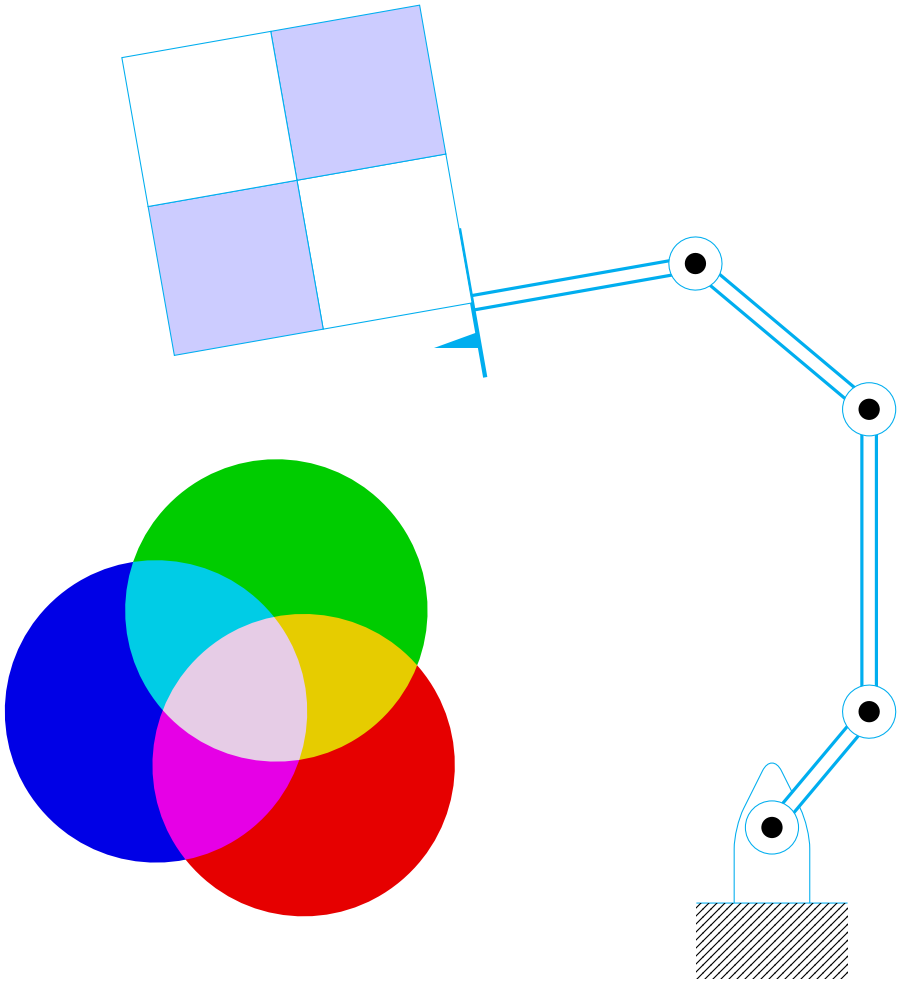
AsuMotion运动控制卡使用手册
(第1版)



AsuMotion编著

版本: 1.490

序

感谢您使用ASU Motion系列运动控制卡，本手册能让使用者更好的了解ASU Motion系列运动控制卡，并提供给使用者安装、使用条件，参数设定、常用功能实现办法，故障排除等相关注意事项。为了能确保使用者正确使用ASU Motion系列运动控制卡，请在安装使用前，详细阅读本使用手册，并请使用者妥善保存。

本手册可能包含技术上不准确的地方、或与产品功能及操作不相符的地方、或印刷错误。我司将根据产品功能的增强或变化而更新本手册的内容，并将定期改进及更新本手册中描述的软硬件产品。更新的内容将会在本手册的新版本中加入，恕不另行通知。本手册中内容仅为用户提供参考指导作用，不保证与实物完全一致，请以实物为准。

ASU Motion系列运动控制卡支持固件程序在线升级，固件程序的更新，带来的功能更新或描述可能和本使用手册描述有出入，敬请谅解。 ASU Motion系列运动控制卡属于精密自动控制电子产品，为了操作者及运动控制卡的安全使用，请务必由专业的电气自动化控制工程人员安装调试及调整参数，本手册中有危险，注意等特别提示的地方，请务必仔细研读。

安全标记的说明

错误使用，可能会导致火灾、以及发生设备损坏，无视注意事项，可能会导致伤害或财产损失

可能会导致遭受电击，无视警告事项，可能会导致死亡或严重伤害。

可能会导致人身中等程度的伤害或轻伤，以及发生设备损坏，无视警告事项，可能会导致死亡或严重伤害。

AsuMotion

2016/11/03 12:18:44

版权申明

AsuMotion保留在不事先通知的情况下，修改本手册中的产品和产品规格等文件的权力。

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

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

本文档中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明外, 其著作权或其他相关权利均属于AsuMotion。未经AsuMotion书面同意，任何人不得以任何方式或形式对本手册内的任何部分进行复制、摘录、备份、修改、传播、翻译成其它语言、将其全部或部分用于商业用途。

免责申明

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

AsuMotion

2016/11/03 12:18:44

产品简介

AsuMotion是一个USB接口的运动控制卡，其工作特征如下：

1. 适用任何具有USB接口的上网本，笔记本，台式机以及平板等PC兼容计算机。
2. 支持Mach3 所有版本，包括目前最新版Mach3 R3.043.066。
3. 支持二次开发，提供所有API函数以供使用,同时支持同步IO操作。
4. 支持所有Windows版本，免USB驱动安装，支持即插即用操作。
5. 全面支持USB热插拔，随时监测USB连线状态，基于Mach3操作下，可实现断线再连操作。
6. 最大支持9轴联动，Mach3下最大支持6轴联动
7. 支持自动对刀，电子手轮，软件限位等功能。
8. 输出频率最高可达4M，完美驱动伺服、步进。
9. 具有状态指示灯，可自动提示USB连接，Mach3连线状态，运行状态一目了然。
10. 拥有16个通用输入接口，可接自由设置共阴共阳。
11. 拥有9个电子开关输出。
12. 16路差分输出，支持CW/CCW输出、正交输出、方向脉冲输出。并可以随意设定每个输出的功能。
13. 2个模拟量输出接口，可以用来控制变频器来控制主轴。完美支持步进、伺服驱动在主轴中的应用。
14. 2个模拟量输入可以用作外部倍率选择，可以通过软件配置调节加工进给速度，主轴速度。
15. 4层电路板精心布线，唯选优质器件，制作精良。
16. 拥有主轴测速功能（支持霍尔元件测速、旋转编码器测速等方式），Mach3中可实时显示主轴转速，完美支持了车床中的主轴应用，及其他需要精确主轴转速的场合。
17. 采用外部24V直流电源供电，达到USB和外部端口真正的隔离，并且外部接口不用 USB供电，让系统工作更稳定。

目录

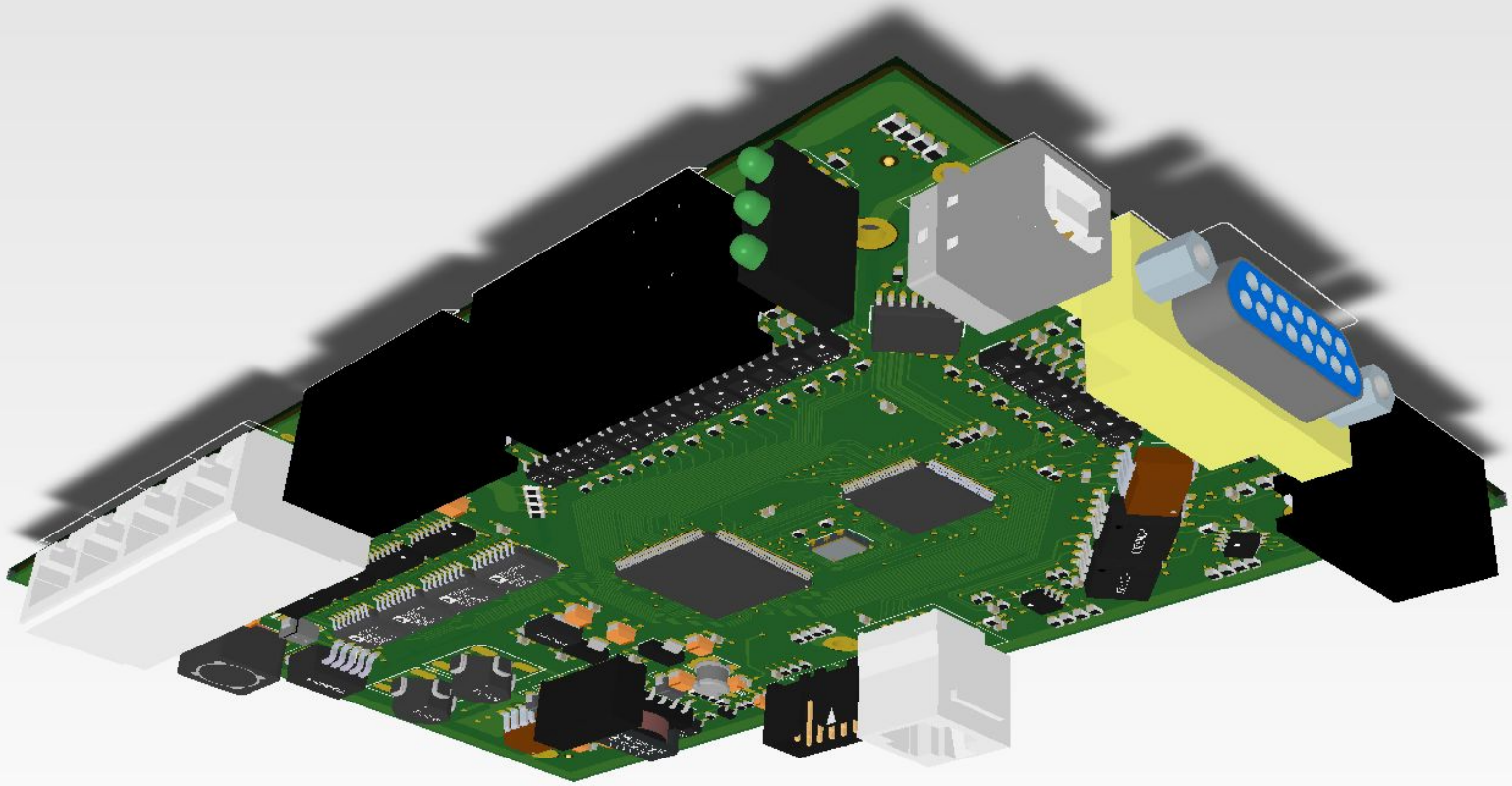
第1章　Mach3版ASU Motion快速使用说明

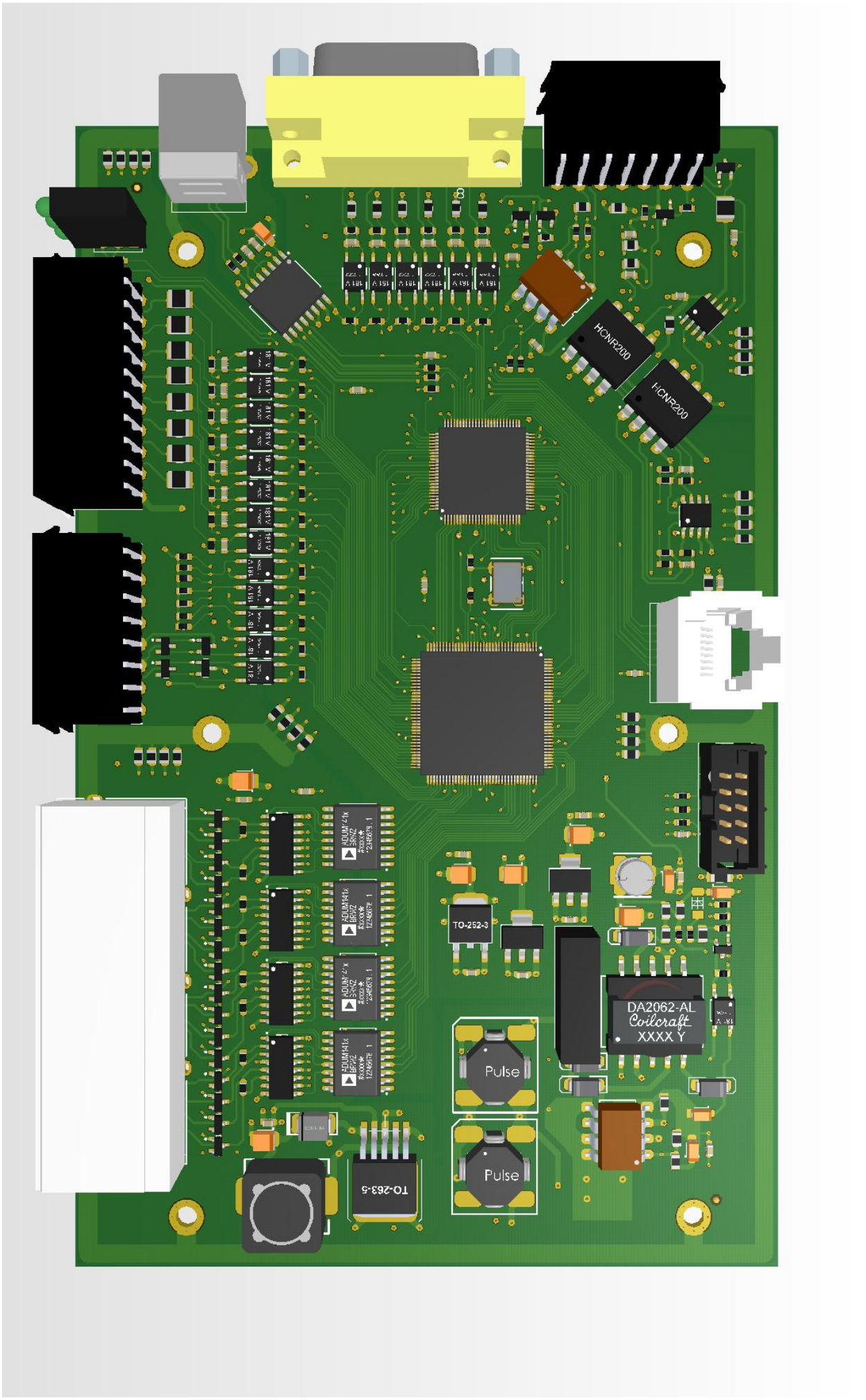
§ 1.1 开箱检查

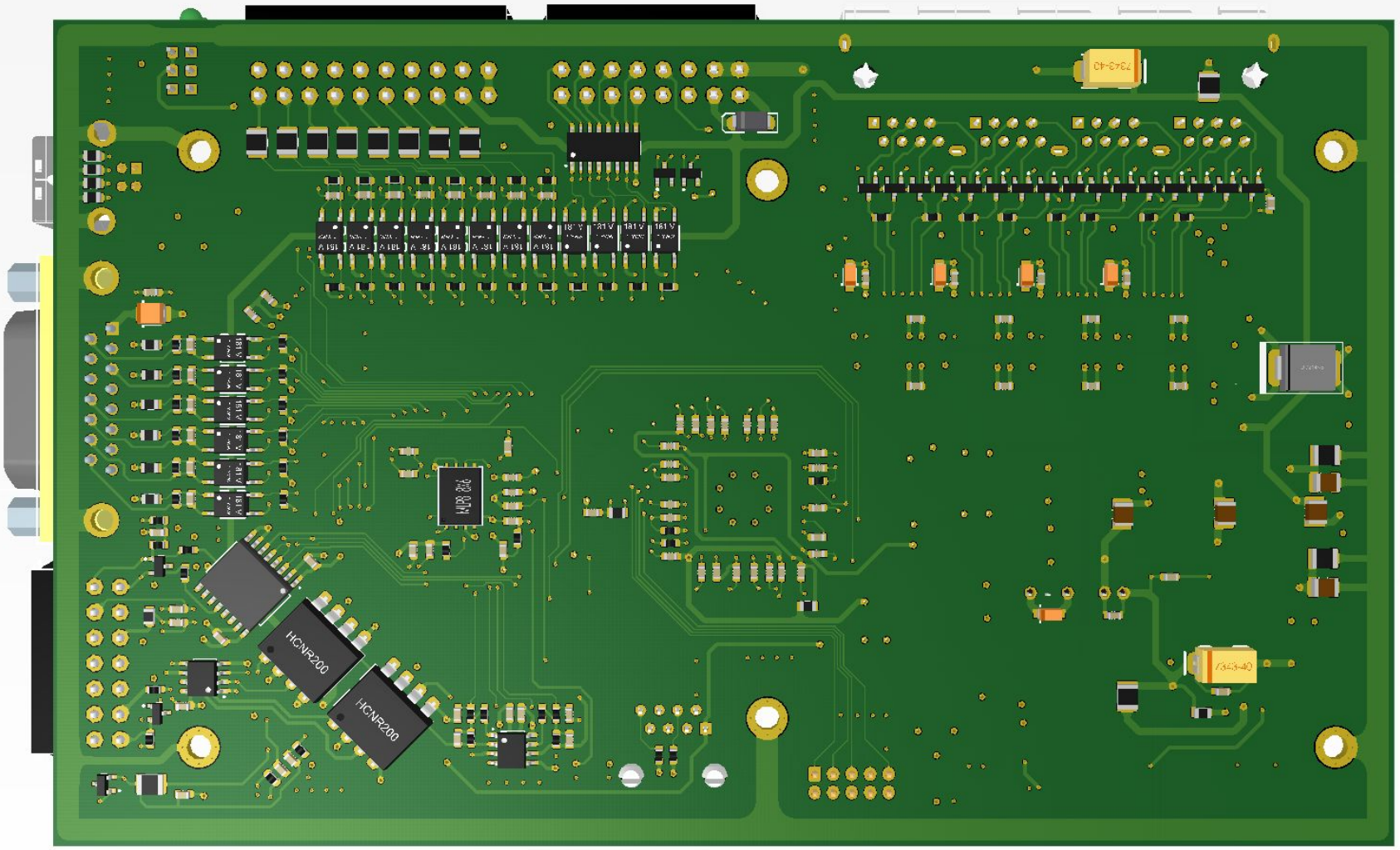
打开包装后，请首先检查运动控制卡的表面是否有机械损坏，然后按照装箱清单或订购合同仔细核对配件是否齐备。如果板卡表面有损坏，或产品内容不符合，请不要使用，立即与经销商联系。

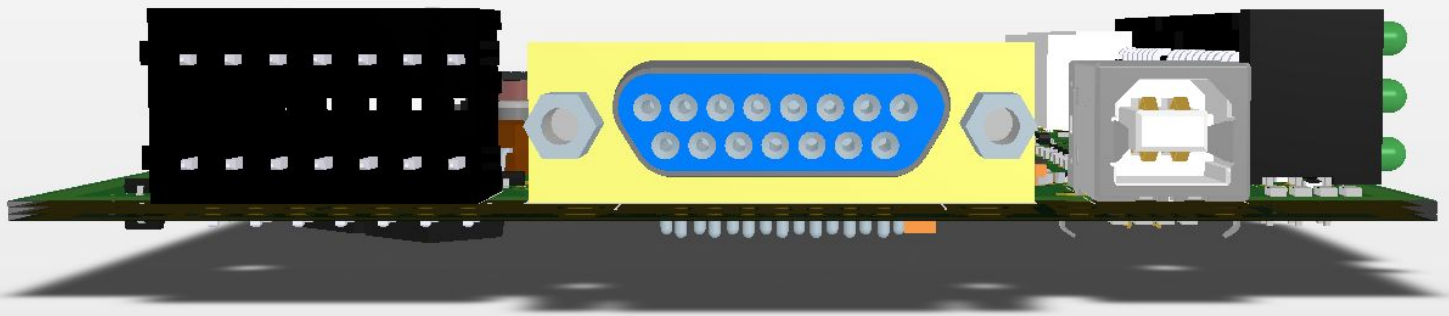
为了防止静电损害运动控制器，请在接触控制器电路或插拔控制器之前触摸有效接地金属物体以泄放身体所携带的静电荷，最好去洗手，通过接触金属水龙头泄放身体所携带的静电荷。在冬天尤其要注意。

§ 1.2 运动控制卡的外形结构




HCNR200
HCNR200
TO-252-3
DA2062-AL
Coilcraft
XXXX Y
Pulse
Pulse
TO-263-5


HCNR200
HCNR200



§ 1.3 主要接口说明

1.3.1 ASU Motion差分输出端口

由于大部分马达驱动器采用差分输入，因此ASU Motion的输出为差分输出，最多支持16路差分输出，每四路差分输出使用一个RJ-45网口输出，因此共有4个RJ-45接口。为了方便说明问题，这里给四个RJ45从上到下依次起名为RJ45_A，RJ45_B，RJ45_C，RJ45_D（靠近板子边缘）具体参考图1.3。RJ45_A接口的默认接线标准定义见表1.1。其他三个RJ45可以依次类推。这些接口可以任意设定输出功能，关于如何设定，请查阅2.6。为了便于用户方便确认水晶头接线的颜色，这里以图片方式说明，见图1.1，图1.2。对于ASU Motion，推荐使用T568B标准。

表 1.1: 接线功能对照表

RJ45_A的RJ45管脚	T568A颜色	T568B颜色	ASU Motion的功能
1	绿白	橙白	Diff0+
2	绿	橙	Diff0-
3	橙白	绿白	Diff1+
4	蓝	蓝	Diff1-
5	蓝白	蓝白	Diff2+
6	橙	绿	Diff2-
7	棕白	棕白	Diff3+
8	棕	棕	Diff3-

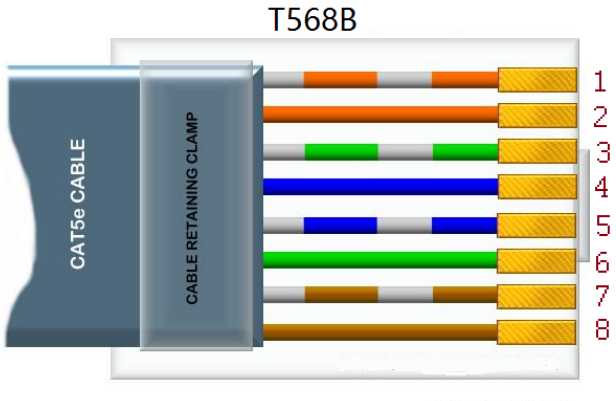


图 1.1: T568B的水晶头接线颜色.

1.3.2 电源输入与数字量输出端子

数字量输出端子为双排16针插座，其管脚间距为3.5mm。管脚定义见表1.2。达林顿管集电极输出的驱动芯片为ULN2003，每路可以最大承受电流最大500mA，电压为24V（最大可耐压50V）。为防止ULN2003烧毁，建议通过ULN2003的工作电流50 mA。下面针对几个特殊管脚进行说明：

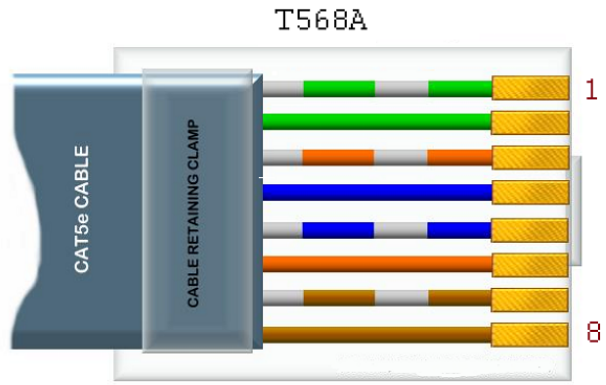


图 1.2: T568A的水晶头接线颜色.

13 ：每个达林顿管集电极输出都有一个二极管连接到此引脚，因此如果达林顿管输出接感性负载时，比如继电器线圈，那么可以将此管脚接继电器线圈的电源端，使得在达林顿管断开是，可以通过二极管，续流到此管脚，用于保护达林顿管不受损害，如果此关脚悬空，那么在接感性负载时，很容易损坏达林顿输出。当感性负载使用10脚的24V电源输出时，必须将此管脚与10脚连接在一起。

7 ：所有达林顿管的发射极输出都连接到此管脚，使用时一般接负载电源的地。如果负载使用10脚的24V电源输出，那么此管脚可以悬空。

8 ：本运动控制卡的电源输入GND。

9 ：本运动控制卡的电源输入24V。

电源输入为24V，在端子的最右侧的8脚和9脚，注意不要接反!

表 1.2: 输出端子的管脚定义

输出端子管脚	管脚默认功能	输出端子管脚	管脚默认功能
1	达林顿管集电极输出0	16	达林顿管集电极输出1
2	达林顿管集电极输出2	15	达林顿管集电极输出3
3	达林顿管集电极输出4	14	达林顿管集电极输出5
4	达林顿管集电极输出6	13	反向二极管阴极
5	开关输出8Y	12	开关输出7Y
6	开关输出8C	11	开关输出7C
7	达林顿管公共GND	10	24V输出
8	电源输入GND	9	电源24V输入

1.3.3 ASU Motion数字量输入接口

数字量输出端子为双排20针，其管脚间距为3.5mm。管脚定义见表1.3。下面针对几个特殊管脚进行说明：

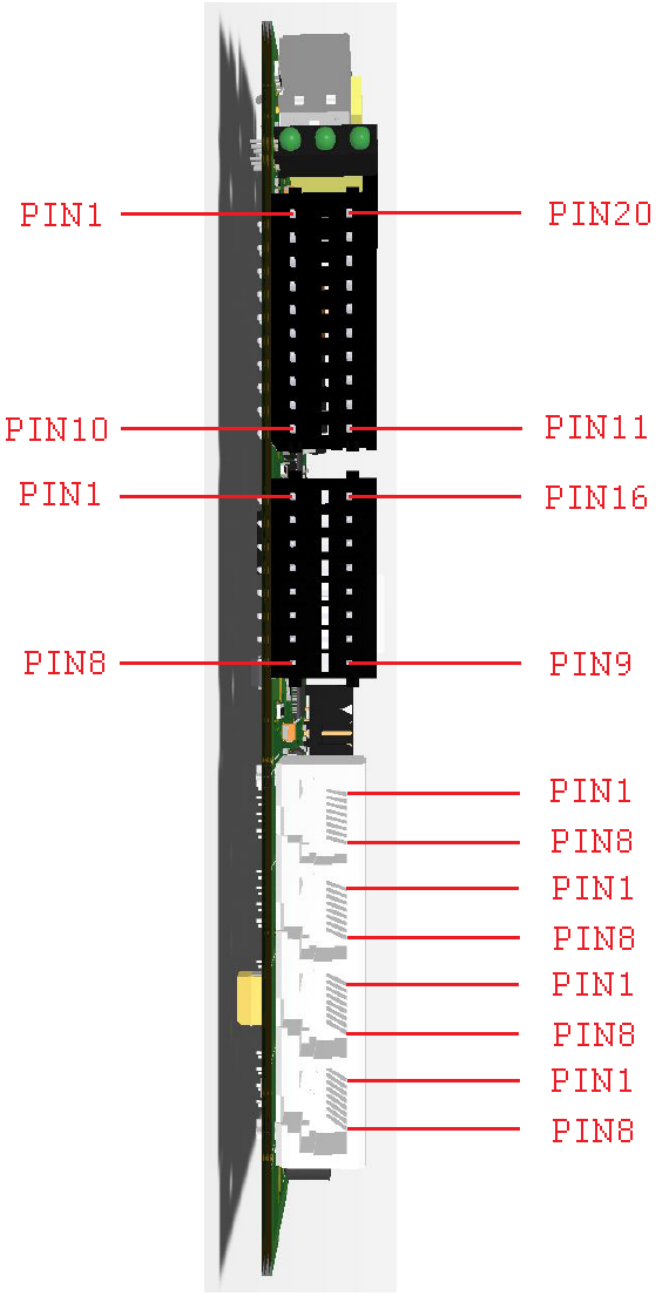


图 1.3: 数字量输入输出与差分端子

10 ：如果选择光耦输入类型为共阴，可以将此管脚与管脚9短接。

11 ：如果选择光耦输入类型为共阳，可以将此管脚与管脚12短接。。

9 ：本运动控制卡的电源输出GND。

12 ：本运动控制卡的电源输出24V。

1. 管脚9和12不是本运动控制卡的电源输入！
2. 管脚10和11不可以同时分别连接GND和24V！

表 1.3: 输入端子的管脚定义

输入端子管脚	管脚功能	输入端子管脚	管脚功能
1	光耦输入0	20	光耦输入1
2	光耦输入2	19	光耦输入3
3	光耦输入4	18	光耦输入5
4	光耦输入6	17	光耦输入7
5	光耦输入8	16	光耦输入9
6	光耦输入10	15	光耦输入11
7	光耦输入12	14	光耦输入13
8	光耦输入14	13	光耦输入15
9	GND	12	24V
10	共阳共阴选择	11	共阳共阴选择

1.3.4 ASUMotion模拟量，正交编码器接口

ASUMotion模拟量接口端子为双排14针，其管脚间距为3.5mm。管脚定义见表1.4。下面针对几个特殊管脚进行说明：

3 ：AGND作为模拟量的地。

4 ：GND作为数字量的地。

12 ：+5VA作为模拟量的电源。

11 ：+5V作为数字量的电源。比如如果管脚5，6，9，10 接正交编码器时，可以用+5V和GND作为编码器的电源。

1.3.5 ASUMotion工业手轮接口

ASUMotion工业手轮接口为双排DB15针母头。管脚定义见表1.5。下面针对几个特殊管脚进行说明：

16 ：作为DB15的金属外壳以及两颗固定螺丝。用来接真正的大地。接地电阻应不大于1欧姆。

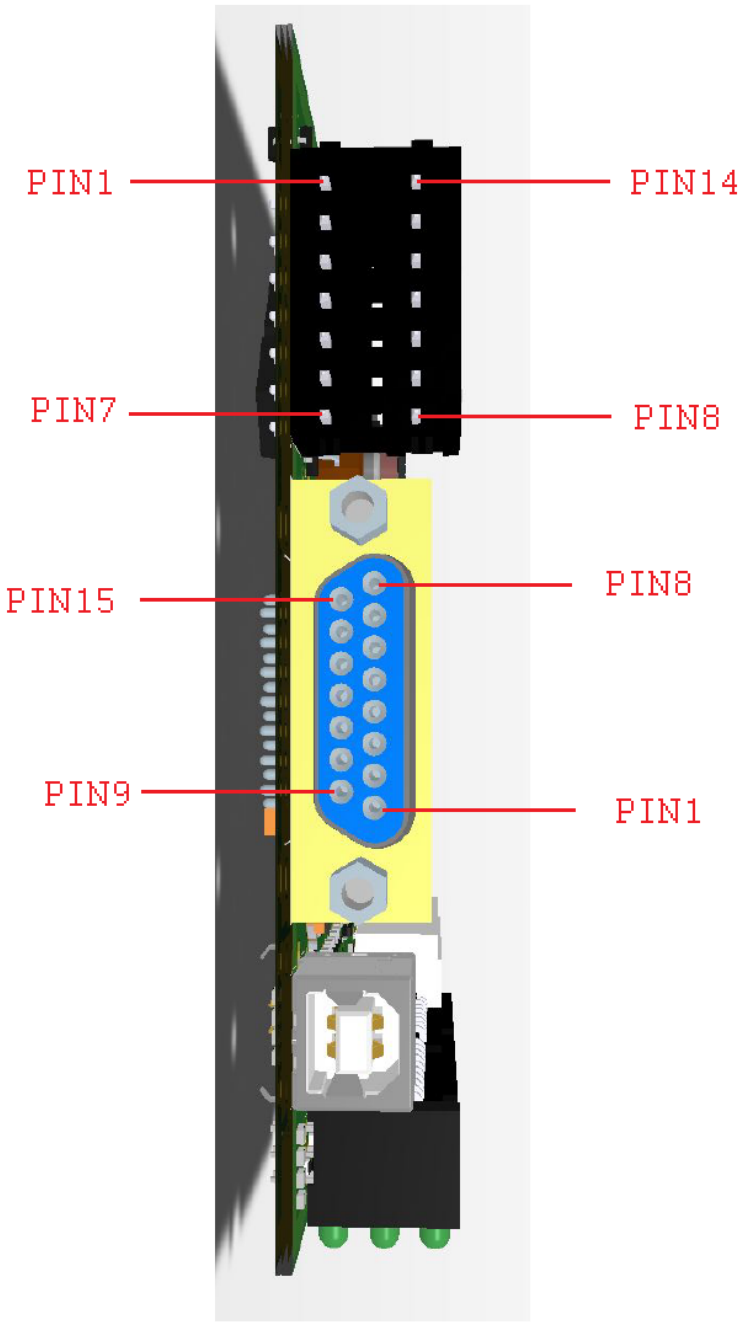


图 1.4: 模拟量，正交编码器手轮端子

表 1.4: 模拟量，脉冲输入，RS485接口管脚定义

输入端子管脚	管脚功能	输入端子管脚	管脚功能
1	模拟量输出0	14	模拟量输入0
2	模拟量输出1	13	模拟量输入1
3	AGND	12	+5VA
4	GND	11	+5V
5	B-	10	B+
6	A-	9	A+
7	RS485B-	8	RS485A+

表 1.5: 工业手轮接口管脚定义

输入端子管脚	管脚功能	输入端子管脚	管脚功能
1	+5V	16	大地
2	编码器A+	15	手轮X轴
3	编码器B+	14	手轮Y轴
4	输入IO17	13	手轮Z轴
5	手轮OFF	12	输入IO16
6	手轮倍率X1	11	GND
7	手轮倍率X10	10	保留
8	手轮倍率X100	9	手轮A轴

1.3.6 ASU Motion USB接口

此处无需赘言。

第2章　Mach3版ASUMotion插件配置与使用

§ 2.1 插件安装与系统调试

2.1.1 插件安装

一般情况下，ASUMotion插件共有5个dll文件，这5个文件的获取，请联系供应商。

1. ASUMotion.dll 运动控制卡插件文件
2. ASUMotion_zh_TW.dll 插件中文繁体语言包
3. ASUMotion_zh_CN.dll 插件中文简体语言包
4. ASUMotion_en_US.dll 插件英文语言包
5. ASUMotion_de_DE.dll 插件德文语言包

ASUMotion插件的安装非常简单，其主要步骤如下：

1. ASUMotion.dll插件放置于Mach3\PlugIns中。
2. 将4个语言包文件放置于Mach3\中，也就是Mach3的根目录。
3. 用USB线将ASUMotion运动控制卡与电脑相连接，首次使用时，请稍等片刻，等待驱动程序自动安装完成。
4. 启动Mach3软件，您会看到运动控制卡的选择对话框，选择带有ASUMotion字样的插件。
5. ASUMotion控制卡上指示灯将闪烁，并且在Mach3的状态栏中显示“USB端口打开成功”，表示 Mach3与控制卡的连接已经完成。

2.1.2 系统调试

2.1.2.1 连接电机与驱动器

在驱动器没有与控制卡连接之前，连接驱动器与电机。请仔细参考驱动器的说明书，正确连接。按照驱动器说明书的要求测试驱动器与电机，确保其工作正常。

为安全起见，建议用户在未完成控制系统的安装、调试前，不要将电机与任何机械装置连接。请检查电机确实没有负载。

2.1.2.2 连接控制卡和电机驱动器

1. 关闭电机驱动器电源，将本控制卡电源关闭。
2. 仔细了解本控制器的差分输出接口信号和电机驱动器的接口定义，妥善连线并避免带电插拔接口。否则，信号连接错误或带电操作可能导致系统或硬件损坏使系统不能正常工作。

2.1.2.3 连接急停开关、原点信号和限位信号

1. 系统的限位开关须接成常闭状态；
2. 原点开关为常开状态。

2.1.2.4 调试输入信号

认真检查接线后，将控制卡上电，并连接usb，然后打开插件的状态显示界面，检查各原点开关，限位信号，急停开关等信号可以准确无误的传递到插件。

2.1.2.5 调试电机转动方向

§ 2.2 状态显示与调试界面

如图2.1所示，最下方有一个其他设置按钮：点击后将出现其他的配置插件功能，如图2.2所示，其功能介绍见后面其它章节。这里介绍下本页面的其他主要功能。

2.2.1 输入输出口仿真

输入输出口仿真可以用来进行对硬件进行仿真，无论其真实的硬件处于什么状态。比如输入口仿真，一旦使能以后，那么物理硬件的输入接口的状态将不影响插件的逻辑。此处设置的值将旁路掉硬件的输入接口。对于输出口仿真，一旦使能以后，无论mach3内部逻辑状态如何，其真实硬件的输出接口将输出此处设置的值。与此类似，模拟量此处只可以进行模拟量输入的仿真。状态显示提供了8个数字量输出状态的显示，以及16个数字量输入状态的显示。输出状态表示了硬件数字量输出的前8个真实值。输入状态代表硬件数字量输入的真实状态，可以用于在连接硬件时，确认硬件连接是否正确。

2.2.2 插件内部运行状态监视

此界面还提供了监视插件内部运行状态的功能，可以同时显示两个变量值。具体什么变量可以通过后面的组合框选取。

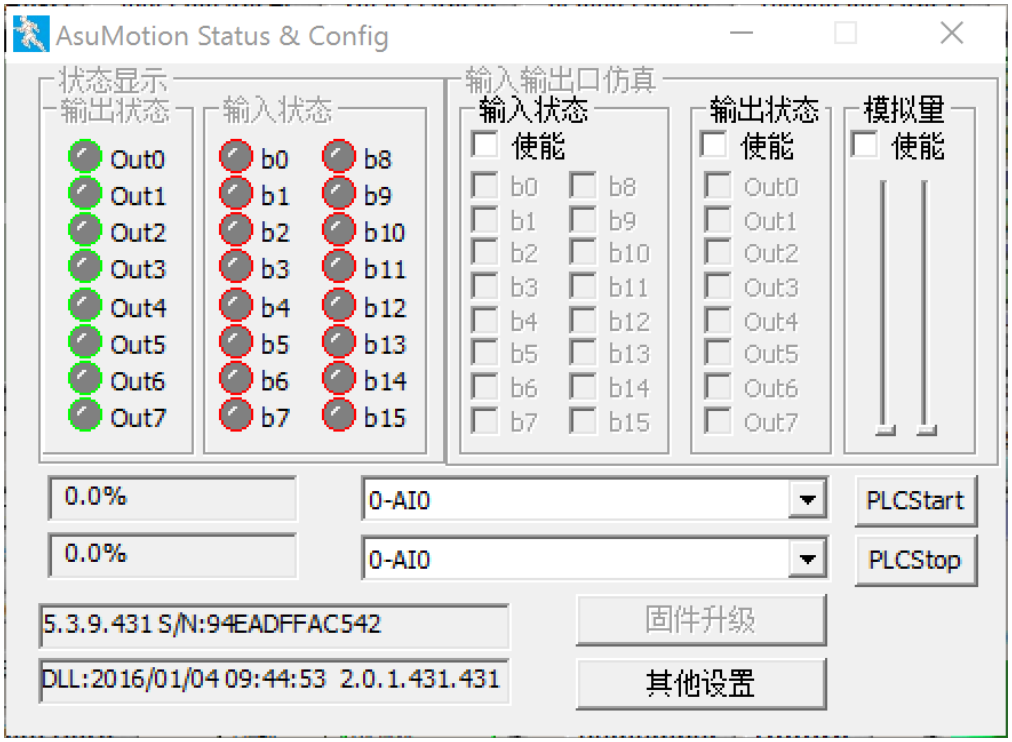


图 2.1: 状态显示与调试界面

2.2.3 固件升级

固件升级按钮：在有新的固件时，将其按照插件的安装方法安装后，如果新的固件版本与当前版本不同，那么此处的固件升级按钮有效。点击可以升级成或者降级成原来的固件。

§ 2.3 主体设置页面

图 2.2: 主体设置

如图2.2所示，主体设置提供了光滑系数，脉冲延时，主轴测量，语言选择等功能。下面将一一介绍：

2.3.1 光滑系数设置

此系数用于插件内部进行分辨率的设置，默认为1/64，其意义为插件内部的运算按照1/64个脉冲进行累加。因此，如果光滑系数越小，那么代表分辨率越高。但是，并不是光滑系数越小越好，当光滑系数设置的比较小时，运动控制卡产生的脉冲频率将会相应的降低。因此，实际使用时应当按照所用马达驱动器的性能以及所用的最大速度来设置。表2.1列出了不同的光滑系数对应的控制卡输出的最大脉冲频率以及对应的单位脉冲数与速度的乘积。

表 2.1: 光滑系数

光滑系数	控制卡输出最大频率	单位脉冲数与速度乘积
$\frac{1}{2}$	$16MHz$	960000000
$\frac{1}{4}$	$8MHz$	480000000
$\frac{1}{8}$	$4MHz$	240000000
$\frac{1}{16}$	$2MHz$	120000000
$\frac{1}{32}$	$1MHz$	60000000
$\frac{1}{64}$	$500KHz$	30000000
$\frac{1}{128}$	$250KHz$	15000000
$\frac{1}{256}$	$125KHz$	7500000
$\frac{1}{1024}$	$62.5KHz$	3250000

2.3.2 脉冲方向延时设置

在使用脉冲+方向的输出方式时，由于在方向信号发生变化的瞬间，脉冲信号是不允许变化的，否则马达驱动器将不知道此时发生的脉冲信号应该作用在正方向还是反方向。因此在方向发生变化后，有必要使得脉冲信号延时一段时间产生，保证产生的脉冲为方向变化后的脉冲。

AsuMotion插件提供了这一选项，这使得在使用某些性能比较低的马达驱动器时，可以保证马达驱动器不丢步。

另外如果马达驱动器采用正交脉冲方式，那么可以设置此处的延时为0μs，也就是此设置对于正交脉冲无意义。当然，也可以不修改此处的值。

2.3.3 主轴测量

2.3.4 语言选择

目前支持4种语言的选择，修改后，退出Mach3，重启Mach3生效。

2.3.5 FRO，SRO，JOG源选择

FRO用于调正进给速率，SRO用于调整主轴速率，JOG用于调整点动速率。这三个设置共有四个选项依次为：

1. 内部：将选择Mach3界面的设置来进行调整相应的速率。
2. 外部0：将选择AsuMotion的模拟量输入0来调整相应的速率，为无级调速，手感比较差。
3. 外部1：将选择AsuMotion的模拟量输入1来调整相应的速率，为无级调速，手感比较差。
4. 外部离散输入0：将选择AsuMotion的模拟量输入0来调整相应的速率，为有级调速，手感好。
5. 外部离散输入1：将选择AsuMotion的模拟量输入1来调整相应的速率，为有级调速，手感好。

§ 2.4 输入信号设置

如图2.3所示，Mach3的输入信号可以通过此界面进行设置，类似于Mach3的原生配置见面，此处可以更好的配置Mach3的输入信号使用AsuMotion的哪一个端口。不需要用户记忆哪个管脚。当配置相对当前配置有修改时，相应的配置后面将有红色* 出现，代表此项配置有修改，同时“应用”按钮有效。点击“应用”按钮将保存当前配置，点击“取消”按钮将放弃当前配置。其配置选项这里就不再说了。

图 2.3: 输入信号设置

§ 2.5 输出信号设置

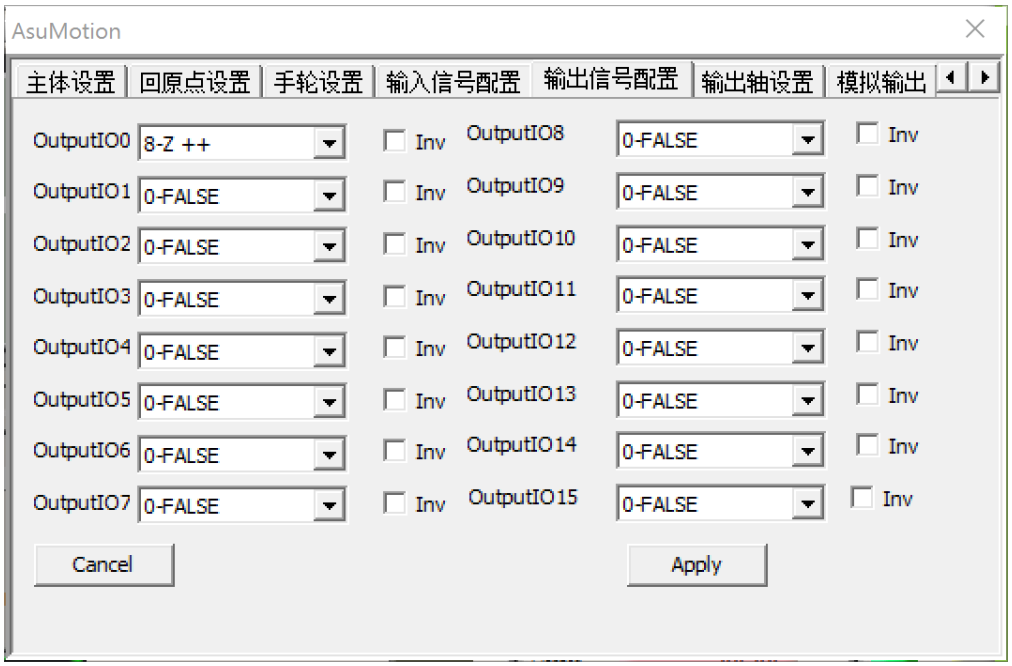


图 2.4: 输出信号设置

如图2.4所示，Mach3以及插件的一些内部逻辑信号可以通过此界面进行设置映射到AsuMotion的输出接口，一旦配置有修改，“应用”按钮有效。点击“应用”按钮将保存当前配置，点击“取消”按钮将放弃当前配置。Inv将输出当前选择信号的非。可供选择的逻辑信号见表2.2

表 2.2: 逻辑信号表

信号序号	信号选项	备注
0	FALSE	输出强制为低电平
1	TURE	输出强制为高电平
2	X ++	请参考Mach3说明书
3	X –	请参考Mach3说明书
4	X Home	请参考Mach3说明书
5	Y ++	请参考Mach3说明书
6	Y –	请参考Mach3说明书
7	Y Home	请参考Mach3说明书
8	Z ++	请参考Mach3说明书
9	Z –	请参考Mach3说明书
10	Z Home	请参考Mach3说明书
11	A ++	请参考Mach3说明书
12	A –	请参考Mach3说明书
13	A Home	请参考Mach3说明书
14	B ++	请参考Mach3说明书
15	B –	请参考Mach3说明书
16	B Home	请参考Mach3说明书
17	C ++	请参考Mach3说明书
18	C –	请参考Mach3说明书
19	C Home	请参考Mach3说明书
20	Input #1	请参考Mach3说明书
21	Input #2	请参考Mach3说明书
22	Input #3	请参考Mach3说明书
23	Input #4	请参考Mach3说明书
24	Probe	请参考Mach3说明书
25	Index	请参考Mach3说明书
26	Limit Ovrd	请参考Mach3说明书
27	Estop	请参考Mach3说明书
28	THC On	请参考Mach3说明书
29	THC Up	请参考Mach3说明书
30	THC Down	请参考Mach3说明书
31	OEM Trig #1	请参考Mach3说明书
32	OEM Trig #2	请参考Mach3说明书

33	OEM Trig #3	请参考Mach3说明书
34	OEM Trig #4	请参考Mach3说明书
35	OEM Trig #5	请参考Mach3说明书
36	OEM Trig #6	请参考Mach3说明书
37	OEM Trig #7	请参考Mach3说明书
38	OEM Trig #8	请参考Mach3说明书
39	OEM Trig #9	请参考Mach3说明书
40	OEM Trig #10	请参考Mach3说明书
41	OEM Trig #11	请参考Mach3说明书
42	OEM Trig #12	请参考Mach3说明书
43	OEM Trig #13	请参考Mach3说明书
44	OEM Trig #14	请参考Mach3说明书
45	OEM Trig #15	请参考Mach3说明书
46	Timing	请参考Mach3说明书
47	Jog X ++	请参考Mach3说明书
48	Jog X –	请参考Mach3说明书
49	Jog Y ++	请参考Mach3说明书
50	Jog Y –	请参考Mach3说明书
51	Jog Z ++	请参考Mach3说明书
52	Jog Z –	请参考Mach3说明书
53	Jog A ++	请参考Mach3说明书
54	Jog A –	请参考Mach3说明书
55	Input53	请参考Mach3说明书
56	Input54	请参考Mach3说明书
60	EXP_A+	手轮编码器A+
61	EXP_B+	手轮编码器B+
62	EXP_OFF	手轮OFF
63	EXP_X	手轮X轴
64	EXP_Y	手轮Y轴
65	EXP_Z	手轮Z轴
66	EXP_A	手轮A轴
67	EXP_X1	手轮X1倍率
68	EXP_X10	手轮X10倍率
69	EXP_X100	手轮X100倍率

70	EXP_Input16	
71	EXP_Input17	
72	EXP_Input18	
73	EXP_Input19	
74	EXP_Input20	
75	EXP_Input21	
80	DIGTRIGGER	
81	ENABLE1	
82	ENABLE2	
83	ENABLE3	
84	ENABLE4	
85	ENABLE5	
86	ENABLE6	
87	OUTPUT#1	
88	OUTPUT#2	
89	OUTPUT#3	
90	OUTPUT#4	
91	OUTPUT#5	
92	OUTPUT#6	
93	CHARGE	
94	CHARGE2	
95	CURRENTHILOW	
96	OUTPUT#7	
97	OUTPUT#8	
98	OUTPUT#9	
99	OUTPUT#10	
100	OUTPUT#11	
101	OUTPUT#12	
102	OUTPUT#13	
103	OUTPUT#14	
104	OUTPUT#15	
105	OUTPUT#16	
106	OUTPUT#17	
107	OUTPUT#18	

108	OUTPUT#19	
109	OUTPUT#20	
110	Input IO_0	
111	Input IO_1	
112	Input IO_2	
113	Input IO_3	
114	Input IO_4	
115	Input IO_5	
116	Input IO_6	
117	Input IO_7	
118	Input IO_8	
119	Input IO_9	
120	Input IO_10	
121	Input IO_11	
122	Input IO_12	
123	Input IO_13	
124	Input IO_14	
125	Input IO_15	
130	PLC_DO_0	
131	PLC_DO_1	
132	PLC_DO_2	
133	PLC_DO_3	
134	PLC_DO_4	

§ 2.6 差分输出口设置

如图2.5所示，用户可以任意选择下拉框中的信号。这使得在不修改硬件改线的情况下，可以做到随意修改信号的连接。


AsuMotion
主体设置
回原点设置
手轮设置
输入信号配置
输出信号配置
差分输出设置
RJ45_A_12管脚
00:X轴的脉冲信号
Inv
RJ45_A_34管
10:X轴的方向信号
Inv
RJ45_A_56管脚
01:Y轴的脉冲信号
Inv
RJ45_A_78管
11:Y轴的方向信号
Inv
RJ45_B_12管脚
02:Z轴的脉冲信号
Inv
RJ45_B_34管
12:Z轴的方向信号
Inv
RJ45_B_56管脚
03:A轴的脉冲信号
Inv
RJ45_B_78管
13:A轴的方向信号
Inv
RJ45_C_12管脚
04:B轴的脉冲信号
Inv
RJ45_C_34管
14:B轴的方向信号
Inv
RJ45_C_56管脚
05:C轴的脉冲信号
Inv
RJ45_C_78管
15:C轴的方向信号
Inv
RJ45_D_12管脚
06:U轴的脉冲信号
Inv
RJ45_D_34管
16:U轴的方向信号
Inv
RJ45_D_56管脚
07:V轴的脉冲信号
Inv
RJ45_D_78管
17:V轴的方向信号
Inv
IO输出 #0 管脚
60:数字输出 0
Inv
IO输出 #1 管
61:数字输出 1
Inv
IO输出 #2 管脚
62:数字输出 2
Inv
IO输出 #3 管
63:数字输出 3
Inv
Step_Dir
Fwd2Rev2Fwd
Apply
Cancel

图 2.5: 差分输出口设置

第3章　运动控制系统的二次开发

AsuMotion是一种开放式系统，可以进行高可靠性的软件扩展，满足用户对其二次开发的需要。系统通用性强，软件的可移植性好，对机床制造商和用户的要求较低。

AsuMotion是USB总线九轴伺服/步进电机运动控制卡，它以高频率脉冲串形式输出，控制伺服/步进电机的运动。该卡能精确地控制所发出的脉冲频率（电机速度）、脉冲个数（电机转角）及脉冲频率变化率（电机加速度），它能满足步进电机的各种复杂的控制要求。可对电机进行位置控制、插补驱动、加速/减速等控制。具有圆弧、直线插补功能。它含有丰富的，功能齐全的软件库函数资源。在Windows9X/2000/XP/Win7/Window10环境下，运动控制器提供C语言函数库和Windows动态链接库，以实现复杂的控制功能。用户能够将这些控制函数与自己控制系统所需的数据处理、界面显示、用户接口等应用程序模块集成在一起，建造符合特定应用要求的控制系统，以适应各种应用领域的要求。使用该运动控制器，要求使用者具有C语言或Windows下使用动态链接库的编程经验。用户可直接使用本章介绍的设备驱动程序函数接口；以最大方便地使您在Visual C++、Visual C#及各种其他软件环境中使用本设备。

功能特点:

1. 独立9轴驱动，AsuMotion可以分别控制9个马达驱动轴的运动。每个轴都可以进行定速驱动，直线加/减速驱动，S曲线加/减速驱动等。9个轴的性能相同。
2. 高精确的速度控制
3. S曲线加/减速驱动
4. 9轴直线插补，可以同时进行9个轴的直线插补。
5. 3轴圆弧插补，可以进行空间圆弧插补。

正是由于 AsuMotion强大的开放式架构，精确的插补功能，使之应用范围十分广泛，在由步进电机和数字式伺服电机组成的基于PC机的运动控制系统中，都可以使用AsuMotion作为核心控制单元，例如：

1. 数控机床、加工中心、机器人等；
2. X-Y-Z 控制台；
3. 绘图仪、雕刻机、印刷机械；
4. 送料装置、云台；
5. 打标机、绕线机；
6. 医疗设备；
7. 包装机械、纺织机械；

等等。利用AsuMotion的动态链接库（DLL），开发者可以很快开发出Windows 平台下的运动控制系统。本控制卡的动态链接库是标准的Windows动态链接库，支持64bit和32bit开发，选用的开发工具应支持Windows标准的动态链接库调用。

以下介绍如何利用常用的开发工具Microsoft Visual C++开发基于Windows 平台的运动控制程序。

§ 3.1 用Visual C++开发控制程序

用户可以使用 Visual studio 2008 或更高版本，来进行Windows平台下运动控制系统开发。在 VC 中调用动态链接库DLL 中函数有两种方法：隐式调用和显式调用。

3.1.1 隐式调用

隐式调用需要如下文件：

1. DLL 函数声明头文件AsuMotionDevice.h
2. 编译连接时用的导入库文件AsuMotionDevicex64.lib 或者AsuMotionDevicex86.lib
3. 动态链接库文件或者AsuMotionDevicex64.dll或者AsuMotionDevicex86.dll

以上文件中请跟供应商联系获取。建立工程之后，在 VC 集成环境中点击“/project/settings…”菜单弹出“project settings”对话框。选“Link”选项卡，在“object/library modules”栏内输入导入库文件。对于64位操作系统平台，文件为AsuMotionDevicex64.lib。对于32位操作系统平台，文件为AsuMotionDevicex86.lib，单击“OK”按钮。在调用DLL 函数的源代码文件开始处包含 AsuMotionDevice.h 头文件。之后则可以按照调用内部函数一样调用DLL 函数。

3.1.2 显式调用

显式调用需要如下文件：

1. DLL 函数声明头文件AsuMotionDevice.h
2. 动态链接库文件AsuMotionDevice.dll

显式调用方法需要调用 Windows API 函数加载和释放动态链接库。方法如下：

1. 调用Windows API 函数LoadLibrary()动态加载DLL
2. 调用Windows API 函数GetProcAddress()取得将要调用的DLL 中函数的指针
3. 用函数指针调用DLL 中函数完成相应功能
4. 在程序结束时或不再使用DLL 中函数时，调用Windows API 函数 FreeLibrary()释放动态链接库

该方法比较烦琐。以上在两种方法均为 VC 中调用动态链接库函数的标准方法，若要获得更具体的调用方法和帮助，请参考微软Visual Stutio 开发文档MSDN 或相关VC 参考书籍中相应部分内容。

§ 3.2 AsuMotion定义的数据类型

3.2.1 函数错误类型AsuMotionError

为了保证AsuMotion函数执行的可靠性与最大可能的诊断函数执行状况，AsuMotion定义了专门的错误类型。对于C语言来说，利用enum定义。其定义如下：

```
typedef enum
{
    AsuMotion_Error_Ok=0,
    AsuMotion_Error_NullPointer=1,
    AsuMotion_Error=2,
    AsuMotion_True=3,
    AsuMotion_False=4,
}AsuMotionError;
```

表3.1列出了各种错误对应的一般意义，其具体意义将在各函数中分别说明。

表 3.1: 故障意义

故障号	故障类型	故障意义
0	AsuMotion_Error_Ok	正常
1	AsuMotion_Error_NullPointer	空指针
2	AsuMotion_Error	有错误发生
3	AsuMotion_True	真
4	AsuMotion_False	假

3.2.2 控制卡设备句柄AsuMotionDevice

其定义如下：

```
typedef void * AsuMotionDevice;
```

用于标识一个已经打开的AsuMotion设备。

3.2.3 位屏蔽类型AsuMotionBitMaskType

此类型定义主要用在可以同时操作多个位相关运动的函数，可以选择其中一个值，或者几个值的或。其定义如下：

```
1
2  typedef enum
3  {
4     AsuMotion_BitMask_00 = 0x00000001uL,
5     AsuMotion_BitMask_01 = 0x00000002uL,
6     AsuMotion_BitMask_02 = 0x00000004uL,
7     AsuMotion_BitMask_03 = 0x00000008uL,
8     AsuMotion_BitMask_04 = 0x00000010uL,
9     AsuMotion_BitMask_05 = 0x00000020uL,
10    AsuMotion_BitMask_06 = 0x00000040uL,
11    AsuMotion_BitMask_07 = 0x00000080uL,
12    AsuMotion_BitMask_08 = 0x00000100uL,
13    AsuMotion_BitMask_09 = 0x00000200uL,
14    AsuMotion_BitMask_10 = 0x00000400uL,
15    AsuMotion_BitMask_11 = 0x00000800uL,
16    AsuMotion_BitMask_12 = 0x00001000uL,
17    AsuMotion_BitMask_13 = 0x00002000uL,
18    AsuMotion_BitMask_14 = 0x00004000uL,
19    AsuMotion_BitMask_15 = 0x00008000uL,
20    AsuMotion_BitMask_16 = 0x00010000uL,
21    AsuMotion_BitMask_17 = 0x00020000uL,
22    AsuMotion_BitMask_18 = 0x00040000uL,
23    AsuMotion_BitMask_19 = 0x00080000uL,
24    AsuMotion_BitMask_20 = 0x00100000uL,
25    AsuMotion_BitMask_21 = 0x00200000uL,
26    AsuMotion_BitMask_22 = 0x00400000uL,
27    AsuMotion_BitMask_23 = 0x00800000uL,
28    AsuMotion_BitMask_24 = 0x01000000uL,
29    AsuMotion_BitMask_25 = 0x02000000uL,
30    AsuMotion_BitMask_26 = 0x04000000uL,
31    AsuMotion_BitMask_27 = 0x08000000uL,
32    AsuMotion_BitMask_28 = 0x10000000uL,
33    AsuMotion_BitMask_29 = 0x20000000uL,
34    AsuMotion_BitMask_30 = 0x40000000uL,
35    AsuMotion_BitMask_31 = 0x80000000uL
36
37 }AsuMotionBitMaskType;
```

3.2.4 轴屏蔽选取类型AsuMotionAxisMaskType

此类型定义主要用在可以同时操作多个轴相关运动的函数，可以选择其中一个值，或者几个值的或。其定义如下：

```
typedef enum
{
    AsuMotion_AxisMask_X=0x0001,
    AsuMotion_AxisMask_Y=0x0002,
    AsuMotion_AxisMask_Z=0x0004,
    AsuMotion_AxisMask_A=0x0008,
    AsuMotion_AxisMask_B=0x0010,
    AsuMotion_AxisMask_C=0x0020,
    AsuMotion_AxisMask_U=0x0040,
    AsuMotion_AxisMask_V=0x0080,
    AsuMotion_AxisMask_W=0x0100,
    AsuMotion_AxisMask_All = 0x01ff,
}AsuMotionAxisMaskType;
```

表 3.2: 多轴选择

程序使用的符号	AxisMask值	所选择的轴
AsuMotion_AxisMask_X	0x0001	X轴
AsuMotion_AxisMask_Y	0x0002	Y轴
AsuMotion_AxisMask_Z	0x0004	Z轴
AsuMotion_AxisMask_A	0x0008	A轴
AsuMotion_AxisMask_B	0x0010	B轴
AsuMotion_AxisMask_C	0x0020	C轴
AsuMotion_AxisMask_U	0x0040	U轴
AsuMotion_AxisMask_V	0x0080	V轴
AsuMotion_AxisMask_W	0x0100	W轴
AsuMotion_AxisMask_All	0x01FF	所有轴

3.2.5 轴索引选取类型AsuMotionAxisIndexType

此类型定义主要用在可以同时仅可操作单个轴相关运动的函数，只能选择其中一个值。具体参见表3.3 其定义如下：

```
typedef enum
{
    AsuMotion_AxisIndex_X = 0x0000,
    AsuMotion_AxisIndex_Y = 0x0001,
    AsuMotion_AxisIndex_Z = 0x0002,
    AsuMotion_AxisIndex_A = 0x0003,
    AsuMotion_AxisIndex_B = 0x0004,
    AsuMotion_AxisIndex_C = 0x0005,
```

```
    AsuMotion_AxisIndex_U = 0x0006,
    AsuMotion_AxisIndex_V = 0x0007,
    AsuMotion_AxisIndex_W = 0x0008
}AsuMotionAxisIndexType;
```

表 3.3: 单轴选择

程序使用的符号	AxisIndex值	所选择的轴
AsuMotion_AxisIndex_X	0x0000	X轴
AsuMotion_AxisIndex_Y	0x0001	Y轴
AsuMotion_AxisIndex_Z	0x0002	Z轴
AsuMotion_AxisIndex_A	0x0003	A轴
AsuMotion_AxisIndex_B	0x0004	B轴
AsuMotion_AxisIndex_C	0x0005	C轴
AsuMotion_AxisIndex_U	0x0006	U轴
AsuMotion_AxisIndex_V	0x0007	V轴
AsuMotion_AxisIndex_W	0x0008	W轴

3.2.6 坐标轴数据AsuMotionAxisData

其定义如下：

```
typedef struct
{
    double x, y, z;
    double a, b, c;
    double u, v, w;
}AsuMotionAxisData;
```

由于AsuMotion支持9个轴的插补，因此其采用的坐标位置含有九个元素。

3.2.7 坐标轴数据AsuMotionAxisDataInt

其定义如下：

```
typedef struct
{
    int x, y, z;
    int a, b, c;
    int u, v, w;
}AsuMotionAxisDataInt;
```

3.2.8 运动函数停止类型AsuMotionStopType

其定义如下：

```
typedef enum
{
    AsuMotion_Stop_Type_Stop = 0,
    AsuMotion_Stop_Type_Exact = 1,
    AsuMotion_Stop_Type_Parabolic = 2,
    AsuMotion_Stop_Type_Tangent = 3,
}AsuMotionStopType;
```

目前AsuMotion支持的停止类型。

3.2.9 笛卡尔坐标类型AsuMotionCartesian

笛卡儿坐标系（Cartesian coordinate system），也称直角坐标系，是一种正交坐标系。二维的直角坐标系是由两条相互垂直、0点重合的数轴构成的。在平面内，任何一点的坐标是根据数轴上对应的点的座标设定的。在平面内，任何一点与坐标的对应关系，类似于数轴上点与坐标的对应关系。在原本的二维直角坐标系，再添加一个垂直于x轴，y轴的坐标轴，称为z轴。假若，这三个坐标轴满足右手定则，则可得到三维的直角坐标系。其定义如下：

```
typedef struct
{
    double x, y, z;
} AsuMotionCartesian;
```

3.2.10 直接规划坐标增量类型AsuMotionDirectMoveInc

其定义如下：

```
typedef struct
{
    short x, y, z;
    short a, b, c;
    short u, v, w;
}AsuMotionDirectMoveInc;
```

3.2.11 控制卡回原点模式AsuMotionHomingType

其定义如下：

```
typedef enum
{
    AsuMotion_Homing_NoMoving = 0x0000,
    AsuMotion_Homing_AtSwitch = 0x0001,
    AsuMotion_Homing_LeaveSwitch = 0x0002,
    AsuMotion_Homing_MultipleHome = 0x0003,
}AsuMotionHomingType;
```

§ 3.3 AsuMotion控制卡设备操作函数

3.3.1 获取设备数量

函数原型：

```
int GetDeviceNum(void);
```

参数：无
返回值：返回当前电脑共插入AsuMotion控制卡的数量。
用法：一般情况下程序应该最先调用此函数，获取当前设备数量。当然，如果明确知道当前有几个设备可以不调用此函数。

3.3.2 获取设备序列号信息

函数原型：

```
AsuMotionError GetDeviceInfo(int Num,char SerialString[]);
```

参数：

Num：如果通过调用GetDeviceNum后知道当前设备数量，那么可以查询第0个设备、第1个设备、以及第n个设备，此处的Num就是设备序号减一。

SerialString：保存设备序列号的缓冲区。

返回值：

AsuMotion_True：当获取成功时。

AsuMotion_False：当获取失败时。

3.3.3 打开设备

函数原型：

```
AsuMotionDevice AsuMotionOpen(int Num);
```

参数：

Num：设备序号减一。

返回值：

AsuMotionDevice：当获取成功时。

3.3.4 关闭设备

函数原型：

```
AsuMotionError AsuMotionClose(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotionError：目前直接返回AsuMotion_True

用于关闭一个曾经打开的设备。

3.3.5 设备初始化为默认

函数原型：

```
AsuMotionError AsuMotionConfigDeviceDefault(AsuMotionDevice AsuMotion)
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_True：配置成功后

AsuMotion_False：配置不成功

§ 3.4 直接控制规划相关函数

3.4.1 添加一次坐标增量

坐标增量为1毫秒的坐标增量。函数原型：

```
AsuMotionDirectPlanAddMoveIncrease(AsuMotionDevice AsuMotion,
      AsuMotionDirectMoveInc const*const inc
   );
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

inc：一个指向AsuMotionDirectMoveInc的指针，此指针指向的对象包含需要设置的坐标增量。此增量为要设置的脉冲数和分频系数的乘积。

返回值：

AsuMotion_Buffer_Full：当前缓冲区满，等待一会再添加。

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.4.2 添加一次IO变化

IO变化为1毫秒的变化。函数原型：

```
AsuMotionError AsuMotionDirectPlanChangeIO(AsuMotionDevice AsuMotion,
   unsigned short DIO[],
   unsigned short AIO[]
)
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

DIO：需要在当前直接规划阶段设置的数字量输出，低位对齐。每位对应一个数字量输出。

AIO：需要在当前直接规划阶段设置的模拟量输出。控制卡为12位DA输出，即4095对应满量程输出。

返回值：

AsuMotion_Buffer_Full：当前缓冲区满，等待一会再添加。

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.4.3 将缓冲区中添加的直接规划刷新

一般情况下，在完成所有的直接规划后，需要调用此函数，来完成缓冲区内的IO，以及坐标增量的规划。如果不调用此函数，那么有可能最后的三次规划不能输出。函数原型：

```
AsuMotionError AsuMotionDirectPlanFlush(AsuMotionDevice AsuMotion)
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Buffer_Full：当前缓冲区满，等待一会再添加。

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

§ 3.5 PC运动规划相关函数

3.5.1 设置当前坐标

函数原型：

```
AsuMotionError AsuMotionSetCurrentPostion(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisData const*const Position);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Position：一个指向AsuMotionAxisData的指针，此指针指向的对象包含需要设置的当前坐标。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.5.2 暂停当前运动

函数原型：

```
AsuMotionError AsuMotionPause(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.5.3 恢复暂停的运动

函数原型：

```
AsuMotionError AsuMotionResume(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.5.4 放弃当前的运动规划

函数原型：

```
AsuMotionError AsuMotionAbort(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.5.5 设置停止类型

函数原型：

```
AsuMotionError AsuMotionSetStopType(AsuMotionDevice AsuMotion,↩
    AsuMotionStopType type,double tolerance);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

type：指定一个停止类型，共有四种停止类型，参见AsuMotionStopType定义

tolerance：指定停止时所能容忍的误差。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

3.5.6 添加直线插补规划

函数原型：

```
AsuMotionError AsuMotionAddLine(AsuMotionDevice AsuMotion,
    AsuMotionAxisData const*const end,
    double vel,
    double ini_maxvel,
    double acc);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

end：指定当前直线的终点坐标。参见AsuMotionAxisData定义

vel：指定当前直线运行的速度。

ini_maxvel：由上一段直线或者曲线转入当前直线或者曲线时，所允许的最大速度。

acc：当前直线运行的加速度

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_PCPlan：当前状态下不能进行PC的规划，因为前面提交的其他操作还未完成。

AsuMotion_Buffer_Full：当前状态下不能添加PC的规划，因为缓冲区已经满了。

3.5.7 添加空间圆弧插补规划

在介绍圆弧插补前，我们先介绍几个常识。球坐标是三维坐标系的一种，用以确定三维空间中点、线、面以及体的位置，它以坐标原点为参考点，由方位角、仰角和距离构成。在数学里，球座标系（英语：Spherical coordinate system）是一种利用球座标（ r , θ , ϕ ）表示一个点在三维空间的位置的三维正交座标系。

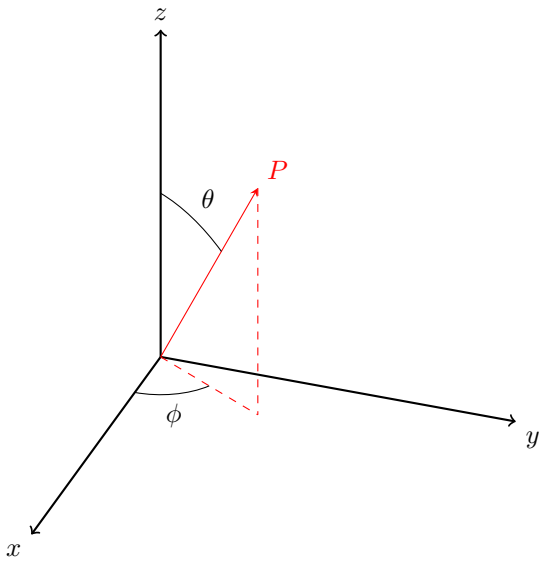


图 3.1: 球坐标系

图 3.1显示了球座标的几何意义：原点与点 P 之间的径向距离 r，原点到点P的连线与正z轴之间的天顶角 θ ，以及原点到点 P 的连线，在 xy平面的投影线，与正x轴之间的方位角 ϕ 。如想要用球座标，找出点 P 在空间的地点，可按照以下步骤：

1. 从原点往正z轴移动r单位，
2. 用右手定则，大拇指往y轴指,x轴与z轴朝其他手指的指向旋转θ角值，
3. 用右手定则，大拇指往z轴指，x轴与y轴朝其他手指的指向旋转ϕ角值。

在介绍AsuMotion运动控制器中的空间圆弧前，我们首先定义法线。法线定义为空间中经过两点的一条有方向的矢量，矢量的起点为坐标原点，终点坐标为normal。为保证法线的存在，normal必须设定为不是原点的一点。一旦定义了法线，那么与法线垂直且经过center的平面作为球座标系的xy平面，与法线平行且方向相同，同时又经过center的向量作为球座标系z轴。这样center将成为球座标系的原点。

了解球坐标系后，理解空间圆弧的插补就很容易了，假设上一次规划的终点坐标也就是当前圆弧插补的起点坐标为（r_s, θ_s, ϕ_s）。当前圆弧插补的终点坐标为（r_e, θ_e, ϕ_e）。其规划的曲线为三维阿基米德螺线，在xy平面的投影为阿基米德螺线，亦称“等速螺线”，它的极坐标方程为:$\rho = a\phi$，这种螺线的每条臂的间距永远等于$2\pi a$。三维阿基米德螺线在z轴的轨迹满足$z = b\phi$。三维阿基米德螺线的笛卡尔坐标方程为：

$$\begin{aligned} x &= \rho\cos(\phi) \\ y &= \rho\sin(\phi) \\ z &= b\phi \end{aligned} \tag{3.1}$$

其中ρ为半径, a，b为参数，ϕ为自变量。

对圆弧的查补来说，$\phi \in [\phi_s, 2n\pi + \phi_e]$，其中$n$为圆弧插补的圈数，对应函数中的turn参数。

$$\rho = r_s\sin(\theta_s) + \frac{r_e\sin(\theta_e) - r_s\sin(\theta_s)}{2n\pi + \phi_e - \phi_s}(\phi - \phi_s) \tag{3.2}$$

$$z = r_s\cos(\theta_s) + \frac{r_e\cos(\theta_e) - r_s\cos(\theta_s)}{2n\pi + \phi_e - \phi_s}(\phi - \phi_s) \tag{3.3}$$

一个典型的圆弧插补如图 3.2所示，由于圆弧插补的运动轨迹为螺纹线，因此经过恰当的设置后，可以用来直接加工螺纹。如果起点和终点到经过center的法线距离相同，那么运动轨迹为圆柱上的某些点。如果起点和终点都在经过center且垂直于法线的平面上，那么运动轨迹为圆上的一段曲线。

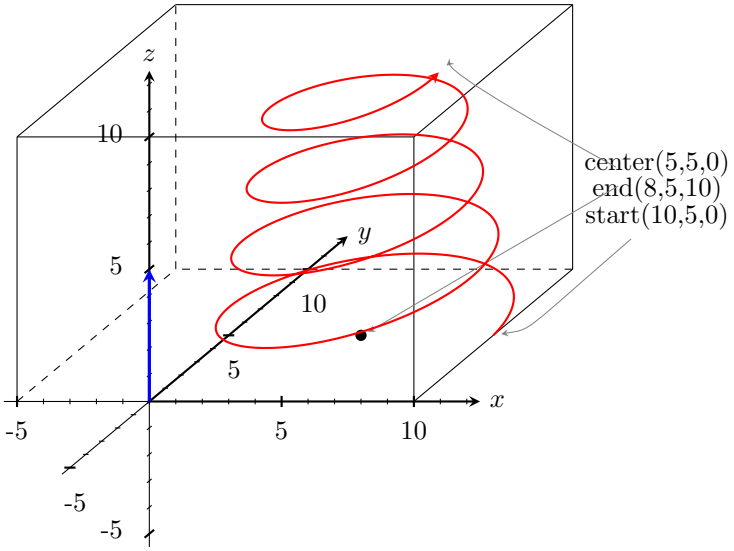


图 3.2: 三维阿基米德螺线：turn=4，normal：(0,0,5)

函数原型：

```
1 AsuMotionError AsuMotionAddCircle(AsuMotionDevice AsuMotion,
```

```
2       AsuMotionAxisData const*const end,
3       AsuMotionCartesian const*const center,
4       AsuMotionCartesian const*const normal,
5       int turn,
6       double vel,
7       double ini_maxvel,
8       double acc
9       );
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

end：指定当前圆弧插补的终点坐标。参见AsuMotionAxisData定义

center：指定当前圆弧插补定义的球坐标系原点。参见AsuMotionCartesian定义

normal：对于空间圆弧插补来说，定义出法线唯一方法为：起点为原点（0，0，0）终点坐标为normal。

turn：指定当前圆弧插补的圈数。

vel：指定当前圆弧运行的速度。

ini_maxvel：由上一段直线或者曲线转入当前圆弧曲线时，所允许的最大速度。

acc：当前圆弧运行的加速度

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_PCPlan：当前状态下不能进行PC的规划，因为前面提交的其他操作还未完成。

AsuMotion_Buffer_Full：当前状态下不能添加PC的规划，因为缓冲区已经满了。

3.5.7.1 XY平面圆弧插补，法线为Z轴正方向，0圈

如图3.3所示，法线为蓝色矢量，朝向Z轴正方向，起始坐标为（10，5，0），终点坐标为（0，5，0），中心坐标为（5，5，0），由于所设定的圈数为0，因此AsuMotion的插补曲线为在XY平面上的半个圆，运动方向如图中箭头所示。

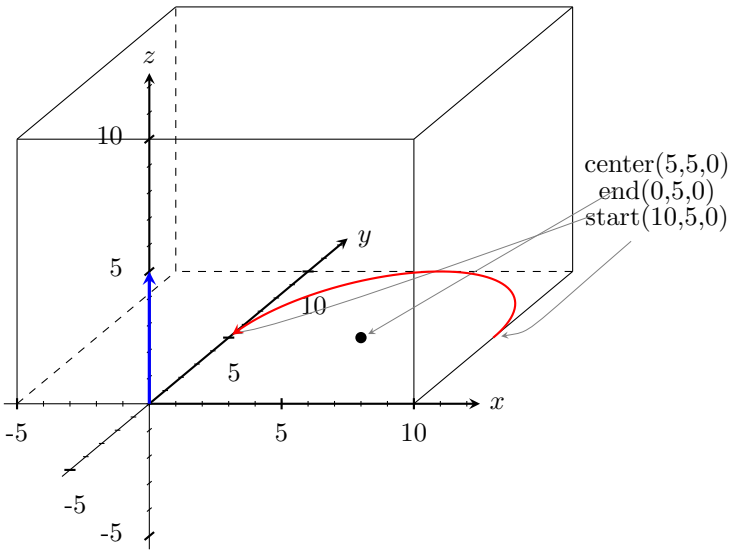


图 3.3: XY平面圆弧插补例子：turn=0，normal：(0,0,5)

3.5.7.2 XY平面圆弧插补，法线为Z轴负方向，0圈

如图3.4所示，法线为蓝色矢量，朝向Z轴负方向，起始坐标为（10，5，0），终点坐标为（0，5，0），中心坐标为（5，5，0），由于所设定的圈数为0，因此AsuMotion的插补曲线为在XY平面上的半个圆，运动方向如图中箭头所示。

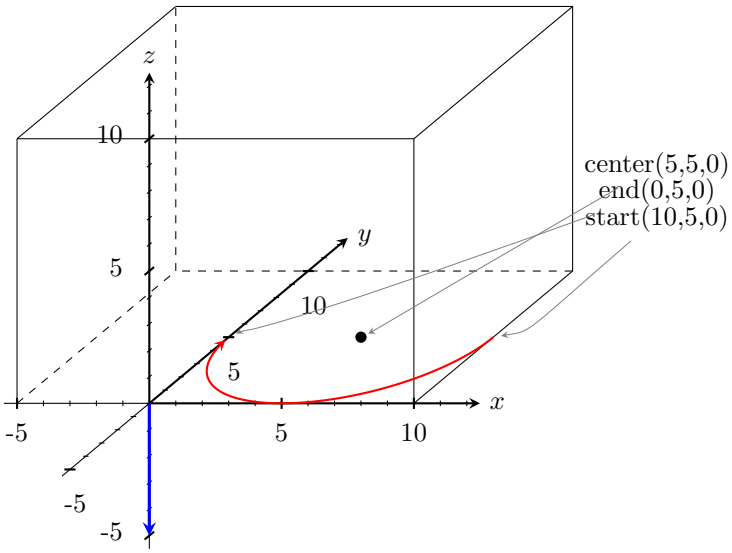


图 3.4: XY平面圆弧插补例子：turn=0，normal：(0,0,-5)

3.5.7.3 XY平面圆弧插补，法线为Z轴正方向，1圈

如图3.5所示，法线为蓝色矢量，朝向Z轴正方向，起始坐标为（10，5，0），终点坐标为（0，5，0），中心坐标为（5，5，0），由于所设定的圈数为1，因此AsuMotion将在XY平面上运动整个圆后，再运动半个圆，运动方向如图中箭头所示。

3.5.7.4 XY平面圆弧插补，法线为Z轴负方向，1圈

如图3.6所示，法线为蓝色矢量，朝向Z轴正方向，起始坐标为（10，5，0），终点坐标为（0，5，0），中

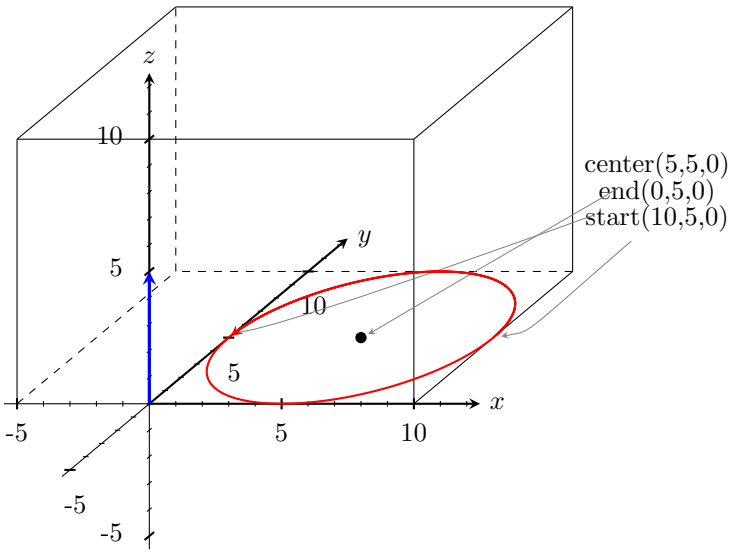


图 3.5: XY平面圆弧插补例子：turn=1，normal：(0,0,5)

心坐标为（5，5，0），由于所设定的圈数为1，因此AsuMotion将在XY平面上运动整个圆后，再运动半个圆，运动方向如图中箭头所示。

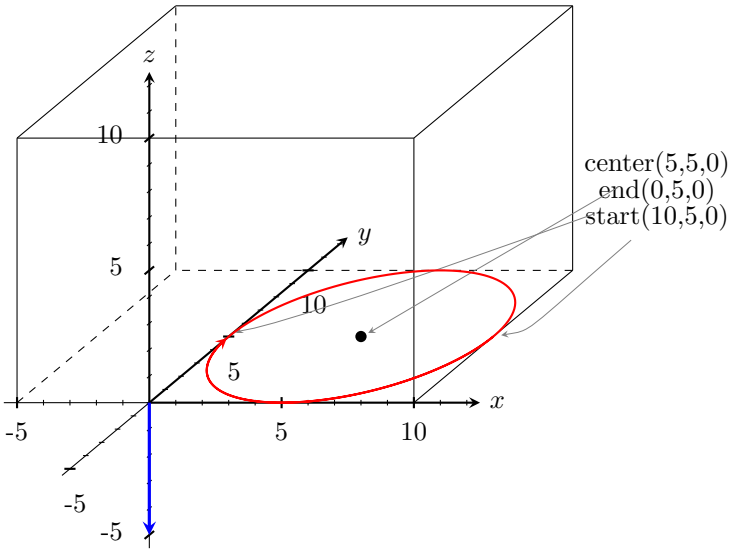


图 3.6: XY平面圆弧插补例子：turn=1，normal：(0,0,-5)

3.5.7.5 空间圆弧插补，法线为Z轴正方向，0圈

如图3.8所示，法线为蓝色矢量，朝向Z轴正方向，起始坐标为（10，5，0），终点坐标为（0，5，5），中心坐标为（5，5，0），由于所设定的圈数为0，因此AsuMotion规划运动的投影将在XY平面上运动半个圆，运动方向如图中箭头所示。

3.5.7.6 空间圆弧插补，法线为Z轴负方向，1圈

如果设置为多圈，那么此种模式可以直接作为螺纹的加工。如图3.8所示，法线为蓝色矢量，朝向Z轴正方向，起始坐标为（10，5，0），终点坐标为（0，5，5），中心坐标为（5，5，0），由于所设定的圈数为1，因此AsuMotion规划运动的投影将在XY平面上运动整个圆后，再运动半个圆，运动方向如图中箭头所示。

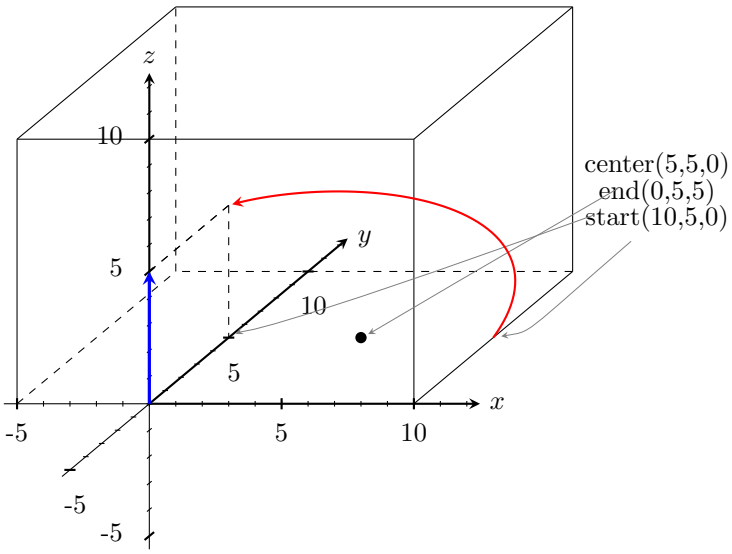


图 3.7: 空间圆弧插补例子：turn=0，normal：(0,0,5)

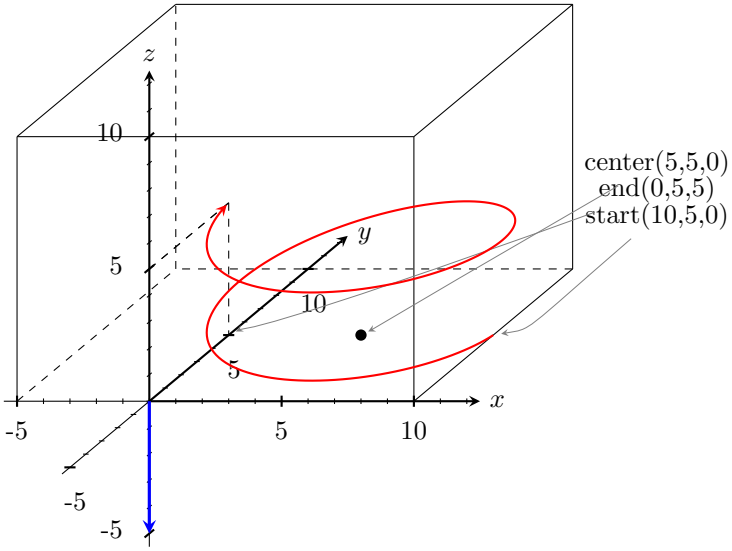


图 3.8: 空间圆弧插补例子：turn=1，normal：(0,0,-5)

3.5.8 添加同步IO直线插补规划

在某些应用场合中，要求输出IO或者输出模拟量与当前的坐标位置精确相关，这就要求输出IO必须和某一段规划同步输出，此时添加直线需要利用此函数。函数原型：

```
AsuMotionError AsuMotionAddLineWithSyncIO(AsuMotionDevice AsuMotion,
    AsuMotionAxisData const*const end,
    double vel,
    double ini_maxvel,
    double acc,
```

```
6       unsigned short DIO[],
7       unsigned short AIO[]);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

end：指定当前直线的终点坐标。参见AsuMotionAxisData定义

vel：指定当前直线运行的速度。

ini_maxvel：由上一段直线或者曲线转入当前直线或者曲线时，所允许的最大速度。

acc：当前直线运行的加速度

DIO：需要在当前规划阶段设置的数字量输出，低位对齐。每位对应一个数字量输出。

AIO：需要在当前规划阶段设置的模拟量输出。控制卡为12位DA输出，即4095对应满量程输出。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_PCPlan：当前状态下不能进行PC的规划，因为前面提交的其他操作还未完成。

AsuMotion_Buffer_Full：当前状态下不能添加PC的规划，因为缓冲区已经满了。

3.5.9 添加同步IO空间圆弧插补规划

函数原型：

```
1  AsuMotionError AsuMotionAddCircleWithSyncIO(AsuMotionDevice AsuMotion,
2    AsuMotionAxisData const*const end,
3    AsuMotionCartesian const*const center,
4    AsuMotionCartesian const*const normal,
5    int turn,
6    double vel,
7    double ini_maxvel,
8    double acc,
9    unsigned short DIO[],
10   unsigned short AIO[]
11   )
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

end：指定当前圆弧的终点坐标。参见AsuMotionAxisData定义

center：指定当前圆弧插补定义的球坐标系原点。参见AsuMotionCartesian定义

normal：对于空间圆弧插补来说，定义出法线唯一方法为：起点为原点（0，0，0）终点坐标为normal。

turn：指定当前圆弧插补的圈数。

vel：指定当前圆弧运行的速度。

ini_maxvel：由上一段直线或者曲线转入当前圆弧曲线时，所允许的最大速度。

acc：当前圆弧运行的加速度

DIO：需要在当前规划阶段设置的数字量输出，低位对齐。每位对应一个数字量输出。

AIO：需要在当前规划阶段设置的模拟量输出。控制卡为12位DA输出，即4095对应满量程输出。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_PCPlan：当前状态下不能进行PC的规划，因为前面提交的其他操作还未完成。

AsuMotion_Buffer_Full：当前状态下不能添加PC的规划，因为缓冲区已经满了。

§ 3.6 运动控制卡规划相关的函数

运动控制卡规划的运动主要用在一些特殊场合，而且其使用有一定的限定，比如必须在运动开始前设置好速度，加速度，在运动过程中是不能修改的。另外由控制卡规划的直线运动并不是联动的。

3.6.1 常速运行

此函数主要用在轴输出为主轴时，让电机以一个恒定的速度持续不停的运行，直到有命令结束。在运行此函数前，需要预先设定运行需要的加速度和最大速度。函数原型：

```
1 AsuMotionError AsuMotionMoveAtConstSpeed(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisMaskType AxisMask);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

AxisMask：配置当前需要运行的轴。其值可以选择表3.2的值，或者表3.2值的或。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_CardPlan：当前状态下不能进行运动控制卡的规划，因为前面提交的其他操作还未完成。

例程：下面代码仅让X轴以常速运行

```
1   AsuMotionSetAccelaration(AsuMotion, &Accelaration);
2   AsuMotionSetMaxSpeed(AsuMotion, &MaxSpeed);
3   AsuMotionMoveAtConstSpeed(AsuMotion,AsuMotion_AxisMask_X);
```

下面代码仅让X轴，Z轴以常速运行

```
1   AsuMotionSetAccelaration(AsuMotion, &Accelaration);
2   AsuMotionSetMaxSpeed(AsuMotion, &MaxSpeed);
3   AsuMotionMoveAtConstSpeed(AsuMotion, (AsuMotionAxisMaskType)(↩
        AsuMotion_AxisMask_X | AsuMotion_AxisMask_Z));
```

3.6.2 点动运行

此函数主要用在手动单独点动某个轴。其停止需要调用以下三个停止函数之一

1. AsuMotionCardPlanStop
2. AsuMotionCardPlanStopAll
3. AsuMotionEStop

这三种停止方式的区别，请参考 3.8中介绍的相关函数。函数原型：

```
1 AsuMotionError AsuMotionJogOn(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisIndexType Axis, AsuMotionAxisData const*const PositionGiven)↩
    ;
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Axis：配置当前需要运行的轴。其值可以选择表3.3的值。

PositionGiven：给定一个点动运行时，机器的目标位置。此位置为相对位置。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_CardPlan：当前状态下不能进行运动控制卡的规划，因为前面提交的其他操作还未完成。

3.6.3 绝对位置移动

此函数主要用在将单个轴或多个轴运动到指定位置。其停止需要调用以下三个停止函数之一

1. AsuMotionCardPlanStop
2. AsuMotionCardPlanStopAll
3. AsuMotionEStop

这三种停止方式的区别，请参考 3.8中介绍的相关函数。函数原型：

```
AsuMotionError AsuMotionMoveAbsolute(AsuMotionDevice AsuMotion,
      AsuMotionAxisMaskType AxisMask,
      AsuMotionAxisData const*const PositionGiven);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

AxisMask：配置当前需要运行的轴。其值可以选择表3.2的值，或者表3.2值的或。

PositionGiven：给定一个点动运行时，机器的目标位置。

返回值：

AsuMotion_Error：配置不成功

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_CardPlan：当前状态下不能进行运动控制卡的规划，因为前面提交的其他操作还未完成。

§ 3.7 运动控制卡状态获取相关的函数

3.7.1 输入口状态获取

此函数用于获取AsuMotion的16个输入IO和手轮IO的状态。函数原型：

```
1 AsuMotionError AsuMotionGetInputIO(AsuMotionDevice AsuMotion, unsigned short↩
      pInput[]);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

pInput：IO状态缓冲区，长度至少为2，函数调用成功后，这里面的值将为IO状态。

返回值：

目前直接返回AsuMotion_True

函数成功调用后，共有两个unsigned short类型的数被返回。其中pInput[0]中存储的是16个输入IO的状态。pInput[1]中存储的是16个手轮输入IO的状态。

pInput[0]和pInput[1]的各位与代表的意义见表3.4

表 3.4: 输入状态的各位对应的物理硬件状态

pInput[0]	状态值	输入端子管脚	pInput[1]	状态值	手轮功能	DB15管脚
位0	光耦输入0	1	位0	MPG_Pin02	A+	2
位1	光耦输入1	20	位1	MPG_Pin03	B+	3
位2	光耦输入2	2	位2	MPG_Pin05	OFF	5
位3	光耦输入3	19	位3	MPG_Pin15	X	15
位4	光耦输入4	3	位4	MPG_Pin14	Y	14
位5	光耦输入5	18	位5	MPG_Pin13	Z	13
位6	光耦输入6	4	位6	MPG_Pin09	A	9
位7	光耦输入7	17	位7	MPG_Pin06	X1	6
位8	光耦输入8	5	位8	MPG_Pin07	X10	7
位9	光耦输入9	16	位9	MPG_Pin08	X100	8
位10	光耦输入10	6	位10	MPG_Pin12	IO	12
位11	光耦输入11	15	位11	MPG_Pin04	IO	4
位12	光耦输入12	7	位12	MPG_Pin10	Reseved	10
位13	光耦输入13	14	位13	MPG_Pin11	GND	11
位14	光耦输入14	8	位14	MPG_Pin01	VCC5V	1
位15	光耦输入15	13	位15	MPG_Pin16	Earth	16

3.7.2 当前机器坐标位置脉冲数获取

函数原型：

```
AsuMotionError AsuMotionGetSteps(AsuMotionDevice AsuMotion, AsuMotionAxisDataInt *const Steps);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Steps：指向一个AsuMotionAxisDataInt的指针

返回值：

目前直接返回AsuMotion_True

3.7.3 当前各坐标轴最大速度获取

函数原型：

```
AsuMotionError AsuMotionGetMaxSpeed(AsuMotionDevice AsuMotion,
AsuMotionAxisData *const MaxSpeed);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

MaxSpeed：最大速度指针。

返回值：

目前直接返回AsuMotion_True

3.7.4 光滑系数获取

关于光滑系数的解释参见 2.3.1

函数原型：

```
AsuMotionError AsuMotionGetSmoothCoff(AsuMotionDevice AsuMotion, unsigned int *pSmoothCoff);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

pSmoothCoff：系数缓冲区，长度至少为1。

返回值：

目前直接返回AsuMotion_True

3.7.5 单位距离脉冲数获取

函数原型：

```
AsuMotionError AsuMotionGetStepsPerUnit(AsuMotionDevice AsuMotion,
AsuMotionAxisDataInt *const StepsPerUnit)
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

StepsPerUnit：指向AsuMotionAxisDataInt的指针

返回值：

目前直接返回AsuMotion_True

3.7.6 查询当前运动卡是否处于运动状态中

函数原型：

```
AsuMotionError AsuMotionIsDone(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error_TURE：当前运动卡处于空闲状态

AsuMotion_Error_FALSE：当前运动卡处于运动状态

§ 3.8 停止运动函数

3.8.1 急停函数

不管当前处于什么状态，直接停止掉运动控制卡，一般在发生危险状况时，调用此函数。此函数调用后，前面所有的规划全部放弃掉，此后不可以继续当前运动。必须重新规划新的运动。函数原型：

```
AsuMotionError AsuMotionEStop(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error_TURE：配置成功后

AsuMotion_Error_FALSE：配置不成功

3.8.2 停止由运动控制卡规划的某个轴的运动

函数原型：

```
AsuMotionError AsuMotionStop(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisMaskType AxisMask);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

AxisMask：配置当前需要运行的轴。其值可以选择表3.2的值，或者表3.2值的或。

返回值：

AsuMotion_Error_TURE：配置成功后

AsuMotion_Error_FALSE：配置不成功

3.8.3 停止由运动控制卡规划的所有轴的运动

函数原型：

```
AsuMotionError AsuMotionStopAll(AsuMotionDevice AsuMotion);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

返回值：

AsuMotion_Error_TURE：配置成功后

AsuMotion_Error_FALSE：配置不成功

3.8.4 停止由PC规划的运动

请参考 3.5.2

§ 3.9 PC规划与运动控制卡规划共用的配置函数

3.9.1 配置机器坐标

函数原型：

```
AsuMotionError AsuMotionSetMachineCoordinate(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisMaskType AxisMask, AsuMotionAxisData const*const Position);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

AxisMask：配置当前需要运行的轴。其值可以选择表3.2的值，或者表3.2值的或。

Position：一个存储各轴机器坐标结构体的指针

返回值：

AsuMotion_Error_TURE：当前运动卡处于空闲状态

AsuMotion_Error_FALSE：当前运动卡处于运动状态

3.9.2 配置单位脉冲数

函数原型：

```
AsuMotionError AsuMotionSetStepsPerUnit(AsuMotionDevice AsuMotion,
      AsuMotionAxisDataInt const*const StepsPerUnit
      );
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

StepsPerUnit：单位脉冲数。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.9.3 配置工作偏移

函数原型：

```
AsuMotionError AsuMotionSetWorkOffset(AsuMotionDevice AsuMotion,
    AsuMotionAxisDataInt const*const WorkOffset
    );
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

WorkOffset：工作偏移。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.9.4 配置光滑系数和脉冲延时

函数原型：

```
AsuMotionError AsuMotionSetSmoothCoff(AsuMotionDevice AsuMotion,
    unsigned short DelayBetweenPulseAndDir,
    int SmoothCoff);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

DelayBetweenPulseAndDir：设置在采用脉冲加方向的输出方式时，为了避免驱动器的误判，需要保证脉冲的有效沿和方向的有效沿之间有充分的时间差。此参数用来设置这个时间差。

SmoothCoff：光滑系数。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.9.5 配置运动卡差分输出的信号映射

函数原型：

```
AsuMotionError AsuMotionSetDifferentialOutputMapping(
    AsuMotionDevice AsuMotion,
    INT8U FunSel[],
    AsuMotionBitMaskType NegMask
 )
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

FunSel：映射号数组，共16个元素，其值得设定将反应16个差分输出的内容，每个映射号对应的输出内容见表3.5。

NegMask：设定16个差分输出口是否反向输出。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

表 3.5: 差分输出映射表

映射号	输出内容	备注
0x00	Step of X Axis	
0x01	Step of Y Axis	
0x02	Step of Z Axis	
0x03	Step of A Axis	
0x04	Step of B Axis	
0x05	Step of C Axis	
0x06	Step of U Axis	
0x07	Step of V Axis	
0x08	Step of W Axis	
0x09	Step of S Axis	
0x0A	Reserve	
0x0B	Reserve	
0x0C	Reserve	
0x0D	Reserve	
0x0E	Reserve	
0x0F	Reserve	
0x10	Dir of X Axis	

0x11	Dir of Y Axis	
0x12	Dir of Z Axis	
0x13	Dir of A Axis	
0x14	Dir of B Axis	
0x15	Dir of C Axis	
0x16	Dir of U Axis	
0x17	Dir of V Axis	
0x18	Dir of W Axis	
0x19	Dir of S Axis	
0x1A	Reserve	
0x1B	Reserve	
0x1C	Reserve	
0x1D	Reserve	
0x1E	Reserve	
0x1F	Reserve	
0x20	A of X Axis	
0x21	A of Y Axis	
0x22	A of Z Axis	
0x23	A of A Axis	
0x24	A of B Axis	
0x25	A of C Axis	
0x26	A of U Axis	
0x27	A of V Axis	
0x28	A of W Axis	
0x29	A of S Axis	
0x2A	Reserve	
0x2B	Reserve	
0x2C	Reserve	
0x2D	Reserve	
0x2E	Reserve	
0x2F	Reserve	
0x30	B of X Axis	
0x31	B of Y Axis	
0x32	B of Z Axis	

0x33	B of A Axis	
0x34	B of B Axis	
0x35	B of C Axis	
0x36	B of U Axis	
0x37	B of V Axis	
0x38	B of W Axis	
0x39	B of S Axis	
0x3A	Reserve	
0x3B	Reserve	
0x3C	Reserve	
0x3D	Reserve	
0x3E	Reserve	
0x3F	Reserve	
0x40	CW of X Axis	
0x41	CW of Y Axis	
0x42	CW of Z Axis	
0x43	CW of A Axis	
0x44	CW of B Axis	
0x45	CW of C Axis	
0x46	CW of U Axis	
0x47	CW of V Axis	
0x48	CW of W Axis	
0x49	CW of S Axis	
0x4A	Reserve	
0x4B	Reserve	
0x4C	Reserve	
0x4D	Reserve	
0x4E	Reserve	
0x4F	Reserve	
0x50	CW of X Axis	
0x51	CW of Y Axis	
0x52	CW of Z Axis	
0x53	CW of A Axis	
0x54	CW of B Axis	

0x55	CW of C Axis	
0x56	CW of U Axis	
0x57	CW of V Axis	
0x58	CW of W Axis	
0x59	CW of S Axis	
0x5A	Reserve	
0x5B	Reserve	
0x5C	Reserve	
0x5D	Reserve	
0x5E	Reserve	
0x5F	Reserve	
0x60	Digital Out 0	
0x61	Digital Out 1	
0x62	Digital Out 2	
0x63	Digital Out 3	
0x64	Digital Out 4	
0x65	Digital Out 5	
0x66	Digital Out 6	
0x67	Digital Out 7	
0x68	Digital Out 8	
0x69	Digital Out 9	
0x6A	Digital Out 10	
0x6B	Digital Out 11	
0x6C	Digital Out 12	
0x6D	Digital Out 13	
0x6E	Digital Out 14	
0x6F	Digital Out 15	
0x70	Cpu Led	
0x71	Run Led	
0x72	MCU Digital Out 2	
0x73	MCU Digital Out 3	
0x74	MCU Digital Out 4	
0x75	MCU Digital Out 5	
0x76	MCU Digital Out 6	

0x77	MCU Digital Out 7	
0x78	MCU Digital Out 8	
0x79	MCU Digital Out 9	
0x7A	MCU Digital Out 10	
0x7B	MCU Digital Out 11	
0x7C	MCU Digital Out 12	
0x7D	MCU Digital Out 13	
0x7E	MCU Digital Out 14	
0x7F	MCU Digital Out 15	

3.9.6 配置运动卡数字量输入功能

函数原型：

```
AsuMotionError AsuMotionSetInputIOEngineDir(AsuMotionDevice AsuMotion,
    unsigned long long InputIOEnable,
    unsigned long long InputIONeg,
    unsigned char InputIOPin[]
  );
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

InputIOEnable：64bitMask，设置数字量输入使能。

InputIONeg：64bitMask，设置数字量输入电平使能。

InputIOPin：64个元素的数组，设置数字量输入的功能,请参见表2.2。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.9.7 配置运动卡模拟量输出和数字量输出

函数原型：

```
AsuMotionError AsuMotionSetOutput(
    AsuMotionDevice AsuMotion,
```

```
    INT16U Sync,
    INT16S AnalogOut[],
    INT16U DigitalOut[]
);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Sync：是否同步输出，在使用PC规划时，可以采用同步输出，使得外部时序能够严格保证。

AnalogOut：两个元素的数组，设置模拟量输出。

DigitalOut：两个元素的数组，设置数字量输出。

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

§ 3.10 回原点相关的参数配置函数

3.10.1 配置回原点的管脚

函数原型：

```
AsuMotionError AsuMotionSetHomingSignal(AsuMotionDevice AsuMotion, unsigned char InputIOPin[])
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

InputIOPin：一个包含9个元素数组，分别代表九个轴的回原点的信号选择，具体参见表 3.6，如果选择的值大于等于32，那么选择的信号为表 3.6的逻辑反，也就是说原来如果高电平有效，那么现在将变成低电平有效。

返回值：

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

表 3.6: 回原点信号选择

InputIOPin	选择的信号	InputIOPin	选择的信号
0	光耦输入0	16	手轮编码器A+
1	光耦输入1	17	手轮编码器B+
2	光耦输入2	18	手轮OFF
3	光耦输入3	19	手轮X轴
4	光耦输入4	20	手轮Y轴
5	光耦输入5	21	手轮Z轴
6	光耦输入6	22	手轮A轴
7	光耦输入7	23	手轮X1倍率
8	光耦输入0	24	手轮X10倍率
9	光耦输入1	25	手轮X100倍率
10	光耦输入2	26	Input16
11	光耦输入3	27	Input17
12	光耦输入4	28	Input18
13	光耦输入5	29	Input19
14	光耦输入6	30	Input20
15	光耦输入7	31	Input21

3.10.2 请求一次回原点

函数原型：

```
AsuMotionError AsuMotionGoHome(AsuMotionDevice AsuMotion,
    AsuMotionHomingType Type,
    AsuMotionAxisMaskType AxisMask,
    unsigned short Count,
    AsuMotionAxisData const*const Acceleration,
    AsuMotionAxisData const*const MaxSpeed)
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Type：设置回原点的方式，目前支持三种方式。

AxisMask：配置当前需要运行的轴。其值可以选择表3.2的值，或者表3.2值的或。

Count：设置回原点采用多次回原点时，回原点的次数。

Acceleration：一个存储各轴加速度的AsuMotionAxisData结构体的指针。

MaxSpeed：一个存储各轴速度的AsuMotionAxisData结构体的指针

返回值：

AsuMotion_Error_Ok：配置成功

AsuMotion_Device_Is_Null：参数AsuMotion为空指针，一般因为没有打开设备导致。

AsuMotion_CurrentState_Isnot_Idle：当前状态下不能进行回原点，因为前面提交的其他操作还未完成。

§ 3.11 运动控制卡规划相关的参数配置函数

3.11.1 配置运动卡规划运动的加速度

函数原型：

```
AsuMotionSetAccelaration(AsuMotionDevice AsuMotion, AsuMotionAxisData const*↩
    const Acceleration);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

Acceleration：一个存储各轴加速度的AsuMotionAxisData结构体的指针

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.11.2 配置运动卡规划运动的最大速度

函数原型：

```
AsuMotionError AsuMotionSetMaxSpeed(AsuMotionDevice AsuMotion,
      AsuMotionAxisData const*const MaxSpeed);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

MaxSpeed：一个存储各轴速度的AsuMotionAxisData结构体的指针

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.11.3 配置正向软限位

此函数对PC规划的运动不起效果，对运动卡规划的常速运动也不起效果。函数原型：

```
1 AsuMotionError AsuMotionSetSoftPositiveLimit(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisData const*const SoftPositiveLimit);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

SoftPositiveLimit：正向软限位坐标的结构体指针

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

3.11.4 配置反向软限位

此函数对PC规划的运动不起效果，对运动卡规划的常速运动也不起效果。函数原型：

```
1 AsuMotionError AsuMotionSetSoftNegtiveLimit(AsuMotionDevice AsuMotion, ↩
    AsuMotionAxisData const*const SoftNegtiveLimit);
```

参数：

AsuMotion：由函数AsuMotionOpen返回的设备句柄。

SoftNegtiveLimit：反向软限位坐标的结构体指针

返回值：

AsuMotion_Error_TURE：配置成功

AsuMotion_Error_FALSE：配置失败

§ 3.12 完整例程

3.12.1 PC规划添加直线

```
#include "stdafx.h"
#include<windows.h>
#include <stdio.h>
#include <conio.h>
#include "AsuMotionDevice.h"
unsigned short DIO[3];
unsigned short AIO[3];
AsuMotionError ret=AsuMotion_Error_Ok;
AsuMotionAxisData PoistionGiven = { 0,0,0,0,0,0,0,0,0 };
AsuMotionAxisDataInt StepsPerUnit = { 2000,2000,2000,2000,2000,2000,2000,2000,2000 };
AsuMotionAxisDataInt WorkOffset= { 0,0,0,0,0,0,0,0,0 };
AsuMotionAxisData Accelaration = { 20,20,20,20,20,20,20,20,20 };
AsuMotionAxisData MaxSpeed = { 60,60,60,60,60,60,60,60,60 };
AsuMotionAxisData Max = { 60,60,60,60,60,60,60,60,60 };
AsuMotionAxisData Min = { 2,2,2,2,2,2,2,2,2 };
AsuMotionDevice AsuMotion;
AsuMotionAxisDataInt fullSteps = { 0,0,0,0,0,0,0,0,0 };
unsigned short IO;
char Serial[100];
char Path[200];
unsigned short Digital[6];
short Analog[6];
unsigned short SpindlePos;
AsuMotionAxisData AsuMotionPos = { 0,0,0,0,0,0,0,0,0 };
unsigned char DiffFunSel[16] = {0x00,0x10,0x01,0x11,0x02,0x12,0x03,0x13,0x04,0x14,0x05,0x15,0x06,0x16,0x07,0x17};

int _tmain(int argc, _TCHAR* argv[])
{
   int NumofAsuMotion=0;
   NumofAsuMotion=GetDeviceNum();
   if(NumofAsuMotion>0)
   {
      printf("Find %d AsuMotionDevice\n",NumofAsuMotion);
   }
   else
   {
      printf("No AsuMotionDevice was found");
      Sleep(1000);
      return 0;
   }
   for (int i=0;i<NumofAsuMotion;i++)
```

```
42    {
43        if (GetDeviceInfo(i,Serial))
44        {
45            printf("AsuMotion  %d    :Serial=%s\n",i,Serial);
46        }
47        else
48        {
49
50        }
51    }
52    if (NumofAsuMotion == 1)
53    {
54        NumofAsuMotion = 0;
55    }
56    else
57    {
58        printf("Which Device would you like to open?\n");
59        scanf_s("%d", &NumofAsuMotion);
60    }
61
62    AsuMotion=AsuMotionOpen(NumofAsuMotion);
63    if (!AsuMotion)
64    {
65        printf("AsuMotion Open Wrong!\n");
66        return 0;
67    }
68    else
69    {
70        printf("AsuMotion Open successes!\n");
71    }
72    ret=AsuMotionConfigDeviceDefault(AsuMotion);
73
74    AsuMotionStopAll(AsuMotion);
75
76    while (ret== AsuMotion_False)
77    {
78        ret = AsuMotionIsDone(AsuMotion);
79    }
80
81    if (!AsuMotionSetSmoothCoff(AsuMotion,
82        100,
83        64))
84    {
```

```
85      return 2;
86  }
87
88  if (!AsuMotionSetStepsPerUnit(AsuMotion,
89     &StepsPerUnit))
90  {
91      return 3;
92  }
93  if (!AsuMotionSetWorkOffset(AsuMotion,
94     &WorkOffset))
95  {
96      return 4;
97  }
98  AsuMotionSetDifferentialOutputMapping(
99      AsuMotion,
100     DiffFunSel,
101     (AsuMotionBitMaskType)0
102     );
103
104 AsuMotionSetAccelaration(AsuMotion, &Accelaration);
105 AsuMotionSetMaxSpeed(AsuMotion, &MaxSpeed);
106 AsuMotionStopAll(AsuMotion);
107
108 ret = AsuMotionIsDone(AsuMotion);
109 while (ret == AsuMotion_False)
110 {
111     ret = AsuMotionIsDone(AsuMotion);
112     AsuMotionGetSteps(AsuMotion, &fullSteps);
113
114     printf("aaax:%10.3f;y:%10.3f;z:%10.3f;a:%10.3f;b:%10.3f;c:%10.3f;u↩
        :%10.3f;v:%10.3f;w:%10.3f\n",
115         (float)fullSteps.x / StepsPerUnit.x,
116         (float)fullSteps.y / StepsPerUnit.y,
117         (float)fullSteps.z / StepsPerUnit.z,
118         (float)fullSteps.a / StepsPerUnit.a,
119         (float)fullSteps.b / StepsPerUnit.b,
120         (float)fullSteps.c / StepsPerUnit.c,
121         (float)fullSteps.u / StepsPerUnit.u,
122         (float)fullSteps.v / StepsPerUnit.v,
123         (float)fullSteps.w / StepsPerUnit.w
124         );
125     Sleep(1000);
126 }
```

```
127
128
129
130
131     AsuMotionSetMachineCoordinate(AsuMotion,&PoistionGiven);
132
133     AsuMotionSetCurrentPostion(AsuMotion,&AsuMotionPos);
134     AsuMotionPos.x=300;
135     AsuMotionPos.y=400;
136     AsuMotionPos.z=500;
137     AsuMotionPos.a=600;
138     AsuMotionPos.b=700;
139     AsuMotionPos.c=800;
140     AsuMotionPos.u = 500;
141     AsuMotionPos.v = 600;
142     AsuMotionPos.w = 700;
143     AsuMotionAddLine(AsuMotion,&AsuMotionPos,200,5000,50);
144     AsuMotionPos.x = -300;
145     AsuMotionPos.y = -400;
146     AsuMotionPos.z = -500;
147     AsuMotionPos.a = -600;
148     AsuMotionPos.b = -700;
149     AsuMotionPos.c = -800;
150     AsuMotionPos.u = -500;
151     AsuMotionPos.v = -600;
152     AsuMotionPos.w = -700;
153     DIO[0] = 0x0701;
154     AsuMotionAddLineWithSyncIO(AsuMotion, &AsuMotionPos, 200, 5000, 50, DIO, ↩
            AIO);
155     DIO[0] = 0x0202;
156     AsuMotionPos.w = -202;
157     AsuMotionAddLineWithSyncIO(AsuMotion, &AsuMotionPos, 200, 5000, 50, DIO, ↩
            AIO);
158     DIO[0]= 0x503;
159     AsuMotionPos.w = 500;
160     AsuMotionAddLineWithSyncIO(AsuMotion, &AsuMotionPos, 200, 5000, 50, DIO, ↩
            AIO);
161     AsuMotionAddLineWithSyncIO(AsuMotion, &AsuMotionPos, 200, 5000, 50, DIO, ↩
            AIO);
162     printf("Press any key to exit\n");
163     do
164     {
165         AsuMotionGetOutputIO(AsuMotion, DIO);
```

```
166         AsuMotionGetSteps(AsuMotion, &fullSteps);
167         printf("x:%10.3f;y:%8.3f;z:%8.3f;a:%8.3f;b:%10.3f;c:%10.3f;u:%10.3f;v↩
                :%10.3f;w:%10.3f:%04x\r",
168             (float)fullSteps.x / StepsPerUnit.x,
169             (float)fullSteps.y / StepsPerUnit.y,
170             (float)fullSteps.z / StepsPerUnit.z,
171             (float)fullSteps.a / StepsPerUnit.a,
172             (float)fullSteps.b / StepsPerUnit.b,
173             (float)fullSteps.c / StepsPerUnit.c,
174             (float)fullSteps.u / StepsPerUnit.u,
175             (float)fullSteps.v / StepsPerUnit.v,
176             (float)fullSteps.w / StepsPerUnit.w,
177             DIO[0]
178             );
179 
180         Sleep(100);
181     } while (!_kbhit());
182 
183     AsuMotionClose(AsuMotion);
184     return 0;
185 }
```

3.12.2 控制卡规划常速运动

```
1  #include "stdafx.h"
2  #include<windows.h>
3  #include <stdio.h>
4  #include <conio.h>
5  #include "AsuMotionDevice.h"
6  AsuMotionError ret=AsuMotion_Error_Ok;
7  AsuMotionAxisData PoistionGiven = { 0,0,0,0,0,0,0,0,0 };
8  AsuMotionAxisDataInt StepsPerUnit = { ↩
       2000,2000,2000,2000,2000,2000,2000,2000,2000 };
9  AsuMotionAxisDataInt WorkOffset= { 0,0,0,0,0,0,0,0,0 };
10 AsuMotionAxisData Accelaration = { 20,20,20,20,20,20,20,20,20 };
11 AsuMotionAxisData MaxSpeed = { 60,60,60,60,60,60,60,60,60 };
12 AsuMotionAxisData Max = { 60,60,60,60,60,60,60,60,60 };
13 AsuMotionAxisData Min = { 2,2,2,2,2,2,2,2,2 };
14 AsuMotionDevice AsuMotion;
15 AsuMotionAxisDataInt fullSteps = { 0,0,0,0,0,0,0,0,0 };
16 unsigned short IO;
```

```
17 char Serial[100];
18 char Path[200];
19 unsigned short Digital[6];
20 short Analog[6];
21 unsigned short SpindlePos;
22 AsuMotionAxisData AsuMotionPos = { 0,0,0,0,0,0,0,0,0 };
23 unsigned char DiffFunSel[16] = {0x00,0x10,0x01,0x11,0x02,0x12,0x03,0x13,0x04↩
    ,0x14,0x05,0x15,0x06,0x16,0x07,0x17};
24
25 int _tmain(int argc, _TCHAR* argv[])
26 {
27    int NumofAsuMotion=0;
28    NumofAsuMotion=GetDeviceNum();
29    if(NumofAsuMotion>0)
30    {
31       printf("Find %d AsuMotionDevice\n",NumofAsuMotion);
32    }
33    else
34    {
35       printf("No AsuMotionDevice was found");
36       Sleep(1000);
37       return 0;
38    }
39    for (int i=0;i<NumofAsuMotion;i++)
40    {
41       if (GetDeviceInfo(i,Serial))
42       {
43          printf("AsuMotion  %d   :Serial=%s\n",i,Serial);
44       }
45       else
46       {
47
48       }
49    }
50    if (NumofAsuMotion == 1)
51    {
52       NumofAsuMotion = 0;
53    }
54    else
55    {
56       printf("Which Device would you like to open?\n");
57       scanf_s("%d", &NumofAsuMotion);
58    }
```

```
59  AsuMotion=AsuMotionOpen(NumofAsuMotion);
60  if (!AsuMotion)
61  {
62     printf("AsuMotion Open Wrong!\n");
63     return 0;
64  }
65  else
66  {
67     printf("AsuMotion Open successes!\n");
68  }
69  ret=AsuMotionConfigDeviceDefault(AsuMotion);
70
71
72
73  if (!AsuMotionSetSmoothCoff(AsuMotion,
74     100,
75     64))
76  {
77     return 2;
78  }
79
80  if (!AsuMotionSetStepsPerUnit(AsuMotion,
81     &StepsPerUnit))
82  {
83     return 3;
84  }
85  if (!AsuMotionSetWorkOffset(AsuMotion,
86     &WorkOffset))
87  {
88     return 4;
89  }
90  AsuMotionSetDifferentialOutputMapping(
91     AsuMotion,
92     DiffFunSel,
93     (AsuMotionBitMaskType)0
94     );
95
96  Sleep(1000);
97  AsuMotionSetAccelaration(AsuMotion, &Accelaration);
98  AsuMotionSetMaxSpeed(AsuMotion, &MaxSpeed);
99  AsuMotionStopAll(AsuMotion);
100
101 ret = AsuMotionIsDone(AsuMotion);
```

```
102    while (ret == AsuMotion_False)
103    {
104        ret = AsuMotionIsDone(AsuMotion);
105        AsuMotionGetSteps(AsuMotion, &fullSteps);
106        printf("ForeMove-x:%10.3f;y:%10.3f;z:%10.3f;a:%10.3f;b:%10.3f;c:%10.3f↩
               ;u:%10.3f;v:%10.3f;w:%10.3f\n",
107            (float)fullSteps.x / StepsPerUnit.x,
108            (float)fullSteps.y / StepsPerUnit.y,
109            (float)fullSteps.z / StepsPerUnit.z,
110            (float)fullSteps.a / StepsPerUnit.a,
111            (float)fullSteps.b / StepsPerUnit.b,
112            (float)fullSteps.c / StepsPerUnit.c,
113            (float)fullSteps.u / StepsPerUnit.u,
114            (float)fullSteps.v / StepsPerUnit.v,
115            (float)fullSteps.w / StepsPerUnit.w
116
117            );
118        Sleep(1000);
119    }
120    AsuMotionSetMachineCoordinate(AsuMotion, &PoistionGiven);
121    AsuMotionGetSteps(AsuMotion, &fullSteps);
122    AsuMotionMoveAtConstSpeed(AsuMotion, (AsuMotionAxisMaskType)(0x1ff));
123
124    printf("Press any key to exit\n");
125    do
126    {
127        AsuMotionGetSteps(AsuMotion, &fullSteps);
128        printf("x:%10.3f;y:%10.3f;z:%10.3f;a:%10.3f;b:%10.3f;c:%10.3f;u:%10.3f↩
               ;v:%10.3f;w:%10.3f\r",
129            (float)fullSteps.x / StepsPerUnit.x,
130            (float)fullSteps.y / StepsPerUnit.y,
131            (float)fullSteps.z / StepsPerUnit.z,
132            (float)fullSteps.a / StepsPerUnit.a,
133            (float)fullSteps.b / StepsPerUnit.b,
134            (float)fullSteps.c / StepsPerUnit.c,
135            (float)fullSteps.u / StepsPerUnit.u,
136            (float)fullSteps.v / StepsPerUnit.v,
137            (float)fullSteps.w / StepsPerUnit.w
138
139            );
140
141        Sleep(1000);
142    } while (!_kbhit());
```

```
143
144     AsuMotionClose(AsuMotion);
145     return 0;
146 }
```