Design of a Stratospheric UAV Flight Simulator

Author:
Si Kai LEE

Supervisor:
Dr András SÓBESTER

This report is submitted in partial fulfilment of the requirements for the Degree of Bachelor of Engineering, Faculty of Engineering and the Environment, University of Southampton.

Individual Project Academic Integrity Statement

Declaration

I, the undersigned, confirm that all the material presented in this Individual Project report is my own work. References to, quotations from and the discussion of the work of any other person have been correctly acknowledged or cited within the report in accordance with University guidelines on Academic Integrity.

Signed: .....
Name: Si Kai Lee

Date: 26th March 2014

Acknowledgements

I would like to thank Dr András Sóbester, my supervisor, for providing me with the resources and advice that enabled me to complete the project.

I would also like to thank Dr Adam Lewis, Dr Oliver Brend, Dr Sarp Akcay and Charlie Mehta for proof-reading this paper, as well as Ryu Fattach who provided the LaTeX template.

The Windows distribution of the UAV Flight Simulator package was provided courtesy of Michael Sansoni.

Abstract

Atmospheric science and meteorology require information on the changing properties of different masses of air to maintain accurate Numerical Weather Prediction (NWP) models. The most common method of obtaining such data is by releasing radiosonde-attached high altitude gas balloons that record ambient conditions as they rise. Unfortunately, the recovery rate of radiosondes is poor which imposes a large cost on meteorological agencies and atmospheric research groups. A possible alternative to radiosondes could be unpowered Unmanned Aerial Vehicles (UAVs) carried by high altitude gas balloons that glide towards a designated heading or location after the balloons pop.

This paper builds the case for the replacing radiosondes with unpowered UAVs that house weather instruments and describes the process of designing a simulator predicting stratospheric UAV glide operations to demonstrate the feasibility the above argument.

List of Abbreviations

AGL	Above Ground Level
GFS	Global Forecast System
GUI	Graphical User Interface
NTNS4	North Texas Near Space 4
NWP	Numerical Weather Prediction
ODE	Ordinary Differential Equation
Re	Reynolds number
SUMO	Small Unmanned Meteorological Observer
TS1	Simulation time step for altitudes above 100m
TS2	Simulation time step for altitudes below 100m
UAV	Unmanned Aerial Vehicle

List of Symbols

α	Glide angle
ρ	Air density
θ	Heading of UAV
θ_W	Direction of wind
C_D	Coefficient of drag
C_L	Coefficient of lift
D	Drag
L	Lift
L/D	Lift to drag ratio
GR	Glide ratio
GR_E	Glide ratio in Earth reference frame
R	Range
S	Wing planform area
V_{GS}	Ground speed
V_{SR}	Sink rate
V_{xy}	Combined airspeed in x and y directions
V_T	True airspeed
$V_{W_{xy}}$	Combined wind speed in x and y directions
V_{W_z}	Wind speed in z direction
V_z	True sink rate
W	Weight
a	Current x coordinates
b	Current y coordinates
c	Reference x coordinates
d	Reference y coordinates
g	Gravitational acceleration
t	Time
x	Distance in longitudinal direction
y	Distance in latitudinal direction
z	Altitude

Contents

Number of Words: 9900

1 Introduction

With the proliferation of miniature sensors and advanced communication technologies, the use of unmanned vehicles have entered public consciousness. However, most are unaware of the role radiosondes play in collecting weather data. This paper begins with a brief on UAVs and radiosondes before presenting the argument for using unpowered UAVs to replace radiosondes.

UAVs are unmanned aircraft that are remotely controlled or autonomous. UAVs carry different payloads such as cameras, scientific instruments or armaments for different missions. Many hobbyists use UAVs to conduct their own operations, proving that UAVs can be readily acquired at low cost and easily configured to suit various purposes. A Textron Systems Aerosonde UAV used for scientific research is shown in Figure 1.

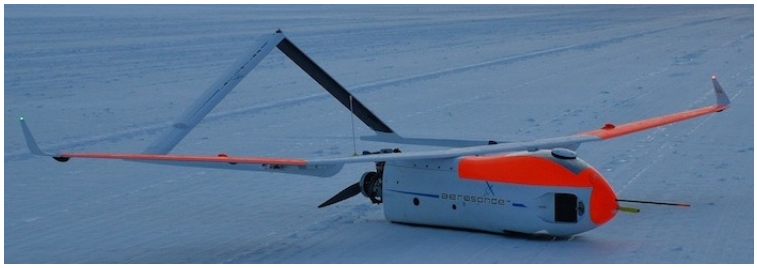


Figure 1: A Textron Systems Aerosonde UAV. [1]

A radiosonde, shown in Figure 2, is a disposable container that houses weather instruments. It is usually suspended below a weather balloon as shown in Figure 3.

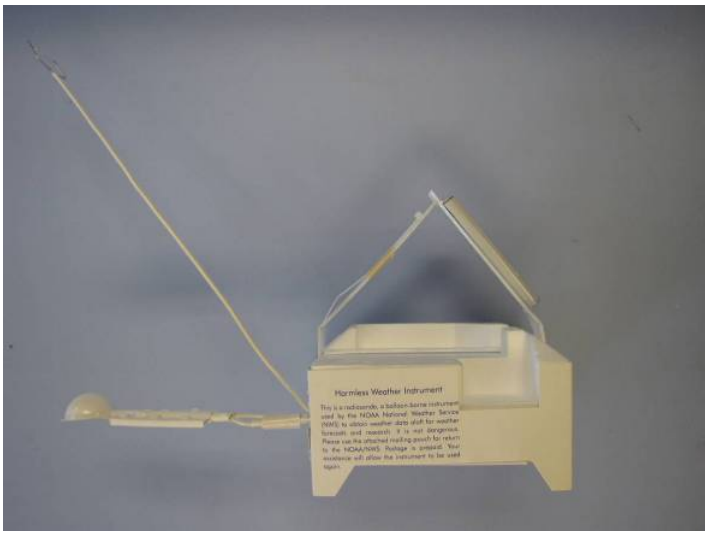


Figure 2: A radiosonde. [2]

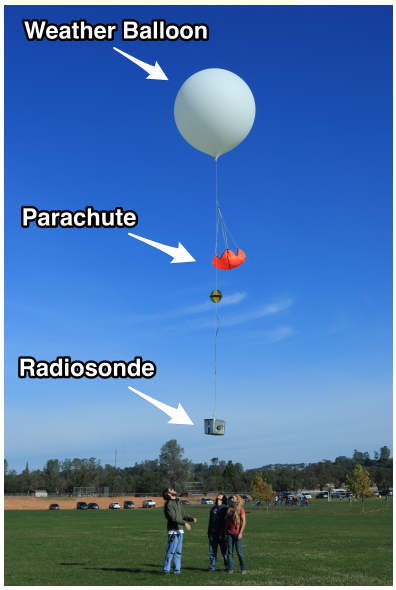


Figure 3: A weather balloon suspending a radiosonde. [3]

Radiosondes are used to measure pressure, temperature and other atmospheric properties and transmit collected data during ascent to weather stations for analysis [2]. They track the position of the weather balloons that they are attached to and use this information to determine local wind speed and direction [2]. Once the weather balloon bursts, data transmissions cease [2].

Each radiosonde costs around $200 [4]. According to the United States' National Weather Service, less than a fifth of the approximately 70,000 radiosondes released annually are recovered for reconditioning [2]. Lost radiosondes cost at least

$$\$200 \cdot 70,000 \cdot 0.8 = \$11,200,000$$

per year in the United States alone. Since the United States only has 92 out of the 800 upper-air observation stations across the globe [2], the total estimated cost of lost radiosondes could be

$$\$11,000,000 \cdot 8 = \$88,000,000$$

per year assuming radiosondes costs and recovery rates are similar worldwide. To reduce the operating costs of meteorological agencies and atmospheric research groups that releases the radiosondes, this paper proposes to house weather instruments in unpowered UAVs instead of radiosondes. Such UAVs could be constructed out of expanded polypropylene or similar materials to house the instruments [5]. Instrument probes would then be placed in cut outs in the nose of the UAV to ensure accurate data collection.

The use of unpowered UAVs in place of radiosondes would reduce trajectory planning errors by:

- Eliminating drag uncertainties associated with partial parachutes deployment as parachutes would no longer be required [6];
- Reducing the influence of the burst weather balloon on the aerodynamics of the balloon-UAV system during descent as UAVs would be more controllable and heavier than radiosondes [6].

Most importantly, the UAVs could be programmed to land in predetermined areas which are determined beforehand. Knowing that the UAVs are likely to be in such areas would aid the search process and improve the possibility of recovery, helping to reduce the costs of missing UAVs. The reduced likelihood of damage to the weather instruments and increased chances of recovery could make the use of more sensitive instruments cost effective. These advantages form a compelling argument for the use of unpowered UAVs over radiosondes.

The possible disadvantages of using UAVs in place of radiosondes are:

- Increased cost per flight as a UAV, including its communications system, is more expensive than a radiosonde housing and require a bigger weather balloon containing more lighter-than-air gases to lift;
- Increased cost of loss incurred if UAV is lost.

Controlling the glide path of the UAV would allow data to be collected in a controlled and deliberate manner during descent, which would increase the number of useful data points collected. Furthermore, a glide path passing through areas of interest could be selected to enable additional data to be collected in those areas.

2 Aims and Objectives

This paper aims to investigate the possibility of replacing radiosondes with unpowered UAVs and demonstrate the ability of unpowered UAVs to glide to a set location.

To achieve the above aims, the following steps were taken:

1. Review prior research conducted;
2. Predict a possible landing location of a UAV that glides through a 2D wind field with constant wind speed and direction;
3. Predict a possible landing location of a UAV that glides through a series of 3D wind fields with constant wind speed and direction;
4. Integrate GFS, Linear4Interpolator and Weather modules from the precursor paper into the simulator;
5. Predict a possible glide path of a UAV dropped from stratospheric altitudes using current and forecast weather data using a simple physics model;
6. Predict a possible glide path of a UAV dropped from stratospheric altitudes using current and forecast weather data using a more realistic physics model;
7. Guide a UAV dropped from stratospheric altitudes to a desired landing siteusing current and forecast weather data using the more realistic physics model;
8. Package the above mentioned methods as separate simulators;
9. Provide a GUI for the simulators to aid users in entering inputs.

To extend the capability of the simulator, the option of plotting contours showing the maximum range of a UAV released from a certain height, referred to as flight contours in this paper, was added. Flight contours indicates the all the possible landing sites of a UAV released from a certain height and sets the limits for the areas searched during recovery.

The next section justifies the use of UAVs as possible replacements for radiosondes using prior research and gives an overview of existing literature on UAV computational modelling and trajectory planning.

3 Review of Existing Literature

3.1 Use of UAVs in Atmospheric and Meteorological Research

Although UAVs have been used in atmospheric science studies since the 1970s [7], prior research must be reviewed to ensure UAVs could adequately replace radiosondes.

The Small Unmanned Meteorological Observer (SUMO) was designed to be a recoverable radiosonde for atmosphere boundary research [5]. When SUMO soundings i.e. the measurement of physical properties at different altitudes were compared with Vaisala radiosonde soundings, deviations were minimal at altitudes above 150m Above Ground Level (AGL) [5]. But at altitudes lower than 150m AGL larger deviations were found [5]. These results were in agreement with the comparison of measurements collected by the Aerosonde UAV and Vaisala radiosondes [8]. Therefore, UAV soundings would probably provide accurate data for the most part as UAVs would be operating predominantly at altitudes where sounding deviations from radiosondes were known to be minimal. However, the UAVs mentioned above have all been powered UAVs operating at relatively low altitudes.

The capability of UAVs to operate in similar conditions as radiosondes (33,000m and higher) [9] was proven on 19 November 2013 when the North Texas Near Space 4 (NTNS4) landed successfully after being dropped from around 30,000m [10]. The use of UAVs in meteorological missions in Antartica [11] and missions involving 130km/hour winds [12] further confirms the ability of UAVs to operate in harsh conditions.

3.2 UAV Computational Modelling and Trajectory Planning

An investigation into work done on computational modelling and trajectory planning of UAVs was conducted to ensure there were no major overlaps with previous research and uncover potential new areas for development.

In the area of computational modelling, the articles reviewed were on building computational models of whole UAVs [13], [14], evaluating aerofoil performance at different Reynolds numbers (Re) [15], [16] and comparing computational models of UAVs [17].

Computational data obtained from modelling UAVs across a wide range of Re could provide key aerodynamic characteristics of the particular UAV such as coefficient of lift, parasite drag and induced drag that could help to build a more accurate physics model. Simulations that use such physics models could account for the

changes of the aerodynamic characteristics of the UAV as it descends and return more accurate predictions. By comparing the predicted glide paths of different UAVs, the UAV most suitable for the actual sounding could be chosen.

Several commercial options such as the Airborne Scientific Flight Planner [18], MaVinci Desktop [19] and Orbit Logic UAV Planner [20] provide detailed UAV trajectory planning. However, they were not suitable for predicting trajectories for unpowered UAVs as they do not consider different wind conditions in different masses of air. Moreover, the changes in atmospheric conditions during soundings would cause UAVs to deviate from programmed glide paths, rendering detailed trajectory planning unnecessary.

The current focus of academic research is on autonomous and evolutionary UAV path planning, [21], [22], path following [23], [24] and on control of multiple cooperating UAVs [25]–[27]. With suitable path planning and path following algorithms, UAVs could alter their headings autonomously in response to changes in weather conditions, ensuring that they would land close to the predicted landing sites. The quality of collected data could be improved by coordinating the trajectories of various UAVs to maximise coverage of the atmosphere.

This paper explores new territory as it seeks to provide a simple pre-launch trajectory planning tool for unpowered UAV operations at stratospheric altitudes that provides users an estimate of possible glide paths and landing sites to aid recovery. The following sections illustrates the design process of the simulators.

4 The Physics Model and Associated Errors

The starting point for building a simulator is the construction of the physics model that defines the object in the time and space of the simulation. In this paper, a relatively simplified aerodynamic model of the gliding UAV was selected to ensure that a working simulator was produced within the given time frame. The basic assumptions made are:

- The uAV was modelled as a flying wing with a point mass as key characteristics such as centre of gravity, pitching moments and aerodynamic centre could only be found if an actual UAV was acquired;
- The lift-to-drag ratio L/D and the coefficient of lift C_L used were the averaged L/D and C_L of the UAV as the values change with a and Re, hence the coefficient of drag C_D obtained from multiplying L/D by C_L was the average C_D of the UAV;
- The wing planform area S was the only size-related wing parameter defined.

4.1 Glide Dynamics

To construct the physics model for the simulator, it is crucial to first understand the dynamics of gliding aircraft. This section presents the basic equations used in the physics model of the simulator.

According to Figure 4, assuming steady conditions and zero wind, there is no net force on the aircraft that is travelling at a constant speed V_T.

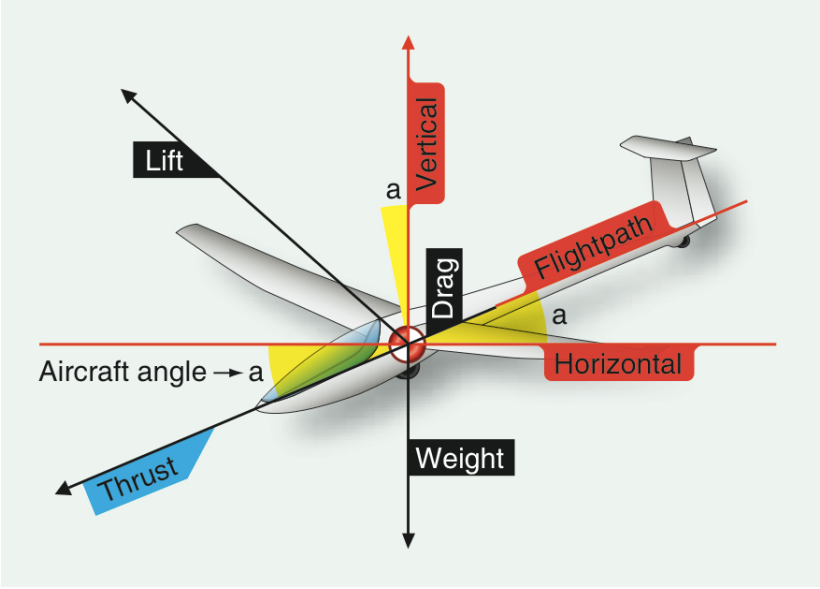


Figure 4: Free Body Diagram of Glider.
[28]

Applying free-body analysis, equilibrium is achieved when weight W is balanced by lift L and drag D [29]

$$L = W\cos a = \frac{1}{2}\rho V_T^2 S C_L, \tag{1}$$

$$D = W\sin a = \frac{1}{2}\rho V_T^2 S C_D, \tag{2}$$

$$a = \tan^{-1}\frac{1}{\frac{L}{D}}, \tag{3}$$

where V_T is the current velocity of the UAV and a is the glide angle (aircraft angle in Figure 4). W is commonly expressed as mass m multiplied by gravitational acceleration g.

From Equations (1) and (2), true airspeed V_T^2 is directly proportional to W as C_L and C_D are assumed to be fixed quantities. Taking the density of air ρ to be constant, L and D increases as W increases, but L/D stays fixed as only V_T^2 increases.

In gliding, L/D is assumed to be equal to the glide ratio GR. GR is defined as the combined airspeed in the x and y directions V_{xy} divided by the true sink rate V_z [28]

$$GR = \frac{L}{D} = \frac{V_{xy}}{V_z}, \tag{4}$$

where L/D and GR must be positive. V_z is defined as positive in the downwards z direction.

The distance travelled by the UAV r in the x and y directions can be obtained from multiplying GR by altitude z

$$r = GR \cdot z. \tag{5}$$

The endurance of the UAV, dictated by V_z, is maximised by minimising V_z. V_z is given by

$$V_z = -\frac{dz}{dt} = -V_T \sin\alpha, \tag{6}$$

where t is time. By increasing L/D and thus decreasing V_z, endurance is improved.

In gliding flight, the frame of reference is the Earth. The ground speed V_{GS} determines how fast the UAV glides relative to the ground and is represented as the sum of V_{xy} and the combined wind speed in the x and y directions $V_{W_{xy}}$ [28]

$$V_{GS} = V_{xy} + V_{W_{xy}}. \tag{7}$$

Similarly, the sink rate V_{SR} is the sum of the true sink rate V_z and the wind speed in the z direction V_{W_z} [28]

$$V_{SR} = V_z + V_{W_z}. \tag{8}$$

The heading θ of the UAV is the $\arctan$ of the ratio of ground speed in the y direction V_{GS_y} to ground speed in the x direction V_{GS_x}

$$\theta = \tan^{-1} \frac{V_{GS_y}}{V_{GS_x}}. \tag{9}$$

θ_W is the direction of the wind in the x and y plane.

Using θ, the x and y displacement are

$$x = r \cos \theta, \tag{10}$$

$$y = r \sin \theta. \tag{11}$$

Lastly, flight duration t is calculated by dividing z with V_{SR}

$$t = \frac{z}{V_{SR}}. \tag{12}$$

4.2 The Physics Model

The physics model is the set of equations that describe the motion of the UAV in the simulator. The initial physics model only considered $V_{W_{xy}}$, θ_W, z and L/D as the factors affecting the motion of the UAV. It did not consider the effect of changing ρ and assumed the UAV dropped at a constant V_z. However, the model did consider changes in wind speed and direction as the UAV descended.

The initial physics model calculated t and the distances travelled in the x and y directions r by:

- Obtaining V_z from Equation (4);
- Obtaining V_{GS} from Equation (7)with V_{xy} assumed to be 0;
- Obtaining t with Equation (12) and thus r by multiplying V_{GS} with t;
- Obtaining x and y from Equations (10) and(11).

As the above model did not account for the effect of changing ρ on L and D of the UAV, the prediction produced would be inaccurate. The inaccuracy was caused by the huge difference in the density of air between 30,000m and ground level. To account for the change in ρ, a more complex model using two ordinary differential equations (ODE) to solve for r and z was implemented.

Balancing L and W,

$$F = L - W\sin a, \tag{13}$$

$$F = \frac{1}{2}\rho V_T^2 S C_L - mg\sin a. \tag{14}$$

To solve for z:

$$V_T = \frac{\dot{z}}{\sin a}, \tag{15}$$

$$m\ddot{z} = \frac{1}{2}\rho\left(\frac{\dot{z}}{\sin a}\right)^2 SC_L - mg\sin a, \tag{16}$$

$$\ddot{z} = \frac{1}{2}\frac{\rho}{m}\left(\frac{\dot{z}}{\sin a}\right)^2 SC_L - g\sin a. \tag{17}$$

Balancing D and W,

$$F = D - W\cos a, \tag{18}$$

$$F = \frac{1}{2}\rho V_T^2 S C_D - mg\cos a. \tag{19}$$

To solve for r:

$$V_T = \frac{\dot{r}}{\cos a}, \tag{20}$$

$$m\ddot{r} = \frac{1}{2}\rho\left(\frac{\dot{r}}{\cos a}\right)^2 SC_D - mg\cos a. \tag{21}$$

$$\ddot{r} = \frac{1}{2}\frac{\rho}{m}\left(\frac{\dot{r}}{\cos a}\right)^2 SC_D - g\cos a. \tag{22}$$

As shown in Equations (13) to (22), the physics model had been expanded to include ρ, S, C_L, C_D, g and m. g is assumed to be 9.81m/s^2.

The boundary conditions used to solve Equation (17) were the current z and current V_z of the UAV while those used to solve Equation (22) were the current r and current V_{xy}. The GFS, Linear4DInterpolator and Weather modules from the precursor paper were used in conjunction to obtain ρ from weather forecasts produced by the National Oceanic and Atmospheric Administration's Global Forecast System (GFS). Numerical Python's odeint function was used to solve the above second order ODEs due to prior familiarity and its ease of use.

The ODE-based physics model calculated r and z by:

- Obtaining z by integrating Equations (15) and (17);
- Obtaining r by integrating Equations (20) and (22);
- Obtaining additional r due to wind by multiplying $V_{W_{xy}}$ by t;
- Obtaining the total r by adding the rs of the previous two steps;
- Obtaining x and y from Equations (10) and (11).

4.3 Errors

The discussions in 4.2 naturally leads to the sources of error associated with the model. These errors are the key metrics in defining the reliability of the simulations. The main aerodynamic assumptions made had been previously mentioned; the following are the main limitations associated with the ODE-based physics model used in the simulators:

- The initial stage of release, the final stage of landing and wind velocity in the z direction were not considered;
- The drag associated with changes in the wind were assumed to be negligible;
- The UAV was assumed to gliding stably in all weather conditions;
- The UAV can change its heading θ freely;
- All sources of drag other than lift-induced drag were ignored.

Although the above list is non-exhaustive, it provides a guide to the possible areas where the model can be improved. Some of these above-mentioned errors are addressed in 8.

As this paper is dependent on the modules produced by its precursor paper, it has inherited the uncertainties associated with those modules. The conditions experienced during the simulations can never be predicted perfectly as the weather system is more complex than any predictive mathematical model used by GFS [30]. Moreover, the ambient properties of the air masses that the UAV passes through are mostly predicted values of GFS data as the finest grid distributed by GFS, 0.5° latitude by 0.5° longitude by 47 vertical levels of the atmosphere by 3 hours of time, does not cover all times at all locations [30]. Hence, the Linear4DInterpolator module was used to produce estimates of wind speeds, wind direction and air densities via linear interpolation. This lack of grid resolution would also yield relatively large error margins.

After considering the uncertainties associated with the trajectory predictions, they are still useful as they provides plausible estimates of the landing sites of UAVs released from altitude. Such estimates would serve to reduce the search radii in retrieval operations should meteorological agencies and atmospheric research groups elect to use UAVs as radiosonde replacements.

5 UAV Flight Simulator

The trajectories are predicted by simulators implements physical models in virtual time and space. The translation and implementation of the physics models that describes the motion of the UAV at a particular time and space into code is the main product of the paper.

In this paper, the choice of programming language was limited to Python as the GFS, Global Tools, Linear4DInterpolator and Weather modules taken from the precursor paper were written in Python. Using other programming languages to interact with the Python objects and scripts would be less ideal as translating between languages would add another layer of complication.

Apart from the need to use Python modules to obtain GFS weather data, Python is an excellent choice of programming language for this paper as:

- Python's object-oriented nature relieves complexity by enabling more complex pieces of code to be built from less complex pieces;
- Python, in the form of Numerical Python, matplotlib and Scientific Python modules, is widely used and well-supported by the scientific community;
- Python is well-supported by general programming community, and hence possess many modules such as PySide that allow it perform simple tasks with other programming languages;
- Python provides many possibilities for further development, including the option to build a web framework that display the simulation results on a webpage or a GUI that gives a visual platform for users to interact with the command-line interface;
- Python, using the likes of PyInstaller and Py2Exe, is easily distributed between operating systems and can be run directly from distributed packages without prior installation.

Most modern applications have a GUI to make them user-friendly. Providing a GUI layer was made a key objective due to the large numbers of inputs required to run the simulators.

To facilitate the development process, most GUIs are built using application programming interfaces such as Qt, Tk, WxWidget. Qt was selected as the Python application programming interface to build the GUI. The reasons for using Qt are as follows:

- Qt has good cross-platform support and can run on most Windows and Unix-based operating systems;
- Qt provides QtCreator that provides a drag-and-drag interface for GUI design;
- Qt is widely used in application development;
- Qt provides a more modern-looking GUI;
- Python possesses existing modules in PySide and PyQt to facilitate Qt integration and user interaction programming.

As important as the GUI may seem, it is only a visual platform for the user to interact with the simulators. Without a proper system architecture to process the inputs and call the corresponding method, no simulations would be carried out. Figure 5 shows the system architecture in Unified Modelling Language.

The following is a short description of the system architecture:

- The interface, which is the GUI, passes user inputs to one of the four available simulators and displays the results of simulations;
- The BaseSimulator module applies the initial physics model described in the previous section and produces instances of the BaseSimulator class that simulates a UAV descending through a layer of the atmosphere and plots the descent of the UAV;
- The LayerSimulator module uses several instances of the Layer class to simulate a UAV descending through several atmospheric layers and plots the descent layer-by-layer;
- The GFS Data Simulator module applies the ODE-based physics model described in 4.2 and produces instances of the Flight_LatLon class that simulate a UAV descending from a certain height to a specific end point and saves the simulation results using the WriteFile module;
- The GFS Local Data Simulator module is similar to the GFS Data Simulator module but the instances of the Flight class simulate a UAV descending from a certain height along a certain direction.

Each module and its corresponding classes and methods listed in Figure 5 are discussed in greater detail in the upcoming subsections.

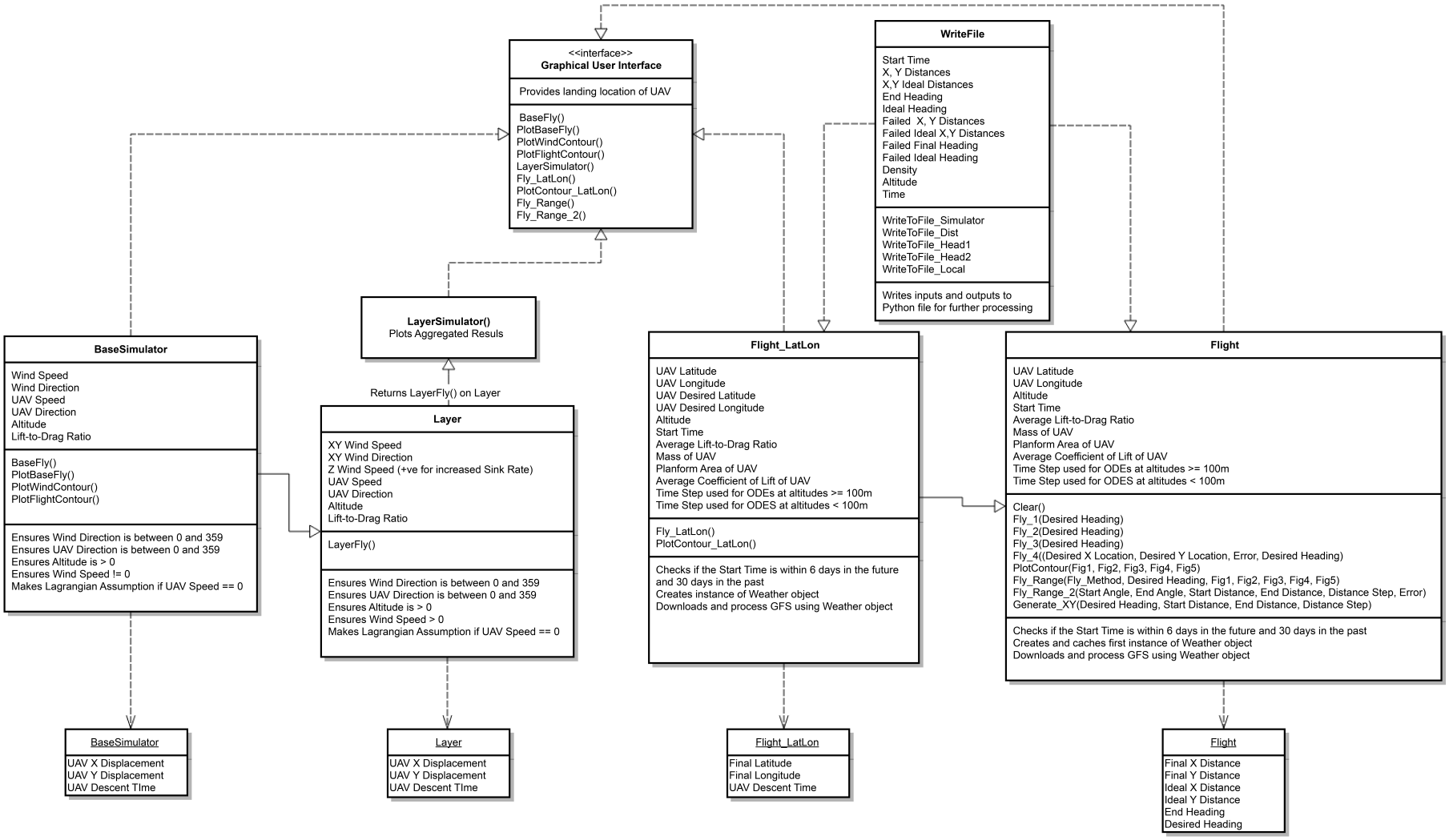


Figure 5: System Architecture.

5.1 BaseSimulator Module

The BaseSimulator module's BaseSimulator class was created to house all the variables and methods required to predict the landing site of a UAV through a single layer of the atmosphere modelled as a 2D wind field. The BaseSimulator module was the proof-of-concept design for a class-based simulator and is the foundation of the other simulator modules.

The BaseFly method returns r, the overall θ and t after taking $V_{W_{xy}}$, θ_W V_{xy}, θ, z and L/D as inputs by applying a variation of the initial physics model described in 4.2:

1. Assigns the wind field and UAV their specified speeds and direction;
2. Computes V_z from Equation (4) and finds t from Equation (12) assuming that V_{SR} = V_z;
3. Split the speed of the wind $V_{W_{xy}}$ into its x and y components by substituting r with $V_{W_{xy}}$ in Equations (10) and(11);
4. Split the speed of the UAV V_{xy} into its x and y components by substituting r with V_{xy} in Equations (10) and(11);
5. Obtain x by adding the x velocity components of the wind and UAV and multiplying the sum by t.
6. Obtain y by adding the y velocity components of the wind and UAV and multiplying the sum by t.
7. Calculate r by applying the Pythagoras' Theorem to x and y.
8. Determine the overall θ from Equation (9);

An important point to note is V_z must be positive. If V_z is not positive, the UAV would either float at the same altitude or rise above the layer, making the predicted trajectory invalid.

The PlotBaseFly method calls BaseFly and plots the starting and ending locations of the UAV and the 2D wind field.

The PlotWindContour method plots the displacement of the UAV by calling BaseFly for all possible integer wind angles (0° to 359°) with the remaining input variables held constant. On the other hand, the PlotFlightContour method plots the displacement of the UAV by calling BaseFly for all possible integer UAV headings (0° to 359°) with the remaining input variables held constant.

5.2 LayerSimulator Module

The LayerSimulator module builds upon the BaseSimulator module by simulating the descent of a UAV through several layers of the atmosphere by representing each atmospheric layer as an instance of the Layer class.

Each atmospheric layer is modelled as a 3D wind field in a Layer instance. This is achieved by making the Layer class an extended BaseSimulator class that takes wind speed in the z direction V_{W_z} as an additional input argument. However, Layer instances only have the LayerFly simulation method. The LayerFly method which is identical to the BaseFly method save for the fact it uses the full form of Equation (8) to calculate the sink rate of the UAV. However, the V_z must be positive and the wind speed in the z direction must be less than the V_z for the same reasons as mentioned in 5.1. Each successive Layer instance after the first inherits the V_{xy}, θ and L/D of the UAV was defined in the first instance and the position of the UAV. Therefore, the successive instances only require the $V_{W_{xy}}$, θ_W and V_{W_z}.

The LayerSimulator method is a wrapper function. It prompts users to define the number of layer desired and the required inputs for each layer. After creating the desired number of Layer instances and simulating the descent of UAV in each instance LayerSimulator aggregates and plots the values returned from the instances in a single diagram before returning the total displacement of the UAV, the heading of UAV based on its total displacement and the total descent time in a Python list.

The LayerSimulator method executes the following operations in chronological order:

1. Prompts the user for the number of atmospheric layers required for the simulation;
2. Creates the empty Python lists Layer inputs (L_I) and LayerFly outputs (LF_O);
3. Simulates the descent of a UAV through the number of atmospheric layers requested using a for-loop;
   (a) If no instances have been created:
      i. Prompts the user to input $V_{W_{xy}}$, V_{W_z}, θ_W V_{xy}, θ, z and L/D;
      ii. Stores user input in L_I;
      iii. Creates Layer instance using user input;
      iv. Runs LayerFly that returns r, θ, t;
      v. Stores returned values in LF_O;
   (b) After the first instance has been created:
      i. Prompts the user to input $V_{W_{xy}}$, V_{W_z}, θ_W and z;
      ii. Stores user input in L_I;
      iii. Creates Layer instance from user input and the speed, heading and L/D of the UAV obtained from L_I;
      iv. Runs LayerFly that returns r, θ, t;
      v. Stores returned values in LF_O;
4. Sums xs and ys of the UAV and ts produced by each layer using a for-loop;
5. Computes total r by applying Pythagoras' Theorem on the sums of x and y;
6. Calculates the overall θ from Equation (9);
7. Plots r layer-by-layer;
8. Returns total r, the overall θ and total t.

5.3 GFS Data Simulator Module

The GFS Data Simulator module uses GFS weather data to simulate the descent of a UAV from altitude. The Flight_LatLon class houses all the variables and methods to simulate and plot the descent of a UAV to a desired location, much like BaseSimulator and Layer. It generates a Fly_LatLon instance containing the simulation and plotting methods. Although the GFS Data Simulator module simulates the descent of UAV through multiple atmospheric layers, it does not represent each layer as a separate discontinuous instance as the downloaded GFS weather data defines the atmosphere as continuous layers which are best contained in a single Flight_LatLon instance.

Fly_LatLon, BaseFly and LayerFly, are methods that simulates the descent of a UAV. However, Fly_LatLon is much more complicated as it is based on the ODE-based physics model instead of the initial physics model. Using a variation of the ODE-based physics model enables Fly_LatLon to simulate a UAV sinking at different rates depending on ρ and V_T that are continuously changing. Fly_LatLon does not only consider the velocity of the UAV in the simulation process: it also considers the UAV's acceleration, making the trajectories predicted by Fly_LatLon much more accurate than those predicted by LayerSimulator.

The Fly_LatLon model assumes that a UAV accelerates from rest as it descends through the atmosphere. The trajectory of the UAV is affected by the local wind conditions as it descents through air masses modelled as a continuous 2D wind field. Wind velocity in the z direction is not considered as GFS does not directly provide it. The heading of the UAV is determined by a distance-based algorithm; this algorithm would be discussed in greater detail in 7.2.

The Fly_LatLon method prints the final latitude, the final longitude and the descent time of the UAV. A step-by-step illustration of the operations performed by Fly_LatLon is shown in Figure 6.

The base latitude and longitude in Figure 6 refers to the coordinates that a set of GFS weather data is centred on. Each set of GFS weather data covers a 3° latitude by 3° longitude column of air. Once the UAV leaves the area projected by the column of air, a new set of GFS weather data would be required. Intuitively, the initial base latitude and longitudes would be the input latitude and longitude.

Time Step 1 and Time Step 2 in Figure 6 represent the simulation time step for altitudes above 100m and altitudes below 100m respectively, where Time Step 1 should be larger than Time Step 2. The smaller Time Step 2 ensures that the simulation would stop at altitudes less than 1m above ground. The WriteToFile_Simulator method from the WriteFile module is used to write the data produced by the simulation to a Python file for further processing.

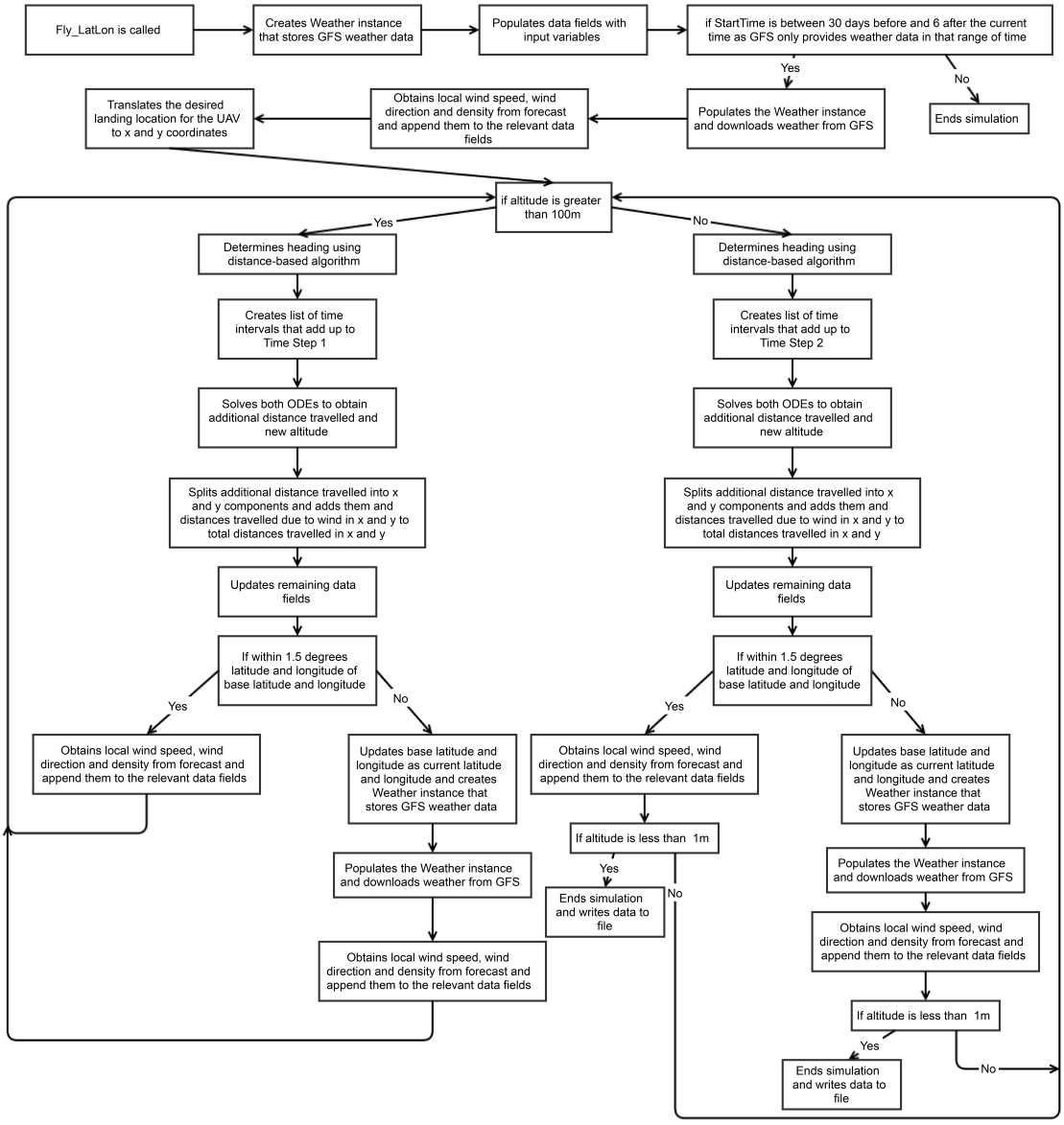


Figure 6: Fly_LatLon Method.

The PlotContour_LatLon method provides the option to plot the following graphs:

1. Density of air against time;
2. Altitude against time;
3. Density of air against altitude;
4. Predicted trajectory of the UAV in 2D;
5. Predicted trajectory of the UAV in 3D.

5.4 GFS Local Data Simulator Module

The GFS Local Data Simulator module, like the GFS Data Simulator module, simulates the descent of a UAV from altitude using GFS weather data. GFS Local Data Simulator, however, flies the UAV in a specific direction. The GFS Local Data Simulator module also improves on the performance of the GFS Data Simulator module for multiple simulations from the same starting points.

The methods available to an instance of the Flight class of the module are:

- Single Simulation:
    1. The Fly_1 method that uses a variant of the distance-based algorithm used in Fly_LatLon;
    2. The Fly_2 method that uses a heading-based algorithm;
    3. The Fly_3 method that uses a variant of the heading-based algorithm used in Fly_2;
    4. The Fly_4 method that is used specifically for Fly_Range_2;
- Multiple Simulations:
    1. The Fly_Range method used to run multiple simulations of Fly_1, Fly_2 and Fly_3 and plots the results of the simulations;
    2. The Fly_Range_2 method used to plot the limits of direction-following simulations by running multiple Fly_4 instances;
- The PlotContour method used to plot results of Fly_1, Fly_2 and Fly_3;
- The GenerateXY method used to generate distances for Fly_4.

The GFS Local Data Simulator module's performance advantage comes from the need to minimise the run times of multiple simulations. The run time of simulations using GFS weather data comprises mainly of the time required to download GFS weather data when a new Weather instance is created. When simulating UAV stratospheric operations, the UAV is released at an altitude of 30,000m. This usually results in at least 3 Weather instance creation events per simulation with each event taking between 2 to 4 minutes. Therefore conducting multiple simulations becomes extremely costly due to the time spent downloading GFS weather data.

When running multiple simulations with the same starting points, each simulation downloads the same initial set of GFS weather data. These repeated downloads can significantly increase the total time required especially when large numbers of simulations are conducted. To eliminate these repeat downloads, the Flight instance in GFS Local Data Simulator saves the Weather instance created for the starting point of the first simulation and reuses it for the rest of the simulations. This reduces the total run time of such simulations.

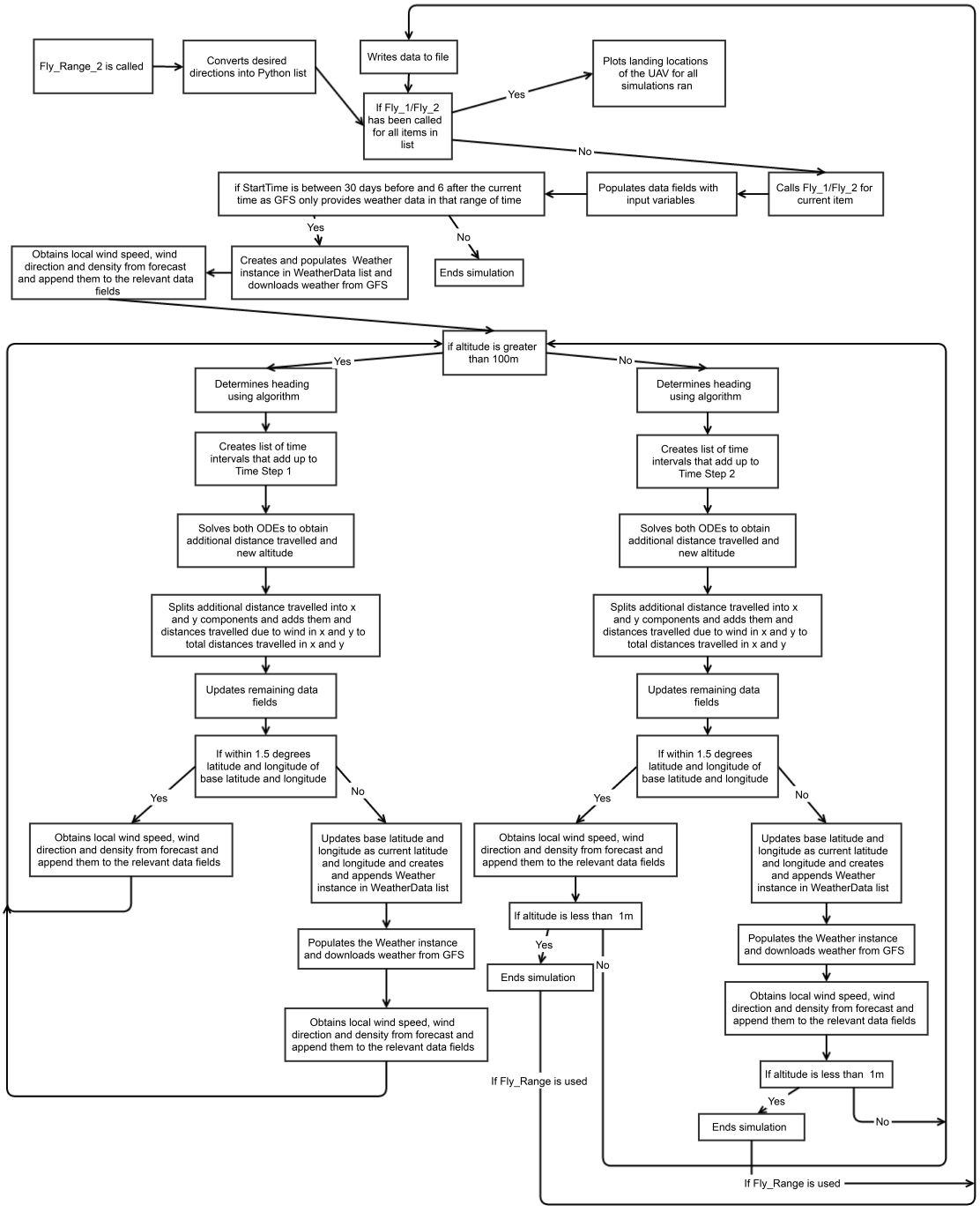


Figure 7: Fly_Range, Fly_1 and Fly_2 Methods.

The Fly_1, Fly_2, Fly_3 methods are largely similar to Fly_LatLon. These similarities can be seen in Figure 7. However, Fly_3 uses different heading algorithms for different altitudes and therefore was not featured in Figure 7 due to spatial constraints. At altitudes greater 60% and less than 25%of the input altitudes, Fly_3 uses Fly_2's heading-based algorithm to determine the heading of the UAV. For other altitudes, the heading of the UAV is set as the desired direction.

Fly_Range is a wrapper function that calls Fly_1, Fly_2 or Fly_3 multiple times for multiple simulations. The operations performed by Fly_Range are shown in Figure 7 together with those performed by Fly_1 and Fly_2. It then plots the flight contour using the predicted end landing sites.

The method Fly_4 assumes that the properties of the surrounding air are constant once the UAV leaves the area covered by GFS weather data. This assumption makes Fly_4 unsuitable for simulating stratospheric UAV operations as the ambient properties of the air varies vastly with altitude.

Fly_Range_2 flies the UAV to set points (generated by the Generate_XY method) across a range of directions. The set points are actually set areas as the input error margin specifies the maximum allowed deviation from the set points for both the x and y directions. The points that the UAV successfully reach are plotted in a flight contour diagram to demonstrate the capability of the UAV in flying in the user-defined directions. The use of Fly_4 is the key to reducing the total time taken for the large number of simulations required for generate flight contours as Fly_4 extends the area covered by GFS weather data indefinitely which eliminates the need for additional GFS weather data downloads. Fly_Range_2, Fly_4 and Generate_XY are represented in Figure 8.

A strategy to minimize the time taken for multiple simulations would be to run the simulator with differing inputs on different command-line windows to maximise the number of downloads cycles that can be carried out at any one time. This strategy works for GUI instances as each instance is based on a separate command-line window.

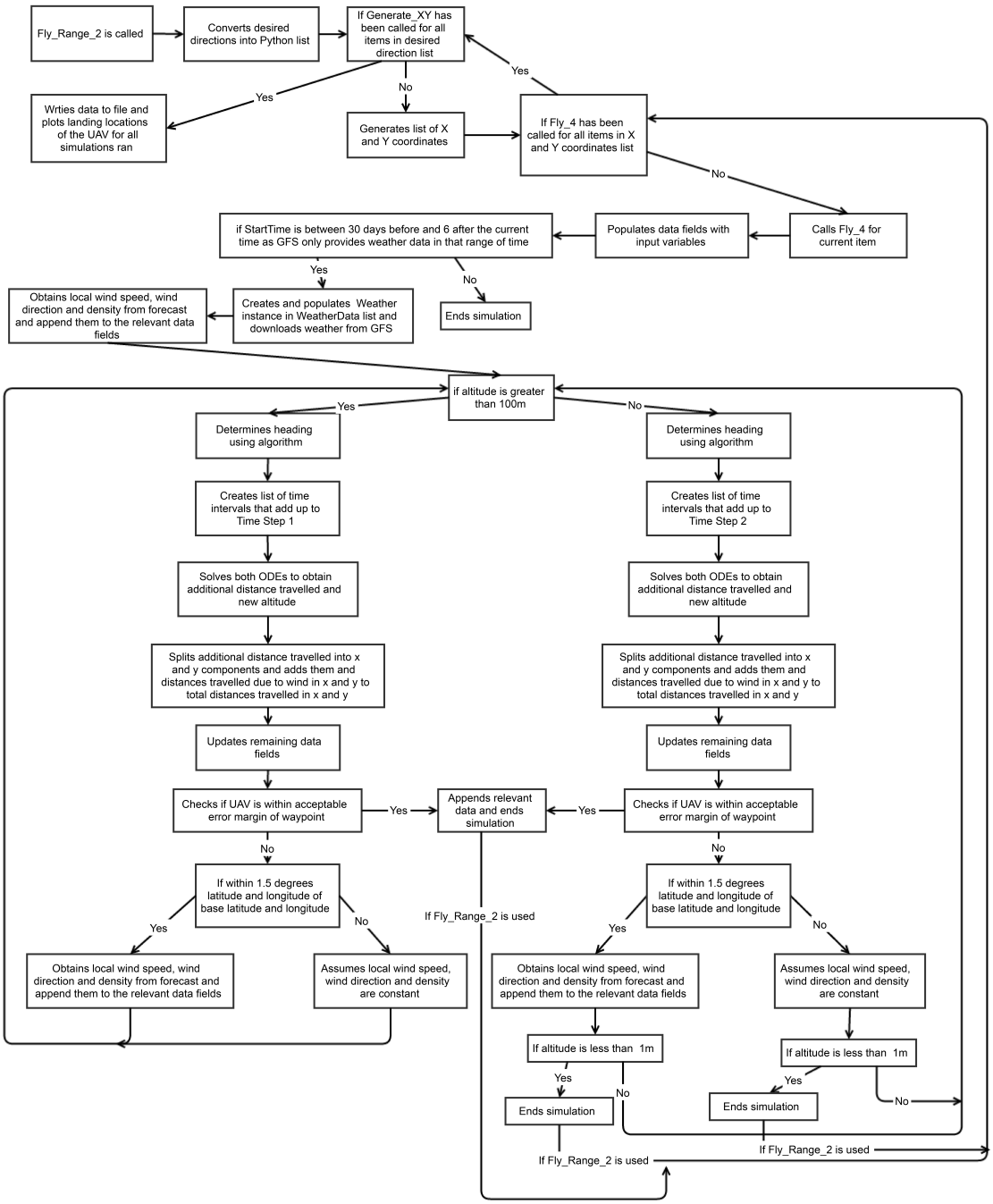


Figure 8: Fly_Range_2, Fly_4 and Generate_XY Methods.

5.5 WriteFile Module

The WriteFile module provides methods for the simulators using GFS weather data to write relevant data into Python files for further processing.

The methods are as follows:

- WriteToFile_Simulator used for Fly_LatLon;
- WriteToFile_Dist used for Fly_1;
- WriteToFile_Head1 used for Fly_2;
- WriteToFile_Head1 used for Fly_3;
- WriteToFile_Local used for Fly_4.

5.6 GUI Module

The GUI module is comprised of two Python scripts: gui.py and UAV_Flight_Simulator.py. As mentioned previously, the GUI was made using Qt, a cross-platform framework generator implemented via C++. To link the C++-based GUI to Python, PySide was used to provide the relevant Python bindings that translates Python-based commands to Qt.

The graphical elements of the GUI was built in Qt using the drag-and-drop QtCreater. It was then converted to Python using PySide-UIC. The generated gui.py script defines the graphical elements of the GUI in Python which is then linked to the user interaction elements of the GUI defined in UAV_Flight_Simulator.py. UAV_Flight_Simulator ensures that the inputs to the required fields are in the correct format before the user runs a simulation and calls the requested simulator when the inputs are verified. Figures 9 to 13 shows the tabs available on the GUI.

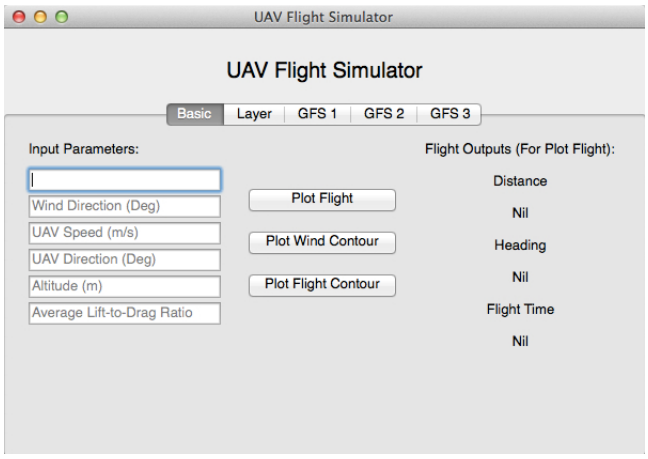


Figure 9: The tab for the BaseSimulator module.

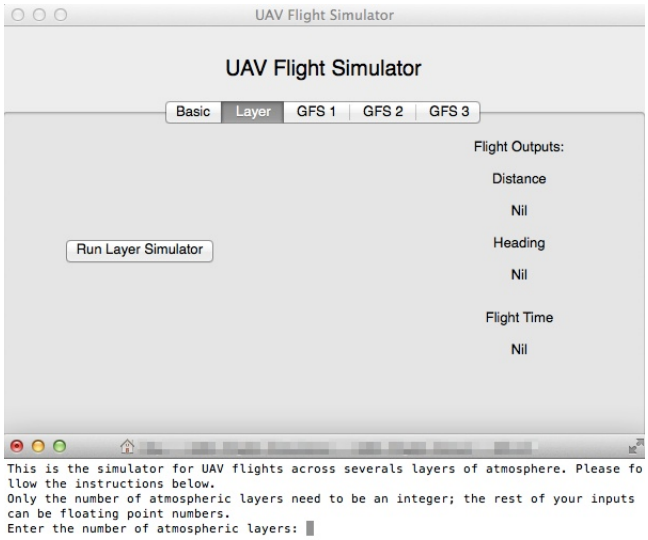


Figure 10: The tab for the LayerSimulator module and the corresponding command-line interface.

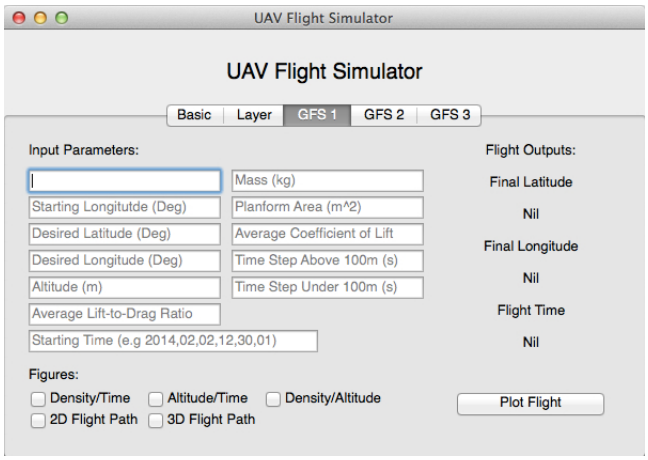


Figure 11: The tab for the GFS Data Simulator module.

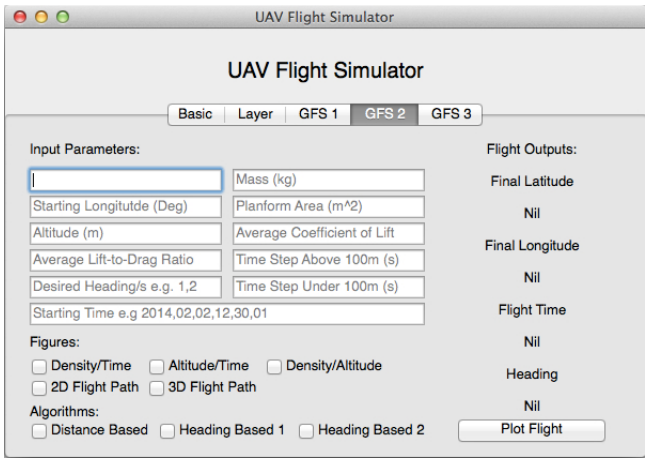


Figure 12: The tab for the Fly_Range method from the GFS Local Data Simulator module.

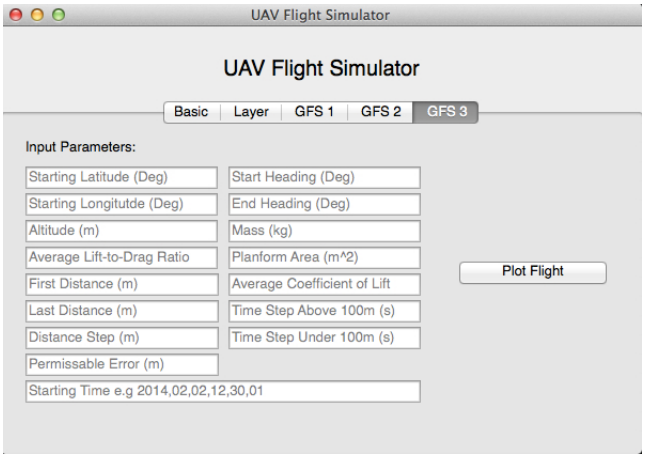


Figure 13: The tab for the Fly_Range_2 method from the GFS Local Data Simulator module.

5.7 Packaging and Distribution

Finally, PyInstaller was used to collect all the scripts and modules used in the paper such as Numerical Python, matplotlib and Scientific Python and the graphical elements used in the GUI and package them in one folder. The folder can be distributed across all OS X systems and the GUI can be run without Python and Qt installations. A Windows distribution generated by Py2Exe was also included with the paper.

6 Code Validation

For the simulator presented in this paper to be of value, the code used to construct these simulators has to be validated to ensure that the simulators functioned as advertised. The verification process is described in the upcoming subsections.

6.1 The BaseSimulator Module

The first method to be evaluated is BaseFly. BaseFly returns the r, θ and t. These values were visualised by PlotBaseFly in Figures 14 and 15.

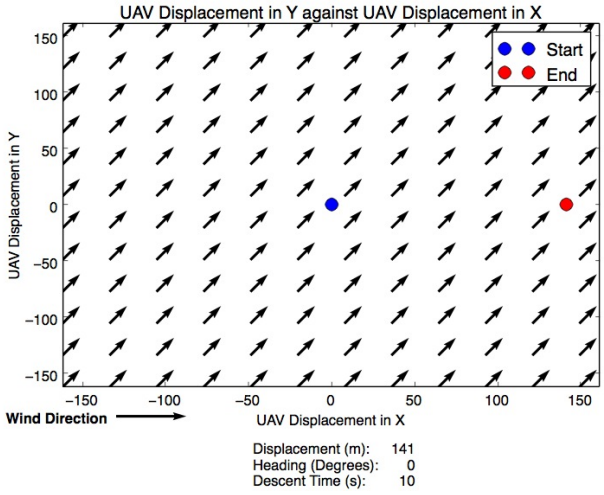


Figure 14: PlotBaseFly.
UAV flying at 10m/s and pointed at -45°, with the wind blowing in 45° at 10m/s.

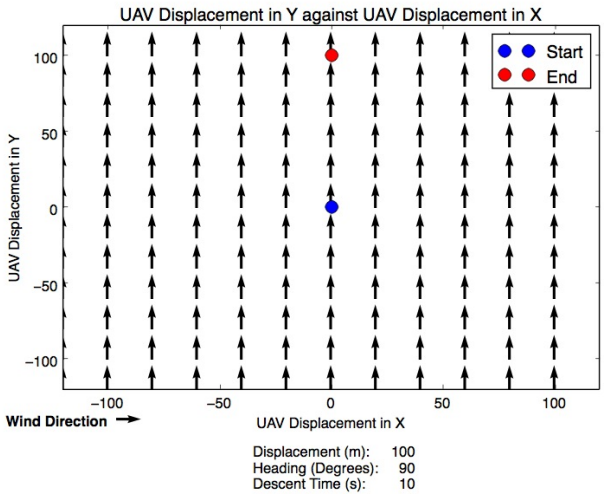


Figure 15: PlotBaseFly.
UAV flying at 10m/s and pointed at -90°, with the wind blowing in 90° at 20m/s.

Descent times of 10 seconds are given in both Figures 14 and 15. From Equation (4), V_z of 1 was obtained from the L/D (or GR) of 10 and the V_{xy} of 10m/s provided by the inputs. Using Equation (12), t was calculated to be 10 seconds as the UAVs sank at 1m/s from 10m, which matches the values shown in Figures 14 and 15.

Applying basic mathematics, a UAV flying at 10m/s at -45° in a wind field with wind blowing at 10m/s at 45° would have a final heading of 0° and travel 141m, which are shown in Figure 14. Similarly, a UAV flying at 10m/s at a heading -90° in a wind field with wind blowing at 20m/s at 90° would have a final heading of 90° and travel 100m as seen in Figure 15.

The results produced by the initial physics model in BaseFly tallies with the plots shown in Figures 14 and 15, therefore proving that BaseFly is valid.

PlotWindContour calls uses a loop to call BaseFly for all θ_W. Figures 16 and 17 were generated by PlotWindContour to demonstrate the validity of the code used to implement the loop.

The inputs provided for Figures 16 and 17 were the same apart from the 10m/s difference V_{xy}. When $V_{W_{xy}}$ was greater than V_{xy}, the starting point of the flights should be inside the displacement contour as the UAV does not possess sufficient speed to overcome the wind. This is illustrated in Figure 16. As V_{xy} is increased, the starting point would move closer to the displacement contour. When V_{xy} is equal to $V_{W_{xy}}$, the starting point would be on the displacement contour which is seen in Figure 17. This validates PlotWindContour.

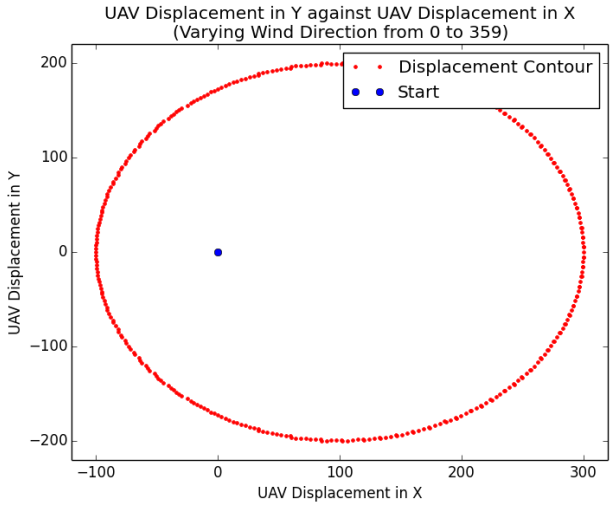


Figure 16: PlotWindContour.
UAV flying at 10m/s in wind blowing at 20m/s.

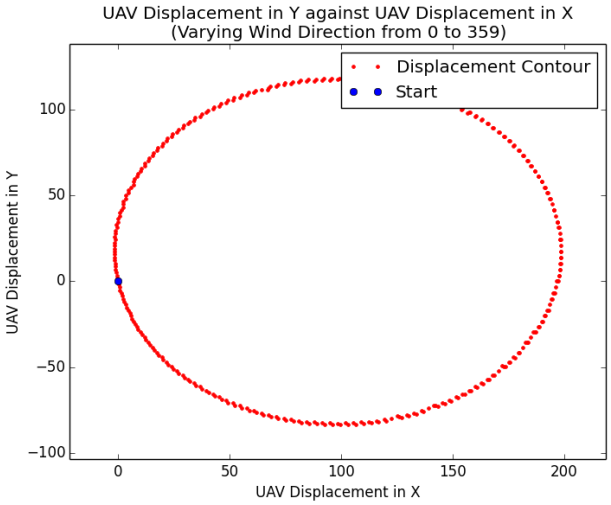


Figure 17: PlotWindContour.
UAV flying at 20m/s in wind blowing at 20m/s.

The verification process of the loop used in PlotFlightContour was similar to that of PlotWindContour as PlotFlightContour calls BaseFly for all θ.

The PlotFlightContour inputs used to generate Figures 18 and 19 were the same except for the 10m/s difference in V_{xy}. If the UAV possessed sufficient speed to overcome the wind, such as the case in Figure 19, the starting point of the flights would be on or within the displacement contour. When V_{xy} was decreased, the starting point would be out of the displacement contour as shown in Figure 18. This difference validates the plotting loop used in PlotFlightContour.

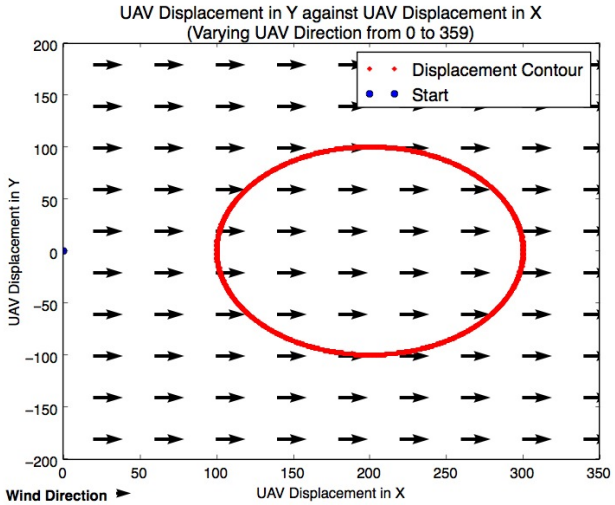


Figure 18: PlotFlightContour.
UAV flying at 10m/s in wind blowing at 20m/s.

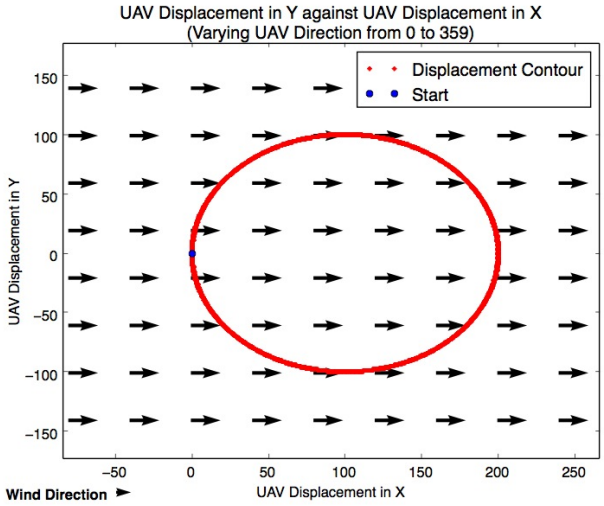


Figure 19: PlotFlightContour.
UAV flying at 20m/s in wind blowing at 20m/s.

6.2 The LayerSimulator Module

The Layer class of the LayerSimulator considered an additional factor in the form the wind speed in the z direction V_{W_z} which is set as positive upwards. To show the V_{W_z} was evaluated correctly, two layers with differing V_{W_z} were plotted in Figure 20. The first layer had a V_{W_z} of 0.5m/s while the second had a V_{W_z} of 0m/s. In Figure 20, the UAV had travelled twice as far in the first layer than the second; this showed that V_{W_z} has been properly implemented in Layer.

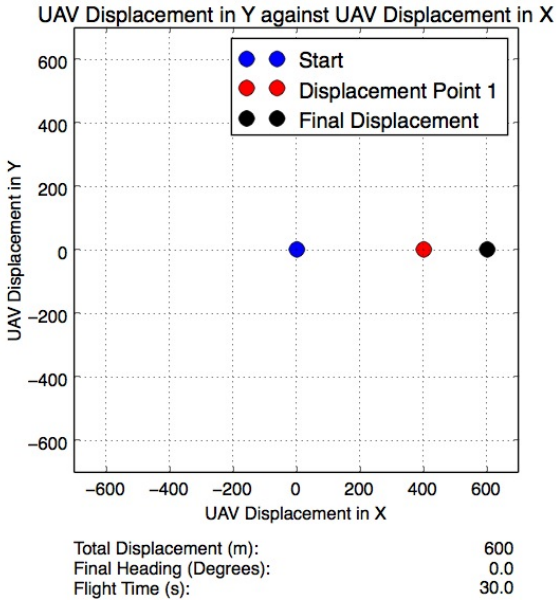


Figure 20: LayerSimulator with 2 Layers.
V_{W_z} is 0.5 in the first layer and 0 in the second.

The other piece of functional code added to LayerSimulator aggregated and plotted the simulation results of the different Layer instances. Figure 21 was plotted to demonstrate the implementation of the added code.

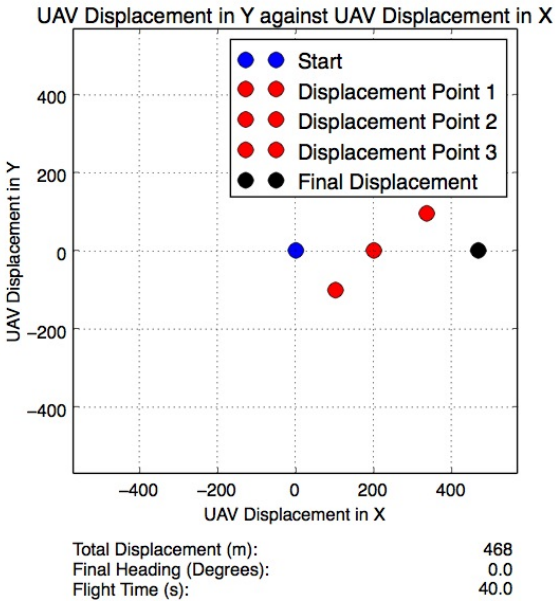


Figure 21: LayerSimulator with 4 Layers.
UAV with heading of -90° in the first layer, 90° in the second,
70° in the third and -70° in the fourth.

The UAV had a heading of 90° in the first layer, -90° in the second, 70° in the third and -70° in the fourth. The aggregation process was validated by the UAV having a final heading of 0° and covering different distances in different layers.

6.3 The GFS Data Simulator Module

To validate the Fly_LatLon method, the following trends must be reflected in Fly_LatLon simulations:

- ρ increases continuously and non-linearly with t;
- z decreases continuously and non-linearly with t;
- ρ decreases continuously and non-linearly with z;
- t is in the same order as the NTNS4 from 30,000m;
- V_{xy} peaks in the early in the flight and drops for the rest of the flight.

According to Figure 22, ρ experienced by the UAV increases non-linearly with t and showed no discontinuities which confirmed the method used to obtain ρ from GFS weather data was correctly implemented.

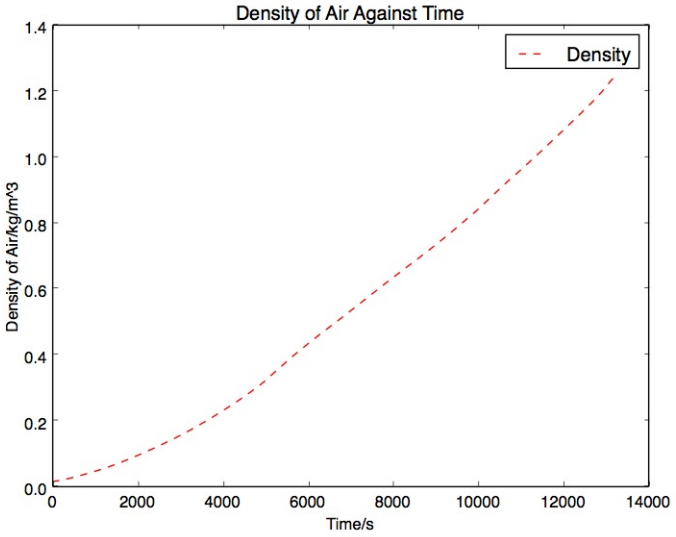


Figure 22: Plot of density of air against time.
Density continuously and non-linearly with time.

Figure 23 shows z decreasing continuously and non-linearly with t from 30,000m to around 0m, which demonstrates the error-free implementation of Equation 17.

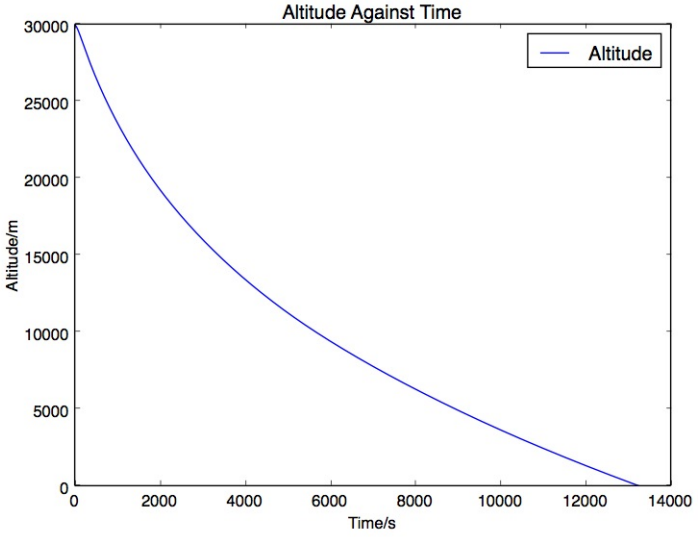


Figure 23: Plot of altitude against time.
Altitude decreases continuously and non-linearly with time.

The trend of ρ decreasing continuously and non-linearly with z can be seen from Figure 24. The relative density of the air at 30,000m is approximately 0.016, which is close to the value of 0.01503 quoted from the International Standard Atmo-

sphere [31]. This provides further evidence for the correct implementation of the two above-mentioned pieces of code.

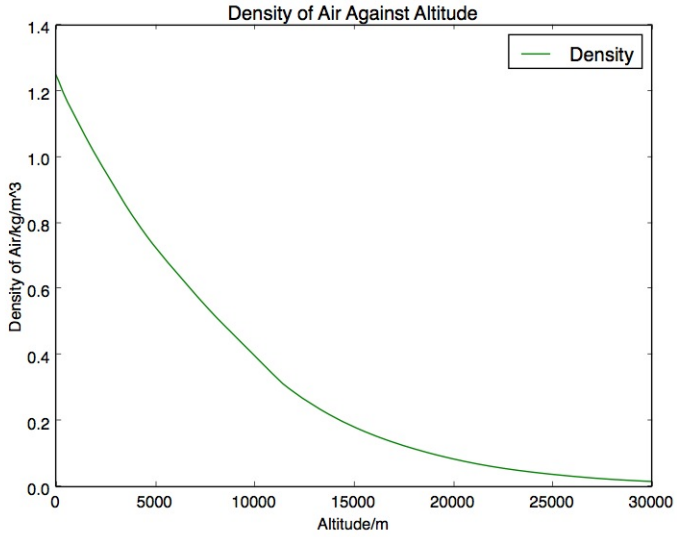


Figure 24: Plot of density of air against altitude.
Density decreases continuously and non-linearly with altitude.

When the UAV was released from 30,000m, t was approximately 3.68 hours while NTNS4 took 1.99 hours to descend from a similar altitude [10]. Although there is a significant difference in t, t was still within the range of values expected which indicate the results produced by Equation 17 is realistic. The two likely reasons for the difference in t are:

- The higher lift-to-drag ratio of the UAV;
- The larger wing planform area of the UAV.

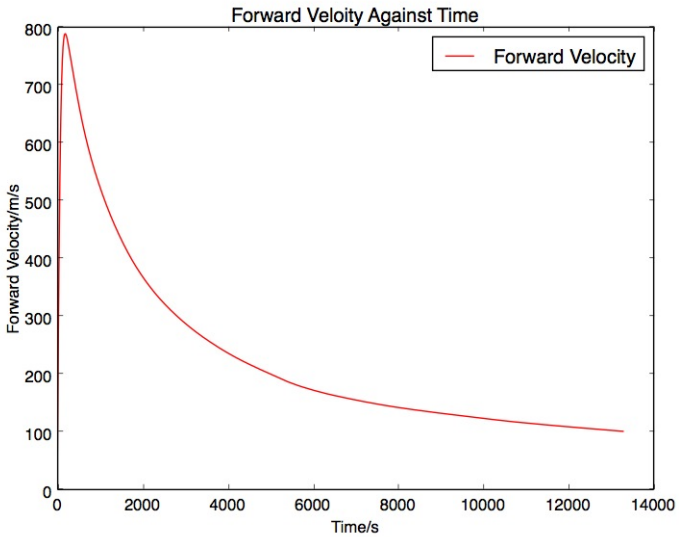


Figure 25: Plot of forward velocity against time.
Forward velocity peaks early on the flight and decreases from then on.

When a UAV is dropped from high altitudes, it would achieve maximum V_{xy} relatively early in its descent as the UAV experiences low drag when flying at such altitudes. After peaking, V_{xy} then drops throughout the flights to reflect the drag increase on the UAV caused by increasing air density. This trend is shown in Figure 25 which indicates the correct implementation of Equation 22. However, the maximum V_{xy} of nearly 800m/s is probably unphysical as the maximum velocity attained by NTNS4 in a dive was around 220m/s [10].

Fly_LatLon is valid as it reflects all the trends laid out at the beginning of 6.3.

6.4 The GFS Local Data Simulator Module

Fly_1, Fly_2 and Fly_3 are copies of Fly_LatLon that employ different heading algorithms so they could considered valid based on Fly_LatLon being valid.

Fly_4, on the other hand, assumes that the ambient conditions are constant once it leaves the area covered by GFS weather data and stops when the UAV reaches the set point. When Fly_4 is called with the same input as specified in the Fly_LatLon simulations for a set point of 5,000m at 45° and an error margin of 150m, the simulation stops at 14728m. When the set distance is increased to 2,500,000m ρ stays constant when the UAV leaves the area covered by the GFS weather data which can be seen in Figure 26. These indicate that the code used in Fly_4 was implemented correctly.

To ensure that the loops in Fly_Range and Fly_Range_2 are valid, the length of the lists written to the Python file storing all the results were compared to the number of simulations ran. When Fly_Range is called for directions from 0° to 359°, there were 360 items in per list as expected. Similarly, when Fly_Range_2 was called for directions from 0° to 359° with 18 set points for each direction, each list in the Python file contained 6480 items. Theses values proved that the loops employed in both methods were valid.

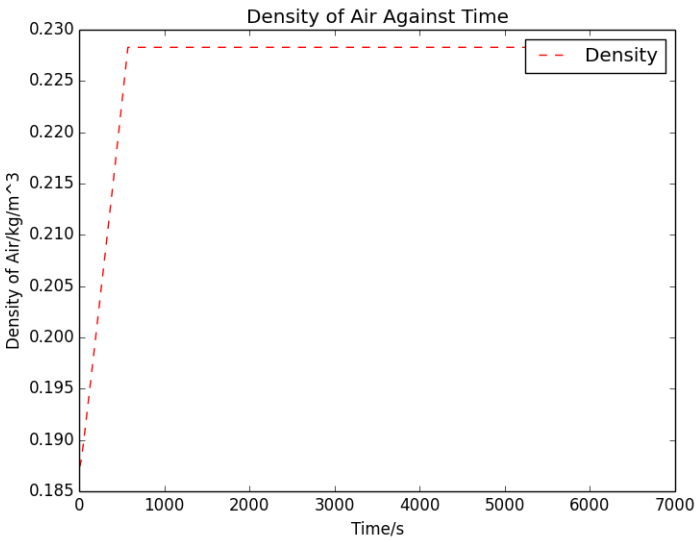


Figure 26: Plot of density of air against time.
Density stops increasing after the UAV leaves the area covered by GFS weather data.

7 Commentary and Discussion

This section includes a discussion on the choice of algorithms used and a commentary on the data produced by various simulations. The only modules considered in this section are the GFS Data Simulator and GFS Local Data Simulator modules that predict UAV glide trajectories using GFS weather data.

7.1 Simulator Inputs

Using realistic input variables for the GFS Data Simulator and GFS Local Data Simulator modules are crucial for accurate predictions. This subsection explains the choice of input variables used for the simulations discussed in 7.2 and 7.4.

All the GFS weather data-based simulations require the following variables:

- Starting latitude;
- Starting longitude;
- Starting altitude;
- Starting time;
- Average lift-to-drag ratio L/D;
- Average coefficient of lift C_L;
- Mass of UAV m;
- Wing planform area S;
- Simulation time step for altitudes above 100m (TS1);
- Simulation time step for altitudes below 100m (TS2).

As the first four variables do not define the aerodynamic characteristics of the UAV and can be arbitrarily chosen, they are not included in this discussion.

Assuming that a UAV is dropped from 30,000m, the Re that the UAV would experience varies from the low thousands to hundreds of thousands due to the large changes in ρ and V_T. By using the L/D and C_L values of the UAV in the most lift-adverse conditions i.e. when Re is between 2000 and 6000, r was deliberately underestimated to provide a baseline estimate for the landing sites of the UAVs.

The L/D for uncambered NACA airfoils at a 2° angle of attack at ultra-low Re varies from around 2 to 6 and the C_L varies from 0.15 to 0.2 [16]. Taking the UAV to be an airfoil of the above variety, the L/D and C_L of the UAV could be estimated to be 5 and 0.15 respectively. The low L/D and C_L values were also chosen to make the

simulations more realistic by reducing the maximum V_{xy} to 530m/s as illustrated in Figure 27. However, the descent time of the UAV was reduced to 1.33 hours which is less than the 1.99 hours taken by NTNS4. The above differences could be attributed to the larger mass of the UAV used in the simulator.

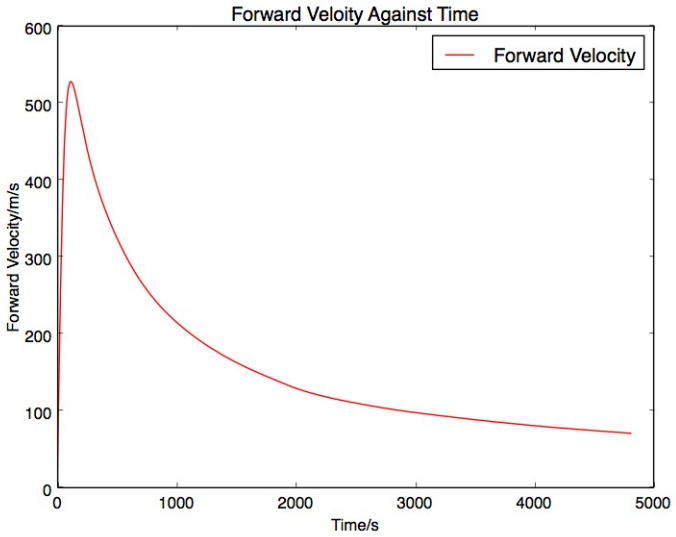


Figure 27: Plot of forward velocity against time.
The peak forward velocity was reduced to 530m/s.

The UAV, when equipped with a Vaisala radiosonde and communication and control systems, could be assumed to weigh 2kg [32]. The UAV would probably have a S around 0.2m^2 as it should be much smaller than the Textron Aerosonde that has a S of 0.6m^2 [33].

The remaining variables are the time steps used for the simulation. Table 1 shows the effect of changing TS1 on the average computing time and the average total absolute percentage error in both x and y directions. The x and y values obtained from simulations with TS1s set at 0.25s were the reference values used to calculate the errors produced by simulations with larger TS1s. From Table 1, using a TS1 of 3s produces the smallest average total absolute percentage error while using relatively little computational resource, therefore a TS1 of 3s was selected. TS2 is not central to producing accurate results as it only covers 100m of altitude and to ensure the simulations stopped close to 0m, so a TS2 as 0.5s was chosen.

Table 2 lists the non-arbitrary input variables used in the simulations. The other variables used the simulations are stated in 7.3.

TS1 (s)	Average Computing Time (s)	Average Total Absolute Error (%)
10	3.7585	0.018
5.0	6.7693	0.029
3.0	10.9581	0.016
1.0	33.1979	0.026
0.5	67.2119	0.018
0.25	131.0510	N.A.

Table 1: A comparison of the average computing time and average total absolute percentage error obtained with different TS1s.

Input Variable	Values
L/D	5
C_L	0.2
m	2kg
S	0.2m^2
TS1	3 seconds
TS2	0.5 seconds

Table 2: A list of the non-arbitrary inputs variables used in the simulations.

7.2 Algorithms

The algorithms determining the heading of the UAV are arguably the most important pieces of code as they are responsible for guiding the UAV in the simulations.

The distance-based algorithm used in Fly_LatLon, Fly_1 and Fly_4 compares the current x coordinates a and current y coordinates of the UAV b against reference x coordinates c and reference y coordinates d to determine the heading of the UAV. In Fly_LatLon and Fly_4, c and d are the coordinates of the desired location relative to the starting point. However, the c and d in Fly_1 are not specified by a desired location as Fly_1 flies the UAV as far as possible along a desired direction. Instead, they are obtained by splitting the current total distance covered by the UAV into x and y coordinates using Equations (10) and (11) with θ set as the desired direction of travel. The heading of the UAV is then set as the $arctan$ of the ratio of the remaining distance to be travelled in the y direction and the remaining distance to

be travelled in the x direction which is expressed as:

$$\theta = \tan^{-1}\left(\frac{d-b}{c-a}\right).$$

The trajectories predicted by a Fly_1 simulation is shown in Figure 28.

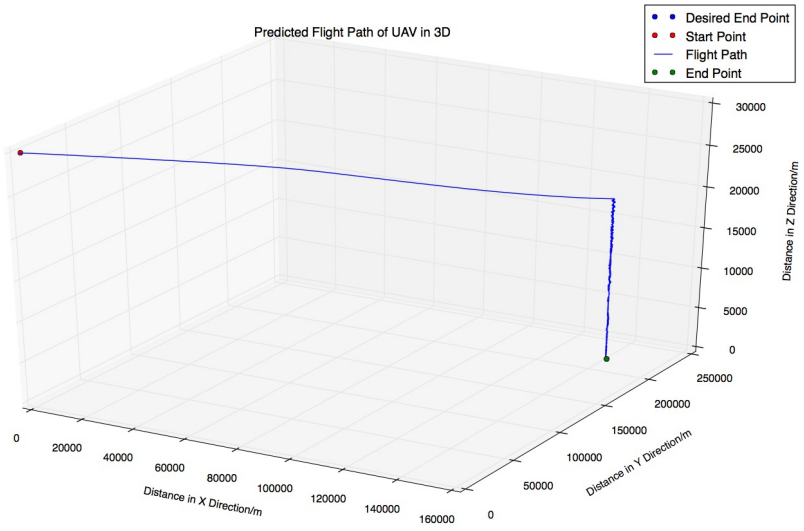


Figure 28: Plot of a Fly_1 simulation.
UAV from 50.1° -5° at 30,000m on 1 Mar 2014 19:31:01 was directed to 52° -3°.

In the process of obtaining the data for this discussion, most UAVs were observed to leave the area covered by the GFS weather data numerous times during Fly_1, Fly_2 and Fly_3 simulations from stratospheric altitudes. To reduce the amount of time spent on simulations, all Fly_1, Fly_2 and Fly_3 simulations presented in this paper were conducted from altitudes of 10,000m and below.

The algorithms used in Fly_1, Fly_2 and Fly_3 to guide the UAV along a certain direction can be classified as path following algorithms. The paths followed are lines that extends indefinitely in the desired direction of travel. Within the path following algorithms, error correction is used to correct any deviations from the path to keep the UAV flying in the desired direction. Error correction can be usually classed as either aggressive or passive.

In this paper, using a path following algorithm with aggressive error correction ensures that the UAV would only fly along the set path. Whenever the UAV deviates from the path, algorithm would direct the UAV back to the point where the deviation occurred before allowing it continue in the desired direction. This causes the UAV to frequently backtrack and sometimes fly in circles, therefore leading to a reduction in the overall distance travelled in the desired direction.

Whereas when path following algorithm with passive error correction is employed, the UAV would be flown in the general direction of the set path with corrections only applied when the UAV deviates too much from the set path. This enables the UAV to fly away from the point of release for most of the simulation, which increases the overall distance travelled in the desired direction.

The path following algorithms with either purely aggressive or purely passive error correction were found to be unsuitable for directing a UAV along a set direction. Therefore the error correction used in Fly_1, Fly_2 and Fly_3 were compromises between aggressive and passive error correction and delivered much better performance. Figure 29 shows the predicted trajectories generated by the use of different error correction and Table 3 displays the overall direction of the predicted trajectories.

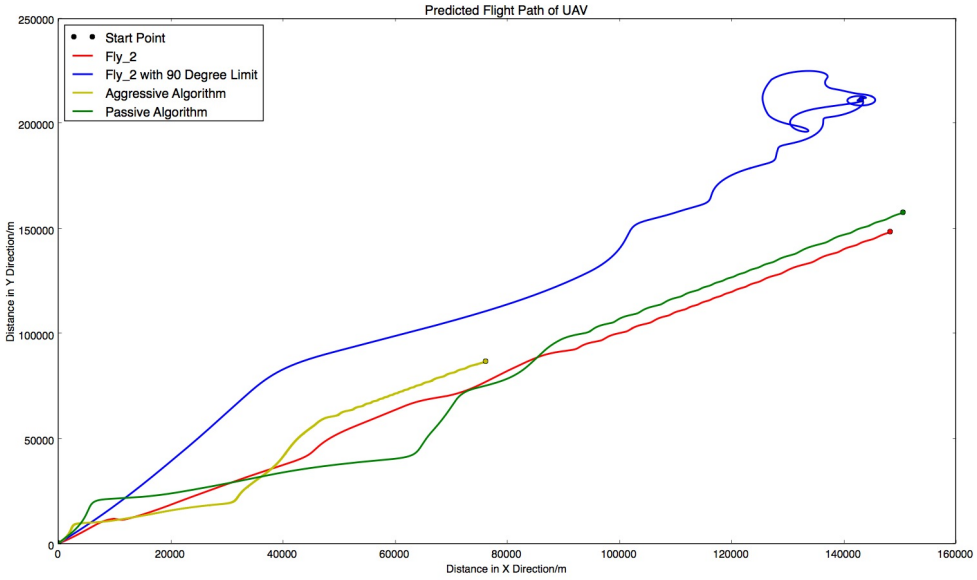


Figure 29: Plot of predicted trajectories generated by different algorithms. All simulated UAVs started from 50.1° -5° at 10,000m on 1 Mar 2014 19:31:01. The desired direction of travel was set as 45°.

Algorithm	Overall Direction
Ideal	45°
Fly_2	45.00737706°
Fly_2 (With 90° Limit)	55.867410007°
Aggressive	48.7635051253°
Passive	46.2989958975°

Table 3: A comparison of the overall direction of different predicted trajectories.

The heading-based algorithm developed for Fly_2 and Fly_3 uses the deviation of the UAV from its intended path to determine the heading of the UAV. The current direction of the UAV is derived from the $arctan$ of the ratio of the current y coordinate and the current x coordinate. The absolute difference between the current direction of the UAV and its desired direction of travel, or the deviation of the UAV, is then computed. If the deviation is greater than 90°, the UAV is simply pointed to the desired direction. For deviations smaller than 90°, the the heading of the UAV is corrected.

Applying too much correction would cause the UAV to spiral is shown in Figure 29. Spiralling is not desirable as it would reduce the overall distance travelled in the desired direction. Hence the amount of correction applied is limited to 45° either side of the desired direction to ensure that any overcorrection can be easily reversed.

The variant of the above heading-based algorithm was implemented in Fly_3 to increase the distance travelled by the UAV in the desired direction. However, as seen from Figure 30, 31 and 32, Fly_3 did not improve the range of the UAV significantly.

A comparison of the average error incurred per simulation of the different algorithms are shown in Table 4. These errors are visualised in Figure 33, further proving the superiority of the error correction used in the Fly algorithms.

Algorithm	Average Error Per Simulation (°)		
	1 Mar 2014 19:31:01	5 Mar 2014 19:31:01	10 Mar 2014 19:31:01
Fly_1	0.0823251728276	0.103079610463	0.0996735412727
Fly_2	0.0866519402243	0.169633978664	0.27671972259
Fly_3	0.087043109293	0.170197207049	0.278307590914
Aggressive	3.81941443975	1.56005079367	0.779170828342
Passive	3.67302273171	1.18744713166	0.66777810973

Table 4: A comparison of the average error per simulation for different algorithms.

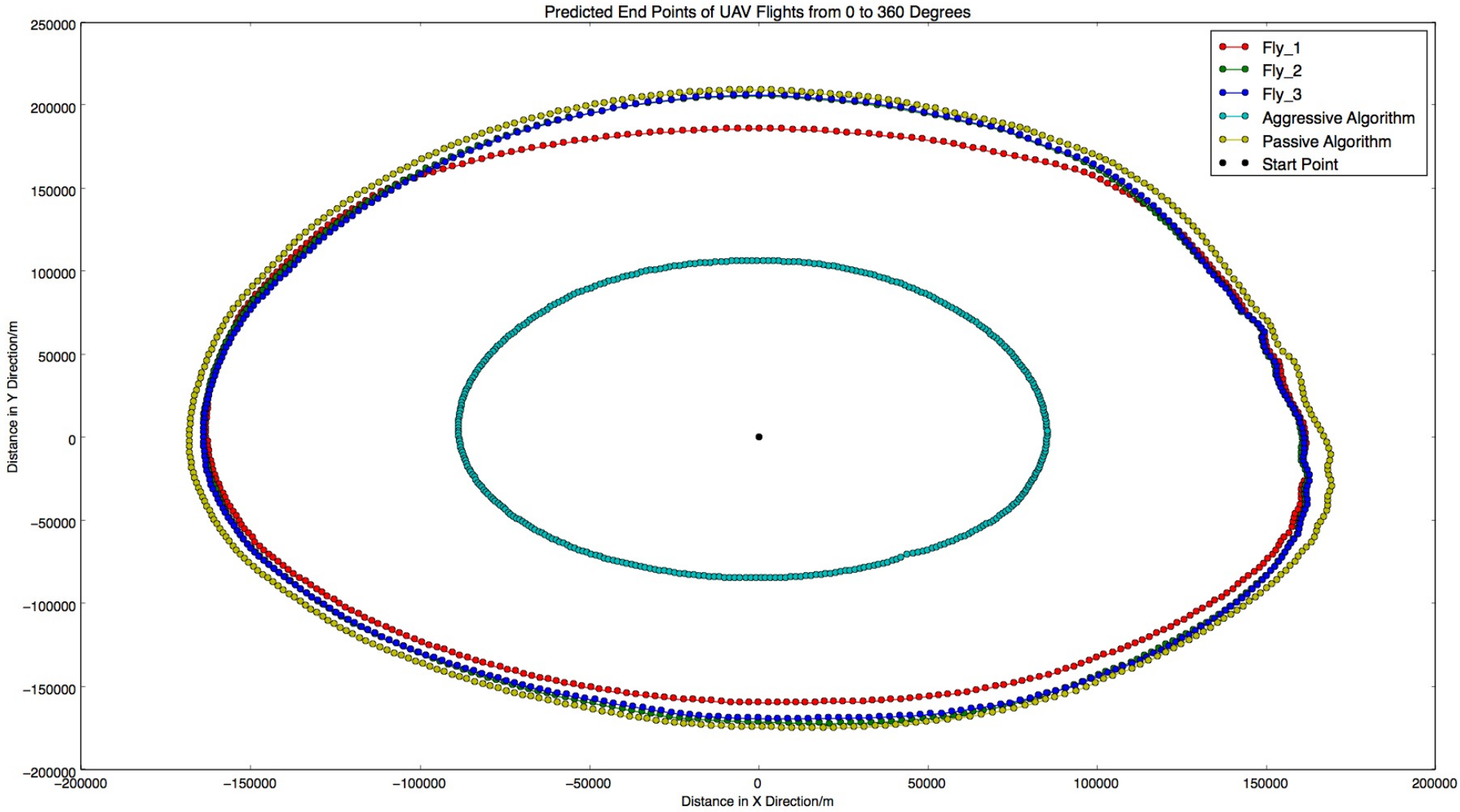


Figure 30: Plot of flight contours produced by different error-correction algorithms.
All simulated UAVs started from 50.1° -5° at 7500m on 1 Mar 2014 19:31:01.

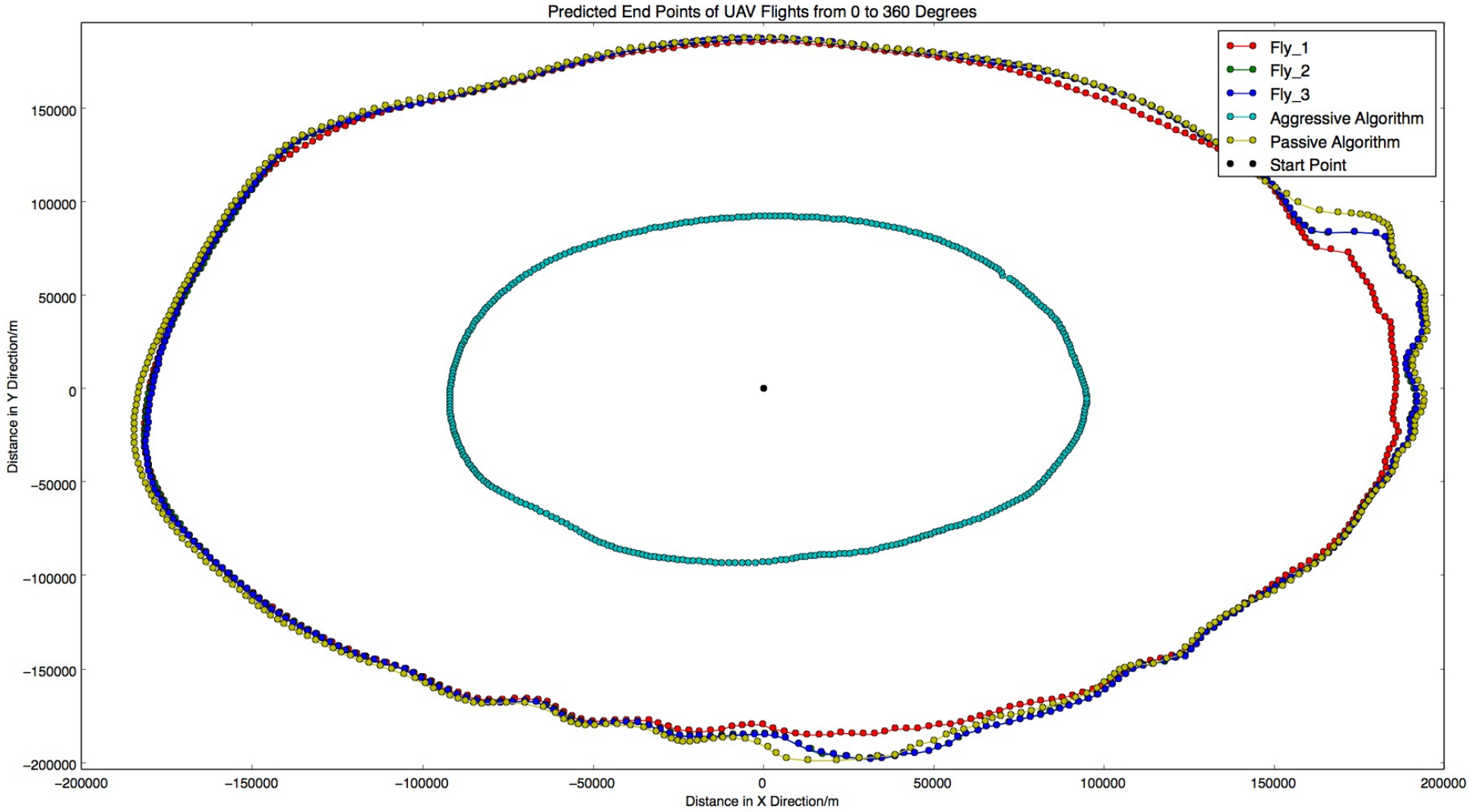


Figure 31: Plot of flight contours produced by different error-correction algorithms.
All simulated UAVs started from 50.1° -5° at 7500m on 5 Mar 2014 19:31:01.

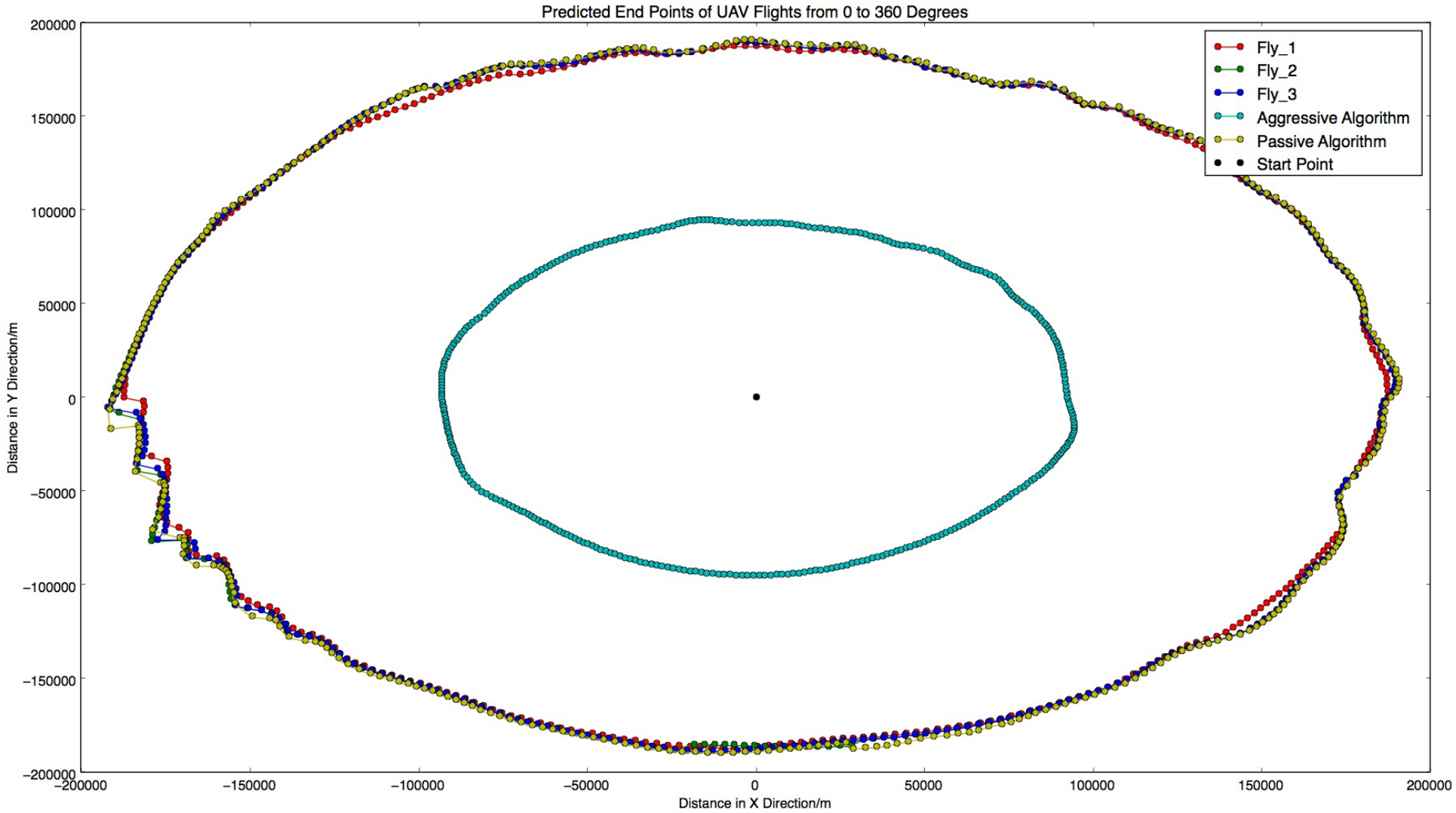


Figure 32: Plot of flight contours produced by different error-correction algorithms.
All simulated UAVs started from 50.1° -5° at 7500m on 10 Mar 2014 19:31:01.

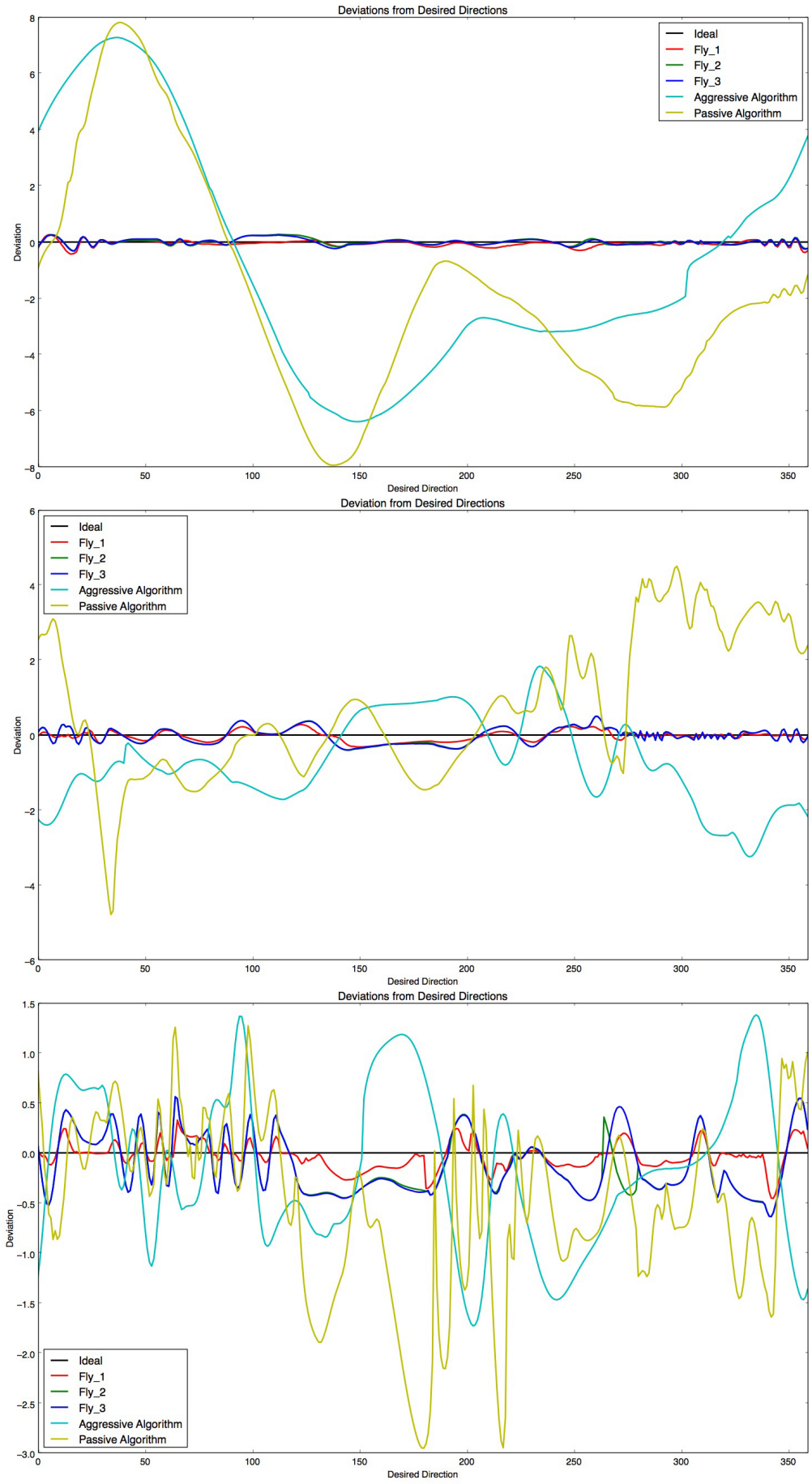


Figure 33: Plots of deviations from desired directions
The simulated UAVs started from 50.1° -5° at an altitude of 7500m on 1 Mar 2014 19:31:01, 5 Mar 2014 19:31:01, 10 Mar 2014 19:31:01 respectively.

The conclusions that could be drawn from the simulations are:

- Using aggressive error correction would always generate the smallest flight contour;
- Using passive error correction would always generate the largest flight contour;
- The average error per simulations produced by heading-based Fly_2, Fly_3, aggressive and passive algorithms were more dependent on weather conditions compared to the distance-based Fly_1 algorithm;
- Among the Fly algorithms, Fly_1 would always generate the smallest flight contour but yield the least average error per simulation;
- The error correction used by Fly_1, Fly_2 and Fly_3 were superior to the purely aggressive or passive error correction regardless of wind conditions.

As Fly_1 provides the best error correction, its algorithm was selected to be implemented in Fly_4. The performance of Fly_4 would be discussed in 7.4.

7.3 Results

The subsection presents the results produced by Fly_1 and Fly_4 simulations. The significance of the results is discussed in the next subsection.

Figure 34 shows the collected predicted trajectory plots of 4 Fly_1 simulations. The desired landing coordinates for all the simulations were 51° -1.4°. Starting from the top, the first pair of plots depict the predicted trajectory of the UAV released from 55° 3.6°, the second pair from 45° 3.6°, the third pair from 55° -6.4° and the last pair from 45° -6.4°. Table5 shows the coordinates produced by the simulations.

Input Coordinates (° °)	Output Coordinates (° °)
N.A.	51 -1.4 (Ideal)
55 3.6	51.0000314697 -1.39968358584
45 4.6	50.999967526 -1.39999025189
55 -6.4	50.7391425869 -1.19005207719
45 -6.4	51.0000427133 -1.39973291753

Table 5: Input and output coordinates of the 4 Fly_1 simulations.

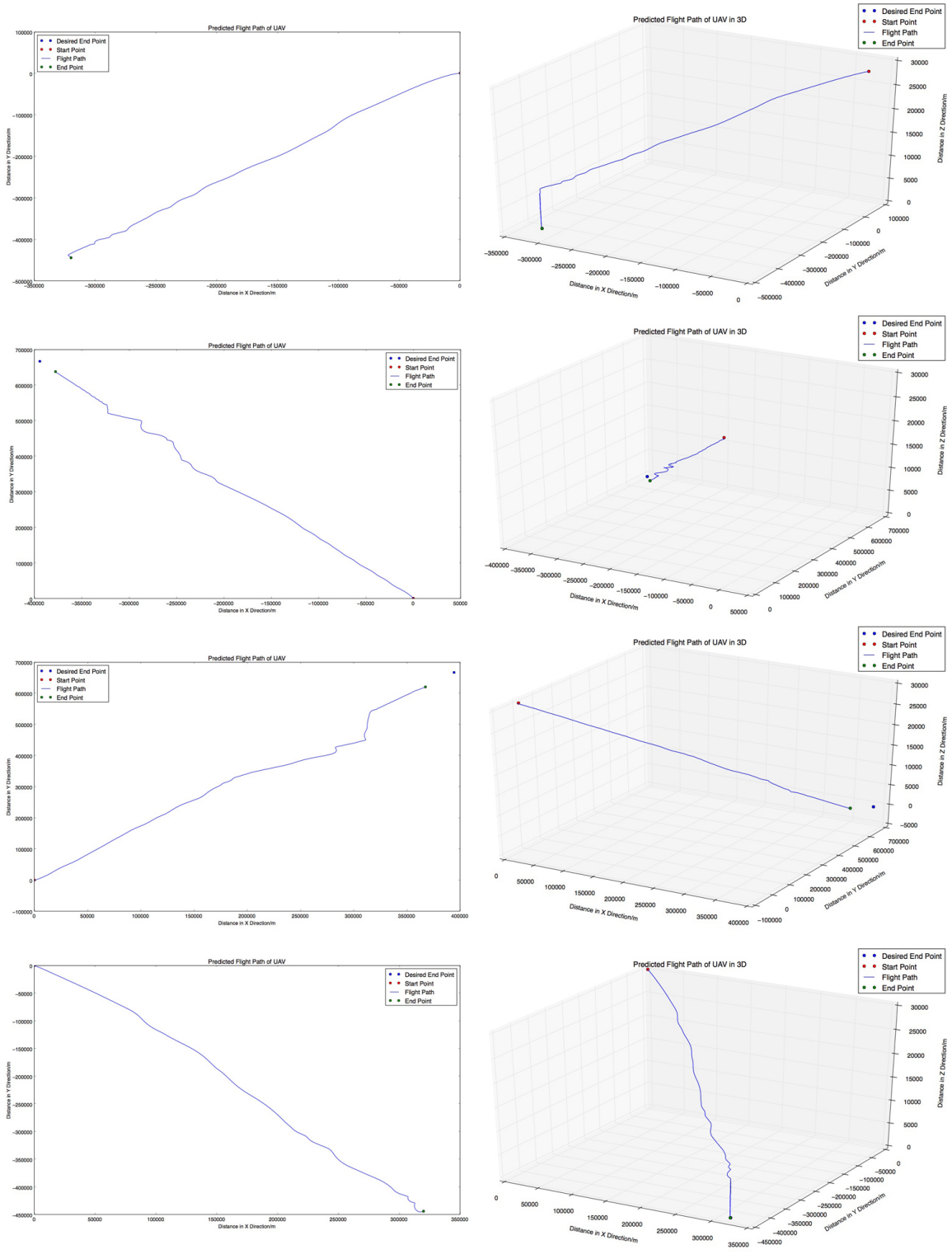


Figure 34: Plots of Fly_1 simulations.
The first pair of plots depict the predicted trajectory of the UAV released from 55° 3.6°, the second from 45° 3.6°, the third from 55° -6.4° and the last from 45° -6.4°.

Two soundings with different start times were simulated using the ASTRA High Altitude Balloon Flight Planner, as seen in Figure 35, from 51° -1.4° from 1 March 2014 19:31 and 10 March 2014 19:31.

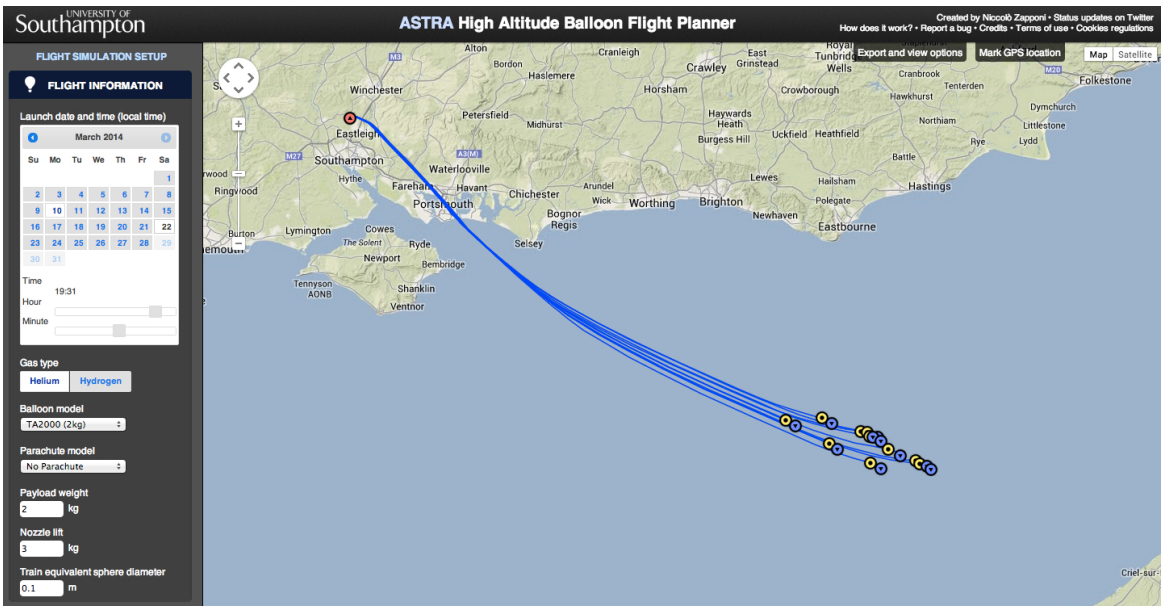


Figure 35: Settings for ASTRA High Altitude Balloon Flight Planner.

The inputs used in the simulations are shown in Table 6. The nozzle lift and train equivalent sphere diameter were arbitrarily selected.

Variables	Input/s
Gas Type	Helium
Balloon Model	TA2000 (2kg)
Parachute Model	No Parachute
Payload Weight	2kg
Nozzle Lift	3kg
Train Equivalent Sphere Diamete	0.1m
Weather Source	Online Forecast, High Quality
Number of Simulation Runs	20
Flight Type	Standard

Table 6: Inputs used in ASTRA High Altitude Balloon Flight Planner.

Fly_1 simulations were then performed from the altitude and coordinates where the balloon popped in the ASTRA simulations to guide the UAV back to 51° -1.4°. These values, together with the Fly_1 outputs, are shown in Table 7

Figure 36 and 37 shows the starting coordinates, the end coordinates, the desired coordinates and the flight path of the UAV produced by the the two simulations superimposed on Google Maps.

Launch Date and Time	Start Coordinates(° °)	Altitude (m)
1 Mar 2014 19:31	50.23539147 0.64237828	35602.88423
10 Mar 2014 19:31	50.33125566 -0.336643243	35962.74666
Release Date and Time	End Coordinates(° °)	
N.A.	51 -1.4 (Ideal)	
1 Mar 2014 21:27:18	51.0002077887 -1.40000270409	
10 Mar 2014 21:17:36	51.0000221209 -1.40006587259	

Table 7: Start coordinates, altitude and end coordinates.

To determine the accuracy of the landing sites of the distance-based algorithm and plot the flight contours of a UAV released from height, Fly_4 simulations were carried out from an altitude of 5,000m with a set error margins across varying distances and desired directions. The flight contours with error margins of 5m and 10m with starting times of 1 March 2014 19:31:01 and 10 March 2014 19:31:01 respectively are shown in Figures 38, 39.

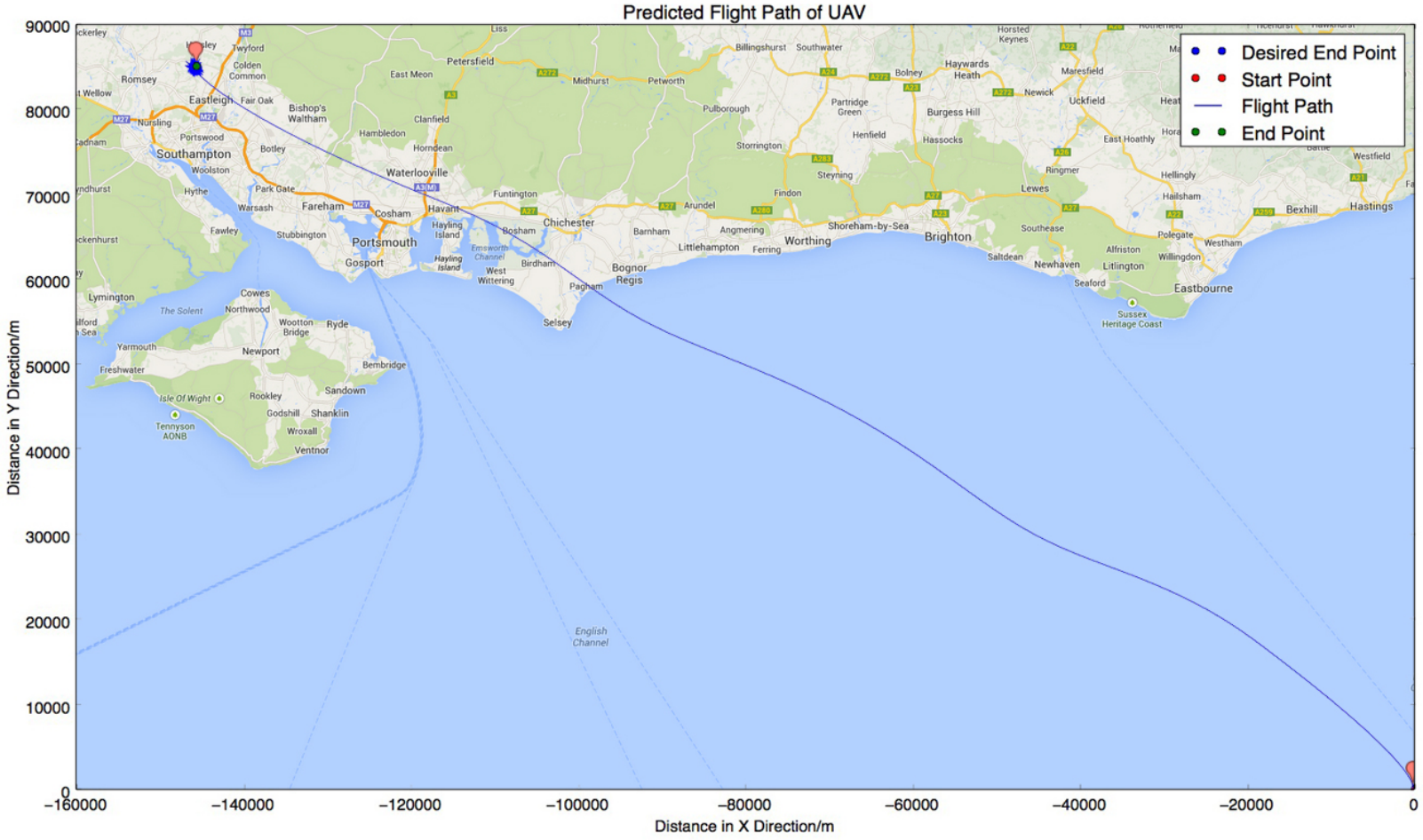


Figure 36: The predicted trajectory of a UAV released on 1 Mar 2014 21:27:18.

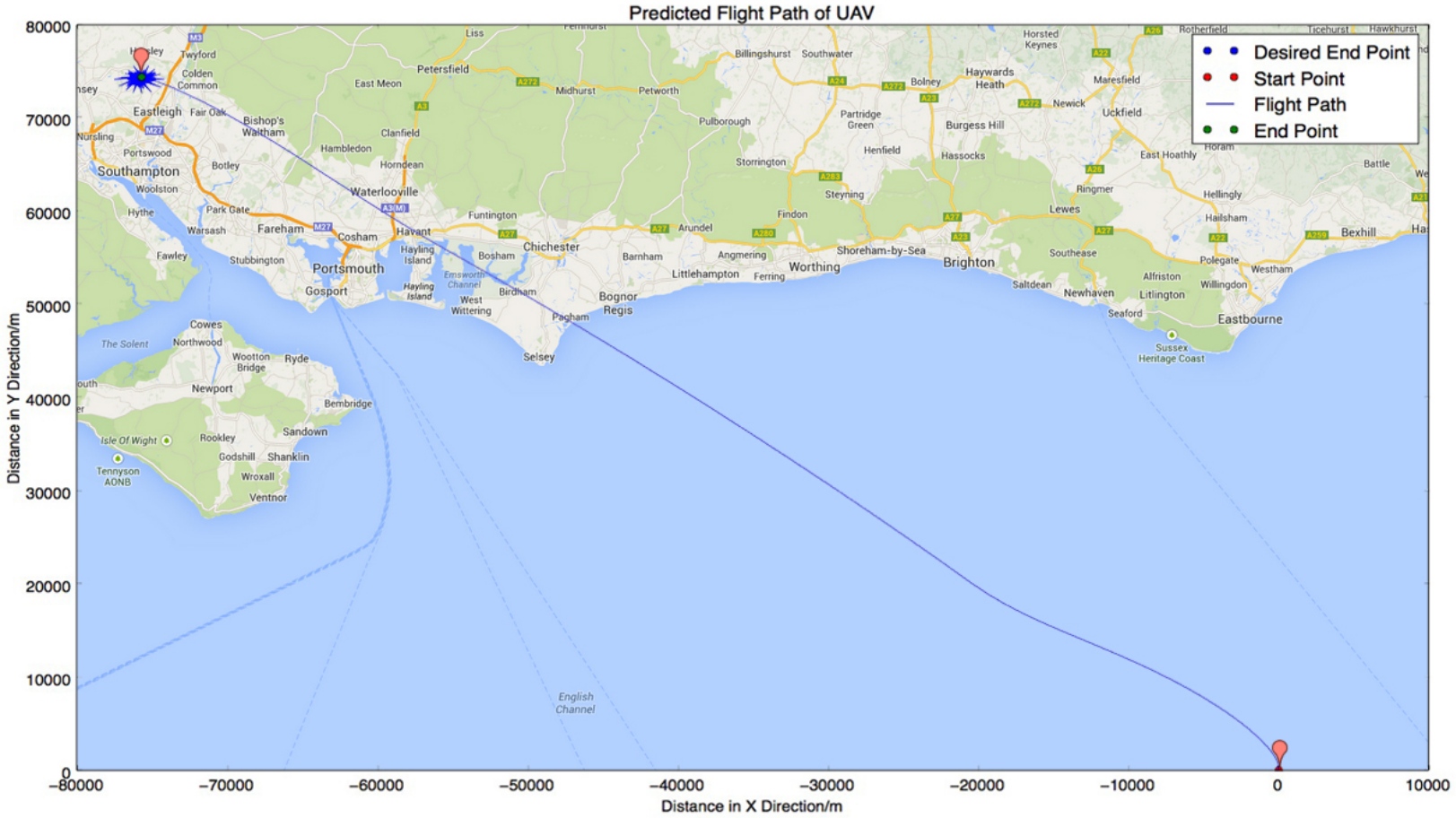


Figure 37: The predicted trajectory of a UAV released on 10 Mar 2014 21:17:36.

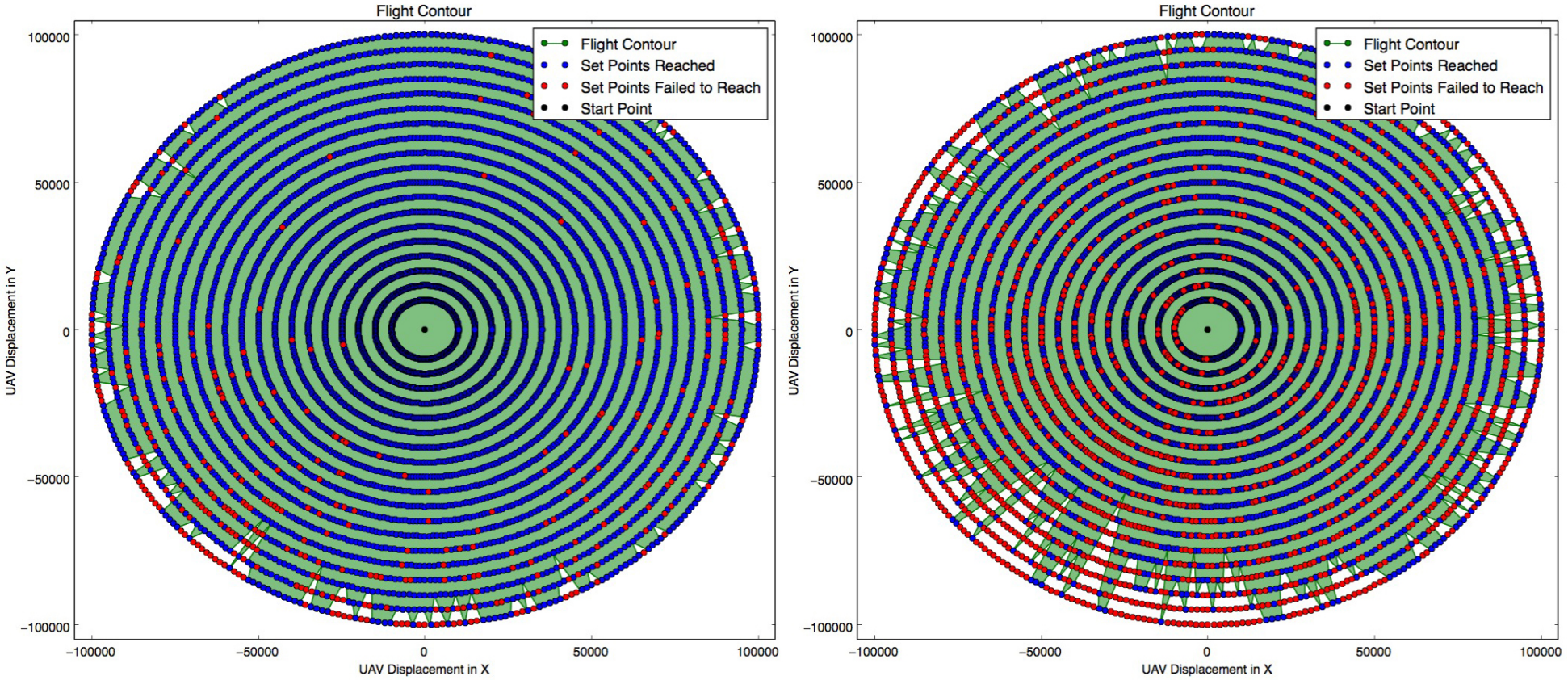


Figure 38: The flight contour plots for the 1 March 2014 simulations.
The error margin of the plot on the left is 10m while the error margin for the plot on the right is 5m.

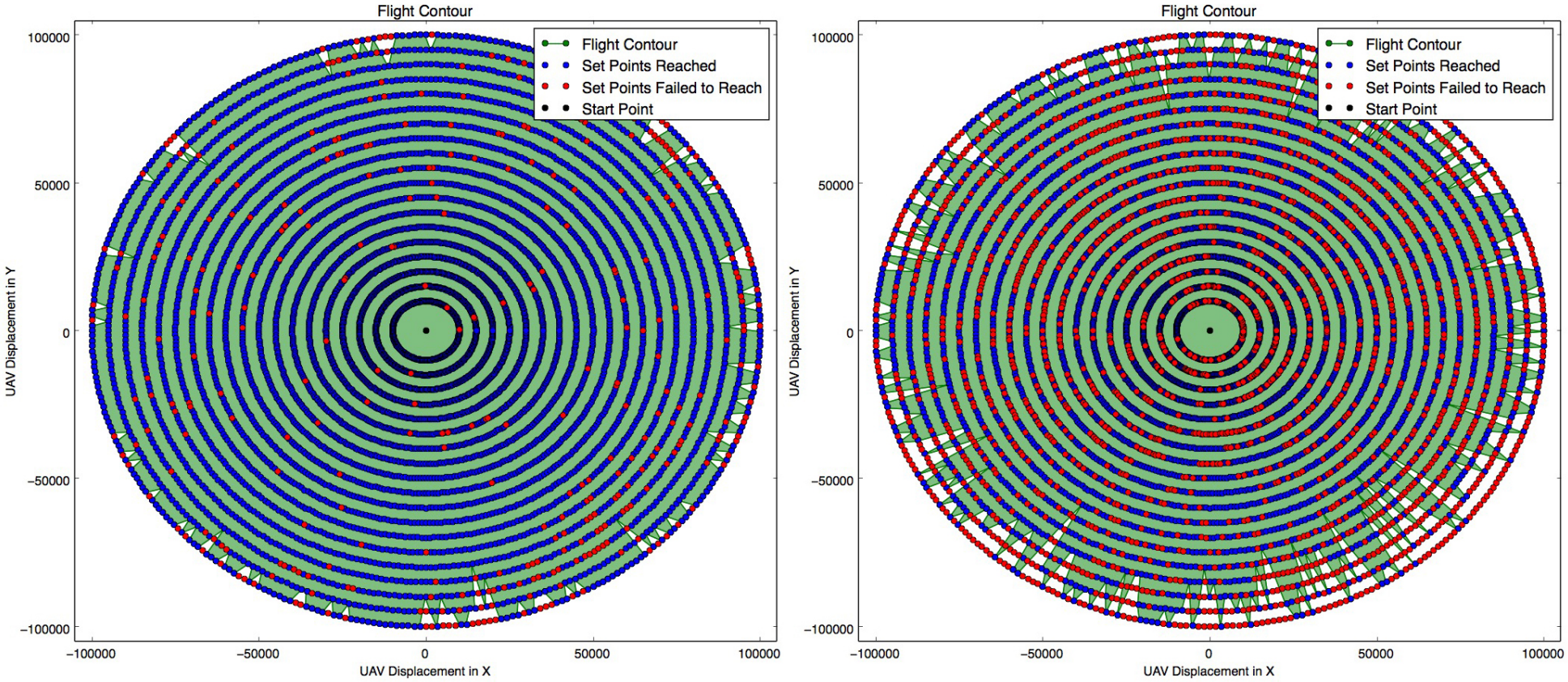


Figure 39: The flight contour plots for the 10 March 2014 simulations.
The error margin of the plot on the left is 10m while the error margin for the plot on the right is 5m.

7.4 Discussion of Results

In the first set of Fly_1 simulations, the starting coordinates were set significantly away from the desired landing coordinates to show the ability of Fly_1's distance-based path following algorithm to guide a UAV precisely to a distant landing site. As seen from Figure 34 and Table 5, the algorithm is able to guide UAVs released from 30,000m to targets within a 400,000m radius with minimal error.

The predicted trajectories produced by second set of Fly_1 simulations (Figures 36 and 37) demonstrates the possibility of a UAV released from a weather balloon gliding back to the launch site. This strengthens the argument for the use of unpowered UAVs to replace radiosondes as the UAVs would be able to glide back to their launch sites with much altitude to spare.

The Fly_4 simulations carried out shows the ability of the distance-based algorithm to guide UAVs to landing sites precisely. These simulations also investigate the degree of accuracy that the algorithm could guide UAVs to landing sites. Figures 38 shows that the algorithm was capable of guiding the UAV to almost all the points that were 10,000m to 100,000m from 0° to 359° to a $400m^2$ area surrounding the target point. When the error margin was decreased to $5m^2$, the UAV was still guided to a $200m^2$ area surrounding the target point for 5115 and 4693 respectively out of 6840 simulations. This was visualised in Figure 39, further underlining the capability of the algorithm. The plots generated from simulations with error margins of 25m, 50m, 150m, 250m and 500m were discarded as the UAV only missed less than 20 points for each set of simulations.

8 Conclusion and Recommendations

In the course of writing this paper, all the objectives listed in 2 were met in the development the UAV Flight Simulator package. The Fly_1 method is able to guide the UAV to a desired location with minimal error and demonstrates the possibility of UAVs released from weather balloons returning to their launch sites. The Fly_Range and Fly_Range_2 methods generates flight contours that covers all the predicted landing sites of the UAV.

All in all, the UAV Flight Simulator package provides an simple pre-launch trajectory planning tool for unpowered UAVs to provide users with an estimate of the possible landing site of a UAV released at a user-defined altitude and location.

To increase the accuracy of the predicted trajectories and address the errors of the physics model, the following improvements could be made:

- Obtain weather data from NWP models specific to the input location to provide higher resolution data sets;
- Account for wind direction in the z direction by obtaining relevant data from sites like XC Skies and SoaringMeteo;
- Produce Skew-T Log-P diagrams from the obtained weather data to inform users of the possibility of thermals in the areas covered by the UAV;
- Consider the different terrain of the Earth;
- Improve the physics model by considering the inertia of the UAV when changing directions and the additional drag on the UAV caused by crosswinds;
- Model the UAV as a complete aircraft by including more aerodynamic characteristics of the simulated UAV in the physics models;
- Provide L/D and C_L values that changes with to Re and angles of attack for different airfoils;
- Release different UAVs from stratospheric altitudes and compared the data collected from such experiments with their predicted trajectories to further refine the physics model;
- Use more advanced guiding algorithms
- Provide a web-based interface for the simulators that plots the predicted trajectories of UAVs on Google Maps;
- Combine the capabilities of the ASTRA High Altitude Balloon Flight Planner and the UAV Flight Simulator package to provide a complete weather balloon-UAV trajectory prediction package.

Although there is still much more to be done to improve the predictions of trajectories for stratospheric UAV operations, the simulator package provided by this paper would aid meteorological agencies and atmospheric research groups in locating the possible landing sites of unpowered UAVs if they are used to replace radiosondes.

A Appendix A

A.1 BaseSimulator Commands

To run the BaseSimulator module from the command-line interface:

```
python
from base_simulator import BaseSimulator
y = BaseSimulator(a, b, c, d, e, f)
y.BaseFly()
y.PlotBaseFly()
y.PlotWindContour()
y.PlotFlightContour()
```

The placeholder variables used the above code are detailed in Table 8:

Placeholder Variables	Name
a	Wind speed
b	Wind direction
c	UAV speed
d	UAV heading
e	Altitude
f	Lift-to-drag ratio

Table 8: List of placeholder variables used in BaseSimulator.

A.2 LayerSimulator Commands

To run the LayerSimulator module from command-line interface:

```
python
from layer_simulator import LayerSimulator
LayerSimulator()
```

A.3 GFS Data Simulator Commands

To run the GFS Data Simulator module from the command-line interface:

```
python
from gfs_data_simulator import Flight_LatLon
y = Flight_LatLon()
y.Fly_LatLon(a, b, c, d, e, f, g, h, i, j, k, l)
y.PlotContour_LatLon(True, True, True, True, False)
```

The placeholder variables used the above code are detailed in Table 9:

Placeholder Variables	Name
a	Starting latitude
b	Starting longitude
c	Desired x coordinate
d	Desired y coordinate
e	Starting altitude
f	Starting time
g	Lift-to-drag ratio
h	Mass of UAV
i	Wing planform area
j	Coefficient of lift
k	Time step for altitudes greater than 100
l	Time step for altitudes less than 100

Table 9: List of placeholder variables used in GFS Data Simulator.

A.4 GFS Local Data Simulator Commands

To run the GFS Local Data Simulator module from the command-line interface:

```
python
from gfs_data_simulator_local import Flight
y = Flight(a, b, c, d, e, f, g, h, i, j)
y.Fly_1(k)
y.Fly_2(k)
y.Fly_3(k)
```

```
y.Fly_4(l, m, n, k)
y.PlotContour(True, True, True, True, False)
y.Fly_Range(o, k, True, True, True, True, True)
y.Fly_Range_2(p, q, r, s, t, n)
```

The placeholder variables used the above code are detailed in Table 10:

Placeholder Variable	Name
a	Starting latitude
b	Starting longitude
c	Starting altitude
d	Starting time
e	Lift-to-drag ratio
f	Mass of UAV
g	Wing planform area
h	Coefficient of lift
i	Time step for altitudes greater than 100
j	Time step for altitudes less than 100
k	Desired direction
l	Desired x landing coordinate
m	Desired y landing coordinate
n	Error margin
o	Fly method
p	First desired direction
q	Last desired direction
r	First desired distance
s	Last desired distance
t	Interval between distances

Table 10: List of placeholder variables used in GFS Local Data Simulator.

B Appendix B

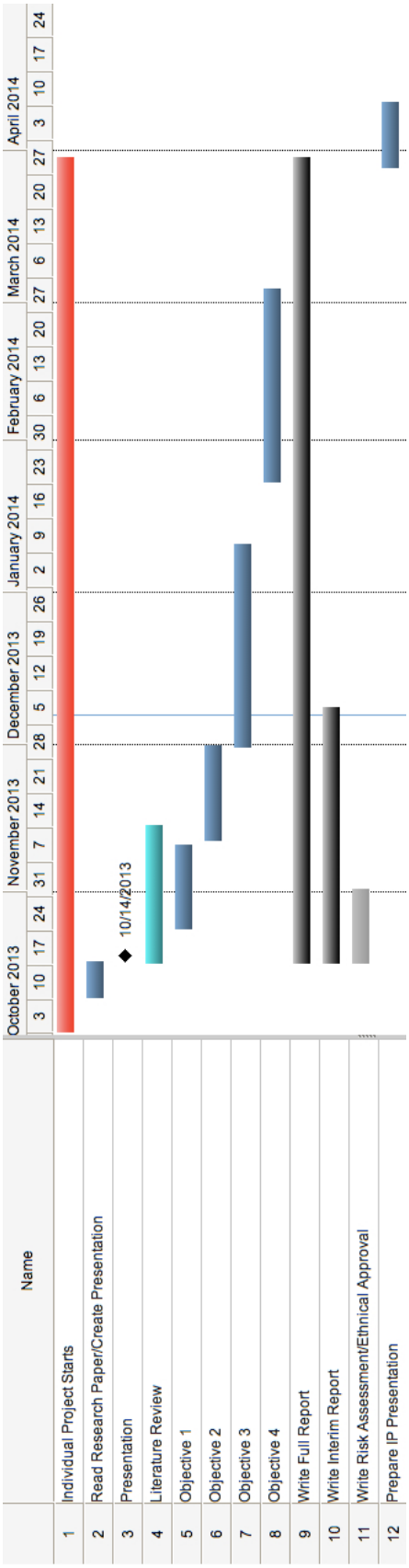


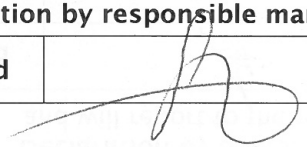
Figure 40: Gantt Chart.

UNIVERSITY OF Southampton

General Health & Safety Risk Assessment Template

Work activity / task	Individual Project				
Assessor(s)	Dr Andras Sobester	**Responsible Manager**	Dr Andras Sobester	**Date**	02/12/13
Faculty / Service	Engineering and Environment	**Academic Unit / Team**	Aeronautics and Astronautics	**Location**	N.A.
Brief description of activity / task	Computational Modelling of UAV Flight				
Additional notes (eg, references, persons at risk, risk factors, etc) [optional]					

Declaration by responsible manager: I confirm that this is a suitable & sufficient risk assessment for the above work activity / task.

Signed		**Print name**	ANDRÁS SÓBESTER	**Date**	5/12/13

Version 1.11, 05 June 2013

1

Figure 41: Risk Assessment.

Declaration by users: I confirm that I have read this risk assessment, will implement the controls outlined herein, and will report to the responsible manager any incidents that occur or any shortcomings I find in this assessment.					
Signed		Print name	SI KAI LEE	Date	08.12.18
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	
Signed		Print name		Date	

Figure 42: Risk Assessment (Continued).

References

[1] Cooperative Institute for Research in Environmental Sciences, *Getting Close — Antarctic UAVs*, http://cires.colorado.edu/blogs/antarcticuavs/2012/09/10/getting-close/, Accessed 29 Nov 2013, Sep. 2012.

[2] NOAA National Weather Service, *Radiosonde Observations*, http://www.ua.nws.noaa.gov/factsheet.htm, Accessed 24 Nov 2013, 2009.

[3] Android in Near Space, *Launching Weather Balloons*, http://android.hibal.org/, Accessed 29 Nov 2013, 2010.

[4] F. Flores, R. Rondanelli, M. Daz, R. Querel, K. Mundnich, L. A. Herrera, D. Pola, and T. Carricajo, "The Life Cycle of a Radiosonde," *Bulletin of the American Meteorological Society*, vol. 94, no. 2, pp. 187–198, 2013.

[5] M. Jonassen, "The Small Unmanned Meteorological Observer (SUMO)-Characterization and Test of a New Measurement System for Atmospheric Boundary Layer Research," Master's thesis, University of Bergen, 2008.

[6] A. Sobester, H. Czerski, N. Zapponi, and I. Castro, "Notes on Meteorological Balloon Mission Planning," in *AIAA Balloon Systems Conference, Daytona Beach, FL, March 2013 , p. 1-14, 2013*, ser. AIAA Balloon Systems (BAL) Conference, 2013.

[7] T. Konrad, M. Hill, J. Rowland, and J. Meyer, "A Small, Radio-Controlled Aircraft as a Platform for Meteorological Sensors," *Appl. Phys. Lab. Tech. Digest*, vol. 11, 1970.

[8] J. R. Soddell, K. McGuffie, and G. J. Holland, "Intercomparison of Atmospheric Soundings from the Aerosonde and Radiosonde," *Journal of Applied Meteorology*, vol. 43, pp. 1260–1269, 9 2004.

[9] Office of the Federal Coordinator for Meteorology, *Federal Meteorological Handbook No. 3*. 1997.

[10] L. Haines, *Autopilot Guides Texan Plane Home From A Dizzying 30,000m*, http://www.theregister.co.uk/2013/11/19/lohan_autopilot_update/, Accessed 29 Nov 2013, Nov. 2013.

[11] R. Brears, *"Using Unmanned Aerial Vehicles in Antarctica"*, Postgraduate Certificate in Antarctic Studies, University of Canterbury, 2011.

[12] NASA Wallops Flight Facility, *NASA and NOAA Fly Unmanned Aircraft into Hurricane Noel*, http://www.nasa.gov/centers/wallops/news/story105.html, Accessed 24 Nov 2013, 2007.

[13] L. Kunzwa, *"Modelling and Simulation of a UH UAV Dynamics"*, BEng Final Year Project Report, University of Hertfordshire, 2011.

[14] L. M. C. S. Tan, “Computational Modeling of the Aerodynamic Characteristics and Flight Mechanics and Control of an Unmanned Aerial Vehicle,” Thesis, Nanyang Technological University, 2008.

[15] N. Ylilammi, “Experimental and Computational Study of Two Flapped Airfoils at Low Reynolds Numbers,” Master’s thesis, Helsinki University of Technology, 2009.

[16] P. J. Kunz, “Aerodynamics and Design for Ultra-Low Reynolds Number Flight,” PhD thesis, Stanford University, 2003.

[17] R. W. Ritchie, “Comparative Evaluation of Computational Techniques For Estimating UAV Aerodynamic, Stability and Control Derivatives,” Master’s thesis, University of Minnesota, 2013.

[18] Airborne Scientific Inc., *OSAC Flight Planning Software*, `http://www.airbornescientific.com/content/osacplanning`, Accessed 27 Nov 2013, 2010.

[19] MAVinci, *Flight Planning Software: MAVinci Desktop*, `http://www.mavinci.de/completesys/desktop`, Accessed 27 Nov 2013, 2013.

[20] Orbit Logic, *UAV Planner*, `http://www.orbitlogic.com/products/uav.php`, Accessed 27 Nov 2013, 2012.

[21] I. K. Nikolos, K. P. Valavanis, N. C. Tsourveloudis, and A. N. Kostaras, “Evolutionary Algorithm Based Offline/Online Path Planner for UAV Navigation,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 33, no. 6, pp. 898–912, 2003.

[22] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi, “An Evolution Based Path Planning Algorithm for Autonomous Motion of a UAV through Uncertain Environments,” in *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, IEEE, vol. 2, 2002, pp. 8D2–1.

[23] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, “Vector Field Path Following for Miniature Air Vehicles,” *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 519–529, 2007.

[24] M. Breivik and T. I. Fossen, “Principles of Guidance-Based Path Following in 2D and 3D,” in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC’05. 44th IEEE Conference on*, IEEE, 2005, pp. 627–634.

[25] J. Tisdale, Z. Kim, and J. Hedrick, “Autonomous UAV Path Planning and Estimation,” *Robotics & Automation Magazine, IEEE*, vol. 16, no. 2, pp. 35–42, 2009.

[26] M. Kress and J. O. Royset, “Aerial Search Optimization Model (ASOM) for UAVs in Special Operations,” *Military Operations Research*, vol. 13, no. 1, pp. 23–33, 2008.

[27] A. Aguiar, I. Kaminer, R. Ghabcheloo, A. Pascoal, N. Hovakimyan, C. Cao, and V. Dobrokhodov, “Coordinated Path Following of Multiple UAVs for Time-Critical Missions in the Presence of Time-Varying Communication Topologies,” in *Proc. 17th IFAC World Congress*, 2008.

[28] Federal Aviation Administration, *Glider Flying Handbook*. 2013.

[29] R. F. Stengel, *Flight Dynamics*. Princeton University Press, 2004.

[30] N. Zapponi, *”Design of a Flight Planning Interface for Stratospheric Balloon Operations”*, Individual Project, University of Southampton, 2013.

[31] The Engineering Toolbox, *International Standard Atmosphere*, `http://www.engineeringtoolbox.com/international-standard-atmosphere-d_985.html`, Accessed 15 Mar 2014.

[32] Vaisala, *Vaisala Radiosonde RS92-D*, Accessed 20 Mar 2013, 2010.

[33] Textron Systems, *Aerosonde Mark 4.7: Redefining Expenditionary*, 2010.