# MAVLAB

## Micro Air Vehicle Communication Toolbox

### 1 Introduction

MAVLAB is a toolbox which enables direct communication between MATLAB and any MAVLink enabled device. It is primarily designed for use with autonomous or semi-autonomous Micro Air Vehicles (MAVs) running ArduPilot firmware. The purpose of MAVLAB is to enable simple integration of MATLAB functionality with existing MAV platforms. This guide runs through the process of installing and using MAVLAB on a PC running Windows.

### 2 MAVLink Protocol

The MAVLink message marshalling library is used by a wide array of MAV autopilots as the primary communication method with ground control software. It is available in a number of different programming languages, and provides a reliable and efficient method of transferring data over Wi-Fi or telemetry radio. Data is transmitted in discrete message packets, the structure of which is shown in table 2.1.

***Table 2.1 –*** *Structure of a MAVLink 1.1 message packet.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte** | 0 | 1 | 2 | 3 | 4 | 5 | 6 to N+6 | N+7 to N+8 |
| **Content** | Start | Length | Sequence | Sys ID | Comp ID | Message | Payload | Checksum |

Each packet consists of the MAVLink message payload itself, and 8 bytes of header information. This header is used to determine the message origin, type, and packet integrity. A summary of the individual components of this header is outlined below:

***Start*** ***–*** Informs the receiving system that the start of a new packet has been reached.

***Length –*** Indicates the length of the payload component of the packet.

***Sequence –*** Increments by one for each packet sent, allowing packet drop to be detected.

***System ID –*** The ID of the system that sent this message (1-255).

***Component ID –*** The ID of the component that sent this message (0-255).

***Message ID –*** Defines which type of message format the following payload contains (0-255).

***Payload –*** The data content of the packet containing one MAVLink message.

***Checksum –*** An ITU X.25 checksum for checking the integrity of the packet.

In MAVLink version 1.X it is possible to have a maximum of 256 different message definitions. These are defined in a set of dialect XML files from which the MAVLink library is generated. Each message contains a number of fields which contain the data being transmitted. Table 2.2 shows the definition of message ID 143, which is used to transmit pressure and temperature information.

***Table 2.2 –*** The MAVLink message definition with ID 143.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| time\_boot\_ms | uint32\_t | Timestamp (milliseconds since system boot) |
| press\_abs | float | Absolute pressure (hectopascal) |
| press\_diff | float | Differential pressure 1 (hectopascal) |
| temperature | int16\_t | Temperature measurement (0.01 degrees celsius) |

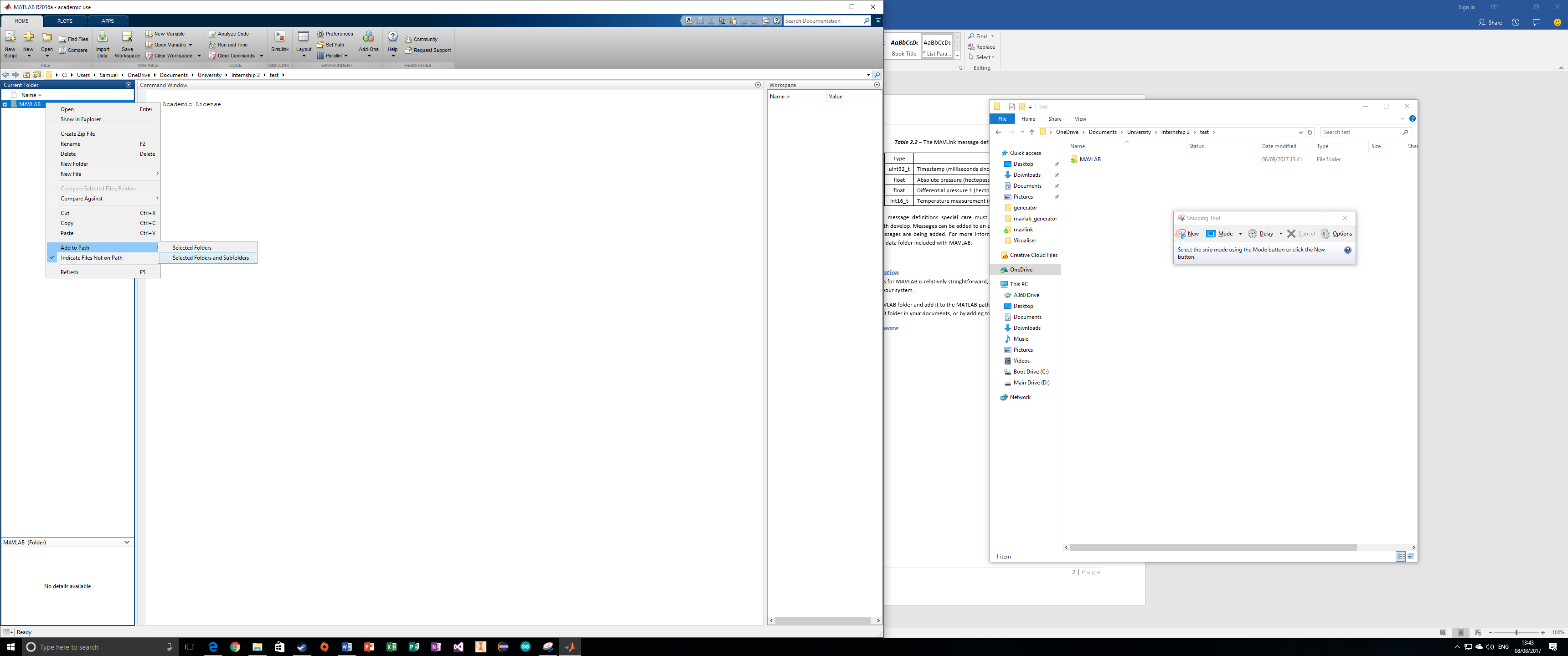
When making custom message definitions special care must be taken that an unused ID is chosen, as otherwise a conflict with develop. Messages can be added to an existing dialect file or in a dedicated new file if a large number of messages are being added. For more information on the XML dialect format, open the *common.xml* file in the data folder included with MAVLAB.

### 3 Installation

#### 3.1 MAVLAB Generation

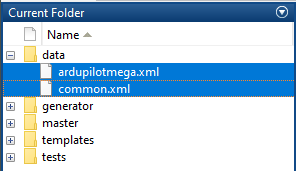
The installation process for MAVLAB is relatively straightforward, however, please ensure that MATLAB 2016 or later is installed on your system.

1. Unzip the *MAVLAB* folder and add it to the MATLAB path. This can be done either by placing it directly in the *MATLAB* folder in your documents, or by adding to path using the MATLAB file browser. Before continuing make sure to double click on the *MAVLAB* folder in MATLAB to ensure that the generator is able to find the necessary files correctly.



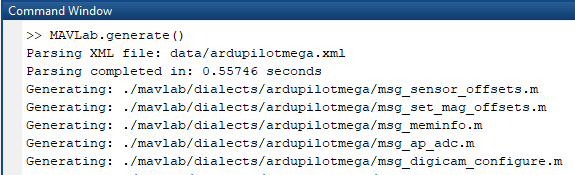
***Figure 3.1.1 –*** Adding MAVLAB to the MATLAB path.

1. Place one or more XML dialect files in the *data* folder. The dialect files should be the same ones used to generate MAVLINK on the MAV you are trying to connect to. These can usually be found in the GitHub repository for ArduPilot or your specific autopilot. Ensure that there are no conflicting message IDs between multiple dialect files, as any duplicate message IDs will cause MAVLAB to throw an error.



***Figure 3.1.2 –*** Add XML dialect files to the data folder.

1. Go to the command window and type in *MAVLab.generate()* in order to start the code generation process. This should only take a few seconds and will produce a new folder called *mavlab* containing the full toolbox for your dialect files.



***Figure 3.1.3 –*** Generating MAVLAB from the data folder.

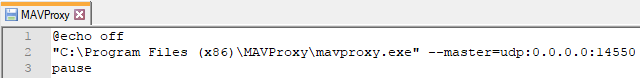
After completing the above 3 steps the full MAVLAB toolbox should be generated correctly. Should generation need to be repeated, such as in the event of changed message definitions, try to ensure that the *mavlab* folder is deleted first.

#### 3.2 Additional Software

Although it is possible to communicate directly with a MAV using the MAVLAB toolbox, it is recommended that two additional programs are installed. These programs are rigorously tested and are known to provide a safe and reliable channel of communications with MAVs.

MAVProxy is a MAVLink routing application which allows multiple ground control software packages to communicate with MAVs simultaneously.

1. Run the installer located in the *software* folder included with MAVLAB. This will install MAVProxy on your system. Make note of the installation path as this is needed in the following steps.
2. Create a file called *MAVProxy.bat* and copy the code shown in figure 3.2.1, replacing the path on the second line with the path to your MAVProxy installation. Double clicking this file will run MAVProxy and attempt to connect to a MAV communicating via UDP on port 14550. See the MAVProxy documentation for alternative communication methods such as telemetry radio via COM port.



***Figure 3.2.1 –*** Batch file for running MAVProxy and auto-connecting to MAVs on port 14550.

1. Append *--out=tcpin:0.0.0.0:14551* to the end of line 2 of the batch file you have just created. This will cause MAVProxy to route communications on the master port to port 14551. Programs such as ground control software and the MAVLAB toolbox can connect to this port in order to talk to any MAV connected to MAVProxy. Repeat this step with ports 14552+ in order to connect additional programs to MAVProxy.

After installation and creation of the batch file, MAVProxy can now be run by double clicking on *MAVProxy.bat*. Remember not to run MAVProxy directly as it will not launch with the correct settings.

Mission Planner is a ground control application which can be used to setup MAVs, as well as to monitor and control them during flight. It is highly recommended that this program as it can act as a safe backup link should problems arise with your MATLAB code.

1. As with MAVProxy, run the installed located in the *software* folder included with MAVLAB.
2. Run Mission Planner, and choose *TCP* from the drop-down menu in the top right. This will allow it to connect via MAVProxy.



***Figure 3.2.2 –*** Set Mission Planner to communicate using TCP.

After completing installation of both MAVProxy and Mission Planner your PC should now be able to communicate with MAVs using MAVLink over UDP. The last step is to ensure that your MAV is setup to communicate correctly with MAVProxy.

#### 3.3 MAV Setup and Testing

After following the steps in the previous section, MAVProxy is expecting to communicate with your MAV over a UDP connection via Wi-Fi. The process for setting up your MAV will vary depending on the autopilot hardware being used. The following steps are for setting up a Navio2 autopilot to communicate via UDP.

1. Type *sudo nano /etc/default/arducopter* into the Linux terminal of the Raspberry Pi hosting the Navio2 autopilot.
2. The top line of the file that opens should read TELEM1=”-A udp:127.0.0.1:14550”. This needs to be changed so that it points to the IP address of your PC. You can find your IP by typing *ipconfig* in the windows command line.

Your MAV should now be setup correctly to communicate with your Windows PC via UDP. Use the steps below to test the connection.

1. Run both MAVProxy and Mission Planner. Remember to run MAVProxy using the batch file created earlier.
2. Connect the battery to your MAV and wait approximately 60 seconds. This will give it time to connect to the wireless network.
3. Hit the connect button in Mission Planner. In the first dialog box that appears type *127.0.0.1* and hit OK. In the next box type the port number *14551* and hit OK. After a few seconds Mission Planner should connect to your MAV and start displaying telemetry data in the left panel.

Now that the communication pathway is correctly set up, you should now be able to connect MAVLAB to your MAV and begin working with the toolbox. If you encountered any problems in the previous steps you may be able to find useful troubleshooting information in the documentation for each piece of software used so far. You should also check that your autopilot is pointed to the correct IP address and that all of your ports are set correctly.

### 4 Understanding MAVLAB

#### 4.1 Establishing a Connection

Now that MAVLAB and its supporting software is installed, you can now begin using it to receive and send MAVLink message packets. The best way to do this is to connect it to MAVProxy using TCP and the *tcpclient* object. An example of how this is done can be found in *example1.m* in the *examples* folder that comes with MAVLAB. Once a TCP connection has been established with MAVProxy you can begin receiving and parsing messages.

#### 4.2 Receiving and Decoding MAVLink Messages

Once a connection has been established the incoming data stream needs to be parsed for MAVLink messages, which can be decoded into useful data using the *MAVLinkParser* class. Each received byte must be passed in turn to the *parseChar()* method, which will return any MAVLink packet that it successfully parses from the data stream. The packet itself can be unpacked into a specific message using the *unpack()* method. Once this is done the data can be accessed and used by any other MATLAB script or function. This is demonstrated in *example2.m* in the examples folder.

#### 4.3 Packing and Sending a MAVLink Message

As well as receiving messages, the MAVLAB toolbox can also be used to transmit messages to a MAV. In order to do this a message must be created and packed into a packet, before being encoded as a stream of data and transmitted using the *tcpclient* object. This can either be done manually in a number of steps, or by using the convenience *send()* method that is included in each message class. Both methods are outlined in *example3.m* in the *examples* folder.