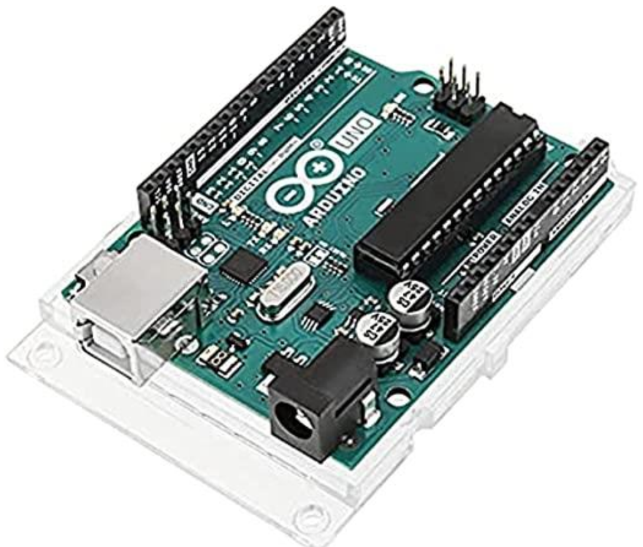
Playing with 'Arduino'

Arduino is a family of low-cost, miniature 'micro controllers' which provide digital and analogue I/O (input/output) and a basic 16MHz processor. They are aimed at education and hobbyists, and are very cheap, have a lot of accessories available, and a wealth of published example software to do all sorts of things. From an RC modeller's point of view, the most interesting features are:

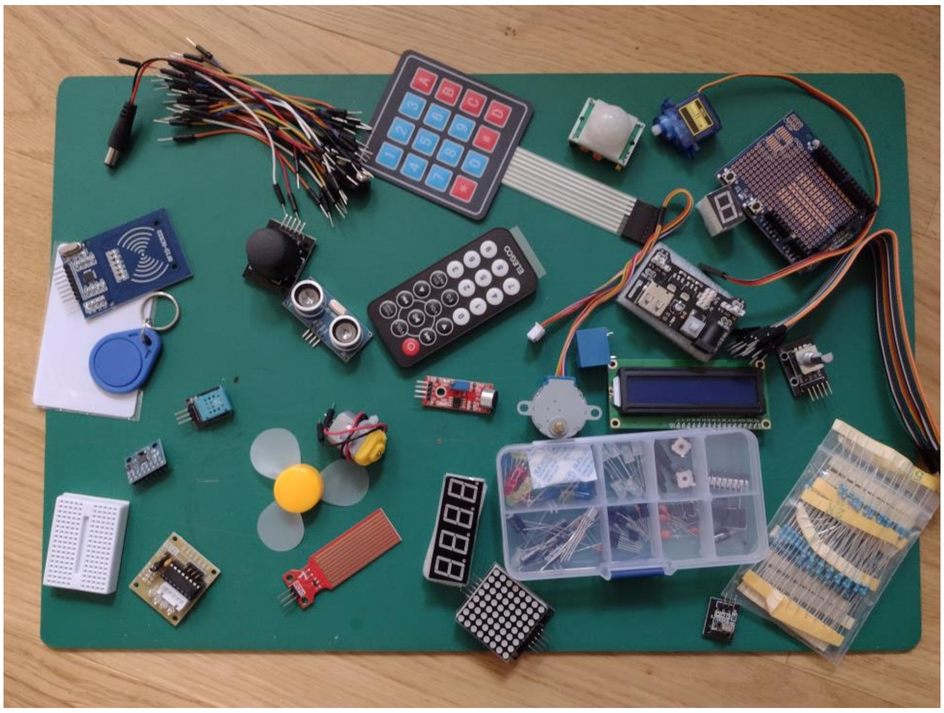
- Their small size, low cost and ability to run off battery power
- Their ability to read and write 'pulse width modulated' servo control data
- Their analogue inputs with analogue to digital converters which can be used to read all sorts of sensor data, plus measure battery voltages
- Their ability to store data on an SD card to provide data logging
- Availability of GPS and 6 axis 'MEMS' giro modules to read position, acceleration and attitude
- Availability of telemetry transceiver modules to transmit data between Arduinos at up to 1km in 2.4GHz band (about £4!)
- A video overlay generator which could be used to insert text and symbols into a video signal – you can use it to add 'Head Up Display' data to the image coming from a FPV camera.



Arduino Uno – about £3.50!

People have built complete auto-pilots, their own RC transmitters and receivers and all sorts of telemetry and data logging systems.

After a bit of hinting, I was fortunate enough to be given a 'starter kit' for Christmas which includes an Arduino 'Uno', a 'breadboard' kit for making little circuits around the board and about 20 accessories (switches, LEDs, a servo, ultrasonic module, keypads, displays, stepper motor, joystick etc etc).


A massive collection of goodies in a £50 starter kit

The starter kit came with a tutorial and a bunch of projects to practice on. Getting started was really easy – just install a USB driver on your PC and then plug the Arduino in via a USB cable, which also powers it. The development software (‘IDE’ or integrated development environment) runs on your PC and gives you an editor to fiddle with software which is then downloaded to the Arduino over the USB cable to run. Arduinos use a language called C++ which is a ‘proper’ engineering language, ideal for controlling hardware (unlike Python or BASIC). For example, the instruction to move a servo to the 90 degree position is:

```
myservo.write(90);
```

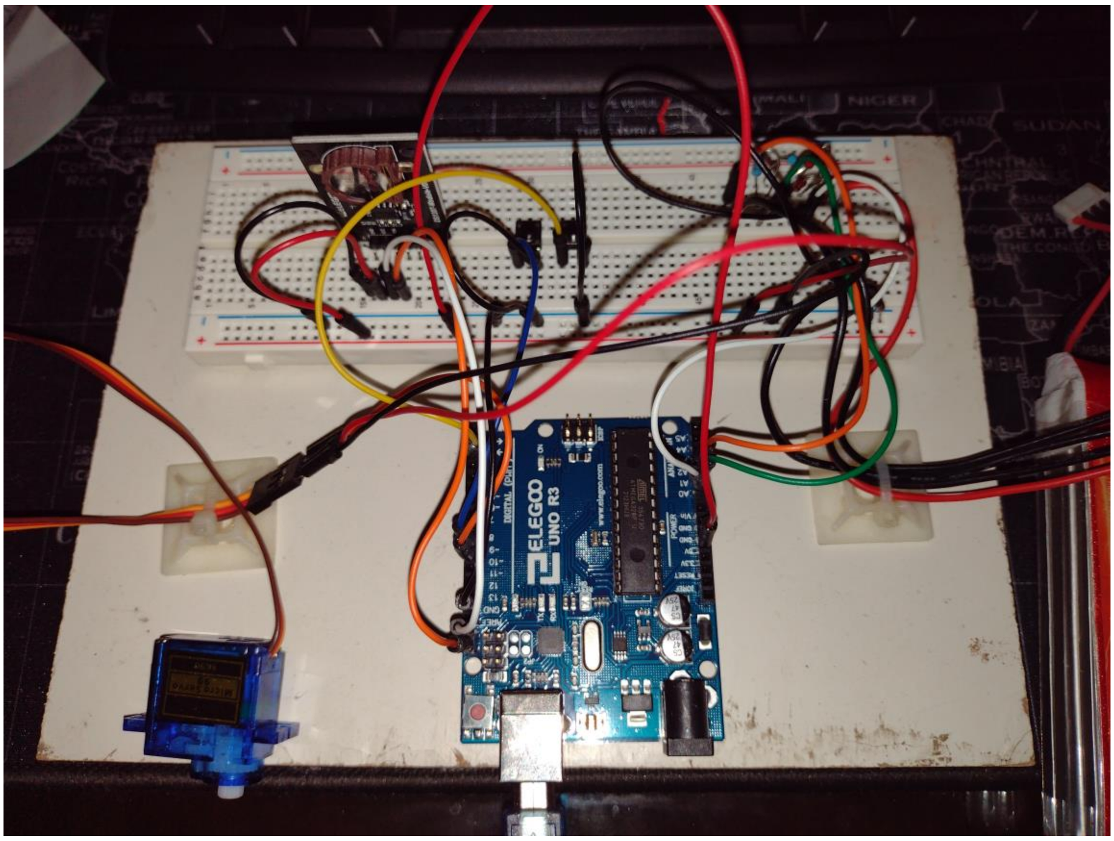
where you had previously defined ‘myservo’, to be of the type ‘servo’, and have specified what pin its input signal is connected to. Not exactly rocket science. C++ has all the control features you would expect to make a program: ‘if’, ‘while’, ‘for’ etc. Basically, a program is just a list of instructions that the processor executes in sequence.   C++ is modular, in that you can write a function separately, defining what is passed to it, what it does and what it returns. You can then ‘call’ that function whenever required elsewhere in the program.

I am fortunate that, as an engineer, I had done some programming in the ‘C’ language about 30 years ago, and as ‘C’ is the predecessor to ‘C++’, there was some familiarity.

The starter program, which comes up in the IDE when you first use it, is a program that makes an LED flash. You can then add something to read a button press to start or change the flashing, for example, and start building up from there. There are examples for almost everything on the arduino.cc website.

I wanted a ‘ground based’ project to start with, so I decided to build a ‘battery calibrator’ which logs the performance of a battery, using the motor and ESC in the model. The idea is that the Arduino would control the ESC via a ‘servo’ output, and run through a sequence of power ramps (going from 0-100% power) interspaced with periods running at 50% (which is about what most of my models cruise at). The cell voltages are captured by the built-in ADCs (Analogue to Digital Converters) on the board, which convert 0-5V to a number between 0 and 1024.   Only the bottom cell in the battery would be in the range 0-5V, so the voltages from the higher cells would be divided by 2 and 3 respectively using a resistor network. The data would be logged to an SD card and would record the cell voltages as the battery is run down.

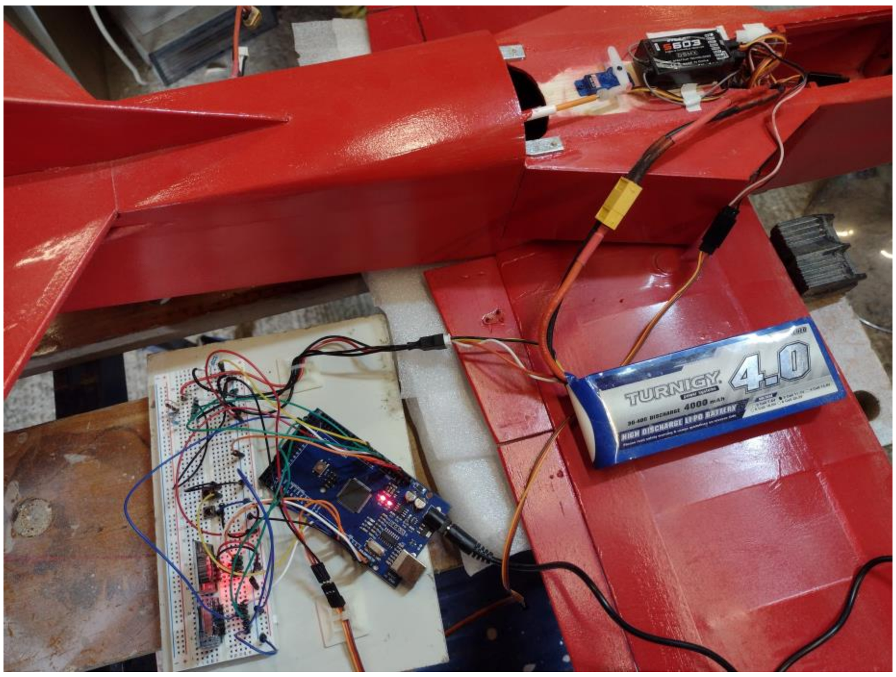
I wired 2 buttons (to start and stop the process) on my prototype and it quits when a cell voltage falls below a preset value (I set it at 3.2V). After a few happy evenings of hacking (using a servo rather than an ESC) I built up the software slowly, logging the data in a text format (.txt) and using ‘comma separated values’, so that the file could be read straight into a spreadsheet like Excel.



Using the reusable ‘breadboard’ is a bit untidy (lots of little link wires connecting things up) but if you want to make something permanent, the Arduino modules are designed to plug onto 2 rows of ‘header’ pins that you could solder onto a prototyping ‘Veroboard’. If you want to go whole hog, you can even design your own printed circuit on an online package and have 2 or 3 made for you in China for a few quid! There are also a number of prefabricated carrier boards to do various things, which are called ‘shields’ in Arduino world.

There are hundreds of little accessory modules that you can buy for a couple of quid. For my project, I got a real-time-clock module (which I only use to create a time-code filename for the results file) and an SD card holder module, which lets me write the results file onto an SD card so I can transfer it to my PC. I have also bought a 30 A current sensor (not yet tried out). These little modules are connected to power and a couple of wires to implement various types of serial channel (for which there are libraries for the IDE so you don't need to do any low-level programming yourself).

I plugged my rig into an ESC/motor and a battery. The servo cable that connects to the ESC needs to have its power wire removed, as the ESC will be providing power down this which may clash with the power supply on the Arduino. I powered up the Arduino and pushed the button I had set up as a start button, and, sure enough, the motor went through the sequence I had programmed, until after a few minutes it cut out.

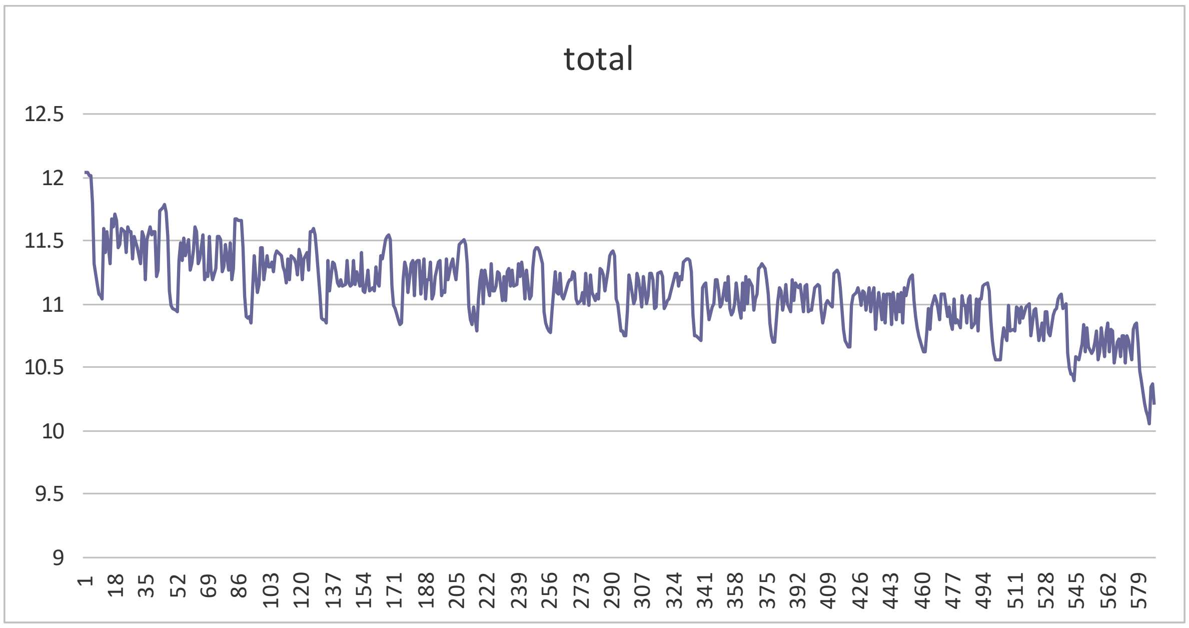


The first few lines of the file looks like this, with a title line and then data.

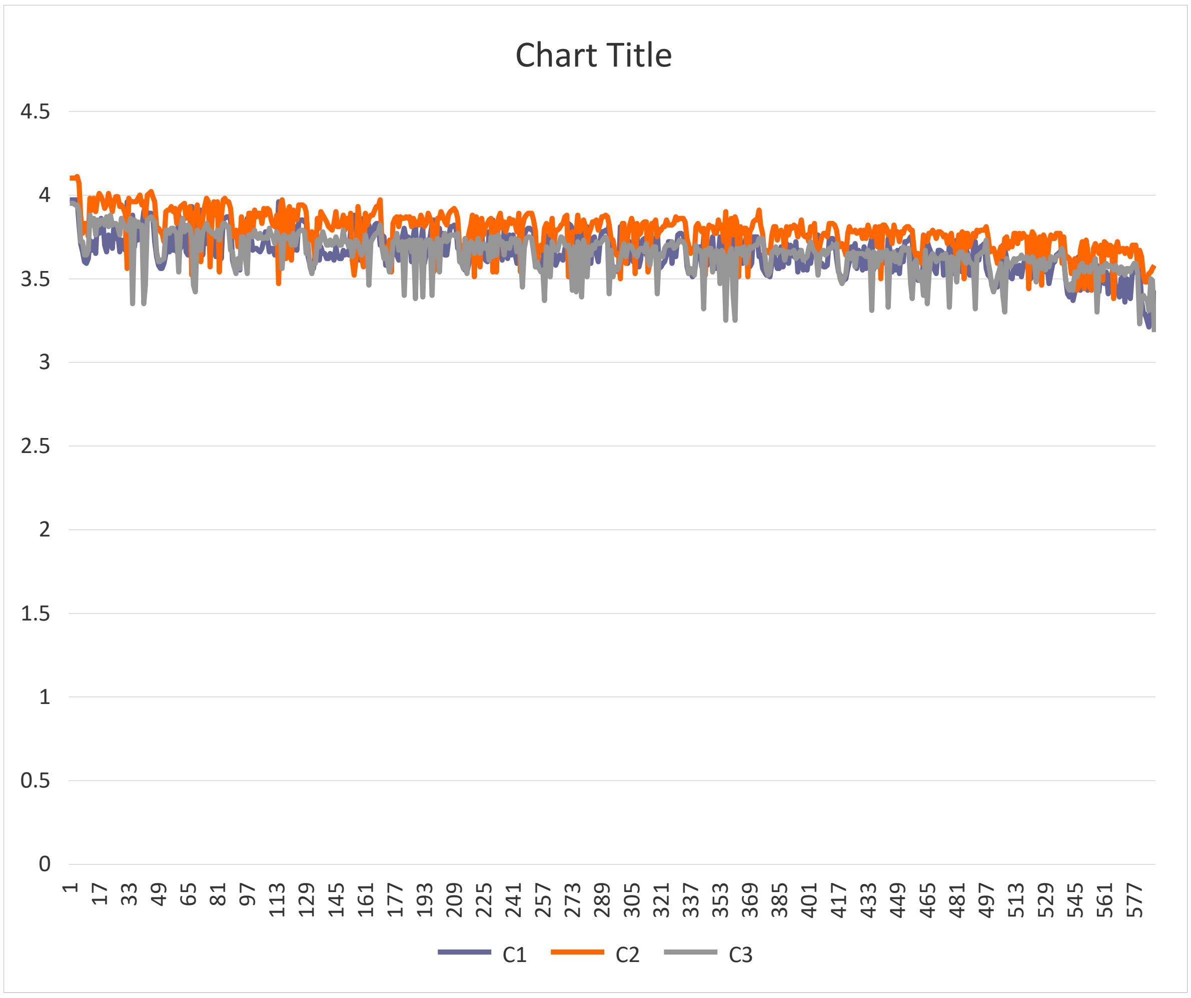
```
time, power, C1, C2, C3, total,
7104, 0.00, 3.97, 4.10, 3.95, 12.03,
8140, 10.00, 3.97, 4.10, 3.95, 12.03,
9177, 20.00, 3.97, 4.10, 3.95, 12.03,
10212, 30.00, 3.97, 4.10, 3.94, 12.01,
11248, 40.00, 3.97, 4.11, 3.94, 12.01,
12283, 50.00, 3.82, 4.07, 3.90, 11.79,
13320, 60.00, 3.71, 3.88, 3.72, 11.31,
14354, 70.00, 3.66, 3.77, 3.72, 11.15,
15388, 80.00, 3.60, 3.83, 3.64, 11.07,
16421, 90.00, 3.59, 3.83, 3.64, 11.06,
```

'Time', in this case, means time in milliseconds since the program started.

Obviously, you choose what data to log in your software. I just chose to log: a timestamp, the ESC power setting, the individual cell voltages and the total voltage. I wrote the program to log data approximately once per second. You can see that C1 (voltage of cell one) drops from 3.97 to 3.59 as the power ramps up. You can also see that the 3 cells are out of balance (it was a part charged battery). This format of data (values separated by commas) can be read into Excel, and turned into a nice chart like this:



This information is useful – it gives you an idea of how the battery voltage decays as it discharges, an indication of how long a battery will last in typical use. The little blimps in the chart are where the power is cycled from 0-100% and gives you an idea of how the battery responds to a changing load.



The chart of the individual cell voltages indicates the condition of the individual cells. I think from this that you can see that C3 is the weak cell in the battery.

What next? There is a version of Arduino called 'Nano' which is tiny (45 x 18mm!) and has a version with a built in 9 axis giro, and Bluetooth, so potentially you could use your smart phone to interact with it. If you paired that with a gps module and an SD card reader, you could easily build a flight logger (or a start towards an auto pilot!). I think I might try something to do with airspeed detection and maybe try some telemetry. Or use the ultrasonic module in my kit to detect distance to ground with an overlay generator to aid FPV landings. Endless fun to be had!

Mark Saunders
January 2022