TM-UAVCAN-V2.3

1 范围及版本定义

1：本文主要描述了电调（ESC）与外部设备通讯的协议。

2：CAN 协议符合 UAVCAN/DRONECAN 协议规范，可以访问 https://github.com/dronecan/libcanard 了解相关协议细节并获取相关例程。

2 术语

表 2-1　　文档术语

术语	解释说明
ESC	Electric Speed Controller　电子调速器
CAN	Controller Area Network　控制器局域网络同时是一种通讯协议。

3 参考文档

《CANBUS 规范 v2.0+中文版.pdf》

4 协议定义

4.1 TM-UAVCAN　协议

TM-UAVCAN 协议基于标准 CANBus 2.0B 协议，基于 29bit 的扩展帧数据帧。

TM-UAVCAN　硬件层配置情况	
CAN_SJW	CAN_SJW_1tq
CAN_BS1	CAN_BS1_4tq
CAN_BS2	CAN_BS2_1tq
SamplePoint	83.3%
BusCloclError	0%

4.1.1　概念

1：Message 帧 - 是广播帧，所有的节点都能够接收到此消息。

2：Service 帧 - 是非广播帧，指定节点 ID，节点发服务请求的时候要求有服务应答。注：所有的传输帧均包括单帧与多帧。

4.1.2 ID field

在 TM-UAVCAN 协议中，我们只用到了 CANBus2.0B 五种帧类型中的数据帧，所有的数据通过数据帧来传输；我们将数据帧的仲裁段 ID 码定义成以下格式：

Message frame

Field name	Priority					Message type ID											Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	1...127						
CAN ID bytes	3					2																0							

Anonymous message frame

Field name	Priority					Discriminator											Lower bits of message type ID								Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values																						0	0						
CAN ID bytes	3					2																0							

Service frame

Field name	Priority					Service type ID								Request not response	Destination node ID							Service not message	Source node ID						
CAN ID bits	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed values															1...127							1	1...127						
CAN ID bytes	3					2								1								0							

针对于 ESC 的 CAN 总线通讯应用，目前只用到了 Message frame 和 Service frame ，因此下文不会涉及到 Anonymous message frame 的解释。

位定义名词解释

<table>
<tr><th>位定义</th><th colspan="3">解释说明</th></tr>
<tr><td rowspan="6">Priority</td><td rowspan="6">1：Priority 表示 CAN 数据帧优先级。
2：优先级数值范围为 0~31。
3：优先级最高为 0，最低为 31。
4：TM-UAVCAN 定义优先级如右表格。</td><td>优先级</td><td>数值</td></tr>
<tr><td>HIGHEST</td><td>0x00</td></tr>
<tr><td>HIGH</td><td>0x08</td></tr>
<tr><td>MEDIUM</td><td>0x10</td></tr>
<tr><td>LOW</td><td>0x18</td></tr>
<tr><td>LOWEST</td><td>0x1F</td></tr>
<tr><td>Message type ID</td><td colspan="3">Message type ID 的范围从 0~65535,包含 0 和 65535。</td></tr>
<tr><td>Service type ID</td><td colspan="3">Service type ID 的范围从 0~255,包含 0 和 255。</td></tr>
<tr><td rowspan="2">Service not message</td><td rowspan="2">表示该数据帧的类型</td><td>0</td><td>表示数据帧为 Message 帧</td></tr>
<tr><td>1</td><td>表示数据帧为 Service 帧</td></tr>
<tr><td rowspan="2">Request not response</td><td rowspan="2">表示该数据帧是请求帧或是应答帧</td><td>0</td><td>表示该帧是 Response 应答帧</td></tr>
<tr><td>1</td><td>表示该帧是 Request 请求帧</td></tr>
<tr><td>Node ID</td><td colspan="3">1：Node ID 由 7bit 组成，其中 0 是保留 ID，代表一个未知的节点。
2：Node ID 取值为 1-127，包含 1-127，其中 126，127 是保留 ID。
3：Node ID 分为 Source Node ID 和 Destination Node ID。
4：Source Node ID 表示节点自身的 ID。
5：Destination Node ID 表示对方的节点 ID。
6：只有 Service 帧才会有 Destination Node ID，需要应答。</td></tr>
</table>

4.1.3 CAN Payload

CANBus2.0B 规定CAN 总线传输每一帧数据最多8Byte ，TM-UAVCAN 协议规定，将8Byte Payload 划分为两部分，包含Transfer Payload 与Tail byte ， 如下图所示：

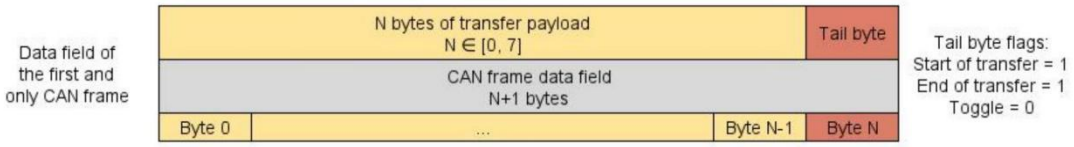
CAN payload

Field name	Transfer payload	Start of transfer / End of transfer / Toggle / Transfer ID							
Payload byte	Up to 7 bytes	Tail byte							
Bit position		7	6	5	4	3	2	1	0

有些数据包有效数据可能会超过 7Byte ， TM-UAVCAN 协议规定，不超过 7Byte 数据帧采用 Single frame Transfer 格式传输，超过7Byte 数据帧采用 Multi frame Transfer 格式传输，以下是对Single frame Transfer 与 Multi frame Transfer 的定义。

Single frame Transfer： 单帧传输用来处理传输的数据不超过 7 个字节的数据流。

Single frame transfer



Multi frame Transfer： 多帧传输用来处理传输的数据超过7 个字节的数据流，此处引入2 个概念：

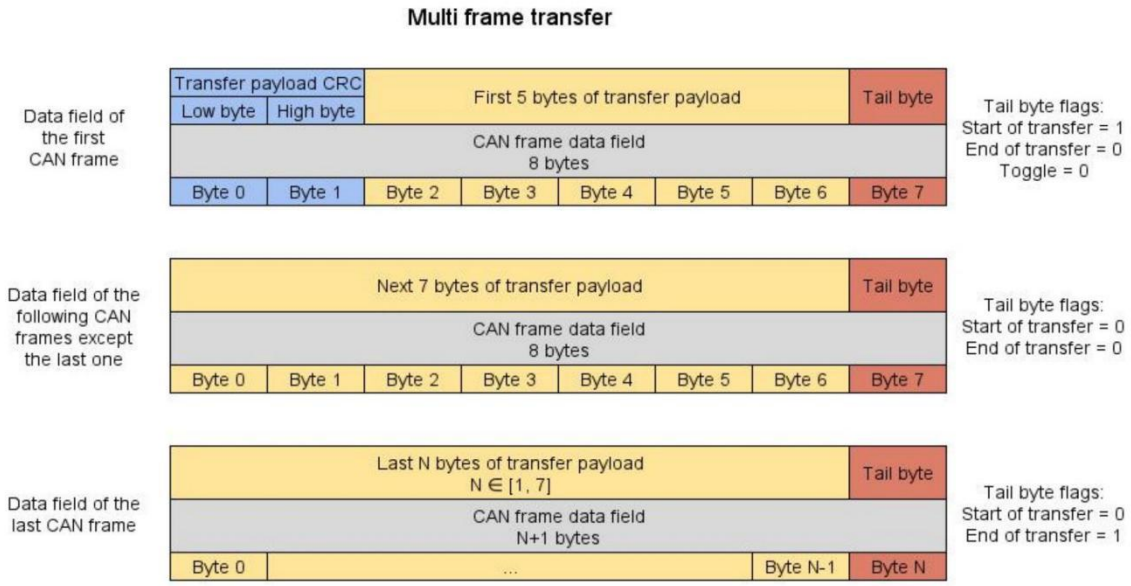
CRC和 翻转位。如下图所示，CRC 是计算所有的 payload 数据的，并且需要添加**数据签名（Signature）**。

Computation of transfer CRC

LSB				Data type signature (64-bit) Little-endian byte order (LSB first)			MSB	N bytes of payload		
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 0	...	Byte N-1
Input of the transfer CRC function										
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	...	Byte N+7

注：带数据签名的CRC 计算方式如上图所示，在有效数据(N bytes of payload)前加上64bit 的数据签名值，然后调用附录给定的CRC 算法对Signature Byte 0 至 Payload Byte N-1 进行校验计算，用户计算此CRC 值时调用附录算法时，按照以下顺序调用即可：添加 64Bit 数据签名：`SignatureCrc = crcAddSignature(0xFFFF, SignatureValue);` 添加 Payload 段数据：`CRC = crcAdd(SignatureCrc, NBytesOfPayload, N);`

Multi frame Transfer 数据帧传输格式：



名词解释

名词	解释说明
数据帧	1：CAN 单次能够发送的数据流，最多可传输 8 个字节。
数据包	1：一包数据可能包含多个数据帧，帧数由传输的数据量决定。
Transfer Payload	1：每帧数据帧传输的有效数据，0~7 字节。
Tail byte	1：每帧数据帧中 Payload 最后一个字节，附加的协议层字段信息。
Signature	1：对于每一个不同的数据帧类型，都有一个 64Bit 的数据签名值。 2：对于一包数据需要多帧传输时，需要使用此签名值参与 CRC 校验。 3：每一个数据类型的具体数值见电调通讯章节数据帧类型表。
Start of transfer	1：对于 Single frame transfer ， start of transfer 位永远为 1。 2：对于 Multi frame transfer ，如果当前帧是数据包的首帧，该位为 1 ，否则为 0。
End of transfer	1：对于 Single frame transfer ， End of transfer 这一 bit 位永远为 1。 2：对于 Multi frame transfer ，如果当前帧是数据包的最后一帧，该位为 1 ，否则为 0。
Toggle bit	1：对于 Single frame transfer ， Toggle bit 这一位永远为 0。 2：对于 Multi frame transfer ，数据包的首帧该位为 0 ，此后每发一帧该位翻转一次。
Transfer ID	1：数值范围 0~31。 2：对于相同 Data Type ID 的数据，每发送一包数据，Transfer ID 加 1 ，0~31 循环累加。 3：对于同一包数据中的多个数据帧，该值不变。

4.2 CAN 电调工作机制

所有 ESC 用 CAN 总线连接，连接方式常用T 型拓补与星型拓补；输入油门可以使用 PWM 模拟油门或者使用 CAN 数字油门，可设值 PWM 油门和 CAN 油门哪个优先使用，如设置 PWM 油门优先，则优先使用 PWM 油门，当 PWM 油门异常但 CAN 油门正常，则自动切换到 CAN 油门，若 CAN 油门也异常，则认为油门异常。

4.2.1 功能

- 电调拥有自己的唯一序列号（SN）。
- 电调上电自动自检，发送自检查询命令可查询自检状态。
- 支持电调节点 ID 设置。
- 支持CAN 总线速率设置。
- 数据上报速率可调节。
- 其它功能视具体电调型号而定。
- 模拟数字双油门输入

4.2.2 总线带宽

TM-UAVCAN 支持多种总线速率设置，可根据实际设备数目情况以及线材设置对应速率，默认 1MHz。

4.3 数据帧类型列表

4.3.1 数据帧类型列表

数据帧类型列表					
帧类型	**帧 ID(十进制)**	**签名码**	**优先级**	**最大字节数**	**数据帧描述**
消息帧	**1030**	0x217F5C87D7EC951D	**LOW**	**36Byte**	RawCommand 原始油门数据
消息帧	**1033**	0x948F5E0B33E0EDEE	**LOW**	**27Byte**	ParamCfg 参数设置数据包
消息帧	**1034**	0xA9AF28AEA2FBB254	**LOW**	**14Byte**	ESCStatus 电调反馈数据
消息帧	**1038**	0xCE2B6D6B6BDC0AE8	**LOW**	**260Byte**	PUSHSCI 上位机下发至电调指令
消息帧	**1039**	0xAACF9B4B2577BC6E	**LOW**	**260Byte**	PUSHCAN 电调数据反馈指令
消息帧	**1332**	0x462875A0ED874302	**LOW**	**74Byte**	ParamGet 参数获取数据包

4.4 数据帧类型说明

4.4.1 RawCommand(1030)

RawCommand 发送帧（N 轴多帧，14*N Bit 有效数据）

数据帧字节序号	参数名称	字节数	数据类型	内容
1-N	油门指令	N(<=20)	INT14	14BIT 油门数据，按照 0-N 的顺序排布，不足 8bit 部分自动填充

电机收到后按照自己的电调 ID 解析油门信息，执行对应动作。

说明：

1：RawCommand 发送油门的命令，为广播帧不需要应答，总线上所有 ESC 同时接收解析。

2：每个油门通道占用 14bit，最高位 bit13 为符号位，数字油门范围为-8191~8191，0 表示 0 油门，8191 表示 满油门，协议暂不支持负数字油门，给定负值判断为油门异常，状态位会返回油门异常，错误油门未达到 超时时间则保持上次正常油门，超时则油门会自动归零。

3：用户根据实际电调数目情况选择使用 N 轴 RawCommand 数据帧，建议单条 CAN 通道最多连接 8 个 CAN 节点以保证通信质量。

4：RawCommand 数据发送格式解析（以四轴为例）：

a：假设我们需要向四个通道发送数值为 1000(0x03E8)的数字油门

待发数据(HEX)：{3E8, 3E8, 3E8, 3E8}

内存格式(HEX)：{E8, 03, E8, 03, E8, 03, E8, 03}

内存格式(BIN)：{11101000, 00000011, 11101000, 00000011, 11101000, 00000011, 11101000, 00000011}

b：油门发送时将原 16Bit 转换为 14Bit 的油门数据(去掉原 16bit 数据最高 2bit)

原 16Bit 数据：

HEX:{0xE8,0x03,0xE8,0x03,0xE8,0x03,0xE8,0x03}

BIN:{1110_1000_0000_0011_1110_1000_0000_0011_1110_1000_0000_0011_1110_1000_0000_0011}

转换为 14Bit 数据(去掉原数据高字节最高 2bit，如红色标记部分)：

HEX：{1110_1000_0000_1111_1010_0000_0011_1110_1000_0000_1111_1010_0000_0011}

BIN：{0xE8,0x0F,0xA0,0x3E,0x80,0xFA,0x03}

发送数字油门数据：

Throttle Data[7]={ 0xE8 , 0x0F , 0xA0 , 0x3E , 0x80 , 0xFA , 0x03}

c：八轴数据格式与四轴数据格式转换方式一致，需添加数据签名校验。

5：CAN 数字油门与 PWM 油门映射关系如下。

CAN 油门周期 2.5ms 400HZ

PWM 油门频率 50-500HZ

植保： 油门脉宽范围为 1050-1950us

行业： 油门脉宽范围为 1040-1940us

a：油门数据用 14bit 表示 -1 ~1 的区间（-8191~+8191）（分辨率为 1/ 16384）0 代表 0 油门，1 代表顺时针最大油门。-1 逆时针最大油门，目前 ESC 支持 PWM 与 CAN 两种油门控制方式。对于传统 PWM 油门， 固化油门为 1040us~1940us 的脉宽(可辨识油门脉宽范围为 800-2200us)，对于 CAN 数字油门，目前暂不支持负油门（负数字油门将会反馈油门异常）。

b：电调内部将 0~8191 的数字油门映射到 1040us~1940us 的 PWM 油门，再去控制电机转动，设置为 CAN 油门方式下，原 PWM 输入口会产生数字油门对应的 PWM 脉宽。

4.4.2 ParamCfg(1033)

ParamCfg 发送帧（27Byte 有效数据）

参数设置发送指令(1033)

序号	参数名称	字节数	数据类型	内容
1	电调 ID	1	UINT8	电调 ID0-19
2-5	待配置电调 UUID	4	UINT32	电调唯一 32 位 ID
6-7	电调 ID	2	UINT16	设置电调 ID
8-9	过压保护阈值	2	UINT16	设置电调过压保护阈值
10-11	过流保护阈值	2	UINT16	设置电调过流保护阈值
12-13	过温保护阈值	2	UINT16	设置电调过温保护阈值
14-15	加速度限制	2	UINT16	设置加速度限制
16-17	减速限制	2	UINT16	设置减速限制
18-19	旋转方向	2	UINT16	设置旋转方向
20	进角设置	1	UINT8	设置进角大小，1-29 映射 1-29°
21	油门优先级和定桨设置	1	UINT8	开启定桨：高四位 1000 关闭定桨：高四位 0000 PWM：低四位 0001 CAN：低四位 0010
22-23	LED 设置	2	UINT16	0-2bit:RGB 灯使能开关，3bit:灯闪烁开关，4-15bit:灯闪烁频率(0.1HZ)
24	总线速率	1	UINT8	0-1Mbps 1-500Kbps 2-250Kbps 3-125Kbps 4-100Kbps 5-50Kbps
25-26	数据回报速率	2	UINT16	回传速率 0-400HZ
27	保存选项	1	UINT8	0-临时设置 1-永久设置

说明：

1：设置电调参数，0xFF 为缺省值，当前值为 0XFF/0xFFFF 则不做修改。设置完成后回复参数设置回报指令(1332)。

2：可以发送全部为 0XFF 的数据帧来获取总线上所有电调的参数信息。

4.4.3 ESCStatus(1034)

EscStatus 返回帧（14Byte 有效数据）

电调状态反馈数据

序号	参数名称	字节数	数据类型	内容
1-4	状态位	4	UINT32	状态位
5-6	电压值	2	FLOAT16	电压值，单位 V
7-8	电流值	2	FLOAT16	电流值，单位 A
9-10	温度值	2	FLOAT16	温度值，单位 K
11-13	转速值	2~3	INT18	转速值，单位 RPM
13-14	油门值	<1	UINT7	油门值，0-100 代表 0-100%
14	电调编号	<1	UINT5	电调编号，0-19 代表 1-20 通道

状态位结构：

状态位结构表							
Bit16-31	Bit12-15			Bit11	Bit10	Bit9	Bit8
编码器数值 0-16383 代 表 0-360°	1-关断模式 2-空闲模式 3-缓启动模式 4-运转模式 5-缓减速模式 6-错误模式 7-低速正转收桨模式 8-低速反转收桨模式			编码器异常	下桥故障	上桥故障	运放异常
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
堵转	电容过温	MOS 过温	油门异常	油门丢失	过流	欠压	过压

说明：

1：EscStatus 发送帧为广播帧不需要应答，总线上所有节点同时接收，仅飞控或调参器参与解析。

2：每个电调按照自己的节点 ID 和电调编号来反馈状态数据，需要提前设置好各个电调的 ID 号避免冲突。

3：用户根据实际需求设置返回速率，避免过高的回报速率带来的高总线负荷与处理负荷。

4.4.4 PUSHSCI(1038)

PUSHSCI 发送帧（最大 260Byte 有效数据）

PUSHSCI 上位机传输数据至电调指令数据结构

序号	参数名称	字节数	数据类型	内容
1-4	数据序列号	4	UINT32	数据包序列号
5-N	数据原始值	0-255	UINT8	数据包数据

数据包结构:

1. 设置电机当前位置为零点

序号	参数名称	字节数	数据类型	内容
1-2	帧头	2	UINT16	0xEC、0x96
3	帧 ID	1	UINT8	0x08
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	0xA1~0xA9:对应 1~9
6	帧长度	1	UINT8	0x07
7	校验和	1	UINT8	（1~6）字节之和取低 8 位

电机收到后当前位置设置为 0 点

2.FOC 参数设置数据包

序号	参数名称	字节数	数据类型	内容
1-2	帧头	2	UINT16	0xEC、0x96
3	帧 ID	1	UINT8	0x1E
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	飞机上有 9 个动力单机 0xA1~0xA9:对应 1~9#单机 动力系统的电调地址可以设置，方便跟实际的装配位置对应。(28027 默认 0xA1,103 按照实际 ID 来)
6	帧长度	1	UINT8	0x1F
7-8	电调 ID	2	UINT16	设置电调 ID
9-10	过压保护阈值	2	UINT16	设置电调过压保护阈值
11-12	过流保护阈值	2	UINT16	设置电调过流保护阈值
13-14	过温保护阈值	2	UINT16	设置电调过温保护阈值
15-16	鸣叫音量	2	UINT16	设置电调鸣叫音量
17-18	加速度限制	2	UINT16	设置加速度限制
19-20	减速限制	2	UINT16	设置减速限制
21-22	旋转方向	2	UINT16	设置旋转方向
23-24	油门优先级	2	UINT16	1：PWM 2：CAN(24bit: 位置环使能)
25-26	LED 设置	2	UINT16	0-2bit:RGB 灯使能开关，3bit:灯闪烁开关，4-15bit:灯闪烁频率(0.1HZ)
27	总线速率	1	UINT8	0-1Mbps 1-500Kbps 2-250Kbps 3-125Kbps 4-100Kbps 5-50Kbps
28-29	数据回报速率	2	UINT16	回传速率 0-400HZ
30	保存选项	1	UINT8	0-临时设置 1-永久设置
31	校验和	1	UINT8	（1~28）字节之和取低 8 位

3. FOC 参数及状态信息获取数据包

序号	参数名称	字节数	数据类型	内容
1-2	帧头	2	UINT16	0xEC、0x96
3	帧 ID	1	UINT8	0x1A
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	0xA1~0xA9:对应 1~9,或者 0xFF 获取总线全部参数
6	帧长度	1	UINT8	7
7	校验和	1	UINT8	（1~6）字节之和取低 8 位

电调收到指令后按照自身类型(FOC)返回对应参数包

4.控制器输出控制数据帧（控制器→动力系统）

序号	参数名称	字节数	数据类型	内容

1-2	帧头	2	UINT16	0xEC、0x96
3	帧 ID	1	UINT8	0x06
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	飞机上有 9 个动力单机 0xA1~0xA9:对应 1~9#单机 动力系统的电调地址可以设置，方便跟实际的装配位置对应。
6	帧长度	1	UINT8	0x0C
7	控制模式	1	UINT8	0x00 非锁桨模式，正常响应飞控 PWM，同时作为动力系统查询命令。 0xEE:电机低速正转收桨 0x22:电机低速反转收桨 0x88:电机锁定当前位置 0x66:电机处于自由态 0x55:电机占空比环 0x5A:电机反向占空比环 0x44:电机电流环 0x4A:电机反向电流环 0x33:电机速度环 0x3A:电机反向速度环 0x11:电机位置环 0x1A:电机反向位置环 0xBB:电机电流刹车环
8-9	电机控制类型 油门百分比/转速/位置	2	UINT16	0-100 对应 0-100%占空比 0-10000 对应 0-10000rpm 0-16383 对应 0-360°
10-11	预留	2	UINT16	0x00
12	校验和	1	UINT8	（1~11）字节之和取低 8 位

说明：

1：PUSHSCI 传输数据至电调指令发送帧为广播帧不需要应答，总线上所有节点同时接收并解析。

2：数据帧中包含 ID 位，每个电调按照自己的节点 ID 和电调编号来解析指令，需要提前设置好各个电调的 ID 号避免错误响应。

3：用户根据实际需求设置发送速率，总线上总帧率应该控制在 2400 帧(CAN 速率 1Mbps)以内，避免过高的回报速率带来的高总线负荷与处理负荷。

4.4.5 PUSHCAN(1039)

PUSHCAN 返回帧（最大 260Byte 有效数据）

PUSHCAN 电调数据反馈指令数据结构

序号	参数名称	字节数	数据类型	内容
1-4	数据序列号	4	UINT32	数据包序列号
5-N	数据原始值	0-255	UINT8	数据包数据

数据包结构:

1. 动力系统反馈状态数据帧（动力系统→控制器）

序号	参数名称	字节数	数据类型	内容
1-2	帧头	2	UINT16	0x7B、0x8C
3	帧 ID	1	UINT8	0x15
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	飞机上有 9 个动力单机 0xA1~0xA9:对应 1~9#单机 动力系统的电调地址可以设置，方便跟实际的装配位置对应。
6	帧长度	1	UINT8	0x1C
7	动力单机 ID 的系统状态	1	UINT8	0x00：非锁桨态 0x11：正常锁桨 0xCC: 系统故障
8-9	电机当前位置	2	UINT16	0-360°，分辨率：0.01°
10-11	接收 PWM 值	2	UINT16	分辨率：0.1us
12	实际输出油门值	1	UINT8	分辨率：0.4%
13-14	电机转速	2	INT16	分辨率：1rpm
15-16	电压	2	UINT16	分辨率：0.01V
17-18	电流	2	INT16	分辨率：0.01A
19-20	温度	2	UINT16	分辨率：0.01℃
21-22	电机错误	2	UINT16	错误位数据
23	电机状态	1	UINT8	0x00
24-25	上调上电次数	2	UINT16	单位秒
26-27	电调运行时间	2	UINT16	次
28	校验和	1	UINT8	（1~27）字节之和取低 8 位

2. FOC 参数设置回传数据包

序号	参数名称	字节数	数据类型	内容
1-2	帧头	2	UINT16	0x7B、0x8C
3	帧 ID	1	UINT8	0x1E
4	帧计数	1	UINT8	0-255 循环累加
5	动力单机 ID	1	UINT8	飞机上有 9 个动力单机 0xA1~0xA9:对应 1~9#单机 动力系统的电调地址可以设置，方便跟实际的装配位置对应。
6	帧长度	1	UINT8	0x37
7-22	电调硬件版本	16	UINT16	电调版本号
23-38	电调软件版本	16	UINT16	电调软件版本
39-40	加速度限制	2	UINT16	加速度限制
41-42	减速限制	2	UINT16	减速限制
43-44	旋转方向	2	UINT16	旋转方向
45-46	位置环使能标志	2	UINT16	位置环使能标志
47-48	油门优先级	2	UINT16	1：PWM 2：CAN
49-50	LED 设置	2	UINT16	0-2bit:RGB 灯使能开关，3bit:灯闪烁开关，4-15bit:灯闪烁频率(0.1HZ)
51	总线速率	1	UINT8	0-1Mbps 1-500Kbps 2-250Kbps 3-125Kbps 4-100Kbps

				5-50Kbps
52-53	数据回报速率	2	UINT16	回传速率 0-400HZ
54	保存选项	1	UINT8	0-临时设置 1-永久设置
55	校验和	1	UINT8	（1~54）字节之和取低 8 位

4.4.6 ParamGet(1332)

ParamGet 返回帧（最大 74Byte 有效数据）

参数设置回报指令(1332)数据结构

序号	参数名称	字节数	数据类型	内容
1	电调 ID	1	UINT8	电调 ID0-19
2-5	待配置电调 UUID	4	UINT32	电调唯一 32 位 ID
6-7	电调 ID	2	UINT16	待获取电调 ID，0XFF 则为总线上全部电调同时反馈
8-9	过压保护阈值	2	UINT16	获取电调过压保护阈值
10-11	过流保护阈值	2	UINT16	获取电调过流保护阈值
12-13	过温保护阈值	2	UINT16	获取电调过温保护阈值
14-15	加速度限制	2	UINT16	获取加速度限制
16-17	减速限制	2	UINT16	获取减速限制
18-19	旋转方向	2	UINT16	获取旋转方向
20	进角设置	1	UINT8	获取进角大小，1-29 映射 1-29°
21-22	上电次数	2	UINT16	获取电调上电总次数
23-26	上电时间	4	UINT32	获取电调上电总时间(秒)
27-30	生产时间	4	UINT32	获取电调生产日期
31-34	故障次数	4	UINT32	获取电调出现故障的累计次数
35	油门优先级和定桨设置	1	UINT8	开启定桨：高四位 1000 关闭定桨：高四位 0000 PWM：低四位 0001 CAN：低四位 0010
36-37	LED 设置	2	UINT16	0-2bit:RGB 灯使能开关，3bit:灯闪烁开关，4-15bit:灯闪烁频率(0.1HZ)
38	总线速率	1	UINT8	0-1Mbps 1-500Kbps 2-250Kbps 3-125Kbps 4-100Kbps 5-50Kbps
39-40	数据回报速率	2	UINT16	回传速率 0-400HZ
41	保存选项	1	UINT8	0-临时设置 1-永久设置
42-73	保留数据	32	UINT32	制造商保留数据

说明：

1：ParamGet 参数获取返回指令为广播帧不需要应答，总线上所有节点同时接收。

2：数据帧中包含 ID 位，仅飞控或者调参器需要对数据进行解析处理，该帧为参数设置、参数获取的反馈指令，用户可按需选用。

3：用户根据实际需求设置发送速率，总线上运行时总帧率应该控制在 2400 帧(CAN 速率 1Mbps)以内，非运行时总帧率应该控制在 4000 帧(CAN 速率 1Mbps)以内，避免过高的回报速率带来的高总线负荷与处理负荷。

5 附件

5.1 CRC support：

CRC 算法描述:

Name: CRC- 16-CCITT-FALSE
Description: http://reveng.sourceforge.net/crc-catalogue/16.htm#crc.cat.crc-16-ccitt-false

Initial value: 0xFFFF

Poly: 0x1021

Reverse: no

Output XOR: 0

Check: 0x29B1

```
/*
 * CRC functions
 */
 uint16_t crcAddByte(uint16_t crc_val, uint8_t byte)
{
   crc_val ^= (uint16_t) ((uint16_t) (byte) << 8);
   for (uint8_t j = 0; j < 8; j++)
   {
      if (crc_val & 0x8000U)
      {
         crc_val = (uint16_t) ((uint16_t) (crc_val << 1) ^ 0x1021U);
      }
      else
      {
         crc_val = (uint16_t) (crc_val << 1);
      }
   }
   return crc_val;
}

 uint16_t crcAddSignature(uint16_t crc_val, uint64_t data_type_signature)
{
   for (uint16_t shift_val = 0; shift_val < 64; shift_val = (uint16_t)(shift_val + 8U))
   {
      crc_val = crcAddByte(crc_val, (uint8_t) (data_type_signature >> shift_val));
   }
   return crc_val;
}
```

```
uint16_t crcAdd(uint16_t crc_val, const uint8_t* bytes, size_t len)
{
   while (len--)
   {
      crc_val = crcAddByte(crc_val, *bytes++);
   }
   return crc_val;
}
```

5.2 float16 support：

```
/*
UAVCAN 浮点数据类型转换
*/
  union FP32
  {
    uint32_t u;
    float f;
  };
const union FP32 f32inf = { 255UL << 23 };
const union FP32 f16inf = { 31UL << 23 };
const union FP32 magic  = { 15UL << 23 };
const uint32_t sign_mask = 0x80000000U;
const uint32_t round_mask = ~0xFFFU;
 uint16_t ConvertFloatToFloat16(float value)
 {
    union FP32 in;
    in.f = value;
    uint32_t sign = in.u & sign_mask;
    in.u ^= sign;
    uint16_t out = 0;

    if (in.u >= f32inf.u)
    {
        out = (in.u > f32inf.u) ? (uint16_t)0x7FFFU : (uint16_t)0x7C00U;
    }
    else
    {
          in.u &= round_mask;
```

```
            in.f *= magic.f;
            in.u -= round_mask;
            if(in.u > f16inf.u)
            {
                in.u = f16inf.u;
            }
             out = (uint16_t)(in.u >> 13U);
     }
     out |= (uint16_t)(sign >> 16U);
     return out;
  }
float ConvertFloat16ToFloat(u16 value)
{
   const union FP32 magic = { (254UL - 15UL) << 23U };
   const union FP32 was_inf_nan = { (127UL + 16UL) << 23U };
   union FP32 out;

   out.u = (value & 0x7FFFU) << 13U;
   out.f *= magic.f;
   if (out.f >= was_inf_nan.f)
   {
       out.u |= 255UL << 23U;
   }
   out.u |= (value & 0x8000UL) << 16U;
   return out.f;
}
```

5.3 DSDL 配置参考

5.3.1 1030.RawCommand

struct { uint8_t len; int14_t data[20]; }cmd;

5.3.2 1033.ParamCfg

uint8 esc_index
uint32 esc_uuid
uint16 esc_id_set
uint16 esc_ov_threshold
uint16 esc_oc_threshold
uint16 esc_ot_threshold
uint16 esc_acc_threshold
uint16 esc_dacc_threshold
int16 esc_rotate_dir
uint8 esc_timing
uint8 esc_signal_priority
uint16 esc_led_mode
uint8 esc_can_rate
uint16 esc_fdb_rate
uint8 esc_save_option

5.3.3 1034.ESC_STATUS

uint32 error_count
float16 voltage
float16 current
float16 temperature
int18 rpm
uint7 power_rating_pct
uint5 esc_index

5.3.4 1038.PUSHSCI

uint32 data_sequence
struct { uint8_t len; uint8_t data[255]; }data;

5.3.5 1039.PUSHCAN

uint32 data_sequence
struct { uint8_t len; uint8_t data[255]; }data;

5.3.6 1332.ParamGet

uint8 esc_index
uint32 esc_uuid
uint16 esc_id_req
uint16 esc_ov_threshold
uint16 esc_oc_threshold
uint16 esc_ot_threshold
uint16 esc_acc_threshold
uint16 esc_dacc_threshold
int16 esc_rotate_dir
uint8 esc_timing
uint16 esc_startup_times
uint32 esc_startup_duration
uint32 esc_product_date
uint32 esc_error_count
uint8 esc_signal_priority
uint16 esc_led_mode
uint8 esc_can_rate
uint16 esc_fdb_rate
uint8 esc_save_option
struct { uint8_t len; uint8_t data[32]; }rsvd;

5.4 参考链接

1.Dronecan 项目:https://dronecan.github.io
2.Dronecan 配置示例:https://github.com/dronecan/libcanard/tree/master/examples
3.ardupilot 项目:https://ardupilot.org
4.PX4 项目:https://px4.io

6 版本历史

日期	版本	变更
2024.5.16	V2.1	第一版协议发布
2024.7.3	V2.2	1、修正了一些文档排版上的错误 2、增加了版本历史 3、删除了 4.3 章节中一些没有的数据帧类型
2025.03.20	V2.3	1、更改了电调状态(ESC_Status)中温度由摄氏度改为开尔文温度 2、更改了 4.3.1 小节对于数据帧类型的描述 3、修正了 PUSHSCI(1038)数据中的电机控制类型 4、修正了 ParamCfg(1033)数据中 21 字节油门优先级内容