Dobot M1 通信协议

文档版本：V1.0

发布日期：2018-11-10

深圳市越疆科技有限公司

免责申明

在法律允许的最大范围内，本手册所描述的产品（含其硬件、软件、固件等）均“按照现状”提供，可能存在瑕疵、错误或故障，越疆不提供任何形式的明示或默示保证，包括但不限于适销性、质量满意度、适合特定目的、不侵犯第三方权利等保证；亦不对使用本手册或使用本公司产品导致的任何特殊、附带、偶然或间接的损害进行赔偿。

在使用本产品前详细阅读本使用手册及网上发布的相关技术文档并了解相关信息，确保在充分了解机器人及其相关知识的前提下使用机械臂。越疆建议您在专业人员的指导下使用本手册。该手册所包含的所有安全方面的信息都不得视为Dobot的保证，即便遵循本手册及相关说明，使用过程中造成的危害或损失依然有可能发生。

本产品的使用者有责任确保遵循相关国家的切实可行的法律法规，确保在越疆机械臂的使用中不存在任何重大危险。

越疆科技有限公司

地址：深圳市南山区同富裕工业城三栋三楼

网址：http://cn.dobot.cc/

前 言

目的

本文档适用于Dobot M1产品上位机与Dobot M1机械臂之间命令/数据交互的通信协议。

读者对象

本手册适用于：

- 客户工程师
- 销售工程师
- 安装调测工程师
- 技术支持工程师

修订记录

时间	修订记录
2018/11/10	第一次发布

符号约定

在本手册中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害
⚠警告	表示有中度或低度潜在危害，如果不能避免，可能导致人员轻微伤害、机械臂毁坏等情况
⚠注意	表示有潜在风险，如果忽视这些文本，可能导致机械臂损坏、数据丢失或不可预知的结果
🕮说明	表示是正文的附加信息，是对正文的强调和补充

目 录

1. 通讯协议

1.1 通信参数

表格 1 通信参数说明

通信参数	详细参数	参数说明
USB 转串口	波特率	115200bps
	数据位	8 位
	停止位	1 位
	校验位	无
UDP	IP	路由等分配
	端口	54321

1.2 协议简介

Dobot M1 机械臂可通过 PC 客户端控制，这些设备与机械臂之间通过特定的通信协议实现数据传输，进而实现控制。目前 Dobot M1 可以通过默认 USB 转串口、局域网（UDP）控制。

物理层每次接收的数据是 8 位原始数据，需要制定通信协议来确定数据传输的开始与结束、以及校验数据的准确性。通信协议一般需要包括数据包的包头、数据负载、负载校验，来保证数据的准确传输。

1.2.1 协议特点

Dobot M1 的通信协议的特点如下：

- 协议指令不定长。
- 协议指令由包头、负载帧长、负载帧、校验组成。
- 指令分为立即指令和队列指令。
- 所有的通信都由主机主动发起，且对于所有通信指令，下位机都会返回数据（无论读写）；对于队列指令，其返回带有 64 位的执行索引值。
- 立即指令将立即被调用执行，所有的读操作都是立即指令。
- 队列指令将被放入下位机队列中，串行地执行。对于写（或设置）操作，其中运动类型的指令都应该是队列命令（比如回零、JOG、PTP 等）。
- 设置参数的指令既可以是立即指令，也可以是队列指令。
- 在主机向下位机发送队列命令前，应查询下位机命令队列的剩余空间（可查询一次，发送多条命令）。
- 立即指令总是立即执行完成；队列指令的执行完成情况可通过查询当前下位机正在执行的队列命令索引，与该条命令的索引（第 4 点中提到的命令中返回的）的比较得到。
- 命令中的参数使用小端模式。

1.2.2 校验计算

在 Dobot M1 的通信协议中，发送端校验计算方法如下：

步骤 1 将负载帧（Payload）中的所有内容按照字节（8位）的方式逐字节相加，得到一个结果R（低8位）；

步骤 2 对结果R（低8位）求二补数，存入校验字节中。

📖说明

二补数：Two’s complement。对于一个 N 位的数字，其二补数等于2^N减去该数。在本协议中，假设上述的结果 R 为 0x0A，其二补数也即上述的校验结果等于(2^8 - 0x0A) = (256 - 10) = 246 = 0xF6。

在接收端，校验一帧数据是否正确的方法如下：

步骤 1 将负载帧（Payload）中的所有内容按照字节（8位）的方式逐字节相加，得到一个结果A（低8位）；

步骤 2 结果A（低8位）与校验字节相加，如果等于0，则说明校验正确。

1.2.3 协议分类

根据实现功能不同，又可分为以下几部分：

- 队列执行控制命令
- 设备信息相关命令
- 公共参数命令
- 回零功能命令
- 手持示教命令
- 点动模式命令
- PTP 模式命令
- CP 模式命令
- WAIT 模式命令
- TRIG 触发相关命令
- I/O 控制命令等

通过分类，将通信协议功能 ID 分成表格 2 中的几大项：

表格 2 功能项划分

功能项划分	Function ID 区间	可用 ID 个数
ProtocolFunctionDeviceInfoBase	[0, 10)	10
ProtocolFunctionPoseBase	[10, 20)	10
ProtocolFunctionALARMBase	[20, 30)	10
ProtocolFunctionHOMEBase	[30, 40)	10
ProtocolFunctionHHTBase	[40, 50)	10
ProtocolFunctionArmOrientationBase	[50, 60)	10
ProtocolFunctionEndEffectorBase	[60, 70)	10
ProtocolFunctionJOGBase	[70, 80)	10
ProtocolFunctionPTPBase	[80, 90)	10
ProtocolFunctionCPBase	[90, 100)	10
ProtocolFunctionARCBase	[100, 110)	10
ProtocolFunctionWAITBase	[110, 120)	10
ProtocolFunctionTRIGBase	[120, 130)	10
ProtocolFunctionEIOBase	[130, 140)	10
ProtocolFunctionCALBase	[140, 150)	10

功能项划分	Function ID 区间	可用 ID 个数
ProtocolFunctionWIFIBase	[150, 160)	10
ProtocolFunctionQueuedCmdBase	[240, 250)	10
ProtocolMax	256	1

📖说明

- 在下文的每条指令说明中，都带有 ID 说明；
- 下文中 Ctrl 字节中，rw 为 Ctrl 字节的第 0 位，isQueued 为 Ctrl 字节的第 1 位。

1.3 设备信息

该部分命令用于设置设备 SN 号、设备名称和设备版本号，并通过命令读回设备当前信息。

1.3.1 设置和获取设备序列号（Set/Get DeviceSN）

- 设置设备序列号（SetDeviceSN），下发的指令包格式如表格 3 所示，返回指令包格式如表格 4 所示。

表格 3 设置设备序列号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+n	0	1	0	char[n] DeviceSN	Payload checksum

表格 4 设置设备序列号返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	0	1	0	Empty	Payload checksum

- 获取设备序列号（GetDeviceSN），下发的指令包格式如表格 5 所示，返回指令包格式如表格 6 所示。

表格 5 获取设备序列号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	0	0	0	Empty	Payload checksum

表格 6 获取设备序列号返回指令包

Header	Len	Payload			Checksum
		ID	Ctrl	Params	
0xAA 0xAA	2+n	0	1	char[n] DeviceSN	Payload checksum

			rw	isQueued		
0xAA 0xAA	2+n	0	0	0	char[n] DeviceSN	Payload checksum

1.3.2 设置和获取设备名称（Set/Get DeviceName）

- 设置设备名称（SetDeviceName），下发的指令包格式如表格 7 所示，返回指令包格式如表格 8 所示。

表格 7 设置设备名称指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+n	1	1	0	char[n] DeviceName	Payload checksum

表格 8 设置设备名称返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	1	1	0	Empty	Payload checksum

- 获取设备名称（GetDeviceName），下发的指令包格式如表格 9 所示，返回指令包格式如表格 10 所示。

表格 9 获取设备名称指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	1	0	0	Empty	Payload checksum

表格 10 获取设备名称返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+n	1	0	0	char[n] DeviceName	Payload checksum

1.3.3 获取设备版本号（GetDeviceVersion）

获取设备版本号（GetDeviceVersion），下发的指令包格式如表格 11所示，返回指令包格式如表格 12。

表格 11 获取设备版本号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	2	0	0	Empty	Payload checksum

表格 12 获取设备版本号返回指令包

Header	Len	Payload							Checksum
		ID	Ctrl		Params				
			rw	isQueued					
0xAA 0xAA	2+4	2	0	0	uint8_t: typeIndex（见程序 1）	uint8_t: majorVersion	uint8_t: minorVersion	uint8_t : revision	Payload checksum

程序 1 typeIndex 定义

```
typeIndex：版本类型
enum FirmwareType{
    NO_SWITCH,
    DOBOT_SWITCH,
    PRINTING_SWITCH,
    DRIVER1_SWITCH,
    DRIVER2_SWITCH,
    DRIVER3_SWITCH,
    DRIVER4_SWITCH,
    DRIVER5_SWITCH,
    FPGA_SWITCH,
    SWITCH_FM_MAX
};
```

1.3.4 获取硬件版本号（GetHardwareVersion）

- 获取硬件版本号（GetHardwareVersion），下发的指令包格式如表格 13 所示，返回指令包格式如表格 14 所示。

表格 13 获取硬件版本号指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	5	0	0	Empty	Payload checksum

表格 14 获取硬件版本号返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+132	5	0	0	HardwareInfo（见程序 2）	Payload checksum

程序 2 HardwareInfo 定义

```
typedef struct tagHardwareInfo{
    char machineNum[11];
    char mainBoard[11];
    char driverRearArm[11];
    char driverFrontArm[11];
    char driverZArm[11];
    char driverRArm[11];
    char encoderRearArm[11];
    char encoderFrontArm[11];
    char encoderZArm[11];
    char encoderRArm[11];
    char brakeBoard[11];
    char endIOBoard[11];
}HardwareInfo;
```

1.4 实时位姿

实现获取实时位姿、校准等功能。

1.4.1 获取实时位姿（GetPose）

获取实时位姿（GetPose），下发的指令包格式如表格 15 所示，返回指令包格式如表格 16 所示。

表格 15 获取实时位姿指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	10	0	0	Empty	Payload checksum

表格 16 获取实时位姿返回指令包

Header	Len	Payload			Checksum
		ID	Ctrl	Params	

			rw	isQueued		
0xAA 0xAA	2+32	10	0	0	Pose（见程序 3）	Payload checksum

程序 3 Pose 定义

```
typedef struct tagPose {
    float x;                    //机械臂坐标系 x
    float y;                    //机械臂坐标系 y
    float z;                    //机械臂坐标系 z
    float r;                    //机械臂坐标系 r
    float jointAngle[4];    //机械臂 4 轴(底座、大臂、小臂、末端)角度
} Pose;
```

1.4.2 校准机械臂（ResetPose）

校准机械臂（ResetPose），下发的指令包格式如表格 17 所示，返回指令包格式如表格 18 所示。

表格 17 校准机械臂指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+9	11	1	0	ResetPoseParams（见程序 4）	Payload checksum

表格 18 校准机械臂返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	11	1	0	Empty	Payload checksum

程序 4 RestPoseParams 定义

```
typedef struct{
    uint8_t manual;         //只能设置为 1
    float frontAngle1       // frontAngle1 和 frontAngle2 分别为机械臂以左右手到达同一点时小臂的角度
    float frontAngle2;
} ResetPoseParams
```

1.5 报警功能

1.5.1 获取报警状态（GetAlarmsState）

获取报警状态（GetAlarmsState），下发的指令包格式如表格 19 所示，返回指令包格式如表格 20 所示。

表格 19 获取系统报警状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	0	0	Empty	Payload checksum

表格 20 获取系统报警状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	20	0	0	uint8_t[32]:alarmsState	Payload checksum

📖说明

数组 alarmsState 中的每一个字节可以标识 8 个报警项的报警状态，且 MSB 在高位，LSB 在低位。报警每个位的具体含义参考报警说明文档。

1.5.2 清除系统的所有报警（ClearAllAlarmsState）

清除系统的所有报警（ClearAllAlarmsState），下发的指令包格式如表格 21 所示，返回指令包格式如表格 22 所示。

表格 21 清除系统报警状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	1	0	Empty	Payload checksum

表格 22 清除系统报警状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	20	1	0	Empty	Payload checksum

1.6 回零功能

该部分为回零功能，包括设置当前位置为初始位置、通过回零开关回零。机械臂默认初始位置为（400,0,0,0）。

1.6.1 设置初始位置（SetHOMECmd）

设置初始位置（SetHOMECmd），即把当前位置设置为(400,0,0)。下发的指令包格式如表格 23 所示，返回指令包格式如表格 24 所示。

表格 23 设置初始位置指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	31	1	0	Empty	Payload checksum

表格 24 设置初始位置返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	31	1	0	Empty	Payload checksum

1.6.2 通过回零开关回零（SetHOMEWithSwitch）

通过回零开关回零（SetHOMEWithSwitch），下发的指令包格式如表格 23 所示，返回指令包格式如表格 24 所示。

表格 25 通过回零开关回零指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	33	1	0 或 1	uint8_t :isResetPars	Payload checksum

- isResetPars 为 0，即机械臂运动至回零位置，将存储在 Flash 中的零位赋给系统参数。一般情况下请设置为 0。
- isResetPars 为 1，即机械臂运动至回零位置，并更新系统参数至 Flash 中。仅在出厂标定时使用。

表格 26 通过回零开关回零返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0 : 2+0; isQueued=1: 2+8	33	1	0 或 1	isQueued=0:Empty；isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

1.7 手持示教功能

1.7.1 设置和获取手持示教触发模式（Set/Get HHTTrigMode）

- 设置手持示教时的存点触发模式（SetHHTTrigMode），下发的指令包格式如表格 27 所示，返回指令包格式如表格 28 所示。

表格 27 设置手持示教触发模式指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	40	1	0	HHTTrigMode（见 程序 5）	Payload checksum

表格 28 设置手持示教触发模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	40	1	0	Empty	Payload checksum

- 获取手持示教时的存点触发模式（GetHHTTrigMode），下发的指令包格式如表格 29 所示，返回指令包格式如表格 30 所示。

表格 29 获取手持示教触发模式指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	40	0	0	Empty	Payload checksum

表格 30 获取手持示教触发模式返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+1	40	0	0	HHTTrigMode（见程序 5）	Payload checksum

程序 5 HHTTrigMode 的定义

```
typedef enum tagHHTTrigMode {
    TriggeredOnKeyReleased,          //按键释放时存点
    TriggeredOnPeriodicInterval      //实时存点
} HHTTrigMode;
```

1.7.2 设置和获取触发模式输出使能/禁止（Set/Get HHTTrigOutputEnabled）

- 设置手持示教触发输出使能状态（SetHHTTrigOutputEnabled），下发的指令包格式如表格 31 所示，返回指令包格式如表格 32 所示。

表格 31 设置触发模式输出使能/禁止指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	41	1	0	uint8_t: isEnabled	Payload checksum

表格 32 设置触发模式输出使能/禁止返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	41	1	0	Empty	Payload checksum

- 获取触发输出使能/禁止状态（GetHHTTrigOutputEnabled），下发的指令包格式如表格 33 所示，返回指令包格式如表格 34 所示。

表格 33 获取触发模式输出使能/禁止指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	41	0	0	Empty	Payload checksum

表格 34 获取触发模式输出使能/禁止返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	41	0	0	uint8_t: isEnabled	Payload checksum

1.7.3 获取触发输出状态（GetHHTTrigOutput）

获取触发输出状态（GetHHTTrigOutput），下发的指令包格式如表格 35所示，返回指令包格式如表格 36所示。

表格 35 获取触发输出状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	42	0	0	Empty	Payload checksum

表格 36 获取触发输出状态返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	42	0	0	uint8_t: isTriggered	Payload checksum

1.8 机械臂方向

1.8.1 设置/获取机械臂方向（Set/Get ArmOrientation）

⚠注意

该命令当前仅适用于 SCARA 机型。

- 设置手臂方向（SetArmOrientation），下发的指令包格式如表格 37 所示，返回指令包格式如表格 38 所示。

表格 37 设置手臂方向指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	50	1	0 or 1	ArmOrientation（见 程序 6）	Payload checksum

表格 38 设置手臂方向返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0:	50	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdI	Payload checksum

	2+0 isQueue d=1:2+1				ndex	

- 获取手臂方向（GetArmOrientation），下发的指令包格式如表格 39所示，返回指令包格式如表格 40所示。

表格 39 获取手臂方向指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	50	0	0	Empty	Payload checksum

表格 40 获取手臂方向返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+1	50	0	0	ArmOrientation（见 程序 6）	Payload checksum

程序 6 ArmOrientation 定义

```
typedef enum tagArmOrientation {
LeftyArmOrientation,
RightyArmOrientation
} ArmOrientation;
```

1.9 末端执行器

1.9.1 设置和获取末端执行器参数（Set/Get EndEffectorParams）

📖说明

该指令仅适用于配套的激光雕刻和 3D 打印套件。

- 设置末端执行器参数（SetEndEffectorParams），下发的指令包格式如表格 41 所示，返回指令包格式如表格 42 所示。

表格 41 设置末端执行器参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+12	60	1	0 or 1	EndEffectorParams	Payload

					（见程序 7）	checksum

表格 42 设置末端执行器参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0:2+0；isQueued=1:2+8	60	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取末端执行器参数（GetEndEffectorParams），下发的指令包格式如表格 43 所示，返回指令包格式如表格 44 所示。

表格 43 获取末端执行器参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	60	0	0	Empty	Payload checksum

表格 44 获取末端执行器参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+12	60	0	0	EndEffectorParams（见 程序 7）	Payload checksum

程序 7 EndEffectorParams 定义

```
typedef struct tagEndEffectorParams {
    float xBias;            //末端 x 轴方向长度
    float yBias;            //末端 y 轴方向长度
    float zBias;            //末端 z 轴方向长度
} EndEffectorParams;
```

1.9.2 设置和获取激光输出（Set/Get EndEffectorLaser）

- 设置激光开关（SetEndEffectorLaser），下发的指令包格式如表格 45 所示，返回指令包格式如表格 46 所示。

表格 45 设置激光开关指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	61	1	0 or 1	uint8_t: enableCtrl	uint8_t: on	Payload checksum

表格 46 设置激光开关返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	61	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

📖说明

末端是否使能（enableCtrl），激光是否开启（on）。

- 获取激光开关状态（GetEndEffectorLaser），下发的指令包格式如表格 47所示，返回指令包格式如表格 48所示。

表格 47 获取激光开关状态指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	61	0	0	Empty	Payload checksum

表格 48 获取激光开关状态指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+2	61	0	0	uint8_t: isCtrlEnabled	uint8_t: isOn	Payload checksum

📖说明

末端是否使能（isCtrlEnabled），激光是否开启（isOn）。

1.10 JOG 功能

设置/获取关节参数、坐标系参数，点动公共参数和执行点动功能。

1.10.1 设置和获取关节坐标系点动参数（Set/Get JOGJointParams）

- 设置关节坐标系点动参数（SetJOGJointParams），下发的指令包格式如表格 49 所示，返回指令包格式如表格 50 所示。

表格 49 设置关节点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	70	1	0 or 1	JOGJointParams（见 程序 8）	Payload checksum

表格 50 设置关节点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	70	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

📖说明

在示教前，需要设置关节的速度和加速度参数。该指令会设置四个关节的速度和加速度。

- 获取关节坐标系点动参数（GetJOGJointParams），下发的指令包格式如表格 51 所示，返回指令包格式如表格 52 所示。

表格 51 获取关节点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	70	0	0	Empty	Payload checksum

表格 52 获取关节点动参数返回指令包

Header	Len	Payload	Checksu

		ID	Ctrl		Params	m
			rw	isQueued		
0xAA 0xAA	2+32	70	0	0	JOGJointParams（见 程序 8）	Payload checksum

程序 8 JOGJointParams 定义

```
typedef struct tagJOGJointParams{
    float velocity[4];          //4 轴关节速度
    float acceleration[4];   //4 轴关节加速度
}JOGJointParams;
```

1.10.2 设置和获取笛卡尔坐标系点动参数（Set/Get JOGCoordinateParams）

- 设置笛卡尔坐标系参数（SetJOGCoordinateParams），下发的指令包格式如表格 53 所示，返回指令包格式如表格 54 所示。

表格 53 设置坐标系参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	71	1	0 or 1	JOGCoordinateParams（见 程序 9）	Payload checksum

表格 54 设置坐标系参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	71	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

📖说明

该指令设置的是笛卡尔坐标系的参数，即 X、Y、Z、R 轴的速度和加速度。同样，需要在示教前设置。

- 获取笛卡尔坐标轴点动参数（GetJOGCoordinateParams），下发的指令包格式如表格 55 所示，返回指令包格式如表格 56 所示。

表格 55 获取坐标轴点动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	71	0	0	Empty	Payload checksum

表格 56 获取坐标轴点动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	71	0	0	JOGCoordinateParams（见 程序 9）	Payload checksum

程序 9 JOGCoordinateParams 定义

```
typedef struct tagJOGCoordinateParams {
    float velocity[4];          //4 轴坐标轴（x,y,z,r）速度
    float acceleration[4];      //4 轴坐标轴（x,y,z,r）加速度
} JOGCoordinateParams;
```

1.10.3 设置和获取点动公共参数（Set/Get JOGCommonParams）

- 设置点动公共参数（SetJOGCommonParams），下发的指令包格式如表格 57 所示，返回指令包格式如表格 58 所示。

表格 57 设置点动公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	72	1	0 or 1	JOGCommonParams（见程序 10）	Payload checksum

表格 58 设置点动公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0:	72	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdI	Payload checksum

	2+0; isQueue d=1: 2+8				ndex	

说明

该指令设需要在示教前设置。

- 获取点动公共参数（GetJOGCommonParams），下发的指令包格式如表格 59 所示，返回指令包格式如表格 60 所示。

表格 59 获取点动公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	72	0	0	Empty	Payload checksum

表格 60 获取点动公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	72	0	0	JOGCommonParams（见程序 10）	Payload checksum

程序 10 JOGCommonParams 定义

```
typedef struct tagJOGCommonParams {
    float velocityRatio;          //速度比例，关节点动和坐标轴点动共用
    float accelerationRatio;      //加速度比例，关节点动和坐标轴点动共用
} JOGCommonParams;
```

1.10.4 执行点动功能（SetJOGCmd）

执行点动功能，下发的指令包格式如表格 61 所示，返回指令包格式如表格 62 所示。

表格 61 执行点动功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	73	1	0 or 1	JOGCmd（程序 11）	Payload

						checksum

表格 62 执行点动功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	73	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 11 JOGCmd 定义

```
typedef struct tagJOGCmd {
    uint8_t isJoint;     //点动方式 0：笛卡尔坐标系点动  1：关节坐标系点动
    uint8_t cmd;         //点动命令（取值范围 0~8）
}JOGCmd;

//点动命令详细说明
enum {
    IDEL,            //无效状态
    AP_DOWN,         // X+/Joint1+
    AN_DOWN,         // X-/Joint1-
    BP_DOWN,         // Y+/Joint2+
    BN_DOWN,         // Y-/Joint2-
    CP_DOWN,         // Z+/Joint3+
    CN_DOWN,         // Z-/Joint3-
    DP_DOWN,         // R+/Joint4+
    DN_DOWN          // R-/Joint4-
};
```

1.11 再现(PTP)功能

再现功能的指令，用于再现相关的运动参数设置，包括关节坐标系参数、笛卡尔坐标系参数，速度和加速度比例参数以及其他相关的参数。

1.11.1 设置和获取关节点位参数（Set/Get PTPJointParams）

这两条命令用于设置和获取再现速度参数，包括关节再现速度和加速度，此命令设置的

速度相关参数仅适用于再现运动，对于 JOG 功能无效。

- 设置关节坐标系点位参数（SetPTPJointParams），用于控制再现运动的速度实现运动的快慢控制。下发的指令包格式如表格 63 所示，返回指令包格式如表格 64 所示。

表格 63 设置关节坐标系点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	80	1	0 or 1	PTPJointParams（程序 12）	Payload checksum

表格 64 设置关节坐标系点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	80	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取关节坐标系点位参数（GetPTPJointParams），下发的指令包格式如表格 65 所示，返回指令包格式如表格 66 所示。

表格 65 获取关节坐标系点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	80	0	0	Empty	Payload checksum

表格 66 获取关节坐标系点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	80	0	0	PTPJointParams（程序 12）	Payload checksum

程序 12 PTPJointParams 定义

```
typedef struct tagPTPJointParams {
    float velocity[4];          //PTP 模式下 4 轴关节速度
    float acceleration[4];      //PTP 模式下 4 轴关节加速度
} PTPJointParams;
```

1.11.2 设置和获取坐标轴点位参数（Set/Get PTPCoordinateParams）

- 设置笛卡尔坐标系点位参数（SetPTPCoordinateParams），下发的指令包格式如表格 67 所示，返回指令包格式如表格 68 所示。

表格 67 设置笛卡尔坐标系点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	81	1	0 or 1	PTPCoordinateParams（见程序 13）	Payload checksum

表格 68 设置笛卡尔坐标系点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	81	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取笛卡尔坐标系点位参数（GetPTPCoordinateParams），下发的指令包格式如表格 69 所示，返回指令包格式如表格 70 所示。

表格 69 获取笛卡尔坐标系点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	81	0	0	Empty	Payload checksum

表格 70 获取笛卡尔坐标系点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+16	81	0	0	PTPCoordinateParams(见程序 13)	Payload checksum

程序 13 PTPCoordinateParams 定义

```
typedef struct tagPTPCoordinateParams {
    float xyzVelocity;          //PTP 模式下 xyz 3 轴坐标轴速度
    float rVelocity;            //PTP 模式下末端速度
    float xyzAcceleration;      //PTP 模式下 xyz 3 轴坐标轴加速度
    float rAccleration;         //PTP 模式下末端加速度
} PTPCoordinateParams;
```

1.11.3 设置和获取门型模式点位参数（Set/Get PTPJumpParams）

- 设置门型模式点位参数（SetPTPJumpParams），下发的指令包格式如表格 71 所示，返回指令包格式如表格 72 所示。

表格 71 设置门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	82	1	0 or 1	PTPJumpParams（(见程序 13)	Payload checksum

表格 72 设置门型模式点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	82	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取门型模式点位参数（GetPTPJumpParams），下发的指令包格式如表格 73 所示，返回指令包格式如表格 74 所示。

表格 73 获取门型模式点位参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+0	82	0	0	Empty	Payload checksum

表格 74 获取门型模式点位参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	82	0	0	PTPJumpParams（见程序 14）	Payload checksum

程序 14 PTPJumpParams 定义

```
typedef struct tagPTPJumpParams {
    float jumpHeight;          //门型模式运动抬升距离
    float zLimit;              //门型模式运动最大抬升高度限制
} PTPJumpParams;
```

1.11.4 设置和获取点位公共参数（Set/Get PTPCommonParams）

设置点位公共参数（SetPTPJointParams），下发的指令包格式如表格 75所示，返回指令包格式如表格 76所示。

表格 75 设置点位公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	83	1	0 or 1	PTPCommonParams（见程序 15）	Payload checksum

表格 76 设置点位公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	83	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

获取点位公共参数（GetPTPJointParams），下发的指令包格式如表格 77所示，返回指令

包格式如表格 78所示。

表格 77 获取点位公共参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	83	0	0	Empty	Payload checksum

表格 78 获取点位公共参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	83	0	0	PTPCommonParams（见程序 15）	Payload checksum

程序 15 PTPCommonParams 定义

```
typedef struct tagPTPCommonParams {
    float velocityRatio;        //PTP 模式速度比例，关节和坐标轴模式共用
    float accelerationRatio; //PTP 模式加速度比例，关节和坐标轴模式共用
} PTPCommonParams;
```

1.11.5 执行点位功能（SetPTPCmd）

执行点位功能（PTPCmd），下发的指令包格式如表格 79所示，返回指令包格式如表格 80所示。

表格 79 执行点位功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	84	1	0 or 1	PTPCmd（见程序 16）	Payload checksum

表格 80 执行点位功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueue	84	1	0 or 1	isQueued=0:Empty	Payload checksum

	d=0: 2+0; isQueue d=1: 2+8				isQueued=1:uint64_t:queued CmdIndex	

程序 16 PTPCmd 定义

```
typedef struct tagPTPCmd {
    uint8_t ptpMode;   //PTP 模式 (取值范围 0~9)
    float x;              //x,y,z,r 为 ptpMode 运动方式的参数，可为坐标、关节角度或者坐标/角度增量
    float y;
    float z;
    float r;
} PTPCmd;

其中，ptpMode 取值如下：
enum {
    PTPJUMPXYZMode,          //门型运动，参数为目标点坐标
    PTPMOVJXYZMode,          //关节运动，参数为目标点坐标
    PTPMOVLXYZMode,          //直线运动，参数为目标点坐标
    PTPJUMPANGLEMode,        //门型运动，参数为目标点关节角度
    PTPMOVJANGLEMode,        //关节运动，参数为目标点关节角度
    PTPMOVLANGLEMode,        //直线运动，参数为目标点关节角度
    PTPMOVJANGLEINCMode,     //关节运动增量模式，参数为目标点关节角度增量
    PTPMOVLXYZINCMode,       //直线运动增量模式，参数为目标点坐标增量
    PTPMOVJXYZINCMode,       //关节运动增量模式，参数为目标点坐标增量
    PTPJUMPMOVLXYZMode,      //门型运动，平移时运动模式为 MOVL
};
```

1.12 CP 功能

连续轨迹功能的指令，用于连续轨迹相关的运动设置和配置。其中包括关节参数、坐标系参数、功能设置参数等。连续轨迹功能和上位机 Dobot CP 对应，可以实现写字、画画、激光雕刻等需要连续轨迹的功能。

1.12.1 设置和获取连续轨迹功能参数（Set/Get CPParams）

这两条命令用于设置和获取连续轨迹参数，包括预设加速度、关节速度及加速度，此命令设置的速度仅适用于连续轨迹运动。

设置连续轨迹运动参数（SetCPParams），用于控制连续轨迹运动的速度实现运动的快慢控制，下发的指令包格式如

表格 81所示，返回指令包格式如表格 82所示。

表格 81 设置连续轨迹运动参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+13	90	1	0 or 1	CPParams（见程序 17）	Payload checksum

表格 82 设置连续轨迹运动参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	90	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取连续运动轨迹参数（GetCPParams），下发的指令包格式如表格 83所示，返回指令包格式如表格 84所示。

表格 83 获取连续运动轨迹参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	90	0	0	Empty	Payload checksum

表格 84 获取连续运动轨迹参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+13	90	0	0	CPParams（见程序 17）	Payload checksum

程序 17 CPParams 定义

```
typedef struct tagCPParams {
    float planAcc;              //规划加速度最大值
```

```
    float junctionVel;          //拐角加速度最大值
    union {
        float acc;              //实际加速度最大值，非实时模式下使用
        float period;           //插补周期，实时模式下使用
    };
    uint8_t realTimeTrack;   //0：非实时模式；  1：实时模式
} CPParams;
```

1.12.2 执行连续轨迹功能（SetCPCmd）

执行连续轨迹功能（SetCPCmd），下发的指令包格式如表格 85 所示，返回指令包格式如表格 86 所示。

表格 85 执行连续轨迹功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	91	1	0 or 1	CPCmd（见程序 18）	Payload checksum

表格 86 执行连续轨迹功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	91	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 18 CPCmd 定义

```
typedef struct tagCPCmd {
    uint8_t cpMode;          //CP 模式   0-相对模式   1-绝对模式
    float x;                 //x 坐标增量(相对模式) / x 轴坐标(绝对模式)
    float y;                 //y 坐标增量(相对模式) / y 轴坐标(绝对模式)
    float z;                 //z 坐标增量(相对模式) / z 轴坐标(绝对模式)
    union {
        float velocity;   // Reserved
        float power;         //激光功率
```

```
    }
} CPCmd;
```

1.12.3 执行连续轨迹灰度雕刻功能（SetCPLECmd）

执行连续轨迹灰度雕刻功能（SetCPLECmd），下发的指令包格式如表格 87所示，返回指令包格式如表格 88所示。

表格 87 执行连续轨迹功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	92	1	0 or 1	CPCmd（见程序 19）	Payload checksum

表格 88 执行连续轨迹功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	92	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 19 CPCmd 定义

```
typedef struct tagCPCmd {
    uint8_t cpMode;          //CP 模式  0-相对模式  1-绝对模式
    float x;                 //x 坐标增量(相对模式) / x 轴坐标(绝对模式)
    float y;                 //y 坐标增量(相对模式) / y 轴坐标(绝对模式)
    float z;                 //z 坐标增量(相对模式) / z 轴坐标(绝对模式)
    union {
        float velocity;      //预留
        float power;         //激光功率 0~100
    }
} CPCmd;
```

1.13 ARC 功能

1.13.1 设置和获取圆弧插补功能参数（Set/Get ARCParams）

- 设置圆弧插补功能参数（SetARCParams），下发的指令包格式如表格 89 所示，返回指令包格式如表格 90 所示。

表格 89 设置圆弧插补功能参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	100	1	0 or 1	ARCParams（见程序 20）	Payload checksum

表格 90 设置圆弧插补功能参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	100	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 获取圆弧插补功能参数（GetARCParams），下发的指令包格式如表格 91所示，返回指令包格式如表格 92所示。

表格 91 获取圆弧插补功能参数指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	100	0	0	Empty	Payload checksum

表格 92 获取圆弧插补功能参数返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+16	100	0	0	ARCParams（见程序 20）	Payload checksum

程序 20 ARCParams 定义

```
typedef struct tagARCParams {
    float xyzVelocity;          //圆弧运动 xyz 三坐标轴速度
    float rVelocity;            //圆弧运动末端旋转速度
    float xyzAcceleration;   //圆弧运动 xyz 三坐标轴加速度
    flaot rAcceleration;      //圆弧运动末端旋转加速度
} ARCParams;
```

1.13.2 执行圆弧插补功能（SetARCCmd）

执行圆弧插补功能（SetARCCmd），下发的指令包格式如表格 93 所示，返回指令包格式如表格 94 所示。

表格 93 执行圆弧插补功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+32	101	1	0 or 1	ARCCmd（见程序 21）	Payload checksum

表格 94 执行圆弧插补功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	101	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 21 ARCCmd 定义

```
typedef struct tagARCCmd {
    struct{
        float x;
        float y;
        float z;
        float r;
    } cirPoint;          //圆弧上任一点坐标
```

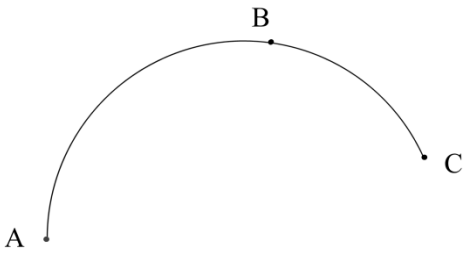
```
    struct {
        float x;
        float y;
        float z;
        float r;
    } toPoint;              //圆弧结束点坐标
} ARCCmd;
```

📖说明

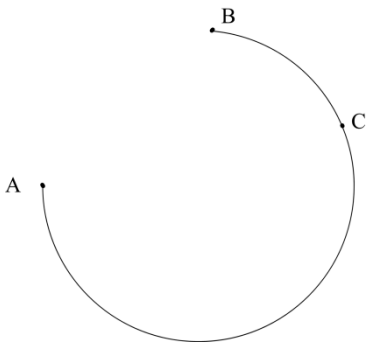
- 圆弧轨迹是空间的圆弧，由当前点、圆弧上任一点和圆弧结束点三点共同确定的.
- 圆弧总是从起点经过圆弧上一点再到结束点。

圆弧轨迹示例：

- A 为当前点，B 为圆弧上任一点，C 为结束点；



- A 为当前点，C 为圆弧上任一点，B 为结束点。



1.13.3 执行整圆插补功能（SetCircleCmd）

执行整圆插补功能（SetCircleCmd），下发的指令包格式如表格 93 所示，返回指令包格式如表格 94 所示。

表格 95 执行整圆插补功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+36	102	1	0 or 1	CircleCmd（见程序 22）	Payload checksum

表格 96 执行整圆插补功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	102	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 22 CircleCmd 定义

```
typedef struct t tagCircleCmd {
    struct{
        float x;
        float y;
        float z;
        float r;
    } cirPoint;          //整圆上任一点坐标

    struct {
        float x;
        float y;
        float z;
        float r;
    } toPoint;           //整圆结束点坐标
    uint32_t count;
}CircleCmd
```

1.14 WAIT 功能

1.14.1 执行时间等待功能（SetWAITCmd）

执行时间等待功能（SetWAITCmd），下发的指令包格式如表格 97 所示，返回指令包格

式如表格 98 所示。

表格 97 执行时间等待功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	110	1	0 or 1	WAITCmd（见程序 23）	Payload checksum

表格 98 执行时间等待功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	110	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 23 WAITCmd 定义

```
typedef struct tagWAITCmd {
    uint32_t timeout;          //单位：ms
} WAITCmd;
```

1.15 TRIG 功能

1.15.1 执行触发功能（SetTRIGCmd）

执行触发功能（SetTRIGCmd），下发的指令包格式如表格 99 所示，返回指令包格式如表格 100 所示。

表格 99 执行触发功能指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+5	120	1	0 or 1	TRIGCmd（见程序 24）	Payload checksum

表格 100 执行触发功能返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	120	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

程序 24 TrigCmd 定义

```
typedef struct tagTrigCmd {
    uint8_t address;          //I/O 地址.若 mode 设置为 0，取值范围：1~24。若 mode 设置为 1，取值范围：1~6
    uint8_t mode;             //触发模式 0：电平触发  1：AD 触发
    uint8_t condition;        //触发条件
                              电平触发：0，等于。1，不等于
                              A/D 触发：0，小于。1，小于等于
                              2，大于等于。3，大于
    uint16_t threshold;       //触发阈值：电平触发，0 或 1。A/D 触发：0~4095
} TRIGCmd;
typedef enum tagIOCondition {
    TRIGInputIOEqual,
    TRIGInputIONotEqual
} IOCondition;

typedef enum tagADCCondition {
    TRIGADCLT,      //Lower than
    TRIGADCLE,      //Lower than or Equal
    TRIGADCGE,      //Greater than or Equal
    TRIGADCGT       //Greater than
} ADCCondition;
```

1.16 EIO 功能

在 Dobot M1 控制器中，所有的扩展 I/O 都是统一编址的。根据现有情况，I/O 的功能包括以下内容：

- 高低电平输出功能。
- 读取输入高低电平功能。

- 读取输入模数转换值功能。

I/O 详细说明，请参见《Dobot M1 用户手册》。

1.16.1 设置和读取 I/O 输出电平（Set/Get IODO）

- 设置 I/O 口输出电平（SetIODO），下发的指令包格式如表格 101 所示，返回指令包格式如表格 102 所示。

表格 101 设置 I/O 口输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	131	1	0 or 1	IODO（见程序 25）	Payload checksum

表格 102 设置 I/O 口输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	isQueued=0: 2+0; isQueued=1: 2+8	131	1	0 or 1	isQueued=0:Empty isQueued=1:uint64_t:queuedCmdIndex	Payload checksum

- 读取I/O输出电平（GetIODO），下发的指令包格式如表格 103所示，返回指令包格式如表格 104所示。

表格 103 读取 I/O 口输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	131	0	0	Empty	Payload checksum

表格 104 读取 I/O 口输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	131	0	0	IODO（见程序 25)）	Payload checksum

程序 25 IODO 定义

```
typedef struct tagIODO {
    uint8_t address;        //I/O 地址（取值范围 1~22）
    uint8_t level;          //输出电平 0：低电平 1：高电平
} IODO;
```

1.16.2 读取 I/O 输入电平（GetIODI）

读取I/O 输入电平（GetIODI），下发的指令包格式如表格 105所示，返回指令包格式如表格 106所示。

表格 105 读取 I/O 输出电平指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	133	0	0	Empty	Payload checksum

表格 106 读取 I/O 输出电平返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+2	133	0	0	IODI（ 程序 26）	Payload checksum

程序 26 IODI 定义

```
typedef struct tagIODI {
    uint8_t address;        //I/O 地址（取值范围 1~24）
    uint8_t level;          //输入电平 0：低电平 1：高电平
}IODI;
```

1.16.3 读取 I/O 模数转换值（GetIOADC）

读取 I/O 模数转换值（GetIOADC），下发的指令包格式如表格 107 所示，返回指令包格式如表格 108 所示。

表格 107 读取 I/O 模数转换值指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	134	0	0	Empty	Payload checksum

表格 108 读取 I/O 模数转换值返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+3	134	0	0	IOADC（见程序 27）	Payload checksum

程序 27 IOADC 定义

```
typedef struct tagIOADC{
    uint8_t address;        //EIO 地址（取值范围 1~6）
    uint16_t value;         //输入 ADC 值，范围 0~4095
}IOADC;
```

1.17 队列执行控制命令

队列执行控制命令主要是用于设置队列命令执行的相关参数，包括命令执行模式（在线/离线）、队列命令缓冲器当前状态、队列命令执行状态（TRUE / FALSE）、队列命令执行控制（START / PAUSE / STOP）。

1.17.1 启动指令队列运行（SetQueuedCmdStartExec）

启动指令队列运行（SetQueuedCmdStartExec），下发的指令包格式如表格 109 所示，返回指令包格式如表格 110 所示。

表格 109 启动指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	240	1	0	Empty	Payload checksum

表格 110 启动指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	240	1	0	Empty	Payload checksum

1.17.2 停止指令队列运行（SetQueuedCmdStopExec）

停止指令队列运行（SetQueuedCmdStopExec），下发的指令包格式如表格 111 所示，返回指令包格式如表格 112 所示。

表格 111 停止指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	241	1	0	Empty	Payload checksum

表格 112 停止指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	241	1	0	Empty	Payload checksum

1.17.3 强制停止指令队列运行（SetQueuedCmdForceStopExec）

强制停止指令队列运行（SetQueuedCmdForceStopExec），下发的指令包格式如表格 113 所示，返回指令包格式如表格 114 所示。

表格 113 强制停止指令队列运行指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	242	1	0	Empty	Payload checksum

表格 114 强制停止指令队列运行返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	242	1	0	Empty	Payload checksum

1.17.4 启动指令队列下载（SetQueuedCmdStartDownload）

启动指令队列下载（SetQueuedCmdStartDownload），下发的指令包格式如表格 115 所示，返回指令包格式如表格 116 所示。

表格 115 启动指令队列下载指令包

Header	Len	Payload					Checksum
		ID	Ctrl		Params		
			rw	isQueued			
0xAA 0xAA	2+8	243	1	0	uint32_t: totalLoop	uint32: linePerLoop	Payload checksum

表格 116 启动指令队列下载返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	243	1	0	Empty	Payload checksum

📖说明

Dobot M1 的控制器支持将指令存储到控制器外部 Flash 中，实现脱机运行。

1.17.5 完成指令队列下载（SetQueuedCmdStopDownload）

完成指令队列下载（SetQueuedCmdStopDownload），下发的指令包格式如表格 117 所示，返回指令包格式如表格 118 所示。

表格 117 完成指令队列下载指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	244	1	0	Empty	Payload checksum

表格 118 完成指令队列下载返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	244	1	0	Empty	Payload checksum

1.17.6 清空指令队列（SetQueuedCmdClear）

清空指令队列（SetQueuedCmdClear），下发的指令包格式如表格 119 所示，返回指令包格式如表格 120 所示。

表格 119 清空指令队列指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	245	1	0	Empty	Payload checksum

表格 120 清空指令队列返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	245	1	0	Empty	Payload checksum

1.17.7 获取指令队列索引（GetQueuedCmdCurrentIndex）

获取指令队列索引（GetQueuedCmdCurrentIndex），下发的指令包格式如表格 121 所示，返回指令包格式如表格 122 所示。

表格 121 获取指令队列索引指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	246	0	0	Empty	Payload checksum

表格 122 获取指令队列索引返回指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+8	246	0	0	uint64_t: queuedCmdCurrentIndex	Payload checksum

📖说明

在 Dobot M1 控制器指令队列机制中，有一个 64 位内部计数索引。当控制器每执行完一条命令式，该计数器自动加一。通过该内部索引，可以查询控制器已经执行了多少条队列指令，以及当前执行到哪条指令（指示运行进度时）。

1.17.8 获取指令队列剩余空间（GetQueuedCmdLeftSpace）

获取指令队列剩余空间（GetQueuedCmdLeftSpace），下发的指令包格式如表格 123 所示，返回指令包格式如表格 124 所示。

表格 123 获取指令队列剩余空间指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+0	247	0	0	Empty	Payload checksum

表格 124 获取指令队列剩余空间指令包

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+4	247	0	0	uint32_t:leftSpace	Payload checksum

📖说明

在 Dobot M1 控制器指令队列机制中，有一个指令队列。在发送队列指令时，应先查询指令队列的剩余空间，若非零，才能向 Dobot M1 控制器发送队列指令。