An open-source, stochastic, six-degrees-of-freedom rocket flight simulator, with a probabilistic trajectory analysis approach

Willem J. Eerland *, Simon Box †, Hans Fangohr ‡ , and András Sóbester §
University of Southampton, Southampton, England SO17 1BJ, United Kingdom

Predicting the flight-path of an unguided rocket can help overcome unnecessary risks. Avoiding residential areas or a car-park can improve the safety of launching a rocket significantly. Furthermore, an accurate landing site prediction facilitates recovery. This paper introduces a six-degrees-of-freedom flight simulator for large unguided model rockets that can fly to altitudes of up to 13 km and then return to earth by parachute. The open-source software package assists the user with the design of rockets, and its simulation core models both the rocket flight and the parachute descent in stochastic wind conditions. Furthermore, the uncertainty in the input variables propagates through the model via a Monte Carlo wrapper, simulating a range of possible flight conditions. The resulting trajectories are captured as a Gaussian process, which assists in the statistical assessment of the flight conditions in the face of uncertainties, such as changes in wind conditions, failure to deploy the parachute, and variations in thrust. This approach also facilitates concise presentation of such uncertainties via visualisation of trajectory ensembles.

Nomenclature

$\mathbb{E}$	= expectation
$\mathbf{k}$	= covariance kernel
M	= number of data-points
$\mathbf{m}$	= mean function
$\boldsymbol{\mu}$	= mean vector
$\mathcal{N}$	= Gaussian function
$\boldsymbol{\Sigma}$	= covariance matrix
σ	= standard deviation
τ	= normalised time [-]
$\boldsymbol{\tau}$	= normalised time vector [-]
$\mathbf{u}$	= vector holding a three-dimensional coordinate [m]
$\mathbf{v}$	= column vector holding three-dimensional coordinates [m]
$\hat{\mathbf{v}}$	= estimated column vector holding three-dimensional coordinates [m]
x	= easting [m]
$\mathbf{x}$	= easting vector [m]
y	= northing [m]
$\mathbf{y}$	= northing vector [m]
z	= altitude [m]
$\mathbf{z}$	= altitude vector [m]

*Ph.D. Candidate, Aeronautics, Astronautics, and Computational Engineering. Member AIAA, w.j.eerland@soton.ac.uk
†Visiting Research Fellow, Transportation Research Group. Member AIAA
‡Professor, Aeronautics, Astronautics, and Computational Engineering
§Associate Professor, Aeronautics, Astronautics, and Computational Engineering. Member AIAA

I. Introduction

LARGE, unguided model rockets are flown extensively by hobbyists[1,2] and in academia[3]. Safety regulation for such events demands a probabilistic assessment of flight trajectories in the context of the forecast conditions and the uncertainty over various aspects of such flights[4]. At the same time, such models may facilitate the assessment of security threats posed by certain improvised rocket-propelled devices.

There are various software packages available on the market that assist the user in designing a rocket, and predict the corresponding flight path; these include commercial solutions such as RockSim[5], and open-source solutions such as OpenRocket[6]. However, all these packages simulate the performance of a rocket under nominal conditions, returning only the expected performance to the user. RockSim Pro[5] does displays confidence bounds around the predicted landing location to the user, but is only available to US citizens.

The Cambridge Rocketry Simulator[7] assists the user to design a concept rocket in a Graphical User Interface (GUI), fly the rocket in a range of possible flight conditions, and obtain confidence bounds on the expected flightpath and landing location by analysing the output trajectory data. The specific contributions of the third version compared to the previous versions are: the improved GUI, which is based on the OpenRocket[6] code, a modification to the simulator core from Box, Bishop and Hunt[8] to incorporate additional stochastic variables, and the expansion of user input to set the of degree uncertainty for the stochastic variables. This puts the Cambridge Rocketry Simulator on a par with RockSim Pro. On top of that, the entire software package is open-source and available online.

Finally, the trajectory analysis approach from Eerland, Box and Sóbester[9], available as an open-source Python toolbox[10], is applied to rocket trajectories. This approach models the trajectory data as a Gaussian process, capturing the general trend and dispersion. The direct application here is the ability to visualise the behaviour of a large dataset without cluttering the screen. Besides being helpful in visualising, it also allows *quantifying* the difference between the simulated scenarios. It visualises two- and three-dimensional data using the Python libraries Matplotlib[11] and Mayavi[12] respectively. The calculations are done using the libraries SciPy and Numpy[13], and all the data handling is done via Pandas[14]. The results seen in this paper are produced in Jupyter Notebooks[15], which are available online as supplemental material.

The remainder of the paper is organised as follows. In section II, the architecture of the program is described, elaborating on the purpose of each part and explaining the theory applied. Furthermore, it includes an overview of the various rocket components and describes the stochastic elements available in the simulator. Finally, it describes the method of using time-cues to model the rocket trajectories as a Gaussian process. Section III analyses the trajectory data generated in four different scenarios, namely, changes in wind conditions, failure to deploy the parachute, and variations in thrust. This section show-cases the usability of the trajectory analysis applied to rocket trajectories. Finally, in section IV, the conclusions are presented.

II. Methodology

This section starts of with an overview of the software architecture. As there is a clear divide between functionality and programming languages, this simultaneously provides an overview of functionality. Subsection B introduces the module that assists the user with the rocket design. Subsection C is an overview of the stochastic input variables. Finally, subsection D elaborates on how the rocket trajectories are modelled as a Gaussian process, useful for visualising the behaviour of numerous trajectories, and quantifying the difference between varying the launch- and atmosphere conditions. Furthermore, it also explains how abrupt changes in the trajectories are dealt with, such as the activation of the second stage, or a parachute deployment.

A. Overview of the architecture

The architecture of the Cambridge Rocketry Simulator features three programming languages. The GUI is written in Java, providing a method for the user to control the input parameters. It also assists with the design of the rocket. The simulator core is written in C++, allowing rapid execution of numerous rocket launches – a requirement for effectively applying the Monte Carlo method to obtain a probabilistic interpretation. A more in-depth review of the software is available online[7], and includes a description of

the support and testing framework. Finally, the visualisation is written in Python. An overview of the architecture is available in Figure 1.

The information between the three components is transferred via Extensible Mark-up Language (XML) documents. Where the input data of the simulator consists of the physics related to the rocket (e.g. moment of inertia as a function of time), and the output data consists of flight-path trajectories (i.e. northing, easting, and altitude). Furthermore, there is an XML file containing parameters related to the degree of uncertainty of the stochastic variables (to be discussed in subsection C). Additional functionality allows the XML output to be exported as a Comma Separated Values (CSV) file. An overview of the headers belonging to this CSV file is available in Table 2.

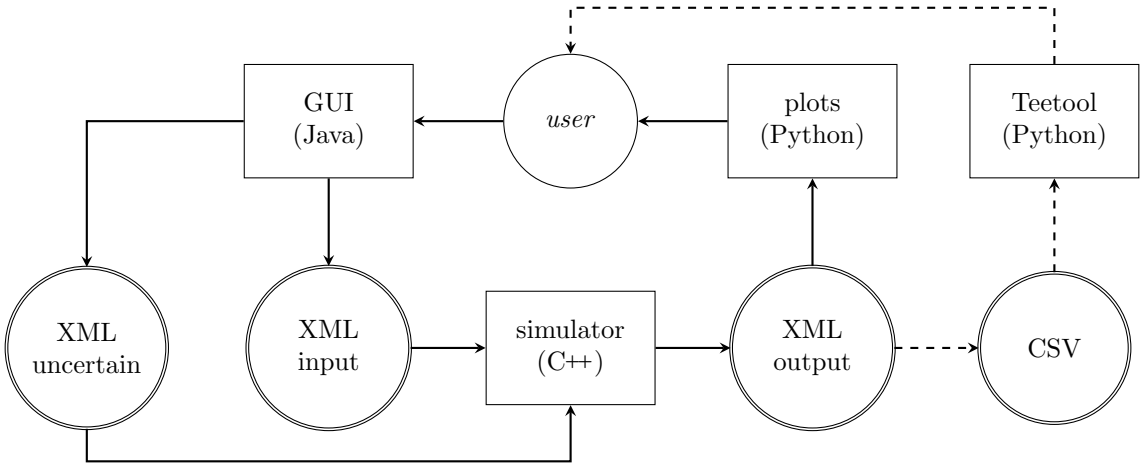


Figure 1: Schematic representation of the components and the corresponding programming languages. The dashed lines represent a path specific to the third version.

Table 2: Description of CSV file headers

Label	**Type**	**Description**
INDEX	Integer	unique index per trajectory id
ID_STR	String	trajectory identifier
ID_STAGE_STR	String	stage identifier (e.g. SingleStage)
TIME_SECONDS_DOUBLE	Double	time in seconds
EASTINGS_METERS_DOUBLE	Double	easting in metres
NORTHINGS_METERS_DOUBLE	Double	northing in metres
ALTITUDE_METERS_DOUBLE	Double	altitude in metres
DISTANCE_METERS_DOUBLE	Double	absolute distance in metres
EVENT_INT	Integer	event identifier (e.g. apogee reached)

B. Designing a rocket

Other than providing an interface to the user and executing both the C++ and Python code-base, the Java module is capable of giving a physical interpretation of the rocket design. To construct the rocket, the components seen in Table 3 are available. When components are added, they are visible in the complete rocket, as seen in Figure 2. There are two main categories; the external components, that influence the aerodynamics and the inertia, and internal components, that only influence the latter. Mass objects are a subset of the internal components, however, these are only modelled as a point-mass. Furthermore, it is possible to include a single or multiple parachutes, where each parachute holds unique properties, such as the activation altitude, drag coefficient, size, and mass. A complete overview of the components and

parameters is included in the user-guide, available in the software repository[7]. These parameters can be modified directly in the XML input file.

Table 3: Available design components

External components	Internal components	Mass objects
Nose cone	Inner tube	Parachute
Body tube	Coupler	Mass component
Transition	Centring ring	
Trapezoidal fins	Bulkhead	

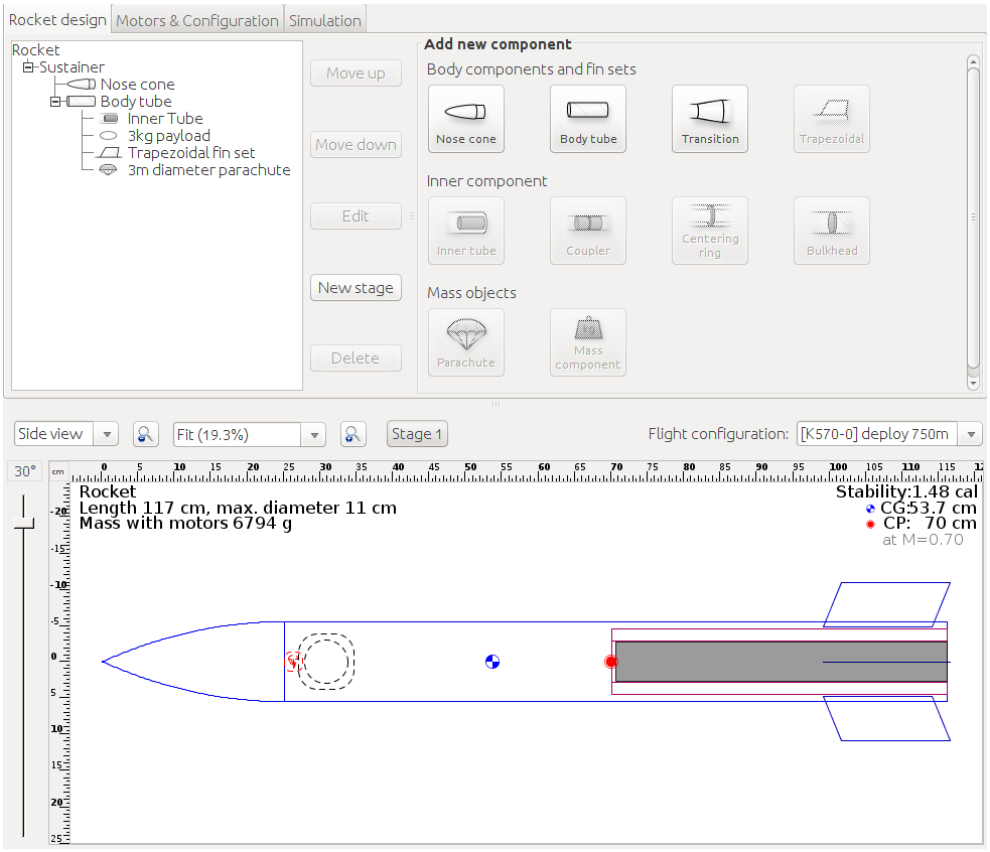


Figure 2: The rocket design tab of the user interface

C. Stochastic elements in the simulation

This subsection describes the process of capturing the uncertainty of an input variable (e.g. drag coefficient) in the output (e.g. landing location), as this is not straightforward in a system with coupled input variables. In simulating systems that have many coupled input variables such as this one, Monte Carlo methods are useful to obtain a probabilistic interpretation. In a nutshell, Monte Carlo methods work by sampling over the probability distribution of the input space, and reflect that uncertainty in the output of the simulation. While this provides a method of obtaining a probabilistic interpretation of the output, here we focus on the probability distribution of the input space.

An overview of the input variables, their probability distribution, the default values, and how they are applied can be seen in Table 4. Not all variables are directly available in the GUI (seen in Figure 3), however, those not visible can be modified in the uncertainty XML file. The probability distribution of all input variables is assumed to be normally distributed, however, the mean value, and the manner in which

the uncertainty (represented as the standard deviation σ) is applied differs. For both angles, declination and azimuth, the random variables is applied as an addition, with a mean equal to 0. The reasoning here is that the positioning uncertainty is typically independent of the variable value. For the other variables the uncertainty is applied as a multiplication, and the mean is equal to 1. The result is an uncertainty as a percentage, causing large values of the variable to have a larger variation than smaller values.

As the thrust curve consists of multiple points as a function of time, the uncertainty is modelled by drawing a sample from the normal distribution, then all thrust values are multiplied with this sample value, causing the entire curve to shift up or down accordingly. It should be noted that there is a ceiling of 250 as a maximum thrust-to-weight ratio set within the simulation. This is to prevent a scenario of unreasonable thrust that would produce numerical errors in the simulator.

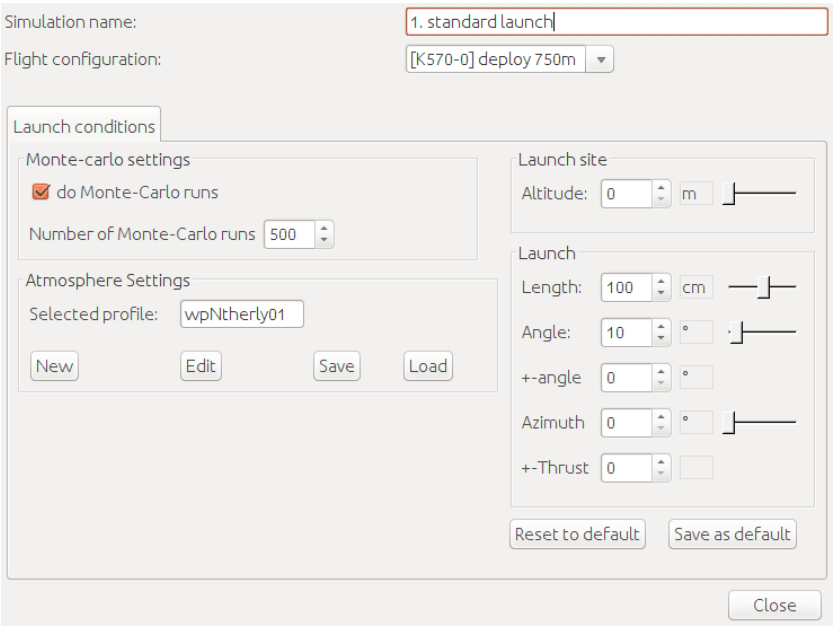


Figure 3: Launch- and atmospheric conditions tab of the user interface

Table 4: Overview of stochastic variables and their implementation

Variable	**Probability distribution**	**How applied**
Drag coefficient	$\mathcal{N}(1, \sigma^2)$, $\sigma = 0.2$	Multiplication
Centre of pressure	$\mathcal{N}(1, \sigma^2)$, $\sigma = 0.1$	Multiplication
Normal coefficient	$\mathcal{N}(1, \sigma^2)$, $\sigma = 0.1$	Multiplication
Parachute drag coefficient	$\mathcal{N}(1, \sigma^2)$, $\sigma = 0.1$	Multiplication
Thrust curve	$\mathcal{N}(1, \sigma^2)$, $\sigma = 0.01$	Multiplication
Declination launch angle	$\mathcal{N}(0, \sigma^2)$, $\sigma = 1$	Addition
Azimuth launch angle	$\mathcal{N}(0, \sigma^2)$, $\sigma = 1$	Addition

D. Modelling the rocket trajectories as a Gaussian process

The previous subsection describes the process of capturing the uncertainty of an input variable in the output via the Monte Carlo method. However, as the input variables are sampled independently from a normal

distribution, it's possible to end up with launch- and atmospheric conditions that correspond with a low probability (i.e. multiple input variables are far away from their nominal value). As the Monte Carlo method simulates more and more conditions, eventually one of these instances will result in an unstable rocket flight, but this one outlier might be a bad representation of the whole. To capture these effects, a trajectory analysis method is implemented to model the spatial distribution of the rocket trajectories as produced by the simulator. This allows for an unrestricted sampling space as the input, which is useful because there is an unknown influence of the variables on the output, while obtaining the spatial statistics based on the unfiltered output from the simulator. The task at hand is to translate the trajectory data into a mean trajectory and capture the dispersion from the mean trajectory. This subsection describes a simplified version of the original method as published in Eerland, Box and Sóbester[9], specifically suited to three-dimensional rocket trajectory data as output by the simulator, which has no missing data, nor any measurement noise. Furthermore, the time-warping approach described here is modified to handle the abrupt changes seen in the rocket trajectories (e.g. the start of the second stage, and the deployment of the parachutes).

Given the trajectory data:

$$\mathbf{v} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} \tag{1}$$

where $\mathbf{v}$ is a column vector and $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are vectors containing the coordinates easting, northing, and altitude respectively. Therefore, when a trajectory has M coordinates, $\mathbf{v}$ is a $(3 \cdot M) \times 1$ vector. As the data are produced by the simulator, information about position, time, and the particular stage of flight is available. By definition, all trajectories start at the normalised time $\tau = 0$, and end at $\tau = 1$. There are time-warping methods available to align the behaviour of the trajectories[16], however, such a method is not required here as the moments that these events occur are well known. These are the moments where there is a change from one stage of flight to the next, for example when the rocket reaches apogee. By synchronizing these moments in the normalised time vector $\boldsymbol{\tau}$, the time-warping is complete. For this reason the label EVENT_INT is included in the CSV file (seen in Table 2), as it refers to a particular stage of the flight. Hence, the normalised time vector $\boldsymbol{\tau}$ to accompany the coordinate vector $\mathbf{v}$ is equal to:

$$\boldsymbol{\tau} = \begin{bmatrix} 0 & \frac{1}{M} & \dots & \frac{M}{M} \end{bmatrix}^\intercal \tag{2}$$

where the points in time when the trajectory changes from one stage of flight to the next are aligned. The values of these points in time are set by the nominal flight. For example, if the nominal flight reaches apogee at $\tau = 0.3$, all trajectories reach apogee at $\tau = 0.3$.

Now, with the additional assumption that the rocket travels linearly between data-points, the coordinate vector $\mathbf{u}$ can be expressed as a function of the normalised time τ:

$$\mathbf{u}(\tau) = [x \ y \ z]^\intercal \tag{3}$$

where x, y, and z are the coordinates from $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, linearly interpolated as a function of $\boldsymbol{\tau}$. All trajectories can now be resampled to a uniform size to the column vector $\hat{\mathbf{v}}$. The structure is identical to $\mathbf{v}$, seen in Equation (1), however, the size is identical between all trajectories. In this paper 100 steps are taken, thus $\hat{\mathbf{v}}$ is a 300×1 vector, for all N trajectories found in the data. The mean vector $\boldsymbol{\mu}$, and the covariance matrix $\boldsymbol{\Sigma}$ are calculated following:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^{N} \{\mathbb{E}[\hat{\mathbf{v}}_n]\} \tag{4}$$

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} \{\hat{\mathbf{v}}_n \hat{\mathbf{v}}_n^\intercal - 2\hat{\mathbf{v}}_n^\intercal \boldsymbol{\mu} + \boldsymbol{\mu}\boldsymbol{\mu}^\intercal\} \tag{5}$$

where

$$p(\mathbf{u}(\boldsymbol{\tau})) \sim \mathcal{N}(\mathbf{u}(\boldsymbol{\tau}) \ |\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{6}$$

This solution fits into a Gaussian Process (GP) framework, where a function is defined as a Gaussian process by a mean function $\mathbf{m}(\tau)$ and a covariance kernel $\mathbf{k}(\tau, \tau')$:

$$p(\mathbf{u}(\tau)) \sim \mathcal{GP}(\mathbf{m}(\tau), \mathbf{k}(\tau, \tau')) \tag{7}$$

where

$$\mathbf{m}(\tau) = \mathbb{E}[\mathbf{u}] = \boldsymbol{\mu} \tag{8}$$

$$\mathbf{k}(\tau, \tau') = \mathbb{E}[(\mathbf{u}(\tau) - \mathbf{m}(\tau))(\mathbf{u}(\tau') - \mathbf{m}(\tau'))] = \boldsymbol{\Sigma} \tag{9}$$

A more intuitive representation of the mean function $\mathbf{m}(\tau)$, given by Equation (8), is a point that moves along a three-dimensional trajectory as a function of τ. In turn, the covariance kernel $\mathbf{k}(\tau, \tau')$, given by Equation (9), can be represented as a three-dimensional ellipsoid at a constant probability. The reference to the GP framework is made, as it provides the mathematical tools necessary to handle noise measurements and missing data, to be applied when either is a problem.

There are several advantages to this approach. Firstly, it is shown how a probabilistic model can capture the general trend and dispersion seen in the trajectory data. The direct application here is the ability to visualise the behaviour of a large dataset without cluttering the screen. Besides helpful in visualising, it also allows *quantifying* the difference between the scenarios. The latter is done by evaluating the number of grid-points found within the one standard deviation ($\sigma = 1$) region for multiple scenarios. Changes in these points reflect changes in the trend modelled from the trajectory data.

III. Results

This section demonstrates the effects of the stochastic input variables to the output trajectory data. For each of the four scenarios seen in this section, 500 trajectories are generated via the Monte Carlo method as described in subsection C. It shows the results of modelling the output data as a Gaussian process, using the additional information on the events as discussed in subsection D. All 500 trajectories produced by the rocket flight simulator are used to learn the corresponding probabilistic model, however, to prevent visual clutter only 50 of the 500 trajectories are shown in the figures.

The flight-path trajectories seen in Figure 4 are generated using the settings seen in Figure 3. In this simulation the wind conditions represent a light northerly wind (no more than 2.2 ms^{-1}), and the parachute deploys at an altitude of 750 metres. Specific launch conditions are a declination angle of 10 degrees, and an azimuth angle of 0 degrees (i.e. launched towards the North). The volume identifying the trend corresponds with the one standard deviation region ($\sigma = 1$).

The first alternate scenario changes the wind-conditions to a strong southerly wind (with an average wind speed up to 13.5 ms^{-1}). An overview of both the northerly and southerly wind-conditions is available in Figure 5. Northerly and southerly are merely labels, and these wind-conditions represent real data measured by a weather balloon. The x-wind and y-wind components correspond with the easting and northing directions respectively, where both vary as a function of altitude. Both wind-conditions have a 0 ms^{-1} vertical wind component. The results are visible in Figure 6, where again 50 trajectories and the one standard deviation region ($\sigma = 1$) are shown. The comparison plot with the standard launch is seen in Figure 7. The blocks used to construct this volume have a volume of 20m$\times$20m$\times$20m, and the blocks relating to the addition, removal, and lack of change in the standard launch versus the southerly wind scenario are coloured green, red, and blue respectively. While the declination angle of the launch tower is 10 degrees in both cases, the southerly wind has a steeper trajectory. This is due to an effect called weathercocking, which was a key part of the validation of the rocket simulator[8].

The second alternate scenario introduces a failure to deploy the parachute, resulting in a fully ballistic flight trajectory. The resulting flight trajectories are visible in Figure 8. The comparison with the standard launch is visible in Figures 9 and 10, whereas expected, the final part of the trajectory shows the difference between the parachute- and ballistic landing. The failure to deploy the parachute is only visible in the final 750m of the descent, as this is where the parachute deploys. Figure 11 shows a two-dimensional slice of the one standard deviation region at a constant altitude for both 750m and 300m. It also includes a two-dimensional projection of the mean trajectory belonging to both scenarios, represented by a dashed line. The dispersion seen until the 750m is almost (it *is* an approximation via a Monte-Carlo method) identical. The models diverge after this point, which is visible in Figure 11b.

Finally, the third alternate scenario introduced a 5% variation in the thrust curve, according to the method discussed in subsection C. This method increases/decreases the thrust curve over the entire duration, where the percentage is sampled from a normal distribution. Similar to the other plots, the 50 trajectories and the one standard deviation region are visible in Figure 12. The comparison plot is seen in Figure 13, indicating that the thrust has a strong correlation with the steepness of the launch trajectory, an effect that can be

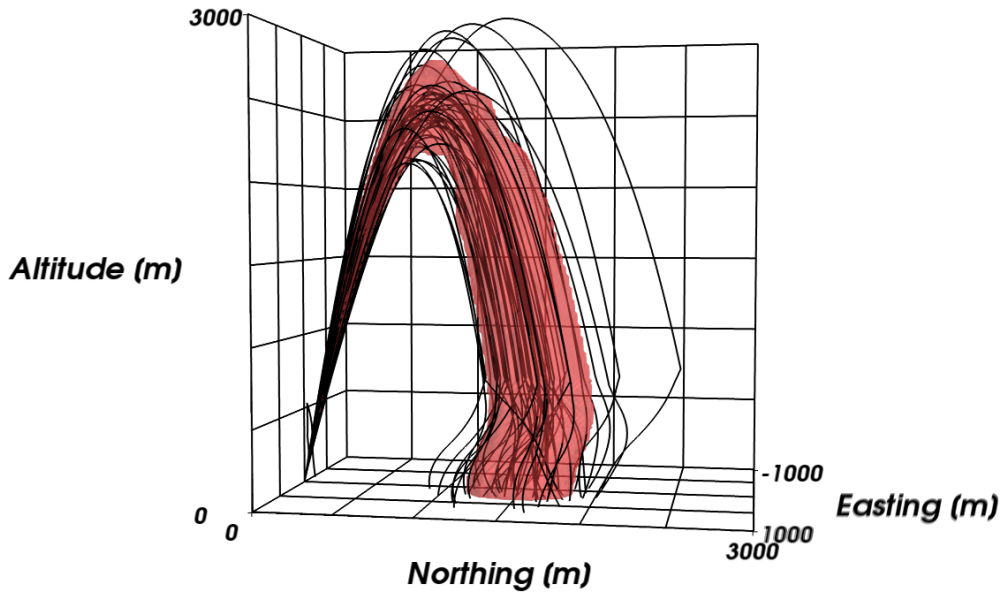


Figure 4: Trajectories produced by standard launch conditions, and the one standard deviation region

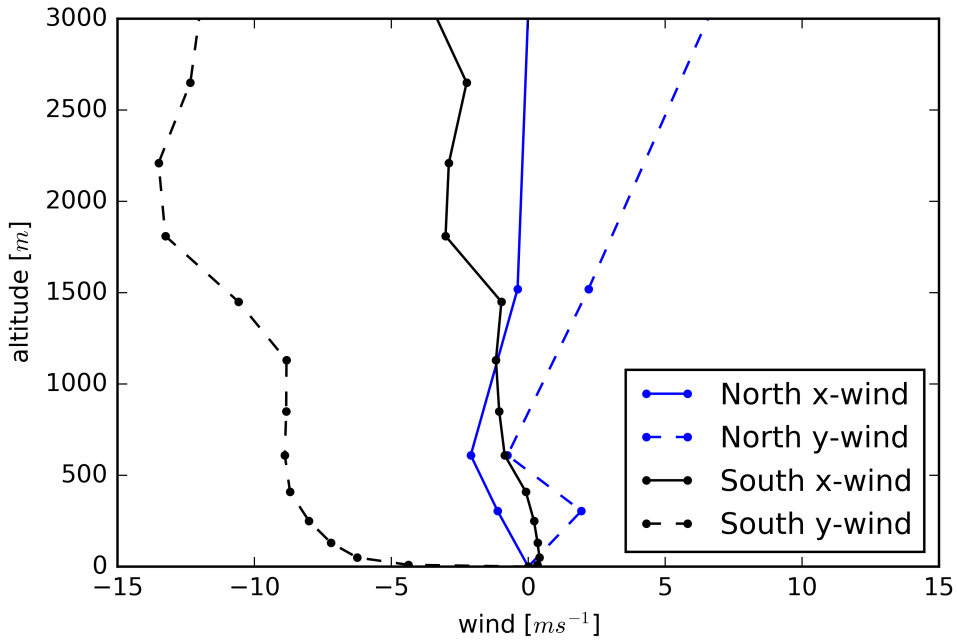


Figure 5: Wind conditions north(erly) wind versus south(erly) wind

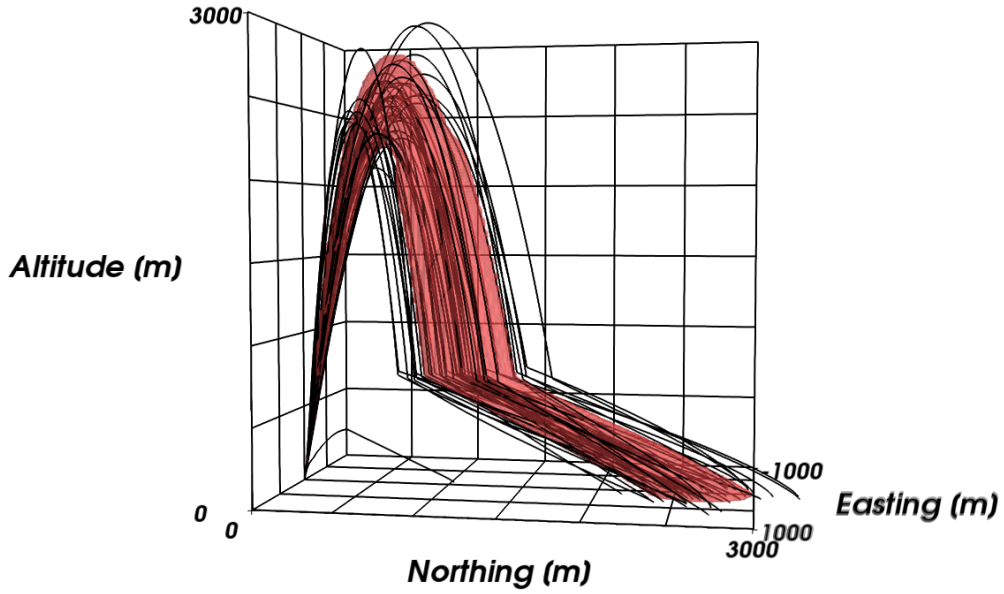


Figure 6: Trajectories produced by southerly wind conditions, and the one standard deviation region

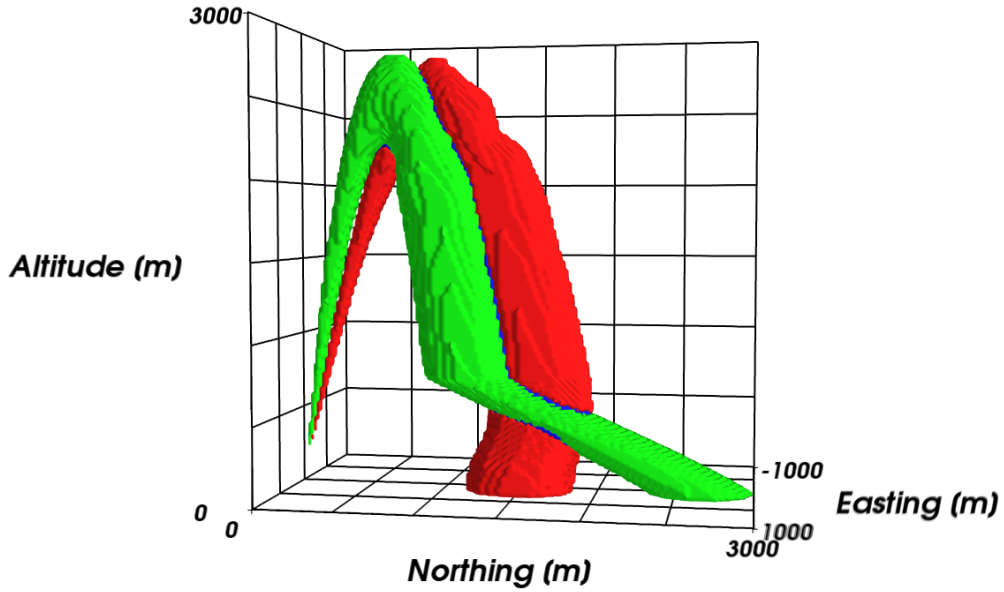


Figure 7: Comparison of the one standard deviation region in the standard launch conditions versus the southerly wind conditions

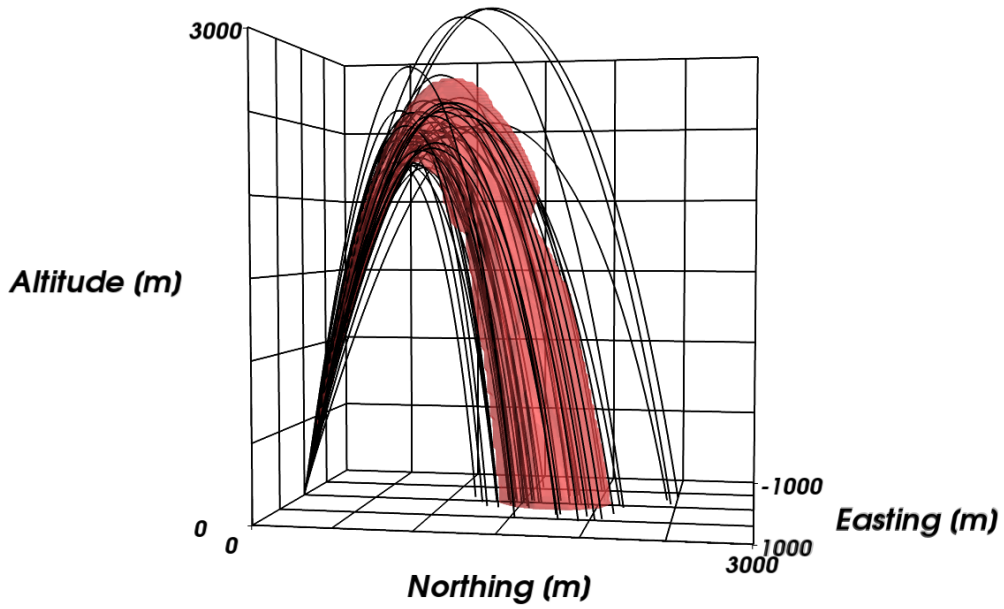


Figure 8: Trajectories produced with the parachute failure condition, and the one standard deviation region

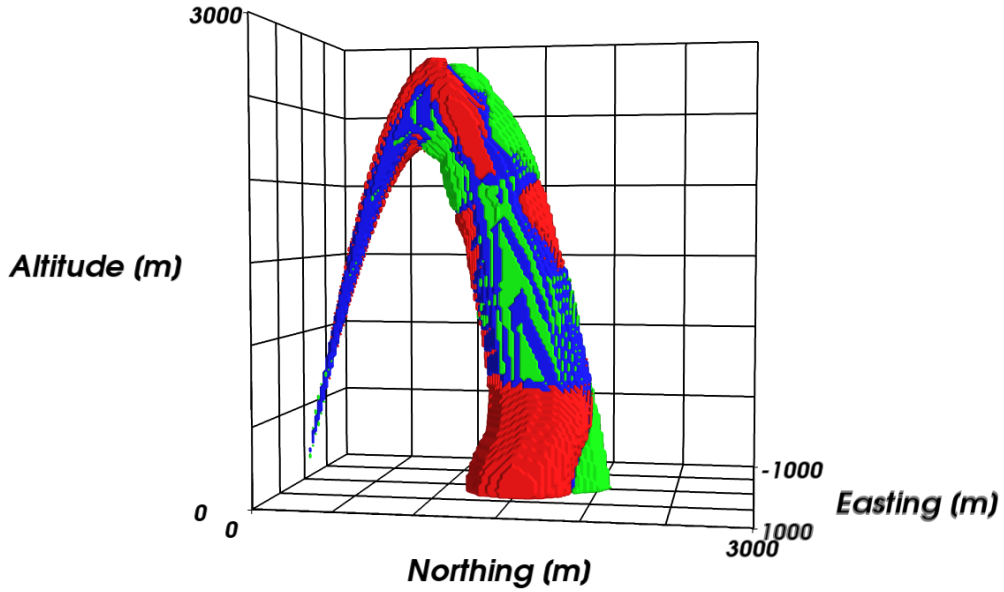


Figure 9: Comparison of the one standard deviation region in the standard launch conditions versus the parachute failure condition

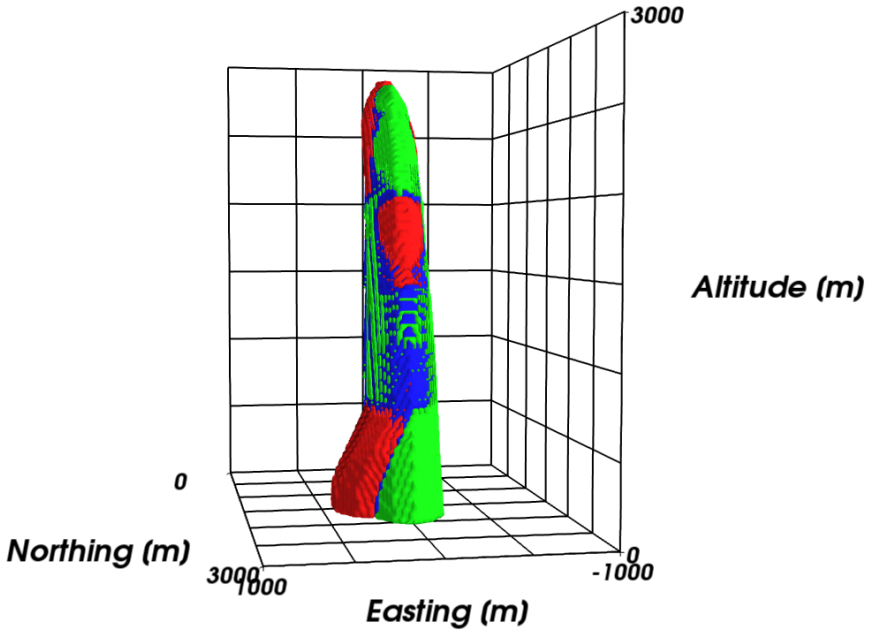
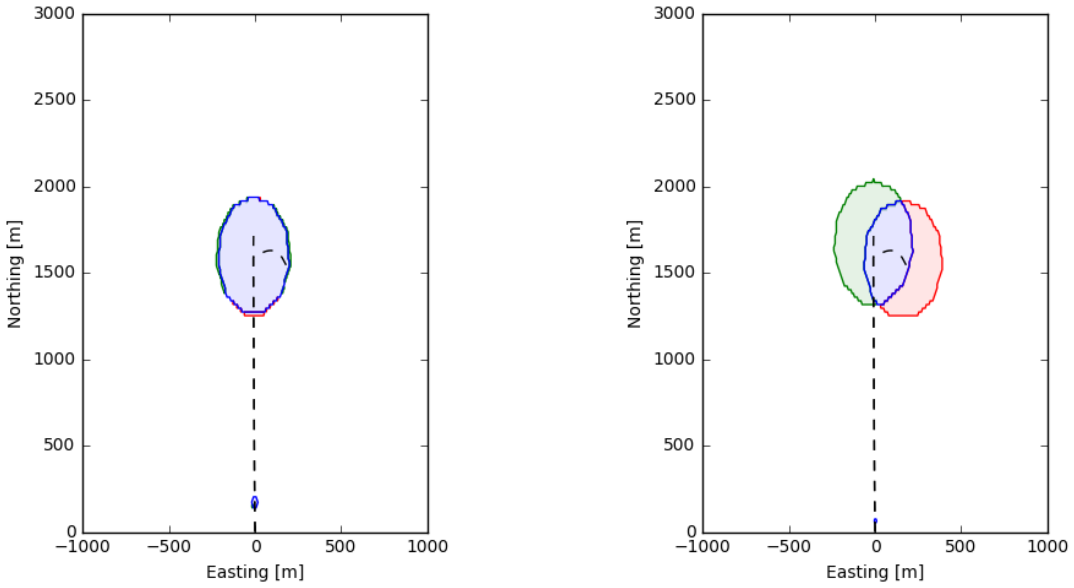


Figure 10: Comparison of the one standard deviation region in the standard launch conditions versus the parachute failure condition, alternate view-point



(a) 750 m

(b) 300 m

Figure 11: One standard deviation region at a constant altitude, comparing the standard launch conditions versus the parachute failure condition

attributed to weathercocking. The two-dimension projection of the one standard deviation region is visible in Figure 14. At a constant altitude of 2100m (Figure 14a) the effect of the thrust variation on the dispersion in the launch phase is pronounced, however, closer to the ground, at 500m (Figure 14b) the differences in divergence between the two scenarios have reduced.

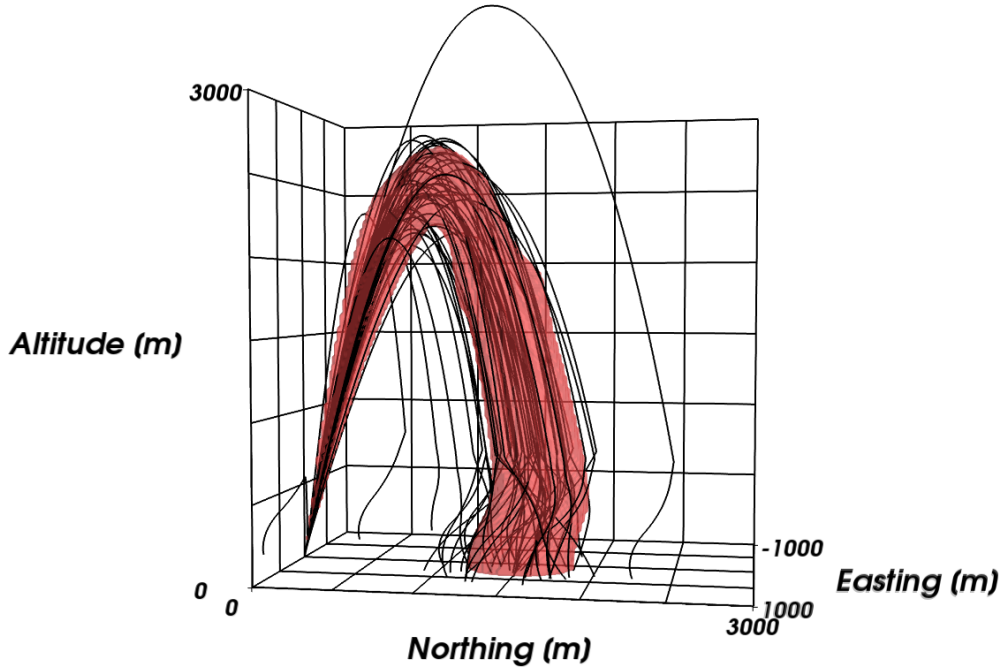


Figure 12: Trajectories produced with the variable thrust curve condition, and the one standard deviation region

The numerical results that quantify the changes seen between the scenarios are visible in Table 5. To demonstrate the sensitivity of the block size, the percentages have been calculated at varying sizes of 100m, 50m, and 20m in all three dimension (easting, northing, altitude). The percentages are related to the total number of blocks identified to be inside the one standard deviation region for the standard conditions, i.e. the region as seen in Figure 4, hence the percentage *removed* and *unchanged* adds up to 100%. The difference in the percentage added and removed relates to the uncertainty region expanding or shrinking with respect to the standard conditions. Based on the percentage unchanged, the variable thrust curve shows the least change, followed by the parachute failure conditions, and the changed wind condition shows most change.

IV. Conclusion

This paper demonstrates the functionality of the Cambridge Rocketry Simulator. The open-source software assists the user in designing a concept rocket, and flying it in a spectrum of flight conditions based on stochastic input variables. The rocket trajectory data available as output from the simulator are modelled using a trajectory analysis approach, which is both capable at visualising trends, and quantifying the change found in different launch- and atmospheric conditions.

The modification to the time-warping in the original trajectory analysis approach, expanding the functionality to accept *time-cues* from the simulator output, removes undesired artefacts previously present when modelling rocket trajectories as a Gaussian process. The *time-cues* here will be those moments where discrete changes occur (e.g. initialisation of the second stage, parachute deployment).

Finally, while the method is capable of modelling the rocket trajectory data as output by the simulator, the trajectories should not always be modelled as a Gaussian process. Such a situation arises when the rocket concept design is close to being dynamically unstable, resulting in a discrete path change for numerous

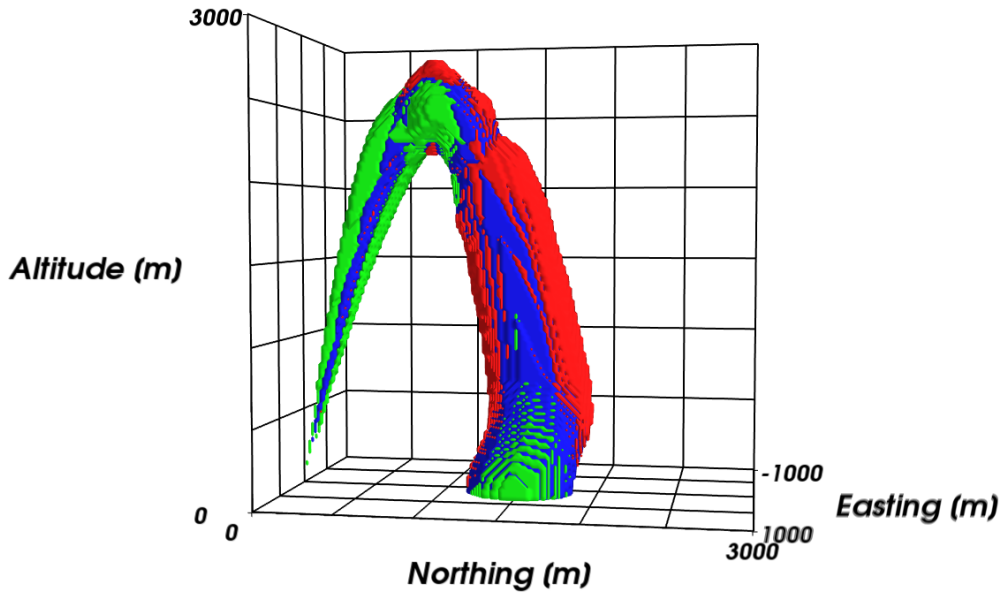
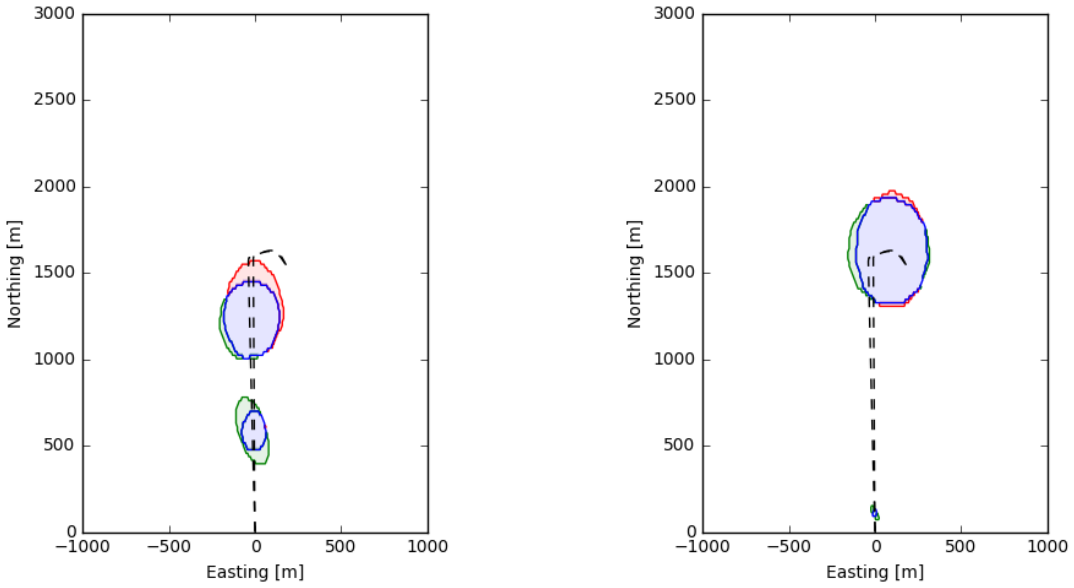


Figure 13: Comparison of the one standard deviation region in the standard launch conditions versus the variable thrust curve condition



(a) 2100 m

(b) 500 m

Figure 14: One standard deviation region at a constant altitude, comparing the standard launch conditions versus the variable thrust curve condition

Table 5: Comparing against the standard conditions

Block volume $[m^3]$	**Added**	**Removed**	**Unchanged**
versus southerly wind conditions			
$100 \times 100 \times 100$	79.7%	78.1%	21.9%
$50 \times 50 \times 50$	76.1%	78.5%	21.5%
$20 \times 20 \times 20$	75.4%	78.3%	21.7%
versus the parachute failure condition			
$100 \times 100 \times 100$	22.4%	15.7%	84.3%
$50 \times 50 \times 50$	20.3%	17.5%	82.5%
$20 \times 20 \times 20$	20.7%	18.1%	81.9%
versus the variable thrust curve condition			
$100 \times 100 \times 100$	21.1%	8.1%	91.9%
$50 \times 50 \times 50$	18.2%	8.6%	91.4%
$20 \times 20 \times 20$	17.6%	8.6%	91.4%

trajectories as the uncertainty analysis varies the centre of pressure. The trajectory analysis method as presented in this paper assumes the trajectories only deviate slightly from the nominal flight-path, therefore when discrete path changes are present in the data, a clustering algorithm should be applied. The trajectory data can also be analysed via other methods, as the output data from the simulator are openly available in both XML and CSV file format.

Acknowledgements

The authors gratefully acknowledge the funding provided under research grant EP/L505067/1 from the Engineering and Physical Sciences Research Council, and the industry sponsor Cunning Running Software Ltd. Supplemental material supporting this study are openly available from the University of Southampton repository at `https://doi.org/10.5258/SOTON/403364`.

References

[1] UKRA. *United Kingdom Rocketry Association*. 2016. URL: `http://www.ukra.org.uk/events` (visited on 02/11/2016).

[2] BBC. *Rocket launch for Bolton students on Mull of Galloway*. 2011. URL: `http://bbc.co.uk/news/uk-england-manchester-14760906` (visited on 02/11/2016).

[3] TU Delft. *DARE: Delft Aerospace Rocket Engineering*. 2016. URL: `http://dare.tudelft.nl` (visited on 02/11/2016).

[4] Commercial Space Transportation. *Supplemental Application Guidance for Unguided Suborbital Launch Vehicles (USLVs)*. Tech. rep. Federal Aviation Administration (FAA), 2007. URL: `https://www.faa.gov/`.

[5] Apogee. *RockSim*. Version 9. 2008. URL: `http://www.apogeerockets.com` (visited on 02/11/2016).

[6] S. Niskanen. *OpenRocket*. Version 15. 2015. URL: `http://openrocket.sourceforge.net` (visited on 02/11/2016).

[7] S. Box and W. J. Eerland. *Cambridge Rocketry Simulator*. Version 3.1. 2016. URL: `http://www.sourceforge.net/p/camrocsim` (visited on 02/11/2016).

[8] S. Box, C. Bishop and H. Hunt. 'Stochastic six-degree-of-freedom flight simulator for passively controlled high-power rockets'. In: *Journal of Aerospace Engineering* 24.1 (2010), pp. 31–45. DOI: `10.1061/(ASCE)AS.1943-5525.0000051`. eprint: `http://eprints.soton.ac.uk/73938/`.

[9] W. J. Eerland, S. Box and A. Sóbester. 'Modeling the Dispersion of Aircraft Trajectories Using Gaussian Processes'. In: *Journal of Guidance, Control, and Dynamics* 39.12 (28th Nov. 2016), pp. 2661–2672. DOI: 10.2514/1.G000537. eprint: http://eprints.soton.ac.uk/399818/.

[10] W. J. Eerland. *Teetool - a trajectory analysis tool*. 2016. URL: http://github.com/WillemEerland/teetool (visited on 02/11/2016).

[11] J. D. Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[12] P. Ramachandran and G. Varoquaux. 'Mayavi: 3D Visualization of Scientific Data'. In: *Computing in Science & Engineering* 13.2 (2011), pp. 40–51. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.35.

[13] S. v. d. Walt, S. C. Colbert and G. Varoquaux. 'The NumPy Array: A Structure for Efficient Numerical Computation'. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30. DOI: 10.1109/MCSE.2011.37.

[14] W. McKinney. 'Data Structures for Statistical Computing in Python'. In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56.

[15] T. Kluyver et al. 'Jupyter Notebooksa publishing format for reproducible computational workflows'. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas: Proceedings of the 20th International Conference on Electronic Publishing*. IOS Press. 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[16] M. Müller. 'Information Retrieval for Music and Motion'. In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Chap. Dynamic Time Warping, pp. 69–84. ISBN: 978-3-540-74048-3. DOI: 10.1007/978-3-540-74048-3_4.