AVDASI 2 Cube Guideline doc

The goal of this document is to lay out a minimum viable product for the avionics portion of the AVDASI 2 course.

1. Cube information

Basic information about the cube. Say what each in/out does and how to do the initial plug in setup.

The cube is a very expensive research grade flight computer so please don't blow it up.

https://docs.cubepilot.org/user-guides/autopilot/the-cube-user-manual

https://ardupilot.org/dev/docs/mavlink-move-servo.html

https://mavlink.io/en/messages/common.html

1.1. Requirements

Year on year the requirements may vary but the basic bits stay the same. Make sure the check the Design Specification and Deliverables documents for specifics.

The student-built avionics should be able to:

- Move the control surfaces
- Talk to the cube via Wifi
- Talk to the cube via Radio
- Log the generated data

- Sense flap angle
- Sense pressure
- Switch between cube modes and arming states

2. Mission Planner

Most useful bits of it

Mission planner is the software used to communicate and control the cube. Your code should replace Mission Planner and provide the same basic functionality. Mission planner is still useful to use for testing and has numerous vital features.

Installation info: https://ardupilot.org/planner/docs/mission-planner-installation.html

https://ardupilot.org/planner/docs/mission-planner-overview.html

2.1. Connecting

Mission Planner can be connected to the Cube over Wifi and USB. If connecting through USB, **make sure to use the buzzer**; do not plug the cable directly into the Cube. In mission planner, on the top right of the window, you will see two selection boxes and a ‘Connect’ button. If you know which COM terminal you plugged the USB into, select it, but there is an AUTO function too. Wait until the buzzer buzzes before clicking connect.

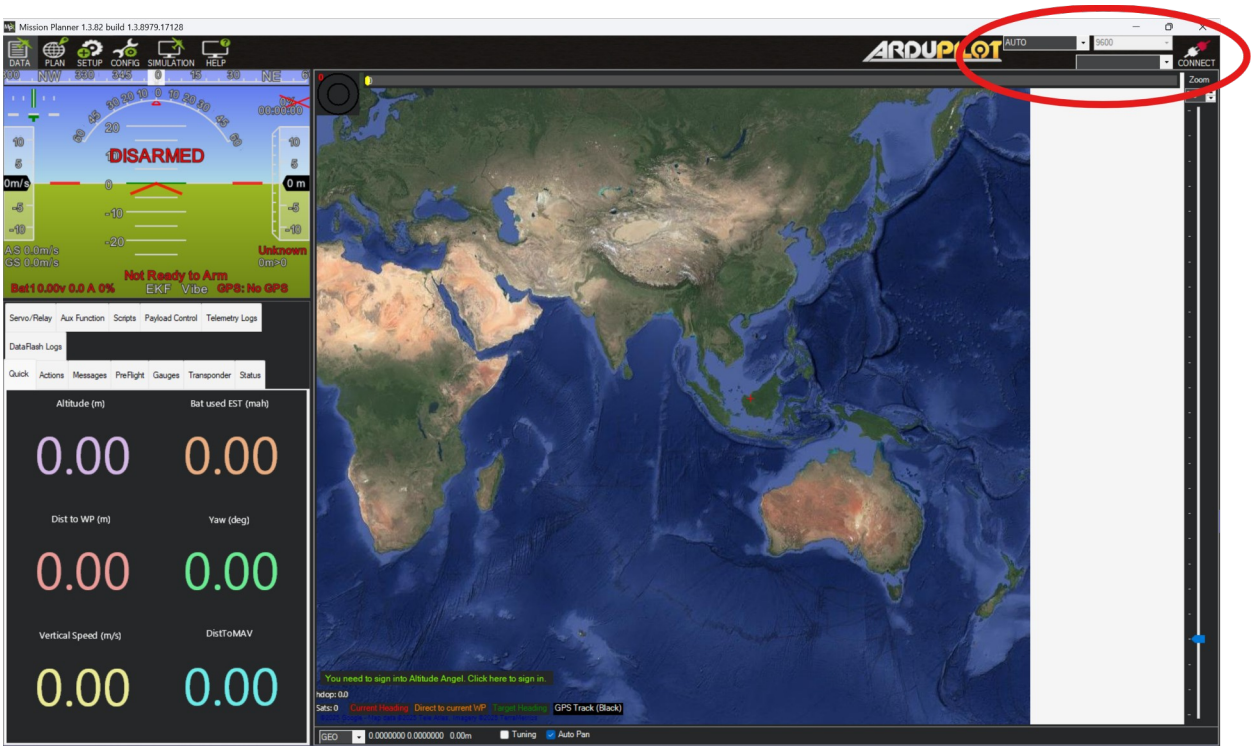

Figure 1 Connecting Mission Planner to the Cube

Using the Wifi module is a bit different but simple. After setting up the Wifi chip (see Part 4 of this document), connect the chip to the Cube's Telem1/Telem2 port and your computer to the Wifi network. Select the UDP setting and hit connect.

2.2. Parameters

When first connecting to the cube, there are a few parameters that need changing. In Mission Planner, a parameter is a configurable setting that tells the flight controller (like the Cube) how to behave. Parameters control everything from flight modes, sensor settings, failsafes, and PID tuning, to things like which servo does what and how fast a drone should climb or descend.

To access parameters, go to config and then full parameter list. This opens up every single parameters available for you to change. Scroll or search for NFT_BUZZ_VOLUME and turn it down so the cube doesn't scream at you every time you turn it on. Then click "write params" and congratulations you've changed the cube settings. Don't forget to save your parameters every so often so you don't lose them.

2.3. Servo/Actions

Ctrl F opens up some sort of settings thing. Click toggle servo safety to turn off safety. In "actions" there's the arm button.

3. Pymavlink

The cube uses MAVLink, a lightweight communication protocol to receive commands and transmit data to and from the Ground Station. Python has a MAVLink module called Pymavlink that will allow to send and capture MAVLink messages.

To be able to do so, first the code must connect to the Cube through MAVLink (see Part 2.1 for how to physically connect to the Cube). An example code will be provided to show a baseline for how to connect to the flight controller with python.

Another example code will show how to send commands over MAVLink (in particular, arming the Cube). Use this to develop your code.

Sending Commands using MAVLINK: Use the command_long_send function of the mavlink library to send control demands to the cube.

This command takes 7 parameters:

(target_system, target_component, command, confirmation, param1 - 7)

4. Wifi

How to set up the Wifi chip: https://ardupilot.org/copter/docs/common-esp8266-telemetry.html

(Note: you should not need to flash the chip, just set a name and password)

Wifi chip documentation: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah-esp8266-breakout.pdf

5. RC

Radio controller

6. I2C & LUA

Ardupilot specific LUA documentation:
https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_Scripting/docs/docs.lua

Ardupilot scripting support page: https://ardupilot.org/copter/docs/common-lua-scripts.html

I2C documentation: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

This part of the document will provide basic information on how to set up I2C from both the hardware (wiring) and software side (through LUA scripts; see example code for further help).

6.1. Wiring of the I2C bus

An I2C serial bus uses two wires (not counting VCC and neutral): SCL – clock & SDA – data. Being a multi-master/multi-slave, half-duplex protocol, multiple controllers and devices can be connected to the same bus, but only one can transmit at a time.

The wiring of the serial bus itself is straightforward: connect the SCL line to the SCL pins and the SDA line to the SDA pins of all devices on the bus, as seen in Figure 2. **It is very important, however, that both the SDA and SCL lines require pull-up resistors** connected to the logical high voltage of the selected microcontroller that will pull the line high when needed. There is a complex way of sizing the resistance of these resistors with high and low resistances both having pros and cons, but generally 4.7 kΩ is perfectly suitable for this kind of application.

If the Cube is directly used as the master, then there is no need for dedicated external pull-up resistors as there are built-in ones in the flight controller.

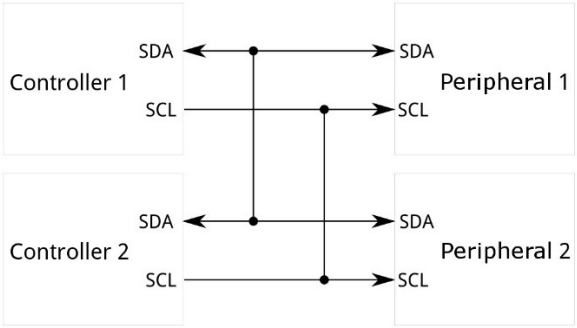

Figure 2 Mock-up of an I2C serial bus[1]

6.2. Enabling scripting on the CubePilot

Before LUA scripts can be executed onboard the flight controller, the scripting feature must be enabled using the parameters of the Cube. Using Mission Planner (or any other similar GS program), locate the SCR_ENABLE parameter and set it to 1 before rebooting the Cube. A new folder should appear in the APM folder named scripts; if it doesn't, it needs to be manually created.

After this is done, any LUA code uploaded into this folder will be automatically executed when the Cube powered on.

6.3. Setting up and Using I2C with LUA scripts

This part tries to explain the basic process of setting up and reading data from an I2C bus. To help understanding, example codes are provided.

[1] https://learn.sparkfun.com/tutorials/i2c/all

The first thing that must be done when working with I2C is to take note of the address of the device that is to be used. To set up a device on the I2C bus, LUA's built-in i2c: module is used. For example, the following line

local sensor = i2c:get_device(bus, ADDR)

will create a variable named sensor as an I2C device on bus *bus* (most likely will be 0 on the Cube) and with I2C address *ADDR*.

To read data from a specific register from a device the <device_var>:read_registers(REG, <int>) syntax can be used. With the above example:

sensor:read_registers(REG, 2)

will read two bytes of data, starting from sensor's register with address REG.

The data read from the registers will come in the shape of an array of bites and need to be broken up into separate binary integers. This is done differently depending on the device's endianness which must be found from the documentation. Figure 3 helps show the difference between little- and big-endian systems. This only changes the order of the bites transferred but must accounted for to get the data accurately.

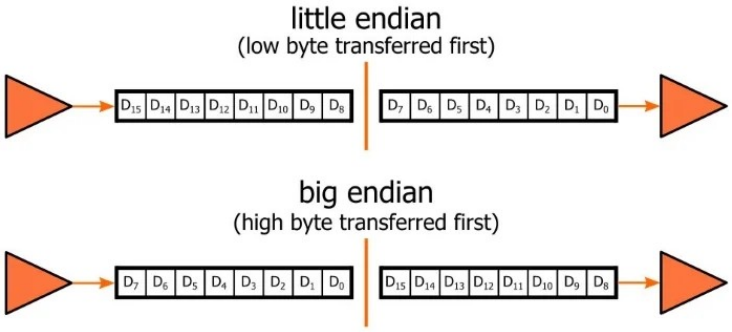

Figure 3 Endianness in a two-byte system[2]

The example code will showcase the basic bit operations that will help extract the data from the bit array the data was transmitted in in both big- and little-endian systems.

7. Data logging

7.1. Downloading Cube logs

The flight controller automatically creates a log of all its sensors (IMUs, temp. sensors, etc.) and its attitude changes during operation into a .bin file. These are necessary to analyse the behaviour

[2] https://www.allaboutcircuits.com/technical-articles/big-endian-little-endian-endianness-byte-arrangement-digital-systems/

of the drone in the wind tunnel and should be downloaded after each test so that the appropriate Teams of the Company can use their data.

7.2. (Optional: logging through LUA scripts)

LUA scripts can create and modify files on the Cube's memory; therefore, it is possible to create logs using LUA scripts (i.e. saving flap angles from the sensor into a CSV). This should only be done for a few sensors only, as it requires a lot of processing power from the flight controller. The example scripts will showcase this as well.

7.3. MatLab data analysis

8. Example code

A series of scripts were written to show a minimum viable ground station. Keep in mind your code will need to be better than this, you can't just copy it and hope for the best. It does provide a baseline to build off of and an example of how to meet the basic functionality needed.

Each script should be able to be executed individually but Main.py allows you to execute them all simultaneously. You don't need to take your approach in your own GS but this is good since you can debug each function independently.

8.1. Main.py

Links all the scripts together. Call each script as a function after defining them up top.

8.2. GS.py

This script is what connects the cube to your ground station. It creates the connection, waits for a heartbeat, and after heartbeat is received it listens and prints messages. There's a wide range of messages and statuses you can listen for from the cube, some more useful than others, so up to you to figure out which ones you want. It also notifies you if the heartbeat is interrupted.

8.3. Arm.py

This script allows you to arm/disarm the cube and toggle its safety switch. Arming allows logging and the safety switch locks all servo movement.

8.4. Servo.py

This script actuates the movement of a servo. You'll have to edit it to allow multiple servos moving simultaneously.

8.5. UI.py

This script is a barebones user interface for a GCS. It uses tkinter python module. This was used for its simplicity but you can use whatever module or language you want (there's some that can make really nice looking UIs). The UI calls functions from the previous arming and servo controller scripts and maps them to buttons. It also allows you to input a specific servo angle and move the servo accordingly.