

# Unmanned Aerial Vehicles Communication System

EE 496 Final Report  
Fall 2015

Department of Electrical Engineering  
University of Hawaii

Hualiang Li

December 3, 2015

Faculty Advisor: Professor Yingfei Dong

## **Abstract**

In the last few years, the interest in Unmanned Aerial Systems (UASs) and their use for an ever increasing range of applications have grown tremendously. This type of aircrafts can be controlled remotely or programmed to fly in an autonomous way. They have been used for different types of applications as individual entities, both in military and civil tasks. To make the UAVs more intelligent, it is desired to make the system to communicate with its ground control station, such that the ground control station monitor flight situations, verify targets, or modify flight modes in a real time. This project focuses on the construction and improvement of communication system between an UAV and a ground control station.

## **1. Introduction**

Recently, Amazon announced that they have been conducted a R&D project aimed at delivering packages to consumers' doorstep by "octocopter" mini-drones with a mere 30-minute delivery time [1]. This is a significant milestone for the application of Unmanned Aerial System (UAS). In the video Amazon released, the delivery drone is able to search for the delivery address. When it arrives the target address, a signal is sent to an Amazon control station. Then Amazon will send a notice to the buyer and ask the buyer to put a pre-made Amazon logo board on the desired landing area. At the end, the drone will search and verify the landing zone, then land right on the board. During this process, one of the most significant challenge is how to communicate with the ground control station efficiently and correctly in a real time matter. In this project, the focus is on improving current UAS communication system by modifying both the ground control station software tool and the UAV firmware source code.

The remainder of this report has the following organization. Section 2 discusses the background of this project. The detail design is presented in Section 3. Section 4 covers the current

project progress and result analysis. Section 5 discusses the possible future work and improvement. Section 6 concludes this report.

## **2. Background**

This section explains current related work including the communication protocol - MAVLink, the hardware used in this project - PIXHawk, and the telemetry radio – 3DR radio.

### **2.1 MAVLink Protocol**

In this project, the MAVLink protocol was used as the majority communication system between UAS and a ground control station. MAVLink is a very lightweight, header-only message marshalling library for micro air vehicles [2]. It can pack C-structs over serial channels with high efficiency and send these packets to the ground control station. It is extensively tested on the PX4, PIXHawk, APM and Parrot AR. Drone platforms and serves there as communication backbone for the MCU/IMU communication as well as for Linux interprocess and ground link communication.

<b>Field name</b>	<b>Index (Bytes)</b>	<b>Purpose</b>
Start-of-frame	0	Denotes the start of frame transmission (v1.0: 0xFE)
Pay-load-length	1	length of payload (n)
Packet sequence	2	Each component counts up his send sequence. Allows to detect packet loss
System ID	3	Identification of the SENDING system. Allows to differentiate different systems on the same network.
Component ID	4	Identification of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
Message ID	5	Identification of the message - the id defines what the payload “means” and how it should be correctly decoded.
Payload	6 to (n+5)	The data into the message, depends on the message id.
CRC	(n+6) to (n+7)	Check-sum of the entire packet, excluding the packet start sign (LSB to MSB)

Table 1. MAVLink v1.0 packet structure [2]

Table 1 shows the components of each MAVLink packet. What the software does is to check that it is a valid message (by checking the checksum and it is not corrupted, if so discards the message). That is one reason, why the baud rate for telemetry is set to 57,600 and not 115,200 bps, which will be introduced later. The lower it is the less errors it is prone to although slower it will be in updating to a Ground station. If you want to get a greater distance with MavLink, it may be a good idea to reduce further the baud rate. However, note that the tested baud rate 57,600 bps would give, in theory, around a mile of radius coverage with 3DR telemetry radio of -112dbm, according to the Signal to Noise Ratio (SNR).

From the above, what should be pay attention to is:

- a) System ID (source of message):** This is the source (i.e, Mission planner) sending a message to the PIXHawk via wireless Telemetry OR USB port. The software does a regular check so as to know that this message is for itself.
- b) Component ID (subsystem within the system):** Any subsystem within the main system. Currently, there are no subsystems as it is assumed that there is only one active vehicle in the system.
- c) Message ID:** What is this message about. This indicate the type of payload, which tells software how to decode this message.
- d) Payload (the actual data) –** This is the real message ground controller or UAV sends.

UAV gets streaming bytes from the air or other physical medium, gets it to the hardware interface, such as UART serial or telemetry and decodes the message in software. Note that the message contains the payload, which the software would extract. The payload from the packets described above are MAVLink messages. Every message is identifiable by the ID field on the packet, and the payload contains the data from the message. An XML document in the MAVlink source has the definition of the data stored in this payload. As MAVLink comes with a auto message generator

using a python code, it is relatively easy to generate a structured message. Since the current MAVLink is able to send sensor data in a fix frequency, it is a good example for us to generate self-defined message. This process will be discussed in Section 3.

## 2.2 PIXHawk and 3DR Radio

PIXHawk is an advanced autopilot system of 3DR. It features transparent for hardware and convenient for re-development. PIXHawk integrates with two advanced processor, STM32F103 backup failsafe 32-bit co-processor provides for manual recovery and has its own power supply if one processor breaks down, delivering incredible performance, flexibility, and reliability for controlling any autonomous vehicle. Users can also adjust the configurations of PIXHawk according to different use and hobbies for different vehicles. In recent years, as one of the main open-source autopilot systems, PIXHawk has gained a profound popularity among developers and hobbyists for being practical, easy to handle, economical.[4] It runs advanced 32 bit ARM Cortex® M4 Processor: NuttX RTOS (68 MHz / 252 MIPS Cortex-M4F) real-time operating system. The firmware is completely open sourced, which is a key reason it is chosen for this project.

3DR Radio is a small size, light weight, with 915 Hz variant and -121dbm receiver sensitivity. It supports transparent serial link, air data rates up to 250kbps and MAVLink protocol framing and status reporting.

## 3. Design Approach

This section introduce the design tool and the instruments to implement this project.

### 3.1 Desgin Tool

In this project, Linux (Ubuntu 14.04) is used as the operation system for laptop. Since the source code for both ground control station and ardupilot sontains more than 200,000 lines of code. It is a good idea to use a code editor/IDE to understand the structure of the source code. Eclipse was chosen for understanding ardupilot project due to its powerfull search funcion and well-organized

project structure feature. Besides, the ground control station, Qgroundcontrol in this project was developed with QT IDE. Same IDE was therefore used to modify ground control station software.

### 3.2 Build the firmware for PIXHawk on Linux with “make”

At the beginning stage of this project, it is necessary to make sure all the devices are working properly. The following steps are the instruments to build the firmware for PIXHawk on Linux to have a first look into this project.

- I. Clone the latest version of the “ardupilot” project from github. It is easier to use git command to do this: *git clone <https://github.com/diydrones/ardupilot.git>*
- II. Use terminal and change directory to the root where the ardupilot project is. From which, it is supposed to find a directory called “ardupilot”.
- III. Run the following command on terminal:

```
cd ardupilot/ArduCopter
```

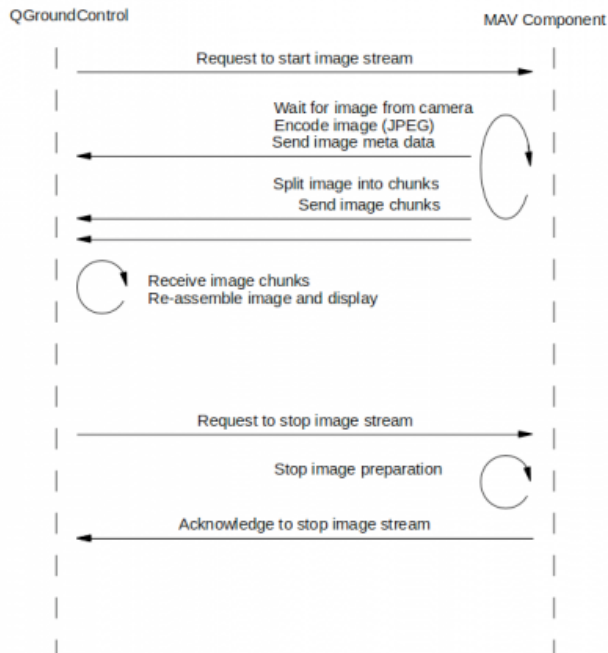
```
make px4-v2
```

- IV. Connect PIXHawk with computer through USB connector. If it is done correctly, there is a specific tone indicating the PIXHawk is successfully connected.
- V. Run the command:

```
make px4-v2-upload
```

It is supposed to see some compiling message from the terminal console after this. At the end, there is possibility to see a hint asking to unplug and re-plug in USB connector. This uploads the compiled file to the PIXHawk. As usual, there is a specific tone indicating the firmware is successfully uploaded.

### 3.3 Data transmission protocol



The complete image data protocol is showed in figure 1 [5]. The image streaming component uses two mavlink messages: A handshake message, `DATA_TRANSMISSION_HANDSHAKE`, to initiate, control and stop the image streaming; and a data container message, `ENCAPSULATED_IMAGE`, to transport the image data (see picture on the right).

(1) The communication is initiated by the QGroundControl with a request to start the stream. To do so, one must set the following fields in the MAVLink message:

target: to the ID of the targeted MAV,

state: to 0 for a request,

id: an ID for the image stream,

Note: For the moment, the image streamer only supports one stream per image type and therefore requires you to set the id to the same integer as the type field.

type: any of the types in the `ENUM MAVLINK_DATA_STREAM_TYPES` in `mavlink.h`,

freq: bigger than 0 for “frames per seconds”, lower than 0 for “seconds per frame”

It is possible to request for a specific image quality. To do so, you must set the quality field. All other fields should be zero in the initial request.

(2) When the targeted MAV receives the handshake request, it sends back an acknowledgment and starts the image stream at the requested framerate. The handshake ACK packet normally contains the same values as requested by the QGroundStation (state set to 1, because it's an ACK), and adds data about the size of the next sent image:

The field packets contains the number of MAVLink ENCAPSULATED\_DATA packets, the field payload specifies the size of the payload of each data packet (normally 252 bytes), and the size field specifies the image size in bytes.

(3) The image data is then split into chunks to fit into normal MAVLink messages. They are then packed into ENCAPSULATED\_DATA packets and sent over MAVLink. Every packet contains a sequence number as well as the ID of the image stream it belongs to. The image streamer now sends periodically new images, there is no further interaction needed. Every new image comes with a new DATA\_TRANSMISSION\_HANDSHAKE ACK packet with updated image size, packets and payload fields. After this ACK packet, the new image arrives as a series of ENCAPSULATED\_DATA packets. Note: The sequence number starts at 0 for every new image of the stream.

(4) To stop an image stream you must send a new DATA\_TRANSMISSION\_HANDSHAKE request packet with the frequency set to 0. The MAV will acknowledge this by sending back an ACK packet containing the same data as in the request.

### 3.4 Qgroundcontrol station architecture

QGroundControl is an object-oriented C++/Qt application allowing to represent and control micro air vehicles. QGroundControl adheres to the model-view-controller and ISO/OSI layer design



patterns [6]. This means that data, data manipulation and user interface representation are separated and that the access on hardware/communication links is abstracted from the application layer.

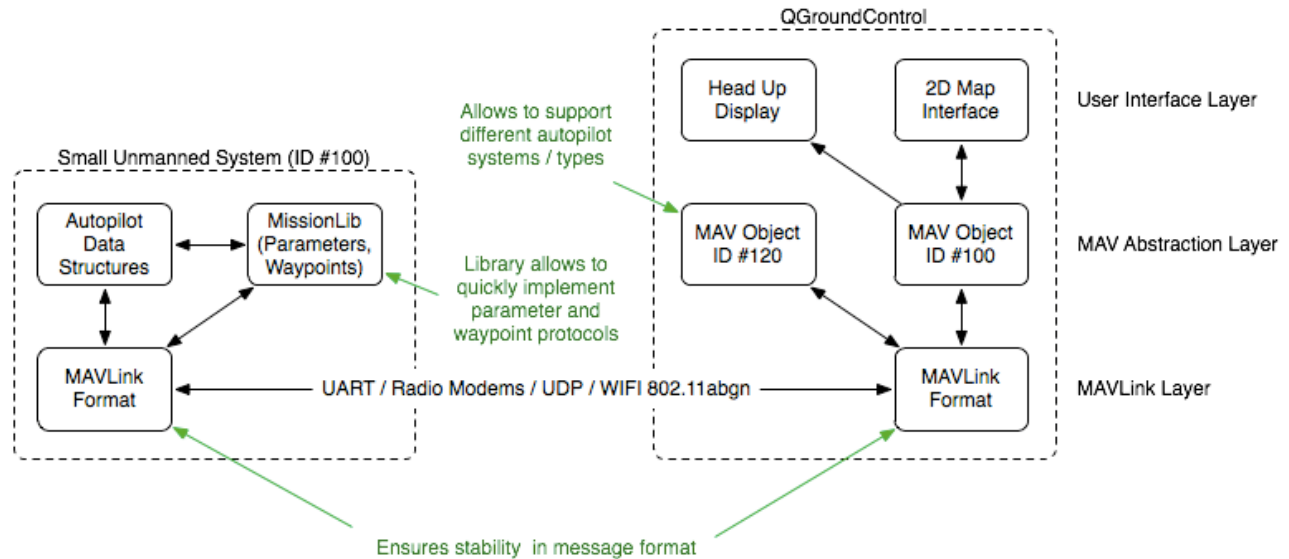


Figure 2. Qgroundcontrol Architecture

Figure 2 shows the architecture of this communication system. From the figure, it is obvious to see that the modification to the Qgroundcontrol tool is similar with the modification to the source code of UAV firmware.

### 3.5 Add new MAVLink message to the communication system

Data and commands are passed between the ground station (i.e Qgroundcontrol etc) using the MAVLink protocol over a serial interface. This part provides some high level advice for adding a new MAVLink message. In current progress of the project, only constant text message is considered. The following steps are the detail instruments to implement MAVLink message between ground control station and UAVs.

I. Decide what kind message is desired to be sent. In this case, `mavlink_msg_aloha` is used. Only one field is applied to this test message. The field is called `test`, the type of this field is an array of `char`, the length of this array is defined to be 50.

II. Modify the XML file under path

*ardupilot/libraries/GCS\_MAVLink/message\_definitions/common.xml*. The message id is assigned to be 144 in this test case. The following is the complete xml message:

```
<message id="144" name="ALOHA">  
    <description>Send a test message to ground control station.</description>  
    <field type="char[50]" name="test">test message sending to ground station  
</field>  
</message>
```

III. Regenerate the include files that will allow the new message to be recognised in the main code. First cd to the ardupilot directory and then run this command:

```
./libraries/GCS_MAVLink/generate.sh
```

If the generate completes successfully you should see that some of the following files have been updated.

```
../libraries/GCS_MAVLink/include/mavlink/v1.0/ardupilotmega/*aloha.h  
../libraries/GCS_MAVLink/include/mavlink/v1.0/ardupilotmega/version.h  
../libraries/GCS_MAVLink/include/mavlink/v1.0/common/version.h
```

The version.h files should simply have a date & time updated in the file but the one of the \*aloha.h files should have the new message defined.

IV. Navigate to the \*aloha.h file, there is several layers of function that implements packeting aloha message, encoding, decoding and sending buffer. The function called

*Mavlink\_msg\_aloha\_send(chan, test)* was used to implements this test.

V. Now change directory to ardupilot/Ardupilot/GCS\_Mavlink.cpp, add following code to add new member function to the class “Copter”. This is the function that would be called in the main function to send constant text message, which is hard coded in the array called “test[50]”.

```
NOINLINE void Copter::send_aloha(mavlink_channel_t chan)//send aloha test

{

    const char test[50]="ALOHA, GPS AUTO library test";

    mavlink_msg_aloha_send(chan, test);

}
```

VI. Add a notification to let system know the frequency, the running time and the new function should run. This is explained as following.

```
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
. . .
. . .
{ gcs_send_heartbeat, 100, 150 },
{ update_notify, 2, 100 },
{ one_hz_loop, 100, 420 },
{ gcs_check_input, 2, 550 },
{ gcs_send_heartbeat, 100, 150 },
{ gcs_send_deferred, 2, 720 },
{ gcs_data_stream_send, 2, 950 },
. . .
. . .
. . .
/*
 * send data streams in the given rate range on both links
 */
static void gcs_data_stream_send(void)
{
    gcs0.data_stream_send();
    if (gcs3.initialised) {
        gcs3.data_stream_send();
    }
}
```

Figure 3. Code Exmample 1

Figure 4. Code Example 2

Figure 3's code exmpale is where real-time systems concept plays a role. It is designed such that certain tasks take certain time, and if they are not finished by then, then they are not proceed. The first parameter is the function name, The second is the 'time it is supposed to take' (in 10ms units, i.e., 2 means, 20ms, i.e. 50Hz, i.e. this function runs 50 times a second). The third parameter is the 'max time beyond which the function should not run'. As a result,

*{send\_aloha, 1000, 1500}*

should be added it is designed to run 1 cicle per second, that it 1Hz. If it does not finish in 1.5 second, this function is ignored. What figue 4 shows is to send data down a link (gcs0 is via USB and gcs3 is via Telemetry). Going down further, this is used to send the data structures back to GCS for display. E.g. what happens when you move your copter with your hands and see Mission Planner's HUD screen? You see copter moving on the screen. We are getting the Attitude data (Pitch, Roll and Yaw)

every time unit. Likewise, we have IMU data, GPS data, Battery data, etc. In this project, we send GPS data to the ground control station after the “aloha” test was run.

VII. Compile and upload the modified project to PIXHawk using method that was described in Section 3.2.

VIII. Connect Ubuntu terminal console with the PIXHawk using

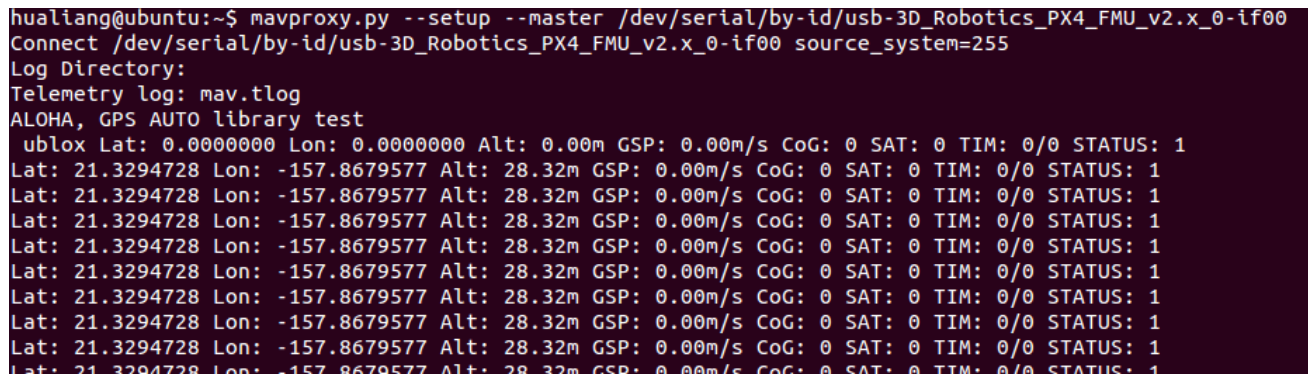
```
mavproxy.py --setup --master /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00
```

This will display the serial port output from the link between UAV and ground control station.

#### 4. Result Analysis and Future Work

This section explains the current progress of this project and how previous and concurrent course work is related to the project.

Figure 5 shows the result output of this project. On the ground control station, in this case, the console of Linux, it successfully print “ALOHA” test and starting to receive GPS sensor data which is sent from the active UAV.



```
hualiang@ubuntu:~$ mavproxy.py --setup --master /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00
Connect /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00 source_system=255
Log Directory:
Telemetry log: mav.tlog
ALOHA, GPS AUTO library test
ublox Lat: 0.0000000 Lon: 0.0000000 Alt: 0.00m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
Lat: 21.3294728 Lon: -157.8679577 Alt: 28.32m GSP: 0.00m/s CoG: 0 SAT: 0 TIM: 0/0 STATUS: 1
```

Figure 5. ALOHA Test Result

This means 1) the new created Mavlink message is successfully packeted by the software program for a constant test message. 2) The ground control station received the correct packet and decode successfully. In the real world situation, this implementation could be used as a basic sending/receiving example and can be further developed to send any kinds of data at any time. From

the economic aspect, this project takes about \$500 so far. It is all spent in hardware setup. All the software are from open source community and free to be used and modified.

At this point in its development, the UAV is in an unfinished state. The future improvement could be done by the following aspect. 1) Improve GUI ground control tool to manage packet decoding instead of using CML tool. 2) Develop an algorithm to manage sending large files between UAV and ground control station, such as an image. 3) Add user interface view for displaying or processing data sending from UAV.

## **5. Conclusion**

For the purposes of this project, the MAVLink protocol is well understood to be applied to this project. The original communication system was well studied as a base example to improve this communication system with more flexible communication method. Even though sending a constant text message will be unlikely used in a real world UAV, it could be the first step to develop a self-defined message communication system between ground control station and Unmanned Aerial Vehicle. As more and more civil applications involve UAV, this communication technology will be a great direction for us computer engineering students to continue with.

## Reference

1. CBSNews. Amazon unveils futuristic plan: Delivery by drone. [Online]. Available: <http://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/>
2. MAVLink Open Source Organization. MAVLink Dev Guide. [Online]. Available: <http://qgroundcontrol.org/mavlink/start>
3. MAVLink Source Code. Github Version control. [Online]. Available: <https://github.com/mavlink/mavlink>
4. PIXHawk inc. PIXHawk User Guide. [Online]. Available: <https://pixhawk.ethz.ch>
5. Image transmission protocol. [Online]. Available: [http://qgroundcontrol.org/mavlink/image\\_transmission\\_protocol](http://qgroundcontrol.org/mavlink/image_transmission_protocol)
6. Qgroundcontrol Developer's Guide. [Online]. Available: <http://qgroundcontrol.org/dev/start>