Learning MavLink

`参考：*Mavlink Tutorial for Absolute Dummies(Part-I)* by Shyam Balasubramanian`

1. 什么是 MavLink?

Micro Air Vehicle Link是用于小型无人机通信的协议（protocol）。
MavLink是只有头文件构成的消息结构库。（*header-only message marshaling library*）

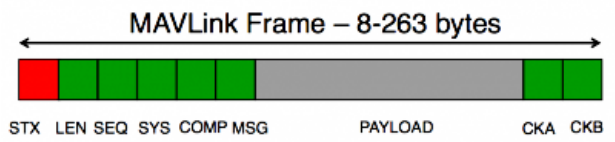
- MAVLink 最早由 Lorenz Meier 于2009年在 LGPL 证书下发布。

详见：
MAVLink_wiki，
mavlink官网，
github

2. MavLink

2.1 MavLink 数据包

MavLink数据包的结构源自 CAN 总线和 SAE AS-4 标准。



字节位	内容	值	说明
0	开始标志位	v1.0: 0xFE (v0.9: 0x55)	*
1	载荷长度	0~255	表示有效载荷（payload）的长度
2	数据包序列号	0~255	每个组件对其发送序列计数，允许丢包检查
3	系统 ID	1~255	发送消息的系统的 ID
4	组件 ID	0~255	发送消息的组件的 ID
5	消息 ID	0~255	定义了有效载荷（payload）的意义，用于解码
6~(N+6)	数据	0~255	消息数据内容，取决于消息 ID

(n+7) ~ (n+8)	校验和

- 数据包的最小长度为 8 字节，用于无有效载荷的应答数据包；
- 数据包的最大长度为 263 字节，用于满载。

2.2 支持数据格式

MavLink支持固定大小的整型数据、IEEE 754 单精度浮点型、数组类型和协议自动添加的 mavlink_version 域。

- char - Characters / strings
- uint8_t - Unsigned 8 bit
- int8_t - Signed 8 bit
- uint16_t - Unsigned 16 bit
- int16_t - Signed 16 bit
- uint32_t - Unsigned 32 bit
- int32_t - Signed 32 bit
- uint64_t - Unsigned 64 bit
- int64_t - Signed 64 bit
- float - IEEE 754 single precision floating point number
- double - IEEE 754 double precision floating point number
- uint8_t_mavlink_version - Unsigned 8 bit field automatically filled on sending with the current MAVLink version （it cannot be written, just read from the packet like a normal uint8_t field）

2.3 性能

协议设计的目标是传输速度和安全性。它允许消息内容检查、消息丢失检测，每个数据包需要 6 字节报头。

链接速度	硬件	刷新率	有效载荷	浮点值
115200	XBee Pro 2.4 GHz	50 Hz	224 bytes	56
115200	XBee Pro 2.4 GHz	100 Hz	109 bytes	27
57600	XBee Pro 2.4 GHz	100 Hz	51 bytes	12
9600	XBee Pro XSC 900	50 Hz	13 bytes	3
115200	XBee Pro XSC 900	20 Hz	42 bytes	10

2.2 MavLink 消息

MavLink消息（message）中，只有心跳包（heartbeat）消息是必选项，其余消息都是可选项。

MAVLink消息由XML定义，然后转换为相应的代码（C/C++, C#, Python）。详细描述：创建新MavLink消息。

XML编译为代码：MAVLink Generator (C/C++, Python)

CRC 冗余校验（CRC Extra Calculation）

3. Waypoint 协议

> 参考：mavlink/waypoint_protocol

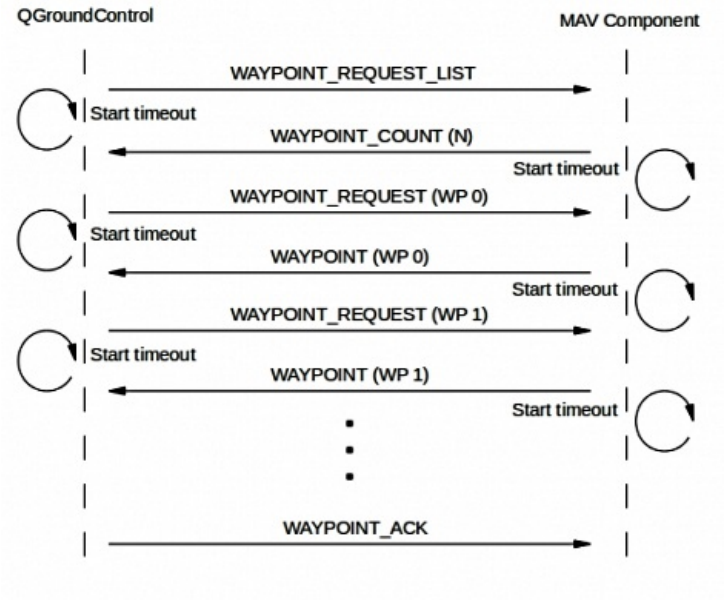
- 定义： **Waypoint 协议**：用于描述航点信息怎样在飞行器传输（发送、接收）。
- 目的：确保信号传输的**一致性**。

3.1 通信与状态机

协议包含不同的消息处理，每次消息交换可能成功完成或失败（航点信息接收端保持上一个状态）。当线路为**空闲(IDLE)**状态时，可以通过特殊指令开始发送（transaction）。指令开始传输时，线路状态置位忙碌。协议通过这个简单的状态机执行。

发送消息的同时，发送端启动计时器。到达指定的时间未收到回复，重发该消息。超过重发次数，发送失败。该重发机制要求每个组件必须有处理重复消息的能力。

3.2 读取 Waypoint list



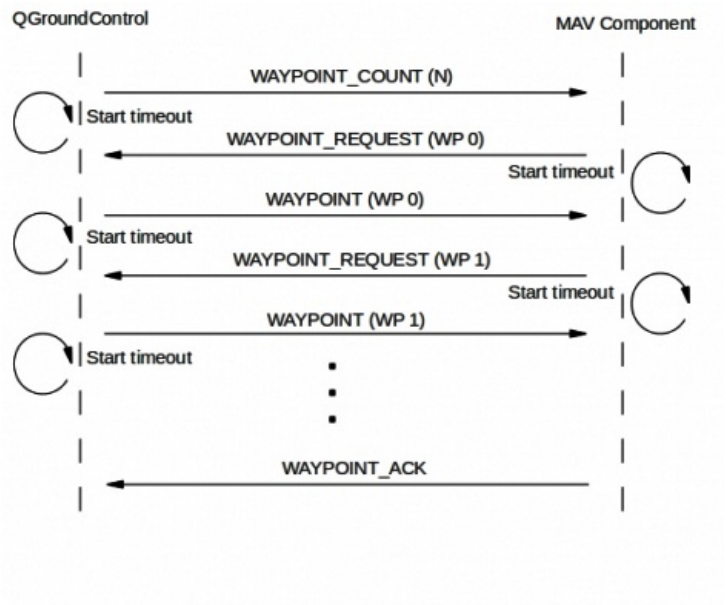
从飞行器取回所有 Waypoint：

- 发送 WAYPOINT_REQUEST_LIST
- 回复 WAYPOINT_COUNT（航点总数）

- 发送 WAYPOINT_REQUEST （WP_number）
- 回复 WAYPOINT （WP_number）
- 发送 WAYPOINT_ACK，结束消息传送。

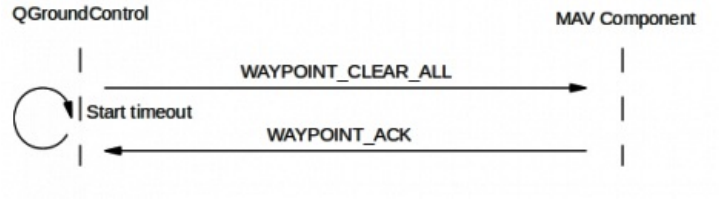
接收端监听 WAYPOINT_REQUEST 消息，直到收到 WAYPOINT_ACK 或 超时 或开始其他消息传送。

3.3 写入 Waypoint list



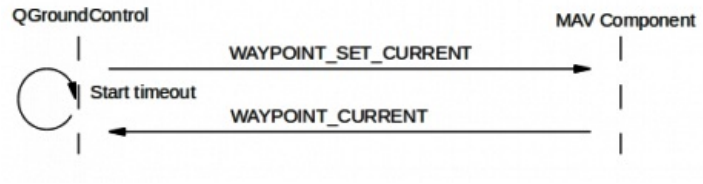
对比3.2节读取 Waypoint 。

3.4 清除 Waypoint list



发送 WAYPOINT_CLEAR_ALL，飞行器清除所有航点后，回复 WAYPOINT_ACK 。

3.5 设置新当前 Waypoint



发送 WAYPOINT_SET_CURRENT ， 飞行器当前航点改变后，回复 WAYPOINT_CURRENT 和当前序列号。

3.6 到达 Waypoint

飞行器到达航点，会广播 WAYPOINT_REACHED 消息。无需应答，也不保证接收到。

3.7 当前 Waypoint 改变

当飞行器改变其当前航点目标（new current target），会广播 WAYPOINT_CURRENT 消息。无需应答，也不保证能够接收到。
建议以较短延时发送两次，保证较高的送达率。

3.8 Waypoint 文件格式

航点文件格式，请注意：各个参数之间的空白部分，是 [tab]（“\t”）。

格式：

```
QGC WPL <VERSION>
<INDEX> <CURRENT WP> <COORD FRAME> <COMMAND> <PARAM1> <PARAM2> <PARAM3> <PARAM4> <PARAM5/X/LONGITUDE> <PARAM6/Y/LATITUDE> <PARAM7/Z/ALTI
```

示例：

```
QGC WPL 110
0    1    0    16    0.149999999999999994    0    0    0    8.54800000000000004    47.3759999999999977    550    1
1    0    0    16    0.149999999999999994    0    0    0    8.54800000000000004    47.3759999999999977    550    1
2    0    0    16    0.149999999999999994    0    0    0    8.54800000000000004    47.3759999999999977    550    1
```

4. 参数协议

板载参数接口允许读写参数（RAM或EEPROM）。该操作既可以在微控制器（microcontroller (e.g. the pxIMU with ARM7) ）上执行，也可以在标准软件（e.g. Linux 进程 px_multitracker ）。

4.1 数据类型

MAVLink v1.0 支持以下数据类型（官网未更新）：[2016/4/26 21 31]

- uint32_t （MAV_PARAM_TYPE_UINT32）

- int32_t （MAV_PARAM_TYPE_INT32）
- float （MAV_PARAM_TYPE_FLOAT）

注意：所有参数都以浮点型的 mavlink_param_union_t 发送。意味着：每个参数都将按字节转换为浮点类型（不是类型转换）。
用于扩展，缩放整型参数，提高最大精度。

```
mavlink_param_union_t param;

int32_t integer = 20000;

param.param_int32 = integer;
param.type = MAV_PARAM_TYPE_INT32;

// Then send the param by providing the float bytes to the send function
mavlink_msg_param_set_send(xx, xx, param.param_float, param.type, xx);
```

4.2 多系统和多组件支持

MavLink 支持在一条链路中存在多台飞行器。还支持在一台飞行器上，有多个支持MavLink的设备。

要获取一个系统的完整参数列表，发送 target_component 0（枚举：MAV_COMP_ID_ALL）。所有板载组件将以自身ID回复（或MAV_COMP_ID_ALL (0)）。

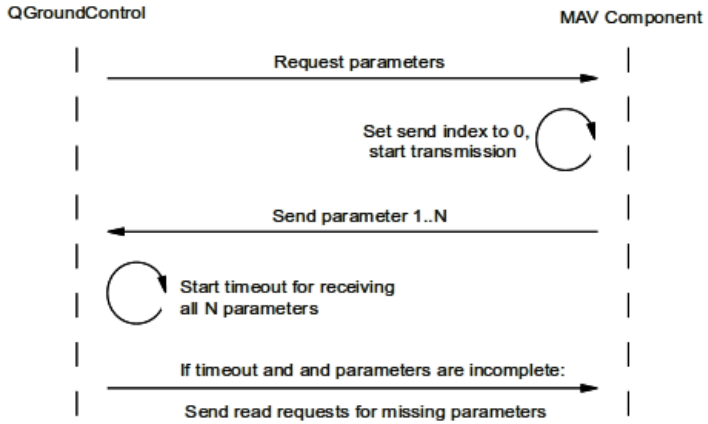
> QGC部分跳过

4.3 通信和状态机

板载参数定义为一个14字符的字符串，并存储了一个浮点值（IEEE 754 单精度，4字节）。此键值对（key→value）有许多重要的属性：

- 可读的变量名对于用户非常有用，而且它足够小；
- 地面站不需要提前知道机载部件的参数有哪些；
- 只要执行MavLink协议，可以支持未知的autopilots设备；
- 添加参数只需修改飞控代码。

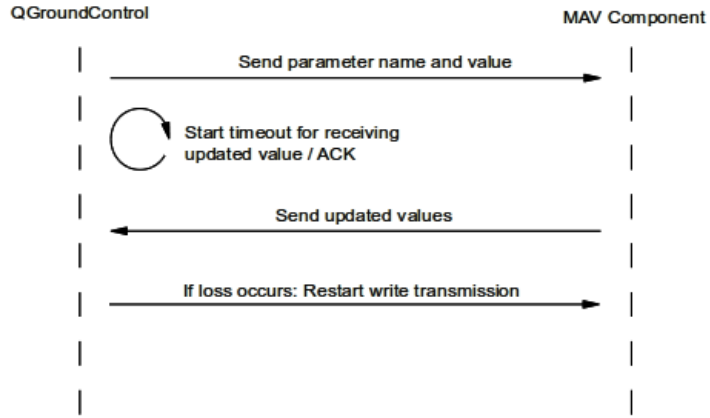
4.4 读取参数



发送 PARAM_REQUEST_LIST 消息，开始读取参数。

读取单个参数:
_PARAM_REQUEST_READ

4.5 写入参数



由于地面站本身没有参数列表，所以写入参数前，必须先读取参数。
写入完成后，飞行器将会返回 PARAM_VALUE 消息（新参数）。

4.6 板载存储

参数接口提供从存储器读取到RAM和从RAM写入存储器。从地面站写入参数到存储器，需要先传输，然后触发写入ROM命令。

4.7 QGC参数文件

QGC可以将当前板载参数值保存为TXT文档。该文档可以用于导入和传输。可用于快速配置相同的飞行器。

5. Onboard Integration

MavLink 库文件全部有**头文件**组成，只需要添加包含 mavlink/include 路径即可。

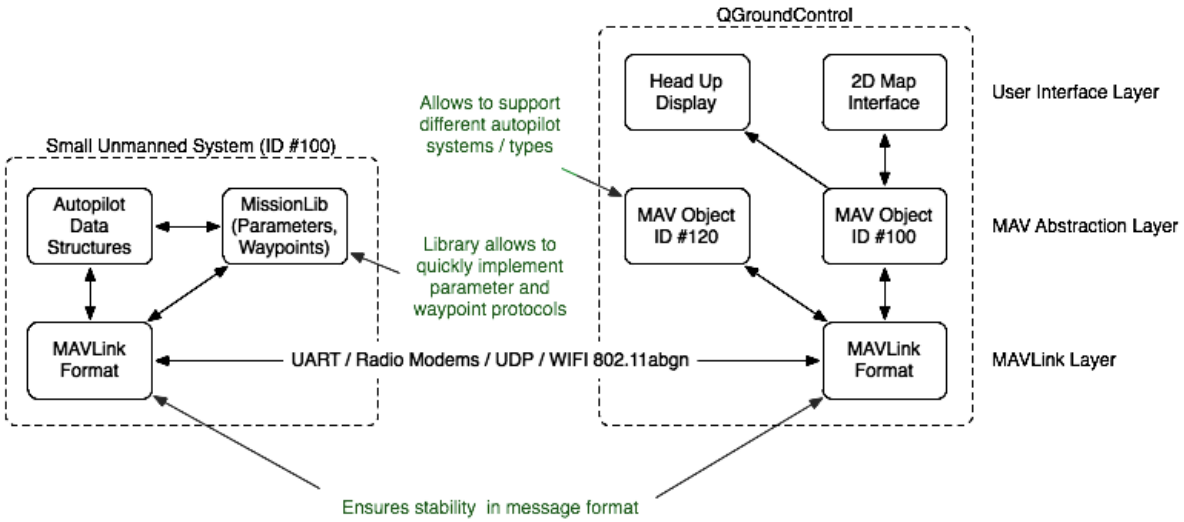
5.1 连接

MavLink是通用的，在地面站中，使用心跳包消息跟踪系统。只有心跳消息按时抵达，系统才会确认连接。所以，请确保心跳消息按时发送（默认为 1Hz，可自定义）。

5.2 整合

整合 MavLink 是非侵入式的，MavLink 提供了操纵参数和航点的库函数，自驾仪只需要从适当的数据结构中读取数据。

MavLink 的消息格式十分稳定，方便支持扩展。在QGC中可以使用非标准格式，不做讨论。参见：官方说明。



5.3 发送数据

```
/* The default UART header for your MCU */
#include "uart.h"
#include <mavlink/v1.0/common/mavlink.h>

mavlink_system_t mavlink_system;

mavlink_system.sysid = 20;   ///< ID 20 for this airplane
mavlink_system.compid = MAV_COMP_ID_IMU; ///< The component sending the message is the IMU, it could be also a Linux process
```

```
// Define the system type, in this case an airplane
uint8_t system_type = MAV_TYPE_FIXED_WING;
uint8_t autopilot_type = MAV_AUTOPILOT_GENERIC;

uint8_t system_mode = MAV_MODE_PREFLIGHT; ///< Booting up
uint32_t custom_mode = 0; ///< Custom mode, can be defined by user/adopter
uint8_t system_state = MAV_STATE_STANDBY; ///< System ready for flight

// Initialize the required buffers
mavlink_message_t msg;
uint8_t buf[MAVLINK_MAX_PACKET_LEN];

// Pack the message
mavlink_msg_heartbeat_pack(mavlink_system.sysid, mavlink_system.compid, &msg, system_type, autopilot_type, system_mode, custom_mode, sys

// Copy the message to the send buffer
uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);

// Send the message with the standard UART send function
// uart0_send might be named differently depending on
// the individual microcontroller / library in use.
uart0_send(buf, len);
```

注释较简单，不解释。待分析过源码后再添加。:)

5.4 接收数据

以下代码段描述怎样发送数据。因为运行时间很短，所以建议该函数以 mainloop rate 运行，且尽快清空串口缓存。

```
#include <mavlink/v1.0/common/mavlink.h>

// Example variable, by declaring them static they're persistent
// and will thus track the system state
static int packet_drops = 0;
static int mode = MAV_MODE_UNINIT; /* Defined in mavlink_types.h, which is included by mavlink.h */

/**
* @brief Receive communication packets and handle them
*
* This function decodes packets on the protocol level and also handles
* their value by calling the appropriate functions.
*/
static void communication_receive(void)
{
    mavlink_message_t msg;
    mavlink_status_t status;
```

```
    // COMMUNICATION THROUGH EXTERNAL UART PORT (XBee serial)

    while(uart0_char_available())
    {
        uint8_t c = uart0_get_char();
        // Try to get a new message
        if(mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {
            // Handle message

            switch(msg.msgid)
            {
            case MAVLINK_MSG_ID_HEARTBEAT:
            {
                  // E.g. read GCS heartbeat and go into
  // comm lost mode if timer times out
            }
            break;
            case MAVLINK_MSG_ID_COMMAND_LONG:
                // EXECUTE ACTION
                break;
            default:
                //Do nothing
                break;
            }
        }

        // And get the next one
    }

    // Update global packet drops counter
    packet_drops += status.packet_rx_drop_count;

    // COMMUNICATION THROUGH SECOND UART

    while(uart1_char_available())
    {
        uint8_t c = uart1_get_char();
        // Try to get a new message
        if(mavlink_parse_char(MAVLINK_COMM_1, c, &msg, &status))
        {
            // Handle message the same way like in for UART0
// you can also consider to write a handle function like
// handle_mavlink(mavlink_channel_t chan, mavlink_message_t* msg)
// Which handles the messages for both or more UARTS
        }

        // And get the next one
    }
```

```
    // Update global packet drops counter
    packet_drops += status.packet_rx_drop_count;
}
```

待分析源码后，另行注释。
其余部分，一并……（包括：Integration with convenience functions、Linux Integration）