AO13 Aerosol sampling

April 29, 2008

Abstract

The vertical distribution of aerosol properties in the atmosphere, for example- salt crystals, sand, ablated soils, and aviation derived black carbon soot in the upper troposphere and lower stratosphere is very poorly quantified. Yet they have a significant impact on radiative transfer in an aerosol laden atmosphere. There has been relatively little research involving direct sampling of aerosol particles at altitude, and published studies have involved aircraft mounted samplers.

A cheap autonomous sonde mounted sampler was designed and constructed together with functional sonde sub-systems, to enable sampling to be conducted over the UK at altitudes up to 30km, with multiple samples taken in different altitude ranges. The sampling device was designed to enable examination with a scanning electron microscope to determine composition, morphology, and particle concentrations to be inferred.

1 Introduction

Aerosol research is a very active field, with particular bearing on many aspects of atmospheric physics. However, the majority of non ground based studies have involved inferring measurements from satellite, not direct measurements or sampling. Blake and Kato [1], and Pusechel at al [3], employed NASA ER-2 and DC-8 research aircraft mounted sampling devices. These aircraft have altitude ceilings of 21Km and 12km respectively, lower than sonde flights, which can reach 30km with ease. In Europe there is also the CARIBIC program [4], employing an Airbus A340 and Boeing 767 300ER in published research programs. Physical sampling for later laboratory analysis allows chemical composition to be determined, along with particle morphology [1].

Meanwhile, the falling price of electronic hardware including GPS receivers and communication equipment has sparked interest in the development of low-cost sounding units to supplement the traditional radiosonde network. A sonde based system is considerably cheaper and more flexible than the aforementioned systems, can sample a greater altitude range, and is not limited by air traffic control regulations to the same extent. This project was inspired by such novel alternative sounding methods, which have recently been pioneered in the UK by the UK high altitude society [12], and Cambridge university spaceflight [13].

Stratospheric sampling conditions: from Blake and Kato [1] number densities for black carbon soot particles are in the $10^5 m^{-3}$ range. These typically consist of a chain structure of globules of around 50nm diameter. The morphology of the particles and their evolution over time are of particular interest, as is the role of aviation in their production. Sulphuric acid particles are also found.

Tropospheric sampling conditions: tropospheric particles are more diverse, primary aerosols of ablated soil particles, sand, and sea salt aerosol are found. Secondary aerosols such as ammonium nitrate are directly formed from the gas phase. Number densities are over two orders of magnitude higher than for stratospheric aerosol.

Aviation is thought to be the largest contributor to black carbon in the stratosphere[1]. During the past few decades, air traffic has increased at a high rate, (see UK aviation CO_2 forecasts 2005-2050 figure 1.3 [10][1]). This shows that since the study by Blake et al in 1995, UK CO_2 emissions have increased by 50%. This would imply a similar increase in black carbon emission. However this may have been confounded to some degree by a cleaner aircraft fleet. This project aims to enable a direct measurement of the current level of black carbon in the Stratosphere and upper Troposphere.

2 Methods

2.1 Aims

A set of aims were drawn up as follows;

1. Demonstrate and design an aerosol sampling device capable of sampling aerosol from the troposphere to stratosphere.
2. Obtain samples of aviation aerosol soot (black carbon) emission.

[1] This can be found in the appendix- figure 30

2.1.1 Science Requirements

From a consideration of the science aims, the list of requirements were drawn up;

1. The collection efficiency as a function of particle size should be well quantified.
2. The sampler should collect a statistically significant sample.
3. There should be statistically insignificant sample contamination.
4. Non-volatile particles should be captured and stored without changes to their chemistry.
5. If multiple samples are taken (e.g. differing altitude ranges) then cross contamination should be insignificant or at least well quantified.

2.2 Hardware design considerations

A typical sonde flight will ascend to reach an apogee altitude of 20 to 30Km above mean sea level around 2 hours after launch. The latex envelope then bursts, leaving the payload to descend by parachute. Ground air temperature in the UK is usually in the 10 to 20°C range, and tropopause temperatures are typically around -50°C, so the payload experiences large temperature changes. The lower temperature limit lies outside the standard industrial temperature range used for electronics, which extends to −40°C. During descent into the troposphere, condensation is a significant risk due to the sonde payload being below the dew-point. The typical solution to these difficulties, adopted by members of UKHAS and others is a 25 to 50mm thick airtight[2] enclosure fabricated from extruded polystyrene or expanded polypropylene. This serves to protect sensitive components on landing, as well as allowing the electronics to heat the inside to a safer temperature.

The avionics are critical to safely recovering the payload after landing, and again UKHAS[12] and CUSF[13] provide useful examples of how to design a successful system. Typically a central computer board based around a microcontroller or an off the shelf single board Linux machine is used. The other electronic hardware is then interfaced with the central "motherboard". A minimal architecture consists of GPS receiver, mobile phone/GSM module, radio datalink, and a release mechanism to cut-down[3].

[2] obviously an airtight closed cell foam enclosure would suffer structural failure due to internal overpressure during ascent, but assembly has been found to never be good enough to be completely airtight, although sufficient to prevent condensation.

[3] "cut-down" is a radiosonde term meaning to trigger an early descent by releasing the balloon from the payload, for example to avoid landing in water.

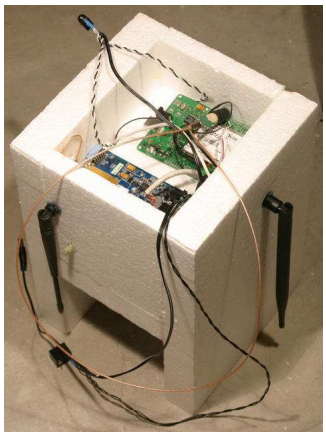

Figure 1: A typical sonde based on an embedded Linux system.

Such a system typically requires in the order of 1 to 2 Watts of electrical power, and it has been found that lithium camera batteries, such as sold by energizer[14] offer the best performance, the data-sheet listing the absolute minimum operating temperature as -40°C.

The radio datalink is perhaps one of the more difficult components, in the UK OFCOM limits its transmissions from high altitude balloons to 10mw on the 434MHz band, using a commercial transmitter module of a licenced design. It is possible to transmit radio-teletype at 300 baud on this band for several hundred kilometres using a pulse shaping transmission scheme to modulate an FM transmitter module. Members of the UK high altitude society have previously designed a system, but the source code has not been released, and there is no printed circuit board design for the project. With this in mind, in 2007 the author designed a radio module using an 8 bit AVR micro-controller, which was employed for the first time on this sonde.

A final consideration is mass, the civil aviation authority does not explicitly set mass limits on radiosondes, but bearing in mind health and safety, and insurance for the project, a mass limit of 2Kg was decided upon.

2.3 Instrument design process

The aerosol sampler designs considered were all drawn from Hinds [2], and evaluated on their suitability based on the science aims and hardware considerations, along with our budget limit of approximately £1000 for the entire system.

From Blake and Kato[1], number densities of stratospheric particles are in the $10^4 m^{-3}$ to $10^5 m^{-3}$ range. About 2×10^2 particles are required to provide statistically significant data for particle morphology and chemistry studies. If our instrument has a collection efficiency in the 10% range and we require 2×10^2 particles, an air sample of up to $0.2m^3$ is required. If multiple samples are to be taken during the ascent,

a reasonable sampling time might be around 40 minutes, or 2×10^3 seconds. This gives a flow rate of between 10^{-4} and $10^{-5}\text{m}^3\text{s}^{-1}$ through the instrument.

In all the sampling techniques considered, particles are deposited onto some sampling surface for later analysis. For analysis of with a scanning electron microscope, a minimum number density on the sampling surface of 1 particle per $100\mu\text{m}^2$ seems reasonable[4]. For a sample of 10^2 to 10^3 particles, this corresponds to a sampling surface area 0.33mm^2 at most.

It is essential to ensure that the flow through the instrument is laminar to prevent aerosol deposition on the instrument walls. Assuming that the airflow cross-section is larger than the sampling surface area (valid in all approaches accept 4), then with a cross section of 1mm^2 for our airstream through the instrument, an airspeed of at least 10ms^{-1} is required (at $10^{-5}\text{m}^3\text{s}^{-1}$). The Reynolds number is given by

$$\frac{\rho v_s L}{\mu} = \frac{v_s L}{\nu} \tag{1}$$

This will be highest at ground level, where density is greatest. For STP air, $\mu = 1.9 \times 10^{-5}$, and $\rho = 1.2\text{Kgm}^{-3}$, so the Reynolds number is given by

$$\frac{1.2 v_s L}{1.9 \times 10^{-5}} = 3.7 \times 10^4 v_s L \tag{2}$$

Within circular pipes the critical Reynolds number is generally accepted to be 2300, where the Reynolds number is based on the pipe diameter and the mean velocity v_s within the pipe. Assuming the lower bound on volumetric sampling rate, i.e. 10^{-2}Litre s^{-1} then if our instrument consists of some circular pipe.

$$\frac{\pi L^2 v_s}{4} = 10^{-5} \longrightarrow v_s = \frac{4 \times 10^{-5}}{\pi L^2} \tag{3}$$

So, if $Re < 2000$ then.

$$v_s L = \frac{4 \times 10^{-5}}{\pi L} < 5.5 \times 10^{-2} \longrightarrow L \gtrsim 2.5 \times 10^{-4} \tag{4}$$

In the upper troposphere and stratosphere, the Reynolds number will be one or two orders of magnitude lower, so the condition on L will decrease greatly. It will clearly be easy to maintain laminar flow in an airstream of 1mm diameter or more, especially considering that lower flow rates will be required in the troposphere due to the higher volumetric number densities of particulates.

Achieving a well characterised sampling efficiency and a high particle density on the sampling surface is non trivial. Four approaches employing the different physical forces illustrated in figure 2 were considered; thermal and electrostatic precipitators, impactors, and filters.

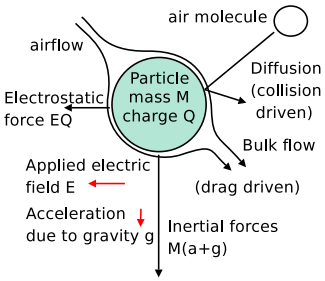

Figure 2: A look at the different effects influencing particle motion in the μm regime.

Laminar flow is especially important for the first three methods, as existing studies have been in the laminar flow region, and collection efficiencies in the case of electrostatic and thermal precipitators especially drops off as we enter the turbulent flow regime.

In the first three approaches the particle's Reynolds numbers is also important, as it moves across the airstream to the deposition surface in a roughly diagonal path. The required particle velocity is thus around the same as the airflow velocity, or 10ms^{-1}. The viscosity and pressure are functions of altitude, and the diameter of black carbon soot is in the 200nm to 2μm range. Stokes's resistance law is valid below a Reynolds number of approximately 1 (Hinds figure 3.1 [2]), so it would be insightful to find the highest Reynolds number in our operating regime. The viscosity decreases with decreasing temperature at altitude via Sutherland's equation (13), but the air density decrease is a far larger effect, so the largest Reynolds number is at ground level for the largest particles. Taking a cross airstream velocity of 10ms^{-1}.

$$Re = \frac{1.2 kgm^{-3} \times 10 ms^{-1} \times 2 \times 10^{-6}}{1.9 \times 10^{-5}} = 1.3 \tag{5}$$

So we are operating in the Stokes regime, and the drag force is;

$$F_{stokes} = 3\pi\mu V d \tag{6}$$

where V is the particle velocity and d the particle diameter. If the mean free path is comparable with particle size, the Cunningham correction factor C_c should be considered.

$$C_c = 1 + \frac{\lambda}{d}[2.34 + 1.05e^{-0.39\frac{d}{\lambda}}] \tag{7}$$

[4] simply by considering the difficulties inherent in using a scanning electron microscope to examine a sparsely populated sampling surface

At STP, the mean free path is given by;

$$\lambda = \frac{1}{\sqrt{2}n\pi d_m^2} = \frac{1.6 \times 10^{18}}{n} = 66nm \quad (8)$$

Where n is molecular number density and d_m molecular collision diameter. This scales with $\frac{1}{n}$. At the top of our altitude range in the middle stratosphere air density is 1% of its surface value, so the mean free path will be 6.6μm. Thus we must divide the Stokes drag by C_c to obtain $F_d = \frac{F_{stokes}}{C_c}$. Over our $\frac{\lambda}{d}$ range of approximately 0 to 3, Cunningham's factor can be approximated to $C_c = [1 + 3\frac{\lambda}{d}] \pm 10\%$.

Thermal precipitator

This technique is based on thermophoresis. The Maxwell-Boltzmann distribution leads to the following momenta distribution;

$$f_p = \sqrt{\left(\frac{1}{2\pi mkT}\right)^3} \exp\left(\frac{-p^2}{2mkT}\right) \quad (9)$$

Giving a mean momentum for a molecule of

$$p_{mean} = \sqrt{2kTm} = \sqrt{2RTM} \quad (10)$$

where M is the molar mass of air. Momentum increases with $\sqrt{T}$. Thus air molecules colliding from the direction of increasing temperature will on average carry greater momenta, and a stationary particle will acquire a net momentum pointing down the gradient.

As this system relies on a temperature gradient, it would require a heater, probably a resistance wire. Power limitations make this hard to achieve. Achieving a high particle density per unit area of the deposition surface is difficult; a small heated area separated from the deposition target by an air gap would require a high airflow velocity between the two plates, limiting the collection efficiency. If the airflow becomes non laminar, collection efficiency will drop off greatly, and an accurate efficiency calculation would require careful calibration.

The change in system performance with altitude will be complex; the thermal conductivity of air is invariant with pressure to a first approximation, however volumetric heat capacity is not, making a constant temperature gradient hard to maintain. The thermophoresis effect will scale with the rate of collisions, and thus with air density. This linear change in thermophoesis force with air density will be counteracted by a linear change in drag force, so air density should have little effect on collection efficiency at small sizes. However, the mean free path changes by two orders of magnitude across our altitude range (equation 8) and thermophoresis will not be significant for particles much larger that the mean free path. Thus across our altitude range thermophoresis goes from being a small effect even for particles at the bottom of our size range to a significant effect across the particle size range. Hence the collection efficiency will vary greatly with altitude and particle size, making objective measurements difficult.

Electrostatic precipitator

The operating principle uses a corona discharge to charge particles, then an electric field to draw them out of the airflow onto a collection target. The most common design is so called single stage, employing a sharpened electrode opposite a deposition target, hence achieving both ionisation from the corona discharge and deposition from the E field in a single step. The corona discharge region should not extend to the deposition target, just to a smaller volume around the electrode tip, generating a drift current of negative ions down to the collection plate. Particulates travelling through this region will become charged, and hence experience a force from the E field. The important characteristics are drag on the particle and equilibrium particle charge. The design from Cheng et al employs a sampling tube with cross sectional area of approximately 100mm^2, so airstream velocity will be in the region of 0.1ms^{-1}.

Assuming a spherical particle, $E_{surface} = \frac{Q}{4\pi\epsilon_0 r^2}$. Equilibrium charge will be reached when $E_{surface}$ is the corona inception field. From Zeboudj and Ikene figures 4 and 5 [7] the charge will thus scale roughly linearly with air density[5], at least for a positive electrode (Cheng et al [6] and all other published designs use a negative electrode). For a 2μm particle, if $E_{breakdown} = 2 \times 10^6 \mathrm{Vm}^{-1}$ then $Q = 2.2 \times 10^{-16}$ coulombs, so in a field of $E_{field} = 10^6 \mathrm{Vm}^{-1}$, force $F = 2.2 \times 10^{-10}$N. The slip corrected form of stokes resistance law then gives;

$$F_{slip} = \frac{3\pi}{C_c}\mu V d = \frac{3.6 \times 10^{-10} V}{C_c} \quad (11)$$

At ground level where $C_c \simeq 1$, terminal velocity $\mathrm{V}_t = 0.6ms^{-1}$, so in an airstream of 0.1ms$^-$1 the

[5]Corona discharge in stratospheric condition appears to be poorly quantified, but the mean free path of the molecules is the most important determining characteristic in electrical breakdown, and hence as atmospheric composition is height invariant, low pressure air is a reasonable model.

largest particles can be collected. Drag scales with air density, approximately cancelling the reduction in equilibrium charge as density decreases, and leading to little change in collection efficiency with altitude if $C_c \simeq 1$. Drag also scales with diameter, and as $Q_{equilibrium}$ scales with $\frac{1}{diameter^2}$, particle terminal velocity should be proportional to size. It thus appears this technique is not effective for small particles, but tropospheric number densities are around two orders of magnitude higher than stratospheric, so the airspeed through the sampler can be reduced until we reach the tropopause. $\lambda \simeq 0.6\mu$m at the bottom of the stratosphere, so $C_c \simeq 10 \longrightarrow V_t \simeq 0.6$ ms^{-1} for 0.2 μm particles. Higher in the stratosphere the velocity will increase as C_c increases. Thus the design functions acceptably in the troposphere and stratosphere.

Yung-Sung Cheng et al [6], Laskin and Cowin [8] and Aerosol technology[2] all describe similar designs, which are well suited to sonde use. Cheng et al also found efficiency curves for their single stage precipitator, based on Morrow et al [9]. However no research into collection efficiency as a function of atmospheric pressure was found. Cheng et al found a significant decrease in collection efficiency with particle size (figure 15), supporting the approximate equation for V_t used here.

This technique has the advantage that a scanning electron microscope or SEM sample holder can be used as a target plane, simplifying the particle analysis considerably.

Impactor

This design relies on inertial effects to separate particles from an accelerating airflow (usually around a curved path). It has been used with success before by NASA on ER-2 and DC8 aircraft [5], however an aircraft mounted implementation can easily have a very high velocity airstream and high volumetric flow rate, something that is harder to achieve on a sonde with power, weight and size limitations.

Assuming a radius of curvature of approximately 1mm and airstream velocity of 10ms^{-1}, the acceleration will be approximately 10^5ms^{-2}. Thus a 2μm diameter particle of density 10^4Kgm^{-3} will experience a force of 4×10^{-9}N, an order of magnitude higher than the electrostatic precipitator. However, black carbon soot has a diffuse structure ([1] figure 1) and particle mass scales with diameter cubed whereas Stokes resistance law scales linearly. So at ground level the terminal velocity of a 200nm particle of density 10^3Kgm^{-3} would be an order of magnitude less than the equivalent electrostatic precipitator case. Assuming a constant volumetric flow rate, particle velocity across the airstream would increase with altitude as the acceleration would remain constant whilst drag scales with air density. Unfortunately at ground level there is insufficient acceleration to deposit the smaller particles, as V_t needs to be of the same order as the airstream flow. Lowering the flow rate will further decrease the $\frac{V_t}{V_{airstream}}$ ratio, as acceleration (and hence $F_{inertial}$) will decrease with $V^2_{airstream}$, and increasing flow or reducing the radius of curvature will cause us to break the Reynolds number condition in equation 4.

At greater altitudes, the drag will decrease, and the slip correction factor will become significant. In the lower stratosphere where $\rho = \frac{\rho_{ground}}{10}$ and $C_c \simeq 10$, V_t will have increased by two orders of magnitude to around 6ms^{-1} so deposition will occur. A sonde mounted impactor may work in the upper troposphere and stratosphere, where air density and hence drag is lower, but it appears impossible to collect particles at the lower end of our size range closer to the ground. Differing particle density would make objective measurements difficult unless the instrument was designed to be virtually 100% efficient. Finally, despite research into existing designs, one that was well studied and easily adapted for sonde use was not found.

Filter This would appear a natural choice, but with the low particle densities in the stratosphere, a high flow rate per area of filter is required. The retrieval process would be extremely difficult if not impossible with a fibre based filter, so only membrane filters would be appropriate. Flow through a filter is linear with differential pressure (equation 12), and to reach the required flow rates of around a meter per second or more, all available membrane filters suitable for the size range required a differential pressure in the regions of several tens of kilo-pascals. With a pump mounted downstream, we are limited to atmospheric pressure across the filter, much less than 10Kpa at middle stratospheric altitude. A pump mounted upstream would lead to contamination and particle size bias problems. Thus a filter is inappropriate for operation at altitude.

2.4 Chosen sampling system

The electrostatic precipitator was chosen for the sampling system, primarily due to the existing and well researched designs, and suitability for use in a small, lightweight and low power instrument.

The relatively high collection efficiency found by Cheng et al suggests that an inline sampling

Figure 3: The electrostatic precipitator was machined from perspex, with a piston at the inlet to prevent contamination.

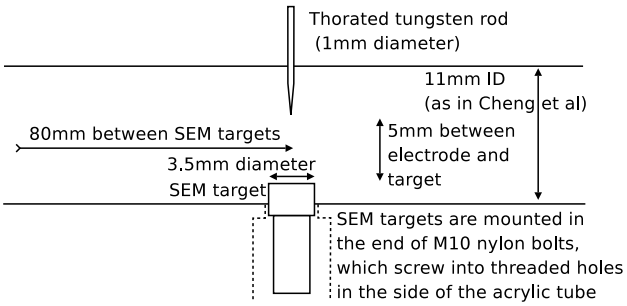

Figure 4: The design was based on Cheng et al, figure 1.

system with multiple target/ioniser pin pairs would be effective. A three sample device was designed, based on three variants of the design from Morrow et al mounted in series in a solid perspex tube. The use of multiple targets is an untried innovation.

Unfortunately this design cannot reach the 0.33mm^2 deposition area target calculated earlier; instead the deposition takes place over an area of around 6mm^2, around 20 times lower particle density. However, with good handling to avoid contamination, it should be possible to locate the particles with some microscope panning. The design is shown in figure 4, and the assembled precipitator in figure 3.

2.5 Avionics

For maximum flexibility the avionics system was designed around the atmel NGW100 embedded Linux board.

This enabled flight software to be easily tested on a Linux PC. Serial ports were then used to

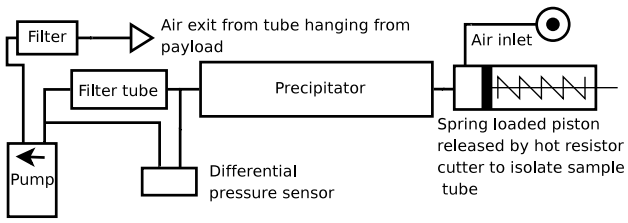

Figure 5: Flow diagram of the air sampling system.

interface with a daughter-board, phone, radio module, and GPS module. The cut-down directly connected to general purpose input output or GPIO port for improved reliability. The NGW100 pulses GPIO high then low on bootup, so a binary coded decimal decoder IC was used to drive the cut-down MOSFETs.

The daughter-board was designed around an atmel atmega168 8 bit AVR micro-controller[6], mainly serving as an IO expander and analogue to digital converter or ADC, but with the pulse width modulation (PWM) output used to drive the pump on the sampling instrument. PWM allows for real time pump control as the ADC reading from a differential pressure sensor connected across the filter between the precipitator tube and pump is proportional to the flow rate. This can be shown by Darcy's law, describing volumetric flow rate Q through a material of permeability κ.

$$Q = \frac{-\kappa A}{\mu} \frac{(P_b - P_a)}{L} \tag{12}$$

In the case of a filter, L is the membrane thickness, and $P_b - P_a$ the pressure drop across the filter. μ is the dynamic viscosity of the airflow, itself a function of temperature, as described by Sutherland's equation.

$$\mu = \mu_0 \frac{T_0 + C}{T + C} \left(\frac{T}{T_0} \right)^{3/2} \tag{13}$$

C is Sutherland's constant (120K for air), T_0 is the reference temperature in kelvin (291.15K for air), and μ_0 is the reference viscosity (1.827×10^{-5} for air).

Over the typical temperature range that the sonde might experience, this leads to a significant change in viscosity, such that using predicted temperatures for the current altitude would not give sufficiently accurate control loop performance. Hence a semiconductor band-gap based

[6] also referred to as a μC

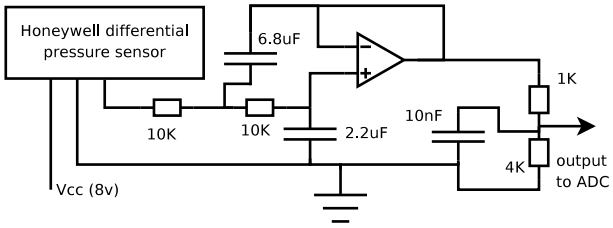

Figure 6: Circuit diagram of the Sallen-Key filter used for the differential pressure sensor.

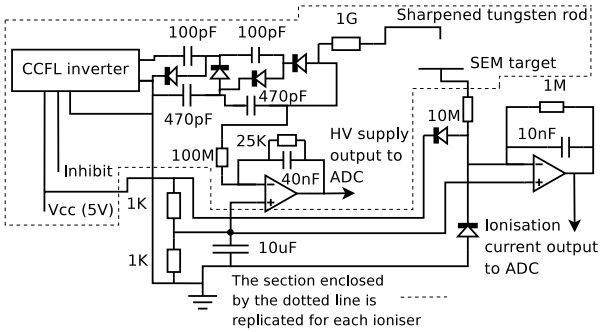

Figure 7: Circuit diagram of the electrostatic precipitator electronics.

sensor IC (LM94022) was placed inside the exit tube from the precipitator directly in the airflow. Unfortunately this was damaged by an mistake during testing, and was replaced by a thermistor and preamp circuit.

Initial testing revealed a potential flaw in the differential pressure control loop; the use of a rotary vane pump lead to high amplitude pressure oscillations in the tubing, and the differential pressure sensor, based on a thin film stress sensor, couples well to high frequency signals. This problem was such that the oscillation amplitude was comparable with the mean value of the differential pressure, leading to a very noisy ADC reading. There was potential for a highly inaccurate pressure figure if a constant phase relationship arose between the ADC sampling rate and a component of the oscillatory signal. The pump may run at any speed down to zero rpm, and as the frequency distribution of the pressure oscillation scales linearly with pump speed there is potential for it to span a large bandwidth. However, investigation with an oscilloscope found insignificant noise below 10Hz. This still presented a slight problem, as it was planned to have a control loop based on error integration with a time constant of around 1 second or less; so a filter with a tight cutoff was required. The Sallen-Key filter in figure 6 was designed with the SPICE simulator to have a cut off edge of approximately 2.5Hz, blocking the oscillatory noise very strongly without impeding the feedback loop.

The precipitator itself requires a high voltage supply for operation, Cheng used 3 to 5 Kv. Research into commercial systems lead to the design of a simple low cost supply employing a cold cathode inverter, designed for LCD back-lights and giving an output of approximately 700v ac, followed by a Cockroft Walton multiplier chain to boost the voltage (see Figure 7). Switching voltages in the Kv range using transistors is quite difficult, so three separate supplies were constructed, one for each sampling electrode. As the dielectric strength of air decreases with decreasing pressure, the two upper atmospheric supplies only had one multiplier stage giving approximately 2.5Kv output, as opposed to 5Kv for the first multiplier.

From Zebboudj et al, it would appear that in the stratosphere, with air density less than 0.1 times its surface value, a supply voltage of less than 1Kv is all that is required. As suggested by Laskin and Cowin, 1GΩ resistors were placed between the HV supplies and electrodes to stabilise the current. For example a current of 1.5uA will lower the electrode voltage to around 1Kv, the inception field at an altitude of around 18 Km or so according to Zebboudj. Thus over a large altitude range current will stabilise at around 1 to 2uA, where Cheng et al obtained high efficiency. The resistors have an additional advantage of ensuring high voltage safety, as the current is limited to a low level and a discharge arc will not cause dangerously high currents.

The high voltage output and ioniser current are monitored by the μC for later analysis.

2.6 Software

There were two main pieces of software to be written, the main control code for the NGW100, in python, and the low level interface code for the μC, in c and compiled with avr-gcc.

The default kernel on the NGW100 does not allow access to the three available serial ports, so a kernel recompile was needed. The pyserial module was then used to interface with the the ports. There is little free space on the NGW100 flash memory, so a 128MB MMC card was used to store data logs and mount python. As the 434MHz radio has a relatively high bit error rate, a Reed-Solomon python module written previously by the Author was used to add forward error correction to the end of the data-packets. This left the actual data human readable to simplify ground station operation. The landing spot

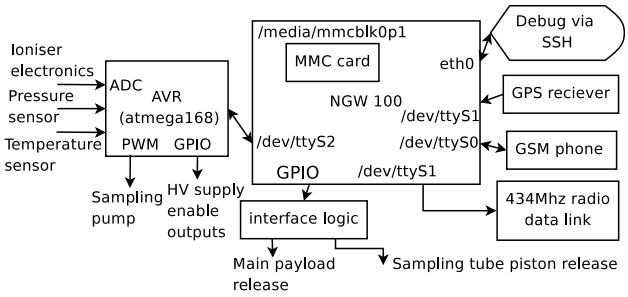

Figure 8: An overview of the payload as it appears from the software's perspective.

prediction code posted on the UK high altitude society website [7] was ported to python, and the parachute drag constants rescaled to the 1.4 meter chute used for payload recovery. This was then combined with a polygon of the East Anglia coastline, and an "exclusion zone" north of London and east of Birmingham, to trigger the cut-down and avoid an urban or sea landing if necessary. Further routines for logging data and relaying telemetry were also written. Communication with the phone was via the extended Hayes AT command set, and the GPS via the standard NMEA serial protocol.

The μC code made heavy use of procyon avrlib [11], a library simplifying tasks such as buffered serial IO and pulse width modulation.

Both programs are in the appendix. The radio modem source and the reed-Solomon module were posted on the UKHAS website.

2.7 Experimental Testing

2.7.1 Aerosol flow testing

This was necessary to investigate precipitator performance, obtain collection efficiency data, and quantify the cross contamination between different electrode/SEM target pairs (figure 4 shows the configuration). An ultrasonic nebuliser was used to generate a H_20 NaCl aerosol, followed by a pair of diffusion dryers and a 5 litre bottle to serve as a neutraliser[8]. A hose was used to supply either the sonde payload or a dilution unit and optical aerosol spectrometer to measure the particle size distribution. The diffusion dryer dehydrated the wet aerosol to give dry NaCl crystals.

The apparatus was observed to be relatively unstable upon being turned on, so the following

[7] http://wiki.ukhas.org.uk/ideas:landing_spot_prediction

[8] This functions by allowing the aerosol charge distribution to reach equilibrium. However, the ^{85}Kr technique employed by Cheng et al would have been more effective.

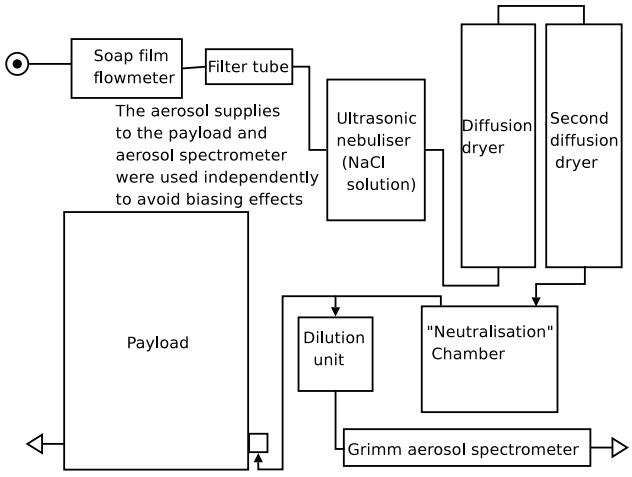

Figure 9: Schematic of the aerosol flow test experiment.

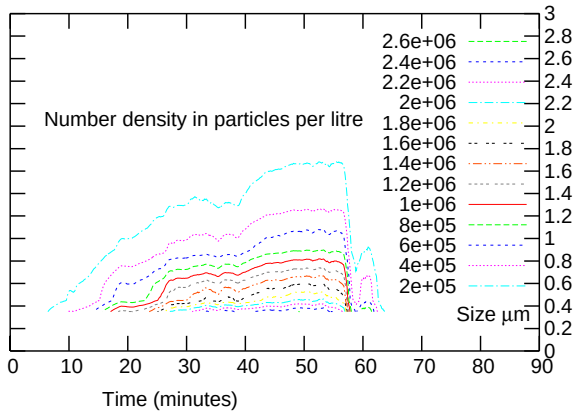

Figure 10: Contour plot of volumetric particle density from ultrasonic nebuliser against size bin, and run time.

graph (figure 10) was obtained from an hour long test for calibration purposes.

To obtain a useful precipitator sample, the nebuliser was planned to be on for 15 minutes, but it was found that the ioniser current dropped to zero after only seven minutes. This resulted in a collected aerosol weight only slightly greater than the accuracy of our balance.

The ionisation current over time is plotted in figure 11. It seems reasonable to assume that decreasing current is due to the coating of salt deposited on the SEM target. As this is an insulator, once a layer has built up a surface charge may accumulate. This reduces the electric field between the electrode rod and target until corona discharge no longer occurs, and the ionisation current drops to zero.

A fit of the form $a + be^{cx}$ was attempted for

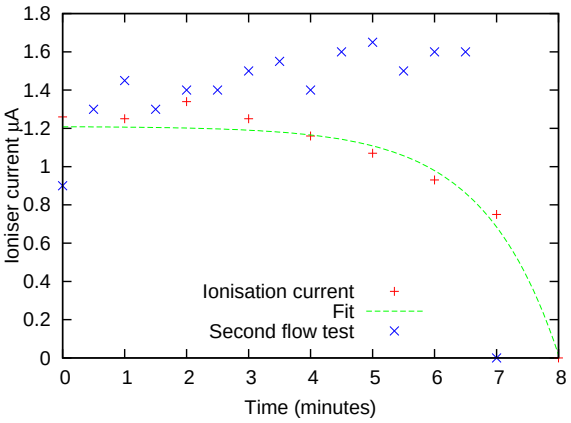

Figure 11: Ioniser current plotted against time, with a function of the form $a + be^{cx}$ fitted to the data.

Variable	Fit Value	Standard error
a	1.21	± 0.0468
b	-0.00167	± 0.00165
c	0.821	± 0.121

Table 1: Fit of exponential to ionisation current.

the first test data. In figure 11 the first, third and forth datapoints are not used in the fit, as the small increase in current at 2 minutes was believed to be due to increasing humidity as the nebuliser is turned on, not the target plane coating effect responsible for the current drop. In a second flow test however, the behaviour was somewhat different, with a higher current and a very sharp ionisation current cut off, so the effect clearly needs further investigation before any conclusions can be reached. One possible explanation is differing humidity of the aerosol stream, the silica gel in the diffusion dryers could have started to saturate by the time of the second test.

Figure 12 shows the third SEM target, which was placed furthest upstream, after the first flow test. It can been seen that the contamination of the second target (next downstream) was minimal (see figure 13) there was little visible deposition on the first target - furthest downstream.

In table 2 it can be seen that the collection ratios are at least $\frac{\Delta M_3}{\Delta M_2} < \frac{1}{3}$ and possibly $< \frac{1}{6}$ from the fact that the mass of the second target was ± one least significant digit both before and after the experiment. Unfortunately the limitations of the balance prevent further investigation.

Interestingly the overall efficiency is low, considerably lower than Cheng et al [6] judging from the filter mass increase. However, some of this

Figure 12: The third SEM target after the test, showing a thick coating of salt.

Figure 13: The second and first targets, the second target showing a slight dusting of salt.

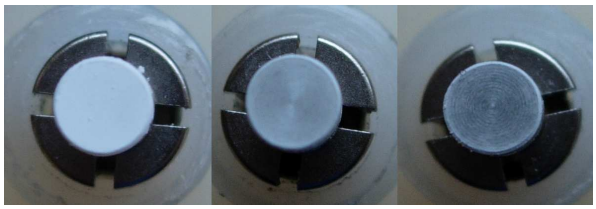

Figure 14: The SEM targets after the second flow test, third is lefter-most, first righter-most. The second showing a lighter coating of salt than the third, and the first only a slight dusting.

Target	Initial Mass (mg)	Final	Difference
1	250.0	250.0	0.0
2	247.(0/1)	247.(0/1)	0.(0/1)
3	247.5	247.8	0.3
Filter	36240.2	36249.2	9.0

Table 2: Measured mass of the SEM targets and Filter Tube before and after the first flow test.

Target	Initial Mass (mg)	Final	Difference
1	249.(8/9)	250.0	0.0
2	246.(8/9)	247.9	0.(0/1)
3	247.4	247.8	0.4
Filter	36248.4	36248.0	-0.4

Table 3: Measured mass of the SEM targets and Filter Tube before and after the second flow test.

mass increase was likely due to moisture absorption, as demonstrated by the second test results, where filter mass decreased. Secondly, although the nebuliser was turned off ten minutes into the experiment, the characterisation test shows that aerosol would have continued to enter the precipitator for some time after. Using the characterisation test results it would be possible to estimate how much aerosol passed through the instrument after the ionisation current dropped to zero. However the characterisation test had a longer run-time before the nebuliser was turned off, so has limited usefulness.

A second flow test was carried out to investigate efficiency further. The nebuliser and pump were turned off as soon as the ionisation current had dropped to zero, to avoid further aerosol entering the instrument. Interestingly, the filter mass was found to decrease (table 3). It was suspected that this was due to dehydration from the low humidity aerosol laden airflow - the filter had been left with both ends exposed to the laboratory air beforehand.

Leaving the filter tube in the laboratory to reach equilibrium humidity resulted in the mass increasing confirming the hypothesis (table 4). It would appear that the filter tube absorbed 0.6 mg of salt, however we do not know how much aerosol is removed from the airstream by impaction with parts of the instrument- there was observed salt deposition on parts of the tubing. The filter mass measurement can thus only be used to put an upper limit on the total collection efficiency of $\frac{0.0004}{0.0004+0.0006} = 40\% \pm 7.2\%$.

An improved method for evaluating deposited aerosol quantities might be to dissolve in distilled water, then measure the conductivity to infer the NaCl content. Unfortunately all conductivity meters researched had insufficient accuracy to detect NaCl in the sub milligramme range. A titration technique might provide an accurate figure.

Time after experiment	Mass
15	36248.7
25	36248.9
54	36248.9
93	36249.0

Table 4: Filter mass in grams versus time in minutes after the filter was removed and allowed to reach humidity equilibrium with the lab air.

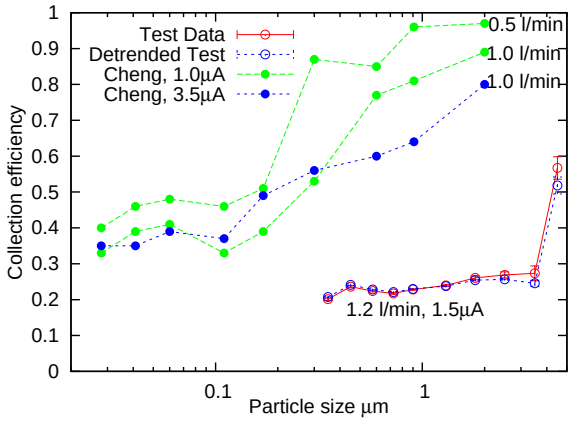

Figure 15: The collection efficiency as a function of particle size, error bars are one standard error. The results from Cheng et al [6] are also shown, along with the results from the detrending technique.

The aerosol spectrometer however offers a more powerful technique; the spectrometer was placed on the electrostatic precipitator exhaust (with the pump and filters bypassed)[9], and the aerosol spectrum analysed with the high voltage on or off. A dilution unit being employed to reduce the particle number density to levels usable by the spectrometer. The results of this experiment are shown in figure 15

The fluctuation of the nebuliser output was a significant problem during the test. From the aerosol spectrometer data it can be seen that although the number densities are far from constant with the ioniser on and off, the variation is mostly due to trends in the nebuliser output (see figure 16). Figure 15 shows separate lines for the

[9]The aerosol spectrometer contains its own pump and volumetric flow control loop, so the payload pump had to be disconnected whilst the spectrometer was in use downstream of the precipitator.

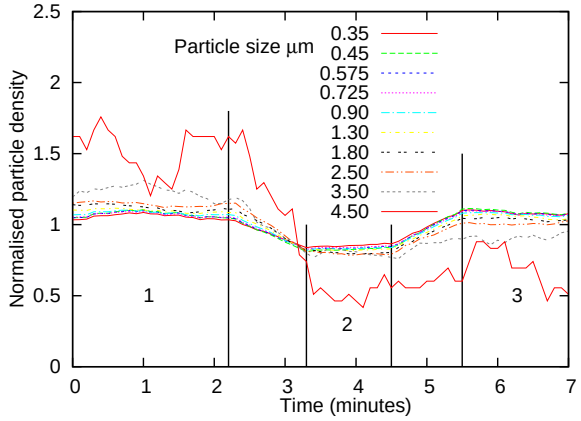

Figure 16: The spectrometer data, with sizebins normalised relative to one another.

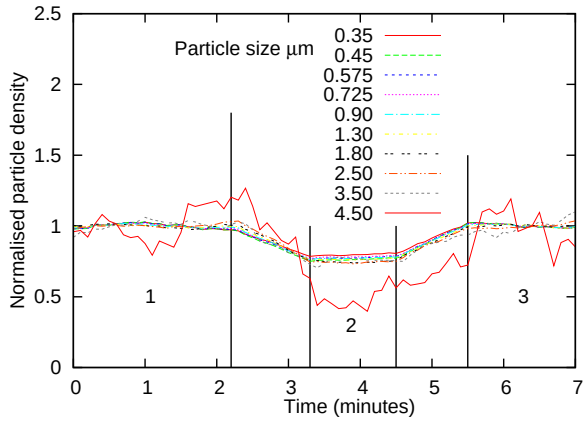

Figure 17: The "detrended" data; particle volumetric number density as a fraction of a linear fit for the time periods labelled by 1 and 3.

results with errors approximated as random and for the results when linearly detrended.

The detrending technique was based on a linear least squared error fit of the data either side of the region where high voltage was applied (using time periods 1 and 3 on the graph). This reduced the particle density fluctuation problem a evident in figure 17. The efficiencies shown in figure 15 were then calculated from a comparison of the data from the time span labelled by 2 to the data from time periods 1 and 3.

A Fourier transform based filter applied over the duration of the experiment might also help to reduce error, as would running the experiment for longer before taking data. The data was recorded around 35 minutes after the nebuliser was turned on - from figure 10 it can be seen that waiting for over 50 minutes would have given more reliable data.

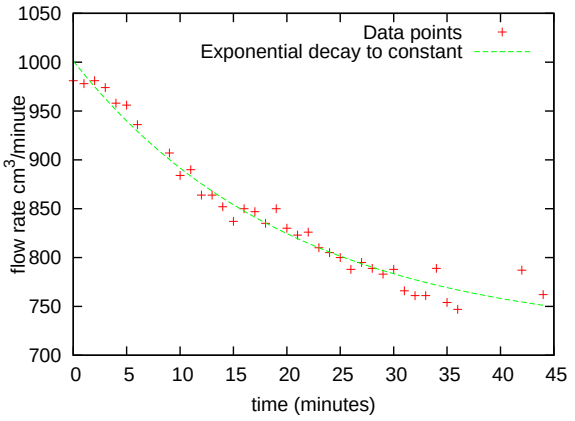

Figure 18: Test data over a 45 minute period, with an fit of the form $a \times e^{bt}$ where t is time in minutes.

Variable	Final Value	Standard Error
a	282.896	± 12.64
b	-0.0490468	± 0.00573
c	718.466	± 14.57

Table 5: Computed fit for figure 18.

The control loop on the AVR driven daughterboard was evaluated during the (much longer duration) first flow test. As expected the volumetric flow rate was found to be linear with differential pressure across the filter (figure 19).

However, during the experiment, a long term drift in flow rate was observed. In figure 18 the flow rate is shown together with an exponential fit.

Setting different control loop set-points for the μC, and fitting with a straight line of the form $y = ax + b$, figure 19 was obtained.

Tables 6 and 7 give the linear fit parameter. It is interesting to note that the gain of the system, or a in the tables, changes by only slightly more than one standard error $\sqrt{3.44^2 + 1.29^2} = 3.67$ as opposed to $43.14 - 38.73 = 4.41$, so it would appear that the exponentially falling response is due to the null of the pressure sensor, or b in our linear fit.

The change in the system null is $87.3 - 257.8 \pm \sqrt{40.19^2 + 79.69^2} = -170.5 \pm 89.3$. The predicted decay is given by $282.9(e^{-45 \times 0.04905} - 1) = -251.8$ so this is within standard errors.

Hence in summary it appears that the pressure sensor null is responsible for the drift in volumetric flow rate. A simple change to the AVR firmware was made to resolve the issue, turning off the pump and measuring the differential pres-

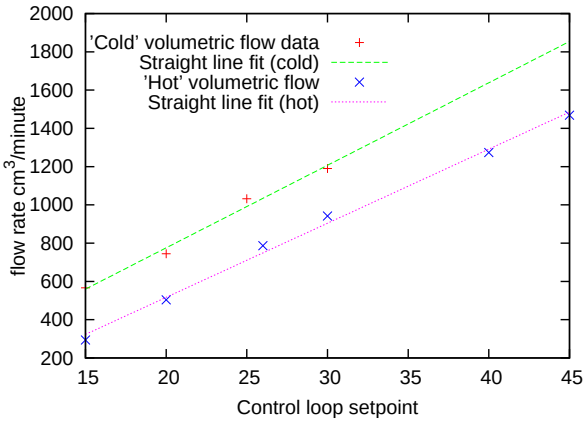

Figure 19: Data and Linear fits for the control loop response straight after turn on, and after 45 minutes of operation

Variable	Fit Value	Standard error
a	43.14	± 3.437
b	-87.3	± 79.69

Table 6: Linear fit to control system response at start of experiment.

Variable	Fit Value	Standard error
a	38.7302	± 1.29
b	-257.834	± 40.19

Table 7: Linear to control system response after 45 minutes.

sure sensor null-point every 15 minutes or thereabouts .

2.7.2 Software validation

As the main flight control software was written in python to run on a Linux system, simply removing the serial interface was all that was required for it to run on a PC. In place of the pyserial module, files were used to simulate input and output from the embedded board. The most important function of the flight software is of course to enable a recovery of the payload after the flight, so GPS input needs to be replicated and output to the radio and phone logged for analysis. The standard protocol for GPS chipsets is NMEA, a data-packet over serial protocol, where each packet has a header, data fields of comma separated variables, and finally parity data to identify errors.

The university of Wyoming have produced an online application for simulating balloon flights [10], which produces an output in KML, an XML format. A simple C program posted on the UKHAS wiki[11] was used to convert simulated flights into NMEA format. The flight code was thus tested under realistic flight conditions, allowing the landing spot prediction and polygon flight boundary, or "geofence" to be debugged, which would otherwise have been impossible. The flow tests provided ample opportunity to debug the hardware interface code and AVR.

[10] This can be found at http://weather.uwyo.edu/polar/balloon_traj.html

[11] Posted at http://wiki.ukhas.org.uk/code:emulator

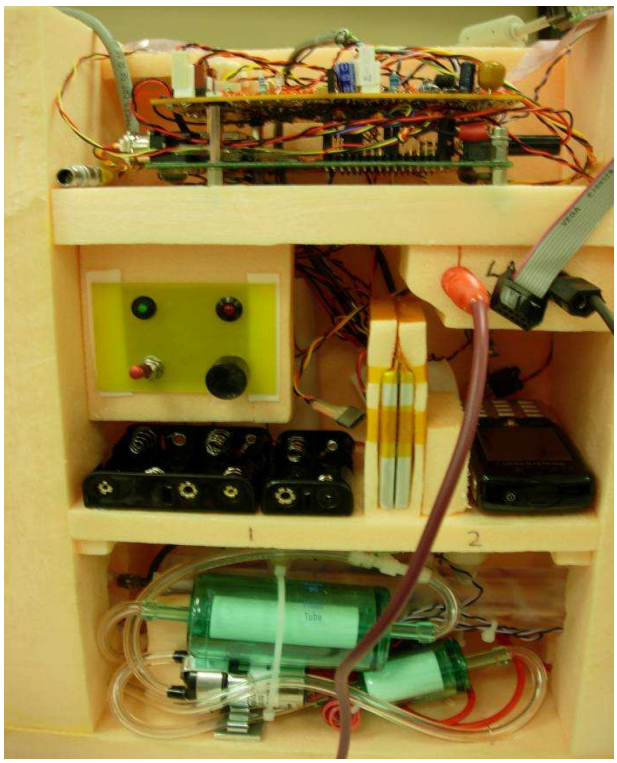

Figure 20: The assembled payload with the enclosure lid removed.

3 Conclusions

The sonde payload serves as a useful tool for atmospheric aerosol analysis, and the department plan to fly it shortly.

The instrument allows three samples to be obtained at different vertical altitude ranges, but there is room for performance improvements. Deposited particle density on the collection surface is low, there is cross contamination between samples, and the overall number of sample targets could be increased to give greater vertical resolution.

A possible improvement would be to use a mechanical means of target replacement. For exam-

ple multiple SEM targets could be used with a rotating mount, avoiding cross contamination issues.

The collection efficiency of the precipitator is much lower than might have been expected from Yung-sung Cheng et al (figure 15). This discrepancy is hard to explain, but the instrument designs are not quite identical. Although Cheng et al did not use the exact flow rate and ionisation current of our test, a comparison of their collection efficiencies at currents and flow rates similar to ours shows that this is a minor factor. A more significant distinguishing factor is likely to be the technique employed for aerosol generation, in particular the absence of ^{85}Kr neutralisers in our apparatus would have lead to a charged aerosol. This would have had significant but as yet unquantified effects on precipitator operation, which is dependent upon particle charge.

References

[1] D.F. Blake and K. Kato, 1995: Latitudinal distribution of black carbon soot. Journal of geophysical research, Vol 100, No. D4, 7195-7202

[2] William C. Hinds, 1999, Aerosol technology, properties, behaviour, and measurement of airborne particles. Wiley.

[3] R.F. Pusechel and Coauthors, 1992: Black carbon (soot) aerosol in the lower stratosphere and upper troposphere. Geophysical Research lett., Vol 19, No. 16, 1659-1662

[4] Brenninkmeijer, C. A. M., and Coauthors, 1999: CARIBIC- Civil aircraft for global measurement of trace gases and aerosols in the tropopause region. J. Atmos. Oceanic Technol., 16, 1373-1383

[5] Snetsinger and Coauthors, 1987: Effects of the March-April 1982 El Chichon eruption on stratospheric aerosols, late 1982 to early 1984. Journal of Geophysical research, Vol 92, No. 14, 761-771

[6] Yung-sung Cheng et al, 1981: Collection efficiencies of a point-to-plane electrostatic precipitator. American industrial hygiene Ass. Journal, 42, 605-610.

[7] Y. Zebboudj and R. Ikene, 2000: Positive corona inception in HVDC configurations under variable air density and humidity conditions. The European journal of Applied Physics, 10, 211-218

[8] Alexander Laskin and James P. Cowin, 2001: On deposition efficiencies of a point-to-plane electrostatic precipitator. Journal of aerosol science, 33, 405-409

[9] Morrow, P.E., & Mercer, T. T. 1964: A point-to-plane electrostatic precipitator for particle size sampling, American industrial hygiene Ass. Journal, 25, 8-14

[10] UK Air Passenger Demand and CO2 Forecasts, Department for transport 2007.

[11] http://www.mil.ufl.edu/~chrisarnold /components/microcontrollerBoard/AVR/avrlib

[12] The UK high altitude society is a not for profit umbrella organisation to facilitate co-operation between UK high altitude balloon groups- http://www.ukhas.org.uk

[13] Cambridge University Spaceflight is a student organisation focused on high altitude and space flight- www.srcf.ucam.org/~cuspaceflight

[14] http://data.energizer.com/PDFs/l91.pdf

APPENDIX

.1 Construction photos

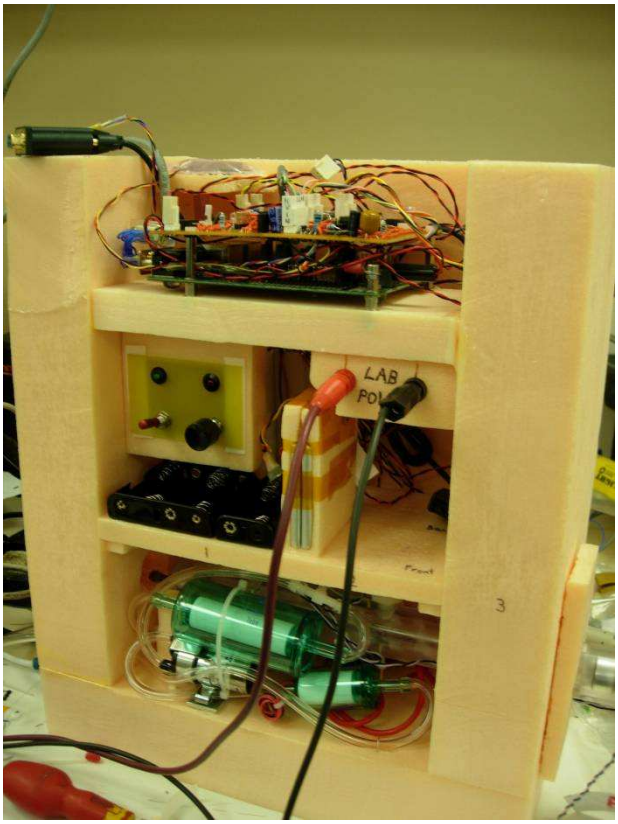

Figure 21: The assembled payload with the enclosure lid removed - side view.

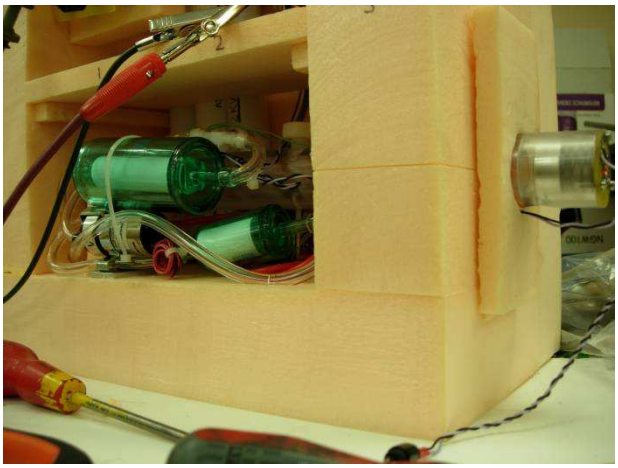

Figure 22: Close up view of the assembled sampling instrument.

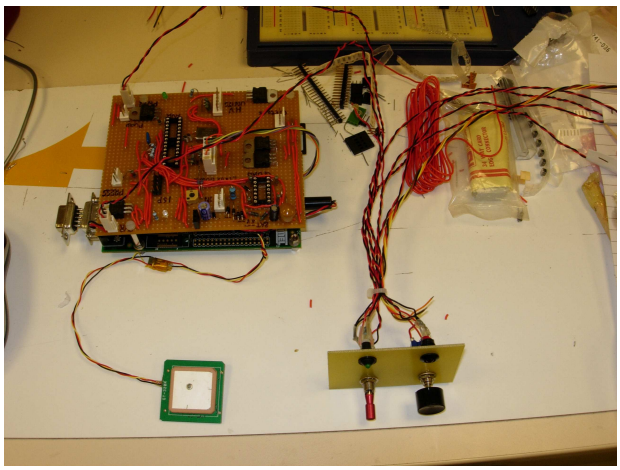

Figure 23: The electronics before installation in the enclosure.

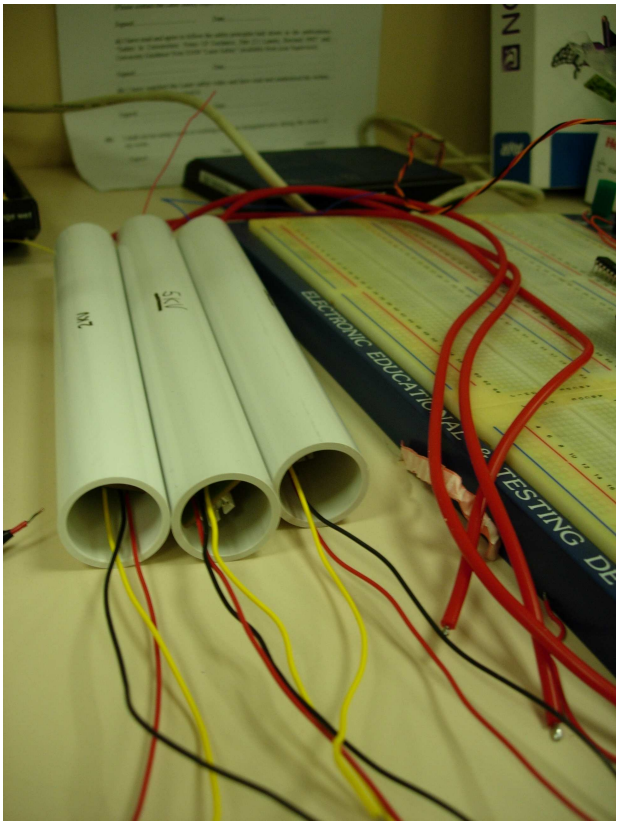

Figure 24: The high voltage supplies were potted in acrylic tube for safety.

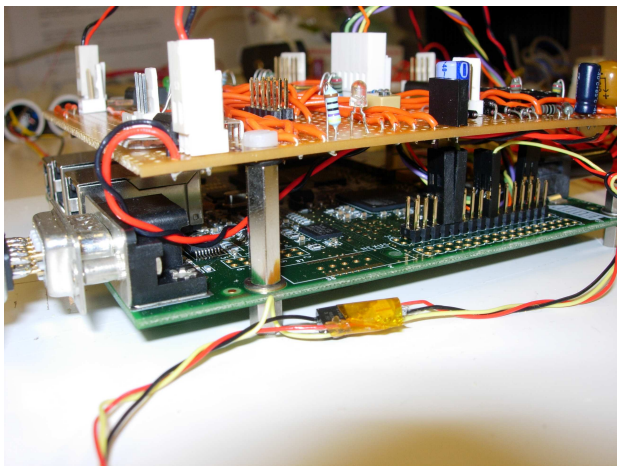

Figure 25: The Linux board with daughterboard on top.

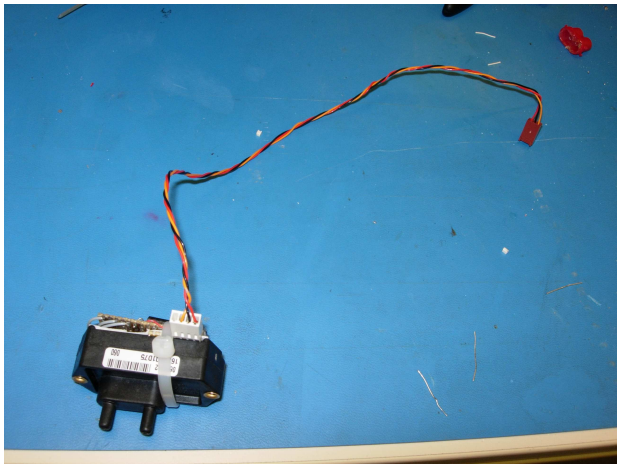

Figure 26: The differential pressure sensor with low pass filter board attached.

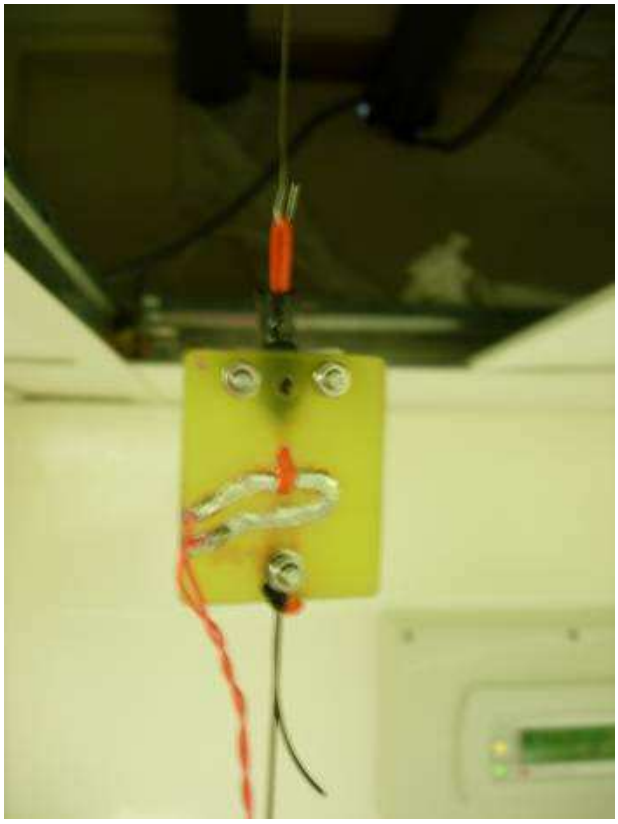

Figure 27: The cutdown mechanism uses a 10Ω resistor to cut through the orange plastic line, releasing the payload from the balloon.

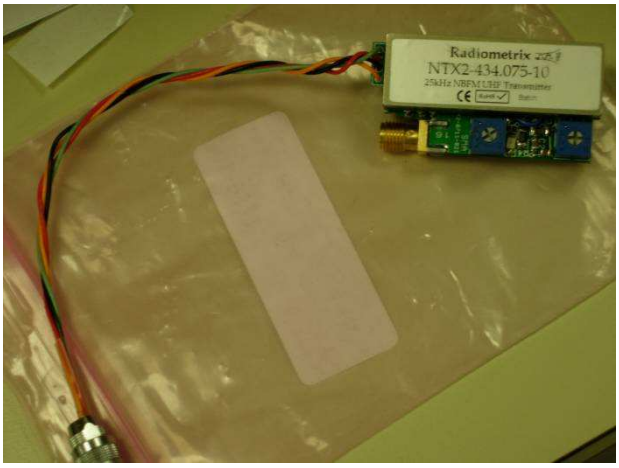

Figure 28: The assembled 434MHz radio datalink.

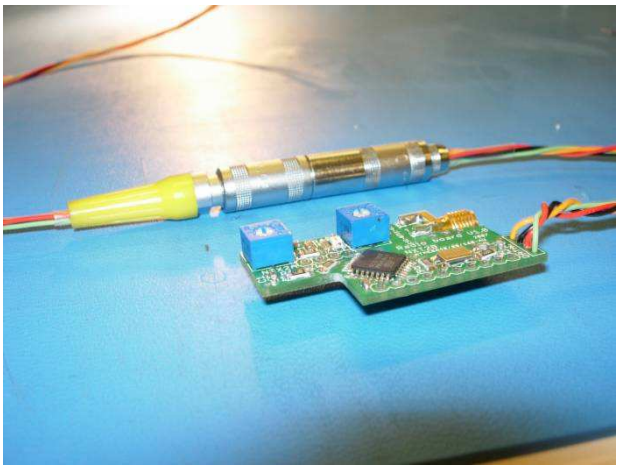

Figure 29: The radio modem board being assembled.

.2 Included figures

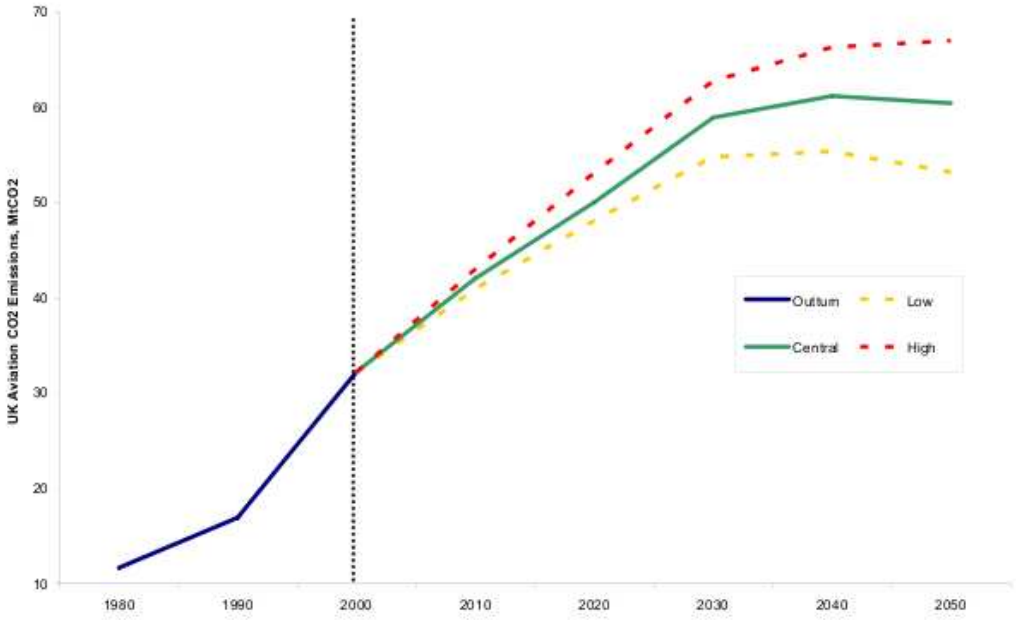

Figure 30: Predicted UK aviation CO_2 emissions from [10].

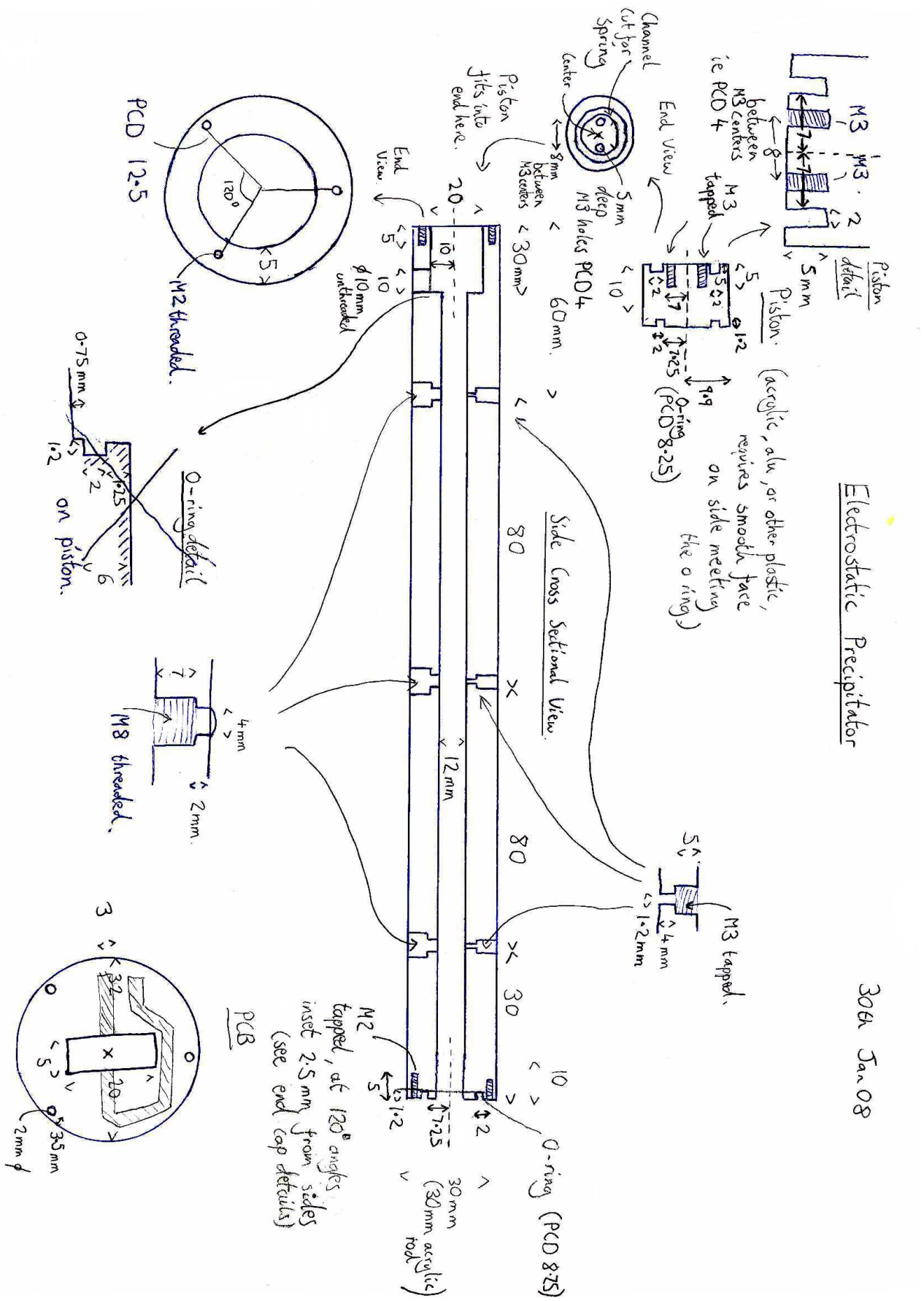

Figure 31: Engineering design for precipitator.

Electrostatic Precipitator Part details.

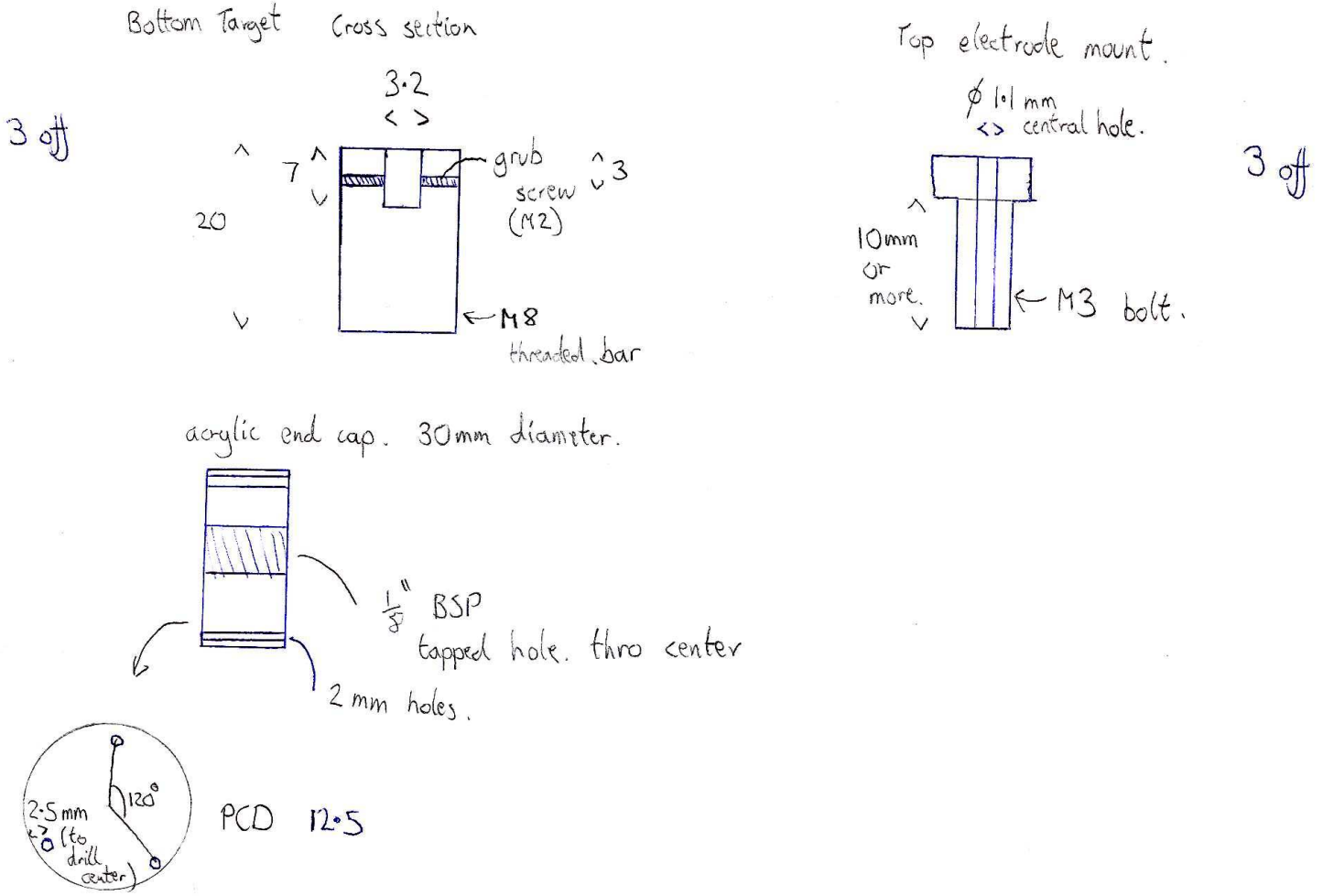

Figure 32: Further precipitator details.

.3 Source code

Listing 1: Source code for daughterboard AVR microcontroller

```
#include "main.h"


int get_char0(FILE* stream)
{
        int c=-1;
        while(c==-1)
        {
                c=uartGetByte();
        }
        //getchar gets the stream
        //loop until we get something
        return (u08) c;
}

int put_char0(char c, FILE *stream)
{
        loop_until_bit_is_set(UCSR0A, UDRE0);
        //unbuffered tx comms
        UDR0 = c;
        return 0;
/*
        if(uartReadyTx || !uartBufferedTx )
        //if buffer is empty or we arent buffered
        {
                uartSendByte(c);
                //sends directly to uart
                return 0;
        }
        else
        {
        //while(!bufferAddToEnd(Txbuff0,c));
        //send the character to buffer
        if(uartAddToTxBuffer(c))
        {
                return 0;
        }
        else
        {
                return -1;
        }*/
}

int pseudoscanf()
{
        char s[20];
        int d;
        fgets(s,20,stdin);
        sscanf(s,"%d",&d);
        return d;
}

float getHVvalue(int n)
{
        if(n>0 && n<4)
        //sanity check
        {
                if (n==1) n=0;
                //corrects mistake on the board
                return hvfactor*((float)a2dConvert10bit(n)-520.0);
```

```
61                  //correct for unbalanced resistors on board
62          }
63          else
64          {
65                  return -1;
66          }
67  }
68
69  void overflow(void)
70  {
71          float delta=i_term*(p_setpoint-pressuredifference());
72          if(position>-delta) position+=delta;
73          //ie we're not going to end up with -ive position
74          if (position>1400.0) position=1400.0;
75          //maximum limit on duty cycle
76          timer1PWMASet((u16)position);
77  }
78
79  void temperatureprint()
80  {
81           int temp=a2dConvert10bit(temppin);
82           printf("%d,%.1f\r\n",temp,-16.436*log(1.11258*((1024.0/(double)temp)-1.0)));
83           //returns the thermistor temperature
84  }
85
86  float pressuredifference()
87  {
88          return (float)pressurefactor*((float)a2dConvert10bit(pressurepin)-pressurenull);
89  }
90
91  void set_control_loop(d)
92  {
93          if(d!=0)
94          {
95                  if(!pump) timer1PWMAOn();
96                  //if we set a target of zero,
97                  //it turns off pwm completely,
98                  //which is handy
99                  pump=TRUE;
100                 timerAttach(1,overflow);
101                 //ISR on pwm falling edge
102                 p_setpoint=(float)d/1000.0;
103         }
104         else
105         {
106                 pump=FALSE;
107                 timerDetach(1);
108                 //ISR off pwm falling edge
109                 timer1PWMAOff();
110         }
111 }
112
113 void HVenable(int n)
114 {
115         if (n<4 && n>-1)
116         {
117                 PORTD|=0b00011100;
118                 //turn it all off
119                 if (n) PORTD&=~(1<<(n+1));
120                 //sending 0 will turn off
121                 //all the HV supplies
122                 printf("HV %d enabled\r\n",n);
```

```
123             }
124             else printf("%d is not a HV channel\r\n",n);
125     }
126
127     void nullset(void)
128     {
129             u08 n;
130             u32 s=0;
131             for(n=0;n<255;n++)
132             {
133                     s+=a2dConvert10bit(pressurepin);
134                     _delay_ms(7);
135             }
136             pressurenull=(float)s;
137             pressurenull/=256;
138             printf("%f = pressurenull\r\n",(double)pressurenull);
139     }
140
141     int main(void)
142     {
143     //int nodata=0;
144     int d;
145     signed char c;
146     //hv outputs
147     DDRD=0b00011100;
148     //set the outputs high ie all HV supplies off
149     PORTD=0b00011100;
150     //pump pwm
151     DDRB=0b00000010;
152     PORTB=0x00;
153     uartInit();
154     uartSetBaudRate(19200);
155     FILE mystdio0 = FDEV_SETUP_STREAM(put_char0, get_char0, _FDEV_SETUP_RW);
156     //so we can printf to the radio
157     stdout = &mystdio0;
158     //set our stdio out function
159     stdin = &mystdio0;
160     sei();
161     //while(1)
162     //{
163     //printf("hello\r\n");
164     //_delay_ms(20);
165     //}
166     timerInit();
167     timer1SetPrescaler(TIMER_CLK_DIV1);
168     timer1PWMInitICR(4095);
169     //timer1PWMAOn();
170     timer1PWMASet(0);
171     a2dInit();
172     puts("setup ok\r\n");
173     nullset();
174     while(1)
175     {
176             c=uartGetByte();
177             //scanf("%c",&c);;
178             //if(c!=-1)
179             //{
180             //printf("%c\r\n",c);
181             switch(c)
182             {
183                     case 'V':
184                     //HV voltage feedback
185                     printf("%.0f\r\n",(double)getHVvalue(pseudoscanf()));
```

```
                    //we only need it to the nearest volt
                    break;
                    case 'D':
                    printf("%d,%.4f\r\n",a2dConvert10bit(pressurepin),
                    (double)pressuredifference());
                    //Pressure feedback
                    break;
                    case 'H':
                    d=pseudoscanf();
                    printf("got %d",d);
                    HVenable(d);
                    //turn on a HV supply
                    break;
                    case 'P':
                    d=pseudoscanf();
                    //set a differential target - note in units of % of full range
                    set_control_loop(d);
                    break;
                    case 'C':
                    d=pseudoscanf();
                    //set a correction factor
                    i_term=(float)d/16.0;
                    break;
                    case 'K':
                    d=(int)(p_setpoint*1000.0);
                    //store the setpoint
                    set_control_loop(0);//turn off pump
                    for(c=0;c<100;c++) _delay_ms(10);//wait 1 second
                    nullset();
                    set_control_loop(d);
                    break;
                    case 'T':
                    temperatureprint();
                    break;
                    case 'I':
                    printf("%.3f\r\n",ionfactor*((double)a2dConvert10bit(ionpin)-520.0));
                    //return the ionisation current-opamp is using the split ie 2.5V rail
                    break;//hence the 512
                    //default:
                    //if (c!=-1) printf("command not found\r\n");
                    //nodata=0;
            }
            //}
            /*else
            {
            nodata++;
            for(d=0;d<50;d++)
            {
                    _delay_ms(10);
            }
            //printf("%d,%f\r\n",nodata,(double)pressuredifference());
}*/
}
return 0;
}
```

Listing 2: Header file for daughterboard AVR (links to procyon avrlib [11])

```
#include <avr/interrupt.h>
#include <stdio.h>
#include <math.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
```

```
#include <avr/pgmspace.h>


#include "global.h"
#include "uart.h"
#include "buffer.h"
#include "a2d.h"
#include "avrlibtypes.h"
#include "avrlibdefs.h"
#include "timerx8.h"

#define pressurepin 4
#define ionpin 1
#define temppin 5

#define hvfactor 12.27                                  //v
#define pressurefactor -0.0002                          //needs to be converted to psi
#define temperaturefactor -0.359                        //degrees C
#define temperaturenull 539.0                             //for the LM94022
#define ionfactor 5.0/1024.0                             //u Amps
#define defaultiterm 50
#define setpointdefault 0

FILE* stream;
int get_char0(FILE* stream);
int put_char0(char c,FILE* stream);
void nullset(void);
void HVenable(int n);
void set_control_loop(int d);
float getHVvalue(int n);
float pressuredifference();
int pseudoscanf();

volatile float pressurenull;
volatile float p_setpoint=setpointdefault;
volatile float position;
volatile float i_term=defaultiterm/16.0;
char pump=FALSE;
```

Listing 3: Python script for NGW100

```
#!/media/mmcblk0p1/install/bin/python
##!/usr/bin/python


import reed_solomon
import serial
import math
import time
import os
TRUE=1
FALSE=0
DEG2RAD=math.pi/180.0
class updatest:
        pass
mynumber="447913335472"
default_target_pressure=40
limit=[200,10000,20000]          #our altitude ranges for the different samples
max_flight_time=7000
callsign="AOPP"
logname="flightlog.txt"
Roottwotwomgovercda = (2*9.81*2/0.72)**0.5 #work this out for our payload mass,
# cda from http://members.aol.com/nakkarocketry/paratest.html
iterations=0
Jxpoints=[]
Jypoints=[]
updatestuff=updatest()
```

```
27
28
29
30  def gps_status():
31          print 'Retrieving GPS status'
32          gpspos_parts=['']
33          while not gpspos_parts[0]=='$GPGGA':
34                  gpspos=ser_gr.readline()
35                  print "got" , gpspos
36                  gpspos_parts = gpspos.split(',')
37          if ((gpspos_parts[6]=='1') or (gpspos_parts[6]=='2') or (gpspos_parts[6]=='3')):
38   #2D or 3D fix/DGPS fix
39                  print 'GPS fix' , gpspos_parts[6] , 'ie valid'
40                  status=TRUE
41          else:
42                  print 'GPS fix' , gpspos_parts[6] , 'ie not valid'
43                  status=FALSE
44          return status
45
46  def gps_data():
47          ser_gr.flushInput()                 #as gps strings will have built up over time
48          print 'Retrieving GPS data'
49          gpspos_parts=['']
50          while not gpspos_parts[0]=='$GPGGA':
51                  gpspos=ser_gr.readline()
52                  gpspos_parts = gpspos.split(',')
53          gpstime=3600*float(gpspos_parts[1][0:2])+60*float(gpspos_parts[1][2:4])\
54          +float(gpspos_parts[1][4:])
55          latitude=float(gpspos_parts[2][0:2])+float(gpspos_parts[2][2:])/60.0
56          if gpspos_parts[3]=="S":
57                  latitude=-latitude
58          longitude=float(gpspos_parts[4][0:3])+float(gpspos_parts[4][3:])/60.0
59          if gpspos_parts[5]=="W":
60                  longitude=-longitude
61          if gpspos_parts[9]=='':
62                  gpspos_parts[9]='0.0'
63          altitude=float(gpspos_parts[9])
64          #print gpstime,latitude,longitude,altitude
65          return gpstime,latitude,longitude,altitude
66
67  def db_stats():
68          ser_daughter.flushInput()#we dont want to mess things up
69          #with old data in the buffer
70          ser_daughter.write("V1\r\n")        #the high voltage values
71          V1=ser_daughter.readline()
72          ser_daughter.write("V2\r\n")
73          V2=ser_daughter.readline()
74          ser_daughter.write("V3\r\n")
75          V3=ser_daughter.readline()
76          ser_daughter.write("I\r\n")          #the total ioniser current
77          IC=ser_daughter.readline()
78          #ser_daughter.write("B\r\n")          #battery voltage
79          #VB=ser_daughter.readline()
80          ser_daughter.write("D\r\n")          #pressure difference
81          PD=ser_daughter.readline()
82          ser_daughter.write("T\r\n")     #new - temperature sensor, to be fitted
83          TS=ser_daughter.readline()
84          return V1[:len(V1)-2],V2[:len(V2)-2],V3[:len(V3)-2],IC[:len(IC)-2]\
85          ,PD[:len(PD)-2],TS[:len(TS)-2]
86
87  def HV_enable(n):
88          stringy='H'+str(n)+"\r\n"
89          ser_daughter.write(stringy)
```

```
90          print ser_daughter.readline()
91
92  def Set_pressure_target(P):
93          ser_daughter.write('P'+str(P)+"\r\n")
94
95  def air_density(alt): # NASA air temperature/pressure/density model
96          if (alt < 11000.0):
97           # below 11Km - Troposphere
98                  temp = 15.04 - (0.00649 * alt)
99                  pres = 101.29 *(((temp + 273.1) / 288.08)**5.256)
100         else:
101                 if (alt < 25000.0):
102          # between 11Km and 25Km - lower Stratosphere
103                         temp = -56.46
104                         pres = 22.65 * math.exp(1.73 - ( 0.000157 * alt))
105                 else:
106          # above 25Km - upper Stratosphere
107                         temp = -131.21 + (0.00299 * alt)
108                         pres = 2.488 *(((temp + 273.1) / 216.6)**-11.388)
109         return(pres / (0.2869 * (temp + 273.1)))
110
111
112 def Updatepredict(Seconds,North,East,Altitude):
113         Deltae=East-updatestuff.Oldeast
114         Deltan=North-updatestuff.Oldnorth
115         Deltaa=Altitude-updatestuff.Oldaltitude
116         Deltat=Seconds-updatestuff.Oldseconds
117         Averagealt=(updatestuff.Oldaltitude+Altitude)/2
118         if Deltaa>0:
119                 K=Roottwotwomgovercda*(air_density(Averagealt)**-0.5)
120                 #we now have the velocity of decent
121                 print "Descent velocity=",K
122                 K=Deltaa/K
123                 #time of decent for this layer
124                 print "descent time=",K
125                 K=K/Deltat
126                 #weighting for this layer
127                 print "Layer weighting=",K
128                 updatestuff.Predictedn+=(K+1)*Deltan
129                 #need to account for ascent drift
130                 updatestuff.Predictede+=Deltae*( (K*math.cos(((North + \
131                 updatestuff.Oldnorth)/2)*DEG2RAD)/math.cos(\
132                 updatestuff.Predictedn*DEG2RAD)) +1)
133         updatestuff.Oldeast=East
134         updatestuff.Oldnorth=North
135         updatestuff.Oldaltitude=Altitude
136         updatestuff.Oldseconds=Seconds
137         print "predicted north=",updatestuff.Predictedn
138         print "predicted east=",updatestuff.Predictede
139         return updatestuff.Predictedn,updatestuff.Predictede
140
141
142 def are_we_inside(gpstime,Xpos,Ypos,altitude):
143         count=False
144         print "checking",Xpos,Ypos
145         for g in range(len(Jxpoints)):
146                 if g==len(Jxpoints)-1:
147                         gplus=0
148                 else:
149                         gplus=g+1
150                 if ((Jypoints[g]>Ypos and Jypoints[gplus]<Ypos) or \
151                 (Jypoints[g]<Ypos and Jypoints[gplus]>Ypos))and(((Jxpoints[g]\
152                 -Xpos)+((Ypos-Jypoints[g])*(Jxpoints[gplus]-Jxpoints[g])/(\
153                 Jypoints[gplus]-Jypoints[g])))>0):
154                         count=not count
155                         print "intercept",g
```

```
156         print 'we are inside=',count
157         return count
158
159 def load_kml(filepath):
160         xpoints=[]
161         ypoints=[]
162         found=False
163         for line in open(filepath,'r').readlines():
164                 if not line.find("<coordinates>")==-1:
165                         found=True
166                 if not line.find("</coordinates>")==-1:
167                         found=False
168                 if found:
169                         line_split=line.split(',')
170                         if len(line_split)==3:
171                                 xpoints+=[float(line_split[0])]
172                                 ypoints+=[float(line_split[1])]
173         return xpoints,ypoints
174
175 def send_sms(s,target):
176         target=str(target)
177         if not len(target)==12:
178                 print "phone number incorrect lenght"
179                 return -1
180         if len(s)>160:
181                 print "text is too long"
182                 return -1
183         E=0
184         D=1
185         stringy=""
186         t=""
187         Bitstring=[0]
188         for G in s:
189                 Numb=ord(G)
190                 for M in range(7):
191                         if (Numb >> M) & 1:
192                                 Bitstring[E]+=D
193                         D*=2
194                         if D==256:
195                                 D=1
196                                 E=E+1
197                                 Bitstring.append(0)
198         print "SMS"
199         stringy="AT+CMGS="+str(14+(int(len(s)*7/8)+1))+"\r\n" #total lenght
200         #print stringy
201         ser_phone.write(stringy)
202         print ser_phone.readline()        #echo the command
203         ser_phone.readline()
204         print ser_phone.readline()        # the "> " is recieved
205         stringy="0011000C91"                # SMS submit, 12 digit international
206         for g in range(6):
207                 stringy+=(target[2*g+1])
208                 stringy+=(target[2*g])
209         stringy+="0000AA"                 #PDU string to Mobile, 4 day validity
210         stringy+="%.2X" % len(s)          #datalenght
211         for G in Bitstring:
212                 stringy+="%.2X" % G    #a load of HEX
213         stringy+=chr(26)                 #send it
214         ser_phone.write(stringy)
215         for n in range(4):
216          print ser_phone.readline()
217 #       print stringy
218         return 0
219
220 def get_smscen():
```

```
        ser_phone.write("AT+CSCA?\r\n")
        ser_phone.readline()
        s=ser_phone.readline()
        s=s.split('"')
        return str(s[1:2])[3:15]

def cutdown():
        ser_gr.write("Cutdown...   ")
        shutdown()
        os.system("echo 0 > /config/gpio/cuttertwo/enabled")    #payload release
        time.sleep(6)
        os.system("echo 1 > /config/gpio/cuttertwo/enabled") #and we are on the way down
        print "released"

def shutdown():
        HV_enable(0)              #all off
        Set_pressure_target(0)    #pump off
        time.sleep(1)             #wait, to avoid smoke contamination
        os.system("echo 0 > /config/gpio/cutterone/enabled")      #cut plunger
        time.sleep(3)
        os.system("echo 1 > /config/gpio/cutterone/enabled")
        print "plunger cut"
        ser_gr.write("Shutdown\r\n")

try:
        print 'AOPP aerosol experiment running'
        #log=open("/media/mmcblk0p1/"+logname,"a+")
        log=open("./"+logname,"a+")
        ser_gr=serial.Serial('/dev/ttyS2',4800, timeout=2, rtscts=1) #we have a CTS
        #line from radio
        #ser_gr=open("radio.txt","r")
        ser_daughter=serial.Serial('/dev/ttyS1',19200, timeout=2)
        #ser_daughter=open("daughter.txt","a+")
        ser_phone=serial.Serial('/dev/ttyS0',9600,timeout=4)
        #ser_phone=open("phone.txt","a+")
        print 'serial is open to gps,radio, and daughterboard'
        ser_gr.write("Hello world")
        #log.write("Hello world")
        reed_solomon.setup_rs()
        ser_gr.write(reed_solomon.encode_string("Hello,\
        I am a reed solomon encoded string :P"))
        #log.write(reed_solomon.encode_string("Hello,\
#       I am a reed solomon encoded string :P"))
#       while not gps_status():
#                ser_gr.write("waiting for the gps to lock")
                 #log.write("waiting for the gps to lock")
        print 'ok, gps is ready, we are at:'
        ser_gr.write("GPS locked")
        #log.write("GPS locked")
        print gps_data()
        print 'now probing daughterboard'
        print db_stats()
        print 'DANGER: turning on HV1'
        HV_enable(1)
        print 'HV1 on, probing board'
        time.sleep(1)
        print db_stats()
        print 'testing other HV channels'
        for x in [2,3]:
                HV_enable(x)
                time.sleep(1)
                print 'HV channel' + str(x) + 'stats:' + str(db_stats())
        HV_enable(0)
```

```
284         print 'ok, now testing the pump @ 20%'
285         Set_pressure_target(20)
286         for i in range(20):
287                 print db_stats()
288                 time.sleep(1)
289         Set_pressure_target(0)
290         print 'pressure set to zero'
291         for i in range(6):
292                 print db_stats()
293                 time.sleep(1)
294         print 'Testing done'
295         print 'opening KML cutdown file'
296         Jxpoints,Jypoints=load_kml('./cutdown.kml')
297         print Jxpoints,Jypoints
298         print 'testing phone'
299         ser_phone.write("AT\r\n")
300         print ser_phone.readline()
301         print ser_phone.readline()
302         #smscen=get_smscen()
303         send_sms("hello world",mynumber)
304         ser_daughter.flushInput()
305         ser_phone.flushInput()
306         system_vector=gps_data()
307         updatestuff.Predictedn=system_vector[1]
308         updatestuff.Predictede=system_vector[2]
309         updatestuff.Oldeast=system_vector[2]                    #start
310         updatestuff.Oldnorth=system_vector[1]
311         updatestuff.Oldaltitude=system_vector[3]
312         updatestuff.Oldseconds=system_vector[0]
313         count=0
314         calibrate=0
315         layercounter=0
316         cut_down=''
317         system_vector=gps_data()
318         maxaltitude=system_vector[3]
319         startuptime=system_vector[0]
320         HV=0
321         pump=FALSE
322         print 'ok launch the balloon'
323         while 1:
324                 print 'in loop'
325                 system_vector=gps_data()
326                 daughter_board=db_stats()
327                 datastring = ','.join(['%.0f' % system_vector[0]]+\
328                 ['%.6f' % system_vector[1]]+['%.6f' % system_vector\
329                 [2]]+['%.0f' % system_vector[3]]+['%.4f' % \
330                 updatestuff.Predictedn]+['%.4f' % updatestuff.Predictede])
331                 datastring+=str(daughter_board)+cut_down
332                 print datastring
333                 log.write(datastring+"\r\n")
334                 ser_gr.write(reed_solomon.encode_string(callsign+datastring))
335                 # ^ all our telemetery over the radio link
336                 #log.write(reed_solomon.encode_string(callsign+datastring))
337                 #t=reed_solomon.encode_string(callsign+datastring)
338                 #print t
339                 #print reed_solomon.decode_string(t,[])
340                 if count==2:
341                         send_sms(datastring,mynumber)      #every third time
342                         count=0
343                         os.system("sync")
344                 count+=1
345                 if calibrate==30:
346                         ser_daughter.write("K\r\n")  #recalibrates the pressure sensor
347                         calibrate=0
```

```
                    calibrate+=1
                    if cut_down=='':
                            if system_vector[3]/100>layercounter:  #100m altitude layers
                                    layercounter+=1                 #we move up a layer
                                    up=Updatepredict(*system_vector)
                                    log.write(str(up)+"\r\n")
                                    if not are_we_inside([0],up[1],up[0],[0]):#update and
                                    #check against the polygon
                                            print "geofence cutdown"
                                            log.write("geofence cutdown\r\n")
                                            cutdown()
                                            cut_down='CG'
                            if system_vector[0]-startuptime>max_flight_time:
                                    print "time cutdown"
                                    log.write("time cutdown\r\n")
                                    cutdown()
                                    cut_down='CT'
                            #if daughter_board[4]<battery_limit:
                            #       print "voltage cutdown"
                            #       log.write("voltage cutdown\r\n")
                            #       cutdown()
                            #       cut_down='CB'
                            if system_vector[3]-maxaltitude<-100:
                                    print "balloon popped"
                                    log.write("balloon popped\r\n")
                                    shutdown()
                                    cut_down='BP'
                    if system_vector[3]>maxaltitude:
                            maxaltitude=system_vector[3]
                    if system_vector[3]>limit[0] and pump==FALSE:
                            print 'turning on the pump (setting pressure target)'
                            HV_enable(1)
                            Set_pressure_target(default_target_pressure)
                            log.write("HV1 on\r\n")
                            pump=TRUE
                            HV=1
                    if system_vector[3]>limit[1] and HV==1:#set the right HV channel
                            HV_enable(2)
                            log.write("HV2 on\r\n")
                            HV=2
                    if system_vector[3]>limit[2] and HV==2:
                            HV_enable(3)
                            log.write("HV3 on\r\n")
                            HV=3
                    time.sleep(10)
                    iterations+=1                                   #10 seconds sleep

finally:
        log.close()
#       ser_phone.write("AT+CFUN=0\r\n")        #phone off
        ser_phone.close()
        ser_gr.close()
        ser_daughter.close()
        print "bye"
```