# Bristol University Airborne Data Logger

Dr Paul Hill 19/08/2014

This document describes the hardware and software setup of the Bristol University Airborne data logger based on the Arducopter hardware and software. The basic firmware has been modified to initialise datalogging without any motors actually being connected and also to provide synchronised triggering of a Cannon 5D camera in video mode.

## Arducopter Firmware

The following source code files have been modified from the standard Ardupilot build.

* **UserCode.ino**
* **Motors.ino**
* **APM\_Config.h**
* **Log.ino**
* **ArudCopter.ino**

The trigger code has been integrated into a branch (3.1) of a fork of the project. Do not use the current version of the Ardupilot code as it is currently broken (for logs within the mission planner software).

<https://github.com/csprh/ardupilot/tree/logger>

## Configuration of the Arducopter Software

Any copter configuration will be OK (e.g. quad).

The following should be configured using the “mission planner” software (obtained from http://ardupilot.com/).

* Configure the compass
* Configure the AMU
* Add the Sonar (EZ0) to the list of optional hardware add-ons.
* Set the logger to produce the correct outputs
  + Set the bitmask LOG\_BITMASK to 24757

LOG\_BITMASK is calculated as follows

**#define MASK\_LOG\_ATTITUDE\_FAST (1<<0)**

**#define MASK\_LOG\_ATTITUDE\_MED (1<<1)**

**#define MASK\_LOG\_GPS (1<<2)**

**#define MASK\_LOG\_CTUN (1<<4)**

**#define MASK\_LOG\_NTUN (1<<5)**

**#define MASK\_LOG\_IMU (1<<7)**

**#define MASK\_LOG\_COMPASS (1<<13)**

**#define MASK\_LOG\_INAV (1<<14)** *// deprecated*

ORIG: 110000010110101 (in dec = 24757)

## Using the Data Logger

* Do not trigger the logger until the system has initialised (i.e. after the red and blue LEDs stop alternating in a fast flashing pattern). This is approximately 10 seconds.
* Press the logger button once.
  + This will “arm” the system and initialise the logging process (this is also characterised by the red and blue LED flashing pattern).
  + This will be indicated by the LED flashing twice at the beginning of the arming period and twice at the end. The camera will then be triggered just after that. Camera trigger will occur approximated 10 seconds after the button is pressed.
* The logging will start a small number (<4) of frames of before the 5D video is triggered.
  + The triggering of the 5D will also enter the first CAM message in the log.
  + You can therefore use this to approximately synchronise with the video.
  + Final synchronisation should be with the ATT data correlated to the video or the LED output.
* A test log of file size = 407kb lasted 48 seconds.
  + Therefore 16Mb of log is capable of logging over 30minutes of rate at this rate.

## Arducopter Logger Hardware

The trigger box is connected to the ardupilot as shown below.

The LED is actually connected to the beeper channel. This therefore flashes before and after the system is “armed”

<http://diydrones.com/profiles/blogs/adding-external-led-indicators-and-a-piezo-beeper-for-arm-and-gps>



## Analysis of the Datalogs

The data logs can be downloaded using the mission planner software after each flight. More detailed information on the content of the logs is shown below and described on the website here.

<http://copter.ardupilot.com/wiki/downloading-and-analyzing-data-logs-in-mission-planner>

A Matlab script generateFPArduLog.m is available that analyses input logs from the arducopter logger and create google earth first person fly through

struct log\_GPS pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_GPS\_MSG),

status : (uint8\_t)gps->status(),

gps\_week\_ms : gps->time\_week\_ms,

gps\_week : gps->time\_week,

num\_sats : gps->num\_sats,

hdop : gps->hdop,

latitude : gps->latitude,

longitude : gps->longitude,

rel\_altitude : relative\_alt,

altitude : gps->altitude\_cm,

ground\_speed : gps->ground\_speed\_cm,

ground\_course : gps->ground\_course\_cd,

vel\_z : gps->velocity\_down(),

apm\_time : hal.scheduler->millis()

}

struct log\_IMU pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_IMU\_MSG),

timestamp : hal.scheduler->millis(),

gyro\_x : gyro.x,

gyro\_y : gyro.y,

gyro\_z : gyro.z,

accel\_x : accel.x,

accel\_y : accel.y,

accel\_z : accel.z

};

struct log\_INAV pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_INAV\_MSG),

baro\_alt : (int16\_t)baro\_alt, // 1 barometer altitude

inav\_alt : (int16\_t)inertial\_nav.get\_altitude(), // 2 accel + baro filtered altitude

inav\_climb\_rate : (int16\_t)inertial\_nav.get\_velocity\_z(), // 3 accel + baro based climb rate

accel\_corr\_x : accel\_corr.x, // 4 accel correction x-axis

accel\_corr\_y : accel\_corr.y, // 5 accel correction y-axis

accel\_corr\_z : accel\_corr.z, // 6 accel correction z-axis

gps\_lat\_from\_home : g\_gps->latitude-home.lat, // 7 lat from home

gps\_lon\_from\_home : g\_gps->longitude-home.lng, // 8 lon from home

inav\_lat\_from\_home : inertial\_nav.get\_latitude\_diff(), // 9 accel based lat from home

inav\_lon\_from\_home : inertial\_nav.get\_longitude\_diff() // 10 accel based lon from home

};

struct log\_Attitude pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_ATTITUDE\_MSG),

roll\_in : (int16\_t)control\_roll,

roll : (int16\_t)ahrs.roll\_sensor,

pitch\_in : (int16\_t)control\_pitch,

pitch : (int16\_t)ahrs.pitch\_sensor,

yaw\_in : (int16\_t)g.rc\_4.control\_in,

yaw : (uint16\_t)ahrs.yaw\_sensor,

nav\_yaw : (uint16\_t)nav\_yaw

};

struct log\_Compass pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_COMPASS\_MSG),

mag\_x : compass.mag\_x,

mag\_y : compass.mag\_y,

mag\_z : compass.mag\_z,

offset\_x : (int16\_t)mag\_offsets.x,

offset\_y : (int16\_t)mag\_offsets.y,

offset\_z : (int16\_t)mag\_offsets.z,

motor\_offset\_x : (int16\_t)mag\_motor\_offsets.x,

motor\_offset\_y : (int16\_t)mag\_motor\_offsets.y,

motor\_offset\_z : (int16\_t)mag\_motor\_offsets.z

};

struct log\_Control\_Tuning pkt = {

LOG\_PACKET\_HEADER\_INIT(LOG\_CONTROL\_TUNING\_MSG),

throttle\_in : g.rc\_3.control\_in,

sonar\_alt : sonar\_alt,

baro\_alt : baro\_alt,

next\_wp\_alt : get\_target\_alt\_for\_reporting() / 100.0f,

desired\_sonar\_alt : (int16\_t)target\_sonar\_alt,

angle\_boost : angle\_boost,

climb\_rate : climb\_rate,

throttle\_out : g.rc\_3.servo\_out,

desired\_climb\_rate : desired\_climb\_rat

};