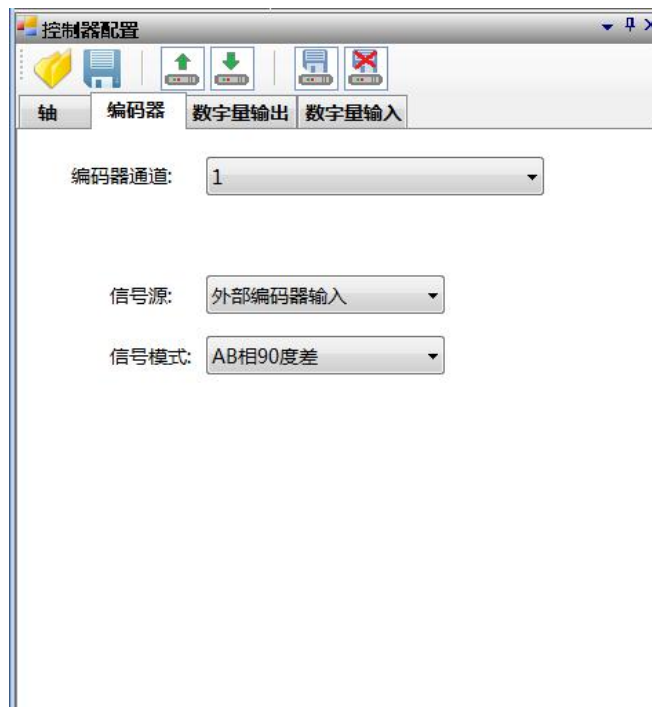


回零配置：起始反向距离： 起始反向运动距离（可选，不用时设为0）；

反向离开开关距离： 反向运动时离开开关距离（可选，不用时设为 0）。



编码器配置，信号源为外部编码器时，信号模式一般为 AB 相

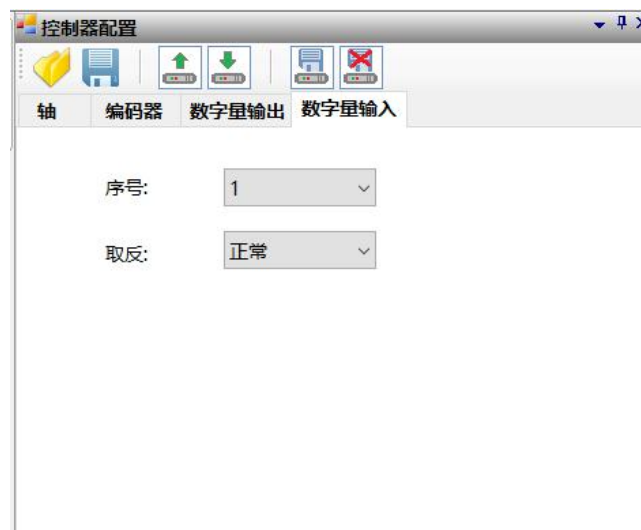


数字量输出取反设置：序号： 1 代表 DO0（按序号排列）；

取反： 设置对应序号位的电平信号取反。



数字量输入配置：序号： 1 代表 DI0（按序号排列）；
取反： 设置对应序号位的电平信号取反。



五、坐标系运动



这个窗口可以进行多轴联动（坐标系）运动测试。

1、指令——可以插入以下八种指令



从左到右分别是：“直线插补”“圆弧插补”“缓冲区等待DI”“缓冲区输出DO”“缓冲区延时”“螺旋线插补”“缓冲区等待位置”“缓冲区单轴跟随运动”；
插入的指令会显示在指令列表当中；

2、运动——可以测试指令列表中的指令



- “最大轴数”——设置坐标系的轴数，最大支持4轴；
- “最大速度”——坐标系运动的最大速度；
- “最大加速度”——坐标系运动的最大加速度；
- “☐使用前瞻预处理”——勾选表示插补启用前瞻预处理；
- “☐以当前位置为原点”——勾选表示插补运动坐标原点为当前位置；
- “☐清除缓冲区”——勾选表示“下载数据”前清除缓冲区；
- “配置”——配置坐标系启用；
- “下载数据”——下载指令列表内的数据；
- “启动”——启动坐标系运动；
- “停止”——停止坐标系运动；
- “其他>>”——坐标系其他功能；

六、激光控制测试

打开菜单中“功能——激光控制测试”。

激光控制测试

通道选择:

1

物理信号类型:

DA输出

物理通道:

0

选项值:

10

激光控制模式:

波形控制模式

更新

时间数组

组号选择:

1

位置源:

外部编码

轴掩码:

1

位置间隔:

100

立即输出

点

t(10000),p(10000)

t(10000),p(20000)

t(10000),p(10000)

时间:

5000

us

能量:

10000

增加

删除

修改

在此功能窗口下可进行激光能量控制测试并可对激光通道、信号类型以及对应的信号值、控制模式和输出能量为多少进行配置，点击“更新”生效。

选项值： 当作为占空比输出时，该值为 PWM 的频率，单位 HZ；

当为频率输出时，该值作为占空比值，（0~100）；

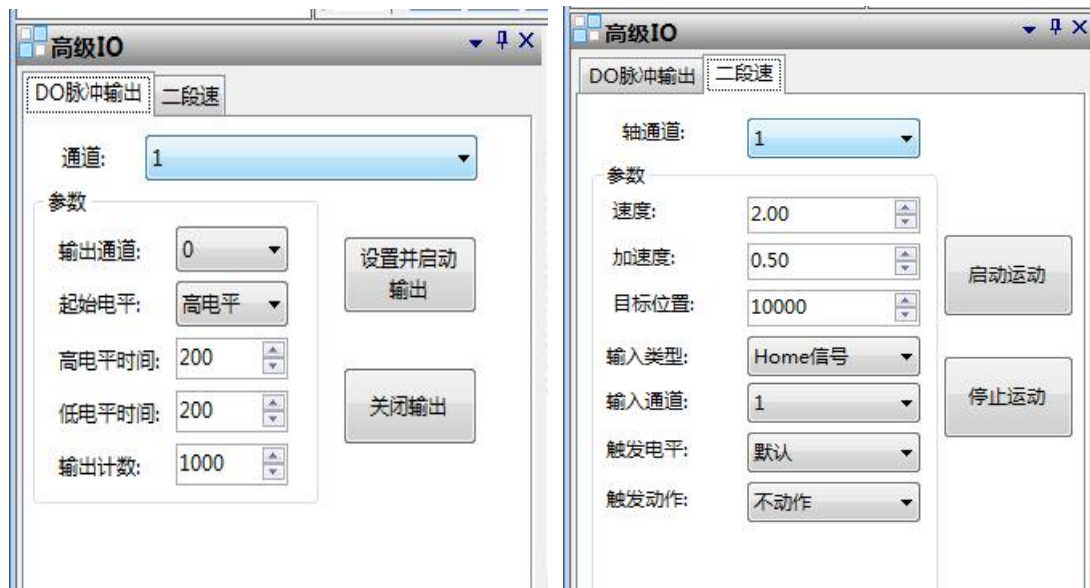
当为频率输出，脉宽固定时，该值为脉宽，单位为 0.5 微秒，取值范围(0,~)

激光控制模式：当为基本控制模式时可在下方设置激光能量，点击“更新能量”生效；

当为波形控制模式时可在下方设置激光能量波形；

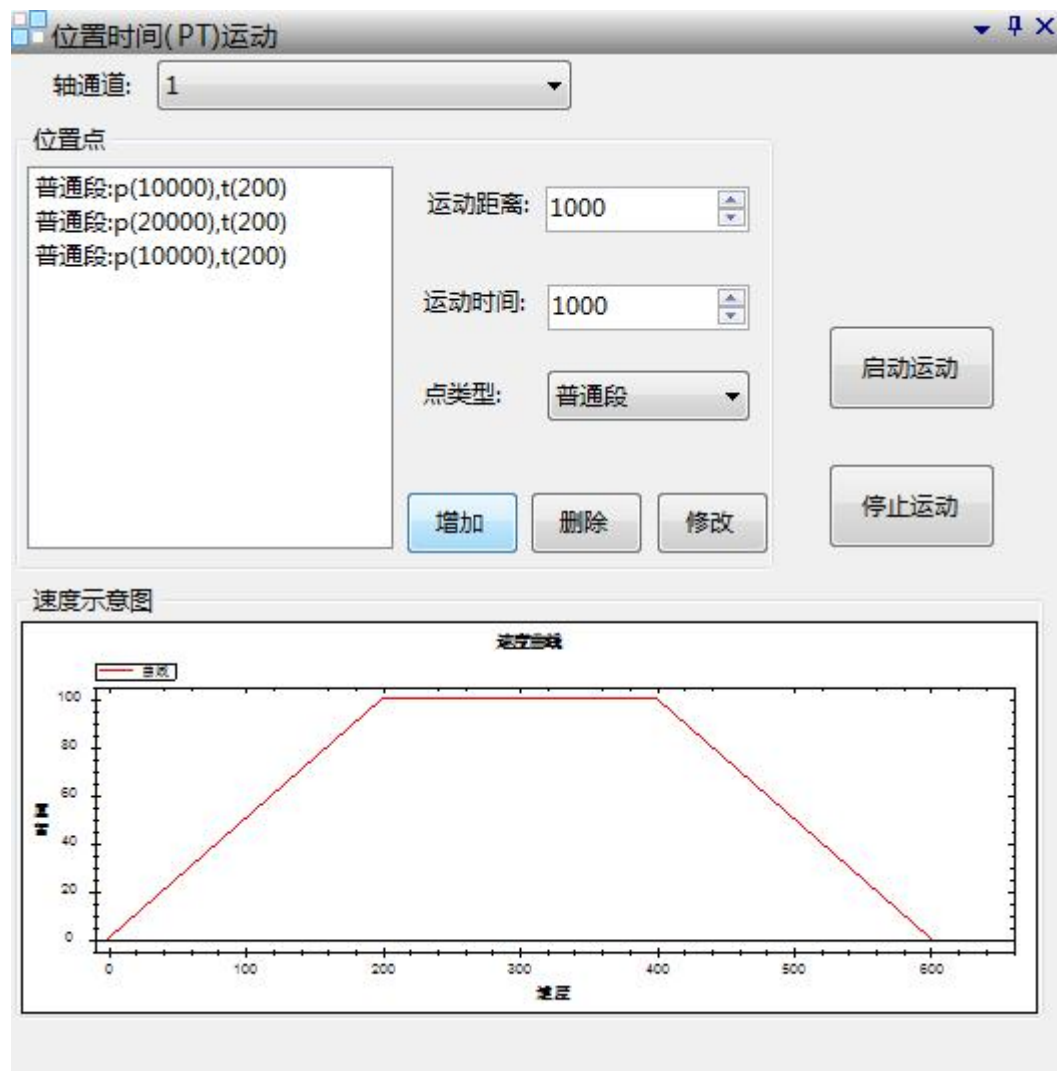
当为位置比较控制模式时可在下方设置比较间隔参数等。

七、高级 IO 功能



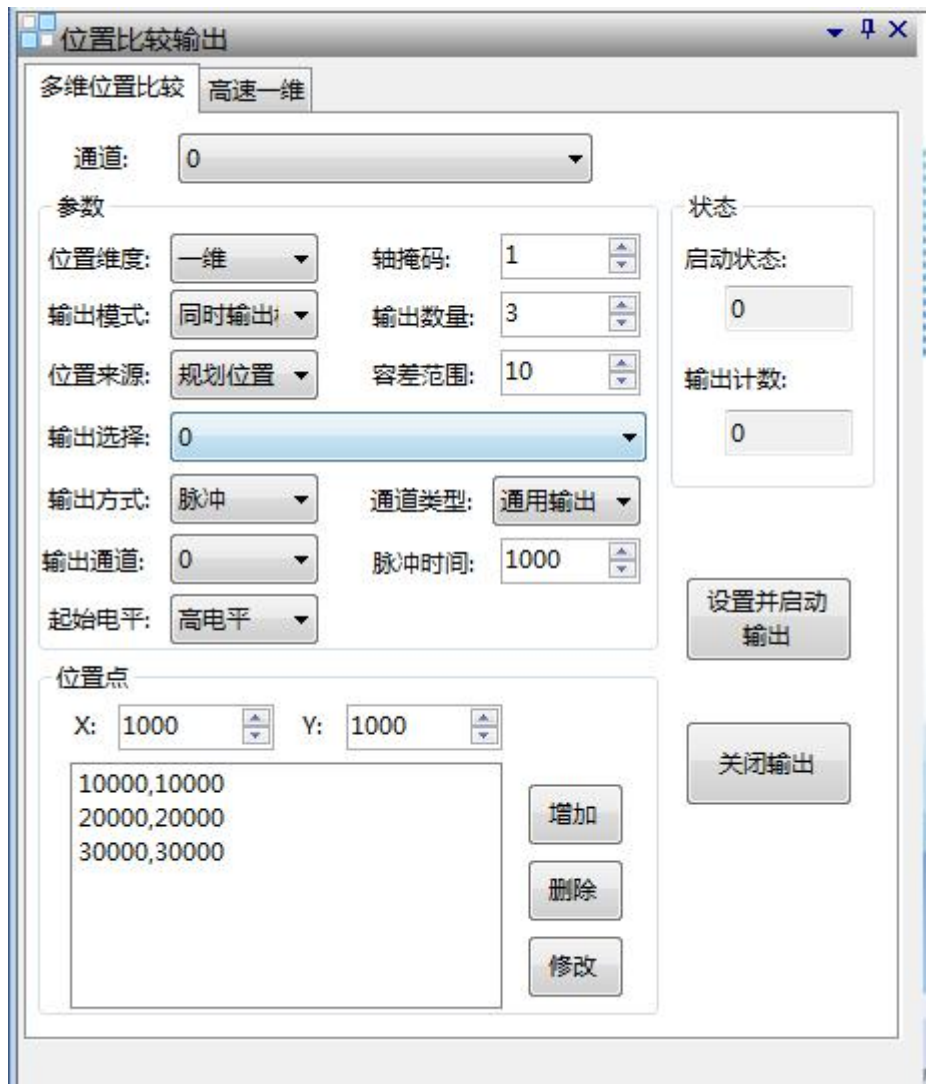
此功能模块可以测试 DO 脉冲输出和 DI 触发改变速度的二段速功能。

八、位置时间（PT）运动



此功能模块可以测试 PT 运动功能，增加一系列 PT 运动数据点，距离单位脉冲，时间单位 ms，可以在速度示意图观察到速度曲线是否合理连续。

九、位置比较输出



此功能模块可以测试位置比较输出功能。

十、其他资源（辅助编码器等）

打开菜单中“功能——其他资源”。

可配置辅助编码器的信号源与信号类型，查看编码器通道返回的位置参数。

可配置模拟量输出通道、范围与查看读数。

可配置模拟量输入通道、范围与查看读数。

其他资源 (辅助编码器)

辅助编码器
模拟量输出
模拟量输入

通道: Encoder 1

读数: 0

清零

配置

信号源: 外部信号输入

信号类型: AB相, 90度

☐ 方向取反

设置

其他资源 (辅助编码器)

辅助编码器
模拟量输出
模拟量输入

通道: DAC 1

范围: -10 ~ 10V

读数: 0

设定:

设置范围

其他资源 (辅助编码器)

辅助编码器
模拟量输出
模拟量输入

通道: ADC 1

范围: -10 ~ 10V

滤波: 0

读数: 0

☐ 显示电压值

设置参数

十一、通用 IO 扩展模块

通用IO扩展模块

扩展IO地址： 2

模块类型： IO32_DA

使能

IO

输出

通道1:	0		设置
通道2:	0		设置
通道3:	0		设置
通道4:	0		设置
通道5:	0		设置

输入

通道1:		通道4:	
通道2:		通道5:	
通道3:			

此功能测试可以测试模拟量模块。

十二、手轮测试。



打开菜单中“功能——手轮测试”。

在此功能窗口下可测试手轮功能；

电机选择：设置跟随手轮的电机；

手轮的运行的加速度、倍率；

运行速度：设置手脉的最大速度；

扩展编码器通道：与其他资源中的辅助编码器对应，设置手脉所对应的编码器（默认为通道一）。

十三、龙门测试

打开菜单中“功能——龙门测试”。



龙门的主动轴与从动轴：选择对应的电机位；
组号：龙门组号,取值范围[0,n]；
允许的误差：龙门保护误差,取值范围(0,...)。

十四、IO 扩展模块

打开菜单中“功能——IO 扩展模块”。



在此功能窗口下可快速查看测试扩展 IO 模块。

扩展 IO 地址对应为硬件上设置的地址，范围为（2——9）。

当模块连接上并地址正确对应时，窗口地址旁边的灯（下图所指）会变绿。



十五、位置捕获

打开菜单中“功能——位置捕获”。

位置捕获		捕获模式	捕获源	捕获电平	捕获位置	
轴1	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴2	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴3	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴4	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴5	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴6	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴7	IO捕获	原点输入作为捕获IO	下降沿		启动	
轴8	IO捕获	原点输入作为捕获IO	下降沿		启动	

在此功能窗口下可通过设置捕获模式、捕获源、捕获电平来捕获查看位置误差。

十六、数据采集

打开菜单中“功能——数据采集”。

数据采集

状态: Stop

计数: 6790

配置

开始采集

停止采集

打开数据文件

数据采集

状态: Stop

计数: 357

曲线选择

曲线: 规划位置

通道: 1

增加

删除

规划位置:1

数据存储路径:

d:\collect.txt

采集间隔: 1

保存

关闭

在此功能窗口下可点击“配置”，对采集数据曲线类型和通道进行选择并点击增加，不需要者可点击后删除便可以。

采集的数据可在数据存储路径自行设置（每次采集如果想保存请先在外部另存再进行下一次采集）。

数据的采集间隔时间单位为 ms，请根据实际情况进行间隔修改。

十七、控制器信息



控制器信息窗口可以读取控制器的 IP 地址和相关版本信息，也可以设置控制器的 IP，设置后需要断电重启才生效。

十八、系统时间及密码

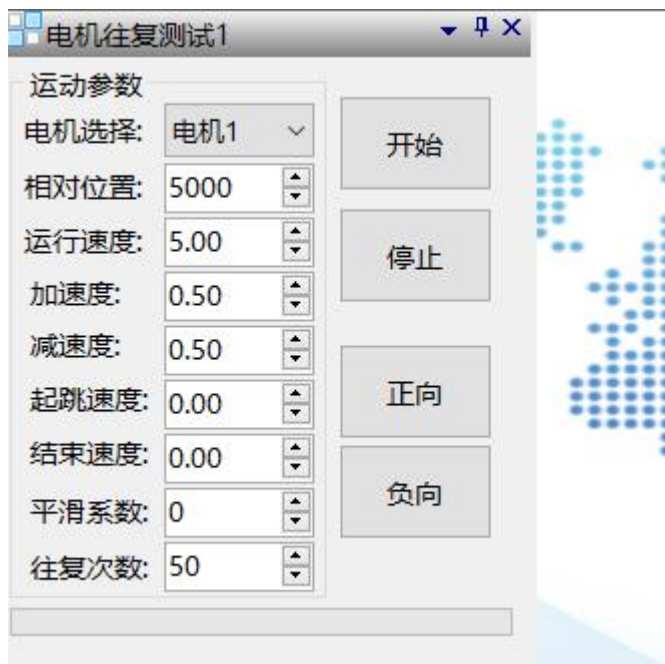
打开菜单中“功能——系统时间及密码”。



在此功能窗口下可修改控制器时间以及设置系统密码。

十九、电机往复测试

打开菜单中“工具——电机往复测试”。
可通过打开不同电机往复测试打开多个窗口。



电机选择： 进行往复测试的轴。
相对位置： 电机进行往复运动到达两端的终点绝对位置。
运行速度、加减速速度、起跳与结束速度按需要调整。

平滑系数： 如无特定需求一般默认。

往复次数： 往复运动的次数。

点击正负向确定运动起始往哪个方向开始运动。（点击开始默认正向）。

二十、错误代码查询

打开菜单中“工具——错误代码查询”。



在此功能窗口下：

在错误号中输入程序或者 demo 反馈回来的错误码进行查询。

具体错误信息会在错误描述中输出。

二十一、指令调用监听器

打开菜单中“工具——指令调用监听器”。

“过滤器”中可设置打印指令的具体范围。

指令关键字：如果在“指令关键字”中输入指令则只会打印出输入的指令（如空白将全部打印）；

忽略指令关键字： 将不打印在此栏输入的指令。

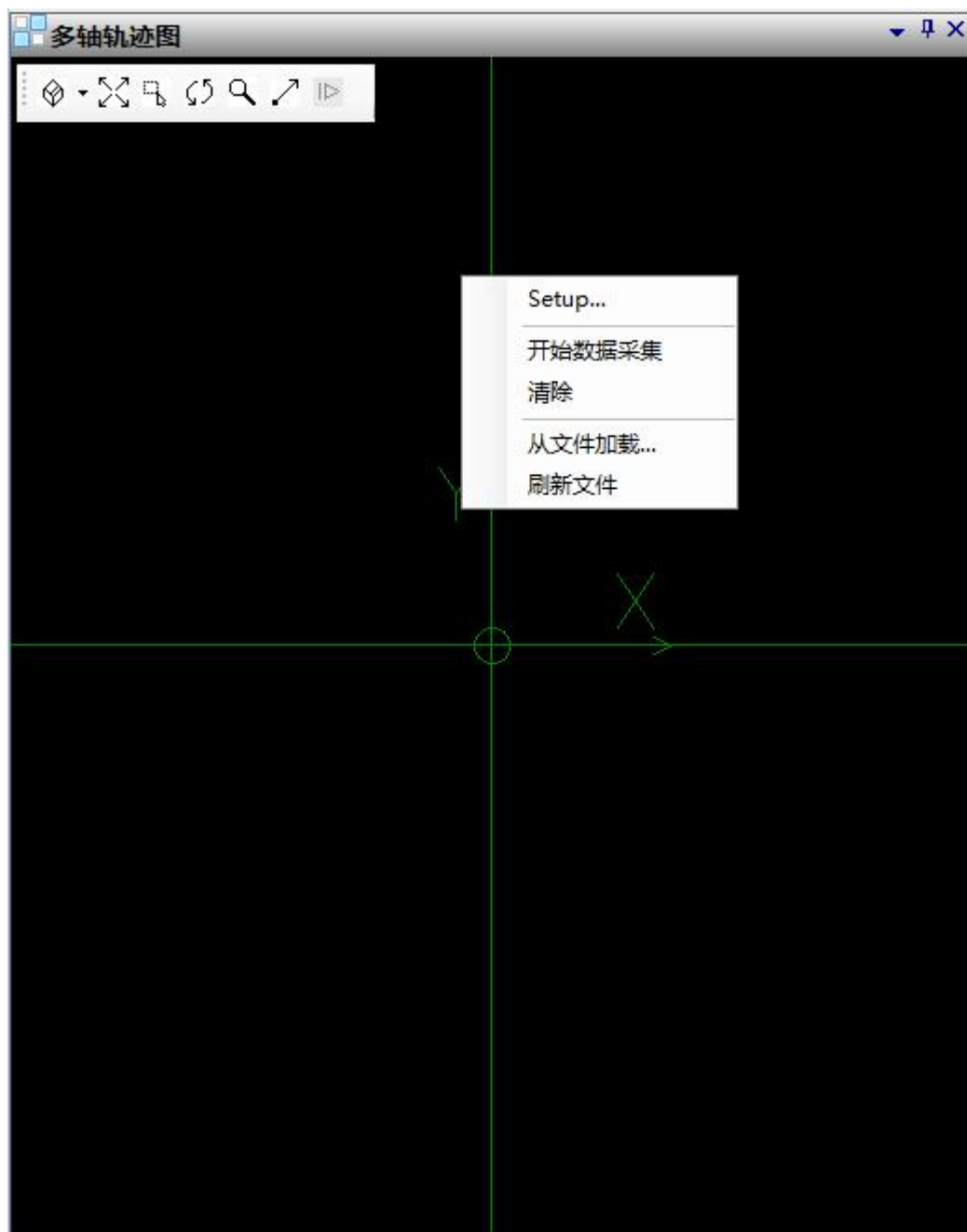


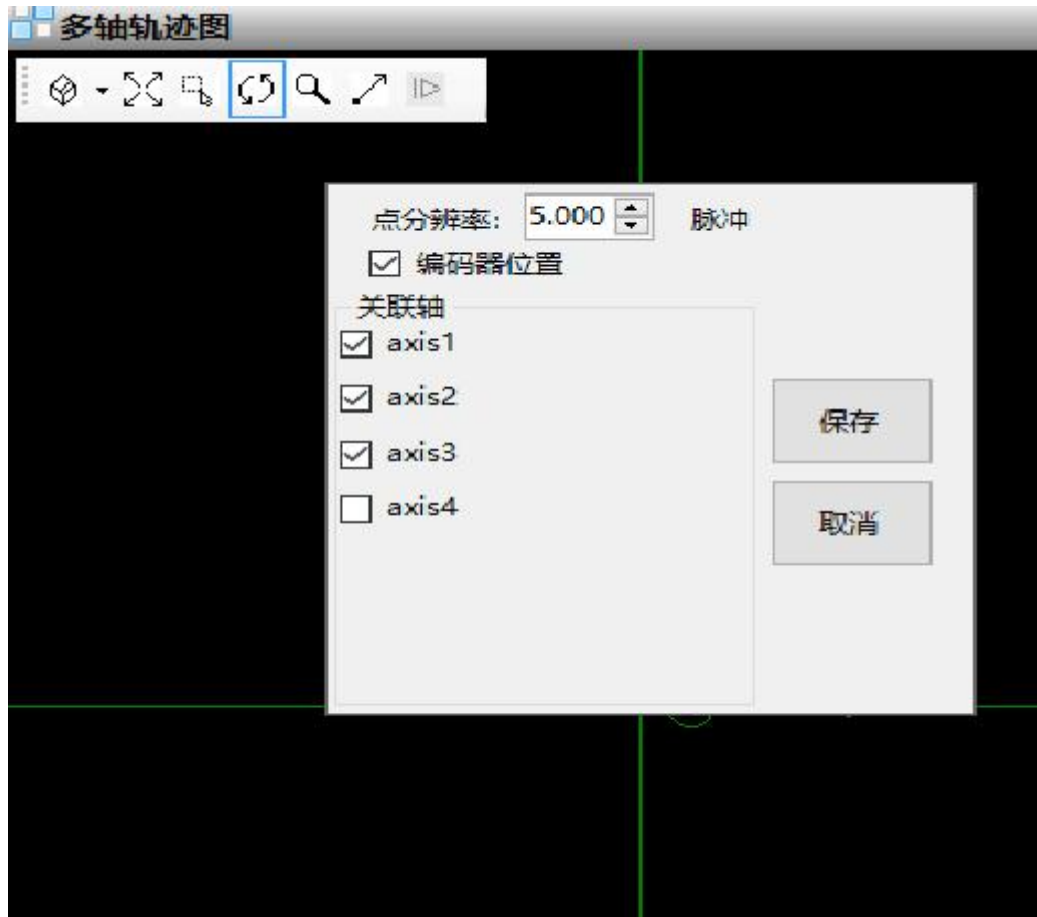
二十二、多轴轨迹图

打开菜单中“工具——多轴轨迹图”。

在此功能窗口下点击“右键”——“Setup”设置点分辨率与关联轴号。

开始数据采集： 点击“右键”——“开始数据采集”。





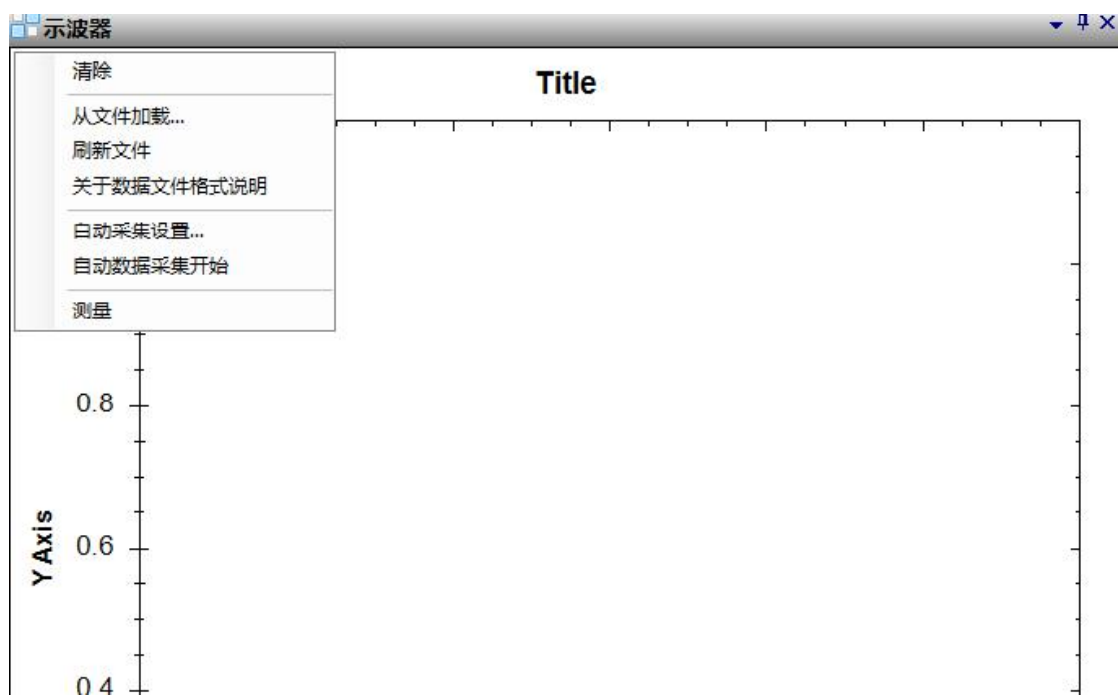
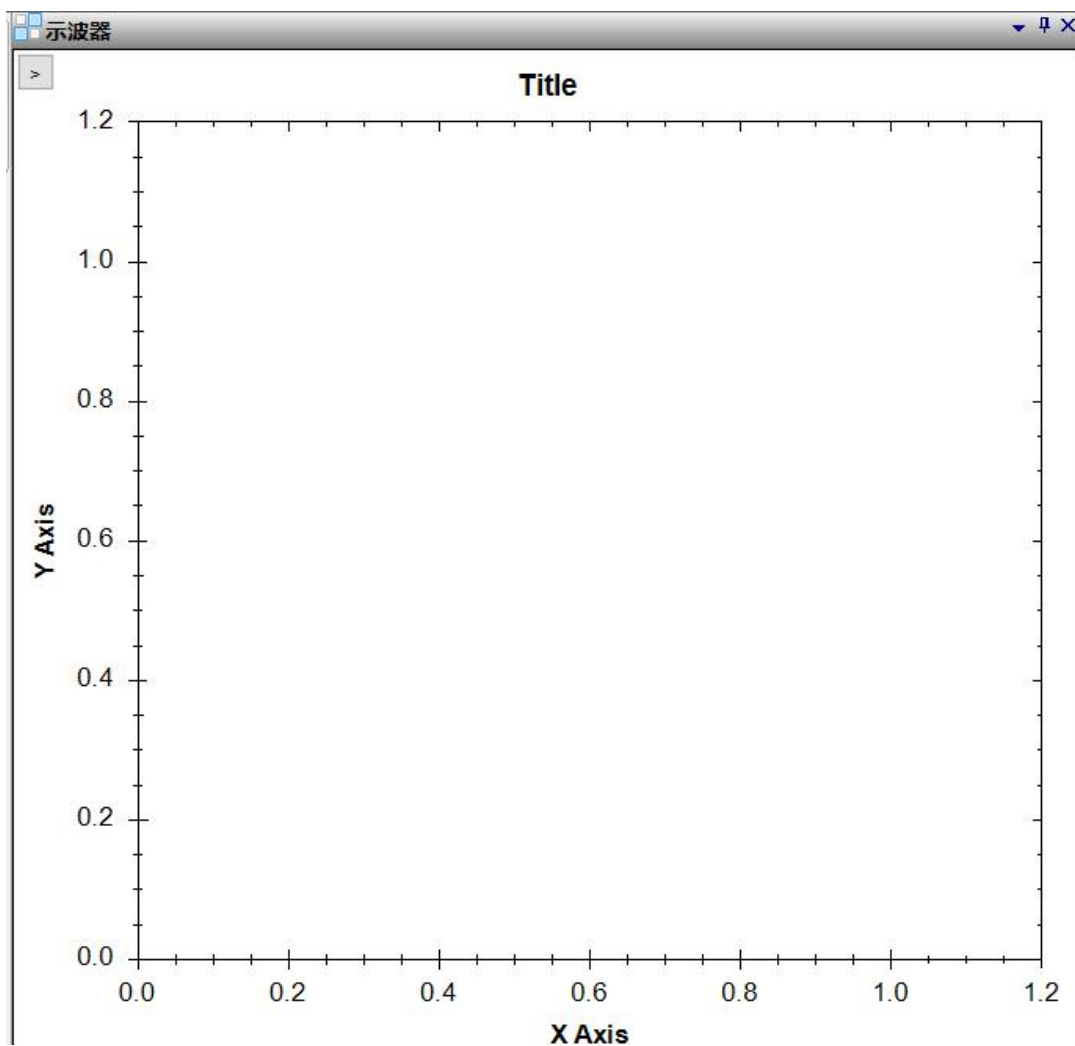
二十三、示波器

打开菜单中“工具——示波器”。

在此功能窗口下点击左上角箭头会出现多种功能选项。

在“自动采集设置”下设置显示的所代表的数据曲线及通道（所测试的轴号）增加后保存即可。（最多可同时采集 5 条曲线的数据显示）

采集间隔： 单位为 ms。





二十四、NMC 指令测试

打开菜单中“工具——NMC 指令测试”。

在此功能窗口下可选择需要调用的指令设置好参数，可添加后一起执行或者直接执行。结果会在下方显示，报错可在“[错误代码查询](#)”里查询。

过滤：按相关字查询需要的指令。

NMC指令测试

指令列表

过滤:

- NMC_GetDIGroup
- NMC_GetDOGroup**
- NMC_ConfigCollect
- NMC_CollectOnOff
- NMC_GetCollectSts
- NMC_GetCollectData
- NMC_ClearCollectSts
- NMC_GetUID
- NMC_GetDllVersion
- NMC_GetMtLibVersion
- NMC_DevGetPara
- NMC_DevSetPara
- NMC_GetLastError
- NMC_SetErrCodeMode
- NMC_UserCmdWrite
- NMC_UserCmdRead
- NMC_DataTransfer
- NMC_SetDebugErrorEn
- NMC_CrdSetTransRotate
- NMC_CrdDelTransRotate
- NMC_CrdSetTransPolar
- NMC_CrdRunToPolarPos

参数区

devHandle:

pDoValue:

groupID:

NMC_GetDIGroup(1,0,0)
 NMC_GetDOGroup(0,0,0)

NMC_GetDIGroup(1,0,0) = 0

pInValue:-32513

