THEOS Project

Author: Phillip Mahoney

Partners: Lawrence Wilkinson
Alex Mackie
Philip Carpenter

Supervisor: Prof Vik Dhillon

2010/2011

Abstract

The THEOS project is a high altitude balloon project that was sent to the edge of space on the 08/04/11 to obtain useful meteorological data for under £700. The project consited of a 1600g balloon, a 4ft parachute and a stryofoam payload filled with an array of meterological equipment.

The payload reached a minimum altiutude of 36100m before bursting, in a flight time of $\approx$ 2hours 25minutes, with an average ascent rate of the around 5m/s and an average decent rate was 24.5m/s. The payload landed at 52.849565N, 0.893740W, and data for pressure, temperature, humidity of the atmosphere, as well as position and acceleration of the payload was recorded throughout the flight. The payload took good quality images and some HD video footage of the flight showing, the curvature of the Earth, as well as any turbulence in the atmopshere.

Simulations for the flight were created with a C++ programme with took into account a range of factors. The simulation results were comparable to other simulation software and the flight data, however the simulations accuracy could be improved upon.

All equations in this report have their parameters defined in the appendix. Any parameters that require their meaning to be explained will be done accordingly.

Contents

1 Introduction

High altitude balloon projects have been used for over a century. For a few hundred pounds, they are capable of reaching altitudes higher than any other method. They are invaluable projects for scientific research, contributing to everything from weather predictions, to the discovery of Cosmic rays, which was one of the most important discoveries for astronomy to date.

These days, high altitude ballooning has become a recreational activity, with balloons being launched every day around the world. The high altitude ballooning community are returning more and more extravagant achievements with time, as materials and technology become more accessible at lower costs.

For the To The Edge Of Space (THEOS) project, a high altitude balloon project, a vessel was designed and built to travel to the edge of space, over 30 km into the atmosphere. The aim of the project was to obtain meteorological data and compare it to current models of atmospheric parameter profiles of the atmosphere with respect to altitude.

The project consists of a team of 4 undergraduates and a supervisor. With a budget of £700, the project was designed to achieve the following criteria:

- A target payload altitude of over 30 kilometres
- Photos showing the curvature of the earth
- Temperature and pressure measurements as a function of altitude
- A means of tracking and recovering the payload safely
- Backup systems for tracking the payload.

Other goals for the project were set, including obtaining live data, humidity data, HD video and back-ups for all electrical systems, however these aims were not a priority, and were included only if the funding and time allowed.

The project was carried out in the style of a real world business proposal and was split into 2 sections. In the first section, the project group was split into two teams of two members. Each team was tasked with coming up with a vessel design and method to achieve the task brief. Both groups were given 3 months to come up with a vessel design proposal, produce projections of the vessel's flight and the results expected from the meteorological equipment. After 3 months the two groups would then come together to discuss which designs, construction plans and vessel components were the most suitable for the budget and task at hand. Negotiation for discounts was forbidden at this stage in the project to avoid any confusion for retailers being contacted by two groups working on the same project.

The second section of the project involved creating the vessel with the specifications finalised at the meeting. The complete design, production and thorough testing of the vessel had to be achieved prior to the defined launch window of the 28th of March to the 8th of April 2011.

Launch took take place at Surprise View, The Peak district on the 8th of April. The weather conditions were perfect for the flight and meteorological data was obtained for all the elements of the task brief. The payload was recovered safely a few kilometers from where it was predicted to land, resulting in the project's success.

2 Preparation and research

2.1 Research into prior group attempts

To understand the main elements involved with high altitude based projects, research into prior groups attempts was carried out prior to the project design. The main groups researched are outlined below:

2.1.1 UK High Altitude Society (UKHAS)[22]

This society provides a website that has a step by step guide for anybody wanting to send balloons to high altitudes for both scientific and recreational purposes. It covers everything from the basic physics involved in balloon flight, to technical details such as how to set up a GPS system from basic components. It also alerts people to the legality and safety issues of high altitude balloon projects, all of which have to be considered prior to launch.

2.1.2 Cambridge University Space Flight programme (CUSF)[10]

CUSF is a ten member team from the University of Cambridge which promotes high altitude balloon experiments over a range of different academic disciplines. The team has produced a useful simulation programme which can predict the path a high altitude balloon will take, based on the payload's average ascent rate, decent rate and GRIB wind data [8](wind data with respect to altitude in a gridded format). The website also has an optimum altitude calculator, which predicts the duration time of a balloon flight and the altitude a balloon will burst at.

2.1.3 ICARUS+HALO [17] [13]

The ICARUS and HALO projects follow a similar task brief to the one defined above. Both projects have had a number of launches and provide detailed diaries of events for every launch. This gives an indication to the factors that need to be considered when launching such a project.

2.2 Key factors to consider

From the research into high altitude projects, there were many topics raised including the method of filling the balloon and how to retrieve the project data once launched. The following key topics were the ones which needed immediate before the project could be taken any further:

2.2.1 Method of Lift

The majority of the high altitude projects researched, obtain lift by using sounding balloons filled with Helium. These sounding balloons are made from a Latex compound and are used as weather balloons on a day to day basis [3]. These sounding balloons are capable of reaching altitudes beyond the 30 km limit set by the task brief[3], making it the cheapest and most reliable way to provide lift. The size of the balloon and the amount of helium required depends on the weight of the main payload.

2.2.2 Tracking the payload and obtaining the data

There are two common ways of recovering data from a payload, the first of which is to transmit the data the payload obtains live during flight. This way, if the payload is lost, the data will still be safe, resulting in the project's success. The second method of data recovery is to store the data on-board the payload and recover the payload post-flight. This is a much less risky way to ensure all of the data is recovered, as long as the payload is recovered in the first place, because live transmission of data relies on having a constant clear line of sight to the payload throughout the flight [6]. This is a condition that is likely to be broken by a range of factors including bad weather, payload descending below a tree-line or the horizon, or any other unexpected

interference of the data signal. Because of this risk, the project used the on board data collection option in order to obtain data.

Recovering the payload requires some form of GPS tracking system, where co-ordinates can be transmitted to the user when the payload lands. This can be achieved in a variety of different ways. The most suitable methods will be discussed further in section 4.4 of this report.

The two main risks of storing the data onboard the payload are that the payload may not be recoverable and the instruments onboard may not be able to store data over the time of the flight in the harsh conditions of the upper atmosphere. Both of these risks were evaluated to be acceptable when the available methods of tracking, on-board data recording and recovery were researched and tested.

2.2.3 Insulation for the payload contents (Operation temperatures)

At high altitudes, the temperature of the atmosphere can drop below -50^oC [1]. These low temperature conditions affect the operation of most of the electrical equipment in these projects, especially in terms of reducing the battery life. As such, insulation of the payload needs to be considered. The researched projects claim that the internal temperature of their payload did not fall much below 273K [22], with a payload housing made from Styrofoam. As such, to save money and weight, whilst maintaining insulation, the material used for constructing the payload was Styrofoam.

2.2.4 Legality and safety.

The safety aspect of the project is a prime concern, as it potentially affects the general public and the aviation industry. Both the legality and the safety conditions of the vessel's flight must be fully evaluated before any launch can be planned. This affects the decision on size and design of any balloon or parachute system used.

As part of the legal requirements for the project, approval from the Civil Aviation Authority (CAA) [22] had to be obtained for launch between the 28th of March, and the 8th of April 2011 from Surprise View, the Peak District. This was to ensure there were no problems caused for air traffic, as the payload travelled through the 8-9km altitude region for a short period of time during flight; the altitude which commercial airliners fly at. The clearance requires notifying the relevant air traffic control regions and the local gliding clubs, both 24 hours and 30 minutes before the payload's flight to ensure the safety of everyone affected by the launch.

To evaluate the risks of the project fully, the CAA and air traffic control required simulations of the vessels path of flight with respect to time. These projections require a basic understanding of the physics behind the flight.

3 Theory

3.1 Atmospheric parameters and flight variables

To simulate the flight of the vessel and produce projections of the expected flight and meteorological data, the 4 main forces influencing the motion of the balloon must be well understood.

- Lift
- Drag
- Gravity

- Cross-winds acting on the payload

Any change in the atmospheric properties with altitude will affect at least 3 of the above forces. As such, a basic understanding of atmospheric parameters must be covered before any conclusions can be drawn about the balloon's flight.

3.1.1 Explanation of pressure and temperature behaviour of the atmosphere

For the purposes of this project, a simple empirical form for the temperature profile was used. The profile can be explained in terms of qualitative physics, but any in depth quantitative explanation is beyond the scope of this project.

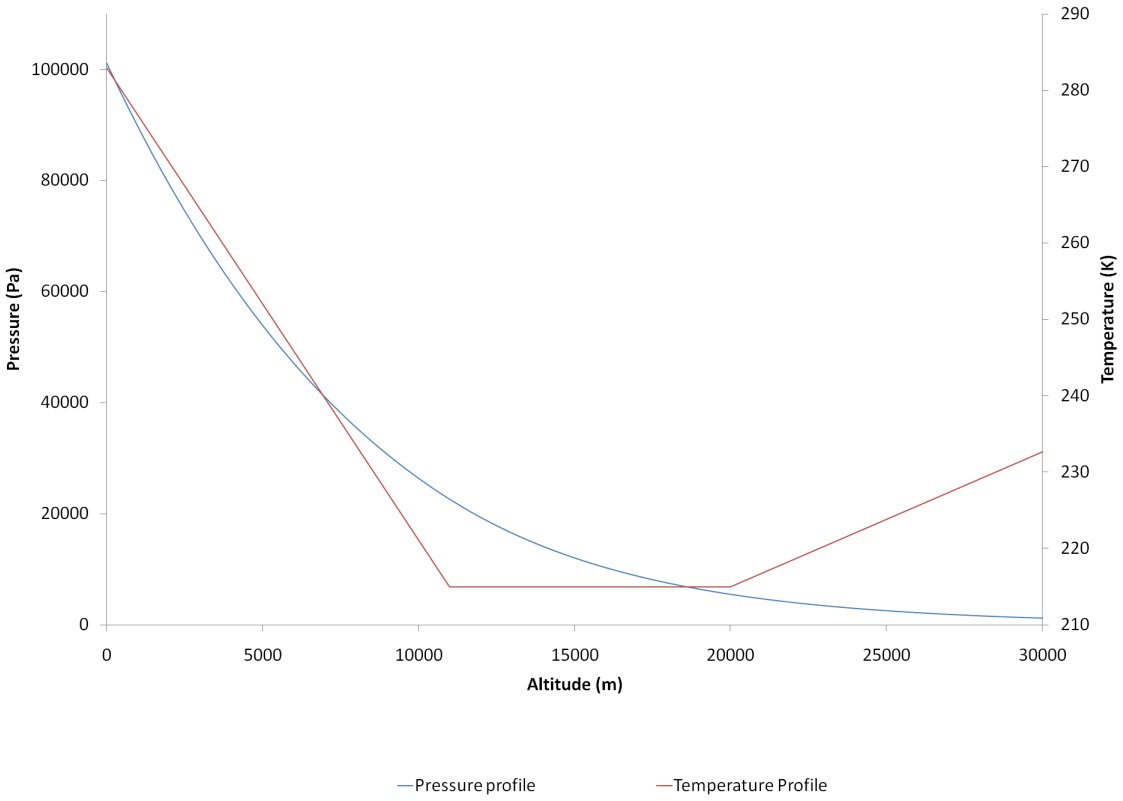

Figure 1: A projection of the temperature pressure profiles of the atmosphere with respect to altitude

Figure (1) shows the temperature and pressure profiles of the Earth's atmosphere [2]. An initial decrease in atmospheric temperature occurs between ground level and 11km of altitude [1]. This is defined as the Troposphere, one of the many layers that the Earth's atmosphere is comprised of. The decrease of the Troposphere's temperature with altitude is a consequence of the Tropospheric pressure gradient, described by equation (4) and the equations (1) and (2) [2] in the next section. The temperature and pressure gradients across the Troposphere provokes convective behavior in the atmosphere, making it turbulent [21]. This turbulence affects the vessel's flight path, so it was accounted for in the simulations in terms of the vessel's horizontal trajectory.

The Troposphere accounts for around 75% of the earths atmosphere, and extends to its boundary, the Tropopause, at around 11km at British latitude [1] [2]. The Tropopause marks the start of a temperature inversion in the atmosphere, which causes clouds to form. As such, the humidity of the Tropospheric layer is considered to be relatively high throughout the layer, depending on the weather conditions up to the Tropopause boundary.

The Tropopause extends from around 11-20 km above the Earths surface. Figure (1) shows the layer's isothermal nature, which comes from the equilibrium of the temperature between the Troposphere and the Stratosphere, which is described below. This layer's properties are very seasonal and position dependent, but it always acts as an atmospheric "lid" to the Troposphere, stoping any water vapour, and therefore humidity, from entering the next atmospheric level [21].

At around 20km and above, the temperature of the atmosphere begins to increase again. This is the region known as the Stratosphere. The source of the temperature increase is the absorption of UV radiation from the sun by gases such as ozone[1]. Due to the low particle densities in the Stratosphere, the temperature of the gas can increase dramatically, even for a limited amount of absorption. This layer extends to 50km in altitude and accounts for 10% of the total amount of the atmosphere. Due to the Tropopause layer, the Stratosphere is much dryer than the Troposphere [21].

The pressure profile varies much more smoothly than the temperature profile throughout the layers of the atmosphere. The profile decreases almost like an exponential decay with altitude, due to the decrease in gravity's hold on the atmosphere with altitude. However, the relationship describing the profile varies, depending if the layer of the atmosphere considered is isothermal or not. This is covered in mathematical detail in the next section[2].

3.1.2 Mathematical descriptions of the temperature and pressure profiles of the atmosphere

Simulating the above descriptions of the temperature and pressure profiles requires a mathematical description of the Earth's atmospheric properties. All parameters to all equations in this project report are summarised in section A of the appendix, but will be covered again in the relevant areas if they are of particular importance.

The pressure of the Earth's atmosphere can be described in terms of two expressions. These expressions depend on whether the layer of the atmosphere displays isothermal behavior or not [2] .

$$P(h) = P_b e^{-gM\frac{h-h_b}{RT_b}} \text{ : Isothermal case} \tag{1}$$

$$P(h) = P_b \left(\frac{T_b}{T_b + L_b(h-h_b)} \right)^{\frac{gM}{RL_b}} \text{ : Non-isothermal case} \tag{2}$$

The P_b, T_b, L_b and h_b terms correspond to pressure, temperature, lapse rate and height at the base of the atmospheric layer in question.

The lapse rate of the atmosphere is a term used to describe the temperature profile of the atmosphere [2] . As such, every layer in the atmosphere can have its temperature profile described by the following simple expression:

$$T = T_b + L_b h \tag{3}$$

3.1.3 Volume of the balloon

The lift of the payload is dependent on the volume of helium used in the balloon (see equation (5). The maximum volume the balloon is capable of defines the maximum altitude which the vessel can reach. Determining how the volume of the balloon depends on the temperature and pressure profiles of the atmosphere, allows the maximum alitiude of the payload can to be predicted.

The volume of the balloon at any given point in the vessels flight can be described by the ideal gas law:

$$PV = nRT \tag{4}$$

To simulate how the volume of the balloon changes with the vessel's flight, the following assumptions have to be made to link the atmosphere's known properties to the conditions in which the balloon exists:

- **The balloon ascends at a slow enough rate and the elasticity of the balloon is negligible such that the pressure inside the balloon is always equal to the pressure outside the balloon.**
  This assumption is a good approximation at ground level. When the balloon is filled, it will only be filled to provide just enough force to launch the balloon. As such, there will be negligible restriction on the balloon's volume at ground level from the elastic contraction force from the balloon's shell. At high altitudes this elastic constriction may play a small role, but it will be considered to be negligible for the purposes of this project.

  Due to the small force accelerating the payload, the slow ascent rate will allow the balloon's internal pressure to equalise with the external pressure throughout the ascent. As a result this is a good assumption to make about the pressure of the balloon.

- **The temperature inside the balloon is in thermal equilibrium with the temperature outside the balloon.**
  Due to the slow ascent rate of the balloon, we assume that the gas inside the balloon has time to reach thermal equilibrium with the surrounding temperature on the balloon's ascent. This assumption is not entirely accurate. However, on testing this assumption in simulations, by finding results when the balloon's internal temperature is constant, the effect of breaching the assumption is small. As such, this assumption is used in all simulations on the project's projections.

The basic parameters that effect the flight path of the balloon have now been covered enough to evaluate the forces that act on the payload.

3.2 Basics of flight simulation

The 4 forces outlined in the previous section are the main forces that influence the path the vessel took thoughout its flight. Simulating these forces requires a basic knowledge of the physics that drives them, over the conditions which the vessel will encounter

3.2.1 Buoyancy force

The lift from the vessel's balloon is described by a take on the Archimedes displacement principle [23]:

- **A body less dense than the fluid in which it is immersed will experience an up-thrust force equal to the mass of the fluid which the body has displaced.**

This means that the upward force the payload balloon exerts is equal to the following expression:

$$F_b = \rho_{Air} V_{Helium} g \tag{5}$$

Equation 5 can be manipulated with two forms of the ideal gas law described below:

$$P = \frac{kT_{Air}\rho_{Air}}{m_a} \text{ and equation 4} \tag{6}$$

By substituting equation (6) and (4) into equation (5) you get a useful expression for the buoyancy force, which is independent of the temperature and pressure around it. This remains true as long as assumptions outline in section 3.1.3 hold true. The expression for the balloon's lift used in all simulations is therefore:

$$F_b = \frac{m_a n R g}{k_b} \tag{7}$$

and the number of moles of helium used to provide this buoyancy force can be expressed by:

$$n = \frac{F_b k_b}{m_a R g} \tag{8}$$

The amount of helium required to lift the payload needs to be known so the cost of helium can be taken into account when designing the vessel.

3.2.2 Gravity

The basic Newtonian force of gravity is given by the following equation:

$$F_g = \frac{GMm}{r^2} \approx mg \tag{9}$$

To achieve lift, the buoyancy from the balloon must overcome this force. The greater the net upward force, the greater the balloon's average ascent rate will be.
It should be noted that the gravitational force acting on the payload and its components is proportional to the square of the distance the payload is from the centre of the Earth. As the payload moves further into the atmosphere, the weakened hold of gravity will cause a slow increase in the payload velocity on ascent. This effect is supported with the reduction of pressure and drag with respect to altitude, as described by equation(10).

3.2.3 Drag

The drag force is given by the following equation [4]:

$$F_{Drag} = \frac{P_{Air} m_a C_d A}{2 k_b T_{Air}} v^2 \tag{10}$$

where C_d is the drag co-effeicient. This coefficient takes into account the amount of drag force a body exerts due to its shape [4]. This is purely an experimental result, making it difficult to evaluate for this project. The drag coefficient for the vessel is taken to be 0.25 on its ascent, and 0.75 on it's descent [10]. The inverted profile of the parachute makes the drag coefficient much more dominant than that of a spherical balloon.

From equation (10), many of the parameters depend on the altitude and conditions the vessel is in. The result of this is that the drag force decreases with altitude, resulting in simulations showing an overall accellerating vessel during its acsent (see figure (5)).

3.2.4 Cross-wind forces

Cross-winds acting on the payload will not only define the path and landing location of the project, but will also determine how smooth the payload ride is, which is essential for obtaining good image evidence for the curvature of the Earth. Cross-winds are very dependent on altitude, due to the turbulent motions of the Earths atmosphere, as outlined in section 3.1.1.
The force that acts on the payload and balloon due to cross-winds can be evaluated in a similar way to the drag force with the following expression:

$$F_{wind} = \frac{P_{Air} m_a C_d A}{2 k_b T_{Air}} (v_{wind} - v)^2 \tag{11}$$

Equation (11) and equation (10) are very simaliar. The main difference between the two is that the velocity is a relative quantity in equation (11) with respect to the cross-winds. Simulating this force required access to wind velocity data, obtained from a source outlined in section 4.1.2.

Now a general understanding of the physics involved with high altitude balloon flights has been covered, simulations, design, construction and launch for the THEOS project can be undertaken.

4 Method

4.1 Simulations

Due to the inaccuracies and unreliability of using excel as a numerical simulation tool, a C++ programme was written to project all the data required for the project. This was a key component for the analysis of the project data. The criteria set for the simulation was as follows:

- To output pressure, temperature, payload velocity, balloon volume and wind velocities as a function of flight time.
- To be able to simulate the path the payload takes both in terms of altitude and North-South, and East-West directions.
- To take into account more physics than a stated ascent and descent rate for the payload's trajectory.
- To be versatile enough that any user can insert their own data and parameters in to the programme.
- To be able to upload a user friendly version of the programme to the project website, so that anybody wishing to complete a similar project can do so.

A brief overview of the programme algorithm and a copy of the final C++ programme source code is given in section E and G in the appendix respectively.
The programme was written as 2 sections which work together to produce an overall result. The first section outlines the general ascent and descent path of the vessel with time, whilst the second section uses wind data to determine the horizontal path of the vessel.

4.1.1 Simulations of the general ascent and descent of the payload

The C++ programme simulates the motion of the vessel, by evaluating the force acting on the payload over small increments in space. The programme makes the assumption that the accelerating force of the payload is constant over these small increments. An increment step size of 0.1m was used in all the simulation results in this report, however the programme allows the user to choose which ever step size they require. The smaller the step size, the more accurate the simulation becomes, but the longer the run time is for the simulation. The programme can simulate the trajectory of the payload during its flight and result return

the projected meteorological data required for analysis.

The programme was designed so that the user has a choice of 3 options to run:

- The first choice is to run the programme for predefined parameters from the day of our project flight. These parameters are stored in the source code of the programme, to make the project's simulation quicker when evaluating the results.

- The second choice is to run the algorithm used in choice 1 with data input by any user. Both the above 1st and 2nd choices use the standard temperature and pressure profiles described in section 3.1, however the second choice allows some of the key parameters to be chosen for the running of the programme.

- The third choice allows the programme to take pressure and temperature data from an official prediction source such as NOAA (see section 4.1.2). This means that data for the day's temperature, pressure and wind as a function of altitude can be put into the simulation to return projections on the payloads trajectory and meteorological data.

The programme was written so that every defined number of steps in altitude, data is recorded in a file for analysis. This value is either preset at 100 for choice 1 or definable by the user for choice 2 and 3. This feature saves running time for the programme. Whilst outputting a smaller number of data points to plot, simulation accuracy is not compromised.

For the programme to return any simulation data, it requires information regarding the wind velocity with respect to altitude.

4.1.2 Simulations for the total flight path: GRIB data and NOMADS data

To simulate the path of the vessel during the flight, a source of data for wind velocity with respect to altitude is needed. Wind velocity as a function of altitude is published in a GRIB format, which can be freely obtained through the National Oceanic and Atmospheric Administration (NOAA) NOMADS programme [20]. NOMADS, is an American publically owned weather simulation site, which provides free wind, temperature and pressure data for the entire world [20]. This is the site which CUSF get their data from.

The NOMADS site provides wind, geopotential altitude and temperature data as a function of pressure, over 26 pressure levels with two levels of resolution, 0.5^o or 1^o [20]. The data can be accessed by following the set of instructions in section C in the appendix.

The C++ programme uses the data from NOMADS by smoothing it over the 26 pressure levels. This provides a more continuous data stream where the data varies with altitude more smoothly than given in the NOMADS format.

The NOMADS data only extends up to an altitude of $\approx$ 30km, thus data had to be extrapolated further upwards to simulate the projections for the project. The method in which this data is extended beyond $\approx$ 30km is dependent on the dataset being manipulated. For example, pressure reaches a minimum at high altitudes, whilst temperature increases throughout the Stratosphere with altitude.

All the key aspects required for the simulations have now been covered, except for the project mass. To evaluate this, the project components, must be known.

4.2 Project Layout

The project was laid out in the format shown in figure (2). Figure (2) shows the payload at around 2 meters from the parachute. The parachute was pre-deployed so when the balloon exploded, the parachute opened immediately on descent. The balloon was attached to the top of the parachute canvas at around 5 meters from the parachute to ensure maximum stability of the payload during its flight. Each component was tethered together by nylon cord due to its light weight, strong properties. Buzzers and flashing LEDs were fitted to the payload to alert anybody of the payload approaching, and assist in finding the payload on landing.

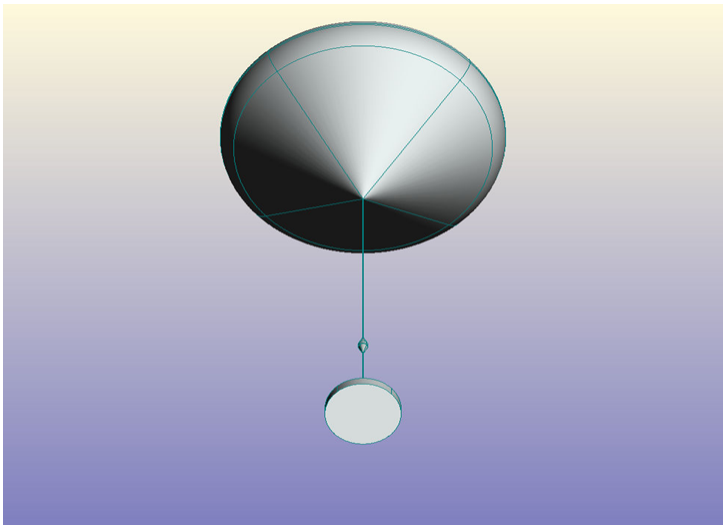

Figure 2: This shows the general layout to the project planned of how the project was layed out. A payload was attached to the base of a parachute, which was then subsequently attached to the balloon from the top of the parachute.

4.3 Project Components: External components

4.3.1 Balloon and parachute

The mass of the payload was estimated to be 1.6kg, meaning that the balloon and parachute with the required properties for a successful project could be selected appropriately.

Two sounding balloon's were bought from the company Hwoyee [14]. The company provided a deal on two of the 1600g sounding balloons, allowing for 2 potential launches. These balloons are capable of achieving altitudes in excess of the 30km target set by the task brief and lifting masses in excess of that of the payload [22]. Sounding balloons are designed to completely shred when they burst, leaving very little mass attached to the payload. This helps in choosing a suitable parachute to achieve a safe payload descent rate.

A 4 ft parachute was bought to handle the weight of the payload. The parachute comes with a loop at the top of the canvas, which the balloon could be tethered to, with nylon cord. The dimensions of the parachute were chosen from the simulation data to return a safe descent rate.

The payload's parachute was tested by launching it off the top of the Hicks building, approximately 20 meters above the ground. The payload was attached to the parachute with nylon cord, and filled with weights equaling the mass value of the contents required for the actual flight. The payload was dropped directly over the edge of the roof and fell for approximately 2 seconds, a descent rate of 10ms^{-1}. This was a higher descent

rate than was expected, and very close to that of a free fall descent. However, the parachute did not have enough time to fully respond to the payload's descent for the entire flight, resulting in the payload being in free fall for most of the descent. The payload received no damage during the test, making us confident that the housing would be able to withstand an actual flight.

4.3.2 Helium

Helium was available from the main helium supplier for the Physics department at the University of Sheffield. The price required for a suitable canister of helium was around £80. However, the department runs a buy back scheme, where the department will buy back any of the helium which wasn't used. This reduced the effective cost of the required helium per flight to be £25, equivalent to just over $3m^3$.

Filling the helium balloon incorporated a cost of a gas adapter. An adapter was made within the physics department for a negligible cost after the purchasing the Totex balloon.

4.4 Project Components: Internal components

The internal measuring components were bought and thouroughly tested prior to the launch. A test box was constructed from Styrofoam, allowing the equipment to be run through cold conditions inside the insulated box. Each component was tested for the following to maximise the probability of obtaining useful data from the flight:

- Cold temperature: To ensure that any data measurements would not affected by the cold conditions of the upper atmosphere.

- Length of operation: To see if the cold and harsh conditions affected the battery life of the equipment in any way.

- Operation through the payload casing: To see if instruments that require external signals to operate could do so through the payload housing.

The properties and functions of the internal components of the payload used for the flight are outlined below:

4.4.1 MSR145[19]

The MSR145 is a small lightweight unit that measures and stores data for temperature, pressure, humidity and acceleration in 3 axes . The unit can be programmed to take all of these measurements over a set time span, or continuously until interrupted by a user. The unit also has a rechargeable power supply and on-board memory which can last from days to weeks depending on the unit's usage. The software used to extract the unit's data has a unique feature for telling the user how much battery and memory life the unit has depending on the usage.

The MSR145 was bought at a discounted price of £141.64 in return for publicity of the company.

4.4.2 Lascar temp logger +Lascar humidity logger [9]

Lascar electronics provided a temperature and a combined temperature and humidity logger for £31.30 to measure the temperature and humidity of the atmosphere [16]. The units are self contained, and download

data through a built in USB adapter. The units were tested thoroughly for battery lifetime and operation temperature. Some free, spare lithium $\frac{1}{2}$ AA batteries were included with the purchase, so both units could be tested at no extra cost.
The temperature logger came with some free, spare thermocouples, capable of operating at the temperatures expected at high altitudes. The thermocouples allow the temperature logger to measure the temperature outside the payload box, without the logger itself being subjected to the harsh conditions of the upper atmosphere.

The humidity logger does not have the option of fitting external probes for data recording. The data is recorded by a probe fitted to the end of the unit's housing. As such, a hole was made for the unit so that the data probe can stick out of the payload into the atmosphere during the payload's flight.

4.4.3 Canon A430 camera [5]

The A430 Canon camera is capable of taking high enough quality images required by the brief. The motivation behind choosing this particular model, other than the low price of £64, was due to its compatibility with CHDK firmware.

CHDK (Canon Hack Development Kit) is a piece of free firmware, which consists of a script that can be simply transferred onto the camera's memory card [18]. This script allows the user to change the settings of how the camera takes its images. The firmware can set the camera so that it takes an image over a variety of time intervals and set the camera to standby mode between images to conserve battery life. The simple nature of this script and the value for money for the camera makes this the method to provide images showing evidence for the curvature of the Earth.

The camera was setup with the firmware script to take images every 10 seconds. The flash on the camera was permanently turned off and the LCD screen on the camera only switched on every time an image was taken. These settings were chosen to save the camera's battery life, and allow anyone to check if the camera was taking images during testing.

4.4.4 Tachyon HD video camera [15]

The Tachyon video camera was obtained for free in return for publicity and a review of the product. The product itself is a prototype compact video camcorder capable of 6 hours of battery life as well as being water resistant. The unit was tested for its recording lifetime and its sound recording quality. The image quality and lifetime of the unit is excellent, but the sound recording quality is not. As such the MSR145 accelerometer data must be used to determine when the balloon explodes.

4.4.5 Xexun

The Xexun unit is a small and lightweight unit available for £64.03, which was used for finding the location of the payload once it landed. The unit sends its own GPS co-ordinates in a text to anyone who calls it. The unit came with a spare rechargeable battery and an O2 sim card to send a text reply to the user. O2 were chosen due to their high signal coverage of the UK. [7]
The unit is not likely to receive a signal above a few kilometres due to how mobile signals are broad-casted. The main concern with the unit was whether it would work again after the payload descends from its maximum altitude. The unit was tested in conditions similar to these and worked. The battery life of the unit lasted over several days, making it suitable for the project flight.

4.4.6 Iphone-Viewranger app[24]

The I-phone was obtained as a donation from a member of staff in the department. It supports a GPS tracker application called view-ranger. This acted as an ideal backup for all the position recording equipment, both in terms of data logging and payload retrieval. The application stores and sends its GPS co-ordinates to an internet application which can be accessed by the user during the payload flight .
The main flaw in using the I-phone is that it requires a mobile signal to operate. The signal becomes too weak to detect above a few kilometres in altitude, resulting in potential gaps in the data recorded. The network chosen for the I-phone was Vodafone due to the good network coverage and low running costs.

4.4.7 Garmin e-Trex H GPS logger [11]

The legal limitations of civil usage of GPS loggers, narrowed the choice of product down to the Garmin based GPS loggers [22].
The Garmin e-Trex H GPS logger was one of the cheapest GPS loggers on the market. This unit has been used on similar previous projects by other groups, which promoted confidence in the units operation for the brief. The unit has a standard battery life of 17 hours and logs latitude, longitude and altitude from GPS triangulation. It also comes in a protective waterproof casing and advertises the ability to work through forest canopy cover, good properties for the task brief. The logger did not come with a computer interface cable or any interpretation software. These could be purchased for a further £25-£30. Instead, a homemade interface cable was fabricated, from researching other projects which have encountered this problem [**?**].

- Interface hardware

One the back of the Garmin e-Trex H, there is a 4 pin socket which their custom cable can interface with. This then connects to a computer via an old RS232 connector. By identifying the role of each pin on the Garmin device, an interface cable can be made from a basic 4- core cable and a female RS232 connector (obtained for free from the physics department). Each pin of the RS232 connector was soldered to the corresponding cable core, which was then attached to the GPS unit by fabricating an interface from a small piece of PVC. This is shown in figure (3) [**?**].

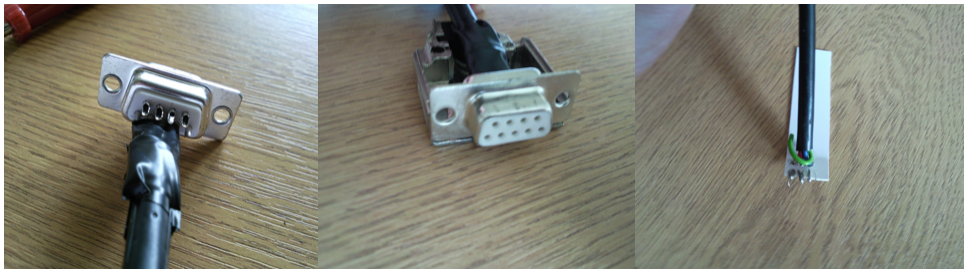

Figure 3: The interface cable used to retrieved the data from the Garmin e-Trex H. The cable in this image is stripped down to show the core cable structure.

To test the cable, the GPS logger was connected to a computer, with an RS232-USB converter, also available for free within the department. By connecting the unit with a hyper window on a PC, the success of the connection was determined. The hyper window outputted a data string in the GPS's default output mode every second. Each string had a number that incremented by 1 every second, showing the unit was outputting a time variable. This confirmed the interface was working on a basic level. To retrieve the data from the unit, information was extracted using third party software.

- Software interface

The Software used to interface with the GPS unit was Easy GPS [12]. This software is free, downloadable, and can interface with a wide range of GPS loggers. The GPS data, in tabulated form, for latitude, longitude and altitude could then be extracted from the Garmin unit with the program, allowing comparisons between simulated and actual data to be drawn.

4.5 Payload design

The payload was designed and constructed in two sections. The first section was a simple box based around the payload components outlined above. It was made from a combination of Styrofoam and light weight sponge, to keep the payload mass down.

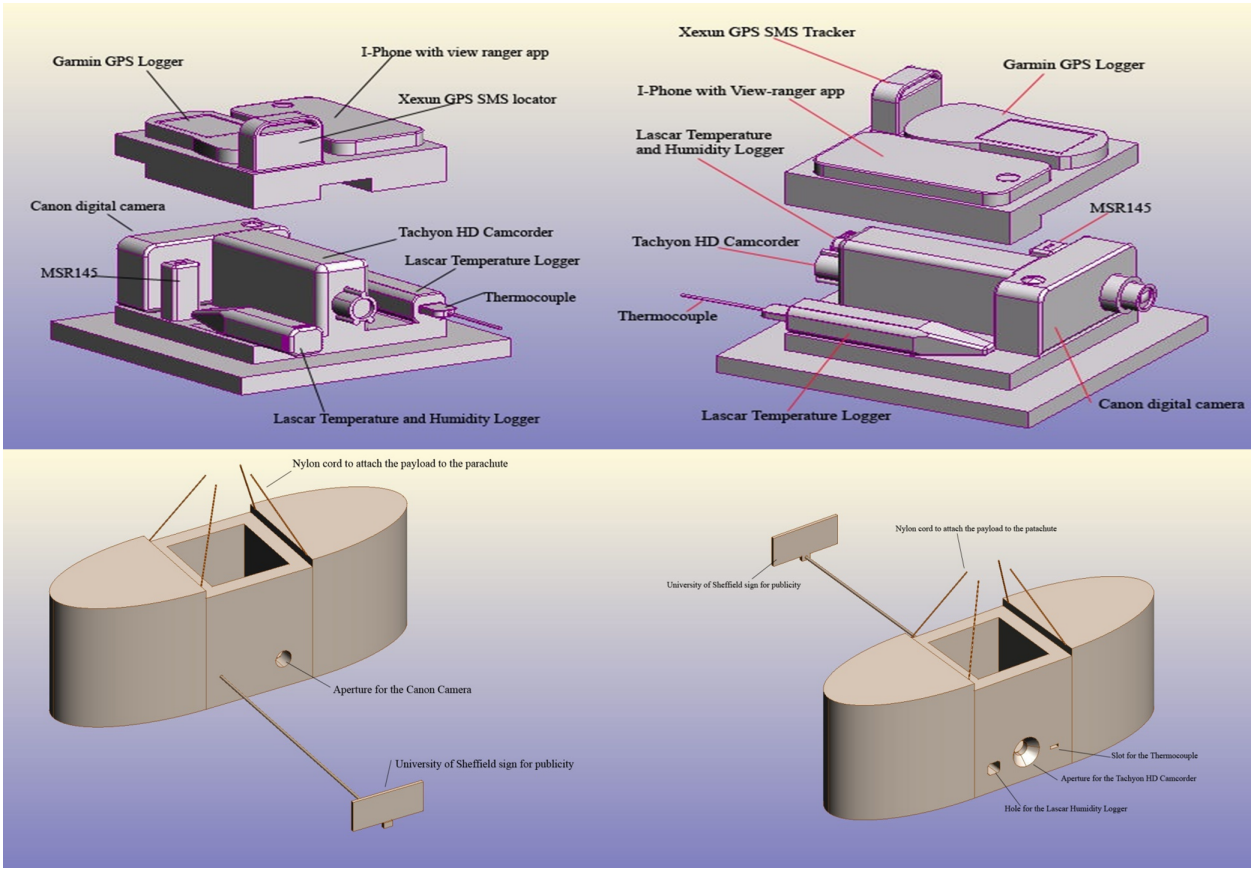

Figure 4: An anotated overview of both the inner sections (upper figures) and outer shell (lower figures) of the payload.

Figure (4) shows the design of the external payload shell and a semi-exploded view of the payload box contents. The top layer and outer walls are moved out of the way, so the payload contents can be viewed. A scaled schematic of both the designs are shown in section D of the appendix.

The internal payload box design was based on making as many of the components as packed together as possible, so weight of any unnecessary insulating material was lost. The general operation of the components also heated the box, thus maintaining a safe operating temperature inside the payload[22].

The box was split over 2 levels. The first level houses all the image and meteorological equipment. The second level houses everything that requires access to external transmitted signals. This layout allows the GPS units to have their receiving antennae near the surface of the payload housing to improve the signal strength [22].

The second part of the payload design is concerned with the overall shape of the payload housing and the method in which the payload is tethered to the parachute. The design of the payload shape came from the advice of a qualified aerospace engineer to minimise spin, and therefore reduce any blurry images taken by the camera. The design is shown at the bottom of figure (4).
Adding a tail to one end of the payload, will decrease the chance of spin even further, but will add weight. If excessive spin had been observed during testing of the payload, then a tail would have been considered.

5 Project launch

The project was planned for the 7th and 8th of April from the Surprise View launch site. The university was contacted on both days, in an attempt to increase the publicity and awareness of high altitude balloon based projects. An overview of the launch attempts are described below.

5.1 Launch 1: 07/04/11

The first launch was aborted due to the lack of communication between air traffic control and the civil aviation authority. A flights worth of helium and a balloon were lost due to this error, however the preparations helped refine the filling strategy for the balloon and served as a trial run to ensure there were no problems with the second launch.

The filling process of the balloon was a delicate task. The balloon had to be handled with Latex gloves to ensure no perforations were made in the thin balloon shell. Once the balloon was filled enough to take its own weight, it was attached to a mass equaling that of the payload. This way the balloon could be filled with the minimum amount of helium required to launch the payload. This volume of helium is less than the stated ground volume of the balloon. This meant that the helium had extra volume to expand into at high altitudes, thus allowing the balloon to explode at higher altitudes than the altitude stated by the balloon manufacturers [10] .

Once the balloon was filled, it was sealed with a combination of duct tape and cable ties at the balloon's neck. The payload contents were then activated and the box was sealed. The payload was then tethered to the parachute and balloon, with 2mm nylon cord with the rough dimensions outlined in section (3) and was ready for launch. It was at this point when the flight plan was rejected from air traffic control, and the project had to be dismantled. The balloon was burst on deflation, resulting in only one opportunity for the project launch.

5.2 Launch 2: 08/04/11

The second launch attempt was a success, with no problems with air traffic control or weather conditions. The same preparation process was used as the previous day, however, clearance for launch from air traffic control was confirmed prior to leaving for the launch site.

Launch commenced at 11:10 and the payload was retrieved from the GPS co-ordinates 52.849565N, 0.893740W, transmitted from Xexun at 13.39. This transpired to be a small field 8km north of Melton Mowbray. The vessel was spotted just south of Nottingham shortly before exploding at 13:05, at over 36km into the atmo-

sphere. The flight lasted 2 hours and 25 minutes and images of the edge of space can be found in the results and analysis.

6 Results and Analysis

The results obtained from simulations and the meteorological measuring equipment is shown in figure (5). The results are plotted over 4 separate graphs all with respect to the time of flight for the vessel. This allows comparisons to easily be drawn between the data simulated and measured.

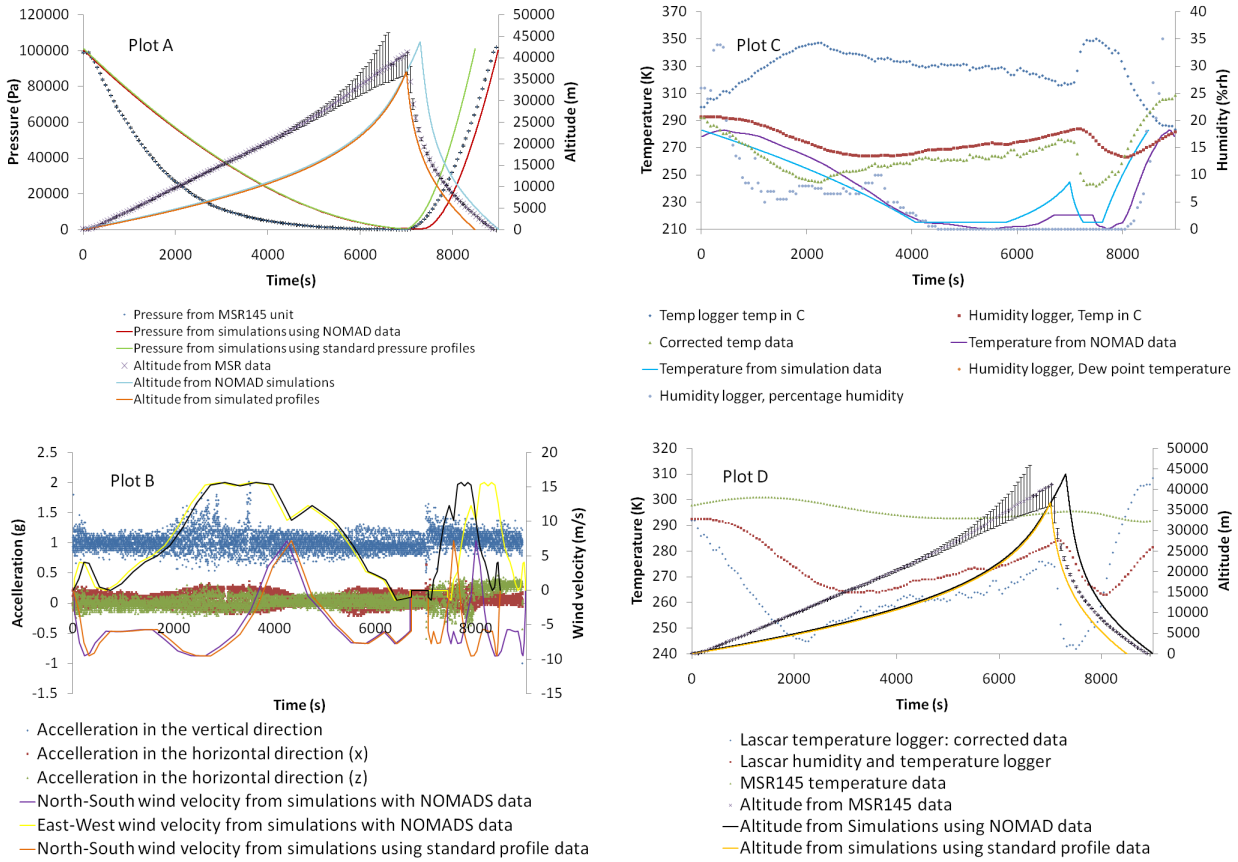

Figure 5: This shows the results of simulations and data recorded for the flight. The figure is split into 4 plots, A, B, C and D which compare relevant data profiles to each other. All the data is plotted with respect to time so all the plots can be cross referenced

6.1 Pressure data

The MSR145 successfully returned the pressure data for the flight shown in figure (5). Plot A shows the MSR pressure data, the standard pressure profile data used in the simulations and the pressure data from the NOMADS website. By converting all the pressure profiles into altitude profiles from equation (1) and (2), the ascent and descent path of the vessel can be analysed over all 3 data sets.
The altitude profiles with respect to time are remarkably different for the MSR145 and the simulation cases. Plot A of figure(5) suggests that the payload ascended quicker than simulations. This behavior may be due to the underestimation of the amount of bouyancy in the balloon. This theory is supported for the standard

profile simulation, which predicts a higher burst altitude than that derived from the MSR145 pressure data.

The simulations suggest that the balloon should accelerate the higher the balloon travels, however, the MSR145 data demonstrates a steady ascent. This may be attributed to the fact that the simulations do not consider the fluctuations in pressure and veritical winds due to turnbulance. The MSR145 was also sealed within the payload; although the payload was not air tight, the flow of air inside the payload was obstructed, in order to maintain the payload's insulation. This could mask, or even removed, the average vertical acceleration behaviour of the payload.

The lowest pressure value measured on the MSR145 was 2.4mb, which from equations (1) and (2) corresponds to an altitude of 41000m. The unit has a measuring uncertainty of $\pm 2.5mb$, meaning the maximum pressure the balloon bursts at is 4.9mb. This pressure provides a lower limit on the vessel's acheived altitude of 36100m.

The upper altitude limit from the MSR145 pressure data is unphysical. This is because the lowest pressure readings with the associated error is less than or equal to zero. As such the unphysical error bars have been removed from the data points in figure (5) to highlight the problem. From the international standard atmosphere, pressure beyond 47km in altitude is less than 8mb [2]. This pressure will cause the balloon to breach the burst volume radius of 4.72m($\approx 8mb$), putting a physical upper limit on the vessel's maximum altitude.

The time the payload reaches its maximum height and the Tropopause can be confirmed by comparing the MSR145 accelerometer data to the altitude profile from the MSR145 pressure data. This is shown in plot B in figure (5).

At $\approx$ 2100s, there is a large amount of variation in acceleration in all axes. This period of varied acceleration corresponds to a height region of $\approx$10000m-15000m. Comparing the altitude and accelleration results to the NOMADS wind-velocity data, this acceleration profile agrees with the wind velocities represented in plot B, figure (5). The drop in high wind velocities mark the end of the Tropopause, as descibed in section 3.1.1.

A sharp drop in the vertical acceleration is also observed at $\approx 7100s$. This is the time at which the pressure data defines a maximum in the payload's altitude. As such the acceleration confirms the point in time when the balloon explodes, which agrees with both the simulation and NOMADS profile based simulations.

6.2 Temperature data

6.2.1 External temperature data.

The Lascar temperature logger obtained data for the flight showing exactly the opposite of what was expected. This is shown in plot C, figure (5).
By plotting this data against the humidity logger temperature , it can be seen that the failed temperature logger outputs a temperature profile that mirrors that of the humidity logger. By transforming the flawed temperature data vertically so the start temperature was 19^o C (confirmed by data from both the humidity and MSR145 loggers prior to the flight) and by reflecting data in the line T=19^oC, a corrected version of the temperature profile was achieved.

The validity of the corrected data cannot be fully trusted, as there may be some form of systematic error in the data manipulation. This is emphasised in plot C, figure (5), where the recorded temperature from the Lascar temperature logger, the NOMADS data for the temperature, and the temperature from a simulated profile are shown. The Lascar temperature value does not agree well with the other datasets, meaning there is more than likely some error associated with the corrected data. However, the overall temperature profile

shape agrees with the simulation results, and the different temperature behaviors of the different levels of atmosphere are clearly present in the plot.

6.2.2 Internal temperature conditions of the payload

The MSR145 monitored the internal temperature of the payload. Plot D in figure (5) shows the MSR145 temperature data against both the Lascar temperature results. The effectiveness of the payload's insulation is shown, with the data curves showing a respectable decline in the temperature stability, the further away the recording apparatus is place from the payload box itself.

The temperature inside the box never dropped below 291.6 $\pm$0.01K, or raises above 300.9$\pm$0.01K during the flight, a good operating temperature range of all the units inside the payload.

6.3 Location data

6.3.1 Garmin e-Trex GPS unit

The Garmin GPS logger failed to take any readings during the payload flight. The unit took readings when the payload was sealed up. However, it stopped collecting data shortly after launch. The reason for this is unclear, as the unit did not experience any conditions it had not already been tested for. As such, all latitude and longitude data must be retrieved from the I-phone backup unit, and altitude data must be recovered from pressure-altitude relations from the MSR145.

6.3.2 I-phone data

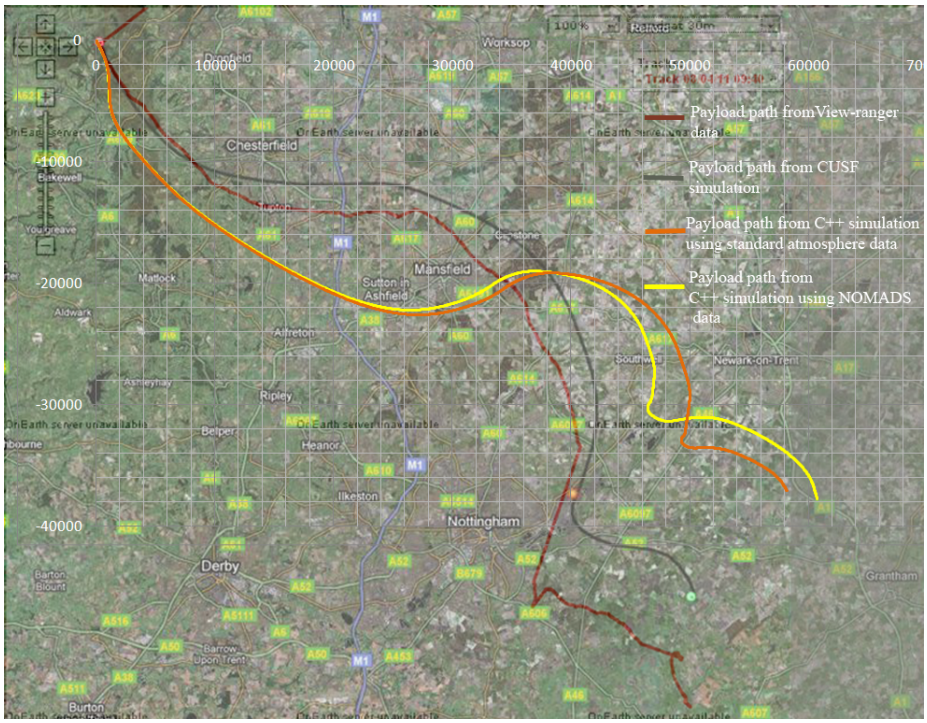

Figure 6: This is the results of the actual flight path from View-ranger data compared to simulations. The thin red line represents the I-phone data, whilst the black line is the simulation according to the Cambridge simulator. The orange and yellow lines are the simulation paths returned from the designed C++ programme corresponding to NOMADS and standard atmospheric profile data respectively

The I-phone lost its signal a few times throughout the flight . However a relatively successful recording of the payload's track was obtained and shown in figure (6).

The payload path(red line) followed the Cambridge simulation path (black line) quite accurately, however the simulation results from the C++programme were not so accurate (orange and yellow lines). Assuming this result is not due to poor statistics, the inaccuracy of the programme could be attributed to the following factors.

- Firstly, the C++ programme does not take into account the change in average molecular weight with respect to altitude. From equation 10, the horizontal force on the payload is proportional to this factor and therefore will affect the horizontal force- altitude profile acting on the payload.

- Secondly the altitude profile from simulations of the payload flight, as seen in figure(5), shows an accelerating profile with the payload altitude. However the MSR145 unit indicates a constant ascent velocity over time. If a constant ascent rate was used, then the vessel would be subjected to higher altitudes, and therefore different wind velocities for different lengths of time. This will alter the vessel's simulated trajectory so that it would presumabley follow the actual flight path better.

- Finally the wind velocity at each level was evaluated so that the wind speeds stated by NOMADS are represented as a smoothed profile as shown in plot B, figure (5). Better simulations require more information about how wind speeds vary with altitude, or more wind data with respect to altitude.

The altitude data from the I-phone is unreliable due to the data loss throughout the payload flight. As such the altitude data for the vessel relies entirely on the MSR145.

6.4 Humidity data

The humidity data for the flight is plotted in Plot C in figure (5). From section 3.1.1, the results returned are what was expected for the humidity profile of the atmosphere. The humidity generally decreases with altitude in the first 5km of ascent. This is just due to the weather conditions of the day. After the first 5 km of altitude, the humidity remained relatively constant up to an altitude of around 16.5km. This is approximately the altitude in which the Tropopause exists, and thus defines the boundary at which water vapour can evaporate to. At 25km, the humidity drops to 0 percent. This is again expected as the start of the Stratosphere exists at this altitude. The reverse relationship is observed upon the payload's descent.

6.5 Images

The Canon camera successfully took images of the horizon, every 10 seconds, for the entire flight, with very little image quality lost due to the movement of the payload. As the payload reached its maximum altitude, the curvature of the earth became very apparent, as shown in figure (7).

Figure 7: An image taken of the Earth near the maximum altitude of the payload

The Tachyon HD video camera took video footage of the entire flight of the payload. However, the lens

condensed over no more than 20 minutes into the flight, thus obscuring any further footage. This may have been avoided if the camera was set up and sealed in a dehumidified room a few days prior to the launch.
By comparing the dew point temperature data from the humidity logger, to the external temperature data in plot B in figure(5), the temperature comes into close proximity to the dew point temperature at $\approx 2000s$. The dew point temperature is the temperature at which water condenses out of air, which may explain the fogging of the HD camera lens near the start of the flight.

7 Conclusion

The flight of the payload was mostly successful, and it acheived all of the goals set by the task brief. Only one of the two planned launches were a success, resulting in the payload reaching a minimum altitude of 36100m. Temperature, humidity, and pressure profiles were also recorded, throughout the flight, which after data reduction, mirrored the simulations. The external temperature recorded for the project returned systematically affected data, which was corrected as much as possible for the data analysis.
The GPS co-ordinates of the payload throughout the flight were not recorded by the primary GPS data logger. The secondary GPS logger recorded some data, however data is missing for the higher altitudes of the flight. This Meant the height of the payload hadto be infered from the pressure data and equations (1) and (2

Pressure measurements were taken by the MSR145. The unit has an error of 2.5mb associated with it, resulting in the maximum altitude for the payload being inferred from standard atmosphere measurements. This returns a maximum altitude of 47km.

From the MSR145, the tempertature inside the payload did not drop below 291.6 $\pm$0.01K. This is a perfectly reasonable operating temperature for the payload's electrical equipment, meaning that the insulating material used for the payload's construction was the perfect choice.

The simulation programme written for the project, took into account most factors that affect the measurements and the trajectory of the payload. The results did not fit the GPS data as well as the Cambridge simulation. This may be a consequence of poor statistics, or due to inaccuracies in the designed C++ programme.
The design, construction and testing of all the components yielded all positive results, and the simulation programme was successfully written so it could be used by anybody wishing to undertake a similar project. All activities and simulations in the project are documented on the project website.
In conclusion, the project was a success on the whole and satisfied the task brief. If more time and funding was available, improvements could be made on the simulation programme, the method of tracking, and the method of obtaining data. The possibility of controlling the payload during some part of the flight would also go towards reducing the risks of the project failing.

8 Acknowledgements

I would like to thank my project partners Lawrence Wilkinson, Philip Carpenter and Alex Mackie for their sterling efforts in the project. Without them, this report would tell a much more unsuccessful story.

References

[1] C. Donald Ahrens. *Meteorology today: an introduction to weather, climate, and the environment.* Thomson Brooks/ cole, 8 edition.

[2] U.S. Airforce. U.s. standard atmopshere 1976. *National Aeronautics space administration*, 1976.

[3] Kaymont Balloons. Supplier of totex balloons. http://www.kaymont.com/, Novemeber 2010.

[4] George Batchelor. *An introduction to fluid dynamics*. Cambridge university press, Cambridge New York, 2000.

[5] Canon. A430 canon camera. http://www.canon.co.uk/for_home/product_finder/cameras/digital_camera/powershot/PowerShot_A430/, January 2011.

[6] COAA. Sondemonitor software and general operation of radiosondes. http://www.coaa.co.uk/sondemonitor.htm, Novemeber 2010.

[7] Uk Mobile coverage calculator. Uk mobile signal coverage. http://ukmobilecoverage.co.uk/, March 2011.

[8] US GRIB data website. Grib us. http://www.grib.us/, February 2011.

[9] Lascar Electronics. Lascar electronics. http://www.lascarelectronics.com, February 2011.

[10] Cambridge University Space flight. Society web page. http://www.srcf.ucam.org/cuspaceflight/, Semptemer 2010.

[11] Garmin. Garmin e-trex h gps specifications. https://buy.garmin.com/shop/shop.do?cID=144&pID=8705&ra=true, December 2010.

[12] Easy GPS. Garmin gps data recovery software. http://www.easygps.com/, March 2011.

[13] Robert Harrison. Icarus project. http://www.robertharrison.org/icarus/wordpress/about/, October 2010.

[14] Hwoyee. Hwoyee sounding balloons. http://www.hwoyee.com/, March 2011.

[15] Tachyon. Inc. Tachyon hd helmet cameras. http://www.tachyoninc.com/, March 2011.

[16] Dataq instruments. Dataq instruments, supplier of lascar products. http://www.dataq.com/, March 2011.

[17] Alexei Karpenko. Halo project. http://www.natrium42.com/halo/flight2/, October 2011.

[18] Canon Hack Development Kit. Firmware to adjust camera settings. http://chdk.wikia.com/wiki/CHDK, November 2010.

[19] MSR. Producer of the msr145, November 2010.

[20] National Oceanic and Atmospheric Administration. Main webpage. http://www.noaa.gov/, March 2011.

[21] Richard J Chorely Roger G Barry. *Atmosphere, Weather and Climate*. Routledge, 9 edition.

[22] UK High Altitude Society. Wikibooks on all aspects of high altitude ballooning. http://wiki.ukhas.org.uk/, October 2010.

[23] T.L.Heath. *The works of Archimedes*. Number 256. Cambridge University Press.

[24] Viewranger. I-phone gps tracking system. http://www.viewranger.com/, November 2010.

Appendices

A Glossary of equation parameters

Parameter	Definition
P(h)	Pressure with respect to altitude
P_b	Pressure at the base of the atmospheric layer
T_b	Temperature at the base of the atmospheric layer
L_b	Lapse rate of the respective atmospheric layer
h	Altitude of vessel
g	Gravitational field strength
M or m_a	Average molecular mass of the atmosphere =0.028964 kg/mol
R	Gas constant = 8.3144
F_b	Buoyancy force
k_b or k	Boltzmann's constant
n	Number of moles
F_g	Gravitational force
v	Velocity of the vessel
v_{wind}	Velocity of the wind
C_d	Drag coefficient (Dependent on payload shape)
m_T	Mass of the payload and its components

B Summary of the payload components for the project

Item	Item price (£)	Postage and packaging(£)	Weight towards payload(Kg)
Xexun	64.03	0	0.05
Iphone	0	0	0.135
Garmin e-Trex GPS Logger	63.99	0	0.15
Lascar EL-USB-TC	0	0	0.07
MSR145	101.62	40.2	0.02
Lascar EL-USB-2	25	6.30+vat	0.07
Tachyon XC HD	0	29.78	0.16
Canon Camera	0	0	0.25
Helium Large Cylinder	74.4	0	n/a
1600g Hwoyee Balloon x 2	79.95x2	3.5	1.6
Parachute 4FT	$45	0	0.02
Cord (20m)	0	0	n/a
LED's x 4	13	0	n/a
Buzzer	12.16	0	n/a
Orange Gaffer Tape x 2	20	0	n/a
Vodafone credit	10	0	n/a
Vaisala RS80	10	0	n/a
Radio receiver	0	0	n/a
	total price=	£688.89	Total weight=1.63kg

Figure 8: A summary of the components bought for the project

C Instructions for obtaining data from the NOMADS site

```
0. Useful url for info: http://nomads.ncdc.noaa.gov/guide/index.php?name=advanced#adv-gdsascii
1. Go to: http://nomads.ncep.noaa.gov/
2. Click on "GFS 1.0x1.0 Degree - OpenDAP"
3. Click on latest forecast at bottom: e.g. "30: gfs20110406/: dir"
4. Click on latest "gfs_*z:" link, e.g. "3: gfs_06z: GFS fcst starting from 06Z06apr2011, downloaded Ap

5. Cut and paste base url under "OPeNDAP/DODS Data URL:", e.g.
"http://nomads.ncep.noaa.gov:9090/dods/gfs/gfs20110406/gfs_06z"
This is 6am forecast on 6th April 2011.

6. Now add the parameters to download the data. Parameters we want are
i. ugrdprs (m/s wind speed in E-W direction)
ii. vgrdprs (m/s wind speed in N-S direction)
iii. hgtprs (geopotential height in m, i.e. altitude corresponding to pressure
1000 975 950 925 900.. 7 5 3 2 1)

Longitude: 0.00000000000E to 359.00000000000E (360 points, avg. res. 1.0 deg)
Latitude: -90.00000000000N to 90.00000000000N (181 points, avg. res. 1.0 deg)
Altitude: 1000.00000000000 to 10.00000000000 (26 points, avg. res. 39.6)
Time: 06Z06APR2011 to 06Z14APR2011 (65 points, avg. res. 0.125 days)
```

```
Values for brackets:
Latitude: x = 53.3+90.0 / 1.0 = 143
Longitude = 358.4-0.0 / 1.0 = 358
Altitude - we want all values, so 0:25
Time = [Number of hours between now and 06:00 on 06 April 2011) / 24] / 0.125 = 4 for 18:00 on 06 April

So the url we want is:
http://nomads.ncep.noaa.gov:9090/dods/gfs/gfs20110406/gfs_06z.ascii?ugrdprs[4][0:25][143][358]

This gives the following output at the foot of the page:

time, [1]
734234.75
lev, [26]
1000.0, 975.0, 950.0, 925.0, 900.0, 850.0, 800.0, 750.0, 700.0, 650.0,
600.0, 550.0, 500.0, 450.0, 400.0, 350.0, 300.0, 250.0, 200.0, 150.0,
100.0, 70.0, 50.0, 30.0, 20.0, 10.0
lat, [1]
53.0
lon, [1]
358.0

The above is correct: latitude and longitude is only known to 1
degree, the pressure goes from 1000->10 and the time value is the
fifth one in the array (this can be checked by entering [0:64] in the
first bracket instead of [4] and checking that the fifth value is
indeed equal to 734234.75
```

D Payload design schematics

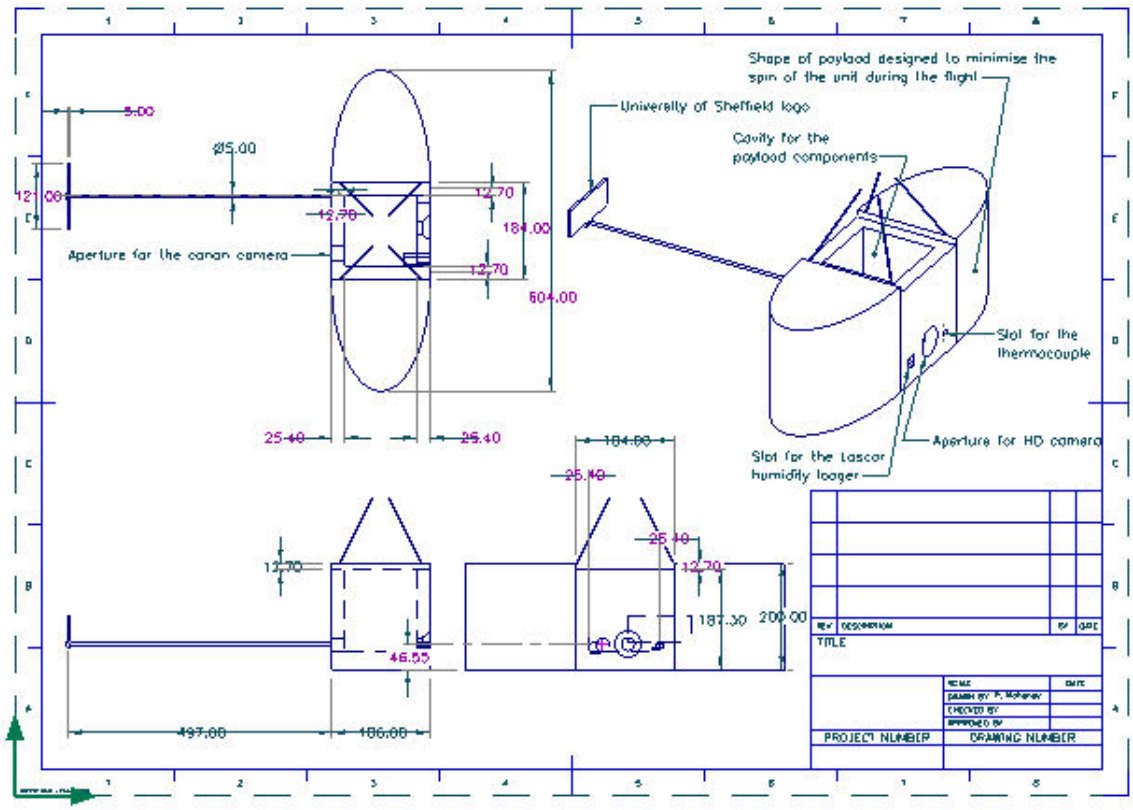

Figure 9: A scaled schematic of the payload shell

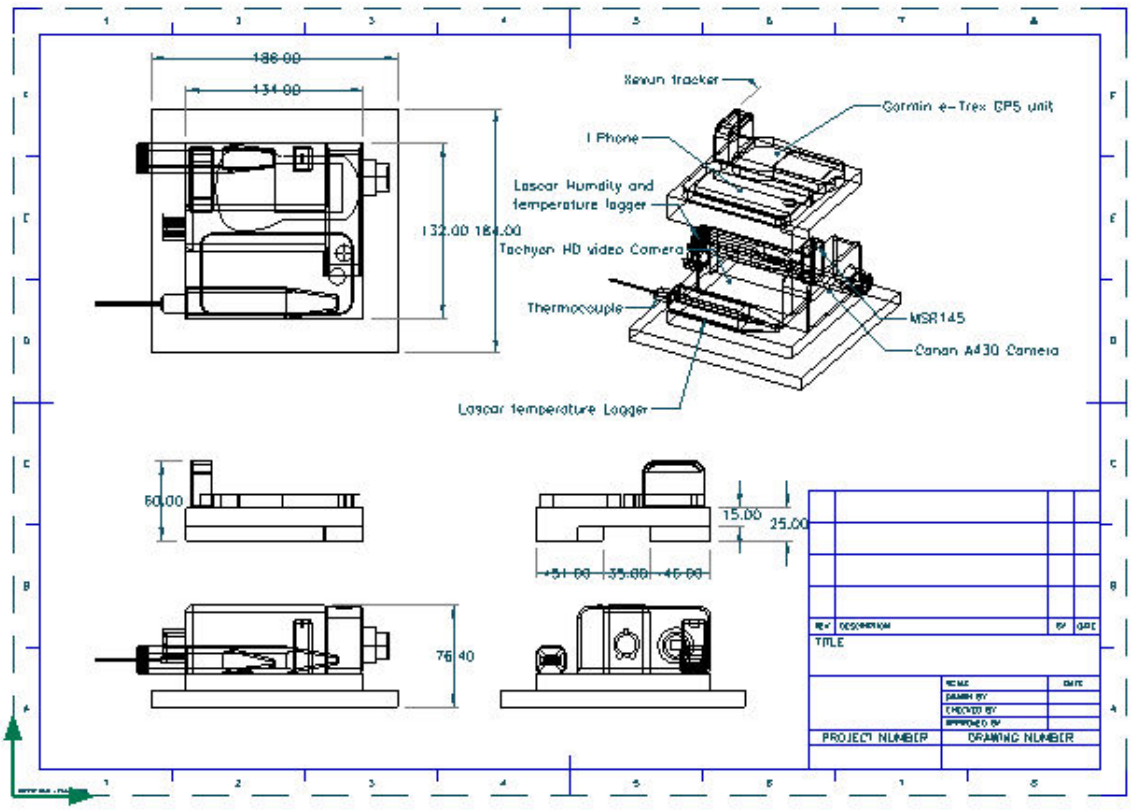

Figure 10: A scaled schematic of the inner payload box

E A copy of the basic algorithm implemented in the C++ simulation

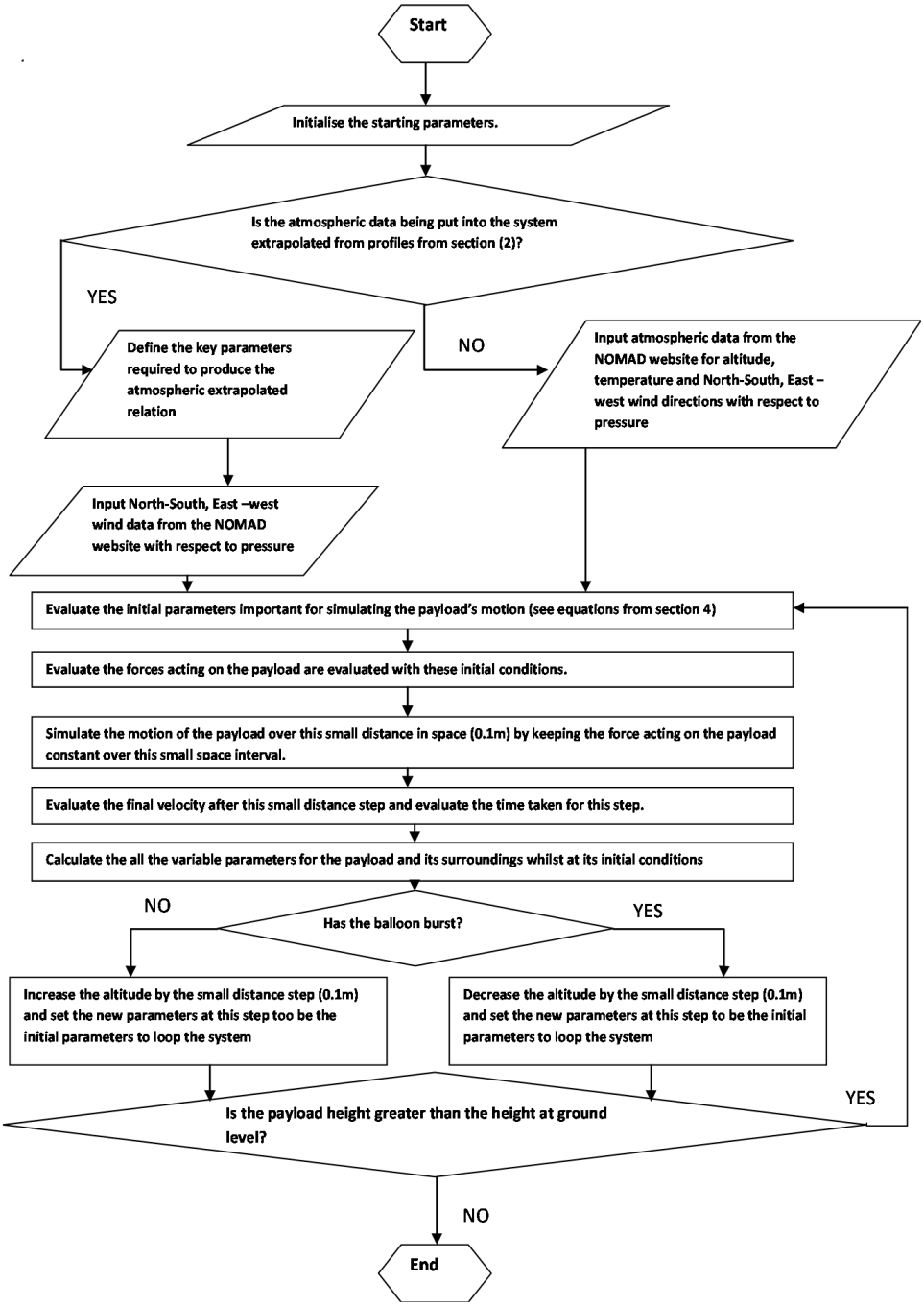

Figure 11: This is a basic map of the C++ algorithm used to evaluate to payload flight and any meteorological data projections.

F Results from testing of the payload components

F.1 Simulation results for the Expected meteorological data obtained from the flight

F.2 MSR145

The total run time for the MSR test was 4.89 hours. The software indicated that the unit still had several days of battery life and memory remaining, making it suitable for the project.

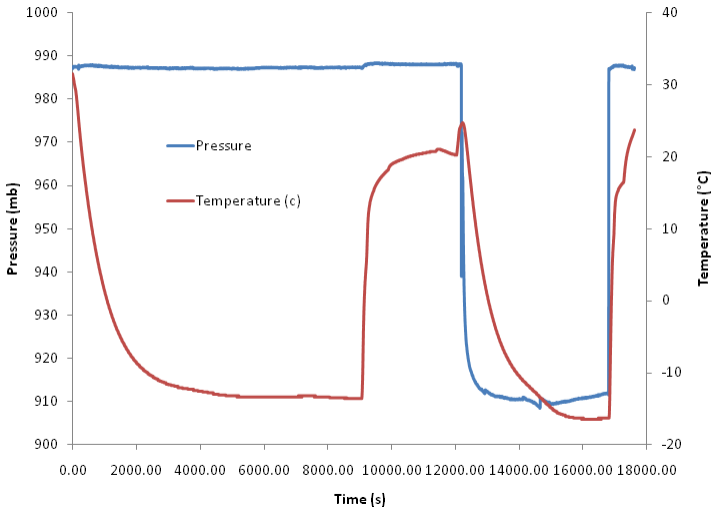

Figure 12: This shows an extended test on the MSR145 unit.

The unit was placed both inside and outside the test box in a -20^{o}C freezer for significant time periods.
A two stage test of the MSR145 was completed on a single charging of the unit. MSR145 was first placed in the test box in the freezer for 2.52 hours. The temperature recorded decreased slowly until it reaches a minimum temperature of -13.6^oC at around 1.34 hours.
The MSR145 measured a small pressure drop pressure in the freezer as a consequence to a rearrangement of the ideal gas law:

$$P = \frac{nRT}{V} \tag{12}$$

The second section of the test consisted of the MSR145 being placed in deflated plastic container (without the test box) in the freezer for a further 1.28 hours. The unit recorded a pressure decrease of around 76mb, as well as an minimum temperature of -16.5^oC. The time taken for the unit to reach this minimum temperature is less than half the time it took in the test box. This demonstrates the good insulation properties of the test box material.

F.3 Lascar temperature and temperature+humidity loggers

The temperature logger was tested both in and out the test box in the freezer. as shown in figure (13).

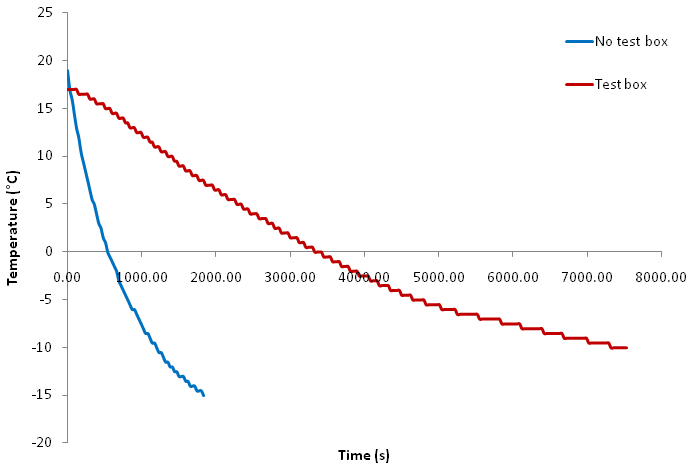

Figure 13: This is the projected meteorological results expected to be returned from the payload after the flight.

This figure demonstrates the insulating properties of the test box. The Temperature logger inside the box took much longer time to reach a minimum steady temperature than when it was not in the box.
The lifetime of both the Temperature and combined temperature and humidity loggers is sufficient with testing lasting beyond 9 hours in a variety of different conditions (outdoors, indoors, sea air, etc).

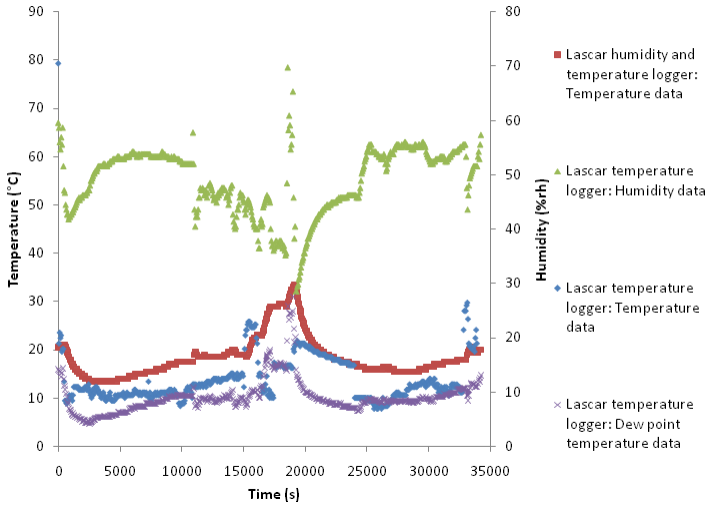

Figure 14: This shows the Lascar Temperature /Humidity and Temeprature logger being tested for an extended period of time in a variety of conditions (indoors, city centre, sea air, etc).

F.4 I-phone Viewranger

The tracker was tested by taking the I-phone inside the test box around Sheffield city centre, whilst other members of the group tracked its whereabouts. The result of the test was correct and accurate, showing

that the unit is capable of receiving a signal through the payload housing material. The operation of the View-ranger application on the I-phone appeared to be a power draining process lasting only an hour or so after the predicted flight time. This may be due to the fact that the I-phone model used is an old one, thus the battery would have been suffering from fatigue.

F.5 Garmin e-Trex H testing

The unit was tested inside the test box to see if a GPS signal could penetrate the payload's insulation. The unit was left inside the box over a journey to ensure the unit could record a track. A test where the unit was placed inside a car whilst inside the test box and driven around the city of Sheffield. The unit logged an accurate track for the journey, making it suitable for the task at hand.

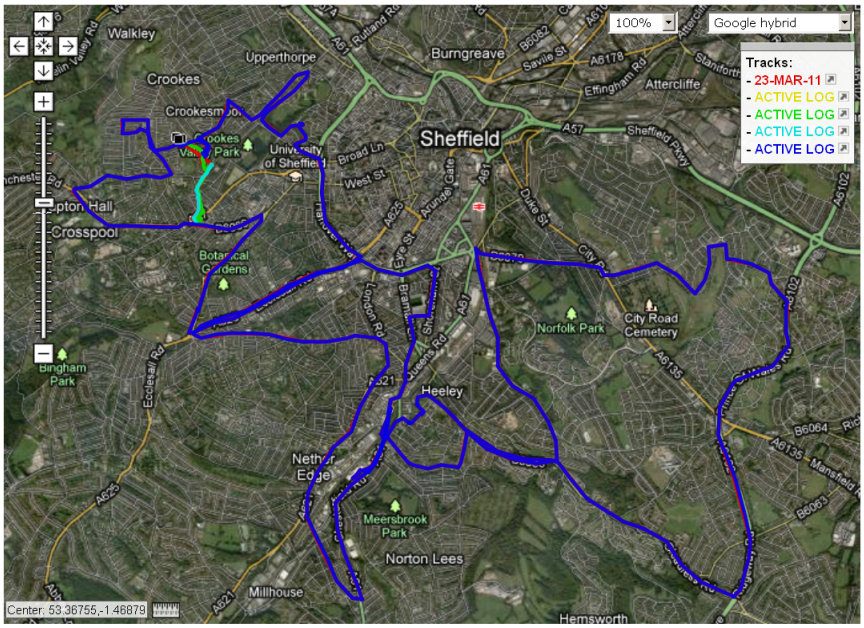

Figure 15: This shows a test of the Garmin e-Trex H unit. The unit travelled for 2 hours inside the testbox, which was in turn inside a car. There are no gaps in data in the returned track

F.6 Xexun

To simulate the conditions for that may cause the Xexun to fail, the unit was placed in the freezer for a period of 1 hour in the test box. It was then taken out and signalled to see if a response was texted back.

The first test resulted in the unit replying after a delay of approximately 10 minutes. The unit was signalled several times prior to the response at 10 minutes and the unit texted back several times after the 10 minutes were up. This indicates that the signals sent to the unit were logged and a reply was sent when the unit was

capable of doing so. This could be attributed to the location of the test which has a low O2 network signal. This process was repeated and the unit was found to respond almost immediately on the second test.

G C++ programme source code

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include <cmath>
#include "altitudeprop.h"

//#include"falling.h"
using namespace std;
int main()
{
    FILE *outfile1;
    force test;
    test.parameters();
    outfile1=fopen("FLIGHT_DATA.csv","w");
    cout<<" the number of moles of gas put into balloon for lift: "<<test.nmoles()<<" " <<endl;

cout<<"programme running.... please wait....."<<endl;

fprintf(outfile1,"Time (s),North-south distance (m),east-west distance (m),Altitude (m),pressure (pascal

while (test.burst<1){

//while (test.volume()<test.burstvol){
 for(int j=0; j<=test.max; ++j){
                        test.temp();
                        test.pressure();
                        test.volume();
                        test.height();
                       //cout<<"grav:"<<test.grav()<<" buoy:"<<test.buoy()<<" drag"<<test.drag()<<" force


                        //fprintf(outfile1,"%f,%f,%f,\n",test.tdistn,test.tdiste,test.height());
                        test.fvelocityassign();
                        test.time();
                        //test.time_step;
                        test.distancen();
                        test.distancee();
                        test.velocity_update();
                        test.increment();
                        }

                        test.temp();
                        test.pressure();
                        test.volume();
                        //cout<<"the height is "<<test.height()<<""<<endl;
```

```
          //cout<<"grav:"<<test.grav()<<" buoy:"<<test.buoy()<<" drag"<<test.drag()<<" for
       //  cout<<"timestep "<<test.time_step<<"time gone"<<test.initial_time<<"alt "<<test
         //system("pause");
          fprintf(outfile1,"%f,%f,%f,%f,%f,%f,%f,%f,%f,\n",test.initial_time,test.tdistn,t
          test.fvelocityassign();
          test.time();
          //test.time_step;
          test.distancen();
          test.distancee();
          test.velocity_update();
          test.increment();
         }
 double top= test.height();
 double halftime=test.initial_time;
 double pop_north=test.tdistn;
 double pop_east=test.tdiste;

/*while(test.height()>=0)

          {test.fvelocity= sqrt((test.ivelocity*test.ivelocity)+2*(test.grav())*test.heigh
          test.velocity_update();
          test.increment();
          cout<<"OVERSHOOT"<<endl;}



  // }



          */
     cout<<"programme running.... now over 50% completed.... please wait....."<<endl;
   test.ivelocity=0;
   test.fvelocity=0;

  while(test.height()>=0&&test.burst==1){
                            //cout<<"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
           for(int y=0; y<test.max; y++){
                           if(test.height()>=0){
                           test.temp();
                           test.pressure();
                           test.height();
                           test.fvelocityassign();
                           test.time();
                           test.distancen();
                           test.distancee();
                           //cout<<"the height is "<<test.height()<<""<<endl;
                             //cout<<"grav:"<<test.grav()<<" bouy:"<<test.buoy()<<" drag"
                             //fprintf(outfile1,"%f,%f,%f,\n",test.tdistn,test.tdiste,tes
```

```
                                        test.velocity_update();
                                        test.decrease();    }
                                        }
                                        if (test.height()>=0){
                                        test.temp();
                                        test.pressure();
                                        test.height();
                                        // test.Area();
                                       // cout<<"the height is "<<test.height()<<""<<endl;
                                        //cout<<"grav:"<<test.grav()<<" bouy:"<<test.buoy()<<" drag"

                                        test.fvelocityassign();
                                        test.time();
                                        test.distancen();
                                        test.distancee();
                                        fprintf(outfile1,"%f,%f,%f,%f,%f,%f,%f,%f,%f,\n",test.initial
                                        test.velocity_update();
                                        test.decrease();}}
                        cout<<"timestep of last point is "<<test.time_step<<""<<endl;
                        system ("pause");
                        cout<<"Balloon bursts at "<<top<<" Time for flight is "<<test.initial_time<<" whi
                        system ("pause");
                        cout<<"Balloon time for ascent "<<halftime<<" Time for descent is "<<test.initial
                        system ("pause");
                        cout<<"the distance the balloon travels north is"<<test.tdistn<<" and east is "<<
                        system ("pause");
                        cout<<"The balloon pops at "<<pop_north<<" north and "<<pop_east<<" from the take
                        system ("pause");
                        cout<< "the ascent rate is "<<top/halftime<<" and decsent rate is "<<top/(test.in
                        system ("pause");

fclose(outfile1);
cout<< "A file has now been saved to the directory with all the relevant data saved inside. This file sh
system ("pause");
cout<<"Thankyou for using the EOS simulation programme... good luck with your project!"<<endl;
system ("pause");
return 0;

}
#ifndef ALTITUDEPROP_H
#define ALTITUDEPROP_H


class force {

      private: double alt;
               double turnover1;
               double turnover2;
               double turnover3;
```

```
        double temp1;
        double temp2;
        double temp3;

        double groundpressure;
        double groundvol;
        double n;
        double groundtemp;
        //double ivelocity;
        //double fvelocity;
        //double initial_time;


public: force();
        double ma;
        double s;
        double para_area;
        double burst;
        double burstvol;
        double cd();
        double i_vwe;
        double time_step;
        double tdistn;
        double tdiste;
        double max;
        double hbase;
        double mass();
        double altpressure();
        double increment();
        double decrease();
        void parameters();
        double height()const;
        double temp();
        double pressure();
        double initial_time;
        double ivelocity;
        double fvelocity;
        double nmoles();
        void fvelocityassign();
        void velocity_update();
        double volume();
        double drag();
        double buoy();
        double grav();
        double Ftotal();
        double fvelocity_update();
        double time();
        double Area();
        double freefall();
        int parameter_choice1;
        int parameter_choice2;
```

```
int parameter_choice3;
double boxmass;
double balloonmass;
double balloonmassburst;
double velocitybn;
double velocitybe;
double distancee();
double distancen();
double vwn();
double vwe();
double cd1;
double cd2;
double ta;
double tb;
double tc;
double td;
double te;
double tf;
double tg;
double th;
double ti;
double tj;
double tk;
double tl;
double tm;
double tn;
double to;
double tp;
double tq;
double tr;
double ts;
double tt;
double tu;
double tv;
double tw;
double tx;
double ty;
double tz;

double pa;
double pb;
double pc;
double pd;
double pe;
double pf;
double pg;
double ph;
double pi;
double pj;
double pk;
double pl;
```

```
double pm;
double pn;
double po;
double pp;
double pq;
double pr;
double ps;
double pt;
double pu;
double pv;
double pw;
double px;
double py;
double pz;

double ha;
double hb;
double hc;
double hd;
double he;
double hf;
double hg;
double hh;
double hi;
double hj;
double hk;
double hl;
double hm;
double hn;
double ho;
double hp;
double hq;
double hr;
double hs;
double ht;
double hu;
double hv;
double hw;
double hx;
double hy;
double hz;
double vina;
double vinb;
double vinc;
 double vind;
 double vine;
 double vinf;
 double ving;
 double vinh;
 double vini;
 double vinj;
```

```
                double vink;
                double vinl;
                double vinm;
                double vinn;
                double vino;
                double vinp;
                double vinq;
                double vinr;
                double vins;
                double vint;
                double vinu;
                double vinv;
                double vinw;
                double vinx;
                double viny;
                double vinz;

                double evina;
                double evinb;
                double evinc;
                double evind;
                double evine;
                double evinf;
                double eving;
                double evinh;
                double evini;
                double evinj;
                double evink;
                double evinl;
                double evinm;
                double evinn;
                double evino;
                double evinp;
                double evinq;
                double evinr;
                double evins;
                double evint;
                double evinu;
                double evinv;
                double evinw;
                double evinx;
                double eviny;
                double evinz;

               };

#endif
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include <cmath>
```

```
#include"altitudeprop.h"

//#include"falling.h"
using namespace std;




force::force() {alt=0;}

void force::parameters()
{ parameter_choice3=1;
  pa=1000;
  pb=975;
  pc=950;
  pd=925;
  pe=900;
  pf=850;
  pg=800;
  ph=750;
  pi=700;
  pj=650;
  pk=600;
  pl=550;
  pm=500;
  pn=450;
  po=400;
  pp=350;
  pq=300;
  pr=250;
  ps=200;
  pt=150;
  pu=100;
  pv=70;
  pw=50;
  px=30;
  py=20;
  pz=10;
cout<<"Hello and welcome to the EOS payload simulation programme"<<endl;
system("pause");
printf("If you wish to run the simulation for data from the EOS flight simulation on the 8th of April, p
scanf("%d",&parameter_choice1);


if (parameter_choice1==1){
max=100;
parameter_choice2=2;
parameter_choice3=1;
      turnover1=11000;
      turnover2=20000;
```

```
    turnover3=50000;
    temp1=215;
    temp2=215;
    temp3=268;
    groundtemp=283;
    groundvol=(nmoles()*8.31*groundtemp)/groundpressure;
    groundpressure=101000;
    ma=28*1.67E-27;
    s=0.1;
    burst=0;
    para_area=0.8485;

  // cout<< "temp1: "<<temp1<<" temp2: "<<temp2<<" temp3: "<<temp3<<" turnover 1, 2 and 3: "<<turnove
    burstvol=(4/3)*3.141592654*pow(4.72,3.0);//((4/3)*3.141592654*4.2*4.2*4.2);
    boxmass=1.6;
    balloonmass=1.6;
    balloonmassburst=0.8;
    //mass=1.0;
    ivelocity=0;
    fvelocity=0;
    initial_time=0;
    cd1=0.25;//0.47;
    cd2=0.5;
    i_vwe=0;
    time_step=0;
    velocitybn=0;
    velocitybe=0;
    tdistn=0;
    tdiste=0;

vina=-2.99;
vinb=-7.6099997;
vinc=-9.53;
vind=-9.139999;
vine=-7.6699996;
vinf=-5.95;
ving=-6.02;
vinh=-5.91;
vini=-5.73;
vinj=-5.72;
vink=-7.17;
vinl=-8.82;
vinm=-9.51;
vinn=-9.559999;
vino=-8.2;
vinp=-6.4;
vinq=-2.8;
vinr=3.5;
vins=7.2000003;
vint=-0.85999995;
vinu=-5.08;
```

```
vinv=-7.6299996;
vinw=-7.74;
vinx=-5.99;
viny=-7.7799997;
vinz=-5.89;


evina=1.6999999;
evinb=4.11;
evinc=3.9499998;
evind=2.23;
evine=0.48999998;
evinf=0.01;
eving=0.98999995;
evinh=2.72;
evini=4.15;
evinj=5.0499997;
evink=6.54;
evinl=9.179999;
evinm=12.66;
evinn=15.32;
evino=15.7;
evinp=15.2;
evinq=15.7;
evinr=15.400001;
evins=10.2;
evint=12.34;
evinu=9.75;
evinv=6.29;
evinw=2.76;
evinx=1.53;
eviny=-1.41;
evinz=-0.96;

ha=230.90701;
hb=438.65402;
hc=652.81104;
hd=873.41003;
he=1100.727;
hf=1573.618;
hg=2072.8381;
hh=2602.1511;
hi=3161.9312;
hj=3754.5342;
hk=4385.187;
hl=5060.1;
hm=5785.81;
hn=6570.4097;
ho=7423.6797;
hp=8364.54;
hq=9413.84;
```

```
hr=10610.67;
hs=12033.64;
ht=13833.2295;
hu=16351.21;
hv=18544.549;
hw=20621.08;
hx=23792.4;
hy=26320.44;
hz=30731.809;
```

```
    cout<<"pre defined values used"<<endl;
    }
    if (parameter_choice1==2){

    printf("This option requires you to enter some data in a similar format to that given by the NOMA
    system("pause");
    printf("The NOMAD website from NOAA provides data for parameters over\n 26 altitude levels define
    system("pause");
    printf("To ensure complete compatability with this programme, any data read into the program must
    printf("[0][12][0], 387\n\n[0][13][0], 335\n\n[0][14][0], 320\n");
    system("pause");
    printf("The text at the top and the bottom of the file should also be removed, \n so all you have
    system("pause");
    printf("the spacing between the data should not be altered...this will cause the\n programme to fa
    system("pause");
    printf("Once this is acheived, save all the data as 'TXT' files. Each files should be \nentitled
    system("pause");
    printf("For temperature data, save the file as temp.txt\n");
    system("pause");
    printf("For height with pressure data, save the file as height.txt\n");
    system("pause");
    printf("For east to west wind data, save the file as ugdprs.txt\n");
    system("pause");
    printf("For north to south wind data, save the file as vgdprs.txt\n");
    system("pause");
    printf("The amount of data you enter depends on the following decision:\n");
    printf("(NOTE!!!! The data files and the programme should be placed\n all in the same directory, s
    system("pause");
    printf("If you want to use standard profiles for temperature and pressure described on the websit
    printf("If you want to input all that data in the these txt files for pressure, temperature, nort
    scanf("%d",&parameter_choice2);

    if (parameter_choice2==1){parameter_choice3=2;
                              printf("You are now putting in ALL data from files you have already obta
                              system("pause");
                              printf("before continuing, please ensure all the files are saved in the
                              system("pause");
```

```
printf("reading in files...\n");
FILE *infiletemp;
FILE *infileup;
FILE *infilenswind;
FILE *infileewwind;
infiletemp=fopen("kelvin.txt","r");


fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n", &ta );
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n", &tb);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tc);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&td);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&te);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tf);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tg);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&th);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&ti);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tj);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tk);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tl);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tm);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tn);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&to);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tp);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tq);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tr);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&ts);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tt);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tu);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tv);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tw);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tx);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&ty);
fscanf(infiletemp, "[%*d][%*d][%*d], %lf\n\n",&tz);


fclose(infiletemp);
printf("temperature data entered....\n");




infileup=fopen("height.txt","r");

fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ha);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hb);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hc);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hd);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&he);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hf);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hg);
```

```
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hh);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hi);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hj);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hk);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hl);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hm);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hn);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ho);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hp);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hq);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hr);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hs);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ht);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hu);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hv);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hw);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hx);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hy);
fscanf(infileup, "[%*d][%*d][%*d], %lf",&hz);
fclose(infileup);
printf("Altitude vs pressaure data entered....\n");

infilenswind=fopen("vgdprs.txt","r");

fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vina);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinb);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinc);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vind);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vine);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinf);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&ving);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinh);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vini);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinj);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vink);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinl);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinm);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinn);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vino);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinp);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinq);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinr);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vins);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vint);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinu);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinv);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinw);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinx);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&viny);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinz);
```

```
                                fclose(infilenswind);
                                printf("North-south wind data entered....\n");



                                infileewwind=fopen("ugdprs.txt","r");

                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evina);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinb);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinc);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evind);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evine);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinf);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&eving);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinh);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evini);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinj);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evink);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinl);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinm);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinn);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evino);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinp);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinq);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinr);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evins);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evint);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinu);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinv);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinw);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinx);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&eviny);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinz);

                                fclose(infileewwind);
                                printf("East-West wind data entered....\n");

system("pause");

      //put in read file commands

              groundtemp=ta;
      groundvol=3.05;//3.33;
      groundpressure=pa*100;
      ma=28*1.67E-27;
      //printf("Please enter the distance steps (resolution) you want to \n evaluate the simulation ove
      //scanf("%lf",&s);
      s=0.1;
      burst=0;
      para_area=1;
      //printf("Please enter the area of the parachute, used for the payload's descent:\n");
```

```
  //scanf("%lf",&para_area);
  //printf("please enter the volume (in cubic meters) at which the balloon explodes:\n");
  //scanf("%lf",&burstvol);
  burstvol=((4/3)*3.141592654*4.72*4.72*4.72);//((4/3)*3.141592654*4.2*4.2*4.2);
  boxmass=1.6;
  //printf("please enter the mass of the payload box:\n");
  //scanf("%lf",&boxmass);
  balloonmass=1.6;
  printf("The programme only passes the data for one altutude value per\n certain number of altitude
  scanf("%lf",&max);
  //printf("please enter the mass of the uninflated balloon:\n");
  //scanf("%lf",&balloonmass);
  balloonmassburst=0.8;
  //printf("please enter the co-efficient of drag for the payload's ascent:\n");
  //scanf("%lf",&cd1);
  cd1=0.25;//0.47;
  //printf("please enter the co-efficient of drag for the payload's descent:\n");
  //scanf("%lf",&cd1);
  cd2=0.5;
  //mass=1.0;



  //printf("please enter the estimated mass of the balloon \n you expect to still be attached after
  //scanf("%lf",&balloonmassburst);
  groundvol=3.05;
  ivelocity=0;
  fvelocity=0;
  initial_time=0;

  i_vwe=0;
  time_step=0;
  velocitybn=0;
  velocitybe=0;
  tdistn=0;
  tdiste=0;                    }
  if (parameter_choice2==2){parameter_choice3=2;
                            parameter_choice2=2;

  //turnover1=11000;
  //turnover2=20000;
  //turnover3=50000;
  //temp1=223;
  //temp2=223;
  //temp3=268;
  //groundtemp=283;

  //groundpressure=101000;

  //para_area=1;
// cout<< "temp1: "<<temp1<<" temp2: "<<temp2<<" temp3: "<<temp3<<" turnover 1, 2 and 3: "<<turnover
```

```
                        printf("Firstly, we will look at the temperature behaviour of the earths
                        printf("The temperature profile of the earth's atmosphere can be approxi
                        printf("These lines start from the ground and work up to a maximum alti
                        printf("The programme needs to know the altitude (in meters) at which th
                        printf("and what the temperature is at the end of each of these lines\n
                        printf("Staring with the altitude at which the line profiles change, The
                        printf("Please enter the height of the end of the first profile line (in
                        scanf("%lf", &turnover1);
                        printf("Please enter the temperature at the hieght of the end of first p
                        scanf("%lf", &temp1);
                        printf("Please enter the height of the end of the second profile line (i
                        scanf("%lf", &turnover2);
                        printf("Please enter the temperature at the hieght of the end of the sec
                        scanf("%lf", &temp2);
                        printf("Please enter the height of the end of the third profile line (in
                        scanf("%lf", &turnover3);
                        printf("Please enter the temperature at the hieght of the end of the thi
                        scanf("%lf", &temp3);
                        printf("finally, Please enter the temperature at ground level\n");
                        scanf("%lf", &groundtemp);
                        printf("That is all the information required in terms of temperature.\n


                        printf("The pressure at ground level (in pascals) is now required\n");
                        printf("Please enter this value, followed by the enter key.\n");
                        scanf("%lf",&groundpressure);
                        printf("Please enter the area of the parachute, used for the payload's d
                        scanf("%lf",&para_area);
                        printf("please enter the volume (in cubic meters) at which the balloon d
                        scanf("%lf",&burstvol);
                        printf("please enter the mass of the payload box");
                        scanf("%lf",&boxmass);
                        printf("please enter the mass of the uninflated balloon");
                        scanf("%lf",&balloonmass);
                        //printf("Please enter the distance steps (resolution) you want to \n ev
//                      scanf("%lf",&s);
                        printf("The programme only passes the data for one altutude value \n per
                        scanf("%lf",&max);
                         //printf("please enter the co-efficient of drag for the payload's ascen
                         //scanf("%lf",&cd1);
                         cd1=0.25;//0.47;
                         //printf("please enter the co-efficient of drag for the payload's desce
                         //scanf("%lf",&cd1);
                          cd2=0.5;



//boxmass=1.6;
//balloonmass=1.6;

//mass=1.0;
```

```
//printf("Please enter the average mass of the air molecules over\n the payload flight in kilogra
//scanf("%lf",&ma)
ma=28*1.67E-27;
//printf("Please enter the distance steps (resolution) you want to \n evaluate thesimulation over
//scanf("%lf",&s);
s=0.1;
ivelocity=0;
fvelocity=0;
initial_time=0;
groundvol=3.05;//3.33;
i_vwe=0;
time_step=0;
velocitybn=0;
velocitybe=0;
tdistn=0;
tdiste=0;
burst=0;
//burstvol=((4/3)*3.141592654*4.2*4.2*4.2);//((4/3)*3.141592654*4.72*4.72*4.72);
//printf("please enter the estimated mass of the balloon \n you expect to still be attached after
//scanf("%lf",&balloonmassburst);
balloonmassburst=0.8;


                        printf("reading in data now....\n");
                        FILE *infilenswind;
                        FILE *infileewwind;
                        FILE *infileup;

                        infileup=fopen("height.txt","r");

                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ha);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hb);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hc);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hd);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&he);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hf);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hg);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hh);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hi);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hj);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hk);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hl);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hm);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hn);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ho);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hp);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hq);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hr);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hs);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&ht);
                        fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hu);
```

```
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hv);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hw);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hx);
fscanf(infileup, "[%*d][%*d][%*d], %lf\n\n",&hy);
fscanf(infileup, "[%*d][%*d][%*d], %lf",&hz);
fclose(infileup);
printf("Altitude vs pressaure data entered....\n");


infilenswind=fopen("vgdprs.txt","r");

fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vina);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinb);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinc);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vind);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vine);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinf);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&ving);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinh);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vini);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinj);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vink);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinl);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinm);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinn);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vino);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinp);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinq);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinr);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vins);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vint);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinu);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinv);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinw);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinx);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&viny);
fscanf(infilenswind, "[%*d][%*d][%*d], %lf\n\n",&vinz);

fclose(infilenswind);
printf("North-south wind data entered.... %f,%f,\n", vina, vinz);
//cout<<"v wind in c out is "<<vina<<" and "<<vinz<<""<<endl;



infileewwind=fopen("ugdprs.txt","r");

fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evina);
fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinb);
fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinc);
fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evind);
fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evine);
```

```
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinf);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&eving);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinh);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evini);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinj);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evink);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinl);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinm);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinn);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evino);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinp);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinq);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinr);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evins);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evint);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinu);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinv);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinw);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinx);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&eviny);
                                fscanf(infileewwind, "[%*d][%*d][%*d], %lf\n\n",&evinz);

                                fclose(infileewwind);
                                printf("East-west wind data entered.... %f, %f\n",evina,evinz);




                                 // cout<< "temp1: "<<temp1<<" temp2: "<<temp2<<" temp3: "<<temp3<<" tu

      //mass=1.0;









                                        }
      }
      }
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include <cmath>
#include "altitudeprop.h"

//#include"falling.h"
using namespace std;
```

```
double force::vwn()
{
double i_vwn=0;

 if(height()==0){i_vwn=0;}
 if (height()>0&&height()<ha){i_vwn=(height()/ha)*(vina);}
 if (height()>=ha&&height()<hb){i_vwn=((height()-ha)/(hb-ha))*(vinb-vina)+vina;}//-3.04
 if (height()>=hb&&height()<hc){i_vwn=((height()-hb)/(hc-hb))*(vinc-vinb)+vinb;}//(-1)*2.77;}
 if (height()>=hc&&height()<hd){i_vwn=((height()-hc)/(hd-hc))*(vind-vinc)+vinc;}//(-1)*2.38999;}
 if (height()>=hd&&height()<he){i_vwn=((height()-hd)/(he-hd))*(vine-vind)+vind;}//(-1)*2.52;}
 if (height()>=he&&height()<hf){i_vwn=((height()-he)/(hf-he))*(vinf-vine)+vine;}//(-1)*4.2599998;}
 if (height()>=hf&&height()<hg){i_vwn=((height()-hf)/(hg-hf))*(ving-vinf)+vinf;}//(-1)*5.62;}
 if (height()>=hg&&height()<hh){i_vwn=((height()-hg)/(hh-hg))*(vinh-ving)+ving;}//(-1)*6.24;}
 if (height()>=hh&&height()<hi){i_vwn=((height()-hh)/(hi-hh))*(vini-vinh)+vinh;}//(-1)*5.5699997;}
 if (height()>=hi&&height()<hj){i_vwn=((height()-hi)/(hj-hi))*(vinj-vini)+vini;}//(-1)*5.2799997;}
 if (height()>=hj&&height()<hk){i_vwn=((height()-hj)/(hk-hj))*(vink-vinj)+vinj;}//(-1)*4.9;}
 if (height()>=hk&&height()<hl){i_vwn=((height()-hk)/(hl-hk))*(vinl-vink)+vink;}//(-1)*4.08;}
 if (height()>=hl&&height()<hm){i_vwn=((height()-hl)/(hm-hl))*(vinm-vinl)+vinl;}//(-1)*3.42;}
 if (height()>=hm&&height()<hn){i_vwn=((height()-hm)/(hn-hm))*(vinn-vinm)+vinm;}//(-1)*5.56;}
 if (height()>=hn&&height()<ho){i_vwn=((height()-hn)/(ho-hn))*(vino-vinn)+vinn;}//(-1)*8;}
 if (height()>=ho&&height()<hp){i_vwn=((height()-ho)/(hp-ho))*(vinp-vino)+vino;}//(-1)*8.6;}
 if (height()>=hp&&height()<hq){i_vwn=((height()-hp)/(hq-hp))*(vinq-vinp)+vinp;}//(-1)*8.6;}
 if (height()>=hq&&height()<hr){i_vwn=((height()-hq)/(hr-hq))*(vinr-vinq)+vinq;}//(-1)*9;}
 if (height()>=hr&&height()<hs){i_vwn=((height()-hr)/(hs-hr))*(vins-vinr)+vinr;}//(-1)*9;}
 if (height()>=hs&&height()<ht){i_vwn=((height()-hs)/(ht-hs))*(vint-vins)+vins;}//(-1)*3.25;}
 if (height()>=ht&&height()<hu){i_vwn=((height()-ht)/(hu-ht))*(vinu-vint)+vint;}//(-1)*2.3899999;}
 if (height()>=hu&&height()<hv){i_vwn=((height()-hu)/(hv-hu))*(vinv-vinu)+vinu;}//(-1)*3.31;}
 if (height()>=hv&&height()<hw){i_vwn=((height()-hv)/(hw-hv))*(vinw-vinv)+vinv;}//(-1)*4.71;}
 if (height()>=hw&&height()<hx){i_vwn=((height()-hw)/(hx-hw))*(vinx-vinw)+vinw;}//(-1)*9.5;}
 if (height()>=hx&&height()<hy){i_vwn=((height()-hx)/(hy-hx))*(viny-vinx)+vinx;}//(-1)*5.8;}
 if (height()>=hy&&height()<hz){i_vwn=((height()-hy)/(hz-hy))*(vinz-viny)+viny;}//(-1)*9;}
 if (height()>hz){i_vwn=0;}
 /*
   if(height()==0){i_vwn=0;}
 if (height()>0&&height()<ha){i_vwn=(vina);}
 if (height()>=ha&&height()<hb){i_vwn=vinb;}//-3.04
 if (height()>=hb&&height()<hc){i_vwn=vinc;}
 if (height()>=hc&&height()<hd){i_vwn=vind;}
 if (height()>=hd&&height()<he){i_vwn=vine;}
 if (height()>=he&&height()<hf){i_vwn=vinf;}
 if (height()>=hf&&height()<hg){i_vwn=ving;}
 if (height()>=hg&&height()<hh){i_vwn=vinh;}
 if (height()>=hh&&height()<hi){i_vwn=vini;}
 if (height()>=hi&&height()<hj){i_vwn=vinj;}
 if (height()>=hj&&height()<hk){i_vwn=vink;}
 if (height()>=hk&&height()<hl){i_vwn=vinl;}
 if (height()>=hl&&height()<hm){i_vwn=vinm;}
 if (height()>=hm&&height()<hn){i_vwn=vinn;}
 if (height()>=hn&&height()<ho){i_vwn=vino;}
 if (height()>=ho&&height()<hp){i_vwn=vinp;}
```

```
 if (height()>=hp&&height()<hq){i_vwn=vinq;}
 if (height()>=hq&&height()<hr){i_vwn=vinr;}
 if (height()>=hr&&height()<hs){i_vwn=vins;}
 if (height()>=hs&&height()<ht){i_vwn=vint;}
 if (height()>=ht&&height()<hu){i_vwn=vinu;}
 if (height()>=hu&&height()<hv){i_vwn=vinv;}
 if (height()>=hv&&height()<hw){i_vwn=vinw;}
 if (height()>=hw&&height()<hx){i_vwn=vinx;}
 if (height()>=hx&&height()<hy){i_vwn=viny;}
 if (height()>=hy&&height()<hz){i_vwn=vinz;}
 if (height()>hz){i_vwn=0;}
 */

// if (height()>25000&&abs(i_vwn)>0){i_vwn=i_vwn+(0.001);}
 //if (height()>25000&&abs(i_vwn)<=0.0000001){i_vwn=0;}


return i_vwn;
}



double force::vwe()
{




 if(height()==0){i_vwe=0;}
 if (height()>0&&height()<ha){i_vwe=(height()/ha)*(evina);}
 if (height()>=ha&&height()<hb){i_vwe=((height()-ha)/(hb-ha))*(evinb-evina)+evina;}//-3.04
 if (height()>=hb&&height()<hc){i_vwe=((height()-hb)/(hc-hb))*(evinc-evinb)+evinb;}//(-1)*2.77;}
 if (height()>=hc&&height()<hd){i_vwe=((height()-hc)/(hd-hc))*(evind-evinc)+evinc;}//(-1)*2.38999;}
 if (height()>=hd&&height()<he){i_vwe=((height()-hd)/(he-hd))*(evine-evind)+evind;}//(-1)*2.52;}
 if (height()>=he&&height()<hf){i_vwe=((height()-he)/(hf-he))*(evinf-evine)+evine;}//(-1)*4.2599998;}
 if (height()>=hf&&height()<hg){i_vwe=((height()-hf)/(hg-hf))*(eving-evinf)+evinf;}//(-1)*5.62;}
 if (height()>=hg&&height()<hh){i_vwe=((height()-hg)/(hh-hg))*(evinh-eving)+eving;}//(-1)*6.24;}
 if (height()>=hh&&height()<hi){i_vwe=((height()-hh)/(hi-hh))*(evini-evinh)+evinh;}//(-1)*5.5699997;}
 if (height()>=hi&&height()<hj){i_vwe=((height()-hi)/(hj-hi))*(evinj-evini)+evini;}//(-1)*5.2799997;}
 if (height()>=hj&&height()<hk){i_vwe=((height()-hj)/(hk-hj))*(evink-evinj)+evinj;}//(-1)*4.9;}
 if (height()>=hk&&height()<hl){i_vwe=((height()-hk)/(hl-hk))*(evinl-evink)+evink;}//(-1)*4.08;}
 if (height()>=hl&&height()<hm){i_vwe=((height()-hl)/(hm-hl))*(evinm-evinl)+evinl;}//(-1)*3.42;}
 if (height()>=hm&&height()<hn){i_vwe=((height()-hm)/(hn-hm))*(evinn-evinm)+evinm;}//(-1)*5.56;}
 if (height()>=hn&&height()<ho){i_vwe=((height()-hn)/(ho-hn))*(evino-evinn)+evinn;}//(-1)*8;}
 if (height()>=ho&&height()<hp){i_vwe=((height()-ho)/(hp-ho))*(evinp-evino)+evino;}//(-1)*8.6;}
 if (height()>=hp&&height()<hq){i_vwe=((height()-hp)/(hq-hp))*(evinq-evinp)+evinp;}//(-1)*8.6;}
 if (height()>=hq&&height()<hr){i_vwe=((height()-hq)/(hr-hq))*(evinr-evinq)+evinq;}//(-1)*9;}
 if (height()>=hr&&height()<hs){i_vwe=((height()-hr)/(hs-hr))*(evins-evinr)+evinr;}//(-1)*9;}
 if (height()>=hs&&height()<ht){i_vwe=((height()-hs)/(ht-hs))*(evint-evins)+evins;}//(-1)*3.25;}
 if (height()>=ht&&height()<hu){i_vwe=((height()-ht)/(hu-ht))*(evinu-evint)+evint;}//(-1)*2.3899999;}
 if (height()>=hu&&height()<hv){i_vwe=((height()-hu)/(hv-hu))*(evinv-evinu)+evinu;}//(-1)*3.31;}
```

```
  if (height()>=hv&&height()<hw){i_vwe=((height()-hv)/(hw-hv))*(evinw-evinv)+evinv;}//(-1)*4.71;}
  if (height()>=hw&&height()<hx){i_vwe=((height()-hw)/(hx-hw))*(evinx-evinw)+evinw;}//(-1)*9.5;}
  if (height()>=hx&&height()<hy){i_vwe=((height()-hx)/(hy-hx))*(eviny-evinx)+evinx;}//(-1)*5.8;}
  if (height()>=hy&&height()<hz){i_vwe=((height()-hy)/(hz-hy))*(evinz-eviny)+eviny;}//(-1)*9;}
  if (height()>hz){i_vwe=0;}

        /*

   if(height()==0){i_vwe=0;}
  if (height()>0&&height()<ha){i_vwe=(evina);}
  if (height()>=ha&&height()<hb){i_vwe=evinb;}//-3.04
  if (height()>=hb&&height()<hc){i_vwe=evinc;}
  if (height()>=hc&&height()<hd){i_vwe=evind;}
  if (height()>=hd&&height()<he){i_vwe=evine;}
  if (height()>=he&&height()<hf){i_vwe=evinf;}
  if (height()>=hf&&height()<hg){i_vwe=eving;}
  if (height()>=hg&&height()<hh){i_vwe=evinh;}
  if (height()>=hh&&height()<hi){i_vwe=evini;}
  if (height()>=hi&&height()<hj){i_vwe=evinj;}
  if (height()>=hj&&height()<hk){i_vwe=evink;}
  if (height()>=hk&&height()<hl){i_vwe=evinl;}
  if (height()>=hl&&height()<hm){i_vwe=evinm;}
  if (height()>=hm&&height()<hn){i_vwe=evinn;}
  if (height()>=hn&&height()<ho){i_vwe=evino;}
  if (height()>=ho&&height()<hp){i_vwe=evinp;}
  if (height()>=hp&&height()<hq){i_vwe=evinq;}
  if (height()>=hq&&height()<hr){i_vwe=evinr;}
  if (height()>=hr&&height()<hs){i_vwe=evins;}
  if (height()>=hs&&height()<ht){i_vwe=evint;}
  if (height()>=ht&&height()<hu){i_vwe=evinu;}
  if (height()>=hu&&height()<hv){i_vwe=evinv;}
  if (height()>=hv&&height()<hw){i_vwe=evinw;}
  if (height()>=hw&&height()<hx){i_vwe=evinx;}
  if (height()>=hx&&height()<hy){i_vwe=eviny;}
  if (height()>=hy&&height()<hz){i_vwe=evinz;}
  if (height()>hz){i_vwe=0;}

 /*if(height()==0){i_vwn=0;}
  if (height()<=10000&&height()>0){i_vwe=i_vwe-1.003;}
  if (height()>10000&&height()<=25000){i_vwe=i_vwe-0.003;}
  if (height()>25000&&abs(i_vwe)>0){i_vwe=i_vwe-(0.007);}
  if (height()>25000&&abs(i_vwe)<=0.000001){i_vwe=0;}
  //cout<<"the velocity of the east wind is"<<i_vwe<<""<<endl;*/
  //i_vwe=-4;
  return i_vwe;
 }




double force::distancen()
```

```
{double distn;



//v=u+at---> works which ever the direction. find v, then direction can be found



//distn=velocitybn*time_step;
 double an= (vwn()-velocitybn);
 double bn=(velocitybn-vwn());
 double xi;
if (burst==0){xi=(pressure()*ma*(cd()+0.1)*Area())/(2*(1.38E-23)*temp());}
if (burst==1){xi=(pressure()*ma*cd()*0.01)/(2*(1.38E-23)*temp());}
 double fdn=xi*an*an/mass();
 if (vwn()>=0&&vwn()>=velocitybn){velocitybn=velocitybn+(fdn*time_step);}
 if (vwn()>=0&&vwn()<velocitybn){velocitybn=velocitybn-(fdn*time_step);}


 if (vwn()<0&&vwn()<=velocitybn){velocitybn=(velocitybn-(fdn*time_step));}

 if (vwn()<0&&vwn()>velocitybn){velocitybn=(velocitybn)+(fdn*time_step);}

 //distn=(velocitybn*time_step)+((0.5*xi*time_step*time_step*an*an)/mass());}
distn=velocitybn*time_step;
 //distn=(velocitybn*time_step)-((0.5*xi*time_step*time_step*an*an)/mass());}

 tdistn=tdistn+distn;



 //cout<<"xi"<<xi<<" vwind"<<vwn()<<"v balloon"<<velocitybn<<"time step"<<time_step<<""<<endl;
 return distn;}

double force::distancee()


{double diste;
double xi;
if (burst==0){xi=(pressure()*ma*cd()*Area())/(2*(1.38E-23)*temp());}
if (burst==1){xi=(pressure()*ma*cd()*0.01)/(2*(1.38E-23)*temp());}


//v=u+at---> works which ever the direction. find v, then direction can be found



//distn=velocitybn*time_step;
 double ae= (vwe()-velocitybe);

 double fde=xi*ae*ae/mass();
```

```
 if (vwe()>=0&&vwe()>=velocitybe){velocitybe=velocitybe+(fde*time_step);}
 if (vwe()>=0&&vwe()<velocitybe){ velocitybe=velocitybe-(fde*time_step);}


 if (vwe()<0&&vwe()<=velocitybe){ velocitybe=(velocitybe-(fde*time_step));}

 if (vwe()<0&&vwe()>velocitybe){velocitybe=(velocitybe)+(fde*time_step);}

 //distn=(velocitybn*time_step)+((0.5*xi*time_step*time_step*an*an)/mass());}
diste=velocitybe*time_step;
 //distn=(velocitybn*time_step)-((0.5*xi*time_step*time_step*an*an)/mass());}

 tdiste=tdiste+diste;




 return diste;}

#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include <cmath>
#include"altitudeprop.h"

//#include"falling.h"
using namespace std;



double force::drag()

{

                  double woop=(pressure()*ma*cd()*Area())/(2*(1.38E-23)*temp());
                  double a=(ivelocity*ivelocity)+((2*(grav()+buoy())*s)/(mass()));
                  double b= (1/woop)-((2*s)/mass());
                  double fd=woop*ivelocity*ivelocity;

                  //double fd=a/b;
                  return -fd;




                  }
    //else{freefall();}


double force::freefall(){
```

```
                        double woop=(pressure()*ma*cd()*para_area)/(2*(1.38E-23)*temp());
                        //double a=(ivelocity*ivelocity)-((2*(grav())*s)/mass());
                        //double b= (1/woop)-((2*s)/mass());
                         double fd=woop*ivelocity*ivelocity;
                        //double fd=a/b;
                        return fd;}



    //if (ivelocity==0){ return 0;}

 //cout<< "drag is "<<fd<<" " <<endl;


double force::buoy()
{
     double fbload=2;//0.65*9.81;//(mass()-2.2)*9.81;  // this fits results adjust later so that it is s
       //double fb=((ma*nmoles()*8.31*groundtemp*9.81)/((1.38E-23)*temp()));

       return fbload;

}


double force::grav()
{double fgb=((-1)*6.67e-27*5.997e24*mass())/((6400000+height())*(6400000+height()));
       double fg= (-1)*mass()*9.81;// upgrade later
       return fg;
}

//


double force::Ftotal()
{double Ft=999999;
     if (burst==0){Ft=buoy()+drag();}
       else {Ft=-grav()-freefall();}
       return Ft;
}



#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include <cmath>
#include"altitudeprop.h"
//#include"falling.h"
using namespace std;
```

```
double force::temp()
{      double tempout;
       double t1=0;
       double t2=0;
       double t3=0;
       //parameters();
  if (parameter_choice2==1){
       if(height()==0){tempout=groundtemp;}
  if (height()>0&&height()<ha){tempout=(ta);}
  if (height()>=ha&&height()<hb){tempout=((height()-ha)/(hb-ha))*(tb-ta)+ta;}//-3.04
  if (height()>=hb&&height()<hc){tempout=((height()-hb)/(hc-hb))*(tc-tb)+tb;}//(-1)*2.77;}
  if (height()>=hc&&height()<hd){tempout=((height()-hc)/(hd-hc))*(td-tc)+tc;}//(-1)*2.38999;}
  if (height()>=hd&&height()<he){tempout=((height()-hd)/(he-hd))*(te-td)+td;}//(-1)*2.52;}
  if (height()>=he&&height()<hf){tempout=((height()-he)/(hf-he))*(tf-te)+te;}//(-1)*4.2599998;}
  if (height()>=hf&&height()<hg){tempout=((height()-hf)/(hg-hf))*(tg-tf)+tf;}//(-1)*5.62;}
  if (height()>=hg&&height()<hh){tempout=((height()-hg)/(hh-hg))*(th-tg)+tg;}//(-1)*6.24;}
  if (height()>=hh&&height()<hi){tempout=((height()-hh)/(hi-hh))*(ti-th)+th;}//(-1)*5.5699997;}
  if (height()>=hi&&height()<hj){tempout=((height()-hi)/(hj-hi))*(tj-ti)+ti;}//(-1)*5.2799997;}
  if (height()>=hj&&height()<hk){tempout=((height()-hj)/(hk-hj))*(tk-tj)+tj;}//(-1)*4.9;}
  if (height()>=hk&&height()<hl){tempout=((height()-hk)/(hl-hk))*(tl-tk)+tk;}//(-1)*4.08;}
  if (height()>=hl&&height()<hm){tempout=((height()-hl)/(hm-hl))*(tm-tl)+tl;}//(-1)*3.42;}
  if (height()>=hm&&height()<hn){tempout=((height()-hm)/(hn-hm))*(tn-tm)+tm;}//(-1)*5.56;}
  if (height()>=hn&&height()<ho){tempout=((height()-hn)/(ho-hn))*(to-tn)+tn;}//(-1)*8;}
  if (height()>=ho&&height()<hp){tempout=((height()-ho)/(hp-ho))*(tp-to)+to;}//(-1)*8.6;}
  if (height()>=hp&&height()<hq){tempout=((height()-hp)/(hq-hp))*(tq-tp)+tp;}//(-1)*8.6;}
  if (height()>=hq&&height()<hr){tempout=((height()-hq)/(hr-hq))*(tr-tq)+tq;}//(-1)*9;}
  if (height()>=hr&&height()<hs){tempout=((height()-hr)/(hs-hr))*(ts-tr)+tr;}//(-1)*9;}
  if (height()>=hs&&height()<ht){tempout=((height()-hs)/(ht-hs))*(tt-ts)+ts;}//(-1)*3.25;}
  if (height()>=ht&&height()<hu){tempout=((height()-ht)/(hu-ht))*(tu-tt)+tt;}//(-1)*2.3899999;}
  if (height()>=hu&&height()<hv){tempout=((height()-hu)/(hv-hu))*(tv-tu)+tu;}//(-1)*3.31;}
  if (height()>=hv&&height()<hw){tempout=((height()-hv)/(hw-hv))*(tw-tv)+tv;}//(-1)*4.71;}
  if (height()>=hw&&height()<hx){tempout=((height()-hw)/(hx-hw))*(tx-tw)+tw;}//(-1)*9.5;}
  if (height()>=hx&&height()<hy){tempout=((height()-hx)/(hy-hx))*(ty-tx)+tx;}//(-1)*5.8;}
  if (height()>=hy&&height()<hz){tempout=((height()-hy)/(hz-hy))*(tz-ty)+ty;}//(-1)*9;}
  if (height()>hz){tempout=tz;
  //cout<<"the payload has breached the maximum hieght of the temperature profile..... fix this to get a
  }
}


       if(parameter_choice2==2){

       if (alt<=turnover1)
                        {     tempout=0;
                              t1= (groundtemp)+ (height()*((temp1-groundtemp)/(turnover1)));
                              tempout=t1;
                        }

       if (alt>turnover1 && alt<=turnover2)
```

```
                        {tempout=0;
                         t2= temp1+(alt*((temp2-temp1)/(turnover2-turnover1)));
                         tempout=t2;}

       if (alt>turnover2 && alt<=turnover3)
                        {tempout=0;
                        double m=((temp3-temp2)/(turnover3-turnover2));
                         t3= (temp2-(turnover2*m))+(alt*m);
                         //cout<< "x is "<<alt<<" y is "<<t3<<"m is "<<m<<""<<endl;
                         tempout=t3;}

       if (height()<0 || height()>turnover3)
                        {cout<< "error... too high or below 0" <<endl;}
}
return tempout;
}

double force::nmoles()
{      double n= ((buoy())*1.38e-23)/(ma*8.31);

       //double n=(groundpressure*groundvol)/(8.31*groundtemp);
return n;
}


double force::pressure()

{double p,tbase1, tbase2, tbase3, pbase1, pbase2, pbase3;
      if (parameter_choice3==1){
                            if (height()<=11000){double a=(288.15)/(288.15+(-0.0065*height()));
                                                 double b=(9.81*0.028964)/(8.31*(-0.0065));
                                                 p=101325*pow(a,b);
                                                 }
                            if(height()>11000 && height()<=20000){p=22632*exp((-1)*((height()-11000)*

                             if (height()>20000&& height()<=32000){       double a=(216.65)/(216.65+
                                                                          double b=(9.81*0.028964)/(8
                                                               p=5474*pow(a,b);}
                             if (height()>32000&& height()<47000){  double a=(228.65)/(228.65+(0.0028
                                                                          double b=(9.81*0.028964)/(8
                                                               p=868*pow(a,b);}
                     //else{//cout<<"payload either too high or conditions not met"<<endl;
                          //p=groundpressure*exp((-1)*(alt/(29.2467*groundtemp)));
                         //}
       return p;}

       if(parameter_choice3==2){       if(height()==0){p=pa;}
  if (height()>0&&height()<ha){p=pa;}
  if (height()>=ha&&height()<hb){p=((height()-ha)/(hb-ha))*(pb-pa)+pa;}//-3.04
  if (height()>=hb&&height()<hc){p=((height()-hb)/(hc-hb))*(pc-pb)+pb;}//(-1)*2.77;}
  if (height()>=hc&&height()<hd){p=((height()-hc)/(hd-hc))*(pd-pc)+pc;}//(-1)*2.38999;}
```

```
  if (height()>=hd&&height()<he){p=((height()-hd)/(he-hd))*(pe-pd)+pd;}//(-1)*2.52;}
  if (height()>=he&&height()<hf){p=((height()-he)/(hf-he))*(pf-pe)+pe;}//(-1)*4.2599998;}
  if (height()>=hf&&height()<hg){p=((height()-hf)/(hg-hf))*(pg-pf)+pf;}//(-1)*5.62;}
  if (height()>=hg&&height()<hh){p=((height()-hg)/(hh-hg))*(ph-pg)+pg;}//(-1)*6.24;}
  if (height()>=hh&&height()<hi){p=((height()-hh)/(hi-hh))*(pi-ph)+ph;}//(-1)*5.5699997;}
  if (height()>=hi&&height()<hj){p=((height()-hi)/(hj-hi))*(pj-pi)+pi;}//(-1)*5.2799997;}
  if (height()>=hj&&height()<hk){p=((height()-hj)/(hk-hj))*(pk-pj)+pj;}//(-1)*4.9;}
  if (height()>=hk&&height()<hl){p=((height()-hk)/(hl-hk))*(pl-pk)+pk;}//(-1)*4.08;}
  if (height()>=hl&&height()<hm){p=((height()-hl)/(hm-hl))*(pm-pl)+pl;}//(-1)*3.42;}
  if (height()>=hm&&height()<hn){p=((height()-hm)/(hn-hm))*(pn-pm)+pm;}//(-1)*5.56;}
  if (height()>=hn&&height()<ho){p=((height()-hn)/(ho-hn))*(po-pn)+pn;}//(-1)*8;}
  if (height()>=ho&&height()<hp){p=((height()-ho)/(hp-ho))*(pp-po)+po;}//(-1)*8.6;}
  if (height()>=hp&&height()<hq){p=((height()-hp)/(hq-hp))*(pq-pp)+pp;}//(-1)*8.6;}
  if (height()>=hq&&height()<hr){p=((height()-hq)/(hr-hq))*(pr-pq)+pq;}//(-1)*9;}
  if (height()>=hr&&height()<hs){p=((height()-hr)/(hs-hr))*(ps-pr)+pr;}//(-1)*9;}
  if (height()>=hs&&height()<ht){p=((height()-hs)/(ht-hs))*(pt-ps)+ps;}//(-1)*3.25;}
  if (height()>=ht&&height()<hu){p=((height()-ht)/(hu-ht))*(pu-pt)+pt;}//(-1)*2.3899999;}
  if (height()>=hu&&height()<hv){p=((height()-hu)/(hv-hu))*(pv-pu)+pu;}//(-1)*3.31;}
  if (height()>=hv&&height()<hw){p=((height()-hv)/(hw-hv))*(pw-pv)+pv;}//(-1)*4.71;}
  if (height()>=hw&&height()<hx){p=((height()-hw)/(hx-hw))*(px-pw)+pw;}//(-1)*9.5;}
  if (height()>=hx&&height()<hy){p=((height()-hx)/(hy-hx))*(py-px)+px;}//(-1)*5.8;}
  if (height()>=hy&&height()<hz){p=((height()-hy)/(hz-hy))*(pz-py)+py;}//(-1)*9;}
  if (height()>hz&&height()<50000){p=((height()-hz)/(50000-hz)*(1-pz)+pz);}
  if (height()>50000){cout<<"payload is too high to simulate any further"<<endl;
                                    p=0;}

  //pz-(((0.75*height())-hz)*pz);



      return p*100; }

}

double force::volume()

{double vhel=0;

if (burst==0){vhel= (nmoles()*8.31*temp())/pressure();}// replaced temp with ground themp here for testi
if (vhel>=burstvol){burst=1;
                    vhel=0;}
if (burst==1){vhel=0;}
return vhel;
}
double force::Area()
{
       double radius= pow(((volume()*3)/(3.141592654*4)),(1/3));
       double ab= 3.141592654* pow(radius,2);
       return ab;
}
```

```
void force::fvelocityassign()
{
if (abs(Ftotal())>=0.00001){ fvelocity= sqrt((ivelocity*ivelocity)+(2*((Ftotal()/mass()))*s));}
//if (abs(Ftotal())>=0.000000001){ fvelocity= sqrt((ivelocity*ivelocity)-(2*(abs(Ftotal()/mass))*s));}
else {fvelocity=ivelocity;}
}

void force::velocity_update()

{ivelocity=fvelocity;}



double force::time()
{double u=(ivelocity);
 double v=(fvelocity);
 time_step=(((mass())*(v-u))/Ftotal());
initial_time=initial_time+time_step;
//cout<< "v is: "<<v<<" and u is: "<<u<<" k:"<<k<<" time: "<<timex<<" "<<endl;
//cout<<"time step"<<time_step<<""<<endl;
return initial_time;
}

double force::increment()
{alt=alt+s;

return alt;

}

double force::decrease()
{
alt=alt-s;

return alt;

}

double force::mass()
{      double tmass;
       if (burst==0)
       {tmass=boxmass+(nmoles()*4*1.67E-27*6.023E23)+balloonmass;
       return tmass;}

       if (burst==1){tmass=1.6+balloonmassburst;
       return tmass;}
       else {cout<<"somethings gone wrong 1"<<endl;
       return -100000;}
       }
```

```
double force::height()const

{return alt;}

double force::cd(){if (burst==1){return cd2;}
                       else{ return cd1;}
                       }
```