Atmospheric Sensing UAV

By Iuliu-Cezar Ardelean

University of Manchester [MEng] Aerospace Engineering Student

[iuliu-cezar.ardelean@student.manchester.ac.uk](mailto:iuliu-cezar.ardelean@student.manchester.ac.uk), 0759 947 9057

August 2016

Contents

[1. Introduction 1](#_Toc460324453)

[2. Setup 2](#_Toc460324454)

[3. Learning 2](#_Toc460324455)

[3.1. Python 2](#_Toc460324456)

[3.2. Arduino 3](#_Toc460324457)

[3.3. MAVLink 4](#_Toc460324458)

[4. Existing hardware and software 5](#_Toc460324459)

[4.1. Hardware 6](#_Toc460324460)

[4.2. Software 6](#_Toc460324461)

[5. Issues 7](#_Toc460324462)

[5.1. CO Sensor 7](#_Toc460324463)

[5.2. Autopilot 8](#_Toc460324464)

[5.2.1. Python solution 8](#_Toc460324465)

[5.2.2. Arduino 8](#_Toc460324466)

[6. Working with data 10](#_Toc460324467)

[6.1. Collecting 10](#_Toc460324468)

[6.2. Processing 10](#_Toc460324469)

[7. Conclusion 11](#_Toc460324470)

[8. References 12](#_Toc460324471)

# Introduction

In this document the author will go through the steps required to program the components of an atmospheric sensing UAV. The components are a Teensy 3.1 microcontroller, an autopilot, and an Alphasense CO-B4 sensor. Previous programming experience is not required, although one should have an affinity for this sort of things. In the end, one should be able to gather and process the data in order to extract meaningful information from it.

# Setup

The first step is to acquire Ubuntu 16.04*,* as it provides a superior development environment (i.e. to Windows) in terms of programming with Python.

If one is natively running Windows, the most elegant solution is to install a Virtual Machine (VM), such as the Oracle VM VirtualBox(latest version available), which is freely available as an Open Source Software, under GNU GPL version 2 terms. [1]

There are numerous online resources available, should one encounter any difficulties. The tutorial followed and adapted by the author can be found at [*http://linus.nci.nih.gov/bdge/installUbuntu.html*](http://linus.nci.nih.gov/bdge/installUbuntu.html) *.*

A VM is recommended as a more viable solution compared to simply installing Ubuntu as a second OS, because should the VM break, one would only have to delete it and create another one.

|  |
| --- |
|  |
| Fig 1.1. – Showing the Ubuntu running VM |

# Learning

Basic knowledge of a programming language is valuable when considered the task presented. The author had previously studied C++, which considerably accelerated his learning process. However, it is possible for one to find courses for those who have little or no programming background.

## Python

The author was required to develop a number of skills and concepts in order to start working on the project. One is required to have a good understanding of Python 2.7 syntax and semantics (i.e. statements, expressions, methods, etc.) The tutorial followed by the author can be found at <https://learnpythonthehardway.org/book/>, and it claims to be available even to those with little or no experience.

Python has a considerable amount of libraries available which can vastly increase its functionality. One of these libraries is PyMAVLink, a library that can be used to communicate to an autopilot using the MAVLink protocol. [2]

Note: In order to fully take advantage of the functionality of Python, it is highly recommended that one also has a good comprehension of what Terminal/Command Shell can be used for.

|  |
| --- |
|  |
| Fig 2.1. –­­ *Learn Python the Hard Way* Table of Contents page |

## Arduino

During the course of the internship the team used the Teensy 3.1 micro-controller, which is supported by the Arduino IDE (<https://www.arduino.cc/en/Main/Software>) using Teensyduino (<https://www.pjrc.com/teensy/td_download.html>) [3], and the retired[12] Arduino Leonardo.

Basic knowledge of Arduino is required to be able to read and process the data from the sensors and the autopilot and send them to the ground control station via telemetry (in conformity with the MAVLink protocol).

The tutorial followed by the author can be found at <http://www.toptechboy.com/>, in the *Arduino Lessons* section. The author also recommends the tutorial called *Using Python with Arduino,* from the same webpage, as it will allow for more powerful data processing.

Note: <https://circuits.io/> provides users with the ability to build virtual circuits and practice/develop Arduino coding skills.

|  |
| --- |
|  |
| Fig 2.2. – Author’s project on <https://circuits.io/> , showing an Arduino Uno board, an ultrasonic distance sensor, a potentiometer and an LCD display. This setup can be used to display the position of any object on the LCD as read by the UDS. |

## MAVLink

MAVLink is a “language” understood both by the AP and the GCS. MAVLink libraries/modules can be used to integrate the AP and the GCS with the microcontroller, in order to send/receive messages (“packets”). [4]

The anatomy of a packet is as follows: start sign, payload length, sequence number, system ID, component ID, message ID, data, and checksum. [4]

* Examples of systems: vehicles, GCS, antenna trackers. [5]
* Examples of components: AP, camera. [5]
* Examples of messages: heartbeats, request data stream. [6]

By default, the GCS has and Vehicle has ; and are regarded as a broadcast ID; if the message does not have and , it is processed by the component and further sent to all known components. [5]

|  |
| --- |
|  |
| Fig 2.3. ­­­– The anatomy of a MAVLink packet [4] |

# Existing hardware and software

The project was already started when the team was presented to it. Much of the work done by the author consisted of methodically analysing others’ work. TeensyComSD.ino, the modified Mission Planner and related files belong to Bilal Kaddouh, from University of Manchester.

## Hardware

|  |
| --- |
|  |
| Fig 3.1. – Shows the complete hardware setup for the project. |

## Software

*TeensyComSD.ino* creates a MAVLink message (*UOM\_Atmospheric\_Data*) with the data from the AP and the CO sensor, then sends it to the GCS and saves it on an SD card.

The message flow schematics can be seen in figures *3.2. i)* and *3.2. ii).*

|  |
| --- |
|  |
| Fig 3.2. i) ­– Showing message flow from AP to GCS |
|  |
| Fig 3.2. ii) ­– Showing message flow from GCS to AP |

# Issues

## CO Sensor

The team was not initially able to receive proper data from the CO sensor, i.e. only received 0’s.

The data Teensy receives from the CO sensor requires some processing in order to get valuable information. During the course of this processing, an integer has to be divided by a larger integer and therefore, without a conversion to *float*, the result is null.

After introducing the conversion, the program runs as expected.

## Autopilot

When one runs the .ino file, the AP will only send heartbeats. This is because one has to send a data stream request message to the AP in order to receive all other parameters (e.g. attitude).

### Python solution

PyMAVLink is a Python library which allows one to communicate with an autopilot. The team wrote a program which receives GPS location and attitude information and logs it into a .txt file.

|  |
| --- |
|  |
| Fig 4.2.1. – The above Python script: 1. Waits for a heartbeat. 2. Requests a data stream. 3. Logs information in a .txt file. This is direct, one-to-one communication between an AP and a GCS. |

### Arduino

The situation above (Python) is ideal because the system consists only of an AP and a GCS. In reality the team also needed to include the microcontroller into the equation.

The procedure for requesting a data stream message looks simple and is as follows:

* Wait for a heartbeat from the AP
* Since one is making the Teensy send the message, one also has to create/pack the message.
* Send a request data stream message

Note: it was suggested that one also sends a heartbeat to the AP before sending the data stream request.

Information about how to do this is however difficult to find. The functions/methods necessary should be in the *Common* library of MAVLink, in the *mavlink\_msg\_request\_data\_stream.h* file.

Here is the author’s best attempt as well as explanation of choices.

|  |
| --- |
|  |
| Fig 5.2.2. a) – Part of *TeensyComSD.ino* file. The program waits for the first heartbeat from the AP, creates a heartbeat message and sends it to the AP, creates a request data stream message and sends it to the AP. |

|  |
| --- |
|  |
| Fig 5.2.2. b) – Part of *mavlink\_msg\_request\_data\_stream.h* file |

The author’s choices:

|  |  |
| --- | --- |
|  | Default for GCS [5] |
|  | Could not find which component of the GCS sends the data stream request. One can introduce a loop to test for all values between 0 and 255 |
|  | Newly created mavlink\_message\_t rdsmsg |
|  | Broadcast [5] |
|  | Broadcast [5] |
|  | All data streams [10] |
|  | Message rate in Hertz (Hz) |
|  | Start |

# Working with data

Python can be used for powerful data manipulation, such as live data plotting or animations.

The libraries/modules required for this are: *pySerial* (for accessing serial ports [7]), *NumPy* (enhanced mathematical and scientific computing features [8]), *matplotlib* (for creating plots, charts, etc. in a MATLAB similar interface [9]), and *drawnow* (for live plotting [13]).

## Collecting

Arduino can be programmed to send data through a serial port and Python has a very simple way of “catching” it. In order to open the serial port you will have to specify its name and the baud rate. The *read()* and *readline()* methods are then used to collect the data [11], after which you can further process it.

|  |
| --- |
|  |
| Fig 5. – Reading information through a serial port. In this example the baud rate is 57600 because it is reading data sent through a telemetry device. |

## Processing

After “catching” the data, one can manipulate it so as to be able to present it in a more human-friendly interface, such as graphs.

|  |
| --- |
|  |
| Fig 6.2. i) – The script is a simple demonstration of how to plot a live stream of the last 50 “batches” of data |

|  |  |
| --- | --- |
|  |  |
| ii) – After the author expels air towards the sensor | iii) – Minimal interaction with the sensor |
| Fig 6.2. ii) and iii) – Showing the output of the above script | |

# Conclusion

In the final sense, this project is about finding a way to gather and process data so as to be able to develop a better understanding of the environment. This document should serve as a guideline for future attempts at finishing the project, yet it cannot pretend to be exhaustive. The author found himself limited by the lack of understanding of the MAVLink protocol and the terrifying absence of information about it online.

# References

|  |  |
| --- | --- |
| [1] | *Oracle VM VirtualBox*, <https://www.virtualbox.org/>, last accessed 30/08/2016 |
| [2] | *pymavlink 2.0.6 : Python Packaging Index*, <https://pypi.python.org/pypi/pymavlink/2.0.6>, last accessed 30/08/2016 |
| [3] | *Teensy 3.2 & 3.1: New Features*, <https://www.pjrc.com/teensy/teensy31.html>, last accessed 30/08/2016 |
| [4] | *MAVLink Micro Air Vehicle Communication Protocol – QGroundControl GCS*, <http://qgroundcontrol.org/mavlink/start>, last accessed 30/08/2016 |
| [5] | *MAVLink Routing in ArduPilot – Dev Documentation*, <http://ardupilot.org/dev/docs/mavlink-routing-in-ardupilot.html>,last accessed 30/08/2016 |
| [6] | *MAVLink Tutorial for Absolute Dummies (Part –I) – DIY Drones,* <http://diydrones.com/group/arducopterusergroup/forum/topics/mavlink-tutorial-for-absolute-dummies-part-i>, last accessed 30/08/2016 |
| [7] | *pySerial – pySerial 3.0 documentation*, <http://pythonhosted.org/pyserial/pyserial.html#overview> , last accessed 30/08/2016 |
| [8] | *NumPy – Numpy*, <http://www.numpy.org/> , last accessed 30/08/2016 |
| [9] | *Matplotlib: python plotting –Matplotlib 1.5.1 documentation*, <http://matplotlib.org/> , last accessed 30/08/2016 |
| [10] | *MAVLINK Common Message set specifications*, <http://mavlink.org/messages/common> , last accessed 30/08/2016 |
| [11] | [*Short introduction – pySerial 3.0 documentation*, https://pythonhosted.org/pyserial/shortintro.html](https://pythonhosted.org/pyserial/shortintro.html) , last accessed 30/08/2016 |
| [12] | *Arduino – ArduinoBoardLeonardo*, <https://www.arduino.cc/en/Main/ArduinoBoardLeonardo> , last accessed 30/08/2016 |
| [13] | *Drawnow 0.44 : Python Package Index*, <https://pypi.python.org/pypi/drawnow/0.44> , last accessed 30/08/2016 |