迈科讯 | AMC Series |

运动控制函数库
使用手册
(QT版)

文档版本

版本号	修订日期	修订内容
V1.00	2019.11.07	初订
V1.01	2019.11.15	修改关于 SDK 环境和 QT5.6.2 配置
V1.02	2022.12.12	新增总线函数
	2025.4.10	更改封面

目录

第一章　运动控制函数库的使用

1.1 运动控制函数库简介

AMC 系列运动控制器，在硬件上由 Arm 实现所有运动功能，并运行于实时操作系统 RT-LINUX。内含运动控制函数库(Motion Control Command Library. MCCL)，可供开发者在A^+PC 模式或单机（Standalone）模式下调用。

A^+PC 模式，用户可以通过个人电脑在 Windows 平台下用熟悉的 VC、VB 等工具来开发、编译、执行应用程序。上位机应用程序通过 Ethernet 网络口与 IMC 控制器进行通信，运动控制功能皆由 IMC 负责运算处理；人机界面及其他应用功能，则由个人电脑负责处理。该模式下的具体应用请参照 VC 版用户使用手册。

Standalone（单机）模式，用户可直接在控制器自带的 Linux 操作系统下，使用 QT 工具来开发、编译和运行应用程序。本章将详细讲解单机模式下的 Linux 开发环境搭建。本手册所有例程基于 QT 开发工具。该模式下用户可选择脉冲或模拟量的传统控制方式，也可选择 EtherCAT 总线方式。

AMC 提供 EtherCAT 总线控制方式，目前支持的 EtherCAT 从站有英威腾、安川、三洋、清能德创总线伺服以及智鼎、倍福 IO 等模块。并提供有丰富的本地 IO 资源，最多可支持 32 轴，用户可以快速方便的建立自己的 EtherCAT 总线控制网络。

1.1.1 运动函数库特点

MCCL 以笛卡尔正交运动坐标系为基础建立 Group，具有很高的指令整合度，客户可以非常方便的实现坐标系各轴的插补、同动以及点位运动。这是该控制器的最大特点之一。在这里，我们将所有的插补运动定义为一般运动。MCCL 提供一般运动、点位运动及 Jog 运动，一般运动包括空间直线、圆弧、圆、螺旋线等插补运动。

MCCL 提供近端 GPIO 及编码器等硬件资源访问，有 16 进 16 出 GPIO 有 1 路编码器或手摇轮输入。

在模拟量输入输出方面，可提供 12bit 精度的模拟电压（-10V ~ 10V）输出 3 个，实现模拟量输出功能。

在计时功能方面，使用者可设定计时器计时时间，当启动计时功能并在计时终了时，可自动触发自定义中断程序，并重新开始计时。此过程将持续至关闭该功能为止。MCCL 也提供 Watch Dog 的功能。

使用 MCCL 并不需要深入了解运动控制中复杂的轨迹规划、定位控制、实时多线程的环境，利用此函数库即可快速开发、整合系统。

1.2 运动控制函数库的使用

1.2.1 运行环境

■Standalone 模式操作系统环境

Linux 系统

■Standalone 模式开发环境

QT 5.4.2

■Standalone 模式使用 MCCL 时，工程中所需要加的文件

	文件名称
QT	mccl.h MCCL_Fun.h amcgpio.h kinematics.h libAmcDrv.so libAmcArmMccl.so libAmcArmRobot.so libAmcGpio.so

1.2.2 Linux 系统下运行环境搭建

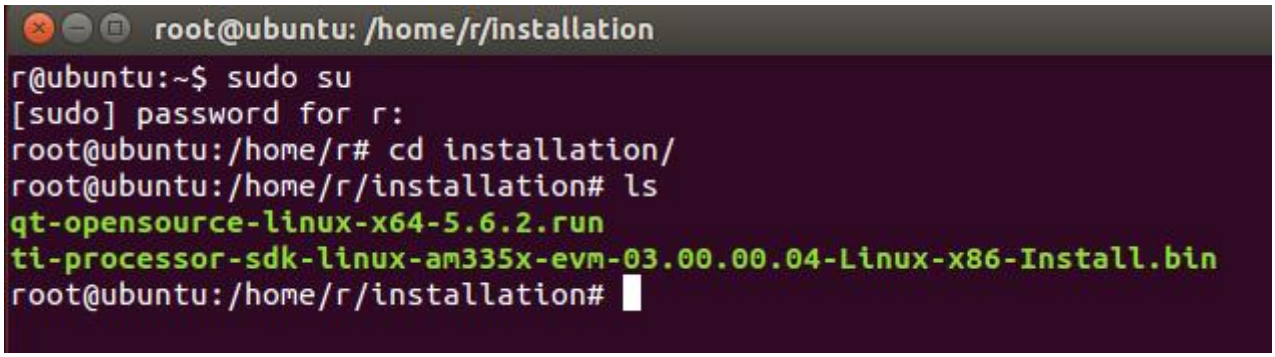
Standalone Linux QT 环境搭建

1.安装 ARM 调试环境:

注：此安装包需要在 64 位的 Ubuntu 系统下才能正常工作。

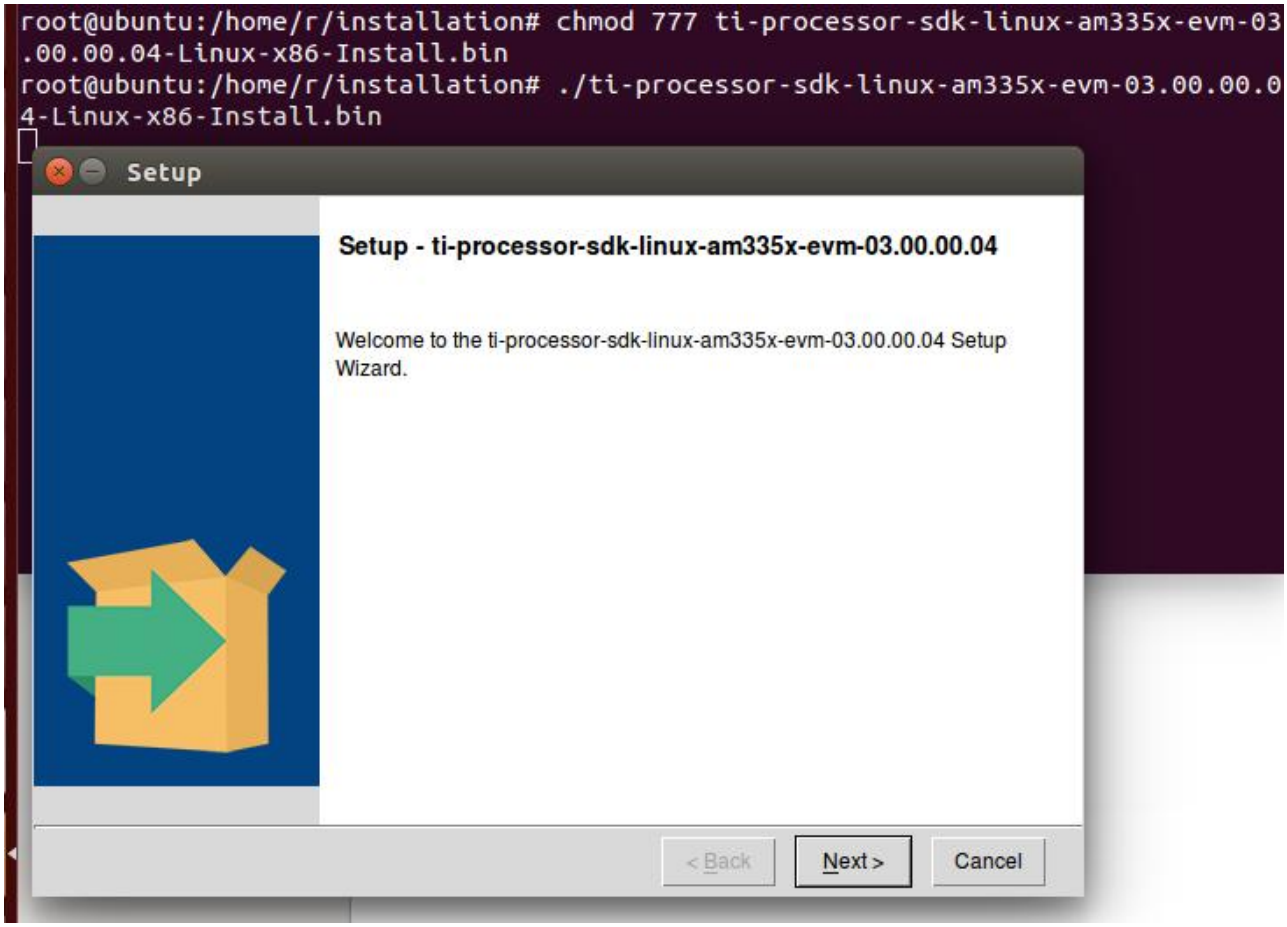
安装 QT 开发环境前，需要安装本控制器的 ARM 调试环境 TI SDK,此处安装包名称为 ti-processor-sdk-linux-am335x-evm-03.00.00.04-Linux-x86-Install.bin，其安装过程如下:

(1)将 ti-processor-sdk-linux-am335x-evm-03.00.00.04-Linux-x86-Install.bin 拷贝至 Linux 系统 home/r/installation/目录下,在控制台下进入 root 账户并进入 installation 文件夹可输入指令 ls 查看当前目录下的资源文件：

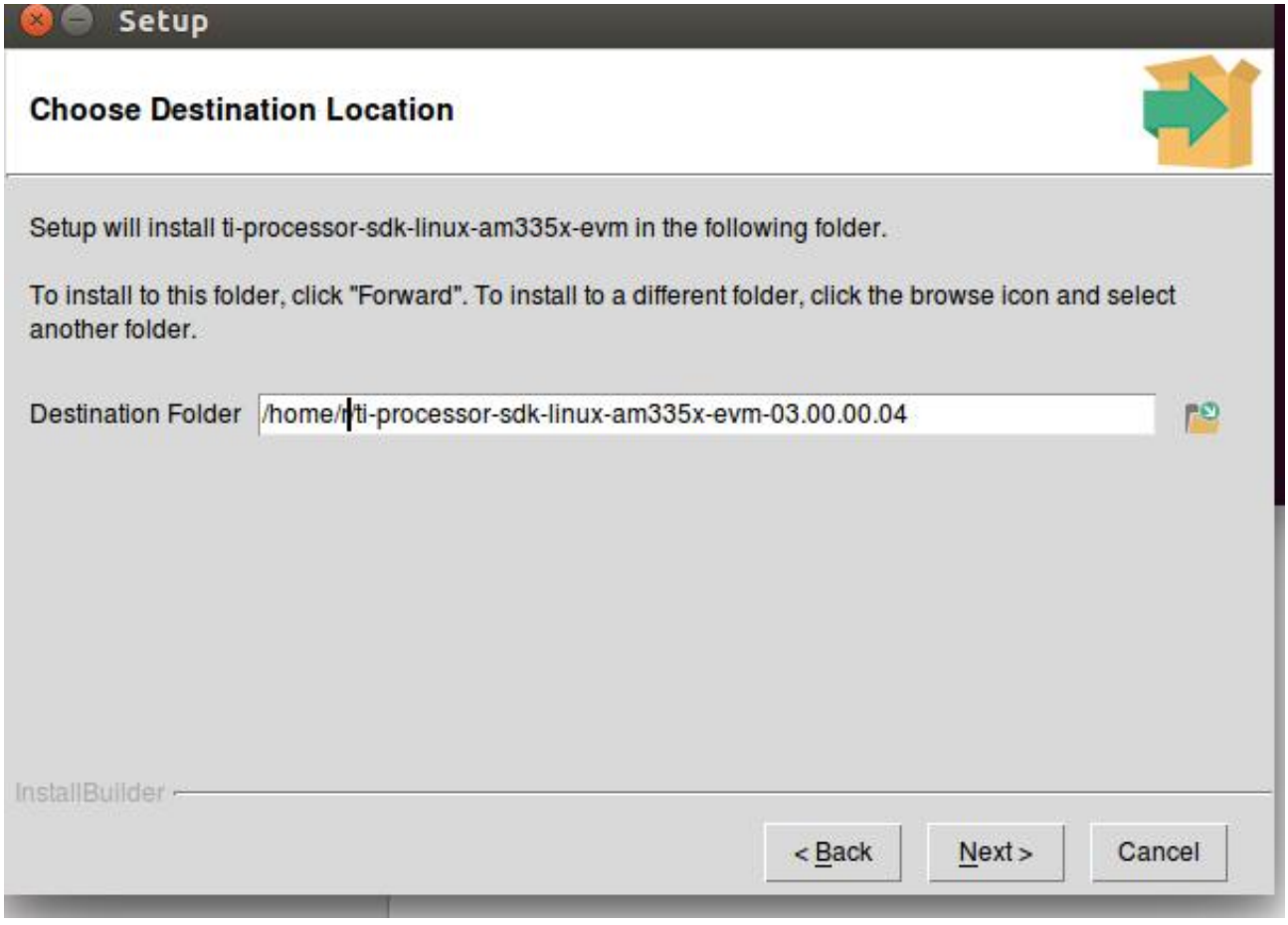


(2)启动控制的 Linux 系统下的控制台工具(快捷键为 Ctrl+Alt+T)，在控制台下输入命令 chmod 777 ti-processor-sdk-linux-am335x-evm-03.00.00.04-Linux-x86-Install.bin，如下图:

(3)控制台输入指令./i-processor-sdk-linux-am335x-evm-03.00.00.04-Linux-x86-Install.bin，会弹出 ti-sdk 的安装向导，如下图:



(4)按向导一直点击下一步，选择好安装路径，可以选择自己喜欢的路径，等待安装完成.

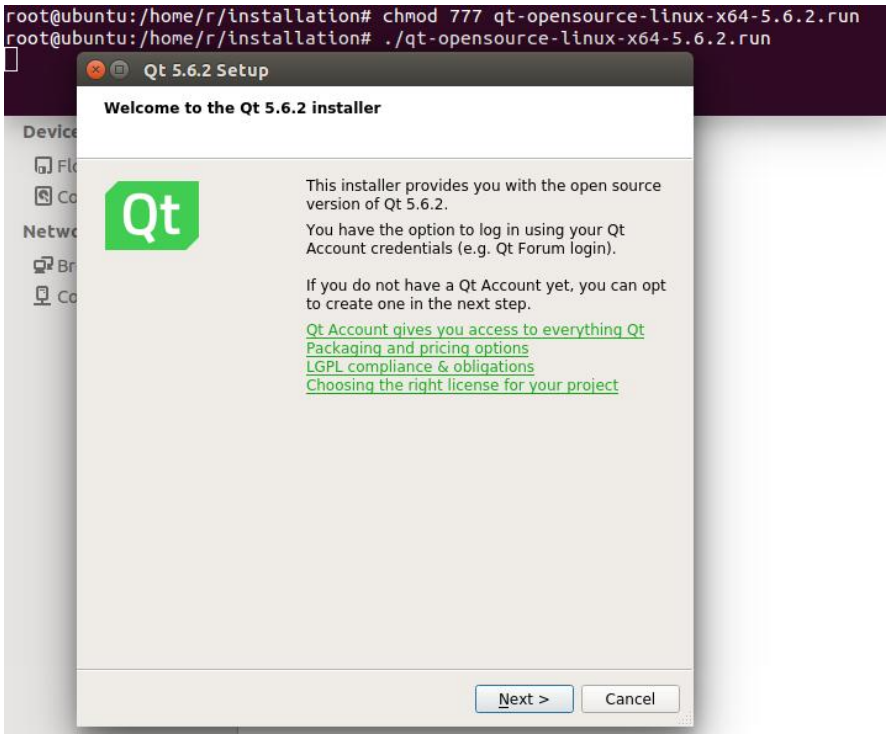


2.安装 Qt5.4.2

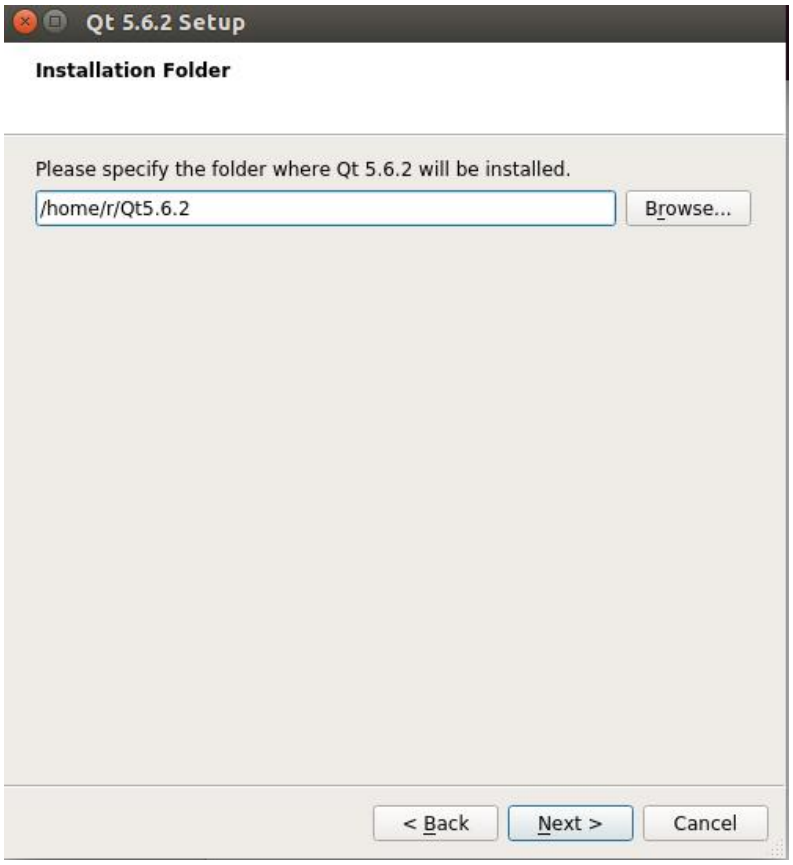
安装资源包为 qt-opensource-linux-x64-5.6.2.run。安装步骤参考如下:

(1)控制台输入指令 chmod 777 qt-opensource-linux-x64-5.6.2.run

(2)输入指令./ qt-opensource-linux-x64-5.6.2.run，弹出 Qt5.6.2 安装向导，按向导一直下一步，直至安装完成。请参考下图：



安装时可以修改自己喜欢的路径。



修改 Qt5.6.2 文件夹和 ti-processor-sdk-linux-am335x-evm-03.00.00.04 文件夹的权限,可以看到 Qt5.6.2 文件夹没有锁头：

chmod -R 777 Qt5.6.2/
chmod -R 777 ti-processor-sdk-linux-am335x-evm-03.00.00.04/

```
root@ubuntu:/home/r/installation# cd ..
root@ubuntu:/home/r# chmod -R 777 Qt5.6.2/
root@ubuntu:/home/r# chmod -R 777 ti-processor-sdk-linux-am335x-evm-03.00.00.04/
```

3.设置 QT 开发工具中的 Arm 开发环境

(1) 将 ti-sdk 加入 qt 中

控制台输入指令

source ti-processor-sdk-linux-am335x-evm-03.00.00.04/linux-devkit/environment-setup (注：此命令即为前述步骤 1 中安装 ti-sdk 后的 envirment-setup 所在的路径)。成功后控制台命令行会有深绿色字体[linux-devkit]~>提示已成功启动 ti-sdk。如下图:

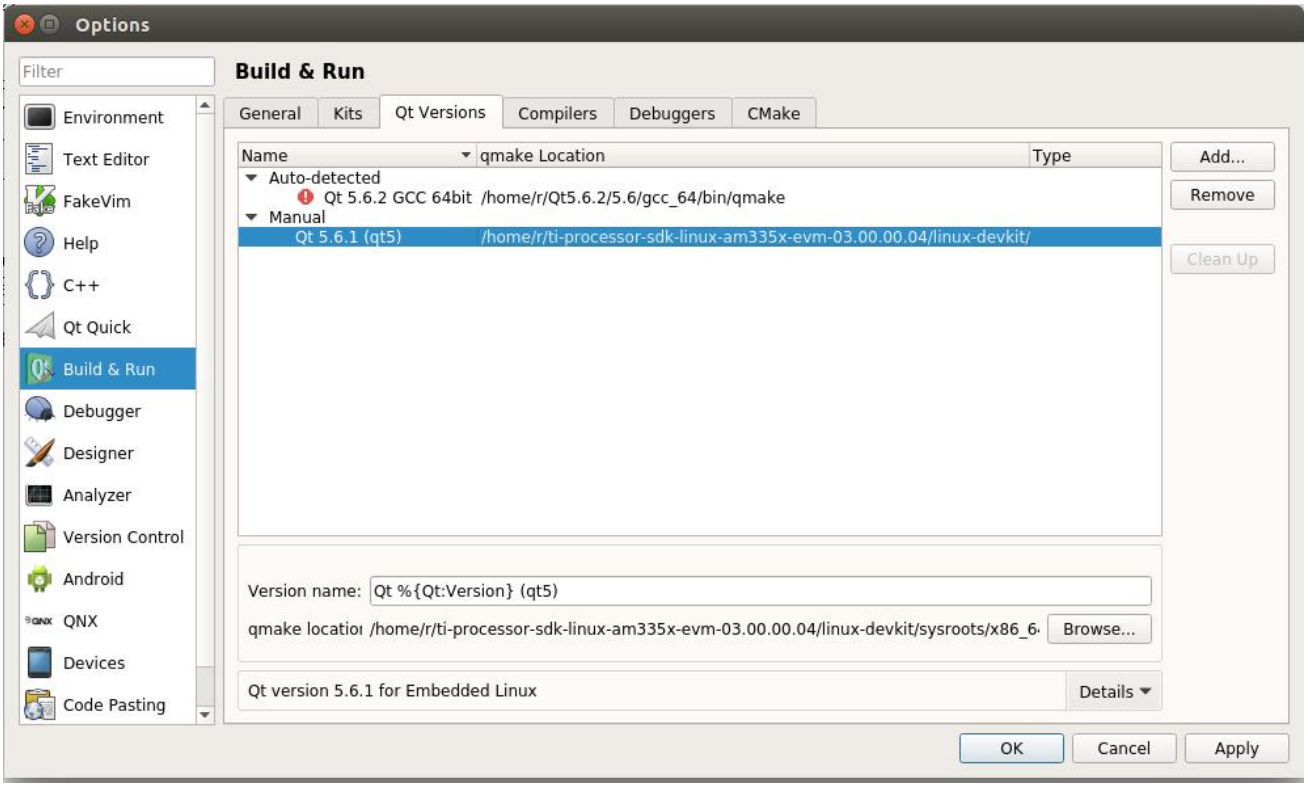
```
root@ubuntu:/home/r# source ti-processor-sdk-linux-am335x-evm-03.00.00.04/linux-
devkit/environment-setup
[linux-devkit]:/home/r>
```

(2) 在[linux-devkit]的环境下启动 qt

输入指令 Qt5.6.2/Tools/QtCreator/bin/qtcreator 启动 qt 开发工具（注此处 Qt5.6.2/Tools/QtCreator/bin/qtcreator 即为上述步骤 2 中安装好 Qt5.4.2 后 qtcreator 工具所在的路径).

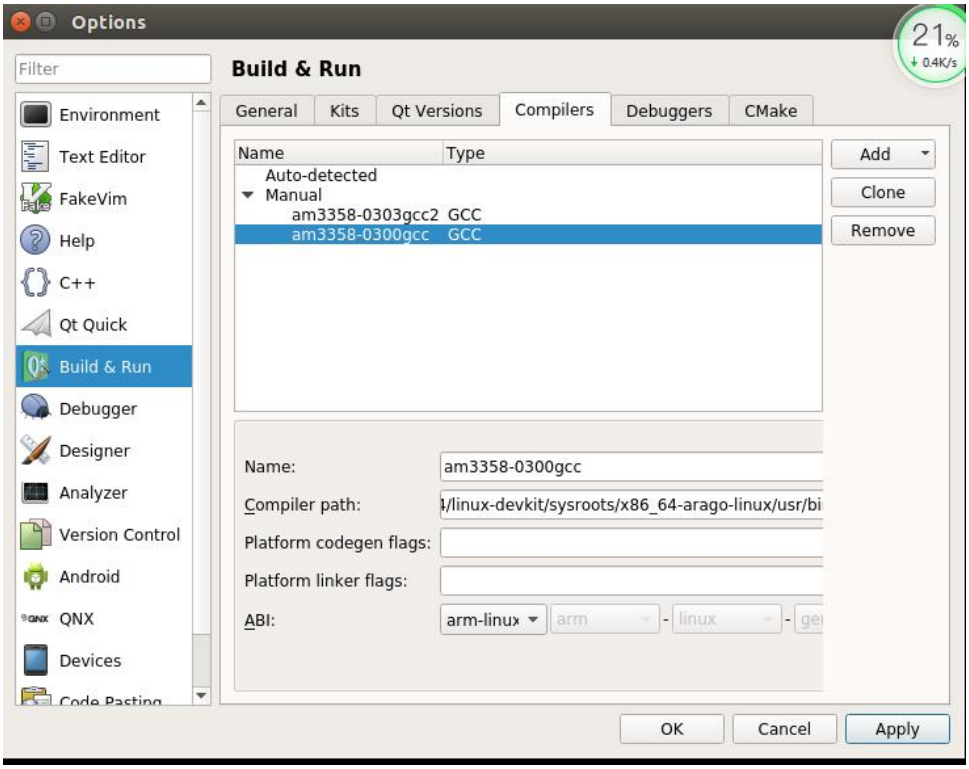
(3) 设置 Qt Version,添加 qmake

启动 QT IDE(集成开发工具)后，点击菜单栏中的 Tools，在弹出的窗体左侧点击 Build&Run，在右侧点击 Qt Version,单击 Add,添加 qmake（路径为 /home/r/ti-processor-sdk-linux-am335x-evm-03.00.00.04/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/qt5/qmake,即之前安装的 ti-sdk 目录下)添加成功后将增加的版本命名为 GCC,如下图：



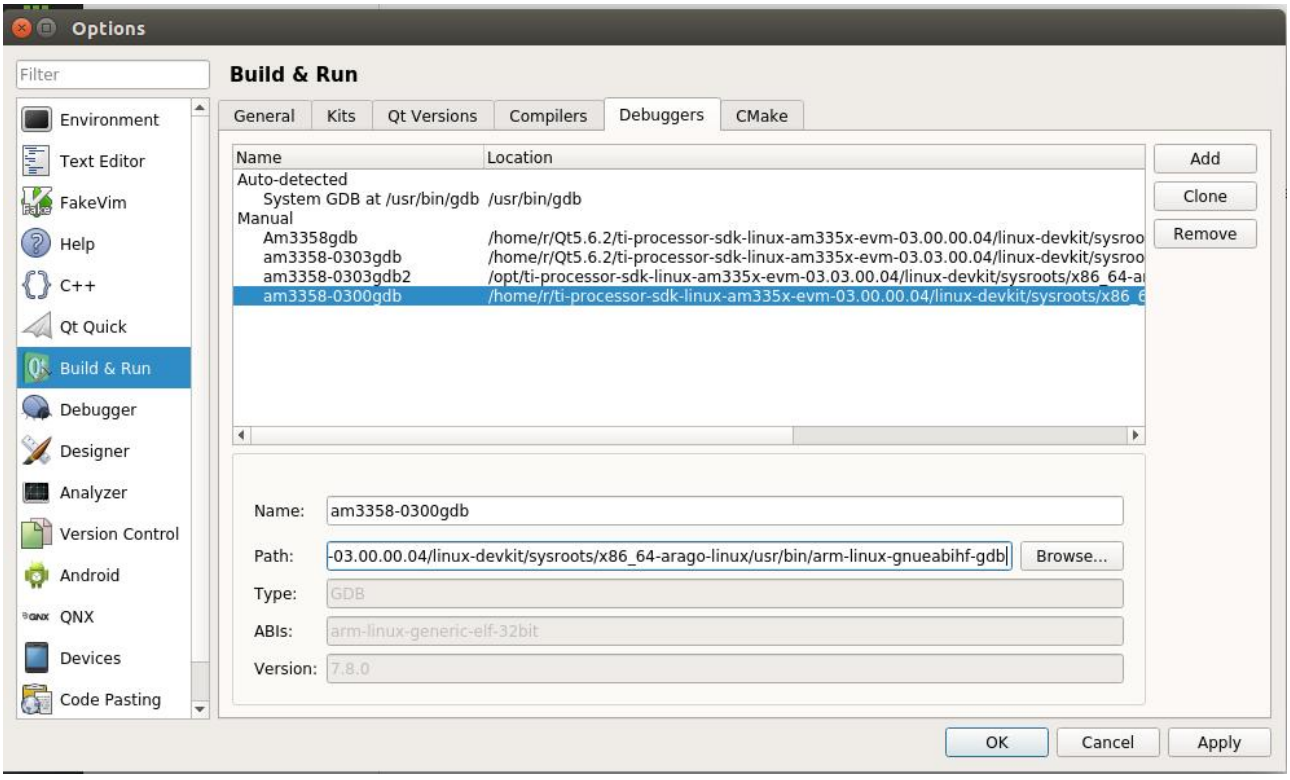
(4) 选择编译器，添加 GCC

点击 complilers, 选择添加 GCC, 编译器路径(/home/r/ti-processor-sdk-linux-am335x-evm-03.00.00.04/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/arm-linux-gnueabihf-gcc),如下图：



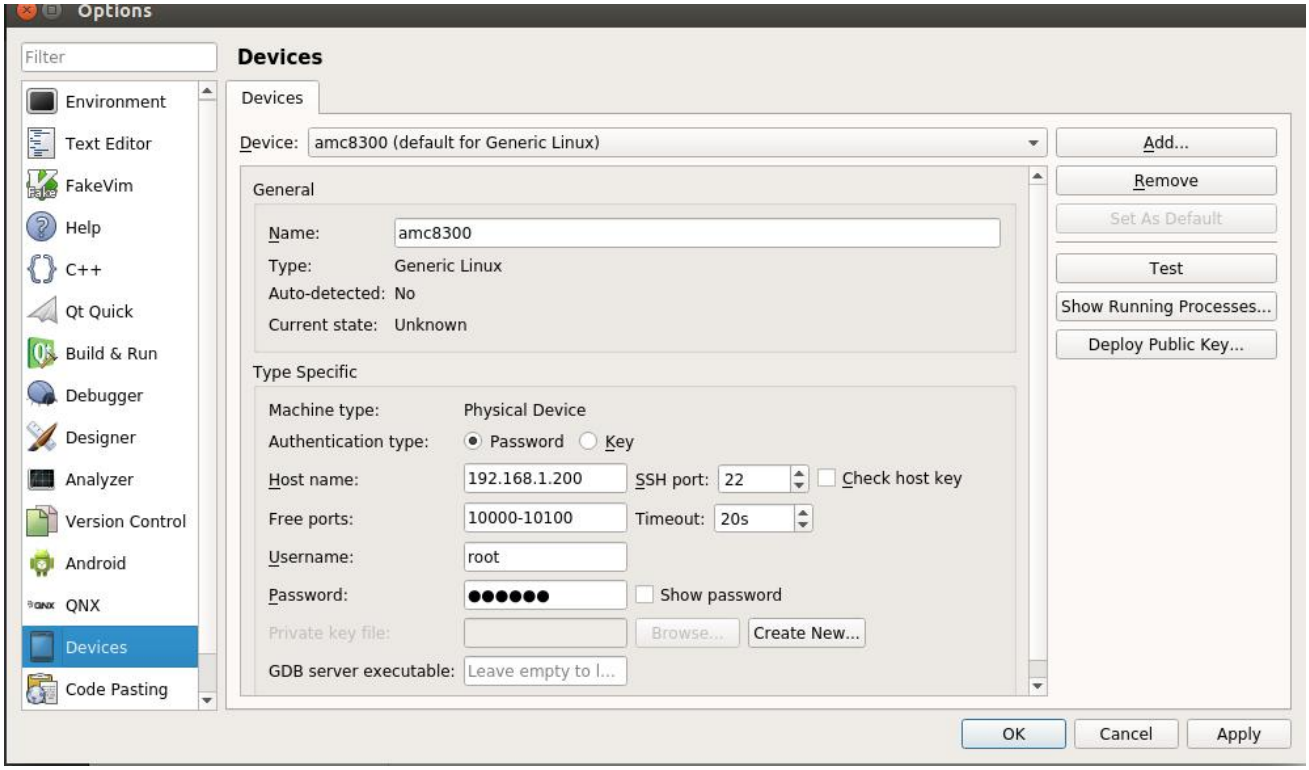
(5) 选择 Debuggers 添加 gdb.

点击 Debuggers，选择添加 gdb，添加路径(/home/r/ti-processor-sdk-linux-am335x-evm-03.00.00.04/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/arm-linux-gnueabihf-gdb)，添加成功后可将添加的 debugger 命名成自定义的名字，例如本示例中叫 Armtest。如下图：



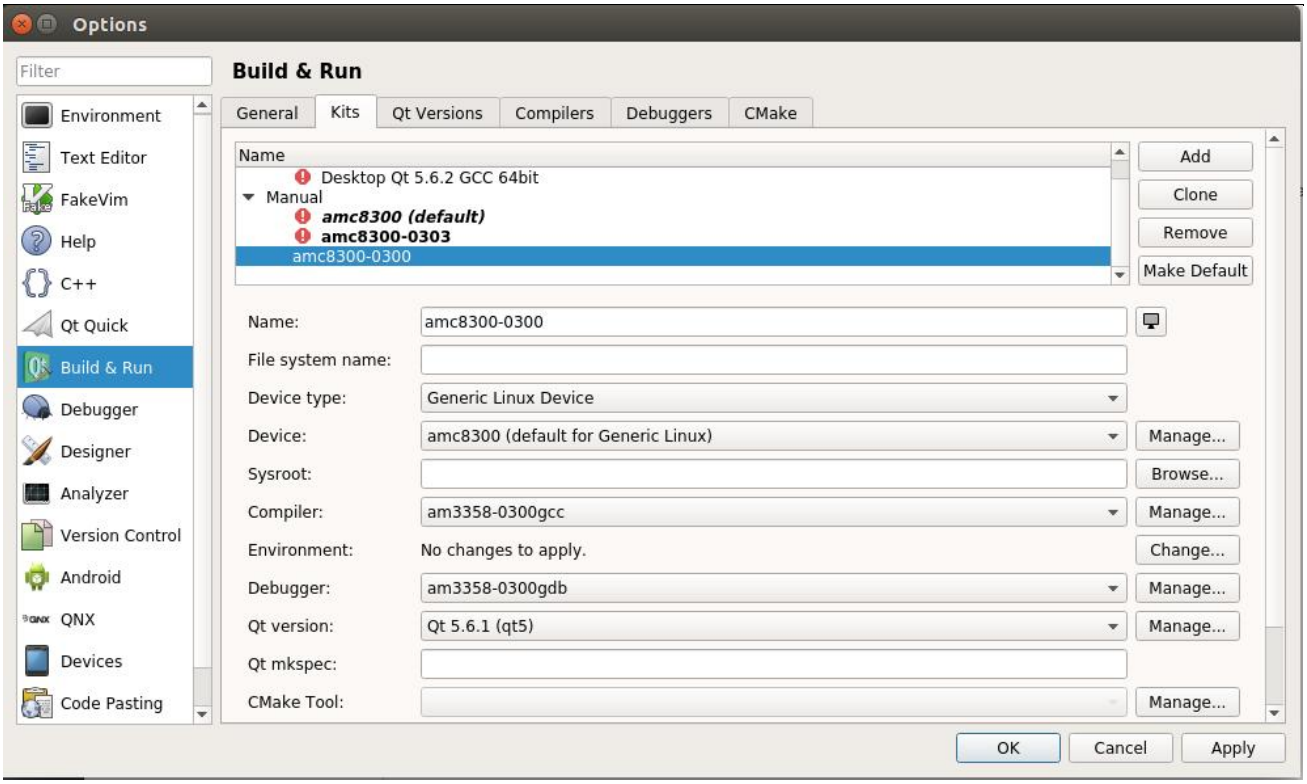
⑹选择 Devices，添加 Generic Linux Device

Tools 下点击左侧选项 Device，单击 ADD，选择 Generic Linux Device，弹出的设备设置向导中，Name 设为 Arm-test2(自定义)，Host Name 设置 192.168.1.200(控制器 IP 地址)，Port:22，username:root，密码：123456。设置成功后结果如下图，可以点击 TEST 进行连接测试是否连接 OK(注，要测试连接，本机 PC 的 IP 地址要设置与 Device 同一个 IP 段，例如 192.168.1.101，PC 机的 IP 设置在网络连接中编辑，不在此具体说明)。



⑺设置 Kits 选项中的 Device、Compiler、Debuggr、Qt Version

在 Tools 下选择 Build&Run，选择 kits,点击 ADD 新增一项命名为 Armtest，将前述过程中所配置的自定义的 Device, Compiler, Debuggr, Version 配置到新增的 kists 选项 Armtest 的相关项中，配置结果抓图如下。



至此，则编程工具的环境配置成功，接下来是新建工程中配置使用此结果。

4.新建 QT 工程中的环境配置。

下面以一最小工程为例，运行在 ARM 控制器上，尚不调用运动控制开发库。

(1)在 ARM 环境下启动 QT，在控制台下分别输入指令

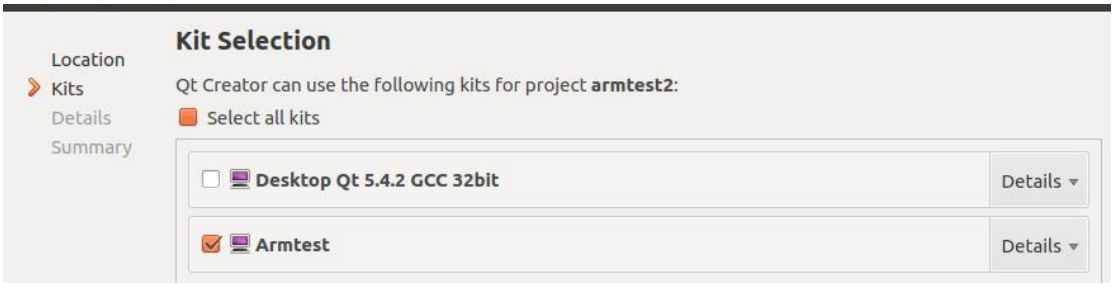
source ti-sdk-am335x-evm-07.00.00.00/linux-devkit/envirment-setup

Qt5.4.2/Tools/QtCreator/bin/qtcreator

启动 QT，新建一个工程，命名 armtest2(自定义)。

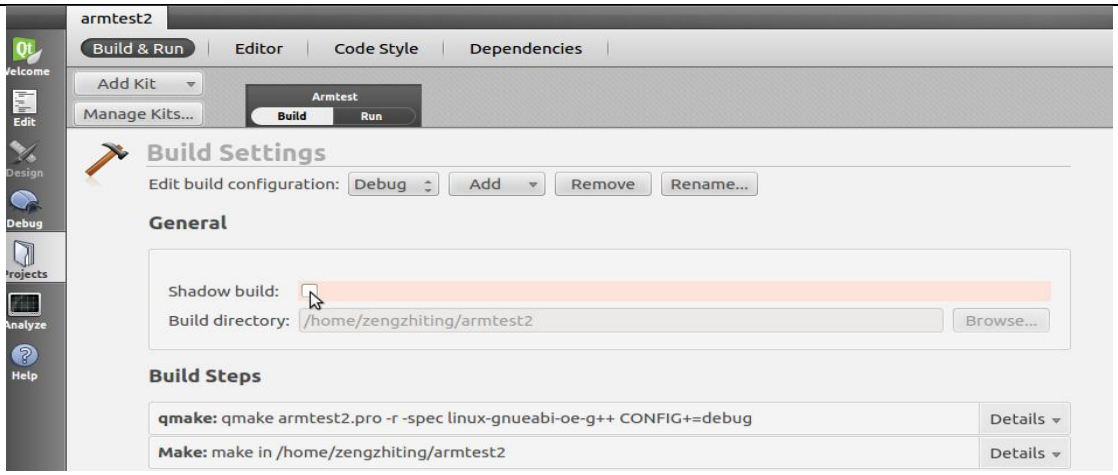
向导第一步选择 Application->Qt Widget Application，

向导第二步弹出的 Kit Selection 中选择前述第 3 步过程中的配置结果 Armtest。



其他使用默认向导配置，生成工程。

(2)工程建好后 ,在 Projects 中把 Shadow build 的钩去掉，在 Run 中添加已设置好的 Device。



(3)在.Pro 文件中添加启动环境代码

D target.path +=/home/root
INSTALLS +=target

至此，则工程 ARM 环境调用配置完成，在 ARM 控制器在线连接的情况下，在工程 armtest2 中按运行按钮，则工程结果运行在 ARM 上，画面在 ARM 控制器的 VGA 所连接的输出终端上（显示器或者示教器）。

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Test1310721
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

Unix
{
    target.path   +=/home/Robot
    INSTALLS +=target
    LIBS += -L$$PWD/Ethercatlibrary/ -lAmcArmMccl -lAmcDrv -lethercat_rtdm -lrtdm -lxenomai -lnative   -lrt -lAmcArmRobot
    INCLUDEPATH += $$PWD/Ethercatlibrary
    DEPENDPATH += $$PWD/Ethercatlibrary
}
```

至此，工程的 ARM 环境调用配置完成，ARM 控制器在线连接的情况下，点击 armtest2 运行按钮，则工程结果运行在 ARM 上，界面将显示在 ARM 控制器 VGA 接口所连接的输出终端上（显示器或者示教器）。

1.2.3 函数库在 QT 中的使用：

QT 如何添加 ARM 板需要的库文件?

假设 ARM 控制器所需的开发包放在 QtLib 文件夹中，需要的步骤如下：

1.将 QtLib 文件夹拷到您建的工程里。

2.在.pro 中添加以下代码：

```
Unix
{
    target.path     +=/home/Robot
    INSTALLS +=target
    LIBS  +=  -L$$PWD/Ethercatlibrary/  -lAmcArmMccl  -lAmcDrv  -lethercat_rtdm  -lrtdm
-lxenomai -lnative     -lrt -lAmcArmRobot
    INCLUDEPATH += $$PWD/Ethercatlibrary
    DEPENDPATH += $$PWD/Ethercatlibrary
}
```

注意：目录要和您实际的目录一样

3.在工程中需要使用 MCCL 函数库的地方加入头文件包含代码

```
#include"QtLib /include/mccl.h"
#include"QtLib /include/MCCL_Fun.h"
#include"QtLib /include/amcgpio.h"
#include"QtLib /include/kinematics.h"
```

至此，用户就可以在 QT 中调用函数库中的任何函数，开始编写应用程序。

1.3 运动轴定义

AMC 控制器的物理轴数目前可用的有 8 个（Channel0 ~ Channel 7），在后面即将推出的 EtherCAT 版本中，将最多扩展到 32 轴。MCCL 使用 Group 的操作概念，Group 可视为一个独立的运动系统，每个 Group 都可最多包含 X、Y、Z、U、V、W、A、B8 个轴进行同动控制。其中 X-Y-Z 为直角正交坐标系运动插补主轴，U、V、W、A、B 为五个辅助轴。各 Group 之间独立运行互不影响，能很好的帮助用户轻松解决多线程的问题。

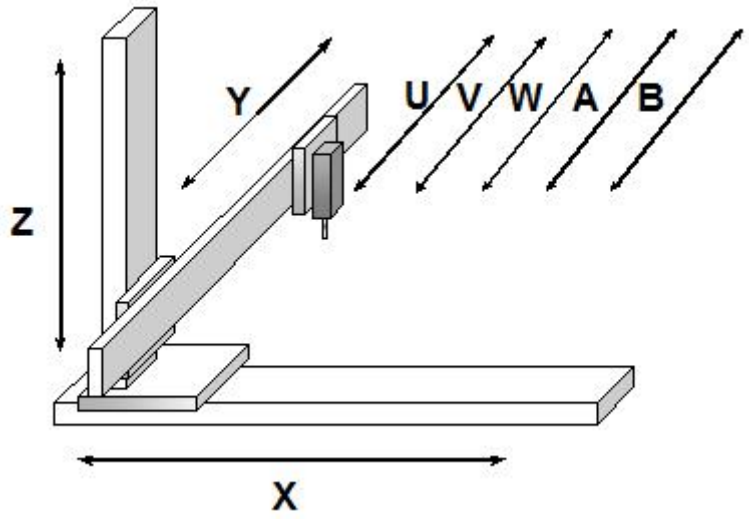


Figure 1.1 三轴直角正交(**X-Y-Z**)，外加五轴辅助轴(**U、V、W、A、**B)

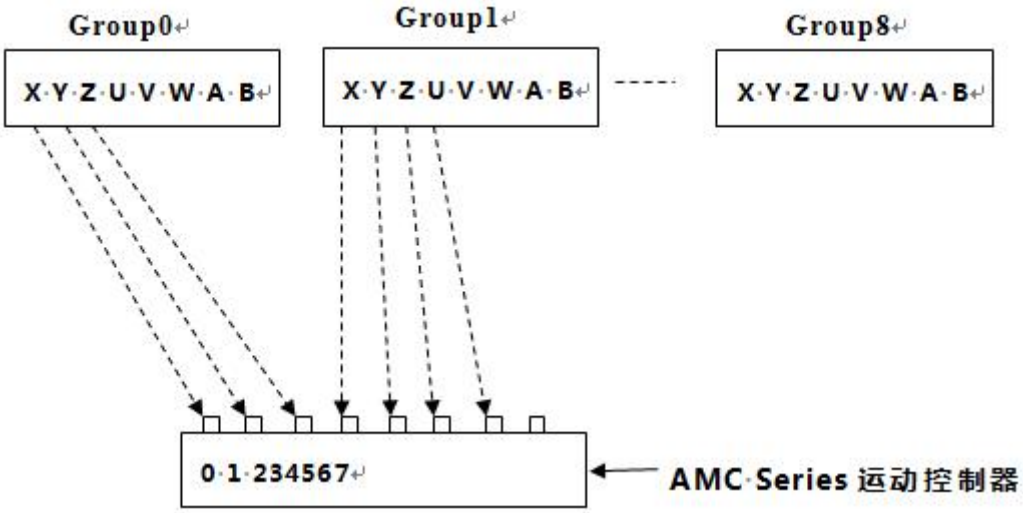


Figure 1.2 Group 设置

详细的 Group 设置将在后面系统初始化章节说明。用户如需单轴运动时，则只对 Group 的某一个轴进行命令操作即可。用户给定的运动命令可使用绝对坐标和相对坐标，无论选用哪种坐标，本函数库都会记录运动位置的绝对坐标（相对于原点）。

1.4 函数库操作特性

调用 MCCL 中的运动函数后系统将相关指令存放在各 Group 专属的运动命令缓冲区（Motion Command Queue）中，并不立即执行。MCCL 通过先入先出（FIFO）的方式，从缓冲区中抓取（Get）运动指令进行解译以及粗插运算。往缓冲区压入新指令动作与运动指令执行可同时进行。

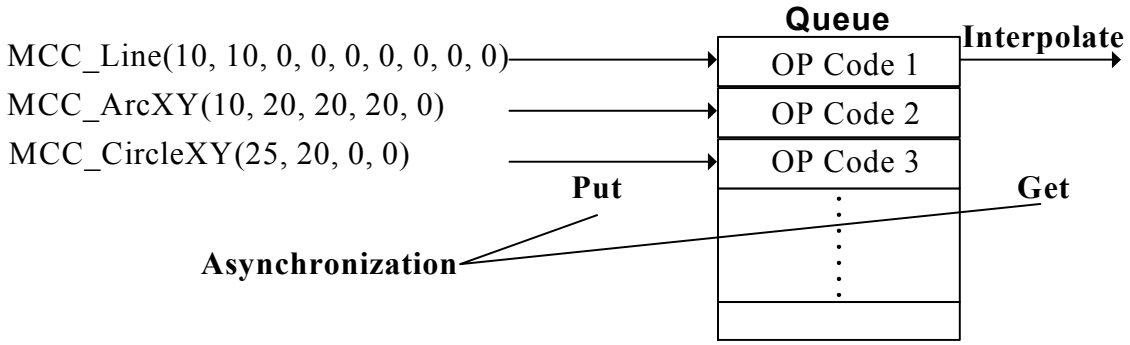


Figure 1.3 运动命令缓冲区

每个 Group 的命令缓冲区最大可储存 10000 条运动指令，用户可通过 MCC_SetCmdQueueSize 改变其大小，而 MCC_GetCmdQueueSize 可读回目前缓冲区大小；*注意：改变缓冲区大小需要在无库存命令时执行。*各 Group 拥有专属的运动命令缓冲区，因此可同时执行属于不同 Group 的运动指令。

会增加运动指令缓冲区库存量的函数包括直线、圆弧、圆、螺旋、点位、Jog 运动、到位确认、平滑运动、运动延迟函数以及自订的 IO 命令。若缓冲区已满，则此笔指令将不被接受，指令返回值为 COMMAND_BUFFER_FULL_ERR.

注意事项：

除以上提到的指令外，其他指令均不置于缓冲区中。指令输入后被被立即执行。例如，要

使 Group0 的 X 轴运动到坐标 10 后输出 Servo-on 信号，再将 X 轴移动到坐标 20 的位置，程序的可能写法如下：

```
MCC_Line(10,0,0,0,0,0,0,0,0);
MCC_SetServoOn(1,0);
MCC_Line(20,0,0,0,0,0,0,0,0);
```

则在 MCC_Line()被放入缓冲区（还未真正执行）时，ServoOn 指令便会被立即执行。因为 MCC_SetServoOn（ ）并不会被放入命令缓冲区中而是直接执行。如果要实现目标功能，用户需进行额外的判断来控制信号的输出动作。如下面的例程：

```
MCC_Line(10,0,0,0,0,0,0,0,0);
While(MCC_GetMotionStatus(0)! = GMS_STOP)
//MCC_GetMotionStatus()  返回值是 GMS_STOP 表示目前全部的运动命令都已执行完成
MCC_SetServoOn(1,0);
MCC_Line(20,0,0,0,0,0,0,0,0);
```

1.5 EtherCAT 函数库使用

AMC EtherCAT 主站最多可控制 32 轴，目前支持的总线伺服有英威腾、清能德创、安川、三洋、汇川、松下、禾川、高创、雷赛等主要品牌，除了拥有该手册介绍的 GPIO 外，还可支持各种 EtherCAT 扩展 IO 模块，为用户提供充足的 IO 资源。运行 EtherCAT 主站需要安装我司提供的专用系统及函数库，除了下面介绍的总线部分函数，其他函数调用方法均跟脉冲模式一致。

EtherCAT 指令列表

指令	说明
EcatSetNumRegOfAxis（ ）	设置 PDO 内容数量
EcatGetNumRegOfAxis（ ）	读取 PDO 内容数量
EcatSetMasterSyncMode（ ）	设置同步时钟跟随模式
EcatGetMasterSyncMode（ ）	读取同步时钟跟随模式
EcatSetAxisNumPerSlave（ ）	设置从站带轴数
EcatSetConfigPDO（ ）	设置是否重新配置从站 PDO
EcatGetConfigPDO（ ）	读取是否重新配置从站 PDO，
EcatSetDCoffset（ ）	设置 SYNC 偏移时间
EcatGetDCoffset（ ）	读取 SYNC 偏移时间,单位 ns
EcatSetDCCycleTime（ ）	设置 SYNC 周期时间
EcatGetDCCycleTime（ ）	读取 SYNC 周期时间,单位 ns
EcatSetDCActiveCode（ ）	設置同步模式
EcatGetDCActiveCode（ ）	读取同步模式
EcatSetChMask（ ）	屏蔽没有接电机的轴
EcatSetInitTime（ ）	设置初始化系统失败超时时间

EcatGetInitTime（ ）	读取初始化系统失败超时时间
MCC_SetECATSlaveNum（ ）	设定连接站点数
GetErrorStatus（ ）	网络连接及伺服报警状态读取
GetErrorCode（ ）	读取伺服报警代码
GetAbsEnc（ ）	读取绝对编码器值
MCC_EcatHome（ ）	启动总线归位
MCC_EcatSetHomeMode（ ）	设置总线伺服的归位模式
MCC_EcatSetHomeZeroSpeed（ ）	设置总线伺服归位低速
MCC_EcatSetHomeSwitchSpeed（ ）	设置总线伺服归位高速
MCC_EcatSetHomeAxis（ ）	设置总线归位轴号
MCC_EcatAbortHome（ ）	停止总线归位
MCC_EcatSetHomeOffset（ ）	设置总线归位的零点偏移值
MCC_EcatGetGoHomeStatus（ ）	获取总线归位状态
MCC_EcatGetLimitSwitchStatus（ ）	获取总线伺服的极限传感器信号状态
MCC_EcatGetHomeSensorStatus（ ）	获取总线伺服的归位传感器信号状态
MCC_EcatCoeSdoDownload（ ）	往总线伺服写 SDO 数据
MCC_EcatCoeSdoUpload（ ）	从总线伺服读取 SDO 数据
EcatClearErrorFlag()	清除总线报警
MCC_SetServoOnWaitTime()	设置 ServoOn 失败等待的时间
ImcSoftFifoClean()	清除 fifo 里的运动储存数据
MCC_ServoOnAll()	ServonOn 所有轴
EcatGetErrorFlag()	读取离线或断开状态

函数详解：

int EcatSetNumRegOfAxis(int nAxis, int nNum)

nAxis:此参数代表第几轴的意思，0 代表 1 轴，1 代表 2 轴，以此类推

nNum：此参数代表 PDO 内容数量，(4 为（摩通、英威腾），6 为（禾川、松下、高创、台达、雷赛），8 为（清能德创））

返回值：等于0时代表读设置成功。

int EcatGetNumRegOfAxis(int nAxis)

nAxis:此参数代表第几轴的意思，0 代表 1 轴，1 代表 2 轴，以此类推

返回值：表示读取到的PDO内容数量。

int EcatSetMasterSyncMode(int nMode)

nMode=0:从站跟随主站时钟

nMode=1:主站跟随从站时钟，通常应当设1

返回值：等于0时代表读设置成功。

int EcatGetMasterSyncMode(void)

返回值：0 表示从站跟随主站时钟，1 表示主站跟随从站时钟

int EcatSetAxisNumPerSlave(int nSlave, int nAxisPerSlave)

nSlave：表示第几个从站,0 代表 1 从站，1 代表 2 从站，以此类推

nAxisPerSlave：表示从站带轴数（1 个从站带几个轴）

返回值：等于0时代表读设置成功。

int EcatSetConfigPDO(int nSlave, char cEnable)

nSlave：表示第几个从站,0 代表 1 从站，1 代表 2 从站，以此类推

cEnable：0,表示不重新配置。1,表示重新配置。通常应当设0。

返回值：等于0时代表读设置成功。

int EcatGetConfigPDO(int nSlave)

nSlave表示第几个从站,0代表1从站，1代表2从站，以此类推

返回值：0代表不重新配置，1代表重新配置

int EcatSetDCoffset(int nSlave, int nOffsetTime_ns)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

nOffsetTime_ns：表示SYNC偏移时间，一般为SYNC周期时间的一半（即同步时间为1ms时，应设（1*1000000）/2=500000，单位ns

返回值：等于0时代表读设置成功。

int EcatGetDCoffset(int nSlave)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

返回值：是SYNC的偏移时间

int EcatSetDCCycleTime(int nSlave, int nCycTime_ns)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

nCycTime_ns：表示SYNC周期时间（即同步时间为1ms时，应设1*1000000=1000000），单位ns

返回值：等于0时代表读设置成功。

int EcatGetDCCycleTime(int nSlave)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

返回值：是SYNC的周期时间,单位ns

int EcatSetDCActiveCode(int nSlave, int nDCActiveCode)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

nDCActiveCode：0表示SM-SYNC模式，0x300表示DC-SYNC模式，通常设为0x300。

返回值：等于0时代表读设置成功。

int EcatGetDCActiveCode(int nSlave)

nSlave：表示第几个从站,0代表1从站，1代表2从站，以此类推

返回值：返回值为0表示SM-SYNC模式，为0x300为DC-SYNC模式。

int GetAbsEnc(int nAxis, DWORD *lowAbs, WORD *highAbs, int64_t *absShift17, int64_t *absShift23)

nAxis：代表读哪个设备轴，按网线连接顺序；

lowAbs：代表单圈位置值；

highAbs：代表多圈圈数;

absShift17：代表17位多圈绝对值总位置值；

absShift23：代表23位多圈绝对值总位置值；

返回值：等于0时代表读取绝对值位置成功。

int GetErrorCode(int nAxis)

nAxis：代表读哪个设备轴。

返回值：伺服最后一次报警的报警代码。

int GetErrorStatus(void)

返回值：从低位开始，1 位代表 1 个轴，报警或断开网线的轴会使当前位置 1；

例如 1 轴报警，则读回来时 0x1;

1、3 轴报警，则读回来为 0x5;

unsigned int EcatSetChMask(int nCh, int nMask)

nCh：代表每个轴,需要屏蔽的轴。

nMask:：是否屏蔽该轴。 1：屏蔽该轴 0：不屏蔽该轴。

返回值：返回值可查看那个轴被屏蔽了，按位表示，那个位的值为 1 则表示哪个轴被屏蔽

int EcatSetInitTime(int nTimeOut)

nTimeOut：设置初始化时多久算超时，单位100ms,如想设10秒钟报警，则设100（100*100ms=10s）,默认不设的话为600。

int EcatSetInitTime(int nTimeOut)

返回值：返回值为读取初始化时多久算超时的时间，单位 100ms，如读 100 则为超过 10 秒钟报警（100*100ms=10s）。

int MCC_SetECATSlaveNum(int nNum, double dfIPOtime)

***nNum*：**站点个数。

dfIPOtime：点个数同步时钟设置，同步该值与初始化系统中设置的 IPO 时间相同。

返回值：0 代表成功，小于 0 不成功，可查看相应错误码。

说明： 该函数必须系在统初始化之前调用。

int MCC_EcatHome( )

返回值：0 代表成功，小于 0 时可查看相应错误码。

说明： 该函数用来启动总线归位。

int MCC_EcatSetHomeMode(int nMode, int nChannel)

nMode：总线归位模式设定。

nChannel：设置归位轴号。

返回值：0 代表成功，小于 0 时可查看相应错误码。

说明： 各 EtherCAT 总线伺服的归位方式均遵照该协议规范设定，具体可参照总线伺服归位说明。

int MCC_EcatSetHomeZeroSpeed(int nSpeed, int nChannel)

nSpeed：总线归位时的低速设置，速度单位由伺服设定，一般默认为 pulse/s。

nChannel：设置归位轴号。

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 该速度通常用于第一步高速高速搜寻之后的低速搜查。

int MCC_EcatSetHomeSwitchSpeed(int nSpeed, int nChannel)

nSpeed：总线归位时的高速设置。

nChannel：设置归位轴号。

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 该速度通常用于第一步高速高速搜寻传感器位置。

int MCC_EcatSetHomeAxis(BYTE cAxisX, BYTE cAxisY, BYTE cAxisZ, BYTE cAxisU, BYTE cAxisV, BYTE cAxisW, BYTE cAxisA, BYTE cAxisB)

cAxisX~B：设置要同时归位的总线伺服轴，1 表示归位，0 表示不参加归位。

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 如需开启某个轴的归位动作，只需将函数相对应的参数设置为 1 即可。

int MCC_EcatAbortGoHome( )

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 该函数用来停止总线归位动作。

int MCC_EcatSetHomeOffset(int nOffset, int nChannel)

***nOffset*：**总线归位后的零点偏移。

nChannel：设置归位轴号。

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 详细可参照总线伺服归位说明。

int MCC_EcatGetGoHomeStatus( )

返回值：32bit，从低位开始，每位代表一个从站轴的归位状态，1 表示正在归位，0 表示归位完成。

说明： 若没有轴在归位运动，则返回值为 0。可按位判断各轴的归位状态。

int MCC_EcatGetLimitSwitchStatus(WORD *pwStatus, WORD wUpDown, WORD wChannel, WORD wCardIndex)

***pwStatus*：**获取的限位传感器状态。

wUpDown: 0 表示负限位，1 表示正限位。

wChannel：需要获取限位传感器状态的从站轴号。

返回值：0 代表成功，非 0 时可查看相应错误码。

说明： 总线系统获取的该极限传感器状态来自从站伺服。在总线系统中，将极限传感器接入总线伺服操作方便且实时性较高。

int MCC_EcatGetHomeSensorStatus(WORD *pwStatus, WORD wChannel, WORD wCardIndex)

pwStatus：获取的归零传感器状态。

wChannel：需要获取归零传感器状态的从站轴号。

返回值：0 代表成功，小于 0 时可查看相应错误码。

说明：总线系统将归位传感器接入从站伺服，通过伺服来进行归位，精度和实时性会比较高。

int MCC_EcatCoeSdoDownload(DWORD dwSlaveID,WORD wObIndex, BYTE byObSubIndex, BYTE* pbyData, size_t dwDataLen)

***dwSlaveID*：**总线伺服 ID 站号。

wObIndex：SDO 数据对象的 ID 号。

byObSubIndex: SDO 数据对象的 Subindex。

pbyData：写入对象数据的 buffer。

dwDataLen：存放对象数据的 buffer 长度

返回值：0 代表成功，小于 0 时可查看相应错误码。

说明： 可根据数据对象地址 ID 对其进行修改设置。ID 列表可参照总线伺服说明书。

int MCC_EcatCoeSdoUpload(DWORD dwSlaveID,WORD wObIndex, BYTE byObSubIndex, BYTE* pbyData, size_t dwDataLen, size_t* pdwOutDataLen)

***dwSlaveID*：**总线伺服 ID 站号。

wObIndex： SDO 数据对象的 ID 号。

byObSubIndex: SDO 数据对象的 Subindex。

pbyData： 存放获取数据的 buffer。

dwDataLen： 存放对象数据的 buffer 长度。

pdwOutDataLen： 获取的实际数据长度。

返回值：0 代表成功，小于 0 时可查看相应错误码。

说明： 可根据数据对象地址 ID 读取任意需要的数据值。ID 列表可参照总线伺服说明书。

int EcatClearErrorFlag(int nAxis)

***nAxis*：**清除哪个轴总线报警，0 代表第一轴。

返回值：0 代表成功，小于 0 时代表有错误。

说明： 清除总线报警。

int MCC_SetServoOnWaitTime(int nWaitTime, int nTryTimes)

***nWaitTime*：**单位 100ms,如想设 10 秒钟退出等待 ServoOn，则设 100（100*100ms=10s）

***nTryTimes*：**单位 100ms,如想设 100 毫秒尝试再 ServoOn，则设 100（100*1ms=100ms）

返回值：0 代表成功，小于 0 时代表有错误。

说明： 设置 ServoOn 失败等待的时间。

int ImcSoftFifoClean(int nAxis)

***nAxis*：**0 代表清除第 1 个轴的 FiFo 运动储存数据

返回值：0 代表成功，小于 0 时代表有错误。

说明： 清除 fifo 里的运动储存数据。

void MCC_CALL MCC_ServoOnAll(void)

说明： ServonOn 所有的轴。

int MCC_CALL EcatGetErrorFlag(void)

返回值：每一個 bit 代表从站是否离线或是断开，bit0 = 1 的話 位置 0 的从站离线或是报错。

说明： 读取从站离线或断开状态。

系统初始化钱总线配置例程：

```
int i=0,l_nServoType,l_ReturnValue,EcatGetNumRegValue=0,EcatGetDCoffsetValue=0,EcatGetDCActiveCodeValue=0,EcatGetConfigPDOValue=0,EcatGetMasterSyncModeValue=0,EcatGetDCCycleTime=0;
        g_nAxisCnt=(ui->comboBox_AxisCnt->currentIndex())+1;
        l_nServoType=ui->comboBox_ServoType->currentIndex();
        INTERPOLATION_TIME =ui->lineEdit_InterTime->text().toInt();
        //(l_nServoType==0 是通用伺服的配置，如英威腾、安川、汇川、松下、禾川、高创、雷赛
        if(l_nServoType==0)
        {
            nAxisPerSlave=1;
            EcatSetMasterSyncMode(1);
            EcatGetMasterSyncModeValue=EcatGetMasterSyncMode();
            qDebug("MasterSyncModeValue=%d  \n",EcatGetMasterSyncModeValue);
            g_nSlaveCnt=g_nAxisCnt;

            for(i=0;i<g_nSlaveCnt;i++)
            {
                EcatSetDCCycleTime(i,INTERPOLATION_TIME*1000000);
                EcatSetDCActiveCode(i,0x300);
                EcatSetDCoffset(i,(INTERPOLATION_TIME*500000));
                EcatSetConfigPDO(i,0);
                EcatSetAxisNumPerSlave(i,nAxisPerSlave);
                EcatGetDCActiveCodeValue=EcatGetDCActiveCode(i);
                EcatGetDCoffsetValue=EcatGetDCoffset(i);
                EcatGetConfigPDOValue=EcatGetConfigPDO(i);
                qDebug("DCActiveCode(%d)=0X%x          tDCoffset(%d)=%d          ConfigPDO(%d)=%d \n",i,EcatGetDCActiveCodeValue,i,EcatGetDCoffsetValue,i,EcatGetConfigPDOValue);
                qDebug("g_nSlaveCnt=%d ,nAxisPerSlave=%d \n",i,nAxisPerSlave);
            }
            for(i=0;i<g_nAxisCnt;i++)
            {
               EcatSetNumRegOfAxis(i,4);
               EcatGetNumRegValue=EcatGetNumRegOfAxis(i);
               qDebug("NumRegValue(%d)=%d \n",i,EcatGetNumRegValue);
            }
        }
        //(l_nServoType==1 是清能德创伺服的配置
        if(l_nServoType==1)
        {
            nAxisPerSlave=1;
            EcatSetMasterSyncMode(1);
```

```
            EcatGetMasterSyncModeValue=EcatGetMasterSyncMode();
            qDebug("MasterSyncModeValue=%d   \n",EcatGetMasterSyncModeValue);
            g_nSlaveCnt=g_nAxisCnt;

            for(i=0;i<g_nSlaveCnt;i++)
            {
                EcatSetDCCycleTime(i,INTERPOLATION_TIME*1000000);
                EcatSetDCActiveCode(i,0x300);
                EcatSetDCoffset(i,(INTERPOLATION_TIME*500000));
                EcatSetConfigPDO(i,0);
                EcatSetAxisNumPerSlave(i,nAxisPerSlave);
                EcatGetDCActiveCodeValue=EcatGetDCActiveCode(i);
                EcatGetDCoffsetValue=EcatGetDCoffset(i);
                EcatGetConfigPDOValue=EcatGetConfigPDO(i);
                qDebug("DCActiveCode(%d)=0X%x          tDCoffset(%d)=%d          ConfigPDO(%d)=%d
\n",i,EcatGetDCActiveCodeValue,i,EcatGetDCoffsetValue,i,EcatGetConfigPDOValue);
                qDebug("g_nSlaveCnt=%d ,nAxisPerSlave=%d \n",i,nAxisPerSlave);
            }
            for(i=0;i<g_nAxisCnt;i++)
            {
               EcatSetNumRegOfAxis(i,8);
               EcatGetNumRegValue=EcatGetNumRegOfAxis(i);
               qDebug("NumRegValue(%d)=%d \n",i,EcatGetNumRegValue);
            }
        }
        //(l_nServoType==2 是星士达伺服的配置
        if(l_nServoType==2)
        {
            nAxisPerSlave=4;
            EcatSetMasterSyncMode(1);
            EcatGetMasterSyncModeValue=EcatGetMasterSyncMode();
            qDebug("MasterSyncModeValue=%d   \n",EcatGetMasterSyncModeValue);
            if((g_nAxisCnt>=0)&&(g_nAxisCnt<=4))
            {
                g_nSlaveCnt=1;
            }
            else
            {
                g_nSlaveCnt=2;
            }

            for(i=0;i<g_nSlaveCnt;i++)
            {
                EcatSetDCCycleTime(i,INTERPOLATION_TIME*1000000);
                EcatSetDCActiveCode(i,0x300);
                EcatSetDCoffset(i,(INTERPOLATION_TIME*500000));
                EcatSetConfigPDO(i,0);
                EcatSetAxisNumPerSlave(i,nAxisPerSlave);
                EcatGetDCActiveCodeValue=EcatGetDCActiveCode(i);
```

```
            EcatGetDCoffsetValue=EcatGetDCoffset(i);
            EcatGetConfigPDOValue=EcatGetConfigPDO(i);
            qDebug("DCActiveCode(%d)=0X%x      tDCoffset(%d)=%d      EcatGetDCCycleTime(%d)=%d
ConfigPDO(%d)=%d
\n",i,EcatGetDCActiveCodeValue,i,EcatGetDCoffsetValue,i,EcatGetDCCycleTime,i,EcatGetConfigPDOValue);
        }
        for(i=0;i<g_nAxisCnt;i++)
        {
            EcatSetNumRegOfAxis(i,4);
            EcatGetNumRegValue=EcatGetNumRegOfAxis(i);
            qDebug("NumRegValue(%d)=%d \n",i,EcatGetNumRegValue);
        }
        for(i=0;i<g_nSlaveCnt;i++)
        {
            EcatSetAxisNumPerSlave(i,nAxisPerSlave);
            qDebug("g_nSlaveCnt(%d)=%d ,nAxisPerSlave=%d \n",i,i,nAxisPerSlave);
        }
    }
    //(l_nServoType==3 是摩通伺服的配置
    if(l_nServoType==3)
    {
        nAxisPerSlave=2;
        EcatSetMasterSyncMode(1);
        EcatGetMasterSyncModeValue=EcatGetMasterSyncMode();
        qDebug("MasterSyncModeValue=%d   \n",EcatGetMasterSyncModeValue);
        g_nSlaveCnt=g_nAxisCnt/2;
        for(i=0;i<g_nSlaveCnt;i++)
        {
            EcatSetDCCycleTime(i,INTERPOLATION_TIME*1000000);
            EcatSetDCActiveCode(i,0x300);
            EcatSetDCoffset(i,(INTERPOLATION_TIME*500000));
            EcatSetConfigPDO(i,0);
            EcatSetAxisNumPerSlave(i,nAxisPerSlave);
            EcatGetDCActiveCodeValue=EcatGetDCActiveCode(i);
            EcatGetDCoffsetValue=EcatGetDCoffset(i);
            EcatGetConfigPDOValue=EcatGetConfigPDO(i);
            qDebug("DCActiveCode(%d)=0X%x      tDCoffset(%d)=%d      EcatGetDCCycleTime(%d)=%d
ConfigPDO(%d)=%d
\n",i,EcatGetDCActiveCodeValue,i,EcatGetDCoffsetValue,i,EcatGetDCCycleTime,i,EcatGetConfigPDOValue);
            qDebug("g_nSlaveCnt=%d ,nAxisPerSlave=%d \n",i,nAxisPerSlave);
        }
        for(i=0;i<g_nAxisCnt;i++)
        {
            EcatSetNumRegOfAxis(i,4);
            EcatGetNumRegValue=EcatGetNumRegOfAxis(i);
            qDebug("NumRegValue(%d)=%d \n",i,EcatGetNumRegValue);
        }

    }
```

1.6 Ethercat IO 相关函数使用

指令	说明
EcatIOGetVersion()	读取 EIO 库版本号
EcatIOInit()	EIO 初始化
EcatIOSetOutPut()	设置 Output 输出 IO
EcatIOGetInPut()	设置 Input 输入 IO
EcatIOGetOutPut()	读取 Output 输出值
EcatIOGetSlacveID()	读取从站 IO 的 ID 号
EcatSetPDOS	设置需要配置 PDO 的轴和初始化
EcatGet6077()	PDO 读取 0x6077
EcatSet60e0()	设置 PDO60e0

int EcatIOGetVersion ()

说明：读取EIO库版本号。

返回值：EIO 库版本号。

int EcatIOInit(int nSlavebuf,int nCycTime_nsbuf,int nOffsetTime_nsbuf,int IoNum);

说明：Ethercat IO 初始化。

***nSlavebuf*：**总线 IO 从站号，从 0 开始，第 4 个从站就应该填 3，当填 65535 的时候代表只调用 PDO 映射功能，不读 EIO。

***nCycTime_nsbuf*：**表示SYNC周期时间(即同步时间为1ms时 ,应设1*1000000=1000000)，单位ns。

nOffsetTime_nsbuf: 表示SYNC偏移时间，一般为SYNC周期时间的一半（即同步时间为1ms时，应设（1*1000000）/2=500000，单位ns。

***IoNum*：**有多少张 EIO 板，

返回值：0 代表成功，小于 0 时代表有错误。

EcatIOSetOutPut(int IoNum, int dwValue)

说明：EIO 的 Output 输出。

***IoNum*：**EIO 板卡号，从 0 开始，第 3 张为 2。

***dwValue*：**输出值总共16位，每一位代表1个输出，例如0x33就是第1、2、5、6个输出有效。

返回值：0 代表成功，小于 0 时代表有错误。

EcatIOGetInPut(int IoNum, int *pdwValue)

说明：读取 EIO 的 Input 输入。

IoNum：EIO 板卡号，从 0 开始，第 3 张为 2。

dwValue：输入值总共有16位，每一位代表1个输入，例如0xfff3就是第3和4两个输入脚有输入。

返回值：0代表成功，小于0时代表有错误。

EcatIOGetOutPut(int IoNum, int *pdwValue)

说明：读取 EIO 的 OutPut 输出状态。

IoNum：EIO 板卡号，从 0 开始，第 3 张为 2。

dwValue：*读取*输出值总共16位，每一位代表1个输出，例如0x33就是第1、2、5、6个输出有效。

返回值：0代表成功，小于0时代表有错误。

EcatIOGetSlacveID()

说明：读取从站 ID 号。

返回值：返回从站ID号。

int EcatSetPDOS(int nAxisBegin,int nAxisEnd,int nAxisPerSlave,int ntype)

说明：PDOS 设置初始化。

nAxisBegin：PDOS 开始的从站号，0 为第 1 从站，1 为第 2 从站以此类推。

nAxisEnd：PDOS结束的从站号，0为第1从站，1为第2从站以此类推。

nAxisPerSlave：从站是 1 拖几的。

ntype：PDOS的类型，每个客户需求不一样，类型也不一样。

返回值：0代表成功，小于0时代表有错误。

int EcatGet6077(int SlvIdx, short *pdata)

说明：读取 PDO6077，其他的读取 PDO 寄存器参考此函数说明。

SlvIdx：从站号，从 0 开始。

pdata：*读取0x6077中的数据*。

返回值：0代表成功，小于0代表有错误。

int EcatSet60e0(int SlvIdx, short data)

说明：设置 PDO60e0，其他的设置 PDO 寄存器参考此函数说明。

SlvIdx：从站号，从 0 开始。

pdata：*设置0x60e0中的数据*。

返回值：0代表成功，小于0代表有错误。

1.7 GPIO 相关函数使用

指令	说明
AMC_EXT_Init（ ）	初始化 GPIO、DA、PWM 和 MPG 等
AMC_EXT_Close（ ）	关闭 GPIO、DA、PWM 和 MPG 等
AMC_EXT_GpioOpen（ ）	开启 GPIO 输入输出功能
AMC_EXT_GpioClose（ ）	关闭 GPIO 输入输出功能
AMC_EXT_GpioGetOutput（ ）	读取 GPIO 输出值
AMC_EXT_GpioGetInput（ ）	读取 GPIO 输入值
AMC_EXT_GpioSetOutput（ ）	设置 GPIO 输出值
AMC_EXT_DaOpen（ ）	开启 DA 模拟量输出功能
AMC_EXT_DaClose（ ）	关闭 DA 模拟量输出功能
AMC_EXT_DaSetValue（ ）	设置 DA 输出值
AMC_EXT_DaStartOuput（ ）	模拟量输出功能开始
AMC_EXT_DaStopOuput（ ）	模拟量输出功能关闭
AMC_EXT_MpgOpen（ ）	手摇轮功能开启
AMC_EXT_MpgClose（ ）	手摇轮功能关闭
AMC_EXT_MpgGet（ ）	读取手摇轮的值
GPIOGetVersion()	读取 GPIO 版本号

int AMC_EXT_Init()

初始化GPIO、DA、PWM和MPG等外部设备。

返回值：0 表示初始化 GPIO、DA、PWM 和 MPG 等外部设备成功调用成功

int AMC_EXT_Close()

关闭GPIO、DA、PWM和MPG等外部设备。

返回值：0 表示关闭 GPIO、DA、PWM 和 MPG 等外部设备成功调用成功

int AMC_EXT_GPIOOpen()

开启GPIO输入输出功能。

返回值：0 表示开启 GPIO 输入输出功能调用成功

int AMC_EXT_GPIOClose()

关闭GPIO输入输出功能。

返回值：0 表示关闭 GPIO 输入输出功能调用成功

int AMC_EXT_GpioGetOutput(int nCh)

读取 GPIO 的输出的值

***nCh*：**0 代表读取第 1 路输出，1 代表读取第 2 路输出，共 16 路。

返回值：0 代表 GPIO 无输出。

1 代表 GPIO 有输出。

int AMC_EXT_GpioGetInput(int nCh)

读取 GPIO 的输入的值

***nCh*：**0 代表读取第 1 路输入，1 代表读取第 2 路输入以此类推，共 16 路。

返回值：0 代表 GPIO 无输入。

1 代表 GPIO 有输入。

int AMC_EXT_GpioSetOutput(int nCh, int nValue)

设置 GPIO 的输出值

***nCh*：**0 代表设置第 1 路输出，1 代表设置第 2 路输出以此类推。

nValue：0 表示无输出，1 表示有输出。

返回值：0 代表 GPIO 输出函数调用成功。

int AMC_EXT_DaOpen()

开启DA模拟量输出功能。

返回值：0 表示开启 DA 模拟量输出调用成功

int AMC_EXT_DaClose()

关闭DA模拟量输出功能。

返回值：0 表示关闭 DA 模拟量输出功能调用成功

int AMC_EXT_DaSetValue(int nCh, double dfPercentage)

设置 DA 模拟量输出值

***nCh*：**0 代表设置第 1 路模拟量，1 代表设置第 2 路模拟量，2 代表设置第 3 路模拟量。

dfPercentage：取值 0 到 1，1 代表 100%，0.02 代表 2%，不可取大于 1 小于 0 的值。

返回值：0 代表模拟量输出成功。

-1 代表取值小于 0。

-2 代表模拟量通道取值小于 0。

-3 代表模拟量通道取值大于 3。

-4 代表模拟量百分比大于 1。

-5 代表模拟量百分比小于 0。

int AMC_EXT_DaStartOuput(int nCh)

读取 GPIO 的输入的值

***nCh*：**0 代表开启第 1 路模拟量输出，1 代表开启第 2 路模拟量输出，2 代表开启第 3 路模拟量输出。

返回值：0 代表模拟量输出开启成功。

-1 代表模拟量输出无值。

-2 代表模拟量输出通道小于 0。

-3 代表模拟量输出通道大于 3。

int AMC_EXT_MpgOpen()

开启MPG手摇轮读取功能。

返回值：0 开启 MPG 读取功能成功

int AMC_EXT_MpgClose()

关闭MPG手摇轮读取功能关闭。

返回值：0 表示关闭 DA 模拟量输出功能调用成功

int AMC_EXT_MpgGet()

读取MPG手摇轮的值。

返回值：位读取到的 MGP 手摇轮的值

int GPIOGetVersion()

说明：读取GPIO库版本号。

返回值：GPIO 库版本号。

GPIO输出与输入例程：

```
int i=0;
AMC_EXT_Init();
AMC_EXT_GpioOpen();
for(i=0;i<16;i++)
{
    AMC_EXT_GpioSetOutput(i,1);
    GPIOOutputChannelValue[i]=AMC_EXT_GpioGetOutput(i);
    GPIOInputChannelValue[i]=AMC_EXT_GpioGetInput(i);
    string[i]=QString("%1").arg(GPIOInputChannelValue[i],2,10);//turn 10 进制
    stringOut[i]=QString("%1").arg(GPIOInputChannelValue[i],2,10);//turn 10 进制
    sleep(500);
}
for(i=0;i<16;i++)
{
    AMC_EXT_GpioSetOutput(i,0);
    GPIOOutputChannelValue[i]=AMC_EXT_GpioGetOutput(i);
    GPIOInputChannelValue[i]=AMC_EXT_GpioGetInput(i);
```

```
    sleep(500);
}
AMC_EXT_GpioClose();
```

DA输出例程：

```
AMC_EXT_Init();
AMC_EXT_DaOpen();
AMC_EXT_DaStartOuput(0);
AMC_EXT_DaStartOuput(1);
AMC_EXT_DaStartOuput(2);
AMC_EXT_DaSetValue(0,0.1);
AMC_EXT_DaSetValue(1,0.1);
AMC_EXT_DaSetValue(2,0.1);
```

MPG读入例程：

```
AMC_EXT_MpgOpen();
l_GetMPGValue=AMC_EXT_MpgGet();
stri=QString("%1").arg(l_GetMPGValue,2,10);
ui->lineEdit_GetMPGValue->setText(stri);
```

第二章 指令返回值及其意义

2.1 指令返回值

运动控制器根据应用程序发送的指令工作，运动控制器指令封装在动态链接库中。运动控制器在接收到主机发送的指令时，将执行结果反馈给系统，指示当前指令是否正确执行。指令返回值的定义如下：

运动控制器指令返回值定义

定义	返回值	说明
NO_ERR	0	函数调用成功
INITIAL_MOTION_ERR	-1	系统尚未启动，请再次呼叫 MCC_InitSystem()
COMMAND_BUFFER_FULL_ERR	-2	运动命令缓冲区已满，此时无法接受此笔命令。
COMMAND_NOTACCEPTED_ERR	-3	系统处于忙碌状态，此时无法接受此笔命令。
COMMAND_NOTFINISHED_ERR	-4	执行中的运动命令尚未完成，此时无法接受此笔命令。
PARAMETER_ERR	-5	调用函数时所传入的参数格式错误
GROUP_PARAMETER_ERR	-6	Group 参数设定错误，所指定为无效的 Group
FEED_RATE_ERR	-7	进给速度未设定或设定错误，请重新呼叫 MCC_SetFeedSpeed()函数
VOLTAGE_COMMAND_NOTCALLED_ERR	-9	因系统或此运动轴使用 V Command 操作模式，限制使用此函数
HOME_COMMAND_NOTCALLED_ERR	-10	目前并不在回原点模式下
HOLD_ILLEGAL_ERR	-11	不适当时机发出暂停(Hold)命令
CONTI_ILLEGAL_ERR	-12	不适当时机发出继续(Continue)命令
ABORT_ILLEGAL_ERR	-13	不适当时机使用弃置(Abort)命令
RUN_TIME_ERR	-14	执行时期产生错误，利用呼叫 MCC_GetErrorCode()所获得的错误信息代码可了解错误的内容

ABORT_NOT_FINISH_ERR	-15	命令弃置动作尚未完成
GROUP_RAN_OUT_ERR	-16	已无多余 Group 可使用

注意事项：

建议用户在编写应用程序时，检测每条指令的返回值，以判断指令执行状态，并建立必要的错误处理机制，保证程序安全可靠运行。

第三章 系统初始化设定

在使用运动控制器进行各种操作前，需要建立网络连接并对控制器进行初始化配置，使运动控制器状态和各种工作模式能够满足客户的要求。MCCL 以结构体数据的形式提供给用户进行机构、编码器、Home 参数的详细设定。

3.1 机构、编码器、归位参数、Group 及坐标系设定

3.1.1 指令列表

系统初始化指令列表

指令	说明	页码
MCC_SetMacParam()	设定机构参数	
MCC_GetMacParam()	读取机构参数	
MCC_SetEncoderConfig()	设定编码器参数	
MCC_GetEncoderConfig()	读取编码器参数	
MCC_SetHomeConfig()	设定归位参数	
MCC_GetHomeConfig()	读取归位参数	
MCC_CreateGroup()	新增运动群组	
MCC_CloseGroup()	关闭指定运动群组	
MCC_CloseAllGroups()	关闭所有运动群组	
MCC_UpdateParam()	系统更新参数	
MCC _SetSysMaxSpeed()	设置一般运动的最大进给速度	
MCC _GetSysMaxSpeed()	获取一般运动的最大进给速度	
MCC_SetAbsolute()	使用绝对坐标运动	
MCC_SetIncrease()	使用相对坐标运动	
MCC_SetCmdQueueSize()	设定运动命令命 FIFO 大小	
MCC_GetCmdQueueSize()	读取运动命令命 FIFO 大小	

3.1.2 重点说明：

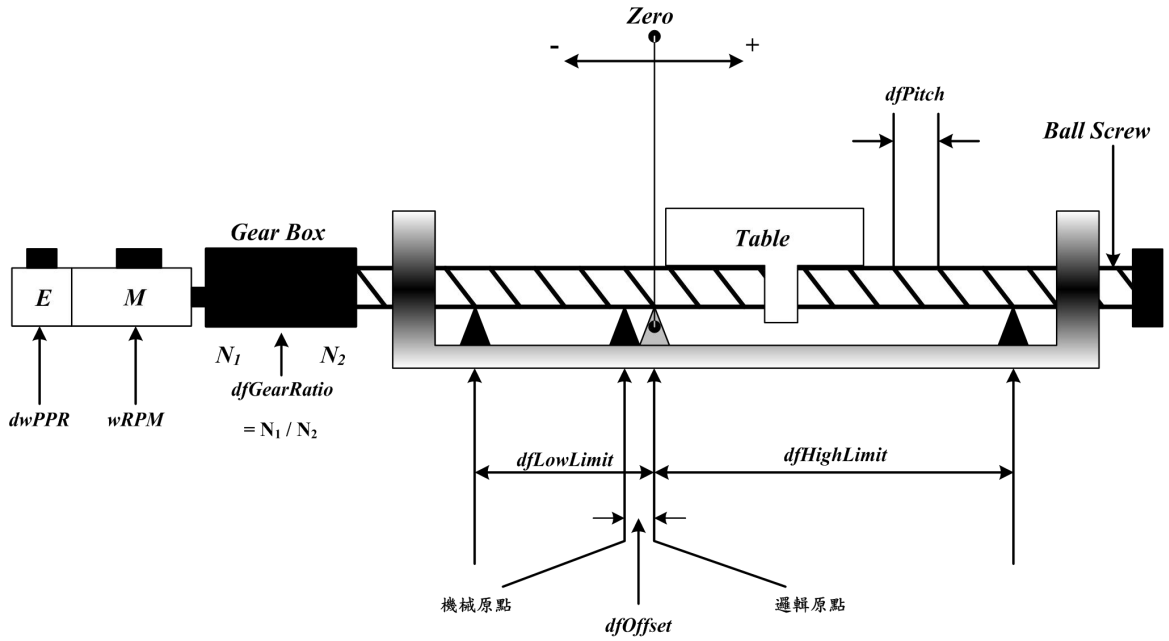


Figure 3.1 机构参数说明

机构参数、编码器参数及归位参数都是针对各物理运动轴来设定。UU(用户定义单位) 对于直线运动，mm 和 inch 可选，默认情况下我们使用 mm。单纯的旋转运动时，用户也可以将 UU 设置为角度单位。

1. 机构参数说明：

```
typedef struct _SYS_MAC_PARAM
{
WORD      wPosToEncoderDir;
WORD      wRPM;
DWORD         dwPPR;
double        dfPitch;
double        dfGearRatio;
double        dfHighLimit;
double        dfLowLimit;
double        dfHighLimitOffset;
double        dfLowLimitOffset;
WORD          wPulseMode;
WORD          wPulseWidth;
WORD          wCommandMode;
WORD      wPaddle;
WORD      wOverTravelUpSensorMode;
WORD      wOverTravelDownSensorMode;
} SYS_MAC_PARAM;
```

参数	说明
wPosToEncoderDir	当编码器反馈值方向与规定的机械运动方向不一致时，可通过设置该参数为“1”来更改方向，而无需更改马达接线。默认值为 0，不改变 Enc 方向。
dwRPM	马达最大安全转速。各轴点对点运动时，所设定转速将不会超过该设定值。 ➔ *See Also* MCC_SetPtPSpeed()

wPPR	马达每转脉冲数。使用直线马达时，该参数可设置为 1（单位：Pulse）。
dfPitch	丝杆导程。如机构中没有丝杆: 旋转运动时，该值可设为 360deg，则 UU 为 deg 若设置为 1，则 UU 为 turn(转) 直线电机时，该值可设为编码器分辨率，UU:mm
dfGearRatio	齿轮减速比。可以简单定义为丝杆每转一圈，电机所转动的圈数。如果结构没有安装减速装置，该值应设为 1.
dfHighLimit	正向软限位，单位为 UU. ➔ *See Also* MCC_SetOverTravelCheck()
dfLowLimit	负向软限位
dfHighLimitOffset	保留，用户需设置为 0.
dfLowLimitOffset	保留，用户需设置为 0.
wPulseWidth	MCC 中无效，需要更改该结构体??
wCommandMode	控制模式： OCM_PULSE脉冲命令(Pulse Command) OCM_VOLTAGE 速度/转矩命令(Velocity Command)
wPaddle	保留，用户需设置为 0.
wOverTravelUpSensorMode	正限位传感器（Limit Switch +）的接线方式
wOverTravelDownSensorMode	负限位传感器（Limit Switch -）的接线方式 SL_NORMAL_OPEN Active High SL_NORMAL_CLOSE Active Low SL_UNUSED 不检查传感器，未安装时可设置

1.设定 Group 群组参数

在使用 MCCL 前需先建立 Group(运动群组)，Group 可视为一独立的运动系统，此系统中主轴坐标系 XYZ 可直接进行插补运动，辅助轴可进行同动。

MCCL 使用 Group 的操作概念，所提供的运动控制函数大部分是以 Group 为对象来操作。每个 Group 均包含 X、Y、Z、U、V、W、A、B 八个运动轴，但各运动轴并不一定要对应到实际的控制器物理轴输出 Channel。MCCL 目前只支持 MCC Series 单张 8 轴运动控制卡，做多可定义 8 个 Group。各 Group 间相互独立，不影响彼此间的运行。我们来试着建立 Group，如下例：

```
int nGroup0, nGroup1;
MCC_CloseAllGroup();              //先关闭所有的 Group
    nGroup0 = MCC_CreatGroup( 0,    //X 对应到 Channel 0
                              1,    //Y 对应到 Channel 1
                              2,    //Z 对应到 Channel 2
                             -1,    //U 不做实际轴输出对应
                             -1,    //V 不做实际轴输出对应
                             -1,    //W 不做实际轴输出对应
                             -1,    //A 不做实际轴输出对应
                             -1,    //B 不做实际轴输出对应
                              0);   // 默认都设置为 0 号卡
```

MCC_CreatGroup()的返回值为新建 Group 的编号（0~8）。

特别说明：

(1) 各 Group 之间独立运行互不影响；

(2) 各 Group 均包含 X、Y、Z、U、V、W、A、B 八个运动轴，各运动轴可规划是否对应至实际物理 Channel 输出，但 Group 中至少有一个运动轴对应至实际 Channel 输出，且不允许有两个轴对应到同一个物理 Channel 输出。

(3) Group 中的各轴是同步的，每个 Group 之间的轴是不同步的。

(4) 为降低 MCCL 对 CPU 的使用率，所使用的 Group 数量应尽量少。

2. 设定系统最大进给速度

通过 MCC_SetSysMaxSpeed()来设置一般运动的最大进给运动速度，单位：UU/sec，该值需要在 MCC_InitSystem()之前调用。当进给速度设定值超过该值时，进给速度使用该最大值。

特别说明：

进给速度是指一般运动中 X/Y/Z 轴的合成速度，U/V/W/A/B 是辅助轴配合 X/Y/Z 同时启停运动。但若某一命令中没有主轴 X/Y/Z 的移动，则所设定的进给速度为辅助轴中行程最大轴的运行速度，其它辅助轴则配合同时启停。

3.2 启动和结束运动控制函数库

指令列表

指令	说明	页码
MCC_InitSystem()	启动运动控制函数库	15
MCC_CloseSystem()	关闭运动控制函数库	
MCC _ EnableDryRun()	开启运动控制函数库模拟运行模式	
MCC_DisableDryRun()	关闭模拟运行功能	
MCC_CheckDryRun()	检查运动模拟功能是否开启	
MCC _ResetMotion()	重启运动控制函数库	
MCC _UpdateParam()	更新系统设定参数	
MCC _SetMaxPulseSpeed()	设置各轴运动速度的最大值（单位：pulse/interpolation time）	
MCC_GetMaxPulseSpeed()	读取各轴运动速度的最大值(单位 :pulse/IPO*)	
MCC_SetMaxPulseAcc()	设置各轴最大的运动加速度（单位：pulse/IPO^2）	
MCC_GetMaxPulseAcc()	读取各轴最大的运动加速度（单位：pulse/IPO^2）	

*IPO: interpolation time

3.2.1 重点说明:

1 在完成机构参数、编码器参数及 Group 设定之后，才能开始使用 MCCL 运动控制函数.

2. 系统初始化函数 MCC_InitSystem()详细参数如下

```
int MCC_InitSystem( int              nInterpolateTime,
SYS_CARD_CONFIG *psCardConfig,
WORD                wCardNo);
```

nInterPolationTime 是插值时间（如下图所示），单位 ms。可设定范围为 1 ~ 50 ms，建议值为 2ms。较小的插值时间使运动轨迹更加平滑，但会增加 CPU 的工作负荷。

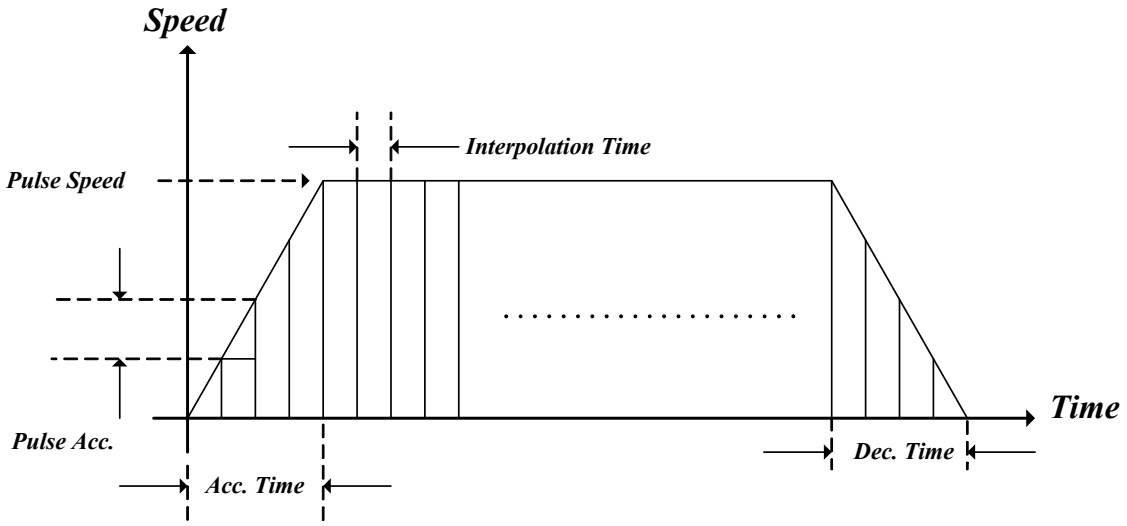


Figure 3.3 轨迹规划设定参数

3. psCardConfig 参数设置

该参数从老版本 MTC 系列控制卡继承而来，用来设定控制卡的类型与使用的 IO 地址（为 ISA 卡设置）。因为 ISA 控制器已经不再使用，所以这里需要设置的只有 CardType 参数:

```
typedef _SYS_CARD_CONFIG
{
  WORD   wCardType ;
  WORD   wCardAddress ;   //可忽略
  WORD   wIRQ_No ;        //可忽略
  WORD   wPaddle ;        //保留
} SYS_CARD_CONFIG ;
```

CardType definition:

0----4 axis ISA card

1----6 axis ISA card

2----4 axis PCI card

3----6 axis PCI card

4----8 axis ARM controller

4 MCC_EnableDryRun()函数可以开启模拟功能，所有的函数基本上都可以在仿真模式下试运行，极大的方便了用户软件问题的查找。

5. MCC_SetMaxPulseSpeed()用来限制各轴在每一插值时间内所能发送的最大 Pulse 量，也即各轴的最大运行速度。设定范围为 1~32767，系统默认设为 32767.

6. MCC_SetMaxPulseAcc()用来限制相邻插值时间之间所发送的 pulse 的变化量，也即各轴最大的加减速度。可设定范围是 1~32767，系统预设值为 32767 Pulses。

7. 在 MCC_InitSystem() 之后，如更改 SetMacParam，EncoderConfig 和 SetHomeConfig，则必须调用 MCC _UpdateParam()来使系统更新设置。该函数作用结果与 MCC_ResetMotion 相似，系统将回复到初始状态。除以上三个参数外，其他参数的设置无需调用该函数。

3.3 例程

例程 3-1 系统初始化

```
#include “initsys.h”
#include "../../newlib/mccl.h"
#include "../../newlib/MCCL_Fun.h

InitSys::InitSys()
{
    //      InitSystem();
}

#define    Card_Index     0
#define    Card_Index     0
#define    Group_Valid    8
#define    NO_ERR         0
```

```
#define     nIPOTime      5     //插值时间设为 5ms
void InitSys::InitSystem()
  {
int nRtn;
int nGroup;
SYS_CARD_CONFIG         stCardConfig;
SYS_MAC_PARAMstMacParam;
SYS_ENCODER_CONFIG    stENCConfig;
SYS_HOME_CONFIG         stHomeConfig;
SYS_CARD_CONFIG         stCardConfig;

nRtn = MCC _CloseSystem();                    //关闭系统
****************机构参数设定****************
stMacParam.wPosToEncoder = 0；//默认不反向
stMacParam.dwPPR              =10000;//每圈脉冲数
stMacParam.wRPM       =3000;   //最大的轴速度
stMacParam.dfPitch = 5.0;//5mm 丝杆导程
stMacParam.dfGearRatio = 1.0;//没有减速装置
stMacParam.dfHighLimit = 500.0;//正软限位单位 mm
stMacParam.dfLowLimit = -500.0;//负软限位单位 mm
stMacParam.wPulseMode = DDA_FMT_CW;//脉冲输出方式
stMacParam.wCommanndMode = OCM_PULSE; //控制方式为脉冲控制
stMacParam..wOverTravelUpSensorMode = SL_UNUSED;
stMacParam..wOverTravelDownSensorMode = SL_UNUSED;//不使用硬限位开关

****************编码器参数设定****************
stENCConfig.wType = ENC_TYPE_AB;//编码器接收方式
stENCConfig.wAInverse       = _NO_;//编码器 A+和 A-是否反相
stENCConfig.wBInverse        =_NO_;//_NO_     0
stENCConfig.wCInverse        =_NO_;//_YES_   1
stENCConfig.wABSwap          =_NO_;
stENCConfig.wInputRate       = 4; //编码器反馈倍率，默认×4

****************归位参数设定****************

stHomeConfig.wMode           = 3;
stHomeConfig.wDirection      = CIR_CW;
stHomeConfig.wSensorMode = SL_NORMAL_OPEN;
stHomeConfig.nIndexCount     =0; //Index 读取的次数
stHomeConfig.dfAccTime = 300;//回零加速时间
stHomeConfig.dfDecTime = 300;
stHomeConfig.dfHighSpeed = 10;//高速段速度
```

```
stHomeConfig.dfLowSpeed      =2;//低速段速度
 //设置八个轴的基本参数
 for(WORD wChannel =0; wChannel <8; wChannel++)
 {
  nRtn = MCC _SetMacParam(&stMacParam, wChannel, Card_Index);//设置 channel 的机构
      参数，Card_Index 为 0，MCC 没有多卡概念
 nRtn =MCC _SetEncoderConfig(&stENCConfig, wChannel, Card_Index);
 nRtn = MCC _SetHomeConfig(&stHomeConfig, wChannel, Card_Index);
 }
nRtn =MCC_CloseAllGroups();//建立 Group 之前先关闭所有 Group
nGroup = MCC_CreatGroup(0,1,2,3,4,5,6,7,Card_Index);// 建立一个 8 轴的 Group, 轴与
                                    Channel 一一对应
if（nGroup<0 ||nGroup> Group_VALID)
{
   qDebug()<<"Creat Group error!";
    Return;
}
nRtn = MCC_SetCmdQueueSize(1000, nGroup) ;//设置运动命令缓冲区大小为 1000 笔，各
        Group 命令 FIFO size 最大为 1000.
stCardConfig.wCardType    = 4;// 0: 4 axis ISA MTC-400/401
                              // 1: 6 axis ISA MTC-600/601
                              // 2: 4 axis PCI MTC-410/411
                              // 3: 6 axis PCI MTC-620/611
                              // 4: 8 axis MCC_800

nRtn = MCC_InitSystem(nIPOTime, stCardConfig, Card_Index);// 控制器初始化
if(nRtn == NO_ERR)
{
  nRtn = MCC_SetSysMaxSpeed(2000);//设置坐标系统的最大进给速度 2000 mm/s
}
else
{
   qDebug()<<"System Initialization Error!";
}


 }
```

这样，我们的控制系统就准备 OK，您可以开始编写自己的应用程序了

第四章　运动状态检测

当用户连接好运动系统（控制器、驱动器和电机）并开始规划运动，可以通过 MCCL 方便的查询当前的系统运动状态，轴错误信息，当前运动位置，运动速度以及指令执行状态参数。

4.1 指令列表

指令	指令说明	页码
MCC_GetMotionStatus()	读取目前的运动状态	
MCC_GetCurCommand()	读取执行中的运动命令相关信息	
MCC_GetCommandCount()	读取运动命令的库存数目	
MCC_ResetCommandIndex()	使运动命令编码值归零	
MCC_GetCurPulseStockCount()	读取目前硬件上的 Pulse 命令库存数目	
MCC_SetMaxPulseStockNum()	设定最大使用的 Pulse 命令库存数目	
MCC_GetMaxPulseStockNum()	读取最大使用的 Pulse 命令库存数目	
MCC_GetErrorCode()	读取现存错误的代码	
MCC_ClearError()	清除现存的错误状态	
MCC_GetCoordType()	读取使用的坐标型态(增量/绝对)	
MCC _GetCurRefPos()	读取各轴位置坐标值(不含补偿)，单位：UU	
MCC _GetCurPos()	读取各轴位置坐标值(含补偿)，单位：UU	
MCC _GetPulsePos()	读取各轴马达位置值(含补偿)，单位：Pulse	
MCC_GetCurFeedSpeed()	读取当前规划的系统进给速度，单位：UU/sec	
MCC_GetFeedSpeed()	读取设定的系统进给速度	
MCC_GetENCValue()	读取编码器的计数值，单位：Pulse	
MCC_GetSpeed()	读取各轴实际规划速度，单位：UU/sec	

4.2 重点说明:

4.2.1 轴状态

调用 MCC_GetMotionStatus()所获得的返回值可以判断系统目前的运动状态，返回值定义如下：

返回值	定义	描述
0	GMS_RUNNING	系统正处于运动状态
1	GMS_STOP	系统已处于停止状态（规划命令完成）
2	GMS_HOLD	系统因调用 MCC_HoldMotion()而处于暂停中
3	GMS_DELAYING	系统调用 MCC_DelayMotion()，下一条运动指令处于延迟执行状态
4	GMS_BLOCKHOLD	
5	GMS_MPGING	系统处于手脉状态

检查系统运行中是否发生错误，我们可以在需要的时候随时调用函数 MCC_GetErrorCode()来进行确认。如系统发生错误（如限位触发、跟随误差过大、超出软限位等），可根据返回值进行排除。若已排除错误，则需要调用 MCC_ClearError（ ）清除系统内的错误记录，否则系统仍无法正常运行。返回值错误码信息请见附录;

4.2.2 命令状态检测

MCCL 提供了两类 FIFO Buffer：①每个 Group 有独立的运动指令缓存；②插值运动命令 FMC(Fine Movement Command)缓存。用户可以通过调用相应函数获取 Buffer 的执行状态.

1. MCC_GetCurCommand()可以获得目前正在执行的运动命令信息，函数原型如下：

```
MCC_GetCurCommand(COMMAND_INFO *pstCurCommand,
                  WORD wGroupIndex)
```

COMMAND_INFO 储存目前执行中的运动命令内容，它被定义为：

```
typedef struct _COMMAND_INFO
{
    int         nType;
    int         nCommandIndex;
    double      dfFeedSpeed;
    double      dfPos[8];
} COMMAND_INFO;
```

参数	说明
nType	运动命令类型： 0--点对点运动 1--直线插补运动 2--顺时针圆弧、圆运动 3--逆时针圆弧、圆运动 4--顺时针螺旋运动 5--逆时针螺旋运动 6--运动延迟

	7--开启平滑运动 8--关闭平滑运动 9--开启到位确认 10--关闭到位确认
nCommandIndex	此运动命令在 Buffer 中的编码
dfFeedSpeed	一般运动进给速度 点位运动运动速度比例 运动延迟目前剩余的延迟时间（ms）
dfPos[8]	目标点位置

另外，用户可以通过 MCC_GetCommandCount()获取目前 Buffer 中剩余运动命令库存量(不包含正在执行的运动命令)。

2. 关于 FMC， MCCL 在规划每一笔运动命令的时候，会根据插值时间将计算的插值命令先放入 FMC 缓存，以解决操作系统实时性的问题。FMC Buffer 库存默认值是 60，用户可根据需求调用函数 MCC_SetMaxPulseStockNum()来设定。若存在 FMC 命令笔数为 0 的情况，必须延长插值时间。当人机操作界面德尔显示出现迟滞现象，也可以考虑延长插值时间。

4.2.3 运动数据获取

MCCL 建立在直角坐标系基础上，用户可以直接使用用户单位（UU）来编写各种运动命令，也可通过 MCC _GetCurPos()等函数来获取坐标值而不需进行任何换算。当然用户也可以通知 MCC_GetPulsePos()来获取以 Pulse 为单位的控制器规划位置。

MCC_GetENCValue()用来读取编码器反馈位置，用户可根据获得的实际位置判断到位情况。

4.3 例程

```
#include <QCoreApplication>
#include <QDebug>
#include "../../newlib/mccl.h"
#include "../../newlib/MCCL_Fun.h"

#define CARD_INDEX                  0
 void InitSystem();
bool _kbhit();

 int main(int argc, char *argv[])
{
 QCoreApplication a(argc, argv);
 int nRtn;
 double    dfCurPosX, dfCurPosY;
 double    dfvelX, dfvelY;
 long      lCurPulseX, lCurPulseY;
 COMMAND_INFO    stCommandInfo;
 int    nCommandCount;
 long       lencposX, lencposY;
 int          g_nGroupIndex = -1;
```

```
InitSystem();//初始化系统
nRtn = MCC_SetServoOn(0, CARD_INDEX); // channel0 上电
nRtn = MCC_SetServoOn(1, CARD_INDEX);// channel1 上电
nRtn = MCC_SetIncrease(g_nGroupIndex);//设置成相对运动
nRtn = MCC_SetAccTime(100, g_nGroupIndex);// 设置运动的加速度时间
nRtn = MCC_SetDecTime(100, g_nGroupIndex);// 设置运动的减速时间
nRtn = MCC_SetFeedSpeed(10, g_nGroupIndex);// 设置运动的进给速度 10mm/s
qDebug()<<"Press any key to start, and ESC key to quit !\n\n";
nRtn = MCC_Line(10, 10, 0, 0, 0, 0, g_nGroupIndex);// 开始 XY 直线插补运动
while(MCC_GetMotionStatus() != GMS_STOP)//检测运动状态
{
nRtn = MCC_GetMotionStatus(g_nGroupIndex);//获取当前运动状态
nRtn = MCC_GetCurFeedSpeed(g_nGroupIndex);//获取当前合成进给速度
nRtn = MCC_GetCurPos(&dfCurPosX, &dfCurPosY,0,0,0,0,0,0,g_nGroupIndex);//获取 XY
                                                    机械坐标值（单位 mm）
nRtn = MCC _GetPulsePos(&lCurPulseX, &lCurPulseY, 0, 0, 0, 0,0,0, g_nGroupIndex);
                                                    //获取 XY 马达位置值（单位 pulse）
nRtn = MCC _GetSpeed(dfvelX, dfvelY,0,0,0,0,0,0, g_nGroupIndex);//获取当前 XY 各自的
                                                    运行速度（mm/s）
nRtn = MCC _GetENCValue(&lencposX, 0, CARD_INDEX );//获取 X 轴编码器位置（单位
                                                    pulse）
nRtn = MCC _GetENCValue(&lencposY, 1, CARD_INDEX);//获取 Y 轴编码器位置（单位
                                                    pulse）
nRtn = MCC _GetCurCommand(&stCommandInfo, g_nGroupIndex);//获取当前运动命令状态
nRtn = MCC _GetCommandCount(&nCommandCount, g_nGroupIndex);//获取当前 FIFO 中
                                                    剩余命令数目
qDebug()<<"Current Pos: x = "<<dfCurPosX<<"Current Pos: y =
"<<dfCurPosY;
qDebug()<<"Encoder Pos: encx ="<<lencposX<<"Encoder Pos: ency
="<<lencposY;
qDebug()<<"Current Spd: velx ="<<dfvelX<<"Current Spd: vely ="<<dfvelY;
qDebug()<<"CommandIndex ="<<stCommandInfo.nCommandIndex;
    }
    nRtn = MCC_CloseSystem();//关闭控制系统
    return a.exec();
```

第五章　运动模式

MCCL 函数库下的运动都是基于 Group 的概念，用户可以在任何需要的时候去创建和关闭 Group。

MCCL 提供 XYZ 直角坐标系下的二/三维直线、圆弧、圆、螺旋插补运动，在 Group 下可直接实现插补及最多 8 轴同动，我们称之为一般运动；而点位运动是非插补运动，Group 中各轴同时启动，根据各自目标位置单独规划运动；JOG 运动可以根据需要设置成三种模式；另外，MCCL 也提供了运动暂停、弃置、平滑以及运动中更改目标速度和位置等进阶功能。

5.1 一般运动（直线、圆、圆弧、螺旋插补）

5.1.1 指令列表

指令列表

函数	说明	页码
MCC_SetAccType()	设定加速形式	
MCC _GetAccType()	读取使用的加速形式	
MCC _SetDecType()	设定减速形式	
MCC _GetDecType()	读取使用的减速形式	
MCC _SetAccTime()	设定加速时间	
MCC _GetAccTime()	读取加速时间	
MCC _SetDecTime()	设定减速时间	
MCC _GetDecTime()	读取减速时间	
MCC _SetFeedSpeed()	设定进给速度	
MCC _Line()	直线插补	
MCC _ArcXYZ()	X-Y-Z 空间任三点圆弧运动	
MCC _ArcXYZ_Aux()	X-Y-Z 空间任三点圆弧运动与辅助轴同动	
MCC _ArcXY()	X-Y 平面圆弧运动	
MCC _ArcYZ()	Y-Z 平面圆弧运动	
MCC _ArcZX()	Z-X 平面圆弧运动	
MCC _ArcXY_Aux ()	X-Y 平面圆弧与辅助轴同动	
MCC _ArcYZ_Aux ()	Y-Z 平面圆弧与辅助轴同动	
MCC _ArcZX_Aux ()	Z-X 平面圆弧与辅助轴同动	
MCC _ArcThetaXY()	X-Y 平面圆弧运动(以旋转角度为参数)	
MCC _ArcThetaYZ()	Y-Z 平面圆弧运动(以旋转角度为参数)	

MCC _ArcThetaZX()	Z-X 平面圆弧运动(以旋转角度为参数)	
MCC _CircleXY()	X-Y 平面全圆运动	
MCC _CircleYZ()	Y-Z 平面全圆运动	
MCC _CircleZX()	Z-X 平面全圆运动	
MCC _CircleXY_Aux ()	X-Y 平面全圆与辅助轴同动	
MCC _CircleYZ_Aux ()	Y-Z 平面全圆与辅助轴同动	
MCC _CircleZX_Aux ()	Z-X 平面全圆与辅助轴同动	
MCC _HelicalXY_Z()	在 X-Y 平面进行圆周运动的螺线运动	
MCC _HelicalYZ_X()	在 Y-Z 平面进行圆周运动的螺线运动	
MCC _HelicalZX_Y()	在 Z-X 平面进行圆周运动的螺线运动	
MCC_HelicalXY_Z_Aux ()	在 X-Y 平面进行圆周运动的螺线运动与辅助轴同动	
MCC_HelicalYZ_X_Aux ()	在 Y-Z 平面进行圆周运动的螺线运动与辅助轴同动	
MCC_HelicalZX_Y_Aux ()	在 Z-X 平面进行圆周运动的螺线运动与辅助轴同动	

5.1.2 重点说明

1. 在开始一般运动前，我们通过 MCC_SetAccTime(),MCC_SetDecTime() 以及 MCC_SetFeedSpeed()先来设置插补运动的加速度和速度。用户可以根据需要将加减速时间设置成不同的值以实现非对称加减速，加减速段可以分别配置成 T 型或 S 型曲线。

2.一般运动的进给速度是指 XYZ 轴的插补速度，辅助轴 U/V/W/A/B 仅配合 XYZ 轴同时开始和结束运动（各自独立做点位运动）。但若该笔命令中没有 XYZ 三轴的位移，则设定的进给速度为辅助轴中行程最长轴的目标速度，其余轴则配合同时启停。

3. 进给速度的设定不能超过 MCC_SetSysMaxSpeed()所设定之值。若超过则使用该最大设定值。

5.1.3 例程

下面的例子我们将简要介绍一下几种运动函数的使用。

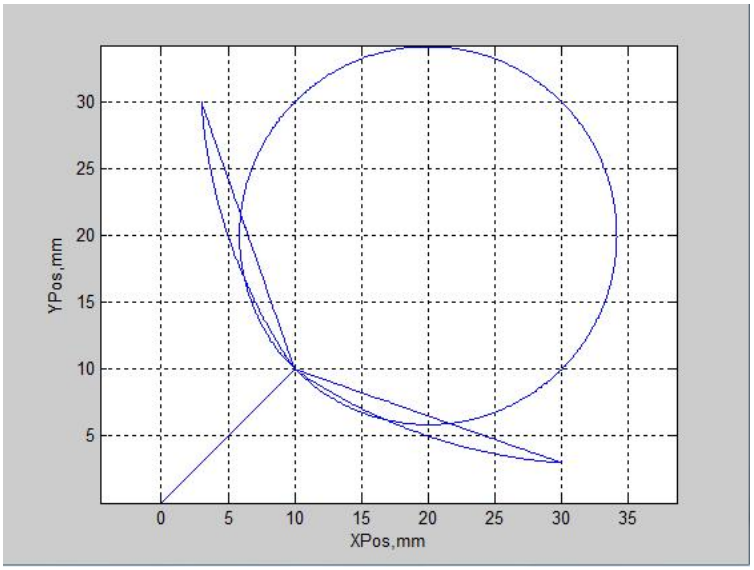


Figure 5.1 插补运动

**********直线/圆/圆弧**********

```
int nRtn = 0;
int nGroup = -1;

InitSystem(); //初始化系统
nRtn = MCC_CloseAllGroups();
nGroup = MCC_CreateGroup(0,1,-1,-1,-1,-1,-1,-1,0); //重新建立 Group
nRtn = MCC_SetAbsolute(nGroup); //设置为绝对坐标
nRtn = MCC_SetServoOn(0,0);    //使能伺服
nRtn = MCC_SetServoOn(1,0);
nRtn = MCC_SetAccType('T',nGroup); //轨迹规划设置为 T 型曲线
nRtn = MCC_SetDecType('T',nGroup);
nRtn = MCC_SetAccTime(100,nGroup); //设置加速度时间
nRtn = MCC_SetDecTime(100,nGroup);

nRtn = MCC_SetFeedSpeed(100,nGroup); //设置进给速度
nRtn = MCC_line(10,10,0,0,0,0,0,0,0,nGroup); //XY 直线插补运动到点（10, 10），单位：mm
nRtn = MCC_CircleXY(20,20,0,nGroup); //XY 平面以点（20, 20）为圆心画圆
nRtn = MCC_ArcXY(5,20,3,30,nGroup); //XY 平面以当前点为起点，经过点(5, 20)走圆弧
                                    插补到达目标点（3, 30）
nRtn = MCC_LineXY(10,10,nGroup); //XY 平面直线插补
nRtn = MCC_CircleXY(20,5,30,3,nGroup); //在 XY 平面，从当前点经点（20，5）走圆弧插
                                    补到目标点（3，30）
nRtn = MCC_LineXY(10,10,nGroup); //在 XY 平面走直线插补回到点（10, 10）, 单位：mm
```

**********螺旋线**********

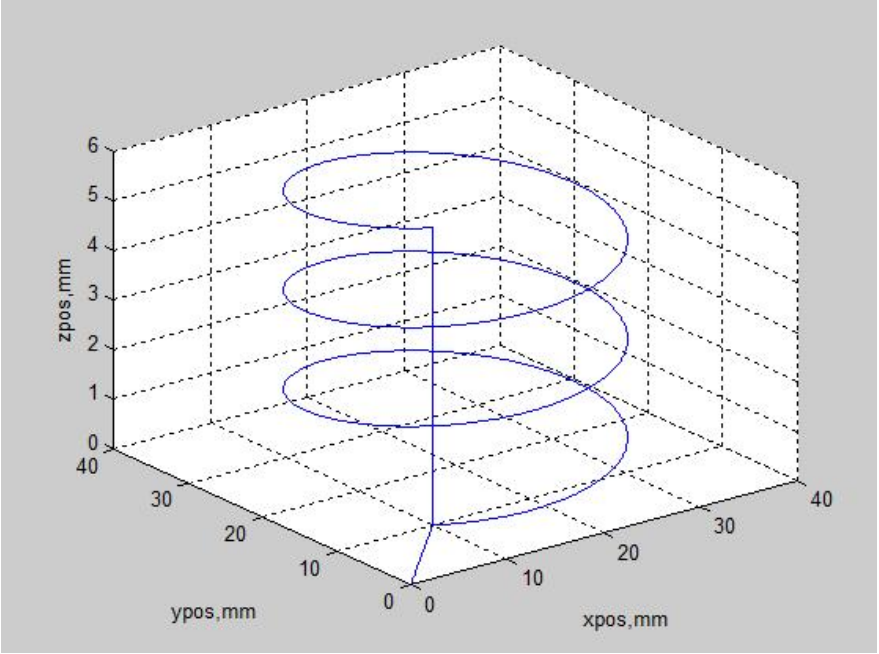


Figure 5.2 螺旋运动

```
int nGroup1 = -1;
nGroup1 = MCC_CreateGroup(2,3,4,-1,-1,-1,-1,-1,0);
nRtn = MCC_SetServoOn(2,0);  //使能伺服轴
nRtn = MCC_SetServoOn(3,0);
nRtn = MCC_SetServoOn(4,0);
nRtn = MCC_SetFeedSpeed(100,nGroup1);  //设置 Group 的进给速度
nRtn = MCC_Line(10,10,0,0,0,0,0,0,0,nGroup1);  //XY 直线插补运动到点（10, 10）
nRtn = MCC_HelicalXY_Z(20,20,10,3,1,nGroup1);//从当前位置开始螺旋运动，XY 平面以
                点（20, 20）为圆心沿 Z 方向以 3mm pitch 值画圆，到达目标 Z 位置 10mm 处
```

5.2 点位运动

5.2.1 指令列表

指令列表

函数名称	说明	页码
MCC_SetPtPSpeed()	设定点位运动速度	
MCC_GetPtPSpeed()	读取设定的点位运动速度	
MCC_PtP()	点对点运动	
MCC_PtPX()	X 轴点对点运动	
MCC_PtPY()	Y 轴点对点运动	
MCC_PtPZ()	Z 轴点对点运动	
MCC_PtPU()	U 轴点对点运动	
MCC_PtPV()	V 轴点对点运动	
MCC_PtPW()	W 轴点对点运动	
MCC_PtPA()	A 轴点对点运动	
MCC_PtPB()	B 轴点对点运动	
MCC _SetPtPAccType()	设定加速形式	
MCC_GetPtPAccType()	读取使用的加速形式	
MCC_SetPtPDecType()	设定减速型式	
MCC_GetPtPDecType()	读取使用的减速速形式	
MCC_SetPtPAccTime()	设定加速时间	
MCC_GetPtPAccTime()	读取使用的加速时间	
MCC_SetPtPDecTime()	设定减速间	
MCC_GetPtPDecTime()	读取使用的减速时间	

特别说明：

点对点运动区别于一般运动，虽然仍以 Group 为基础，但并不进行插补。同一 Group 各轴同时开启运动，但彼此独立进行规划，所以并不保证同时结束运动。MCCL 也提供了单轴的点位运动函数供用户调用，例如 MCC_PtPX()。

点位运动的速度采用最大安全速度的比例来设定，计算方法为：

各轴点位运动速度 = 各轴最大安全速度 ×（速度比例值/100）

其中

各轴的最大安全速度 =（RPM/60）×Pitch/Gear Ratio

点位运动也可以分别设置加减速时间及加减速形式。用户可根据需要调用。

5.2.2 例程

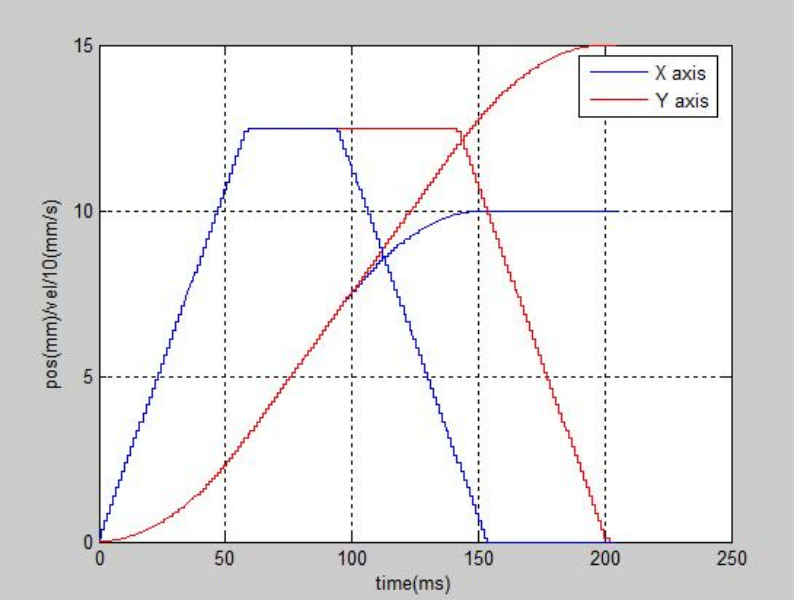


Figure 5.3 Line VS PTP

```
int nRtn = 0;
int nGroup = -1;
long lposx,lposy;
InitSystem();                    //初始化系统
nRtn = MCC_CloseAllGroups();
nGroup = MCC_CreateGroup(0,1,-1,-1,-1,-1,-1,-1,0);  //重新建立 Group
nRtn = MCC_SetPtPAccType('S',nGroup);   //设置 PtP 运动的轨迹规划为 S 型曲线
nRtn = MCC_SetPtPDecType('S',nGroup);
nRtn = MCC_SetPtPAccTime(50,nGroup);  //设置 PtP 运动加速时间
nRtn = MCC_SetPtPDecTime(50,nGroup);
nRtn = MCC_SetPtPSpeed(50,nGroup);   //设置 PtP 运动速度比率
nRtn = MCC_PtP(10,15,0,0,0,0,0,0,nGroup);  //XY 轴执行 PtP 运动到目标点（10, 15），单
                                            位：mm
while(MCC_GetMotionStatus() != GMS_STOP)  //等待运动结束
{
   nRtn = MCC_GetCurPos(&lposx,&lposy,0,0,0,0,0,0,nGroup);
}
nRtn = MCC_TimeDelay(300); //等待 300ms
nRtn = MCC_CloseSystem(); //关闭系统
```

5.3 JOG 运动

5.3.1 函数列表

函数	说明	页码
MCC_JogPulse()	按指定脉冲（Pulse）时动	
MCC_JogSpace()	指定距离（mm）时动	
MCC_JogConti()	连续时动	

JOG 运动基于单轴运行，MCCL 提供了三种 JOG 模式。

1.MCC_JogPulse()要求特定轴按照指定的位移量（最大位移量为 2048 个 Pulse）。JOG 运动开始之前，轴的运动状态应处于停止状态（MCC_GetMotionStatus()的返回值为 GMS_STOP）。

MCC_JogPulse(10, 0, 0);
位移量(pulse) 指定轴 Group 编号

2. MCC_JogSpace()轴按照指定的速度比例（同 PtP 运动）移动指定的位移量（单位：UU）。在调用该函数时，轴需处于运动停止状态。可以使用 MCC_AbortMotionEx()停止该 JOG 运动。

MCC_JogSpace(1, 20, 0, 0);
位移量 速度比例 指定轴 Group 编号

3. 连续 JOG 运动 MCC_JogConti()指定轴按照设定的速度比例与方向，移动到用户设定的有效工作区间边界才停止。在调用该函数时，轴需处于运动停止状态。可以使用 MCC_AbortMotionEx()停止该 JOG 运动。

MCC_JogConti(1, 20, 0, 0);
位移方向 速度比例 指定轴 Group 编号
(1:正向, -1:反向)

5.4 进阶轨迹规划

5.4.1 指令列表

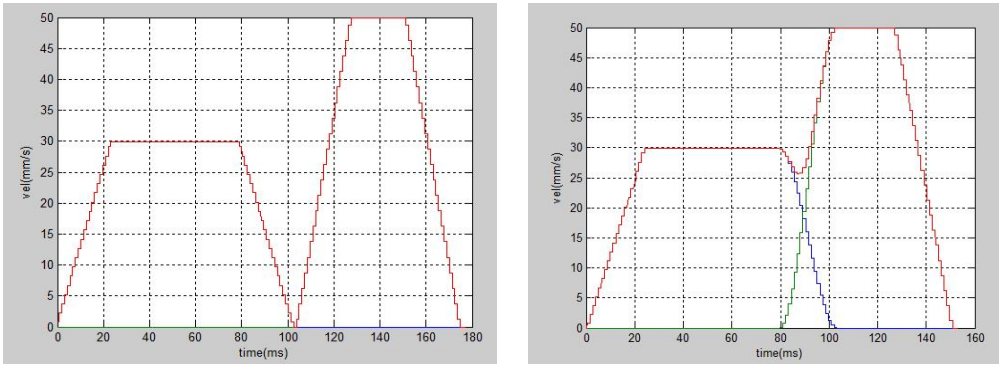
指令列表

函数	说明	页码
MCC_HoldMotion()	暂停运动	
MCC _ContiMotion()	继续运动	
MCC_AbortMotion()	紧急停止，并放弃所有未执行之运动命令	
MCC_AbortMotionEx()	以设定的减速时间减速至停止，并放弃所有未执行之运动命令	
MCC_EnableBlend()	开启平滑运动功能	

MCC_DisableBlend()	关闭平滑运动功能	
MCC_CheckBlend()	检查是否开启平滑运动功能	
MCC_DelayMotion()	设定运动延迟时间	
MCC_CheckDelay()	检查是否进入运动延迟状态	
MCC_OverrideSpeed()	设定一般运动的速度比例	
MCC_GetOverrideRate()	读取使用的一般运动速度比例	

5.4.2 运动平滑（Motion Blending）功能

MCC_EnableBlend()开启平滑运动功能，该功能可实现直线-直线、直线-圆弧、圆弧-圆弧间的连续平滑运动。可通过设置各段的加减速时间来调整接触点的平滑度和运动快慢。下图是未开启和开启 blend 功能的对比：



No Belnd Blend

Figure 5.4 Blend

5.4.3 例程：

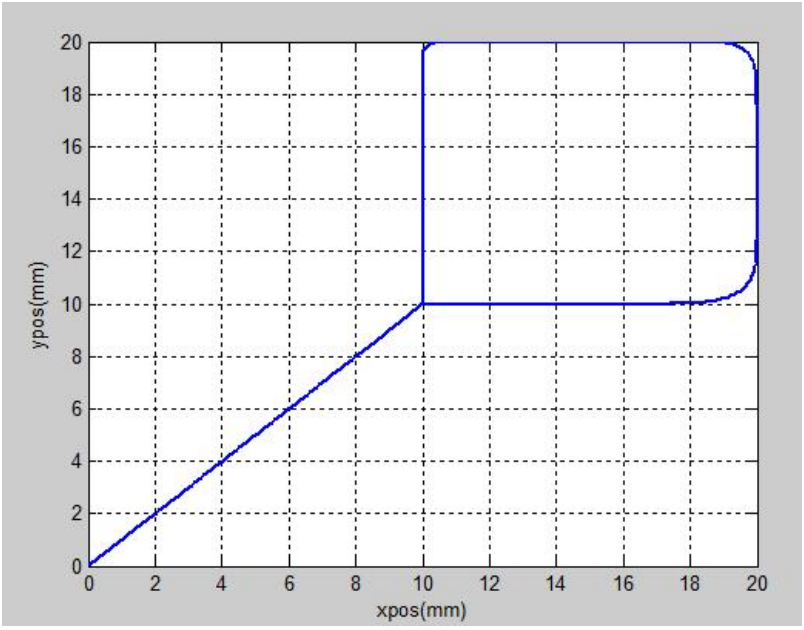


Figure 5.5 Blend example

```
int nRtn = 0;
int nGroup = -1;
long lposx,lposy;
InitSystem();   //初始化系统
nRtn = MCC_CloseAllGroups();
nGroup = MCC_CreateGroup(0,1,-1,-1,-1,-1,-1,-1,0); //重新建立 Group
nRtn = MCC_SetAbsolute(nGroup); //设置为绝对坐标系
nRtn = MCC_SetAccType('T',nGroup); //轨迹设置为 T 型运动规划
nRtn = MCC_SetDecType('T',nGroup);
nRtn = MCC_SetAccTime(60,nGroup); //设置加速度时间
nRtn = MCC_SetDecTime(60,nGroup);
nRtn = MCC_SetFeedSpeed(100,nGroup); //设置进给速度
nRtn = MCC_line(10,10,0,0,0,0,0,0,0,nGroup); //XY 直线插补到目标点（10, 10）
nRtn = MCC_EnableBlend(nGroup); //开启平滑功能
nRtn = MCC_line(20,10,0,0,0,0,0,0,0,nGroup); //XY 直线插补运动到点（20,10）
nRtn = MCC_SetAccTime(50,nGroup); //重新设置加速度时间
nRtn = MCC_SetDecTime(50,nGroup);
nRtn = MCC_line(20,20,0,0,0,0,0,0,0,nGroup); //XY 直线插补走到点(20, 10)
nRtn = MCC_SetAccTime(20,nGroup); //再次改变加减速度时间
nRtn = MCC_SetDecTime(20,nGroup)
nRtn = MCC_line(10,20,0,0,0,0,0,0,0,nGroup); //XY 直线插补走到点（10, 20）
nRtn = MCC_SetAccTime(5,nGroup);   //继续改变加减速度时间
nRtn = MCC_SetDecTime(5,nGroup)
nRtn = MCC_line(10,10,0,0,0,0,0,0,0,nGroup); // XY 直线插补走回点
（10, 10）
while(MCC_GetMotionStatus() != GMS_STOP) //等待运动结束
{
   nRtn = MCC_GetCurPos(&lposx,&lposy,0,0,0,0,0,0,nGroup);
}
nRtn = MCC_TimeDelay(300); //等待 300ms
nRtn = MCC_DisableBlend(nGroup); //关闭平滑功能
nRtn = MCC_CloseSystem();   //关闭系统
```

在该例中我们在使用使用 Blend 功能时，通过更改运行线段的加减速来实现不同的转角轨迹。后一段的加速时间越小（即加速度越大）轨迹越接近真实轨迹。Blend 指令将增加命令缓冲区命令笔数。

5.4.4 运动中更改速度（速度强制）

如果需要在运动之中更改进给速度，可以使用速度强制功能。此功能可将执行中的运动速度加速或减速到要求的速度值。如下图所示：

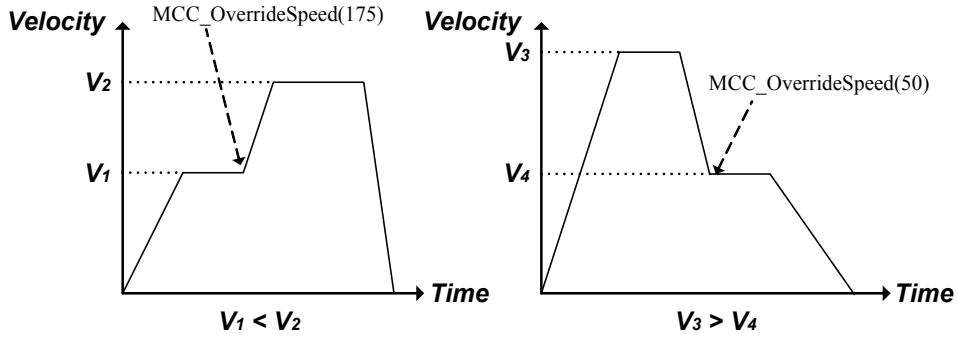


Figure 5.6 速度强制改变

上图中，$V_2 = V_1 \times 175 / 100$(因调用 MCC_OverrideSpeed(175))；同理，$V_4 = V_3 \times 50 / 100$（调用 MCC_OverrideSpeed(50)）。

原进给速度是指 MCC_SetFeedSpeed()或 MCC_SetPtPSpeed()所设定的速度。***特别注意：使用 MCC_OverrideSpeed()后，将影响后面所有运动的速度，而不只是当前正在执行中的运动。***

5.4.5 运动延迟，运动暂停、持续、弃置

1 可以使用 MCC_DelayMotion()强制延迟执行下一个运动命令，延迟的时间以毫秒(ms)为单位，如下例：

```
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, 1);-------- A
MCC_DelayMotion(200, 1);
MCC_Line(15, 15, 15, 0, 0, 0, 0, 0, 1);-------- B
```

该指令进入了命令缓冲区 Queue 中，在执行完运动命令 A 后将延迟 200 ms，再继续执行运动命令 B。

2. 调用 MCC_HoldMotion()暂停执行中的运动命令（此时以等减速的方式停止运动），待调用 MCC_ContiMotion()后，才继续执行该笔命令尚未完成的部分。此时也可以使用 MCC_AbortMotionEx()来舍弃尚未完成的部分。

MCC_AbortMotionEx()可使用指定的减速时间來停止运动，若目前已在 Hold 状态则减速时间参数将被忽略。

5.5 到位确认（In-Position）

指令列表

函数	说明	页码
MCC_SetInPosMode()	设定到位确认使用模式	
MCC_SetInPosMaxCheckTime()	设定到位检查时间	
MCC_SetInPosSettleTime()	设定到位持续时间	
MCC_EnableInPos()	开启到位确认功能	
MCC_DisableInPos()	关闭到位确认功能	
MCC_SetInPosToleranceEx()	设定到位误差容许范围	
MCC_GetInPosToleranceEx()	读取设定的到位误差容许范围	
MCC_GetInPosStatus()	确认实际位置是否已满足到位确认条件	

MCCL 所提供的到位确认功能在确保执行中的运动命令已实际到达目标位置(容许位置误差范围内) 后，才继续执行下一条指令，否则弃置后续命令并产生错误记录 (亦可选择忽略)。

用户调用 MCC_EnableInPos()开启到位检测功能，MCCL 在发完位置指令之后，开始检测是否满足到位条件；若是，则执行下一条命令，但若一直等到所设定之最大检查时间(MCC_SetInPosMaxCheckTime()函数设定)截止前仍未定位，则弃置后续命令并产生错误记录。最大检查时间如下图所示：

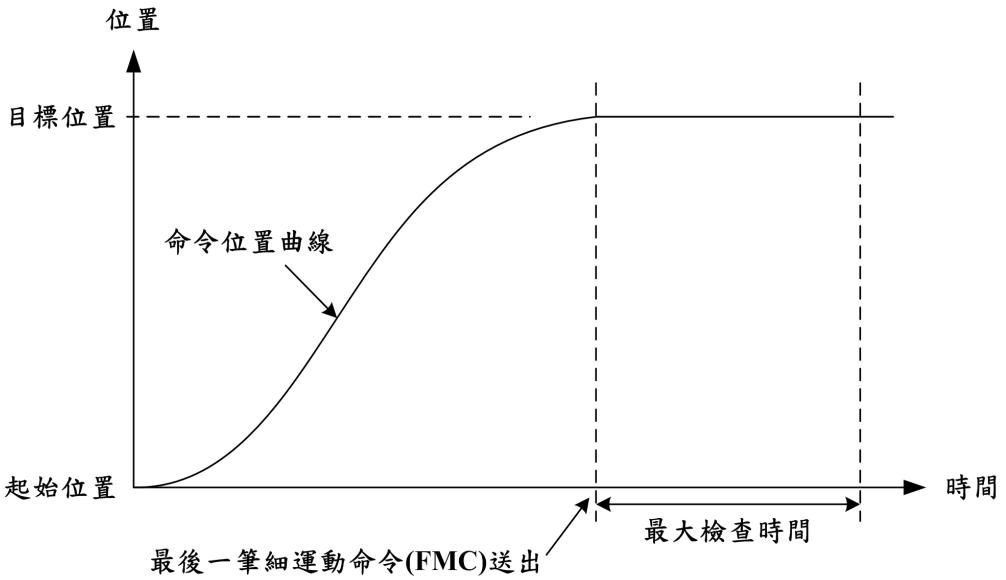


Figure 6.1 最大检查时间示意图

MCCL 提供了四种到位确认模式，用户可通过调用函数 MCC_SetInPosMode()选择合适的方式。以下为各模式介绍：

模式 IPM_ONETIME_BLOCK：

Group 中各轴的位置误差都小于或等于定位误差允许范围(通过调用

MCC_SetInPosToleranceEx()设定，单位：UU)，若到最大检查时间截止前未满足以上条件，则弃置后续命令并产生错误记录(可调用 MCC_GetErrorCode()获得错误码)。

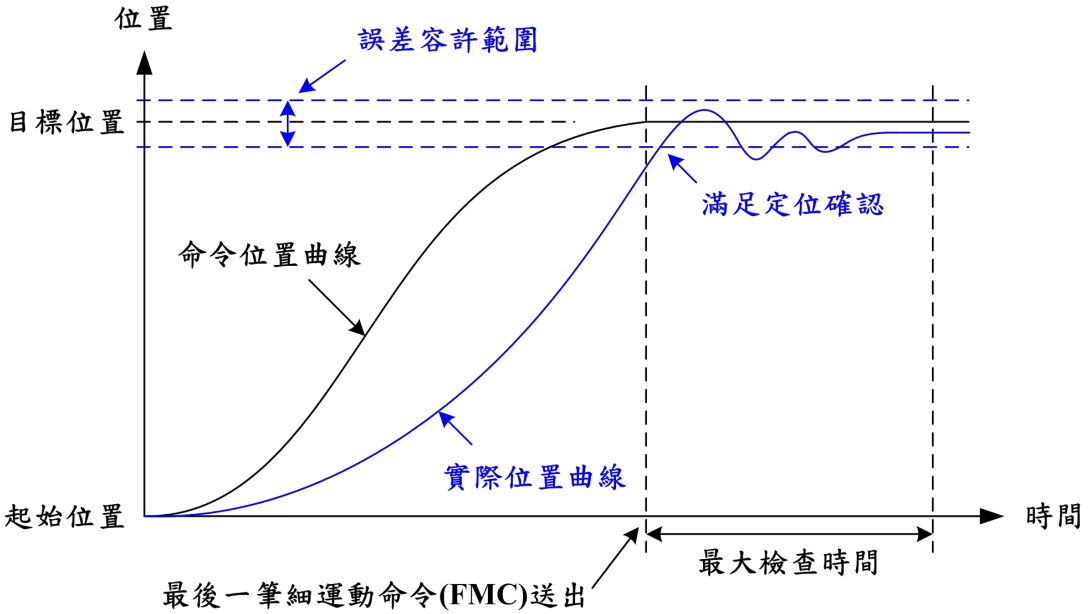


Figure6.2　IPM_ONETIME_BLOCK 模式示意图

模式 IPM_ONETIME_UNBLOCK：

此模式与 IPM_ONETIME_BLOCK 模式的定位条件相同 ,差别仅在于当最大检查时间截止时若未满足定位条件，并不产生错误记录而后续命令。

模式 IPM_SETTLE _BLOCK：

Group 中各轴的位置误差皆小于或等于定位误差容许范围(可调用 MCC_SetInPosToleranceEx() 设定，单位：UU)，并持续一段稳定时间(可调用 MCC_SetInPosSettleTime()来设定，单位：ms)，若到最大检查时间截止前皆未满足上述条件，则弃置后续命令并产生错误记录(MCC_GetErrorCode()可获得错误码)。如下图所示：

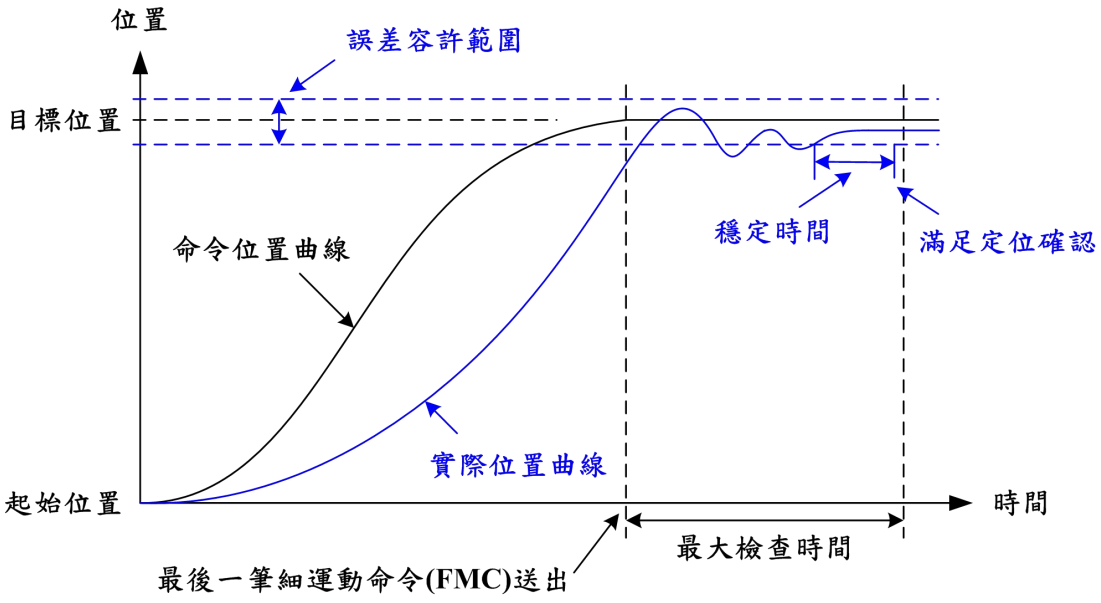


Figure 6.3　IPM_SETTLE _BLOCK 模式示意图

模式 IPM_SETTLE _UNBLOCK：

该模式与 IPM_SETTLE_BLOCK 模式的定位条件相同，差别仅在于当最大检查时间截止时若未满足定位条件，并不产生错误记录而直接执行后续命令。

定位容许误差越大，则完成到位确认所需时间相对越少，但轨迹误差会较大(反之则误差较小)。用户可跟据实际应用需求来设定。调用 MCC_GetInPosStatus()可获取 Group 中各运动轴的到位状态。

注意：

因到位确认功能是比较机台实际位置与目标位置是否进入容许误差范围，开启此功能须配接编码器获取实际机台位置。

系統一旦判定到位成功，则不再进行定位确认判断(保持在定位成功的状态，即使实际位置又偏离定位允许范围)，直到有新的运动命令下达。

5.6 跟随误差检测（Tracking error）

指令列表

函数	说明	页码
MCC_EnableTrackError()	开启轨迹误差检查功能	
MCC_DisableTrackError()	关闭轨迹误差检查功能	
MCC_SetTrackErrorLimit()	设定轨迹误差容许范围	
MCC_GetTrackErrorLimit()	读取轨迹误差容许范围	

跟随误差：在整个运动过程中，运动轴命令位置与实际位置的误差。

一般情况下，跟随误差大小跟机构刚性，运动加速度，PID 控制参数设定等有关。过大的跟随误差表示运动偏离规划路径较远，或者已经发生撞机。所以 MCCL 提供函数 MCC_SetTrackErrorLimit()来对误差进行保护，用户可调用 MCC_EnableTrackError()开启此次功能。下图为跟踪误差示意图：

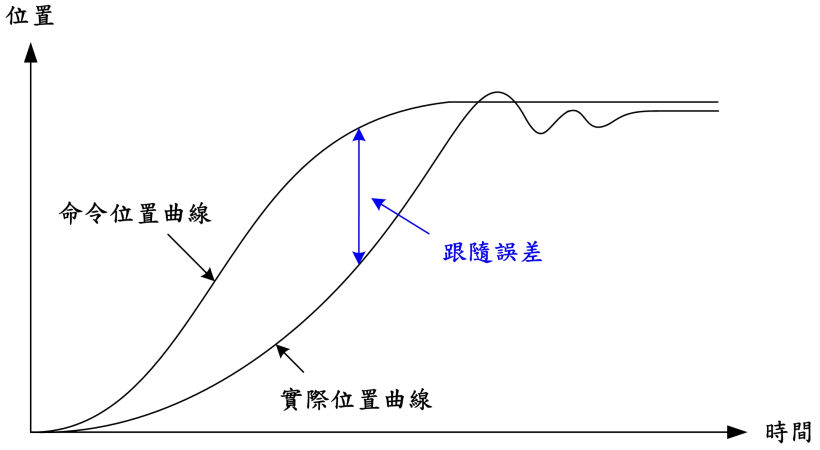


Figure 6.4 Tracking error

特别说明：

当使用 MCCL，在任何情况下若调用 MCC_GetErrorCode()得到非零返回值，表示该 Group 已产生错误记录，处理方式如下：

(1) 判断错误类型并排除错误(用户需自行在程序中定义)；

(2) 呼叫 MCC_ClearError()清除错误记录；

(3) 系统继续正常运作。

5.7 齿轮齿隙、背隙补偿

参数列表

函数	说明	页码
MCC_SetCompParam()	设定齿轮齿隙、间隙补偿参数	
MCC_UpdateCompParam()	反应更新后的齿轮齿隙、间隙补偿参数	

机台在做定位控制时，齿轮、丝杆的固有制造精度往往会使机台的运动产生一定的位置误差。反向间隙误差是指由于传动链中机械间隙的存在，执行部件在运动过程中，当运动方向发生改变时，执行部件的运动量与理论量存在误差，最后将反映为叠加至工件上的加工精度的误差。

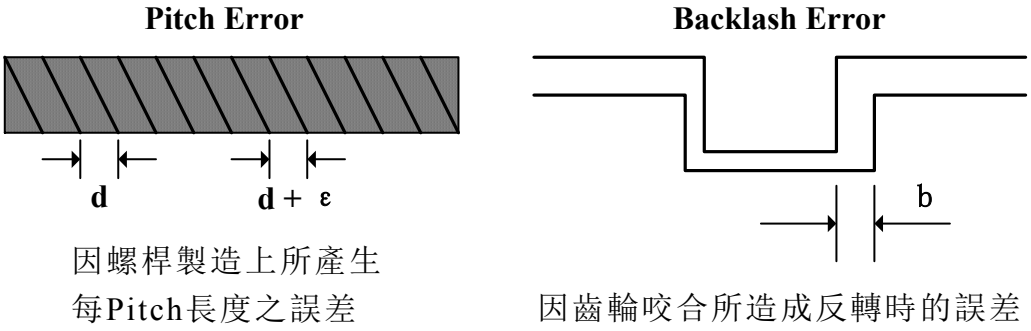


Figure 齿轮齿隙、间隙误差

用户可将机台行程分成等距多条小线段(如下图 Figure)，使用激光干涉仪等测试设备，在正负方向来回扫描一次，并记录各段误差值，建成正负向补偿表。正、负向误差补偿表为一个二位数组，存放各轴所有的补偿点补偿值。所有补偿点均以同一点为参考基准(参考下图)。用户需给定 *dwInterval*、*wHome_No*、正、负向补偿表(*nForwardTable* 与 *nBackwardTable*)，并调用补偿函数 MCC_SetCompParam()与 MCC_UpdateCompParam() ,即可进行位置补偿。MCCL 为每轴提供 256 个补偿点，最多可将机台轴的行程分成 255 个补偿区段，在各区段间使用线性补偿。

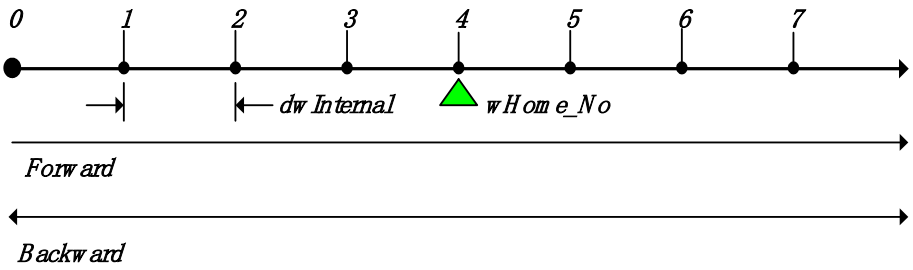


Figure 6.5 补偿区段

使用补偿功能时，补偿参数的内容必须涵盖机台全部的工作行程，以避免产生不正常的动作，因此应在完成原点归位动作后启动补偿功能，可以调用 MCC_GetGoHomeStatus()检查原点归位动作是否已完成(函数返回值为 1，表示归位完成)。

停止补偿功能的方法是补偿参数中的 *dwInterval* 设为 0，例如停止 Channel 0 的补偿功能可执行下面的代码：

```
SYS_COMP_PARAM      stUserCompParam;
stUserCompParam.dwInterval = 0;

MCC_SetCompParam(&stUserCompParam, 0, 0);
MCC_UpdateCompParam();
```

在使用补偿功能前需先设定补偿参数，补偿参数定义如下：

```
typedef struct _SYS_COMP_PARAM
{
    DWORD          dwInterval;
    WORD           wHome_No;
    WORD           wPaddle;
    int            nForwardTable[256];
    int            nBackwardTable[256];
} SYS_COMP_PARAM;
```

参数	说明
dwInterval	补偿区段的间距，单位：Pulse 若此值<= 0，则不进行补偿功能
wHome_No	参考位置补偿点编号
wPaddle;	预留
nForwardTable[256];	指向正向补偿表的指针变量
nBackwardTable[256]	指向负向补偿表的指针变量

以上图为例，若将 X 轴的工作行程划分为 7 段，也就是共需测量 8 个补偿点(0 ~ 7)，其中原点位于编号 4 的补偿点上，也就是在完成原点归位动作后，系统处于编号 4 的位置上。若设置 *dwInterval* 为 10000(pulses)，则正向工作范围是 10000 × (7 – 4) = 30000 (pulses)，负向工作范围是 10000 × (4 - 0) = 40000 (pulses)。机台参数中的 *dwHighLimit* 和 *dwLowLimit* 不能与之相冲突。各轴的补偿参数要分开设定，下面为 X 轴补偿参数示例：

```
SYS_COMP_PARAM      stUserCompParam;

stUserCompParam.dwInterval          = 10000;
stUserCompParam.wHome_No            = 4;

stUserCompParam.nForwardTable[0]    = 22;//  单位为 pulse
stUserCompParam.nForwardTable[1]    = 20;
stUserCompParam.nForwardTable[2]    = 15;
stUserCompParam.nForwardTable[3]    = 11;
stUserCompParam.nForwardTable[4]    = 0; //原点所在位置，设为 0
stUserCompParam.nForwardTable[5]    = 10;
stUserCompParam.nForwardTable[6]    = 12;
```

```
stUserCompParam.nForwardTable[7]     = 15;

MCC_SetCompParam(&stUserCompParam, 0, CARD_INDEX);
MCC_UpdateCompParam();
```

如上所述，使用者最多可将行程分成 255 个补偿区段，而在每一个补偿区段中将使用线性补偿的方法进行补偿。例如目前机台的 X 轴（正处于编号 4 的位置上）需向右前进 15000 pulses，由误差补偿表（参考 stUserCompParam）可得知此位置位于 nForwardTable[5]与 nForwardTable[6]所定义胡区段之间(即介于 10000 pulses 与 20000 pulses 之间)，因 nForwardTable[5]之值为 10、nForwardTable[6] = 12，且 nForwardTable[6]- nForwardTable[5]= 12 - 10 = 2，因此系统实际发送 15000 + 10 + (int)((15000 – 10000)/ 10000 × 2) = 15000 + 10 + 1 = 15011 pulses。

5.8 例程

到位确认：

```
#include <QCoreApplication>
#include <QDebug>
#include "stdio.h"

#include "../../newlib/mccl.h"
#include "../../newlib/MCCL_Fun.h"

#define CARD_INDEX                      0
#define ESC_KEY                         27
int       nGroup = -1;

 int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  char        cKey;
  BYTE        byInPos0, byInPos1, byInPos2, byInPos3, byInPos4, byInPos5, byInPos6,
              byInPos7;
     WORD     wCardType = 4;
  int         nErrCode;
     int        nTemp =0;
   int        nRtn = 0;
  double      dfCurPosX, dfCurPosY, dfCurPosZ, dfCurPosU, dfCurPosV, dfCurPosW,
              dfCurPosA, dfCurPosB;
  InitSystem();   //系统初始化
  Qstring str;
  MCC_SetServoOn(0, CARD_INDEX); //使能伺服
  MCC_SetServoOn(1, CARD_INDEX);
  MCC_SetInPosMaxCheckTime(20, g_nGroup);// 设置到位检查时间：  20 X
(interpolation time interval)
    MCC_SetInPosMode(IPM_ONETIME_BLOCK, nGroup);//设置到位检测模式为检
测时间内到位即可，若为满足弃置命令
    MCC_SetInPosToleranceEx(0.01, 1000, 1000, 1000, 1000, 1000, 1000,1000,
nGroup);//设置到位误差允许范围 (unit : pulse)
    MCC_SetIncrease(nGroup);//使用相对坐标
    MCC_SetAccTime(300, nGroup);//设置加速度时间为 300 ms
    MCC_SetDecTime(300, nGroup);
```

```
        MCC_SetFeedSpeed(10, nGroup); //设置进给速度(unit : mm/sec)
        qDebug()<<"Press any key to start, and ESC key to quit !\n\n";
        qDebug()<<"1. Press ESCAPE key to quit\n";
        qDebug()<<"2. Press 'L' key to execute line motion\n";
        qDebug()<<"3. Press 'E' key to enable InPos motion\n";
        qDebug()<<"4. Press 'D' key to disable InPos motion\n");
        qDebug()<<"5. Press 'C' key to clear error\n\n";
        while (1)
        {
            cKey = getwchar();

            if (cKey == ESC_KEY)
            {
                MCC_AbortMotionEx(150, nGroup);//以设定减速时间停止当前运动并
    弃置命令队列中后续运动命令，减速时间设为 0 时运动急停
                break;
                }
            else if (cKey == 'l' || cKey == 'L')
                MCC_Line(10, 0, 0, 0, 0, 0, 0, 0, nGroup);
            else if (cKey == 'e' || cKey == 'E')
                MCC_EnableInPos(nGroup);//开启到位检测功能
            else if (cKey == 'd' || cKey == 'D')
                nTemp = MCC_DisableInPos(nGroup);
            else if (cKey == 'c' || cKey == 'C')
                MCC_ClearError(nGroup);
            }

        MCC_GetCurPos(&dfCurPosX, &dfCurPosY, &dfCurPosZ, &dfCurPosU, &dfCurPosV,
        &dfCurPosW, &dfCurPosA, &dfCurPosB nGroup);//获取当前位置(unit : mm)
            nErrCode = MCC_GetErrorCode(nGroup);//获取错误码
      // 当 status == 255, 确认到位
    MCC_GetInPosStatus(&byInPos0, &byInPos1, &byInPos2, &byInPos3, &byInPos4,
&byInPos5，&byInPos6，&byInPos7，nGroup);
    str.sprintf("Cur. Pos: %1    Status: %2   Err: 0x%3    Dis-InPos: %4 \r")
                .arg(dfCurPosX)
                .arg(byInPos0)
                .arg(MCC_GetErrorCode())
                .arg(nTemp);
        qDebug()<<str;
            }

            MCC_CloseSystem();//关闭系统
            printf("\n\n");
            return a.exec();
        }
```

第六章　安全机制

本章介绍 MCC 控制器提供的所有可用安全机制，包括软硬限位、急停、平滑停止、跟随误差极限停止。

指令列表

函数	说明	页码
MCC_SetOverTravelCheck()	设定软件过行程保护功能	
MCC_GetOverTravelCheck()	读取软件过行程保护的设定	
MCC_GetErrorCode()	获取错误码	
MCC_ClearError()	清除错误	

6.1 报警

运动控制器提供专用的驱动器报警信号输入接口。当检测到驱动器报警信号后，运动控制器将关闭该轴的伺服使能，停止运动规划。

驱动器报警信号产生后，应执行以下操作：

（1）确定引起驱动器报警的原因，并加以改正；

（2）复位驱动器；

6.2 硬件急停与命令停止

AMC Series 控制器提供了硬件 Emergency Stop IO，用户可接入急停按钮，在紧急情况下快速关闭控制器，以防止严重撞机等事故的发生，保障机械与操作人员的安全。可以调用 MCC_GetEmgcStopStatus()函数来查询该接口信号状态。

若用户在机台运行过程中需要进行停止操作（如搜查停止等），可以调用函数 MCC_AbortMotionEx()以设定的减速时间减速至停止，并放弃所有未执行的运动命令，当减速时间设为零时，则运动急停。

6.3 跟随误差极限

对于伺服系统而言，当出现异常情况时（如电机故障、编码器信号反向或机械故障造成电机堵转等），电机的实际位置与规划位置会出现较大的偏差。为了及时检测这些情况，提高系统的安全性，用户可通过函数 MCC_SetTrackErrorLimit()来设置最大跟随误差极限。当跟随误差超过设定值时，系统报错停止，调用 MCC_GetErrorCode()可获取相应的错误码。

详细的操作说明请见“**6.3 跟随误差检测**”章节。

第七章　指令详解

系统功能设置：

void MCC_GetVersion(char* *strVersion*)

Description	读取函数库版本
指令类型	立即执行指令
Parameters	*strVersion* 指向一内存缓冲区，用来接收函数库版本
Return Value	0　　成功 非零　失败，传回值的意义请参考 **2.1 指令返回值**

int MCC_CreateGroup(int *xMapToCh*，
int *yMapToCh*，
int *zMapToCh*，
int *uMapToCh*，
int *vMapToCh*，
int *wMapToCh*，
int *aMapToCh*，
int *bMapToCh*，
int *nCardIndex = 0*）

Description	此函数用来建立一个新的运动群组。 在呼叫 MCC 中与运动群组有关的函数（Ex: MCC_Line）之前，必需先建立群组，并得到新建立群组的编号，作为其传入参数之一。	
指令类型	立即执行指令	章节页码
Parameters	*xMapToCh* 指定此 Group 中 X 轴所对应的控制器输出 Channel(0~7)；其余参数与之类似。	
Return Value	大于等于零　新建立之群组编号 小于零　失败，传回值的意义请参考 **2.1 函数返回值**	

int MCC_CloseGroup(int *nGroupIndex*)

Description	关闭指定的群组
指令类型	立即执行指令
Parameters	*nGroupIndex*　group 编号
Return Value	0　　成功 非零　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CloseAllGroups()

Description	关闭系统中所有群组。在第一次调用 MCC_CreateGroup 之前，请先呼叫此函数
指令类型	立即执行指令
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetMacParam(SYS_MAC_PARAM *pstMacParam,
WORD *wChannel*,
WORD *wCardIndex = 0)*

Description	设定指定控制轴的机构参数	
指令类型	立即执行指令	章节页码
Parameters	*pstMacParam* 指向一 SYS_MAC_PARAM 结构，内含欲设定之机构参数 *wChannel* 运动控制卡的输出 Channel(0 ~ 7)	
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**	

int MCC_GetMacParam(SYS_MAC_PARAM *pstMacParam,
WORD *wChannel,*
WORD *wCardIndex = 0*)

Description	读取指定控制轴的机构参数内容
指令类型	立即执行指令
Parameters	*pstMacParam* 指向一 SYS_MAC_PARAM 结构，用来接收欲读取之机构参数内容 *wChannel* 运动控制卡的输出 Channel(0 ~ 7)
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_UpdateParam()

Description	更新设置的机构、编码器、原点复归参数。在调用过 MCC_InitSystem 后，如果再调用 MCC_SetMacParam、MCC_SetEncoderConfig 变更相关的参数，则需使用此函数更新设定值；但须注意，呼叫此函数之结果与 MCC_ResetMotion 相似，系统将回到初始状态	
指令类型	立即执行指令	章节页码

Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetCmdQueueSize(int *nSize*，
WORD *wGroupIndex*)

Description	设定运动命令缓冲区的大小		
指令类型	立即执行指令	章节页码	
Parameters	*nSize* 运动命令缓冲区的大小(单位为运动令) *wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_GetCmdQueueSize(WORD *wGroupIndex)*

Description	读取运动命令缓冲区的大小
指令类型	立即执行指令
Parameters	*n wGroupIndex* group 编号
Return Value	大于或等于零 运动命令缓冲区的大小(单位为运动命令) 小于零 失败

int MCC_InitSystem(int *nInterpolateTime*，
SYS_CARD_CONFIG * *pstCardConfig*，
WORD *wCardNo = 1*)

Description	启动运动控制器 除了 MCC_CreateGroup、MCC_SetMacParam、MCC_SetEncoderConfig、MCC_SetHomeConfig、MCC_SetCompParam 之外，在使用其他函数之前，必须先呼叫此函数，此函数仅需呼叫一次即可。		
指令类型	立即执行指令	章节页码	
Parameters	*nInterpolateTime* 插值时间，单位为 ms，设定范围为 1ms ~ 50ms，较小的插值时间会使控制器拥有较佳的运作性能，但需视系统运行负荷而定。一般系统使用 2ms 的设定值即可 *pstCardConfig* 运动控制卡硬件参数 *wCardNo* 运动控制卡的使用张数		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_CloseSystem()

Description	关闭运动控制函数库
指令类型	立即执行指令

Return Value	0 成功 非零 失败

int MCC_ResetMotion()

Description	重置运动控制函数库。调用后，将清除错误状态，并使直角坐标值与马达坐标值归零，系统会回到如同调用 MC_InitSystem 过后的初始状态
指令类型	立即执行指令
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_EnableDryRun()

Description	开启运动空跑功能。开启后，运动命令计算后并不输出，此时可利用 MCC_GetCurPos 与 MCC_GetPulsePos 读取所需的坐标值进行分析或绘图		
指令类型	立即执行指令	章节页码	
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_DisableDryRun()

Description	关闭运动空跑功能
指令类型	立即执行指令
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CheckDryRun()

Description	检查运动空跑功能设定状态
指令类型	立即执行指令
Return Value	0 运动空跑功能已开启 1 运动空跑功能关闭中 其他 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetSysMaxSpeed(double *dfMaxSpeed*)

Description	设定一般运动(直线、圆弧、圆、螺线)进给速度的上限，避免在使用 MCC_SetFeedSpeed 时所设定的进给速度超出系统的工作范围，单位:UU/sec		
指令类型	立即执行指令	章节页码	

Parameters	*dfMaxSpeed* 进给速度的上限
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

double MCC_GetSysMaxSpeed()

Description	读取一般运动(直线、圆弧、圆、螺线)进给速度的上限，单位:UU/sec
指令类型	立即执行指令
Return Value	进给速度的上限

int MCC_GetCurGroup()

Description	读取当前所对应的组(Group)		
指令类型	立即执行指令	章节页码	
Return Value	-1 读取当前组失败 其他值 返回当前 Group 编号		

int MCC_SetCurGroup(WORD *wGroupIndex*)

Description	设置当前运动控制的组(Group)
指令类型	立即执行指令
Parameters	wGroupIndex 设置当前运动组(Group)编号
Return Value	-1 设置失败 其他值 返回当前设置的 Group 编号

int MCC_SetServoOn(WORD *wChannel*, WORD *wCardIndex= 0*)

Description	使能指定控制轴伺服驱动		
指令类型	立即执行指令	章节页码	
Parameters	*wChannel* 运动控制卡的输出 Channel(0 ~ 7)		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_SetServoOff(WORD *wChannel*，WORD *wCardIndex = 0*)

Description	关闭指定控制轴伺服驱动
指令类型	立即执行指令

Parameters	*wChannel* 运动控制卡的输出 Channel(0 ~ 7)
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

系统坐标

int MCC_SetAbsolute(WORD *wGroupIndex*)

Description	采用绝对坐标
指令类型	缓冲区指令
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetIncrease(WORD *wGroupIndex*)

Description	使用增量坐标
指令类型	缓冲区指令 章节页码
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetCoordType(WORD *wGroupIndex*)

Description	读取使用的坐标模式
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 使用增量坐标 1 使用绝对坐标 其他 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetCurRefPos(double **pdfX*,
double **pdfY*,
double **pdfZ*,
double **pdfU*,
double **pdfV*,
double **pdfW*,
double **pdfA*,
double **pdfB*,
WORD *wGroupIndex*)

Description	读取当前各轴直角坐标位置值*(不含补偿)*		
指令类型	立即执行指令	章节页码	
Parameters	*pdfX ~ pdfB* 指向一 double 值，用来存放 X ~ B 各轴目前的直角坐标位置值（不含补偿），单位：UU wGroupIndex group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_GetCurPos(double *pdfX,
double *pdfY,
double *pdfZ,
double *pdfU,
double *pdfV,
double *pdfW,
double *pdfA,
double *pdfB,
WORD *wGroupIndex*)

Description	读取目前各轴直角坐标位置值*(含补偿)*
指令类型	立即执行指令
Parameters	*pdfX ~ pdfB* 指向一 double 值，用来存放 X ~ B 各轴目前位置之直角坐标值（含补偿） wGroupIndex group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetPulsePos(long *plX,
long *plY,
long *plZ,
long *plU,
long *plV,
long *plW,
long *plA,
long *plB,
WORD *wGroupIndex*)

Description	读取目前各轴马达坐标位置值 *(或称为 pulse 坐标值，含补偿)*
指令类型	立即执行指令
Parameters	*plX ~ plB* 指向一 long 值，用来存放 X ~ B 各轴目前位置的马达坐标值(含补偿) wGroupIndex group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DefineOrigin(WORD *wAxis*, WORD *wGroupIndex*)

Description	使特定 group 指定的运动轴之坐标值归零，使用此函数需该 group 为运动停止状态
指令类型	立即执行指令　　章节页码
Parameters	*wAxis*　　指定的运动轴编号 0～7 分别代表 X～B 轴 wGroupIndex　　group 编号
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DefinePosHere(WORD *wGroupIndex*,
DWORD *dwAxisMask*)

Description	修正当前的系统坐标值使与机台实际位置相符。 在某些情况下有可能使用手动方式移动机台，这时候机台实际的位置与运动控制函数库中的系统坐标值将产生偏差，假使系统*有安装编码器*，则在呼叫此函数成功后将利用编码器的计数值修正系统坐标值，此时系统坐标值将反应机台真实的位置
指令类型	立即执行指令　　章节页码
Parameters	*wGroupIndex*　　group 编号 *dwAxisMask* 指定的运动轴，指定参数可为 MCC_AXIS_X　X　轴　； MCC_AXIS_Y　　Y 轴 MCC_AXIS_Z　　Z 轴；　MCC_AXIS_U　　U 轴 MCC_AXIS_V　V 轴；　MCC_AXIS_W　　W 轴 MCC_AXIS_A　A 轴；　MCC_AXIS_B　　B 轴 MCC_AXIS_ALL　　全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例： (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DefinePos(WORD *wAxis*,　DOUBLE　*dfPos*，
WORD　*wGroupIndex*)

Description	设定目前的系统坐标值。 在某些情况下有可能机台需要重新启动，这时候控制卡上的编码器位置将被归零，当运动控制函数库重新初始化后，此时机台实际的位置与运动控制函数库中的系统坐标值将产生偏差，假使系统*有安装绝对值编码器*，则在呼叫此函数成功后将利用绝对值编码器的计数值修正系统坐标值，此时系统坐标值将反应机台真实的位置
指令类型	立即执行指令
Parameters	*wAxis*　指定的运动轴编号 0～7 分别代表 X～B 轴 *dfPos*设定的系统坐标值 *wGroupIndex*　　group 编号
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

过行程保护

int MCC_GetOverTravelCheck(int *pnOTCheck0,
int *pnOTCheck1,
int *pnOTCheck2,
int *pnOTCheck3,
int *pnOTCheck4,
int *pnOTCheck5,
int *pnOTCheck6,
int *pnOTCheck7,
WORD wGroupIndex)

Description	读取软件过行程保护的设定情形
指令类型	立即执行指令
Parameters	*pnOTCheck0 ~ pnOTCheck7* 指向一 int 值，用来存放 X~B 各轴目前软件过行程保护的设定情形， 1 表示开启； 0 则表示关闭 wGroupIndex group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

直线、圆弧、圆、螺线运动(一般运动)

int MCC_SetAccType(char *cAccType*, WORD *wGroupIndex*)

Description	设定在进行一般运动的加速型式		
指令类型	缓存区指令	章节页码	
Parameters	*cAccType* 各轴的加速型式，可设定为： 'T' 使用梯形加速曲线 'S' 使用 S 形加速曲线 wGroupIndex group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_GetAccType(WORD *wGroupIndex*)

Description	读取在进行一般运动时使用的加速型式
指令类型	立即执行指令
Parameters	*c wGroupIndex* group 编号
Return Value	0 目前使用梯形加速曲线 1 目前使用 S 形加速曲线

int MCC_SetDecType(char *cDecType*, WORD *wGroupIndex*)

Description	设定在进行一般运动的减速型式		
指令类型	缓存区指令	章节页码	
Parameters	*cDecType* 各轴的减速型式， 'T' 使用梯形减速曲线 'S' 使用 S 形减速曲线 wGroupIndex group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_GetDecType(WORD *wGroupIndex*)

Description	读取在进行一般运动时使用的减速型式
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 目前使用梯形减速曲线 1 目前使用 S 形减速曲线

int MCC_SetAccTime(double *dfAccTime*,
WORD *wGroupIndex*）

Description	设定一般运动加速到稳定速度所需的时间
指令类型	缓存区指令 章节页码
Parameters	*dfAccTime* 要求的加速时间，必须大于 0，单位为 ms。 *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

double MCC_GetAccTime(WORD *wGroupIndex*)

Description	读取一般运动加速到设定速度所需的时间
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	一般运动加速到设定速度所需的时间，单位为 ms。

int MCC_SetDecTime(double *dfDecTime*, WORD *wGroupIndex*)

Description	设定一般运动由设定速度减速至停止运动所需的时间
指令类型	缓存区指令 章节页码
Parameters	*dfDecTime* 要求的减速时间，必须大于 0，单位为 ms。 *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

double MCC_GetDecTime(WORD *wGroupIndex*)

Description	读取一般运动由设定速度减速至停止运动所需的时间
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	进行一般运动时由稳定速度减速至停止运动所需的时间，单位为 ms。

double MCC_SetFeedSpeed(double *dfFeedSpeed*,
WORD *wGroupIndex*)

Description	设定一般运动的进给速度，单位为 UU/sec；此值不可为 0。但一般运动实际操作时的进给速度需参考是否曾使用 MCC_OverrideSpeed()设定进给速度倍率，例如最后一次设定进给速度倍率为使用 MCC_OverrideSpeed(150)，假使此时呼叫 MCC_SetFeedSpeed(10)，则一般运动实际使用的进给速度为 10×150% = 15		
指令类型	缓存区指令	章节页码	
Parameters	*dfFeedSpeed* 要求的进给速度 *wGroupIndex* group 编号		
Return Value	实际设定的进给速度		

double MCC_GetFeedSpeed(WORD *wGroupIndex*)

Description	读取一般运动所设定的进给速度。利用此函数所获得的进给速度并未加入使用 MCC_OverrideSpeed()后对实际进给速度的影响，而只是单纯传回 MCC_SetFeedSpeed() 时所使用的设定值，此部份请参考对 MCC_SetFeedSpeed()的说明。		
指令类型	立即执行指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	目前设定的进给速度		

double MCC_GetCurFeedSpeed(WORD *wGroupIndex*)

Description	读取机台目前实际的进给速度。
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	机台目前实际的进给速度

int MCC_GetSpeed(double *pdfVel0,
double *pdfVel1,
double *pdfVel2,
double *pdfVel3,
double *pdfVel4,
double *pdfVel5,
double *pdfVel6,
double *pdfVel7,
WORD wGroupIndex)

Description	读取各轴目前的进给速度值		
指令类型	立即执行指令	章节页码	
Parameters	*pdfVel0 ~ pdfVel7* 指向一 double 值，用来存放各轴目前的进给速度值 *wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_Line(double dfX0,
double dfX1,
double dfX2,
double dfX3,
double dfX4,
double dfX5,
double dfX6,
double dfX7,
WORD wGroupIndex,
DWORD dwAxisMask)

Description	从当前位置以直线插补方式运动到指定的目标点。		
指令类型	缓存区指令	章节页码	
Parameters	*dfX0 ~ dfX7* 目标点的坐标值 *wGroupIndex* group 编号 *dwAxisMask* 指定欲发生作用的轴，指定参数可为 MCC_AXIS_X X 轴；MCC_AXIS_Y Y 轴 MCC_AXIS_Z Z 轴；MCC_AXIS_U U 轴 MCC_AXIS_V V 轴；MCC_AXIS_W W 轴 MCC_AXIS_A A 轴； MCC_AXIS_B B 轴 MCC_AXIS_ALL 全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例： (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)		

Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_LineX(double *dfX*, WORD *wGroupIndex*)

Description	X 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfX*　　目标点的坐标值 *wGroupIndex*　　group 编号		
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineY(double *dfY*, WORD *wGroupIndex*)

Description	Y 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfY*　　目标点的坐标值 *wGroupIndex*　　group 编号		
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineZ(double *dfZ*, WORD *wGroupIndex*,)

Description	Z 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfZ*　　目标点的坐标值 *wGroupIndex*　　group 编号		
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineU(double *dfU*, WORD *wGroupIndex*)

Description	U 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfU*　　目标点的坐标值 *wGroupIndex*　　group 编号		
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineV(double *dfV*, WORD *wGroupIndex*,)

Description	V 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfV* 目标点的坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineW(double *dfW*, WORD *wGroupIndex*,)

Description	W 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfW* 目标点的坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_LineA(double *dfA*, WORD *wGroupIndex*,)

Description	A 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfA* 目标点的坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

nt MCC_LineB(double *dfB*, WORD *wGroupIndex*,)

Description	B 轴从目前位置直线运动到指定的目标点		
指令类型	缓存区指令	章节页码	
Parameters	*dfB* 目标点的坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ArcXYZ(double *dfRX0*,
double *dfRX1*,
double *dfRX2*,
double *dfX0*,
double *dfX1*,
double *dfX2*,
WORD *wGroupIndex*)

Description	在 X-Y-Z 空间上，以圆弧运动的方式从当前位置经过指定的参考点到达目标点。		
指令类型	缓存区指令	章节页码	
Parameters	*dfRX0 ~dfRX2* 参考点的 X-Y-Z 轴坐标值 *dfDX0~dfDX2* 目标点的 X-Y-Z 轴坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ArcXYZ_Aux(double *dfRX0*,
double *dfRX1*,
double *dfRX2*,
double *dfX0*,
double *dfX1*,
double *dfX2*,
double *dfX3*,
double *dfX4*,
double *dfX5*,
double *dfX6*,
double *dfX7*,
WORD *wGroupIndex*)

Description	在 X-Y-Z 空间上，以圆弧运动的方式从当前位置经过指定的参考点到目标点，且 U、V、W、A、B 轴以同动方式进行直线运动。		
指令类型	缓存区指令	章节页码	
Parameters	*dfRX0 ~dfRX2* 参考点的 X-Y-Z 轴坐标值 *dfX0~dfX2* 目标点的 X-Y-Z 轴坐标值 *dfX3~dfX7* 目标点的 U-V-W-A-B 轴坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ArcXY(double *dfRX0*,
double *dfRX1*,
double *dfX0*,
double *dfX1*,
WORD *wGroupIndex*)

Description	在 X-Y 平面上，以圆弧运动的方式从当前位置经过指定的参考点到达目标点。		
指令类型	缓存区指令	章节页码	
Parameters	*dfRX0*, *dfRX1* 参考点的 X-Y 轴坐标值 *dfX0*, *dfX1* 目标点的 X-Y 轴坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ArcYZ(double *dfRX1*,
double *dfRX2*,
double *dfX1*,
double *dfX2*,
WORD *wGroupIndex*)

Description	在 Y-Z 平面上，以圆弧运动的方式从目前位置经过指定的参考点到目标点。		
指令类型	缓存区指令	章节页码	
Parameters	*dfRX1*, *dfRX2* 参考点的 Y-Z 轴坐标值 *dfX1*, *dfX2* 目标点的 Y-Z 轴坐标值 *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ArcZX(double *dfRX2*,
double *dfRX0*,
double *dfX2*,
double *dfX0*,
WORD *wGroupIndex*)

Description	在 Z-X 平面上，以圆弧运动的方式从目前位置经过指定的参考点到目标点。
指令类型	缓存区指令
Parameters	*dfRX2*, *dfRX0* 参考点的 Z-X 轴坐标值 *dfX2*, *dfX0* 目标点的 Z-X 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcXY_Aux(double *dfRX0*,
double *dfRX1*,
double *dfX0*,
double *dfX1*,
double *dfX3*,
double *dfX4*,
double *dfX5*,
double *dfX6*,
double *dfX7*,
WORD *wGroupIndex*)

Description	在 X-Y 平面上，以圆弧运动的方式从目前位置经过指定的参考点到目标点，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*dfRX0*, *dfRX1* 参考点的 X-Y 轴坐标值 *dfX0*, *dfX1* 目标点的 X-Y 轴坐标值 *dfX3~dfX7* 目标点的 U-V-W-A-B 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcYZ_Aux(double *dfRX1*,
double *dfRX2*,
double *dfX1*,
double *dfX2*,
double *dfX3*,
double *dfX4*,
double *dfX5*,
double *dfX6*,
double *dfX7*,
WORD *wGroupIndex*)

Description	在 Y-Z 平面上，以圆弧运动的方式从目前位置经过指定的参考点到目标点，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*dfRX1*, *dfRX2* 参考点的 Y-Z 轴坐标值 *dfX1*, *dfX2* 目标点的 Y-Z 轴坐标值 *dfX3~dfX7* 目标点的 U-V-W-A-B 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcZX_Aux(double *dfRX2*,
double *dfRX0*,
double *dfX2*,
double *dfX0*,
double *dfX3*,
double *dfX4*,
double *dfX5*,
double *dfX6*,
double *dfX7*,
WORD *wGroupIndex*)

Description	在 Z-X 平面上，以圆弧运动的方式从目前位置经过指定的参考点到目标点，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*dfRX2*, *dfRX0*　参考点的 Z-X 轴坐标值 *dfX2*, *dfX0*　目标点的 Z-X 轴坐标值 *dfX3~dfX7*　目标点的 U-V-W-A-B 轴坐标值 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcThetaXY(double *dfCX*,
double *dfXY*,
double *dfTheta*,
WORD *wGroupIndex*)

Description	在 X-Y 平面上，以指定的圆心与移动角度进行圆弧运动。移动角度为负则进行顺时针运动，移动角度为正则进行逆时针运动。
指令类型	缓存区指令
Parameters	*dfCX*, *dfCY*　指定的圆心坐标值 *dfTheta*　移动角度 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcThetaYZ(double *dfCY*,
double *dfCZ*,
double *dfTheta*,
WORD *wGroupIndex*)

Description	在 Y-Z 平面上，以指定的圆心与移动角度进行圆弧运动。移动角度为负则进行顺时针运动，移动角度为正则进行逆时针运动。
指令类型	缓存区指令
Parameters	*dfCY*, *dfCZ*　指定的圆心坐标值 *dfTheta*　移动角度 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ArcThetaZX(double *dfCZ*,
double *dfCX*,
doubled *dfTheta*,
WORD *wGroupIndex*)

Description	在 Z-X 平面上，以指定的圆心与移动角度进行圆弧运动。移动角度为负则进行顺时针运动，移动角度为正则进行逆时针运动。
指令类型	缓存区指令
Parameters	*dfX2*, *dfX0* 指定的圆心坐标值 *dfTheta* 移动角度 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleXY(double *dfCX*,
double *dfCY*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	在 X-Y 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动。
指令类型	缓存区指令 章节页码
Parameters	*dfCX*, *dfCY* 圆心的 X-Y 轴坐标值 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleYZ(double *dfCY*,
double *dfCZ*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	在 Y-Z 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动。
指令类型	缓存区指令 章节页码
Parameters	*dfCY*, *dfCZ* 圆心的 Y-Z 轴坐标值 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleZX(double *dfCZ*, double *dfCX*, BYTE *byCirDir*, WORD *wGroupIndex*)

Description	在 Z-X 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动
指令类型	缓存区指令
Parameters	*dfCZ*, *dfCX* 圆心的 Z-X 轴坐标值 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleXY_Aux(double *dfCX*,
double *dfCY*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	在 X-Y 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*fCX*, *dfCY* 圆心的 X-Y 轴坐标值 *dfU*, *dfV*, *dfW*, *dfA*, *dfB* 目标点的 U-V-W-A-B 轴坐标值 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleYZ_Aux(double *dfCY*,
double *dfCZ*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	在 Y-Z 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*dfCY*, *dfCZ* 圆心的 Y-Z 轴坐标值 *dfU*, *dfV*, *dfW*, *dfA*, *dfB* 目标点的 U-V-W-A-B 轴坐标值 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码

	小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CircleZX_Aux(double *dfCZ*,
double *dfCX*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	在 Z-X 平面上，由目前位置执行一个以指定点为圆心坐标的全圆轨迹运动，且 U、V、W、A、B 轴以同动方式进行直线运动。
指令类型	缓存区指令
Parameters	*dfCZ, dfCX*　　圆心的 Z-X 轴坐标值 *dfU*,　*dfV*,*dfW*,　*dfA*, *dfB* 目标点的 U-V-W-A-B 轴坐标值 *byCirDir*　　运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex*　group 编号
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_HelicalXY_Z(double *dfCX*,
double *dfCY*,
double *dfZ*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 X-Y 平面进行圆周运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 X-Y 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 Z 轴坐标值，		
指令类型	缓存区指令	章节页码	
Parameters	*dfCX, dfCY*　圆周运动圆心的 X-Y 轴坐标值 *dfZ*　　目标点的 Z 轴坐标值 *dfPitch*　　在 X-Y 平面进行一个整圆运动后，Z 轴所移动的距离，此值必须大于 0。 *byCirDir*　　运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex*　group 编号		
Return Value	大于或等于 0　　MCCL 给予此运动命令的编码 小于 0　　失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_HelicalYZ_X(double *dfCY*,
double *dfCZ*,
double *dfX*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 Y-Z 平面进行圆周运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 Y-Z 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 X 轴坐标值，
指令类型	缓存区指令
Parameters	*dfCY, dfCZ* 圆周运动圆心的 Y-Z 轴坐标值 *dfX* 目标点的 X 轴坐标值 *dfPitch* 在 Y-Z 平面进行一个整圆运动后，X 轴所移动的距离，此值必须大于 0。 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_HelicalZX_Y(double *dfCZ*,
double *dfCX*,
double *dfY*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 Z-X 平面进行圆周运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 Z-X 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 Y 轴坐标值，
指令类型	缓存区指令
Parameters	*dfCZ, dfCX* 圆周运动圆心的 Z-X 轴坐标值 *dfY* 目标点的 Y 轴坐标值 *dfPitch* 在 Z-X 平面进行一个整圆运动后，Y 轴所移动的距离，此值必须大于 0。 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_HelicalXY_Z_Aux(double *dfCX*,
double *dfCY*,
double *dfZ*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 X-Y 平面进行圆周运动，且 U、V、W、A、B 轴以同动方式进行直线运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 X-Y 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 Z 轴坐标值，
指令类型	缓存区指令
Parameters	*dfCX*, *dfCY* 圆周运动圆心的 X-Y 轴坐标值 *dfZ* 目标点的 Z 轴坐标值 *dfU*, *dfV*, *dfW*, *dfA*, *dfB* 目标点的 U-V-W-A-B 轴坐标值 *dfPitch* 在 X-Y 平面进行一个整圆运动后，Z 轴所移动的距离，此值必须大于 0。 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_HelicalYZ_X_Aux(double *dfCY*,
double *dfCZ*,
double *dfX*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 Y-Z 平面进行圆周运动，且 U、V、W、A、B 轴以同动方式进行直线运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 Y-Z 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 X 轴坐标值，
指令类型	缓存区指令
Parameters	*dfCY*, *dfCZ* 圆周运动圆心的 Y-Z 轴坐标值 *dfX* 目标点的 X 轴坐标值 *dfU*, *dfV*, *dfW*, *dfA*, *dfB* 目标点的 U-V-W-A-B 轴坐标值 *dfPitch* 在 Y-Z 平面进行一个整圆运动后，X 轴所移动的距离，此

	值必须大于 0。 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_HelicalZX_Y_Aux(double *dfCZ*,
double *dfCX*,
double *dfY*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
double *dfPitch*,
BYTE *byCirDir*,
WORD *wGroupIndex*)

Description	由当前位置执行螺线运动，在 Z-X 平面进行圆周运动，且 U、V、W、A、B 轴以同动方式进行直线运动，可利用 MCC_SetFeedSpeed()设定此圆周运动的速度。使用此函数必须指定 Z-X 平面上圆周运动的圆心坐标值，圆周半径由目前位置与圆心坐标值决定，并需指定目标点的 Y 轴坐标值，
指令类型	缓存区指令
Parameters	*dfCZ, dfCX* 圆周运动圆心的 Z-X 轴坐标值 *dfY* 目标点的 Y 轴坐标值 *dfU, dfV, dfW, dfA, dfB* 目标点的 U-V-W-A-B 轴坐标值 *dfPitch* 在 Z-X 平面进行一个整圆运动后，Y 轴所移动的距离，此值必须大于 0。 *byCirDir* 运动方向，0 为顺时针运动，1 为逆时针运动。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

点到点运动

double MCC_SetPtPSpeed(double *dfRatio*, WORD *wGroupInde*)

Description	设定点对点运动时的速度比例，点对点运动时各轴的速度(UU/s)等于（马达最大转速/60）× pitch 数 / 齿轮减速比) × (速度比例 / 100) 其中马达最大转速(*wRPM*)、pitch 数(*dfPitch*)、齿轮减速比(*dfGearRatio*)定义在机构参数中。 但点对点运动实际操作时的进给速度需参考是否曾使用 MCC_OverrideSpeed()设定运动进给速度倍率	
指令类型	缓存区指令	章节页码
Parameters	*dfRatio* 速度比例，其值必须大于 0 且小于等于 100 *wGroupIndex* group 编号	

Return Value	实际设定的速度比例

double MCC_GetPtPSpeed(WORD *wGroupIndex*)

Description	读取点对点运动时使用的速度比例
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	大于或等于 0 点对点运动时使用的速度比例 小于 0 失败，传回值意义请参考 **2.1 函数返回值**

int MCC_PtP(double *dfX*,
double *dfY*,
double *dfZ*,
double *dfU*,
double *dfV*,
double *dfW*,
double *dfA*,
double *dfB*,
WORD *wGroupIndex*,
DWORD *dwAxisMask*)

Description	从当前位置以设定的进给速度比例点对点运动至指定的目标点。		
指令类型	缓存区指令	章节页码	
Parameters	*dfX, dfY, dfZ* 目标点的 X-Y-Z 轴坐标值 *dfU, dfV, dfW* 目标点的 U-V-W 轴坐标值 *dfA, dfB* 目标点的 A-B 轴坐标值 *wGroupIndex* group 编号 *dwAxisMask* 指定欲发生作用的轴，指定参数可为 MCC_AXIS_X X 轴；MCC_AXIS_Y Y 轴 MCC_AXIS_Z Z 轴; MCC_AXIS_U U 轴 MCC_AXIS_V V 轴；MCC_AXIS_W W 轴 MCC_AXIS_A A 轴; MCC_AXIS_B B 轴 MCC_AXIS_ALL 全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例： (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_PtPX(double *dfX*, WORD *wGroupIndex*)

Description	X 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfX* 目标点的 X 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPY(double *dfY*, WORD *wGroupIndex*)

Description	Y 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfY* 目标点的 Y 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPZ(double *dfZ*, WORD *wGroupIndex*)

Description	Z 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfZ* 目标点的 Z 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPU(double *dfU*, WORD *wGroupIndex*)

Description	U 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfU* 目标点的 U 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPV(double *dfV*, WORD *wGroupIndex*)

Description	V 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfV* 目标点的 V 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPW(double *dfW*, WORD *wGroupIndex*)

Description	W 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*fW* 目标点的 W 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPA(double *dfA*, WORD *wGroupIndex*)

Description	A 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfA* 目标点的 A 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_PtPB(double *dfB*, WORD *wGroupIndex*)

Description	B 轴从当前位置以设定的进给速度比例点对点运动至指定的目标点。
指令类型	缓存区指令
Parameters	*dfB* 目标点的 B 轴坐标值 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetPtPAccType(char *cType0*,
char *cType1*,
char *cType2*,
char *cType3*,
char *cType4*,
char *cType5*,
char *cType6*,
char *cType7*,
WORD *wGroupIndex*)

Description	设定点对点运动时使用的加速形型式，各轴使用独立的加速型式
指令类型	缓存区指令
Parameters	cType0 ~ cType7 各轴的加速型式，可设定为： 'T' 使用梯形加速曲线 'S' 使用 S 形加速曲线 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetPtPAccType(char **pcType0*,
char **pcType1*,
char **pcType2*,
char **pcType3*,
char **pcType4*,
char **pcType5*,
char **pcType6*,
char **pcType7*,
WORD *wGroupIndex*)

Description	读取点对点运动时使用的加速型式
指令类型	缓存区指令
Parameters	*pcType0 ~ *pcType7 各轴的加速型式，0 表示使用梯形加速曲线，1 表示使用 S 形加速曲线 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetPtPDecType(char *cType0*, char *cType1*, char *cType2*, char *cType3*, char *cType4*, char *cType5*, char *cType6*, char *cType7*, WORD *wGroupIndex*)

Description	设定点对点运动时使用的减速型式，各轴使用独立的减速型式
指令类型	缓存区指令
Parameters	cType0 ~ cType7 各轴的减速型式，可设定为： 'T' 使用梯形减速曲线 'S' 使用 S 形减速曲线 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetPtPDecType(char **pcType0*, char **pcType1*, charpcType2*, char **pcType3*, char **pcType4*, char**pcType5*, har **pcType6*, char **pcType7*, WORD *wGroupIndex*)**

Description	读取在进行点对点运动时使用的减速型式
指令类型	缓存区指令
Parameters	*pcType0 ~ *pcType7 各轴的减速型式，0 表示使用梯形减速曲线，1 表示使用 S 形减速曲线。 *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetPtPAccTime(double *dfTime0*,
double *dfTime1*,
double *dfTime2*,
double *dfTime3*,
double *dfTime4*,
double *dfTime5*,
double *dfTime6*,
double *dfTime7*,
WORD *wGroupIndex*)

Description	设定在进行点对点运动时加速到稳定速度所需的时间，各轴使用各自独立的加速时间
指令类型	缓冲区指令
Parameters	dfTime0 ~ dfTime7 各轴的加速时间，必须大于 0，单位为 ms。 *wGroupIndex* group 编
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetPtPAccTime(double **pdfTime0*, double **pdfTime1*, double **pdfTime2*, double **pdfTime3*, double **pdfTime4*, double **pdfTime5*, double **pdfTime6*, double **pdfTime7*, WORD *wGroupIndex*)

Description	设定在进行点对点运动时加速到稳定速度所需的时间，各轴使用各自独立的加速时间
指令类型	缓存区指令
Parameters	pdfTime0 ~ pdfTime7　各轴的加速时间，必须大于 0，单位为 ms。 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetPtPDecTime(double *dfTime0*,
double *dfTime1*,
double *dfTime2*,
double *dfTime3*,
double *dfTime4*,
double *dfTime5*,
double *dfTime6*,
double *dfTime7*,
WORD *wGroupIndex*)

Description	设定点对点运动时由设定速度减速到停止运动所需的时间，各轴使用独立的减速时间
指令类型	缓冲区指令
Parameters	dfTime0 ~ dfTime7　各轴的减速时间，单位为 ms。 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetPtPDecTime(double **pdfTime0*, double **pdfTime1*, double **pdfTime2*, double **pdfTime3*, double **pdfTime4*, double **pdfTime5*, double **pdfTime6*, double **pdfTime7*, WORD *wGroupIndex*)

Description	读取点对点运动时由设定速度减速到停止运动所需的时间，各轴使用独立的减速时间
指令类型	缓冲区指令
Parameters	pdfTime0 ~ pdfTime7　各轴的减速时间，单位为 ms。 *wGroupIndex*　group 编号
Return Value	大于或等于 0　MCCL 给予此运动命令的编码 小于 0　失败，传回值的意义请参考 **2.1 函数返回值**

任意曲线运动

int MCC_CustomMotion(CUSTOM_START_MOTION *pfnStartMotion*,
CUSTOM_INTERPOLATION *pfnInterpolation*,
CUSTOM_CLEANUP *pfnCleanUp*,
void **pvBuffer*,
WORD *wGroupIndex*,
DWORD *dwAxisMask*)

Description	使用任意曲线运动的方式，从目前位置以设定的进给速度，根据用户给定的插值点回传值进行轨迹运动。使用此函数前使用者必须编写三个回传(CallBack)函数，提供给 MCCL 运动控制函数库在运动过程中呼叫。
指令类型	缓冲区指令
Parameters	*pfnStartMotion* 自定任意曲线起始运动时的回调函数指标，当 MCCL 起始一笔任意曲线运动指令时，首先会呼叫此函数，此函数必须回传即将执行的该笔任意曲线命令的总位移行程 *pfnInterpolation* 自定任意曲线插值运动过程的回调函数指标，当 MCCL 进行一笔任意曲线插值运动时，过程会不断的呼叫此函数，此函数必须回传运动过程的下一个插值点位移量(Offset) *pfnCleanUp* 自定任意曲线结束时的回调函数指标，当 MCCL 完成一笔任意曲线运动时，结束时会呼叫此函数，此函数必须负责将任意曲线所使用的系统资源释放，如内存释放等 **pvBuffer* 指向内存起始位置，用户可将计算任意曲线所需的信息封装到一块内存区块中，并将此内存的起始位置传给 MCCL，当 MCCL 呼叫这些回调函数(CallBack)时，同时也会将内存位置传给回调函数使用。 *wGroupIndex* group 编号 *dwAxisMask* 指定欲发生作用的轴，指定参数可为: MCC_AXIS_X X 轴; MCC_AXIS_Y Y 轴 MCC_AXIS_Z Z 轴; MCC_AXIS_U U 轴 MCC_AXIS_V V 轴; MCC_AXIS_W W 轴 MCC_AXIS_A A 轴; MCC_AXIS_B B 轴 MCC_AXIS_ALL 全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例: (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_CustomMotionEx(
CUSTOM_START_MOTION *pfnStartMotion*,
CUSTOM_INTERPOLATION *pfnInterpolation*,
CUSTOM_CLEANUP *pfnCleanUp*,
CUSTOM_BLENDING_START*pfnBlendingStart*,
CUSTOM_BLENDING *pfnBlending*,
CUSTOM_BLENDING_END *pfnBlendingEnd*,
void **pvBuffer*,
WORD *wGroupIndex*,
DWORD *dwAxisMask*)

Description	使用任意曲线运动的方式，从目前位置以设定的进给速度，根据用户给定的插值点回传值进行轨迹运动。使用此函数前使用者必须编写六个回传(CallBack)函数，提供给 MCCL 运动控制函数库在运动过程中呼叫。
指令类型	缓冲区指令
Parameters	*pfnStartMotion* 自定任意曲线起始运动时的回调函数指标，当 MCCL 起始一笔任意曲线运动指令时，首先会呼叫此函数，此函数必须回传即将执行的该笔任意曲线命令的总位移行程。 *pfnInterpolation* 自定任意曲线插值运动过程的回调函数指标，当 MCCL 进行一笔任意曲线插值运动时，过程会不断的呼叫此函数，此函数必须回传运动过程的下一个插值点位移量(Offset） *pfnCleanUp* 自定任意曲线结束时的回调函数指标，当 MCCL 完成一笔任意曲线运动时，结束时会呼叫此函数，此函数必须负责将任意曲线所使用的系统资源释放，如内存释放等. *pfnBlendingStart* 自定任意曲线进行连续路径的回调函数指针，当 MCCL 起始一笔任意曲线连续路径规划时，会先呼叫此函数，使用者可在此函数内将连续路径所需的信息先行规划。 *pfnBlending* 自定任意曲线在进行连续路径插值运动过程的回调函数指标，当 MCCL 进行一笔任意曲线连续路径时，过程会不断的呼叫此函数，此函数必须回传运动过程的下一个插值点位移量(Offset)。 *pfnBlendingEnd* 自定任意曲线在进行连续路径结束时的回调函数指标，当 MCCL 结束一笔任意曲线连续路径运动时，会呼叫此函数，使用者可在此函数中将所使用的系统资源释放回。 **pvBuffer* 指向内存起始位置，用户可将计算任意曲线所需的信息封装到一块内存区块中，并将此内存的起始位置传给 MCCL，当 MCCL 呼叫这些回调函数(CallBack)时，同时也会将内存位置传给回调函数使用。 *wGroupIndex* group 编号

	dwAxisMask 指定欲发生作用的轴，指定参数可为: MCC_AXIS_X X 轴; MCC_AXIS_Y Y 轴 MCC_AXIS_Z Z 轴; MCC_AXIS_U U 轴 MCC_AXIS_V V 轴; MCC_AXIS_W W 轴 MCC_AXIS_A A 轴; MCC_AXIS_B B 轴 MCC_AXIS_ALL 全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例： (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

JOG 运动

int MCC_JogPulse(int *nPulse*, char *cAxis*, WORD *wGroupIndex*)

Description	微动(脉冲运动)。在其他运动命令皆已执行完成后(此时呼叫 MCC_GetMotionStatus()所获得的返回值应为 GMS_STOP)，依照指定的位移量(pulse 数)及方向运动指定轴。 此函数为手动程序的细调模式且需处在运动停止状态时呼叫才有效。脉冲运动并没有包含加减速的动作，因此给定的位移量不宜过大，避免机台过度震动
指令类型	缓冲区指令
Parameters	*nPulse* 位移量，单位为 pulse，可给定范围为-2048 ~ 2048 *cAxis* 要求进行脉冲运动的运动轴编号(0 ~ 7 代表 X ~ B 轴) *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_JogSpace(double *dfOffset*, int *nRatio*, char *cAxis*, WORD *wGroupIndex*)

Description	时动(单步运动)。在其他运动命令皆已执行完成后(此时呼叫 MCC_GetMotionStatus()所获得的返回值应为 GMS_STOP)，依照指定的位移量(增量)及速度比例(与点对点运动的速度比例意义相同)运动指定轴。
指令类型	缓冲区指令
Parameters	*dfOffset* 位移量，单位为 UU *nRatio* 速度比例，其值必须大于 0 且小于等于 100 *cAxis* 要求进行单步运动的运动轴之编号(0 ~ 7 代表 X ~ B 轴) *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_JogConti(int *nDir*, int *nRatio*, char *cAxis*, WORD *wGroupIndex*)

Description	连续时动。在其他运动命令皆已执行完成后(此时呼叫 MCC_GetMotionStatus()所获得的返回值应为 GMS_STOP)，依照指定的方向及速度比例(与点对点运动的速度比例意义相同)运动指定轴，并移动到有效工作区间的边界才停止(机构参数定义了有效工作区间的范围)。
指令类型	缓冲区指令
Parameters	*nDir* 连续时动的方向，可设定为： 1 往正方向运动; -1 往负方向运动 *nRatio* 速度比例，其值必须大于 0 且小于等于 100 *cAxis* 要求进行单步运动的运动轴之编号(0～7 代表 X～B 轴) *wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

运动状态检测

int MCC_GetMotionStatus(WORD *wGroupIndex*)

Description	检测系统目前的运动状态
指令类型	立即执行指令 章节页码
Parameters	*wGroupIndex* group 编号
Return Value	0 GMS_RUNNING 处于运动状态，尚有运动命令未执行完成 1 GMS_STOP 处于停止状态，已无库存运动命令 2 GMS_HOLD 处于暂停状态(因使用者呼叫 MCC_HoldMotion) 3 GMS_DELAYING处于延迟状态(因使用者呼叫 MCC_DelayMotion) 4 GMS_BLOCKHOLD 5 GMS_MPGING 其他 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetCurCommand(COMMAND_INFO* *pstCurCmdInfo*, WORD *wGroupIndex*)

Description	读取执行中运动命令的相关信息，包括运动命令类型、运动命令编码、要求的进给速度与目标点位置等。
指令类型	立即执行指令 章节页码
Parameters	*pstCurCmdInfo* 指向一 COMMAND_INFO 结构，用来存放执行中之运动命令内容，定义如下： typedef struct _COMMAND_INFO{ int nType; int nCommandIndex; double dfFeedSpeed; double dfPos[MAX_AXIS_NUM]; } COMMAND_INFO; *nType*：运动命令类型: 0 点对点运动

	1 直线运动; 2 顺时针圆弧、圆运动 3 逆时针圆弧、圆运动 4 顺时针螺线运动 5 逆时针螺线运动 6 运动延迟命令 7 开启平滑运动 8 关闭平滑运动 9 开启定位确认 10 关闭定位确认 *nCommandIndex*：运动命令编码 *dfFeedSpeed*：一般运动 规划的进给速度 点对点运动 规划的速度比例 运动延迟 目前剩余的延迟时间(单位：ms) *dfPos[]*： 目标点的绝对位置坐标 *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetCommandCount(int *pnCmdCount, WORD wGroupIndex)

Description	读取运动命令缓冲区中尚未执行的运动命令库存数目。
指令类型	立即执行指令
Parameters	*pnCmdCount* 指向一 int 值，用来存放运动命令库存数目 *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_ResetCommandIndex(WORD wGroupIndex)

Description	使运动命令编码值归零。运动命令编码相当于 MCCL 给每一笔运动命令的识别码，此函数可将缓冲区运动命令编码值从 0 开始计数。
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetCurPulseStockCount(WORD *pwStockCount,
WORD wChannel,
WORD wCardIndex = 0)

Description	读取目前硬件上的轨迹插补命令库存数目。在运动过程中，读取的命令库存数目不应小于 60，这样才能保有稳定的运动控制性能；如不能达到此要求，请增加插值时间(重新呼叫 MCC_InitSystem)。		
指令类型	立即执行指令	章节页码	
Parameters	*pwStockCount* 指向一 WORD 值，用来存放 Pulse 命令库存数目 *wChannel* 运动控制卡的输出 Channel(0 ~ 7)		

Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetMaxPulseStockNum(WORD *nMaxStockNum*,)

Description	设定硬件 FIFO 的使用个数。硬件 FIFO 的使用个数应该考虑操作系统的实时，设定越少的使用个数，操作系统要具备的实时性能越强，设定越多的使用个数，操作系统对实时性能的容忍性越大，对运动控制性能的稳定性越高；如不能达到运动控制性能的稳定性的要求时，请设定较多的使用个数或增加插值时间。
指令类型	立即执行指令
Parameters	*wMaxStockNum* FIFO 最大使用个数
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetMaxPulseStockNum(WORD **wMaxStockNum*,)

Description	读取硬件 FIFO 的最大使用个数。
指令类型	立即执行指令
Parameters	*wMaxStockNum* FIFO 最大使用个数
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetErrorCode(WORD *wGroupIndex*)

Description	读取当前错误记录，用来检查系统工作时是否发生错误。 系统工作中应随时(例如每 100ms)呼叫此函数确认系统目前工作状态，若发现错误记录产生，则须采取相对应的错误恢复处理(Error Recovery)。		
指令类型	立即执行指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	0 无错误 其它 错误代码(请参考 **IV.错误信息代码**)		

int MCC_ClearError(WORD *wGroupIndex*)

Description	在系统发生错误后，若已排除相应错误，必须使用此函数清除系统内的错误记录，否则系统仍无法正常运作。
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

定位控制

int MCC_SetCompParam(SYS_COMP_PARAM *_pstCompParam_, WORD _wChannel_, WORD _wCardIndex = 0_)

Description	设定齿轮齿隙、间隙补偿参数，用户可先设定补偿参数的内容，再利用此函数将补偿参数传入，最后呼叫 MCC_UpdatetCompParam()，补偿参数的内容必须涵盖机台全部的工作行程，以避免产生不正常的动作。
指令类型	立即执行指令　　章节页码
Parameters	*pstCompParam* 指向一 SYS_COMP_PARAM 结构，用来描述补偿参数 *wChannel*　　运动控制卡的输出 Channel(0 ~ 7)
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_UpdateCompParam()

Description	更新齿轮齿隙、间隙补偿参数。在调用过函数 MCC_SetCompParam () 后，需执行此函数系统才会更新设定值。
指令类型	立即执行指令
Parameters	
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetMaxPulseSpeed(int _nPulse0_,
int _nPulse1_,
int _nPulse2_,
int _nPulse3_,
int _nPulse4_,
int _nPulse5_,
int _nPulse6_,
int _nPulse7_,
WORD _wCardIndex = 0_)

Description	设定各轴最大 pulse 速度的上限。最大 pulse 速度的上限用来限制一插值单位时间内，各轴能送出的最大 pulse 数，避免机台的速度超出工作范围。
指令类型	立即执行指令　　章节页码
Parameters	*nPulse0~nPulse7*　各轴最大 pulse 速度的上限。 设定范围为 1~32767，适当值需视机台特性与插值时间而定。
Return Value	0　　成功 非零　　失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetMaxPulseSpeed(int *pnPulse0, int *pnPulse1, int *pnPulse2, int *pnPulse3, int *pnPulse4, int *pnPulse5, int *pnPulse6, int *pnPulse7, WORD wCardIndex = 0)

Description	读取各轴最大 pulse 速度的上限
指令类型	立即执行指令
Parameters	*pnPulse0~pnPulse7* 指向一 int 值，用来存放各轴最大 pulse 速度的上限
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetMaxPulseAcc(int *nPulse0*,
int *nPulse1*,
int *nPulse2*,
int *nPulse3*,
int *nPulse4*,
int *nPulse5*,
int *nPulse6*,
int *nPulse7*,
WORD *wCardIndex = 0*)

Description	设定各轴最大 pulse 加速度的上限。最大 pulse 加速度的上限用来限制任连续两插值时间中，各轴所送出 pulse 数最大的差量，可避免机台的加(减)速度超出工作范围。。
指令类型	立即执行指令
Parameters	*nPulse0~nPulse7* 各轴最大 pulse 加速度的上限。 设定范围为 1~32767，适当值需视机台特性与插值时间而定。
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetMaxPulseAcc(int* *pnPulse0*, int* *pnPulse1*, int* *pnPulse2*, int* *pnPulse3*, int* *pnPulse4*, int* *pnPulse5*, int* *pnPulse6*, int* *pnPulse7*, WORD *wCardIndex = 0*)

Description	读取各轴最大 pulse 加速度的上限
指令类型	立即执行指令
Parameters	*pnPulse0~pnPulse7* 指向一 int 值，用来存放各轴最大 pulse 加速度的上限
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetInPosMode(WORD *wMode*, WORD *wGroupIndex*)

Description	设定到位确认模式。		
指令类型	立即执行指令	章节页码	
Parameters	*wMode* 定位确认模式 *wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_SetInPosMaxCheckTime(WORD *wMaxCheckTimet*,
WORD *wGroupIndex*)

Description	设定到位确认最大检查时间。		
指令类型	立即执行指令	章节页码	
Parameters	*wMaxCheckTime* 到位确认最大检查时间，单位 ms *wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ SetInPosSettleTime(WORD *wSettleTime*,
WORD *wGroupIndex*)

Description	设定到位确认持续时间。		
指令类型	立即执行指令	章节页码	
Parameters	*wSettleTime* 到位确认持续时间，单位 ms。 *wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_EnableInPos(WORD *wGroupIndex*)

Description	开启到位确认功能。		
指令类型	缓冲区指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_DisableInPos(WORD *wGroupIndex*)

Description	关闭定位确认功能。
指令类型	缓冲区指令
Parameters	*wGroupIndex* group 编号
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetInPosToleranceEx(double *dfTolerance0*,
double *dfTolerance1*,
double *dfTolerance2*,
double *dfTolerance3*,
double *dfTolerance4*,
double *dfTolerance5*,
double *dfTolerance6*,
double *dfTolerance7*,
WORD *wGroupIndex*)

Description	设定各轴到位误差容许范围。
指令类型	立即执行指令 章节页码
Parameters	*dfTolerance0 ~ dfTolerance7* 各轴定位误差容许范围，单位为 UU *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetInPosToleranceEx(double* *pdfTolerance0*, double* *pdfTolerance1*, double* *pdfTolerance2*, double* *pdfTolerance3*, double* *pdfTolerance4*, double* *pdfTolerance5*, double* *pdfTolerance6*, double* *pdfTolerance7*, WORD *wGroupIndex*)

Description	读取各轴到位误差容许范围
指令类型	立即执行指令
Parameters	*erance0~pdfTolerance7* 指向一 double 值，用来存放各轴定位误差容许范围，单位为 UU *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetInPosStatus(BYTE* *pbyInPos0*, BYTE* *pbyInPos1*, BYTE* *pbyInPos2*, BYTE* *pbyInPos3*, BYTE* *pbyInPos4*, BYTE* *pbyInPos5*, BYTE* *pbyInPos6*, BYTE* *pbyInPos7*, WORD *wGroupIndex*)

Description	读取各轴到位确认状态
指令类型	立即执行指令
Parameters	*pbyInPos0~pbyInPos7* 指向一 BYTE 值，用来存放各轴的定位确认状态，0xff(255)表示已满足到位确认条件，0 则表示尚未满足定位确认条件 *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_EnableTrackError(WORD *wGroupIndex*, DWORD *dwAxisMask*)

Description	开启跟随误差侦测功能。
指令类型	立即执行指令 章节页码
Parameters	*wGroupIndex* group 编号 *dwAxisMask* 指定欲发生作用的轴，指定参数可为： MCC_AXIS_X X 轴；MCC_AXIS_Y Y 轴 MCC_AXIS_Z Z 轴；MCC_AXIS_U U 轴 MCC_AXIS_V V 轴；MCC_AXIS_W W 轴 MCC_AXIS_A A 轴；MCC_AXIS_B B 轴 MCC_AXIS_ALL 全部运动轴 以上参数可自由组合，以作用在 X、Z 与 V 轴上为例： (MCC_AXIS_X | MCC_AXIS_Z | MCC_AXIS_V)
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

nt MCC_DisableTrackError(WORD wGroupIndex)

Description	关闭检查轨迹跟随误差功能。
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetTrackErrorLimit(double *dfLimit*, char *cAxis*, WORD *wGroupIndex*)

Description	设定跟随误差容许范围。
指令类型	立即执行指令
Parameters	*dfLimit* 跟随误差容许范围，单位为 UU

	cAxis 运动轴编号(0 ~ 7 代表 X ~ B 轴) *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_GetTrackErrorLimit(double **pdfLimit*, char *cAxis*, WORD *wGroupIndex*)

Description	读取跟随误差容许范围。
指令类型	立即执行指令
Parameters	*pdfLimit* 指向一 double 值，用来存放跟随误差容许范围，单位为 UU *cAxis* 运动轴编号(0 ~ 7 代表 X ~ B 轴) *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

进阶轨迹规划

int MCC_HoldMotion(WORD *wGroupIndex*)

Description	暂时停止运动，在运动进行中使用此函数才有意义。呼叫此函数后将减速至停止运动，在减速至停止前，若呼叫 MCC_GetMotionStatus()所获得的返回值仍为 GMS_RUNNING，必须等到运动完全停止后，才会得到 GMS_HOLD 的返回值。		
指令类型	立即执行指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_ContiMotion(WORD *wGroupIndex*)

Description	继续执行未完成的运动命令，但必须在运动暂停状态使用此函数才意义。
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_AbortMotion(WORD *wGroupIndex*)

Description	紧急停止，并放弃所有未执行的运动命令。注意，***在使用此函数之后，必须等到系统进入 GMS_STOP 状态，才能下达后续运动命令***，否则将得到 ABORT_NOT_FINISH_ERR(-15)的返回值。		
指令类型	立即执行指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_AbortMotionEx(double *dfDecTime*, WORD *wGroupIndex*)

Description	以设定的减速时间，减速至停止并抛弃后续全部运动命令。呼叫此函数后将减速至停止运动，在减速至停止前，若呼叫 MCC_GetMotionStatus()所获得的返回值仍为 GMS_RUNNING，必须等到运动完全停止后，才会得到 GMS_STOP 的返回值。注意，***在使用此函数之后，必须等到系统进入 GMS_STOP 状态，才能下达后续运动命令***，否则将得到 ABORT_NOT_FINISH_ERR(-15)的回传值。
指令类型	立即执行指令
Parameters	*dfDecTime* 要求减速的时间，单位：ms *wGroupIndex* group 编号
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_EnableBlend(WORD *wGroupIndex*)

Description	开启平滑运动功能。呼叫此函数后，以连续路径方式进行轨迹规划。		
指令类型	缓冲区指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_DisableBlend(WORD *wGroupIndex*)

Description	关闭连续运动功能。成功呼叫此函数将增加运动命令的库存数目。		
指令类型	缓冲区指令	章节页码	
Parameters	*wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_CheckBlend(WORD *wGroupIndex*)

Description	检查是否开启连续运动功能
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 已开启连续运动功能 1 未开启连续运动功能 其他 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DelayMotion(DWORD *dwTime*, WORD *wGroupIndex*)

Description	设定运动延迟时间，强制延迟执行下一条运动命令。		
指令类型	缓冲区指令	章节页码	
Parameters	*dwTime* 延迟时间，单位为 ms *wGroupIndex* group 编号		
Return Value	大于或等于 0 MCCL 给予此运动命令的编码 小于 0 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_CheckDelay(WORD *wGroupIndex*)

Description	检查目前是否进入运动延迟状态(此时若呼叫 MCC_GetMotionStatus() 将得到 GMS_DELAYING 的回传值)
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	0 未处于运动延迟状态 1 处于运动延迟状态 其他 失败，传回值的意义请参考 **2.1 函数返回值**

double MCC_OverrideSpeed(double *dfRate*, WORD *wGroupIndex*)

Description	设定一般运动的速度强制比例，使用此项函数将实时变动一般运动的进给速度。		
指令类型	缓冲区指令	章节页码	
Parameters	*dfRate* *dfRate* 为变更进给速度为原来进给速度的百分比再乘以 100，也就是说一般运动新的进给速度将等于(dfFeedSpeed × *dfRate* / 100)，dfFeedSpeed 为原来使用 MCC_SetFeedSpeed()所设定的进给速度。 *dfRate* 的设定值必须大于 0；若更新后的进给速度超过 MCC_SetSysMaxSpeed()的设定值，则新的进给速度将等于此设定值。 *wGroupIndex* group 编号		
Return Value	大于 0 实际设定的速度强制比例 其他 失败，传回值的意义请参考 **2.1 函数返回值**		

double MCC_GetOverrideRate(WORD *wGroupIndex*)

Description	读取一般运动当前使用的速度强制比例
指令类型	立即执行指令
Parameters	*wGroupIndex* group 编号
Return Value	大于 0 一般运动目前使用的速度强制比例 其他 失败，传回值的意义请参考 **2.1 函数返回值**

定时器与 Watch Dog 控制

int MCC_SetTimer(DWORD *dwValue*, WORD *wCardIndex = 0*)

Description	设置定时器的计时周期，在每个计时周期可触发自定义的近端输入 IO 中断服务函数。		
指令类型	立即执行指令	章节页码	
Parameters	*dwValue* 计时周期，单位为 10ns，设定范围为 1 ~ 2^{32}		
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**		

int MCC_EnableTimer(WORD *wCardIndex = 0*)

Description	开启定时器计时功能
指令类型	立即执行指令

Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DisableTimer(WORD *wCardIndex = 0*)

Description	关闭定时器计时功能
指令类型	立即执行指令
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_EnableTimerTrigger(WORD *wCardIndex = 0*)

Description	开启定时器在每个计时周期时触发近端输入 IO 中断服务函数的功能
指令类型	立即执行指令 章节页码
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DisableTimerTrigger(WORD *wCardIndex = 0*)

Description	关闭定时器在每个计时周期触发近端输入中断服务函数的功能
指令类型	立即执行指令
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetWatchDogTimer(WORD *wValue*, WORD *wCardIndex = 0*)

Description	设定 watch dog 的倒数计时时间。一旦 watch dog 计时终了时会产生硬件 reset 的信号，如不想产生 reset 信号，则在计时终了前可利用 MCC_RefreshWatchDogTimer()使 watch dog 计时器重新计数。
指令类型	立即执行指令 章节页码
Parameters	*wValue* watch dog 的倒数计时时间。单位为 MCC_SetTimer() 所设定的定时器计时周期，设定范围为 1 ~ 2^{32}
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_SetWatchDogResetPeriod(WORD *wValue*,
WORD *wCardIndex = 0*)

Description	设定在 watch dog 计时终了时所产生硬件 reset 信号的持续时间
指令类型	立即执行指令
Parameters	*wValue* 硬件 reset 信号持续时间，单位为 10ns
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_EnableWatchDogTimer(WORD *wCardIndex = 0*)

Description	开启 watch dog 功能
指令类型	立即执行指令
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_DisableWatchDogTimer(WORD *wCardIndex = 0*)

Description	关闭 watch dog 功能
指令类型	立即执行指令
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

int MCC_RefreshWatchDogTimer(WORD *wCardIndex = 0*)

Description	重置 watch dog，避免 watch dog 计时终了产生硬件 reset 信号。
指令类型	立即执行指令
Parameters	*wCardIndex* 运动控制卡的编号 0
Return Value	0 成功 非零 失败，传回值的意义请参考 **2.1 函数返回值**

附录：

Ⅰ错误信息代码

错误代码说明

0xF101 尚未初使化運動控制函数库

0xF102 急停信号触发报警

0xF104 在使用圆弧运动命令時给定的参数不合理

0xF105 正向运动学错误

0xF107 Custom Motion Error

0xF203 进给速度太快，超过每个插值時間内允許输出的 Pulse 数

0xF204 进给加速度太快，超過每個插值時間内允许輸出的 Pulse 增量数

0xF301 X 轴坐标值超出机构参数设定的工作范围

0xF302 Y 轴坐标值超出机构参数设定的工作范围

0xF303 Z 轴坐标值超出机构参数设定的工作范围

0xF304 U 轴坐标值超出机构参数设定的工作范围

0xF305 V 轴坐标值超出机构参数设定的工作范围

0xF306 W 轴坐标值超出机构参数设定的工作范围

0xF307 A 轴坐标值超出机构参数设定的工作范围

0xF308 B 轴坐标值超出机构参数设定的工作范围

0xF401 运动命令在执行时发生错误

0xF501 定位请确认错误

0xF701 X 轴碰到硬极限开关

0xF702 Y 轴碰到硬极限开关

0xF703 Z 轴碰到硬极限开关

0xF704 U 轴碰到硬极限开关

0xF705 V 轴碰到硬极限开关

0xF706 W 轴碰到硬极限开关

0xF707 A 轴碰到硬极限开关

0xF708 B 轴碰到硬极限开关

0xF801 X 轴跟随误差超出设定容许范围

0xF802 Y 轴跟随误差超出设定容许范围

0xF803 Z 轴跟随误差超出设定容许范围

0xF804 U 轴跟随误差超出设定容许范围

0xF805 V 轴跟随误差超出设定容许范围

0xF806 W 轴跟随误差超出设定容许范围

0xF807 A 轴跟随误差超出设定容许范围

0xF808 B 轴跟随误差超出设定容许范围

II 运动函数库初始值设定

下表所列出的是调用 MCC_InitSystem()后，MCCL 的初始设定，初始设定若无法满足使用者的需要，可调用相关函数进行更改。

初始设定内容	初始设定	相关函数
命令缓冲区大小	10000 笔命令	MCC_SetCmdQueueSize() MCC_GetCmdQueueSize()
运动空跑功能	未开启	MCC_EnableDryRun() MCC_DisableDryRun()
机台允许的的最大进给速度	100	MCC_SetSysMaxSpeed() MCC_GetSysMaxSpeed()
系統坐标型式	绝对坐标	MCC_SetAbsolute() MCC_SetIncrease() MCC_GetCoordType()
各轴允许的最大 Pulse 加速度	32767	MCC_SetMaxPulseAcc() MCC_GetMaxPulseAcc()
各轴允许的最大 Pulse 速度	32767	MCC_SetMaxPulseSpeed() MCC_GetMaxPulseSpeed()
软限位检查	未开启	MCC_SetOverTravelCheck() MCC_GetOverTravelCheck
硬限位开关检查	未开启	MCC_EnableLimitSwitchCheck() MCC_DisableLimitSwitchCheck()
位置控制閉迴路使用的的比例增益	64	MCC_SetPGain() MCC_GetPGain()
进行直线、圆弧、圆、螺线运动时各轴使用的加、減速型式	S 型曲线	MCC_SetAccType() MCC_GetAccType() MCC_SetDecType() MCC_GetDecType()
进行直线、圆弧、圆、螺线运动时各轴使用的加、減速时间	300 ms	MCC_SetAccTime() MCC_GetAccTime() MCC_SetDecTime() MCC_GetDecTime()
进行直线、圆弧、圆、螺线运动时使用的进给速度	1	MCC_SetFeedSpeed() MCC_GetFeedSpeed()
进行点对点运动时各轴使用的速度比例	1	MCC_SetPtPSpeed() MCC_GetPtPSpeed()
到位确认最大检查时间	1000 ms	MCC_SetInPosMaxCheckTime()
到位确认持续时间	100 ms	MCC_SetInPosSettleTime()
到位确认容许误差范围	∞	MCC_SetInPosToleranceEx() MCC_GetInPosToleranceEx()
到位确认功能	未开启	MCC_EnableInPos() MCC_DiableInPos()
连续运动功能	未开启	MCC_EnableBlend() MCC_DisnableBlend()
跟随误差检测功能	未开启	MCC_EnableTrackError() MCC_DisnableTrackError()
跟随误差允许范围	∞	MCC_SetTrackErrorLimit() MCC_GetTrackErrorLimit()

Ⅲ 控制器固定 IP 设定

1.找到控制器 SD 卡/root/home 目录下的 interfaces 文件；

2.将 interfaces 文件拷贝到/etc/network 目录下，将之前该目录下的 interfaces 文件覆盖掉。具体操作如下：

在 Linux 系统的 Terminal 中输入指令：

cp interfaces /etc/network/interfaces

3.关机重新启动，即可将 IP 设为固定地址：192.168.1.200