Improving climb and descent profiles in BlueSky using ATC performance models

Master thesis report

Folkert C. Schornagel

TU Delft Delft University of Technology

March 7, 2022

Preface

This report wraps up the research conducted for the master thesis. This research has been a challenging one. I decided to step out of the familiar research field I was in, and went into the field of ATM. Although the research has taken quite the time, I look back at a period in which I significantly developed several skills. Technically this meant personally improving in Python. This was a strong desire, as I really wanted to leave the university knowing that programming had become a strong skill of mine. Although such a technical skill is helpful, a master thesis also helps very much in gaining the according organisational skills. This means planning, but also means staying the leader in your own project. It has been a challenging process, but one I look at with content as I did my best.

There are a certain amount of people I would like to note down and thank for their work, as they have helped very much during this research. Each has done so in their own way.

First and foremost, but perhaps also evidently, is the staff that helped me during this research. That is, Jacco Hoekstra, Joost Ellerbroek and Junzi Sun. Both Joost and Junzi have helped me in the first stages of the research to set up all necessities. Although a sense of shame arises when I think back at the silly questions I have asked them, I am grateful that they treated each question with seriousness.
Jacco, although the rush of our meetings was sometimes an annoyance, I am grateful for the very accurate guidance throughout the project. I do believe you truly understand the core of what such a research project is supposed to teach a student. My choice for you was based on your empathy, I can gladly say that choice was correct.[1] Moreover, during the whole research I have lived on on the long talks we had at the start of the project. COVID-19 was just a few months old, and we spent countless hours discussing the exponential functions associated with the spread of the disease. That social aspect of working together is something I have very much appreciated.

Then, a very big thank you to George. When I believed my entire model was failing, you were the one to tell me on a tuesday afternoon: ”we’ll fix this”. Initially I truly did not believe that we would, but we eventually did. Words fail to describe how much you have helped me, as you spent countless hours with me on Spyder code in your free time. It shows what you would do for a friend, and I am truly grateful for that.

The other big contributor: a special thanks to my brother. Throughout the research you have helped me in several ways. This started off with coding, but ended up in repeatedly completely wrecking my way of writing. The amount of comments sometimes blurred my ability to appreciate that you were helping me. In the end, we both know that your proposed way of writing was the better one.

Lastly, thank you mom and dad, uncle Folkert, aunt Esther, Pien, Charlotte, Hugo, Stijn, Simon, Riemer, Marijke and Jip for all the countless hours I have spent complaining about graduating. Although it might not always have seemed so, it made graduating bearable.

I wish you a pleasant read,

Folkert Schornagel
Rotterdam
March 7, 2022

[1] As reflected by Jacco himself: https://www.tudelft.nl/lr/organisatie/onze-hoogleraren/profiel-van-een-prof/jacco-hoekstra

Summary

As air traffic continues to grow, research into air traffic control (ATC) is essential to maintain the capacity of airports and ensure the safety of all. It is essential to perform simulations in order to test conceptual ideas. However, present day ATC researchers create simulated environments, but these are plagued by inconsistencies. New concepts are proposed using custom defined metrics, but this in itself leads to a biased solution.
The ATC simulator BlueSky (built by the TU Delft) strives to become a solution to that problem. By increasing the functionality of BlueSky, users are able to simulate their own challenging scenarios and find answers to ATC problems. Currently, BlueSky contains robust, basic simulation performance but lacks functionality for advanced use-cases. This performance is achieved by using one of the performance models present in BlueSky, being OpenAP or BADA.

BlueSky contains the option to simulate aircraft using either a set speed or altitude with an optional given vertical speed. Thrust is maintained using an autothrottle controlled by the autopilot. As this significantly limits the options, the following main question was set up as basis of the research: ”How can BlueSky and its performance models be expanded to ensure that the simulator itself, during climb and descent phases, is able to simulate representative flight behaviour while refraining from using full autopilot logic?”. As the question suggests, climb and descent behaviour was the main focus of this paper.

The solution of which was the use of thrust commands in combination with the option to set the flight path angle. The methodology to address this objective and its proposed solution has been as follows. BlueSky required a new conceptual logic which would be implemented in its core code. By defining a command sequence logic, it could be determined whether vertical speed, speed or throttle was the main variable to fulfil the equations of motion. This implemented logic was to be validated using real aircraft data. This data had to be processed as input for BlueSky. The data was reduced using the Ramer-Douglas-Peucker algorithm, and subsequently Fuzzy logic was used to identify the according flight phase of each segment. Using an energy balance, thrust settings were derived to obtain a full flight command format. These consisted of combinations of speed, altitude, flight path angle and thrust settings.

Results showed different outcomes regarding performance. Descent flights were accurately simulated, whereas climb flights showed significant issues with the thrust setting. This difference was attributed to the fact that descent flights depend far less on the thrust setting than climb flights. Primary reason for the incorrect thrust settings is that OpenAP showed surprising behaviour when it came to force balance. This caused the thrust setting (as result of the preprocessing) to be unrealistic, and led to the fact that this caused too much discrepancy when simulating using either BADA or OpenAP. A further sensitivity analysis supported this claim by showing that a small change in values causes significant difference. Overall, this prevented the model from being validated using large amounts of data.
Therefore, it was concluded that thrust settings and flight path angle functionality are a step in the enabling of simulation of advanced use cases. That is, representative flight behaviour is in place for descent flights but climb flights require more improvement, as basic rule of thumbs present in BlueSky have not been excelled by this new model.

Further research recommends looking into the role of the flight phase identification and how that affects performance. Furthermore, preprocessing and execution should both be done using model that is capable of ensuring a correct output when considering force balance. Lastly, if possible a separate validation of both the input values as implemented model would ensure a more robust methodology.

Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

ADS-B	Automatic Dependent Surveillance-Broadcast
ATC	Air Traffic Control
BADA	Base of Aircraft Data
CAS	Calibrated Airspeed
CDA	Continuous Descent Approach
DoF	Degrees-of-Freedom
FL	Flight Level
FMA	Flight Mode Annunciator
FMS	Flight Management System
FPA	Flight Path Angle
GUI	Graphical User Interface
LNAV	Lateral Navigation
MCP	Mode Control Panel
MTOW	Maximum Take-Off Weight
OEW	Operating Empty Weight
OOP	Object-Orientated Programming
PFD	Primary Flight Display
RDP	Ramer-Douglas-Peucker
RoC	Rate of Climb
ROM	Reduced Order Model
TAS	True Airspeed
TOC	Top of Climb
TOD	Top of Descent
VNAV	Vertical Navigation

List of Symbols

γ	Flight path angle
ρ	Density
ρ_0	Density at sea level
ax_{max}	Maximum acceleration
D	Drag
D_x	Distance
E	Total energy
L	length
M	Mach
m	mass
p	Pressure
p_0	Pressure at sea level
q_{dyn}	Impact pressure
R	Gas constant
T	Thrust, Temperature
T_{max}, T_{avail}	Maximum available thrust
T_{net}	Net thrust
T_s	Thrust setting
V	Speed
W	Weight

1

Introduction

Unveiling a new, multi-million, aircraft engine is often a showcase of brilliant engineering, though it only partly provides the answer to current day challenges.

Ever since humans started flying, aircraft have made impressive strides. From a simple flight lasting 12 seconds more than a hundred years ago, aircraft nowadays are capable of traversing entire continents. Improvements on aircraft have decreased fuel consumption and increased efficiency by large amounts, but how much is this all worth if incorrectly used?
Now, it does not mean that all aircraft are flown by laymen, but efficiently using these machines is something different than creating efficient machines. To do so, allocation of aircraft is a topic which is just as important of an research field than the aircraft itself. Aircraft are controlled from towers at airports and other command centres, which is the basis of Air Traffic Control (ATC). Controllers try to maintain a safe airspace, whilst creating and maintaining as much capacity as possible to ensure a continuous flow of traffic. Combining both research into the allocation of aircraft, as the research into the aircraft itself, provides the necessary combination to take further steps in reducing the carbon footprint of aircraft overall.

Optimising the relation between safety and maximising the traffic throughput is a delicate one, where ATC research comes into play. Concepts are theoretically researched, but should be simulated before being tested in practise. Simulators provide an environment in which real-world scenarios with numerous aircraft can be reproduced. This creates a safe space to investigate novel concepts. However, that is exactly where the problem is, and actually is twofold. Some concepts are still difficult to even research, as the amount of simulators with extensive capabilities is thin spread. Secondly, many researchers creating simulated environments propose new concepts using custom defined metrics. This often leads to an improved programme, but actually lacks robustness as conditions were optimal. Comparing results among different simulations is often a difficult task, leading to different parties claiming their solution is 'the way to go' [1].

BlueSky sets itself apart achieving a simulator that is free of bias. BlueSky is an regular ATC simulator, but distinguishes itself by being based on different concepts than current simulators. Results should become more comparable by stimulating commonality, e.g. metrics. To create such a platform, multiple hurdles are to be solved. One of which is the scarcity of data, as aircraft performance has to be modelled. Aircraft manufacturers hardly give out any specific data on aircraft. In this case, one can think of drag or lift performance, which is essential in setting up characteristic curves for aircraft. To surpass this issue, performance models have been created. These models can be seen as an interpretation of the aircraft its overall performance, based on empirical and historical data. BlueSky itself contains two major performance models, being OpenAP and BADA. The first is created at the TU Delft by Junzi Sun, where the latter is a cooperative effort from Eurocontrol [2, 3].

1.1 Problem definition

The challenge within BlueSky, and its performance models, is that it is still very much in development phase. This means that many use cases and scenarios are not ready to be simulated yet, as BlueSky lacks that specific functionality. Currently, BlueSky relies very much on autopilot behaviour. Meaning, users are able to define speeds and altitudes, but are hardly given the option to further specify the aircraft its settings. This significantly impacts the climb and descent performance of BlueSky. Aircraft are flown with the autothrottle turned on, even in descent, which causes aircraft to have little freedom in *how* the descent is executed. Moreover, the different performance models approach the problem in their own way. Expanding the possibilities en ensuring a consistent interface among the performance models are two prerequisites to create a more capable simulator.

Therefore, the problem at hand is the expansion of BlueSky to function seamlessly for users with detailed knowledge of aircraft performance and autopilot logic, while maintaining the user-friendliness for users without said knowledge. Users who want to control whether a descent is an idle descent, defined by thrust setting or fixed flight path angle need this freedom. Here, thrust setting is defined as the possibility of virtually controlling the thrust levers, whereas the flight path angle describes the angle between horizontal and vertical speed.

1.2 Research question & objective

To tackle this problem, the main research question has been formulated as follows:

Research question
”How can BlueSky and its performance models be expanded to ensure that the simulator itself, during climb and descent phases, is able to simulate representative flight behaviour while refraining from using full autopilot logic?”

The research question poses a challenge to improve the climb and descent performance. As stated in the problem definition, this can be achieved by increasing the *functionality* during climb and descent. The problem definition already gave away that providing the functionality of an idle descent or fixed flight path angle are indicated solutions to the research question. Therefore, the objective of this research is as follows:

Research objective
“To achieve a consistent, advanced performance model interface for BlueSky to create more representative flight behaviour, by means of disabling the automation and expanding - among every performance model- the advanced user input of thrust setting and flight path angle whilst maintaining the user-friendliness of BlueSky”.

To aid in answering this question, and reach the research objective, several sub questions were set up to provide intermediate steps. These questions are as follows:

- How is current autopilot logic able to cope with manual thrust commands and how does this differ from autothrottle situations?
- How do current performance models like OpenAP and BADA simulate kinematic behaviour and how is this different from so called physics/kinetics approach and real world performance?
- What is the current logic of BlueSky regarding given commands and how can this logic cycle be further improved using new commands like a thrust setting?
- What type of input parameters and data sets should be used to create an equal playing field for comparison between the original Bluesky and its updated version?

These sub questions were drawn up as they represent a process from what a current situation is, towards a new logic. In the end, the research should show that extra functionality has been added, and is properly working. This can only be done by properly investigating what is intuitive behaviour and thus basis for new functions.

1.3 Scope

Simulating means doing concessions. Never has aircraft performance been simulated with results identical to real world behaviour, as this requires advanced data and an vast amount of resources. So, knowing to what the research is limited, how should the results be interpreted?

The scope of this research limits itself to increasing the performance in descent and climb performance. This means that the cruise phase is left out of this research, as it has less characteristics that require improvement. Furthermore, BlueSky and OpenAP have a certain amount of aircraft that are supported. Therefore, the research will only use those rather than every aircraft available on the market. The aircraft data used is ADS-B data as filed by the receiver. This means that weather conditions are not taken into account, as the data lacked this information. Lastly, during the research an emphasis will lie on the performance model OpenAP. As BADA is becoming more and more a proprietary model, it is more useful to improve the open-source model. However, BADA will be intensively used throughout the research as it is a proven and working model.

Scoping limits the research and can also impose certain simplifications. Every simulator has a certain degree of accuracy and so does BlueSky, but it is important to realise how this accuracy should be used. Simulators like BlueSky approximate aircraft using point-mass models. This means that with 'representative behaviour' (as stated in the research question), an approximation of actual behaviour is meant. Results should give a indication whether an aircraft can travel a certain distance using set settings, but should not be seen as a high fidelity model. Therefore, BlueSky should give a proper indication whether a certain command or move is feasible, it however does not calculate 1-on-1 performance of aircraft when compared to reality.

1.4 Research approach

This leaves the question on how to set about in this research. This section describes the 'how' of the research question.

Answering the main question requires a validated answer. Several steps exist in this research to come to a properly reasoned answer. As stated earlier, knowing what is intuitive and in accordance with reality is an important starting point. If implemented functions work well but leave a lot to be answered as to how they should be used, their impact is reduced significantly. Instead, by using comparable equipment a good understanding is attained of present day aircraft logic. Moreover, to then validate what has been created, an extensive processing of data has to take place. Using real aircraft data, the new command structure can be tested by implementing commands following those specific trajectories. This then provides the opportunity to compare the result of such a command structure to the real flight. This should give an indication whether the new logic provides a proper alternative to the current functionality.

1.5 Contribution

Research without purpose results in little motivation to strive for the most optimal result. Research should always provide a certain contribution in order to be relevant.

As stated in the introduction, ATC research has an important role in aviation research. Contributions in itself to ATC research are meaningful as they provide new insights to this field. However, current ATC research is hindered by the little constructive collaboration taking place in this field. Researchers pose their own solutions as optimal using self defined metrics. As this provides a limited answer to other researchers, the result is hardly usable at all. It is difficult to determine whether the solution is really most optimal, or is simply a push in a certain direction. By improving BlueSky, the need for these alternative simulations becomes less. This, by no means, means that BlueSky should become the sole reference in this field, but it would provide a good standard in how customers can do their own research into relevant concepts.

This bring the question to the importance of this research specifically. Not only does it contribute to the level of scale mentioned above, it also contributes to BlueSky itself. Thrust settings can be seen as a basic input for pilots, therefore necessary to be implemented as soon as possible. Ideas like these and path angles are needed to bring BlueSky to a point where it contains all the basic functionality to be a proper alternative. Providing BlueSky with thrust settings also enables the research into idle thrust when looking on a larger scale at this functionality. Aircraft are more than their engines alone, and also contain a lot of potential when it comes to glide flights or other manoeuvres executed with as minimal engine power as possible. Reduction in thrust means a reduction in noise and air pollution, two important factors in current day aircraft research.

1.6 Thesis outline

By now the idea of the research is properly introduced, after which its conceptual and technical sides are discussed. This is done throughout this report which is structured as follows.

The main topics, being performance models and BlueSky, are extensively discussed in Chapter 2. This includes the idea behind performance models, and thus how BlueSky calculates the performance it simulates. This is followed by the methodology. This chapter, Chapter 3, contains a look into the command structure and thus how the new logic should follow this structure. This then also explains how the aircraft data is used and formatted to end up with the though out structure. The concept for validation is also discussed to give an idea of what has been explored to prove the accuracy of the model. This chapter is followed by the results of the experiment, seen in Chapter 4. These show the new performance of BlueSky. A further insight in these results and the entire research is given in Chapter 5, as the research required an extensive look at how it was carried out. The conclusion is provided in Chapter 6, also providing ideas for further research.

2

Background

This chapter focuses on information that was required to start the research. This background information posed the first step into understanding what the different main topics of the scope were, of which performance models and the ATC simulator BlueSky were most important.

2.1 Performance models & simulators

Understanding the way a performance model works provides the knowledge how these can be used. Performance models do not offer the exact solution to all unknown parameters, but instead provide a sufficient level of detail regarding large scale simulating. These large scale simulations have been a major way in which theoretical traffic concepts can be validated. This opens op a large variety of parameters to change: one can differ in airports, aircraft, waypoints and further navigational data. Most of the navigational data is fixed and compiled in data sets. However, not all data is as straight forward as that. Aircraft performance data is complex and hard to get by, aircraft manufacturers hardly give out any specific data on aircraft. In order to attain plausible results for such a problem, performance models are created to provide sufficient accurate values for ATC problems. These models calculate speeds, altitudes, rates of climb and other essential variables to execute basic simulator behaviour. This is done by combining collected empirical data with empirical equations. Making the fuel flow a function of the thrust using set constants, is an example of which. A performance model can be seen as the combination between the set of equations that are used for the simulation, and what these compute using the fed data from data sheets. Thus, results are generally speaking: the basic aerodynamics, engine performance and equations of motion.

This chapter elaborates on how a performance model within ATC research is set up. These models are the main research objects of the conducted research and therefore require to be fully understood. This ranges from the set of equations that are used, to the actual currently implemented models and how they behave. As obvious by now, ATC research requires simplifications. Section 2.1.1 treats this first step, by evaluating what the issue is with a high degree of freedom.

2.1.1 Complications with high DoF

Several complications arise when trying to achieve high accuracy with a model. Let alone doing this for multiple aircraft. These complications are listed below.

- **Complicated in terms of computational time**
  Although not always a deal breaker, but obvious, is the fact that computational time increases with complexity. This decides whether a real time simulation is possible or not.

  Nowadays, most computers are able to simulate and compute a 6 DoF model of an aircraft. Therefore, having an ATC simulator with 6 DoF models would not be a problem in

itself, rather the multiplication of this concept. Simulators for ATC should be able to handle large number of aircraft, making the simulation extremely computational intensive in the case of 6 DoF. This emits the option of real time simulating, whilst this option is essential to BlueSky.

- **Complicated in terms of data**
  The more parameters one would like to model, the less computational power there is to include a large variety of aircraft. Aircraft data is scarcely available, and if there is, it is not sufficient to set up 6 DoF models. All types with insufficient data would have to be reduced to expansions or interpolations of the present 6 DoF models, making the simulator unnecessary complex.

  Less parameters means more aircraft in the simulator.

- **Unsteady simulation**
  Lastly, but most important, is a numerical reason. A 6 DoF model, or any model using non-linearities, makes use of translational and rotational axes. These introduce nonlinear differential equations, requiring higher order numerical integration. The latter computation is unstable when using large time steps, where Δt in BlueSky by default is 0.05 seconds. Therefore, to maintain a time sufficient simulation, non-linearities requiring higher order integration are avoided.

In addition to the reasons described above preventing the use of a 6 DoF model in ATC, one also should ask themselves to what level of detail is actually required. Of course, reducing to the highest order possible is an option. However, if the simulation does not require such accuracy, one can justly permit to make further assumptions and simplify. In ATC research, a point mass model is sufficient. How that is done can be read in Section 2.1.2.

2.1.2 From 6 DoF to point mass

Representations of a model can range anywhere from a DNS simulation to a simple point on which forces act. Figuring out what the required level of detail of the simulation is, governs the degree to which is modelled. Here is where the idea of reduced order models (ROMs) come into play. When considering ATC research, the primary goal is to optimise trajectories in many possible ways, whilst keeping the capacity of the network on a high level. Simultaneously, the safety of all aircraft should be guaranteed [4]. Different degrees are possible, discussed in subsequent subsections.
As it is clear that creating a 6 DoF model requires a lot of effort, the point mass is introduced. The point mass tends to keep the model as simple as possible, whilst conserving the essential characteristics for large scale simulations, like ATC research. The aircraft shape is reduced to a point, therefore assuming advanced aircraft behaviour to be negligible. This can be illustrated with a high DoF, which creates a rigid body using equation of motion both in the translational axis in the rotational axes. These axes are coupled, which can be observed in e.g. a 'Dutch Roll'. If a point mass were to rotate in any axis, its behaviour would be equal for all as it is a symmetric body.

Stepping down from a 6 DoF would be a 3 DoF (like a point mass), in which all all rotational motion is omitted. Using a 3 DoF model results in an object which can either move in lateral, longitudinal or vertical direction. To attain such a 3 DoF model, the following assumptions listed hereafter are required. Also, note that next to these assumptions, further assumptions like *flat earth*, *atmosphere at rest* and the use of *conventional aircraft*, are in place [5].

- **Pitch dynamics are instantaneous**
  Pitch is assumed to be instant, therefore omitting the delay in stick movement and aircraft response. The angle of attack is instantaneous and pitching moments are reduced to zero, as these moments are in equilibrium.

- **Symmetric flight**
  Symmetric flight is defined as a flight in which the side-slip is never nonzero. There is no side velocity, no side force or any yawing moment.
- **Acting forces**
  The sole acting forces on the aircraft are thrust, lift, drag and weight, acting on the centre of gravity.

Note that these assumptions also cause the dynamic modes, where translational and rotational motion is coupled, are omitted. This leads to the following set of equations of motion, seen in Equation (2.1), Equation (2.2) and Equation (2.3). These equations are for illustrative purposes and are not further elaborated.

$$\text{Airspeed} \qquad \dot{V}_a = \frac{T - D}{m} - gS_{y_a} \tag{2.1}$$

$$\text{Flight path angle} \qquad \dot{\gamma}_a = \frac{LC_\phi - mgC_{y_a}}{mV_a} \tag{2.2}$$

$$\text{Heading} \qquad \dot{\Phi} = \frac{LS_\phi}{mV_aC_{y_a}} \tag{2.3}$$

Using a 3 DoF is very common, but can be easlily expanded to 4 DoF model. An example can be seen in [6], where the roll rate is actually included. This result in a 4 DoF model, which can be useful when banking performance is also important. Such a model would include the equation for rolling, as seen in Equation (2.4).

$$\text{Roll rate} \qquad \dot{p} = L_p p + L_{\delta_a}\delta_a \tag{2.4}$$

When referring to the point mass model, we are referring to the 3 DoF model in this paper. Meaning, all other aerodynamic forces are neglected, apart from lift and drag. Naturally, weight and thrust are taken into account, creating a model with four forces. These forces, in combination with axes, define angles such as the flight path angle. A sketch of this model can be seen in Figure 2.1.

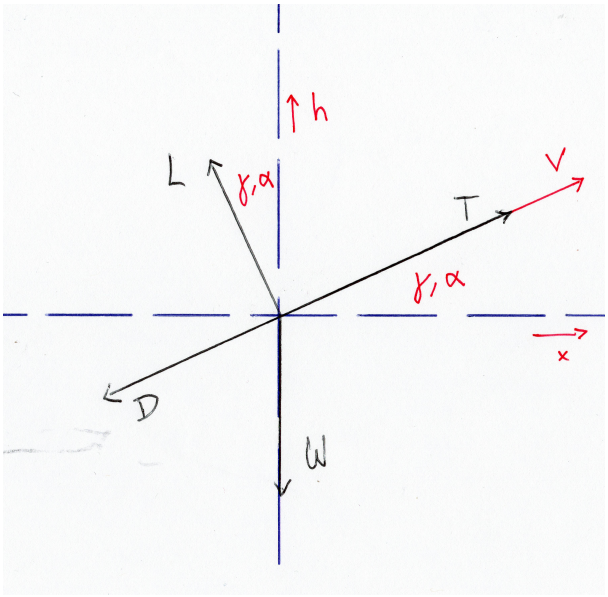

Figure 2.1: Sketch of acting forces and angles of interest

Figure 2.1 shows a visualisation of the aforementioned assumption 'Acting forces'. Thrust and velocity are always aligned, so that the flight path angle (γ) is always equal to the angle of

attack (α). When treated as two different angles, accelerations cannot be assumed to be in line with the forces which results in a relatively too complex situation. Therefore, the drag will be in line with the trust, and life and weight in relation using the flight path angle. How these forces are modelled can be found in Section 2.1.3.

2.1.3 Deducing forces

A second major obstacle in modelling aircraft is to evaluate what 'simplified' performance would be. The main challenge is to model the four governing forces that are acting on the aircraft. As most performance is unavailable, the forces are determined through different methods. General information like engines, wing area or span are easily obtainable. Whereas data regarding exact lift and drag coefficients is heavily restricted by plane manufacturers. What follows is a brief description of how lift, thrust, weight, drag and fuel flow are generally determined. Note that fuel flow is not a force, however just as relevant and thus also listed.

- **Lift**
  A rough approximation of the lift generated at different phases of the flight can be attained by using basic characteristics of the model of an airplane (e.g. wing span). Results are surprisingly accurate by using the general lift formula, when accounting for the effect of the fuselage as well [7].

- **Thrust**
  The calculation of thrust is done with formulae and set coefficients, which are then used to construct polynomials. Manufacturers do give out maximum thrust values of engines. Using a thrust setting, one can create an simulating environment with variable thrust by adjusting the setting. Most performance models actually include engine swap features as well, creating aircraft with variable engines to account for different versions of the plane.

  Note that the maximum thrust, for example, decreases with altitude. The maximum is calculated as a function of engine type, pressure altitude, airspeed and temperature following e.g. the BADA performance model [3].

- **Drag**
  Essential in aircraft modelling is the determination of both lift and drag. Drag is somewhat more complicated than lift, as other parts than the wing play a significant role as well in determining drag. The wing is mainly responsible for the lift-induced drag, where the aircraft as a whole induces parasite drag. To model both lift and drag realistically, computational intensive flow models are required. These slow down simulations and cause an almost Pareto-principle situation. Therefore, in ATM modelling, where point-mass modelling is used, exact determination of lift and drag is not possible. Moreover, many of the important angles, like angle of attack, are omitted, thus making it impossible to make exact calculations. Instead, the drag polar polar of aircraft is used, built up from stochastic models [8]. Much of the performance is derived from these polars, therefore playing a major role in estimating performance.

- **Weight**
  Weight is paradoxically one of the less difficult but extremely difficult parameters to determine. Manufacturers do actually give out weights like its maximum take-off weight or operating empty weight. The challenge lies in determining what the weight of a specific aircraft has been. Performance models use relatively simple methods to provide an average weight for an aircraft type. It is often up to the user to define a weight according to their input.

- **Fuel Flow**
  The fuel flow is also one of the challenging parameters. This is one of the parameters engine manufacturers are not all too generous to provide, therefore it is constructed using constants or polynomials. Generally, it is split up in cruise, descent, ascent, and perhaps phases in idle mode, though this may vary between the different models [9]. The essential

idea among the models remains the same, which is the splitting up of the different flight phases. Generally, it does not require many other variables, but overall, either thrust or speed are required to calculate the fuel flow [2].

Aircraft optimisation is often concerned with maximising the lift over drag. A result of which is the climb and descent performance, and fuel flow. After all, maximising lift over drag causes the aircraft to experience as little drag as possible in flight, reducing fuel consumption.

Knowing how performance models generally work, a closer look is taken at the models present in BlueSky, discussed in Section 2.1.4.

2.1.4 Current models

Performance models in BlueSky build upon the idea of the point mass model present in BlueSky. That is, each of the performance models contain a way of calculating the required performance parameters using their own defined set of data. These are all approximations of real aircraft, though each performance model does their calculations their own way, and deduce the required data in a different way. The current performance models (within the scope) present in BlueSky are named BADA (made by Eurocontrol) and OpenAP (created by Junzi Sun).

Performance models can be distinguished in a number of ways. The primary difference in performance models is whether it is a kinematic model, physics model or combination of the two.
Models using physics (in literature also known as kinetics) use forces to come to the state of the aircraft. This can be seen as a combination of mass, aerodynamic forces and angles to establish a free body diagram. Doing so provides the model with resulting forces. Subsequently, these forces are used to calculate accelerations and determine the speed. The result is the position of the aircraft combined with its performance parameters like fuel flow.
The kinematic model focuses on aircraft motions, therefore evaluating speed, altitude, vertical rate, distances and so forth. With kinematics, the aircraft is evaluated from a performance point of view. Parameters like speeds are controlled, by using standard or determined values for acceleration. Determining these values can, for example, be achieved by calculating the change in energy of an aircraft. As last step, the kinematic model manages to estimate the forces acting on the aircraft with the determined speeds [10].
Summarising, in a physics model the displacement is derived from forces, in a kinematic model the forces are derived from the displacement. Kinematic models will always have to estimate, therefore being less accurate than physics models.

With this division in mind, the two performance models present in BlueSky are subsequently discussed to understand the workings behind a model.

2.1.5 BADA

Widely used, and around for already a very long time, is the Base of Aircraft DAta (BADA). It has become the standard in ATC research, though recently BADA has been developing their fourth version of the model. Since then, as this version relies heavily on proprietary data, BADA 3.0 differs significantly from BADA 4.0. Currently, BlueSky uses the third version for BADA, which is the one discussed. Reason being, BADA 4.0 is proprietary and therefore not an option if BlueSky is to be kept as available as it is.

BADA is a typical example of a kinematic performance model to determine forces [11]. This can be seen in Figure 2.2, in which Aircraft Performance model is the part calculating forces, and Airline Procedure Model is the kinematic data part of the model [12].

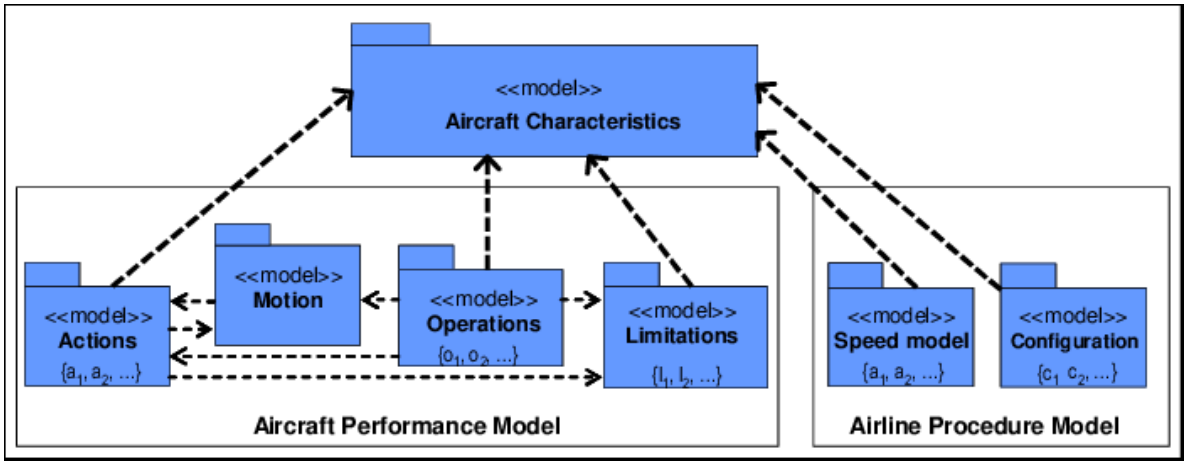

Figure 2.2: Structure of the BADA performance model

Figure 2.2 shows how a typical model can work. The procedure model contains kinematic data for normal operation of the aircraft. On the other hand, 'Actions' in the performance model contains the forces acting on the aircraft, derived from the ARPM. These are processed in the 'Motion' part through equations. 'Operations' covers any other operating modes not part of the aforementioned two. These three are all controlled by the 'Limitations' module, preventing the aircraft to show any unrealistic behaviour.
The ARPM contains speeds for different configurations. These are split up in climb, cruise and descent. Both the APM and ARPM use the 'Aircraft Characteristics'. This last block contains coefficients for each specific aircraft so both APM and ARPM can calculate their respective values [9].

2.1.6 OpenAP

Although BADA is extensive and covers many grounds to simulate realistic aircraft behaviour, its recent developments caused BADA to become more and more filled with proprietary data (see BADA version 4.0). One of the cornerstones of BlueSky, as mentioned in Section 2.2.1, is to maintain an open data structure. Hence to fill the gap a new performance model was created to keep BlueSky free from restrictions: OpenAP. There are a lot of similarities with BADA, as it is not intended to be a replacement for BADA. The goal is to provide an open-source model that can be used freely [2].
OpenAP follows a different structure than BADA, of which the structure of the former can be seen in Figure 2.3. OpenAP is a combination of a kinematic and dynamic model. In literature, dynamics is seen as physics.

Within 'Properties', OpenAP contains all requires parameters for the engines of the aircraft. These include all different options possible, but also their characteristic performance. This includes the bypass ratio, fuel flow coefficients and so forth.
The 'Kinematic performance' basically contains a database, built up on the results of the statistical model 'WRAP'. During each segment of the flight (e.g. takeoff, climb or final approach), a set of performance parameters are calculated. WRAP contains the functions to calculate the optimal, minimum and maximum of the values [10].

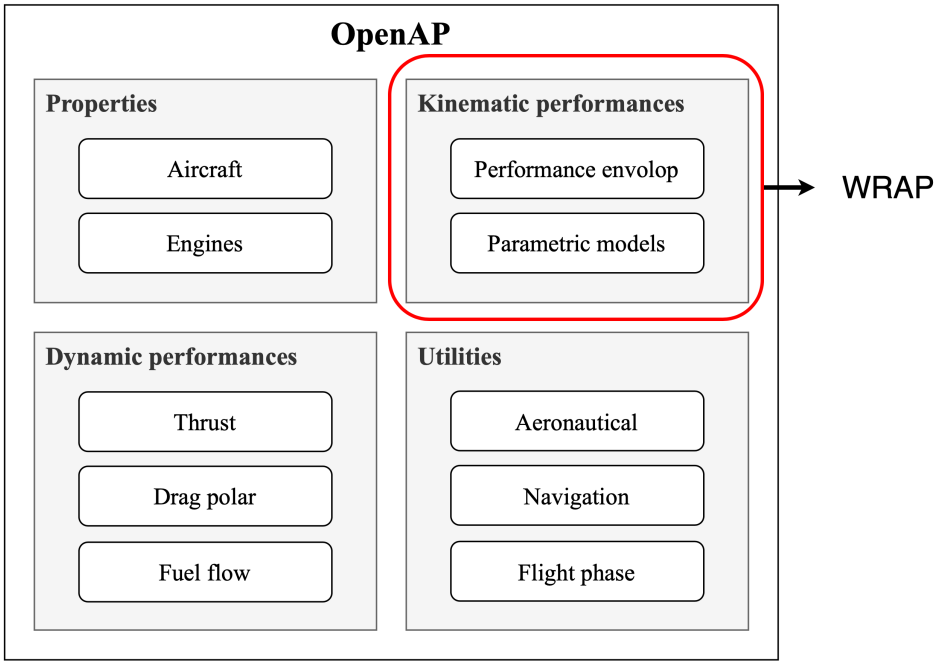

Figure 2.3: OpenAP structure with WRAP indicated

Note that OpenAP is actually a 4DoF model, as elaborated on in Section 2.1.2, therefore containing a roll rate equation. Furthermore, this module of OpenAP calculates the thrust, drag and fuel flow. Therefore, it can be said that this part deals with the mass and forces, where the 'Kinematic performance' deals with the distances, speeds and other performance parameters. Lastly, the 'Utilities' module contains all other necessities for calculation related to performance. Primary example of such necessity is the determination of the flight phase. By comparing altitude, speed and other parameters, the module determines which phase of the flight the aircraft is in.

Conclusions from the paper denote that the thrust model actually performs better than the BADA model, at low altitudes that is. Increasing the altitude decreases the difference between the models, though it can be said that OpenAP performs better overall for thrust. The other parameters are comparable, which is in itself praiseworthy, as OpenAP achieves this result purely with open data, contrary to BADA.

Knowing the idea behind BADA and OpenAP, Section 2.2 elaborates how BlueSky is set up and how these performance models are used within.

2.2 BlueSky ATC simulator

Performance models are rather limited if used without navigational data. After all, the focus of the paper is on modelling within air transportation research. Although simulators rely heavily on well performing performance models, it is composed of more aspects to create a fully functional simulator for ATC.
As mentioned in Chapter 1, the ATC simulator in use is BlueSky. Though still very new, its concept is not. BlueSky inherits much of its functionality from the traffic manager TMX.[1] Readers familiar with TMX will find similarities between the two simulators.

[1] https://cs.lr.tudelft.nl/atm/software/bluesky/ [Cited: 18-05-2021]

2.2.1 Background

Within ATC, one of the main challenges is the dynamic environment in combination with numerous aircraft present in that environment. There are numerous factors to take into account, such as other aircraft, atmosphere, wind and more. All of these factors influence each other in their own way. Consider for example a continuous descent approach (CDA). During a CDA the aircraft is on approach at a constant flight path angle. During which, the aircraft maintains said angle to prevent unnecessary fuel burn and noise as during a regular approach. In such a scenario, one could of course propose the correct aircraft settings and atmosphere properties, but it is far from realistic. Moreover, collision danger can be issue during a CDA, which is essential in the simulation to account for.
Exactly that is what the current issue is with ATC research, ideas are often well researched, but quite specific for one application. It prevents the reader from drawing a proper conclusion, as propositions are not compared, rather excluded. The result is a forced proposal which won't necessarily work for all readers [13].

This is where BlueSky has made the difference. The user should be able to compare different kinds of scenarios, and should be able to compare the proposed solutions to draw a conclusion for themselves. Previous research has always proposed their solution as optimal, creating a situation where every researcher claims that their idea is better than the other. BlueSky aims to simulate real-time, up to thousands of aircraft. These can be given a flightplan defined by the user, and the program is able to produce real-time results presented in a graphical way. Researchers are able to evaluate at every point in time what their flightplan does to the aircraft.

Secondly, next to the fact that there are several other ATC research projects available (albeit sufficiently modelled or not), there is another key aspect to consider with BlueSky which sets it apart from the others. In order for the program to be future-proof, BlueSky has a total of five cornerstones on which it is built. Adhering to these five makes BlueSky accessible for all, both in terms of difficulty and time. These cornerstones are listed below [1].

- **Open Data**
  The used data within BlueSky should be without issues regarding proprietary data. All data used should be accessible for all, without any restrictions.
- **Open Source**
  One of the essential ideas is that BlueSky should be tailor-made for every case. By releasing the software as open source, the user is able to implement whatever he or she wishes.
- **Free**
  By avoiding large development teams, BlueSky can be offered free of charge. This makes it free of charge for the (academic) user.
- **Multi-platform**
  Using BlueSky across different platforms should be without hassle. The same level of quality and performance should be expected on every platform to make it accessible for every computer.
- **Acceptability**
  The architecture should be simple to understand, and adjustable where required. Users with some background knowledge should be able to comprehend the program within the first hour of using it.

These cornerstones should be adhered to when making changes to BlueSky, as they give BlueSky its unique character. With the general idea of BlueSky in mind, Section 2.2.2 elaborates on the specifics of the code architecture of BlueSky.

2.2.2 Structure

When first starting BlueSky, users see a GUI as shown in Figure 2.4. This is the main window of BlueSky in which everything takes place. The user is able to scroll through the world map, whilst simultaneously observing all the aircraft flying in real-time. The aircraft are characterised by an unique identifier, together with their respective speed and altitude.

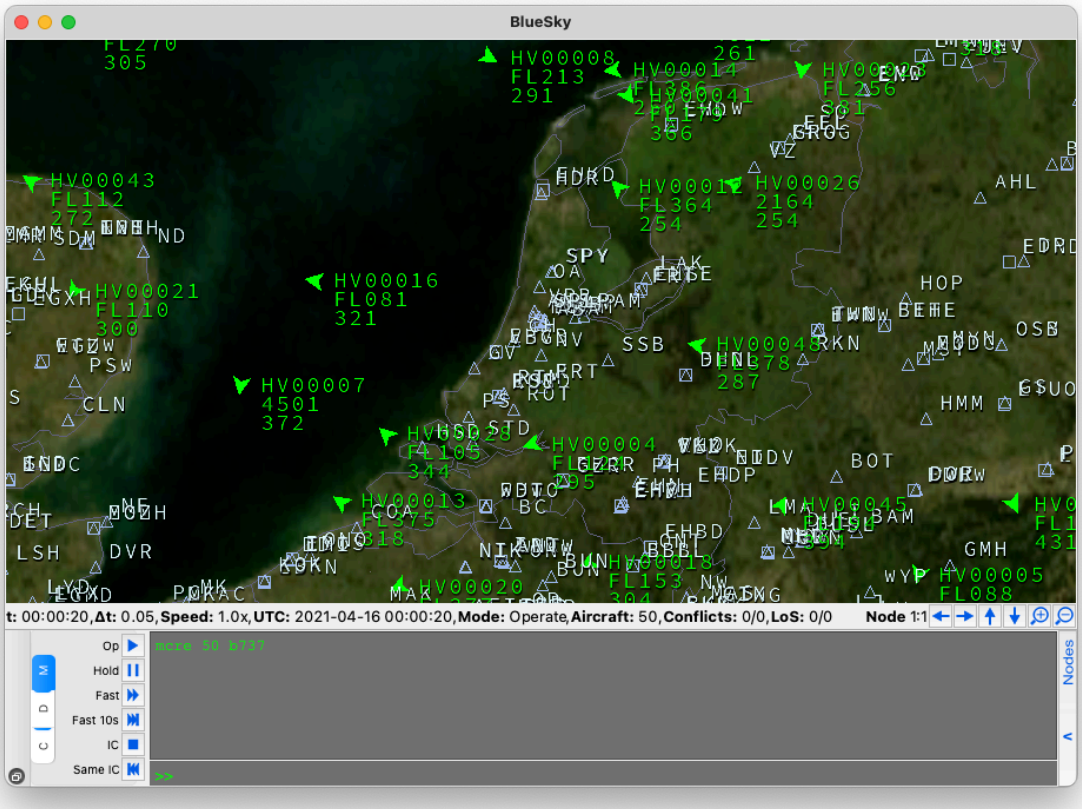

Figure 2.4: The GUI of BlueSky

The GUI also shows the distinction between two important sets of data present in BlueSky, namely the navigation data and the performance data. The former contains all the information on airports, waypoints and other geographical information. The performance data contains all information on aircraft logic, autopilot settings and other relevant criteria for the aircraft to fly its specific route and heading.

This GUI is the result of several submodules working together. Caluclating aircraft performance is not the only module, as the simulation itself with navigational data requires a basis as well. A simple schematic overview can be seen in Figure 2.5.

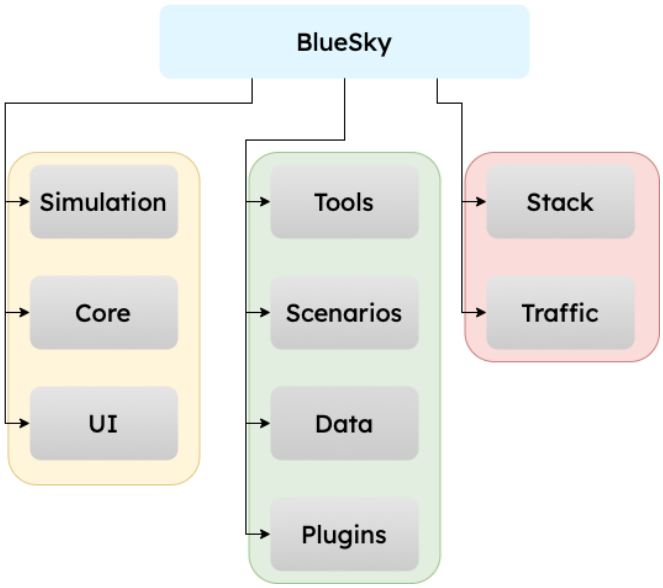

Figure 2.5: General segments which make up BlueSky

Figure 2.5 shows a very rudimentary overview of BlueSky its file structure. On the left, shown in the yellow block, are the Core, Simulation and UI. These contain the coding for the handling of all files, setting up simulation (and containing navigational database and network) and the GUI, respectively. This block is the backbone of BlueSky, but will not be touched during the entire course of the research.

In the green block are the supporting modules that will be used. Module 'Tools' contains all the necessary conversions and standard values needed in aircraft simulation. For example, the use of the ISA standard atmosphere, or the conversion between TAS, CAS and IAS. Secondly, the Plugin module contains the standard idea of plugins. These can be activated or deactivated, which depends on the choice of the user. They are not essential to BlueSky itself.
Thirdly, all necessary data files are in the Data module. This module is constantly called, as it contains all specifics constantly required in the simulation.
Fourthly, Scenarios is a collection of user written text documents. These can be read and run by BlueSky to follow a user defined trajectory.

Lastly, the red block contains the two most important modules for this research. The Stack module contains and processes all commands given by the user. It recognises, and allocates the corresponding modules to perform its task. Many of these modules can be found in the Traffic module. The traffic module contains all kinetic and kinematic performance that is associated with aircraft in general. This ranges from location of the aircraft, the temperature it is currently flying in, or its airspeed. Traffic creates and updates objects, as well as creating and updating the submodules of the Traffic module. Examples of these submodules are the autopilot, conflict resolution or windfield generation.

2.2.3 Coding architecture

One of the important aspects of BlueSky is that it is able to simulate a real-time environment, up to 1000s of aircraft simultaneously. Achieving said goal takes an optimised code structure in which computational time is kept to a minimum. BlueSky manages to do so in by making use of two nifty code enhancements.

First of all, BlueSky makes use of Object-Oriented Programming. Using OOP has multiple advantages (like the ease of troubleshooting) of which inheritance is especially important for

BlueSky. Inheritance creates the possibility to make use of class functions and class/instance variables.
One of such examples is found in the performance models. BlueSky contains multiple performance models, which all share certain aircraft parameters. By defining overlapping parameters in a class function, which can subsequently imported in each of the performance models, all performance are built on the same basis and reduce the amount of necessary code. An example of this code structure can be seen in Figure 2.6.

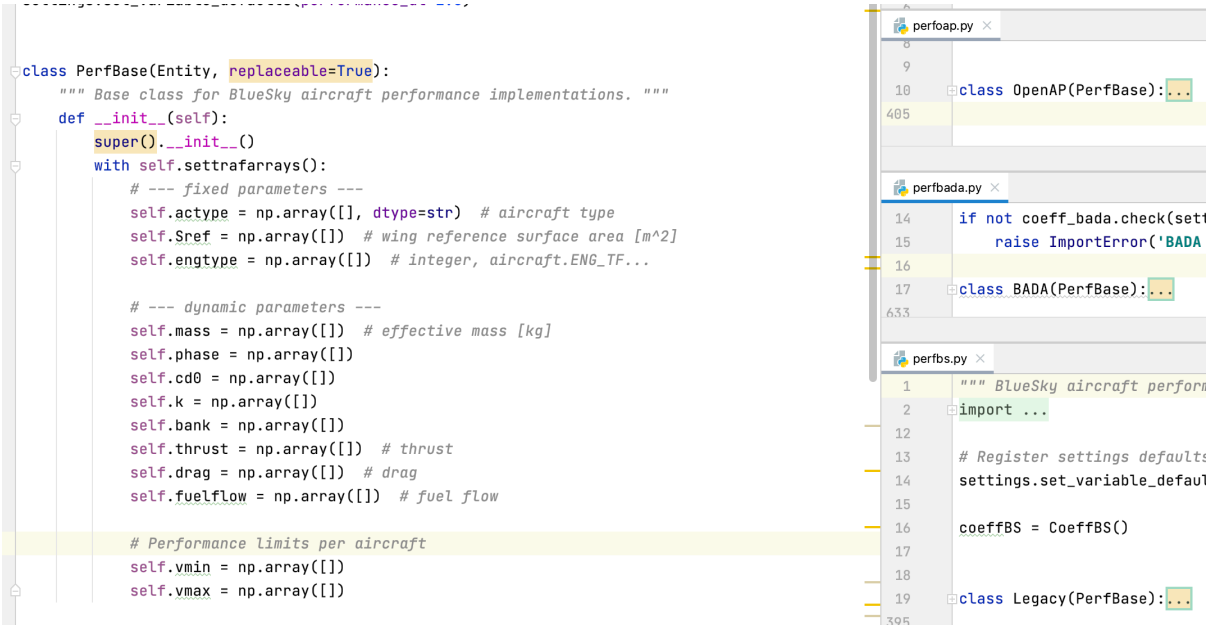

Figure 2.6: Inheritance of the PerfBase class in the three difference performance models, OpenAP, BADA and Legacy

Secondly, data in BlueSky is vectorised. Although the less experienced user may think that BlueSky keeps track of the basic aircraft data, looking under the hood shows a whole different perspective.
During a simulation, over 100 parameters are tracked of each aircraft. This is a combination of the data found in the Traffic module (the central aircraft operations file), as well as the data present in the Performance module. Both of these record the more obvious parameters as speed, as well as zero-lift coefficient, Oswald factor or drag coefficient of the landing gear. It requires no further explanation that the amount of data during simulation is vast, and optimising said data certainly offers advantages.
Therefore, all instance variables are listed in lists or Numpy arrays, also seen in Figure 2.6 under the *__init__* of PerfBase. Doing so enables the option to do array computations, instead of calculating the specific parameters for each aircraft separately. Moreover, the code remains the same regardless of the amount of aircraft present in the simulation. The sole difference is the amount of entries *in* the arrays.[2]

2.2.4 Autopilot

Autopilots in current aircraft are complex machines, often consisting of multiple computers for redundancy. Needless to say, BlueSky uses a 'light' version of the autopilot as we know it. Though less complex, the autopilot still should, and does, match the functionalities of a regular autopilot.

The autopilot modes that will be discussed focus on their working within BlueSky, but will be

[2] https://github.com/TUDelft-CNS-ATM/bluesky/wiki/vectorizing [Cited:28-04-2021]

compared to reality as well. The former is required to understand BlueSky, the latter to gain insight in the coupling of new functions which are to be implemented.

Autothrottle

Currently present in BlueSky is the autothrottle. Using said function gives the autopilot the possibility to control the thrust of the aircraft. Using actuators, the levers are constantly adjusted to compensate for any de- or acceleration imposed on the aircraft. This causes the aircraft to fly at a constant speed or altitude depending on the autopilot settings. Using autothrottle the aircraft will try to maintain its set speed, no matter the orientation of the aircraft.

BlueSky currently works in an autothrottle 'always on' mode. Whilst it is only possible to set a certain speed or altitude, in reality this would be taken care of by the autothrottle if considering a simple longitudinal acceleration. This can become more complicated, in the case of speed on elevator, or speed on throttle. More on this can be found in Section 2.2.4.

LNAV

Nowadays, almost every flight starts with an entry into the Flight Management System (FMS). The FMS contains all navigational data that a pilot requires, and hence can enter the airport at the destination, and waypoints of importance on the journey in between.
Whenever the aircraft is manually flown and brought into the air, pilots often switch to the LNAV mode. In said mode, the autopilot controls the lateral position of the aircraft, making sure the heading is correct. Simplifying, it can be said that LNAV makes the aircraft follow a line that is projected on the ground. LNAV does not tell the aircraft what altitude to fly, or at what speed. For this reason, the pilot is required to manually control these other two factors.

When using LNAV mode in the 'speed' mode of BlueSky, LNAV will be able to perform the required turns. After all, the speed command will make sure that the aircraft flies at the fixed speed thus preventing the aircraft from losing performance in turns. However, when switching to a throttle command, the aircraft will have a varying speed. Turning radius is dependent on speed, which in its place is dependent on the combination of throttle and atmospheric conditions. This can cause the aircraft to make turns earlier of later to make sure the correct heading is conserved.

VNAV

The Vertical Navigation (VNAV) is the more complex system of the autopilot. VNAV actually makes sure that the aircraft flies the desired vertical flight profile (that is, its altitude profile against distance flown), whereas LNAV is solely concerned with the correct heading. The combination of LNAV and VNAV is how most aircraft are flown nowadays. The combination controls a three dimensional profile of the flight envelope, thus virtually controlling every aspect of the flight. With VNAV, waypoints can be given an altitude and speed, guiding the aircraft through the air. VNAV will take care of all procedures to pass these points in correct order.

Within VNAV the distinction between VNAV SPD and VNAV PTH is made. PTH follows the precalculated path with altitude constraints, using elevator for path and throttle for speed. However, it is not always possible to follow this path. Whenever the path is incorrectly entered, or the aircraft deviates for whatever reason, VNAV SPD kicks in. This can also happen if a MCP command is given, which overrules the VNAV function. When in VNAV SPD, the throttle is set and a speed must be flown. This can only achieved by using the elevator for the speed, using the potential energy from the altitude [14, 15].

It is required to introduce two definitions here, called Top of Climb (ToC) and Top of Descent (ToD). Whenever a pilot enters his route, the computer determines what the altitude is at what point the aircraft will either start the descent, or finish the climb. This is calculated once at the

start of the flight. These points are called the ToD and ToC, respectively. For BlueSky, these points are constantly calculated. Reason being, the user is assumed to be constantly changing parameters. Therefore, determining a single path is not feasible. The result is that BlueSky its VNAV function always uses the PTH mode.

2.3 Combining BlueSky and performance models

To connect both the section on performance models and on BlueSky, this last section shows how these are intertwined. Flow diagrams were created in order to understand how the different modules present in BlueSky worked together. Note that each performance works differently, and thus the performance logic can differ. This does not however influence the following diagrams as they are to give a general idea of the data flows.

Aircraft in BlueSky are both created as updated. Their creation happens once, the update cycle is ran frequently (standard every 0.05 seconds). The creation cycle can be seen in Figure 2.7.

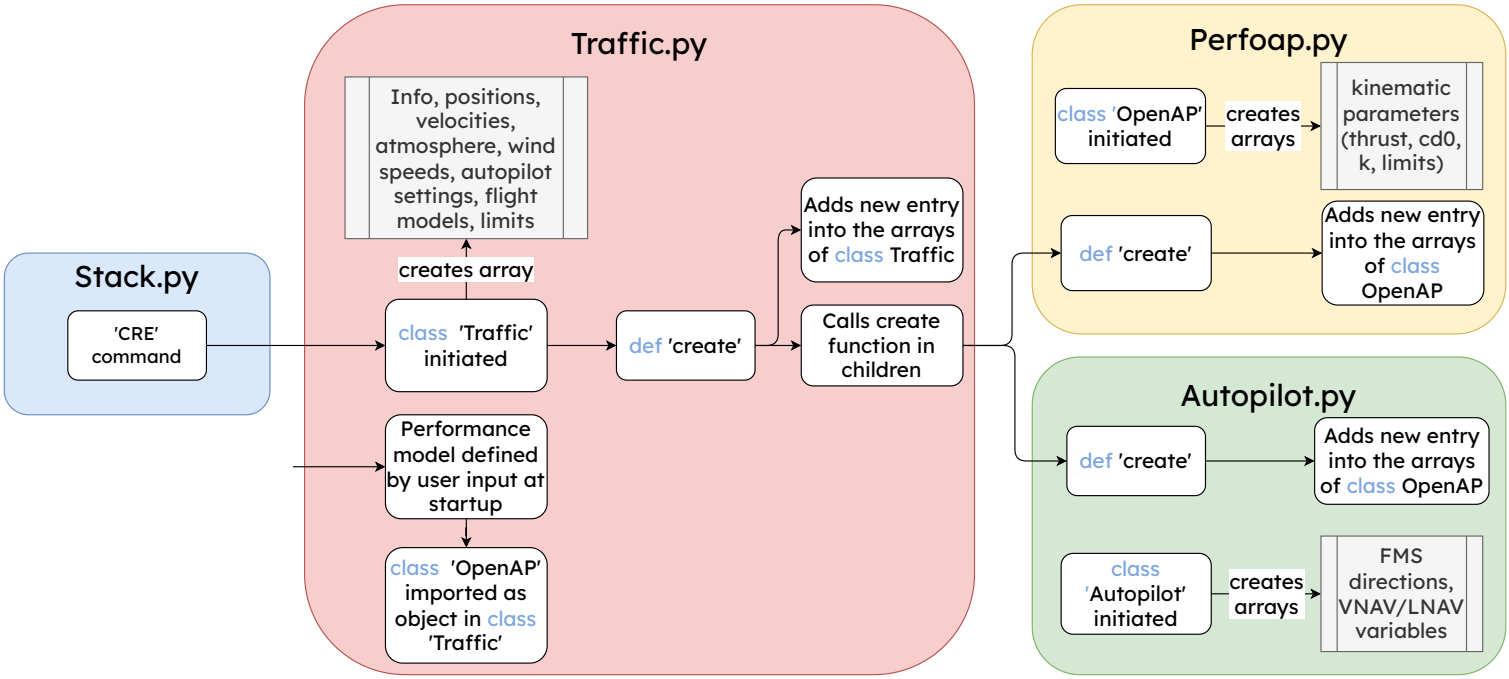

Figure 2.7: Creation cycle within BlueSky

What can be seen in the preceding image are the 'Stack', 'Traffic', 'Perfoap' and 'Autopilot' modules. These four form the separate building blocks of creating an aircraft. This starts in the Stack module, where a large amount of different commands can be issued. In this case, the 'cre' command is used, starting the cycle. Subsequently, in the traffic module, the class 'Traffic' creates a large amount of arrays all containing different parameters, e.g. velocity, altitude but also the flight models. Meaning, which file should be used for the autopilot, or which file for the performance model (as the user has choice in which performance model to use). This can be seen in the figure together with the initiation of the Traffic class, denoted by 'defined by user input'.

The next step in the process is the 'create' function in the Traffic module. This adds a new entry to the array (or multiple entries, depending on the amount of created aircraft). Each entry in this array corresponds to an aircraft and its value for that particular parameter. Simultaneously, def 'create' calls the children (autopilot, performance model etc.) classes. This feature of OOP enables that multiple classes are called and are run in parallel. This means that both the

create cycle of the Autopilot, as well the one of OpenAP, have started. Both contain their own create function, creating arrays for their respective parameters of interest. The newly created aircraft is then added as an entry to each of these arrays. Again, the first time the class is initiated, the arrays are created, as seen in Figure 2.7 by 'creates arrays'.

After the creation of an aircraft, it can perform all sorts of manoeuvres. If the user changes the vertical speed, the altitude must change due to this command. Changing the speed will respectively have an impact on distance to a waypoint. This ask for frequent updating of the aircraft its state, done in the update cycle. This flow diagram can be seen in Figure 2.8

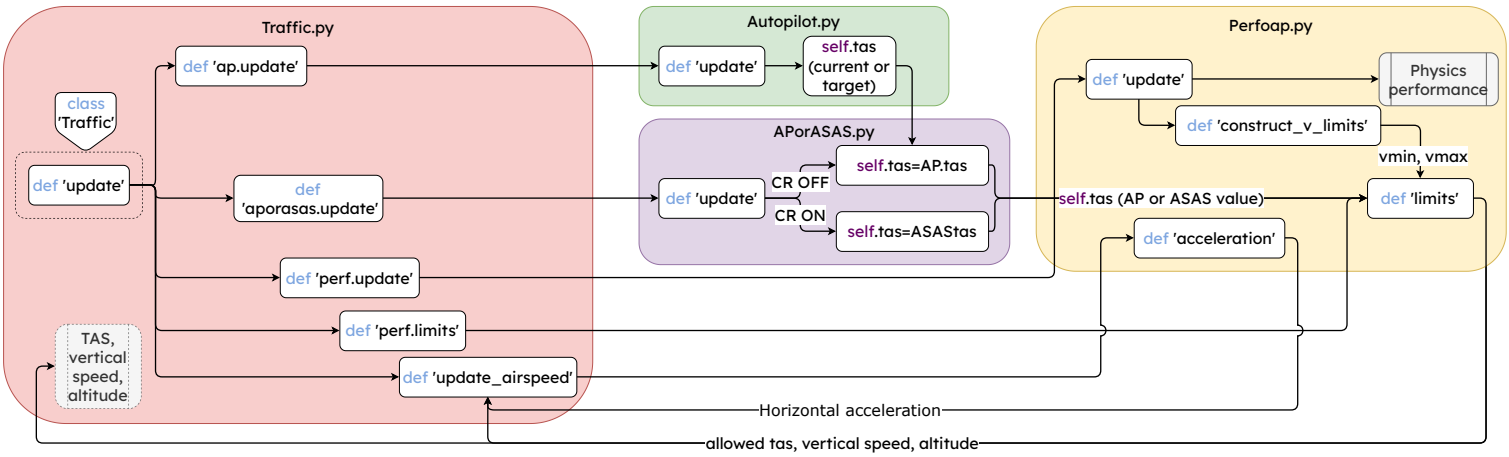

Figure 2.8: Update cycle within BlueSky

The figure will be explained by going over the five update functions seen in the cascade of the Traffic.py module. Note that the cascade structure in the red module indicates the order of execution. The general update function is part of the Traffic class, located in -surprisingly- the Traffic module. The cycle starts with the update of the autopilot function.

Firstly, the general update function (Traffic.py) initiates the update function of the Autopilot module (Autopilot.py), which outputs a speed value (self.tas). Depending on whether this value differs from the current speed determines whether an acceleration is required. This speed value is fed to the APorASAS module (APorASAS.py).
The APorASAS module is initiated after the update of the Autopilot module, which determines whether there is a conflict (intersecting trajectories) ahead or not. Depending on the situation, the self.tas value from the Autopilot is used, or the self.tas from APorASAS. Meaning, the speed from either one is chosen as governing speed.
Subsequently, the def perf.update is called in Traffic.py, arriving at the 'Perfoap' module. Within, several parameters like thrust, fuel flow and drag are calculated. Simultaneously, the def 'construct_v_limits' calculates the limits of several performance parameters, which are fed to def 'limits'. The latter is called after the def perf.update within the Traffic module is called.
As the last step, the calculated limits are used in the last update function def 'update_airspeed'. Using the 'acceleration' in the Perfoap module, calculating the acceleration.

These flow diagrams wrap up the literature required to continue the research. With the knowledge of the main topics, a methodology was set up to conduct the actual research. This is further discussed in Chapter 3.

3

Methodology

This chapter will elaborate on the steps taken in this research to generate results. The actual research and steps were set up after the results of Chapter 2. This chapter will discuss the way the pre- and postprocessing has taken place, what the new logic was and how this was implemented. Lastly, the idea and reasoning behind the validation is also laid out, including the idea behind the sensitivity analysis.

3.1 Set-up

In this section, the way this research has been set up is briefly discussed in helicopter view, after which each of the components are extensively discussed in the subsequent sections.

Firstly, Section 3.2 discusses how BlueSky was used to run simulations to visualise its performance. This gave a qualitative idea of the way BlueSky simulates and thus what qualitative points could be improved. It also gave an insight into the coding of BlueSky, and thus provided an idea of possible areas of interest within the code.

The next step was the conceptual approach of the problem. This meant figuring out what kind of logic would make most sense to the user, and correspond as much to reality as possible. A flow diagram has been set up which formed the main foundation behind the implemented material, using the basic equations of motion and simulators. This can all be found in Section 3.3.

This was continued by the preprocessing of the data. Normally one would work out the logic, implement the idea and test. It was advised to format the data first, to prevent cherry picking as much as possible. The main point was that a thrust and FPA setting would be part of the options. The section itself, Section 3.4, will discuss in detail how the sorting has been carried out, including application of Fuzzy logic to identify flight phases. This pre-processing was done using the editor Spyder with the programming language Python.

As the data had been processed by this point, Section 3.5 discusses how the implementation of Section 3.3 has been done. Which and what parts of BlueSky have been changed, what variables have been introduced and how does it work in practise. These questions are answered in this section. This implementation was executed in PyCharm using the Python programming language.
Followed by Section 3.6. In this section the though behind the validation is explained. Why are the metrics used as they are, and why do they indicate a 'good' result? It will not show any results yet, but merely explain the way the validation has been thought out. The chapter is finishes with describing the sensitivity analysis its concept in Section 3.7.

3.2 Input for BlueSky

BlueSky has been treated in Section 2.2. It treated its principles, but did not provide an inside look at the technical performance side of BlueSky. This section does exactly that to understand how its command structure works, and what its current capabilities of simulating are. This entails its standard workings as well as more advanced features. Firstly, its command structure is discussed in Section 3.2.1 and how that results in performance shown in Section 3.2.2, after which several figures will show in a simplistic way where BlueSky still has issues in Section 3.2.3.

3.2.1 Commands of BlueSky

The command structure will be explained from very simple, to more advanced options. Note that not all commands are discussed, the list of possibilities is already rather infinite.[1] Rather, commands that are of interest and used in this research are explained. First up is the creation of aircraft. Aircraft in BlueSky are created using a create command: 'CRE'. The user defines all of the following: a callsign, aircraft type, latitude, longitude, heading, altitude and speed. Without any further commands, the aircraft will fly at that altitude, at that speed. These create commands can look as follows:

```
00:00:00.00 > CRE 020119-[0] A320 50.2 2.6 44.9 32200 274
00:00:00.00 > CRE 020124-[0] A380 49.9 2.5 60.5 30325 281
00:00:00.00 > CRE 06A063-[0] B737 52.2 7.2 74.2 27900 269
```

Thereafter, the user has the option of, for example, giving a different speed or altitude. This can be done with either the 'SPD' (calibrated airspeed or Mach number) command or the 'ALT' (altitude) command. Both are given in knots (CAS) or fraction (Mach) and feet (altitude), respectively. The SPD command provides no further option, the aircraft is simply accelerated by a rudimentary calculation for acceleration. On the other hand, the altitude command provides some extra functionality. The user can define a vertical speed. Meaning, with what rate of climb or descent does the aircraft proceed towards the new altitude. If no vertical speed is issued, a standard vertical speed of 1500 feet per minute (7.62 meter per second) is used. This is a rule of thumb, which is surprisingly often rather accurate, but creates a standardised climb or descent profile. Thus, it prevents every form of variation as it is a fixed value. Examples of SPD and ALT commands can be seen below, where the altitude command is given a different vertical speed than the default value.

```
00:00:00.00 > SPD 020119-[0] 0.76

00:00:00.00 > ALT 020124-[0] 26375 1200
```

Another set of commands that are of interest are the 'AT' commands: ATALT, ATSPD and ATDIST. Each of these commands are triggers, the attached command is only executed if the aircraft reaches a certain altitude, speed or distance. This has an extensive range of possibilities, as almost every command can be attached to one of these mentioned ones. Though, as it is most relevant for this research, it is used to issue a new speed or altitude command. Say an aircraft 'X' is in descent, but may only decelerate to 200 knots when reaching FL200. A typical command would look as follows:

```
00:00:00.00 > X ATALT 20000, SPD X 200
```

Note that these commands are rather 'simple' in their own manner. They are not executed in chronological order: they are simply saved as a list of triggers. If an aircraft has a certain ATALT command, but reaches this altitude twice in the flight profile (imagine a parabolic altitude profile), it is triggered at the first occurrence. Therefore, to prevent such behaviour, each command has to be stacked with another to delay its activation. This can be done by stacking an ATALT with an ATSPD. Only will the ATSPD be active if the altitude is reached.

[1] For a full list of possibilities, see: https://github.com/TUDelft-CNS-ATM/bluesky/wiki/Command-Reference

The ATSPD command functions similarly to the ATALT command, though the ATDIST command has some extra functionality. This command is executed if a certain distance is flown. This is expressed in the form of nautical miles, and uses a starting point from where is counted. An option is giving a waypoint or latitude and longitude combination, but the position of the aircraft itself is also a possibility. This would look as follows:

```
00:00:00.00 > 44CE64-[0] ATALT 8025.0, 44CE64-[0] ATDIST 44CE64-[0] 3.95 ALT 44CE64-[0] 7025.0
```

This command works as follows. The ATDIST command becomes active only after reaching 8025 feet. At this point, flight 44CE64-[0] receives the ATDIST command, and at that moment uses the position of 44CE64-[0] itself to fly for 3.95 nautical miles. If that distance is reached, only then a new altitude is issued. These kind of command structures are necessary for this research as waypoints are not used. An example of which are level segments, where no speed change occurs.

3.2.2 Current logic

Knowing what the different commands do, the according logic is presented next. This logic was studied to understand how current commands work and function in combination with each other. The representation is rather simple and can be seen in Figure 3.1.

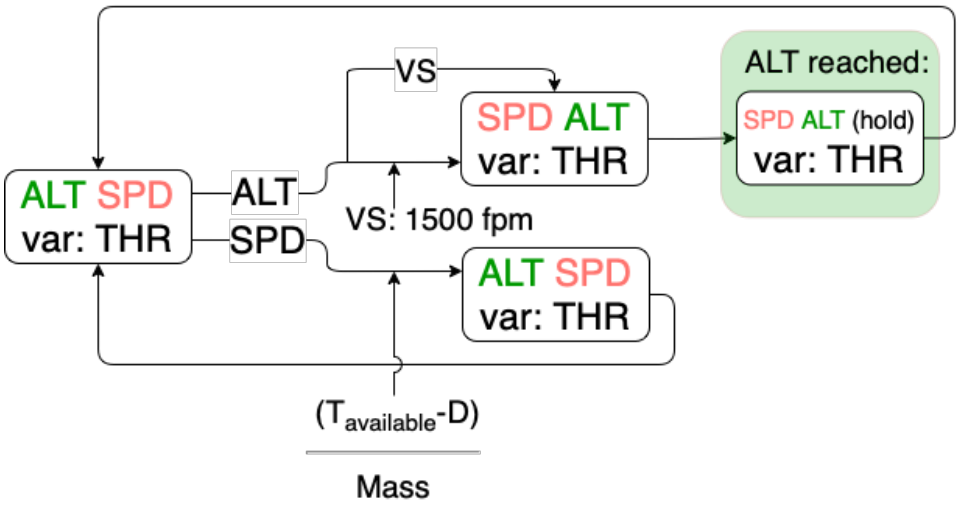

Figure 3.1: Current logic present in BlueSky (using OpenAP)

Aircraft always have an active altitude and speed command, be it going towards or being at. When an altitude is given, the aircraft will change and hold that altitude. The user has the possibility to give a vertical speed, but otherwise the standard value is used. If a SPD is given, acceleration is calculated using the method of the chosen performance model. In the case of BADA this is 0.5 $\frac{m}{s^2}$. In the case of OpenAP, the net thrust is calculated using the maximum available thrust and drag. If the acceleration calculated is smaller than 0.5, it is set to 0.5. Moreover, if an altitude command is given, a fixed rat eof 1500 feet per minute is used. The user is able to define another vertical speed, but this value represents the default.

At all times the thrust is variable (and thus in an automatic mode) to provide the necessary force to achieve every command given. The logic is therefore limited as it provides no option to change the thrust. Changing this would greatly increase the command combinations and thus aircraft behaviour. To demonstrate the current performance, Section 3.2.3 shows the performance in a visual way.

Another point, not included in the diagram but described conceptually is the VNAV logic. BlueSky contains a logic to execute a flight in VNAV mode. Section 2.2.4 already touched upon the concept. In BlueSky, the ToD or ToC are calculated using a set vertical speed per distance flown.

Meaning, it is calculated where the aircraft should start its descent, in order to meet the waypoint criteria. BlueSky uses the rule of thumb of 3000 feet per 10 nautical miles. A better look provides that this is based on the 1500 feet per minute as well (and a speed of 300 knots).[2]

3.2.3 Limitations of BlueSky

This subsection elaborates on BlueSky its performance together with the textual explanations given in Section 3.2.1 and Section 3.2.2. By means of an example situation the current performance is shown, and what its desired performance should be.

Imagine a flight being created at FL250 with a speed of 250 knots. The flight is given the command to descend to FL235. The aircraft will start descending with the usual 1500 feet per minute. Simultaneously, an command is given to decrease the speed to 230 knots when arriving at FL240. Thus, the aircraft will also decelerate, next to descend, when passing FL240, seen in the snippet below.

```
00:00:00.00 > CRE AC1 B737 52 3 FL250 250
00:00:00.00 > ALT AC1 FL235
00:00:00.00 > AC1 ATALT FL240, SPD X 230
```

The result can be seen in Figure 3.2 on the left side. This is BlueSky its current way of flying such trajectory. During this whole flight, all commands are possible, the thrust is simply changed to facilitate every possible command. Note that both the deceleration as the vertical speed is fixed. The slopes of these lines do not change in any case, except when a different vertical speed is explicitly given.

Now imagine that same flight, but using a thrust setting. The aircraft receives the same commands, but now also receives the command to reduce thrust to idle right from the start. The right hand side of Figure 3.2 shows a theoretical, expected performance.
The aircraft will descend, but due to the fact that the engines are in idle mode, speed will now diminish as well. After all, the priority is reaching the new altitude, using the given thrust. However, at FL240 the speed command is given as well. At this point, the aircraft must choose to decelerate to this speed. During this time, the thrust setting remains the same. Therefore, to attain a quicker deceleration, the nose must be tilted up to reduce the speed even quicker. This can be clearly seen in Figure 3.2 after passing the purple, vertical line. Altitude is reducing slower, therefore ensuring a quicker deceleration. The following step is reaching the point of 230 knots, indicated by the yellow line. The plane must be kept at this speed, with the idle thrust setting. The aircraft will continue to slow down due to the thrust setting, but a quicker descent must exchange this potential energy into kinetic energy to prevent deceleration.

[2] Refer to https://en.wikipedia.org/wiki/Rule_of_three_(aeronautics) for this rule of thumb [Cited: 18-02-2022]

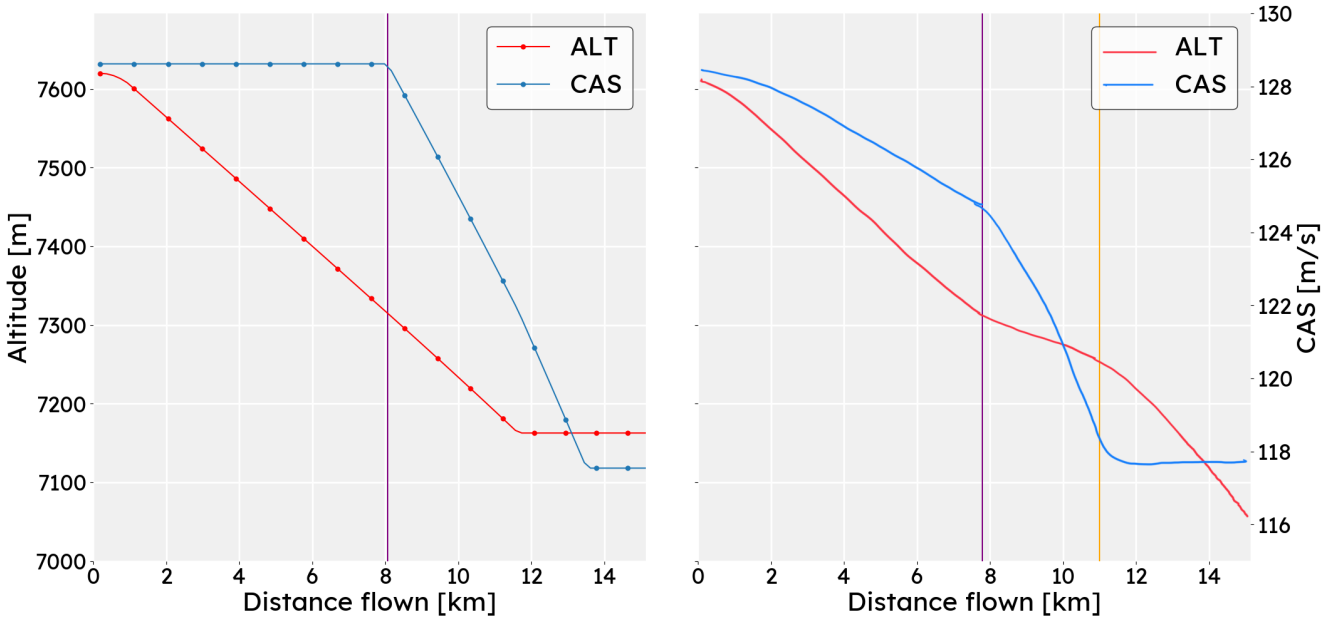

Figure 3.2: Current and proposed BlueSky altitude speed coupling

Figure 3.2 shows in a very simple, yet efficient way how BlueSky currently works presently and in the new situation. Basically, thrust is currently seen as an infinite parameter. The user can command the aircraft to go to every altitude or speed (be it within the limitations) without worrying about potential lack of thrust. This provides very static behaviour, as BlueSky will never show any unexpected behaviour due to the lack of more complex logic.

Secondly, BlueSky does not only miss this essential parameter in the speed-altitude coupling, but also misses the option to set the flight path angle. Using the flight path angle can be an essential tool for pilots, as it provides the relation between vertical speed and speed itself, or the relation between travelled distance and altitude change, seen in equation Equation (3.1).

$$\gamma = \sin^{-1}\left(\frac{VS}{SPD}\right) = \sin^{-1}\left(\frac{\Delta h}{\Delta x}\right) \tag{3.1}$$

Currently, the FPA is not a parameter within BlueSky. However, pilots use the parameter daily. There exists a crucial difference between vertical speed and a flight path angle. Using the figures seen in Figure 3.3 this will be explained. Note that the purple lines again are drawn to represent behaviour to be achieved.

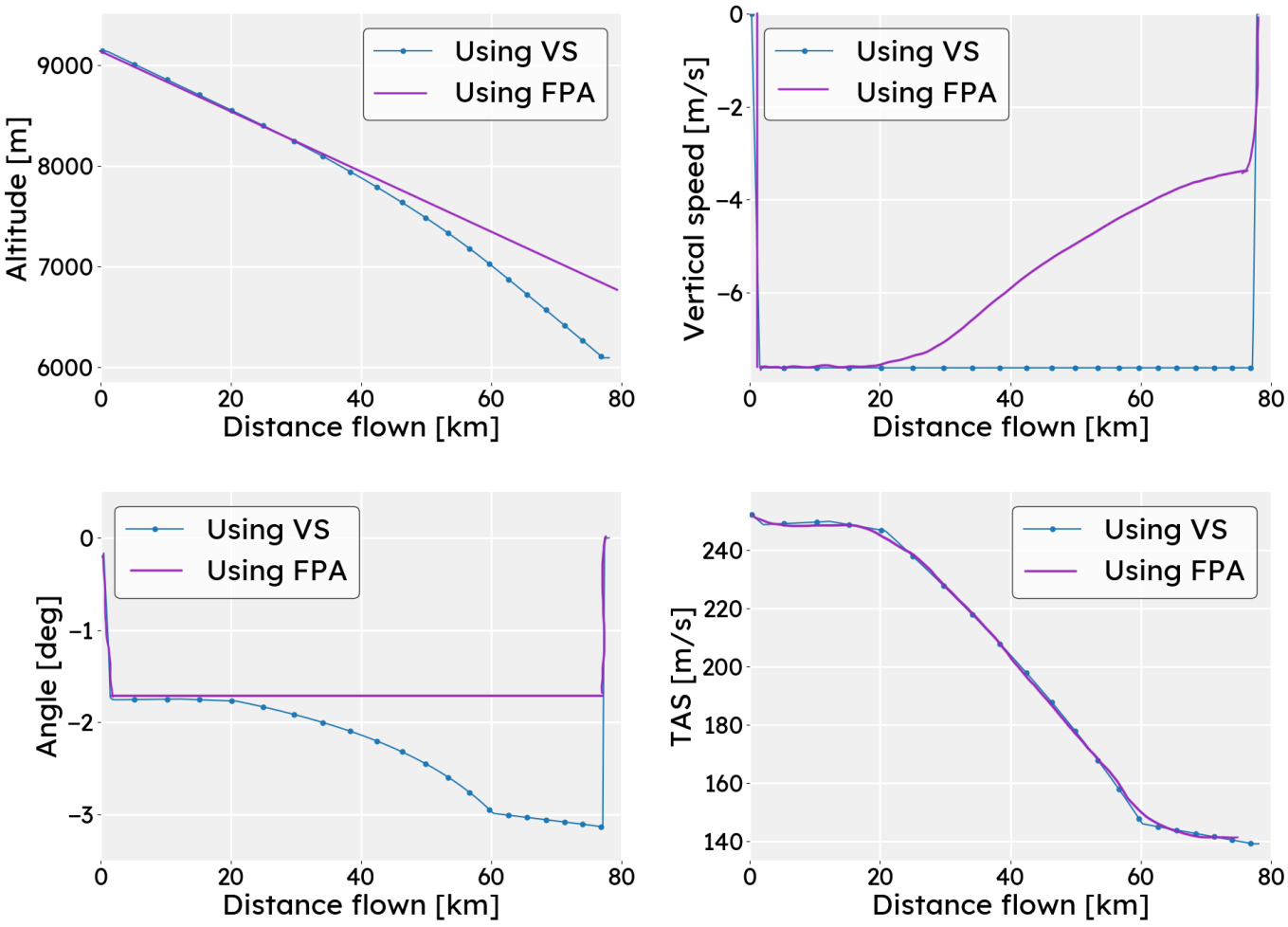

Figure 3.3: Current and proposed BlueSky descent options

The blue lines represent current BlueSky behaviour. The aircraft simulated has been created with the following command file.

```
00:00:00.00 > CRE A-[0] A320 52 3 250 fl300 320
00:00:01.01 > ALT A-[0] FL200
00:00:01.01 > A-[0] ATALT FL280 SPD A-[0] 200
```

What can be seen, is that the vertical speed becomes 1500 feet per minute right away. The altitude drops, but after 20 travelled kilometres, the altitude of FL280 is reached. At this point the speed is reduced to 200. What happens is that, although vertical speed remains constant during the flight, the altitude drops quicker per travelled distance. Reason being, the vertical rate remains constant, whilst less ground is covered by the overall speed of the aircraft. The example of Figure 3.3 shows very well how vertical speed should be used with care when approaching an airport.

Using the FPA can solve these problems. As mentioned in Equation (3.1), the relation determines the way the altitude is changed. If, instead of the 1500 feet per minute, the aircraft had been given a FPA, the situation would be different. Figure 3.3 shows that situation using the drawn purple lines. A constant FPA would provide a constant altitude decrease when considering travelled distance. To maintain that angle, vertical speed must be changed as well.

Both Figure 3.2 and Figure 3.3 have shown how BlueSky misses two essential ideas relevant for descent and climb performance. Both have also been illustrated to give an idea of what the expected situation will be when introducing these ideas. The next step was to incorporate these ideas into a new version of BlueSky, which will be treated in Section 3.3 and subsequent sections.
Moreover, by this point it was decided to first set up a method to outperform the rules of thumb for vertical and horizontal displacement. Logic behind VNAV was to be evaluated as well, but was considered as an extension of the basic simulation performance. Therefore, first the basic logic had to be working before any steps with VNAV could be made.

3.3 Expanding current logic

Due to the current environment, seen in Section 3.2.2, the idea was to expand BlueSky with a thrust model and the option to give flight path angle commands. This would significantly aid in descent and climb flights as the commands options increased the possibilities.

Expanding the logic meant familiarising with current, real-world autopilot logic. This would provide wat is most intuitive. To do so, the flight simulator FlightGear was used.[3] This simulator focuses more on realism than graphics, and therefore provided a good hands-on idea of flying with an autopilot. This meant turning on the autopilot, but turning off the autothrottle. This gave an idea of present day logic how this would fit in a logic scheme for BlueSky, be it with the addition of personal preference. Meaning, not all of FlightGear its logic was copied 1-on-1.

3.3.1 Equations of Motion

Understanding how thrust settings work require a new glance at equations of motion. Considering the forces seen in Figure 3.4. Note that this is a heavy simplified model of an aircraft, point-mass, considering only the four governing forces.

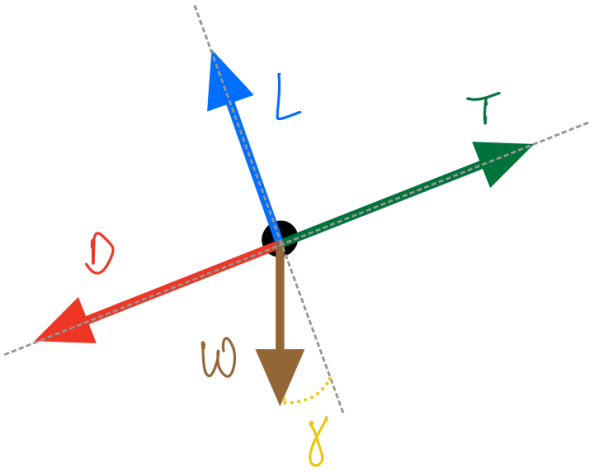

Figure 3.4: Forces equilibirum for point mass

To attain forces equilibrium, the equations of motions can be set up in horizontal and vertical direction, seen in Equation (3.3) and Equation (3.2). Note that weight here is in Newton, and represents a force rather than mass.

$$F_y = ma_y = \sum_{y\uparrow+} L - W \cdot cos(\gamma) \tag{3.2}$$

[3] See: https://www.flightgear.org

$$F_x = ma_x = m\frac{dV}{dt} = \sum_{x\to+} T - D - W \cdot sin(\gamma) \tag{3.3}$$

Equation (3.3) is then multiplied by V to obtain Equation (3.4).

$$mV \cdot \frac{dV}{dt} = VT - DV - W \cdot \frac{dh}{dt} \tag{3.4}$$

Rearranging and noting that $V \cdot sin(\gamma)$ is equal to the vertical displacement per unit time yields Equation (3.5).

$$W \cdot \frac{dh}{dt} + mV \cdot \frac{dV}{dt} = (T - D)\, V \tag{3.5}$$

Which can also be seen as an energy balance, seen in Equation (3.6). This is not further used, and merely shown to demonstrate the energy principle behind the force equation. However, it did provide inspiration to determine the thrust setting, further elaborated on in Section 3.4.6.

$$\frac{dU_{pot}}{dt} + \frac{dU_{kin}}{dt} = T_{net} \cdot V \tag{3.6}$$

A couple of things were deduced from Equation (3.5). In a regular autothrottle case, this equation is constantly kept in balance by a varying thrust. For example, any vertical or horizontal displacements can be compensated by changing the thrust to keep the speed constant. Consequently, thrust can be seen as the 'free' variable, meaning it changes depending on the other parameters. This is necessary, as this is just *one* equation but *three* variables, namely thrust, speed and vertical speed. Though now a thrust setting is introduced as well, which makes the value for thrust a given, rather than a free variable. This would mean that both $\frac{dV}{dt}$ and $\frac{dV}{dt}$ are a result rather than a setting.
To be able to solve this equation without causing unexpected behaviour, there should constantly be a free variable and two others known. Meaning, if a thrust is given, there should at least be a speed or altitude issued as well to solve the equation. This 'triangle' of thrust, speed and vertical speed is a constant process to determine the unknown variable.

3.3.2 New logic

Section 3.3.1 showed that there should always be a balance of variables when changing thrust settings when considering Equation (3.5). Meaning, if one variable is set, another variable should be (indirectly) as well to let the third variable be free. This means that current command logic should be expanded, referring to Section 3.2. Currently, SPD and ALT are commands to directly control the aircraft, with the option of supplying a vertical speed to change the rate of climb. It was decided to expand this with a THR (thrust setting) and FPA (flight path angle). Thrust would be an input in the form of the unit interval, or a percentage from 0% to 100%. The flight path angle would have similar functionality as vertical speed, it can be provided in case of an altitude change.

This meant that a whole new range of command combinations was created. Referring back to Section 3.3.1, it was clear that each aircraft would have to have a combination of two commands in order to solve the equation. Therefore, the last two commands of every aircraft had to be saved to determine its behaviour. For example, if a speed command is given and is immediately followed by a thrust command, the speed and thrust must be held at those indicated positions. This means that altitude must be sacrificed (either in descending or climbing manner, depending on the values) in order to maintain the set speed. Note that the behaviour would be identical if first the thrust command would be given, and then the speed command.
The background behind this choice is that it was assumed that the user would give commands in the order of his preference. If a command is 'new', it is obvious that it must be executed. The longer ago in terms of order a command was issued, the less importance it would have.

That being one example, the entire logic has been worked out and is presented in Figure 3.5. This figure should be seen as a flow chart, indicating what variable is free whenever two other commands are given. It is important to note that commands are stored chronologically. Meaning, THR SPD means that SPD has been given as most recent command. Although it does not make a difference whether the sequence is THR SPD or SPD THR, it does decide which command is kept in case of a new command. The first command is omitted, the second is moved one position to the left, and the newest command is put in the most recent position.

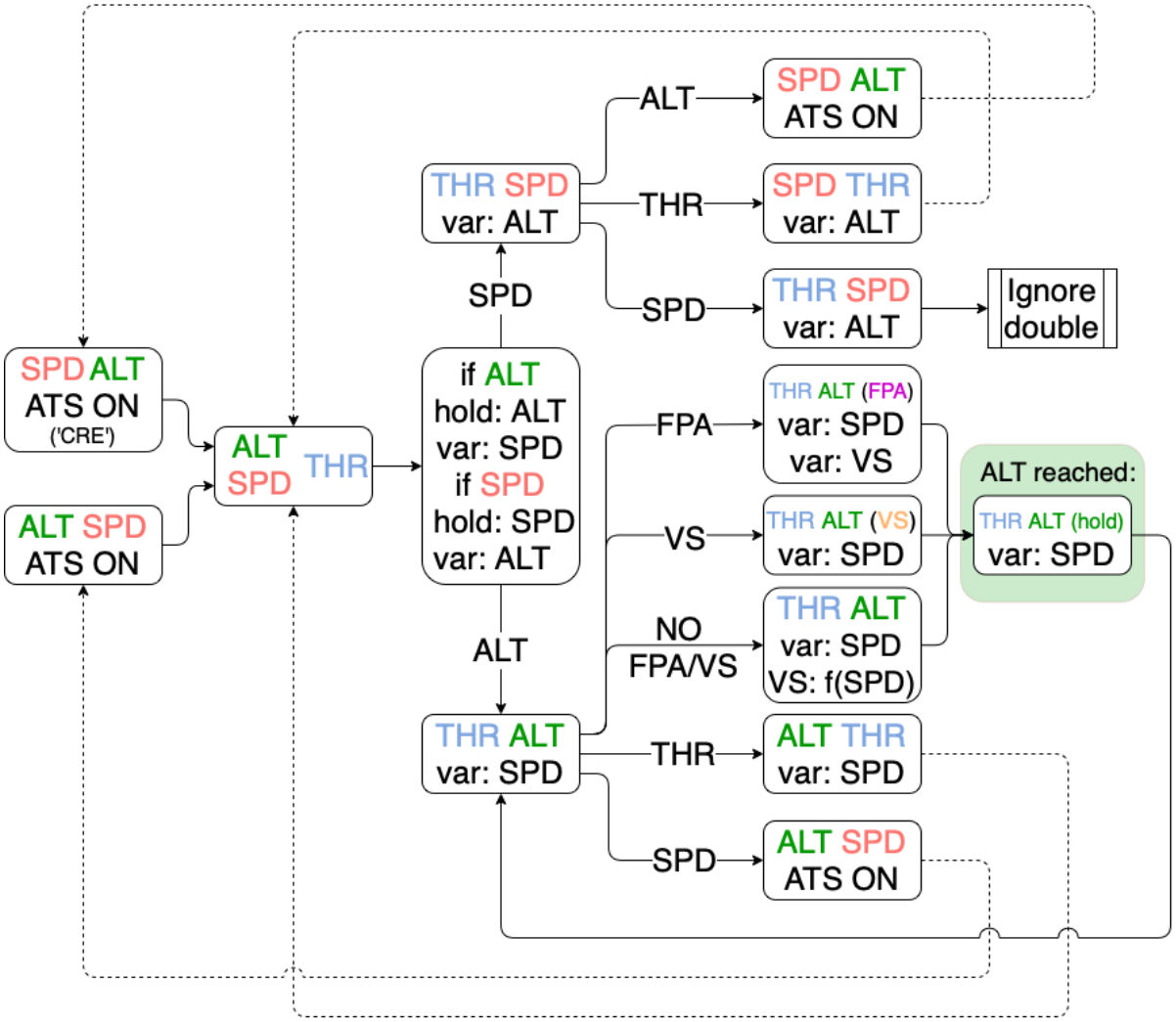

Figure 3.5: New logic for BlueSky

What follows is a textual explanation of how Figure 3.5 should be read.

The figure starts in the situation that a SPD and ALT (or ALT and SPD) command have been issued, which is the case upon creation of an aircraft. This sequence of SPD and ALT was a deliberate choice: when issuing a thrust command hereafter, the new command order would be ALT THR, meaning that speed would be free. It made more sense to hold altitude in case of a thrust command right after creation, to prevent the aircraft from falling out of the sky. Note that it is not possible to create an aircraft with a thrust setting, this can only be issued after creation.

Moving to the second column of block(s), depending on whether the altitude was last issued, or speed, decides what happens next, seen in the third column. If the combination would be ALT

THR, altitude would be held, and speed would be free. Vice versa if the combination would be SPD THR. If again a SPD or ALT commando is issued hereafter, the combinations would become either THR SPD or THR ALT. It is assumed to be clear by now what variable is free. From both these aforementioned commands a range of possibilities arise. The possibilities after THR SPD are discussed first.

Giving an THR command provides a earlier explained option. Issuing a new SPD command causes the speed to be adjusted, but is not double registered as this would otherwise have SPD SPD as result. This would indicate two free variables, something which cannot be solved for (referring to Equation (3.5)). However, changing the altitude with an ALT command would yield the combination SPD ALT. This would mean that both speed and altitude would be fixed, therefore requiring thrust to be free. This circles back to the original BlueSky: autothrottle must be turned on to facilitate this. Hence this option also circles back with a dotted line to the beginning of the chart, the situation is identical to the starting conditions.

Next up are the possibilities after THR ALT. It is assumed that a aircraft is flying at a certain altitude and is issued another altitude. One option is that the aircraft is not given a vertical speed (NO FPA/VS), but this would mean that $\frac{dh}{dt}$ as well as $\frac{dV}{dt}$ are both free. To prevent this from happening, the rule of thumb present in BlueSky is used. The default vertical speed of 1500 feet per minute is used to descend. Two other options would be to either give a vertical speed or FPA command. The first would have identical behaviour as NO FPA/VS, albeit with a different vertical speed than the default. In case of a FPA, another issue would arise, explained below the figure and seen on the left hand side of Figure 3.6. Where γ denotes the FPA, VS the vertical speed and SPD the speed of the aircraft.

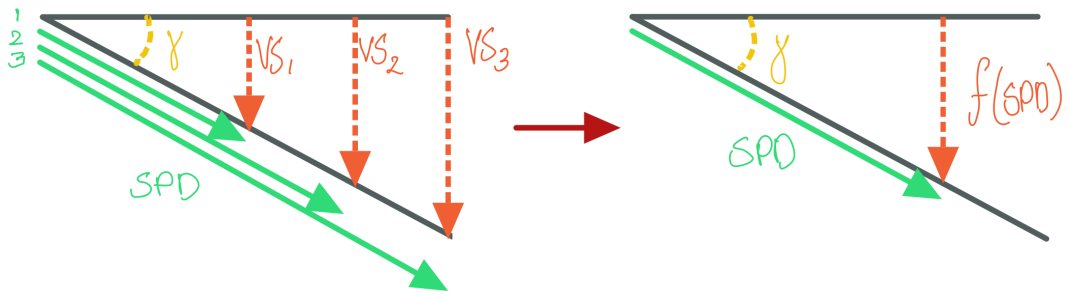

Figure 3.6: Vertical speed as function of speed

Using flight path angles merely expresses the ratio between speed and vertical speed (or change in altitude and change in speed). An unlimited amount of possibilities arise when giving a FPA, as any combination of SPD and VS represent that angle. Setting the vertical speed at the fixed value of 1500 neither presents the solution: this would make the speed also a fixed value thus over constraining the situation. Instead, the vertical speed is a function of the speed, therefore keeping the flight path angle at the desired value.

It holds that in any of these cases the aircraft will eventually reach the target altitude, and hold altitude there. Speed will remain the free variable at that point. One other option is that during the altitude change (with or without a FPA or VS) a THR or SPD command is issued. Needless to say, the aircraft will continue towards the target altitude, albeit with a different thrust setting. In case of a speed setting, the program will resort to autothrottle thus returning to the default vertical speed (which can then be set again).

The logic as presented was regarded to be intuitive and provide the necessary means to expand the way descent and climb flights were executed. To perform these flights, input data had to be generated and formatted in a way to trigger as much as possible of the presented logic. This is discussed in Section 3.4.

3.4 Preprocessing of data

Input data had to be generated to determine whether the logic worked, and build a framework to actually test whether it was true to reality. To tackle both, actual aircraft data was used to generate use cases which the simulator would mimic. The data used for this input file was ADS-B data. This data is simple, yet useful data sent out by aircraft is time periods of seconds. The sending frequency ranges from every second to tens of second. It is therefore a good representation of the flight profile. This data has been collected using a receiver positioned on the roof of the Aerospace Engineering faculty building in Delft, The Netherlands. The data had been collected over the first few months of 2018, where the data of April has been used for this project. One day of data included approximately 4 million lines in Microsoft Excel. These lines contained the following parameters of interest:

- ICAO (unique identifier for an aircraft)
- Latitude & longitude
- Altitude (in feet)
- Rate of climb (altitude change per unit time)
- True Airspeed
- Mach
- Temperature

These parameters were all used in the following subsections describing code groups. The eventual idea behind this piece of code was to write an unsorted .CSV file to a .SCN file containing several different commands. For clarity, the output file is briefly laid out for the reader to better understand what the subsequent code steps have been necessary for. To understand the actual workings of the commands, refer to Section 3.2.1.

Starting off was the creation of the logger within BlueSky, to make sure that output data was being saved to a textfile. Secondly, the creation of aircraft had to be done for all aircraft. This meant providing an ICAO, aircraft type, latitude, longitude, heading, altitude and speed. For sake of clarity, the example of Section 3.2.1 is repeated below.

```
00:00:00.00 > CRE 020119-[0] A320 50.21 2.61 44.91 32200.0 274.23
00:00:00.00 > CRE 020124-[0] A320 49.89 2.49 60.45 30325.0 281.87
00:00:00.00 > CRE 06A063-[0] B77W 52.21 7.21 74.19 27900.0 269.44
```

Subsequently, a first command would have to be issued right away, to tell the aircraft what do to following its creation. This was a combination of two of the three following options: altitude, speed and thrust. In case an altitude was issued, a flight path angle would be issued as well. This was to make sure that the path of the actual flight would be precisely followed as this would tell the aircraft with what slope it had to execute its descent. Demonstrated by the following text slice. Note that the ICAO is necessary to direct the command at a certain aircraft.

```
00:00:01.00 > THR 020119-[0] 0.0
00:00:01.00 > SPD 020119-[0] 0.76

00:00:01.00 > THR 020124-[0] 0.0
00:00:01.00 > ALT 020124-[0] 26375.0
00:00:01.00 > FPA 020124-[0] -2.18
```

Lastly, the remainder of the data points (as the first two have been issued by this point) would be issued. Normally, BlueSky primarily uses either direct, geolocation or the so called 'AT' commands. The first means a command which is immediately executed, the second is a latitude & longitude (be it a predefined waypoint) based command, whereas the latter has been explained in Section 3.2.1. These AT commands have been used in the scenario file, seen below.

```
00:00:01.00 > 44CE64-[0] ATALT 8025.0, SPD 44CE64-[0] 0.44
00:00:01.00 > 44CE64-[0] ATALT 8025.0, ALT 44CE64-[0] 8025.0
00:00:01.00 > 44CE64-[0] ATALT 8025.0, 44CE64-[0] ATDIST 44CE64-[0],3.95 , THR 44CE64-[0] 0.0
00:00:01.00 > 44CE64-[0] ATALT 8025.0, 44CE64-[0] ATDIST 44CE64-[0],3.95 , ALT 44CE64-[0] 7025.0
00:00:01.00 > 44CE64-[0] ATALT 8025.0, 44CE64-[0] ATDIST 44CE64-[0],3.95 , FPA 44CE64-[0] -4.22
00:00:01.00 > 44CE64-[0] ATALT 7025.0, THR 44CE64-[0] 0.0
00:00:01.00 > 44CE64-[0] ATALT 7025.0, ALT 44CE64-[0] 7000.0
00:00:01.00 > 44CE64-[0] ATSPD 217.08, THR 44CE64-[0] 0.0
00:00:01.00 > 44CE64-[0] ATSPD 217.08, SPD 44CE64-[0] 217.0
```

The above part shows a portion of how this would look like. By purely using AT commands, the aircraft is not bound by time or position. It is simply assigned to follow commands when reaching a next state. Ideally, it corresponds, time-wise, with the real aircraft data as much as possible.

The choice for these commands is rather obvious. Using direct commands is not possible: it would mean that all time stamps would be given beforehand. It is simply not possible to know beforehand at what timestamp the command would have to be issued. Perhaps it would be close, but this would let the aircraft follow the time profile, rather than the flight profile. Secondly, using coordinates shares the same reason. It would make the aircraft follow the travelled path over ground (and thus its location profile), but does not follow its flight profile. In this simulation it was all about achieving the same speeds and altitudes as the original flight profile. After all, the aircraft profile had to be followed, but it does not matter whether this happened at exactly the place on the map.

To set up the code in a similar fashion as the idea discussed, the following subsections treat each of the major steps in the code to pre-process the data.

3.4.1 Filtering

Up first was the filtering of aircraft. The ADS-B contained a lot of aircraft (and even air balloons) which could not be simulated by BlueSky. Using OpenSky its database, the ICAOs were coupled to an aircraft type.[4] This selection of aircraft can be seen in Table 3.1.

Table 3.1: Types of aircraft included in the input file

Airbus						
A319	A320	A321	A330 -2 -3	A340 -3	A350 -9	A380 -8
Boeing						
B737 -4 -7 -8 -9	B747 -4 -8	B757 -2	B767 -3	B777 -3	B787 -8 -9	
Cessna	**Embraer**	**De Havilland**	Lockheed Martin			
C550	E170 E190	DHC-8	F16			

Using this table, all undesired aircraft were omitted from the ADS-B data. This provided a dataset (sorted by ICAO) with workable aircraft. Flights were further separated by grouping using time. If two subsequent time points were longer apart than 1800 seconds (20 minutes), then this was considered a different flight. This is relevant as carriers often carry out multiple flights per day for popular destinations. To distinguish these flights, a suffix was added to the ICAO in the form of '[n]'.

[4] https://opensky-network.org/datasets/metadata/ [Cited: 18-01-2022]

A further requirement was imposed that the amount of ADS-B data points had to be at least 30. This omitted the flights that were simply cruise flights or flights on the edge of the scan area of the ADS-B receiver.
Lastly, all flights were omitted that showed behaviour of parabolic behaviour. These were possibly test flights, but did not show realistic domestic airliner behaviour. These flight were also removed by restricting the average slope of the altitude profile and setting a maximum altitude as first data point.

3.4.2 Transformations

The data had to be adjusted as it lacked many key parameters. The TAS needed to be transformed as all speeds in BlueSky (and in general for pilots) are in expressed in CAS. The conversion used the altitude to determine the ISA parameters pressure, density and temperature. Subsequently, the following two formula were used to determine the CAS for every TAS. Equation (3.7) shows the formula for *impact* pressure.

$$q_{dyn} = p \cdot \left(\left[1 + \frac{\rho \cdot TAS^2}{7 \cdot p} \right]^{\frac{7}{2}-1} \right) \tag{3.7}$$

Equation (3.7) is used to account for compressibility effects, which is a step to get from TAS to EAS. In which, p denotes pressure and ρ denotes density. As EAS itself is not relevant in this case, it is skipped and transformed to calculate CAS (in knots) directly. This is shown in Equation (3.8).

$$CAS = \sqrt{7 \cdot \frac{p_0}{\rho_0} \cdot \left(\left[\frac{q_{dyn}}{p_0} + 1 \right]^{\frac{2}{7}} - 1 \right)} \tag{3.8}$$

Furthermore, as both the Mach number and temperature from the data as listed in Section 3.4 showed many missing entries, Mach number itself had to be recalculated as well. Using the ISA values to determine the temperature, the Mach number was calculated using Equation (3.9).

$$M = \frac{TAS}{\sqrt{\gamma \cdot R \cdot T}} \tag{3.9}$$

Lastly, the rate of climb was recalculated as well as these values were rounded and regarded as not sufficiently accurate. This was achieved by simply dividing the altitude difference over the time difference. This assumed that the rate of climb was kept constant between every ADS-B data point.

3.4.3 Ramer-Douglas-Peucker

The next main point was to reduce the amount of data points per flight. The issue of the data was the overcompleteness. By having almost data for every second of the flight, there is little for the simulator to 'think'. Every point would be fed to the computer and one could hardly speak of segments. The desire was to create legs, which each could be characterised by being a climb, descent or level flight. In these phases, there could either be a deceleration or acceleration, or a constant speed. To let the simulator think for itself, and thus demonstrate its capability, a lot of data points had to be omitted. The only remaining data points of interest were positions in which a flight fase changed. Meaning, if a climb flight transitioned into a level flight at a certain altitude, that would be a point to conserve.

The Ramer-Douglas-Peucker algorithm was used to aid in this process, which is briefly explained. The most important, and only, parameter is ϵ. This symbol is a specified distance, in which either lie or do not lie. Considering an arbitrary line that is made out of multiple points, and a line is drawn between the first and last point (representing a bandwidth). The point furthest away is first evaluated. If this point lies further away than the distance specified by ϵ

(measured from the drawn line), then the point is kept. Drawing a line between the first point, and this new point repeats this procedure. If the interlying furthest point is within the ϵ distance they are removed. This concept can be seen in Figure 3.7. An explanation of the concept follows below the figure.

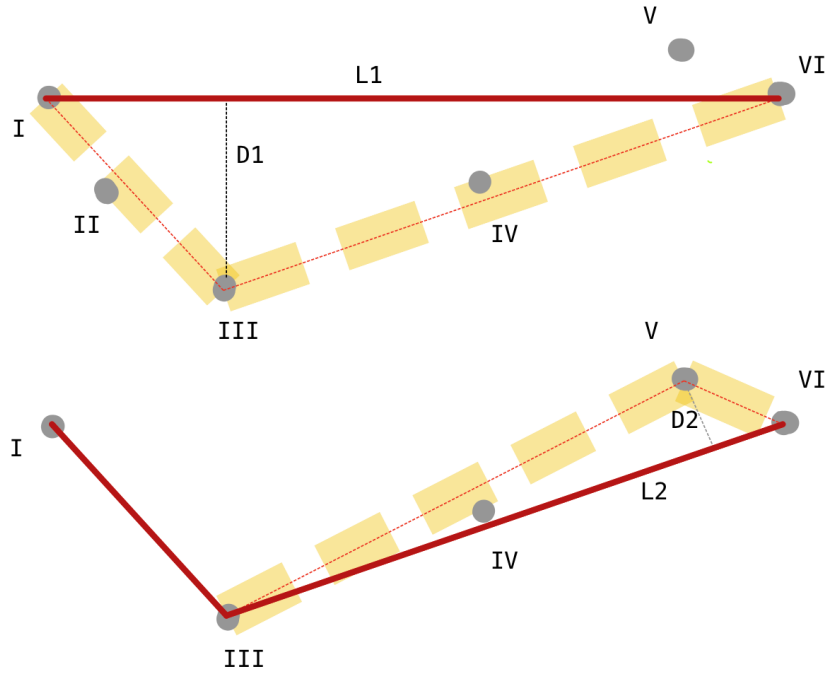

Figure 3.7: Visualisation of the Ramer-Douglas-Peucker algorithm

Consider the points I to VI seen in the top sketch of Figure 3.7. Point I and VI are connected by L_1, as they are furthest away from each other. Distance D_1 is drawn to the point furthest away from this line L_1, being point III. The line between I and III is surrounded by a yellow bandwidth, which represents the ϵ. It can be seen that point II falls within this range, therefore being removed. Subsequently, a line is drawn from point III to the furthest point VI, seen in the bottom sketch of Figure 3.7. Here, point V is furthest from L_2 where a distance D_2 is used as distance for the point furthest away. The result is that IV falls within the range, thus being removed. As there are no further points to be removed, the result is a line between the points I, III, V and VI. The larger epsilon is, the more points are reduced and the larger the simplification.

For this research, an ϵ factor of 25 was introduced, based on iterative visual inspection. This algorithm drastically reduced the amount of data points and formed a range of data values that were manageable whilst keeping the curve as much as possible true to the original data. To give a visual idea of this concept, Figure 3.8 shows the result of three flights.

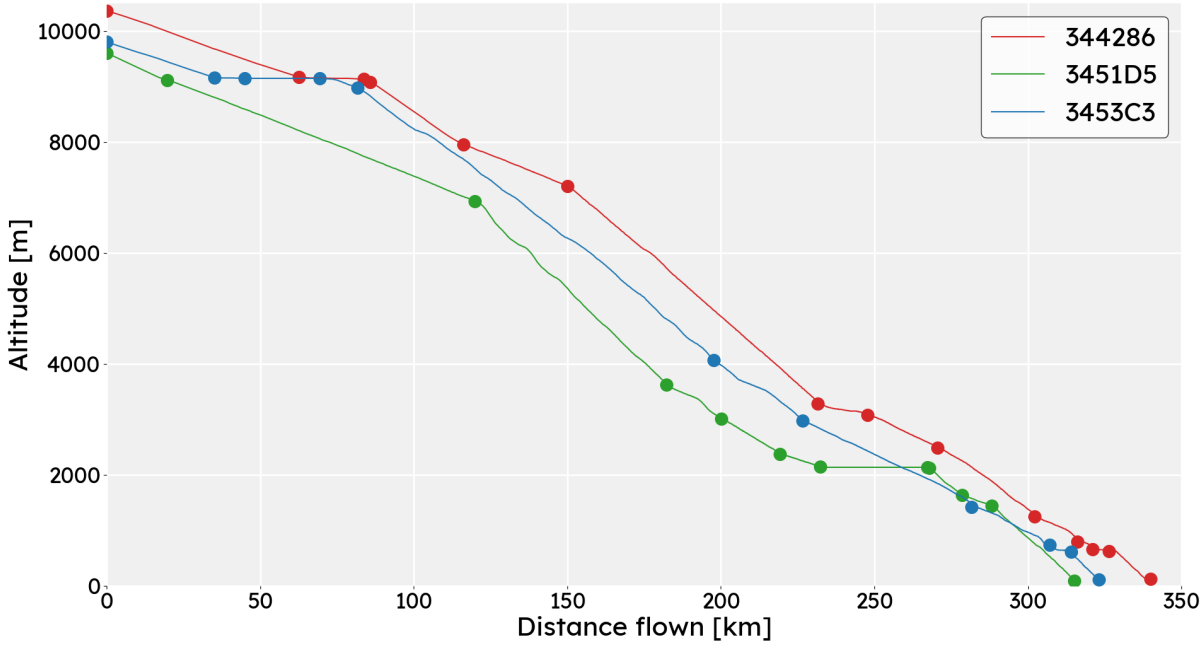

Figure 3.8: Three flights with their reduced RDP version

These lines represent the ADS-B data, with their reduced format represented by the dots. The method is not flawless: some points are superfluous, but it was regarded as impossible to find an optimum ϵ for all thousands of flights. This would require too much time to visually inspect every flight. Instead, the general value of 25 was used for all flights. This value is rather on the lower end than on the higher: generally there are more points than required, rather than too few. This would cause short segments in some cases.

3.4.4 Flight path angle

Aircraft changing altitude would do so with a flight path angle, or have a set throttle and speed, so that altitude becomes variable. Considering the first option meant that the flight path angle had to be known for all altitude changes. Later logic (mentioned in Section 3.4.5) would indicate which altitude changes made use of what way of descending.
To determine the FPA, the aircraft its altitude and travelled distance was required. Another possibility was to use the rate of climb and speed per unit time. However it was, as stated in Section 3.4, chosen not to use the rate of climb from the .CSV file due to its deficiency.

Unfortunately the data set contained neither the FPA or travelled distance. Therefore, using the GeoPy package, the latitude and longitude points were transformed to degrees coordinates using the Point function. Subsequently, these points were used in the distance function, calculating the geodesic distance between two data points. This option only presents the geodesic or great-circle distance, not the euclidean. Both take into account the curvature of the earth (in slightly different ways). Note that altitude was explicitly not inserted as this would not provide longitudinal travelled distance but rather the actual travelled distance.

The FPA was then calculated using the distance to fly and the difference in altitude between two adjacent points. Note that the distance here used was nautical miles, as this units forms the input for distance in BlueSky.

3.4.5 Fuzzy Logic

At each of the segments between data points, the aircraft had to be evaluated to determine which phase it was in. Meaning, was it ascending, descending or flying level? And, did the aircraft accelerate, decelerate or fly at constant speed? The combination of these possibilities made that each flight segment had to be classified by one of the nine of combinations possible. This required arbitration. Simply deciding that a certain specified range of flight path angles is considered descent (and others not) is subjective and not scientific. Therefore, a Fuzzy logic was used to distinguish phases.

Fuzzy logic can be seen as an allocation of membership functions in a continuum [16]. Instead of setting hard boundaries as mentioned earlier, a variable can belong to either state A or state B, depending on how these are defined. The user can set up these regions A and B by specifying probability functions. It is up to the user to decide what the shape of the function is and how large the bandwidth between certain and zero possibility is. Admittedly, this also comes down to reasoning and literature to decide what is reasonable.
However, the real difference is the assignment of memberships. Memberships can be seen as the probability that a certain value falls within a certain range. For example, if flight path angle of an aircraft is negative, its probability that the aircraft is climbing, is low. However, if its angle would be a large positive number, the probability of climbing is equal to one. Both input as output has a membership function. Membership function of a climb, accelerating flight will require a large probability for speed change, flight path angle and rate of climb. Output memberships are assigned using the individual probabilities.

The states themselves were the rate of climb, flight path angle and minimum speed change. Rate of climb was determined using the altitude and time. The angle was deduced using co-ordinates and altitudes. Therefore, both represented a different entity. The speed change is the determination of speed change between two points. This was evaluated for both CAS and Mach, so one of the two was the minimum change. If this minimum change was sufficiently small, the segment could be considered as 'constant speed'. These three states each had a probability function set up. Which are shown in Figure 3.9 [17].

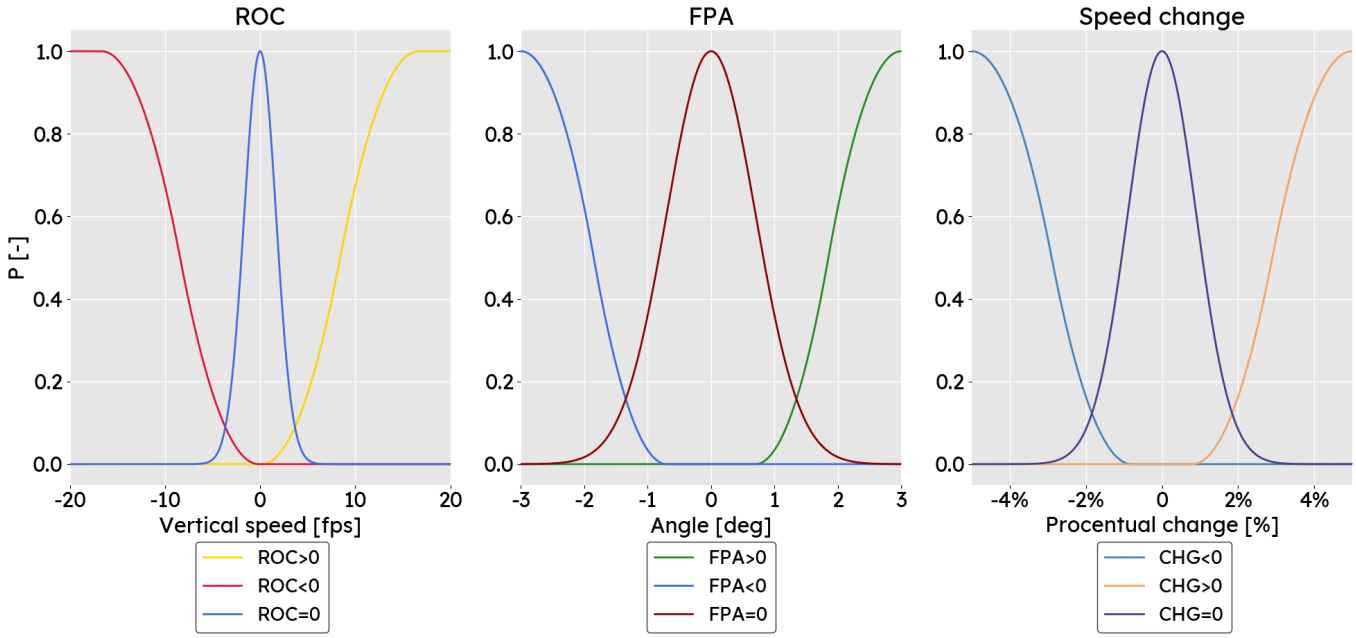

Figure 3.9: Probabilities of the fuzzy states (RoC, FPA, Speed change)

In these graphs three types of functions can be distinguished. To set up the left side of each state, Z-functions were used. For the right side, S-functions were used and around zero a normal distribution was used. The Z- and S-function are simple functions where a start and end is given, where respectively the the probability is zero and one.
For example, when considering the flight path angle, it can be seen that from an angle of -3 the probability of a 'FPA minus' state is 1. Thus, the angle is guaranteed to be a 'minus' angle if larger then -3. Another example would be a speed change of 0.5%. The probability of it being a 'zero change' case is around 0.8.

The following step was to set up the different phases the segment could be allocated to. As mentioned in the introduction of the section, there were a total of nine different states. Table 3.2 shows these and how they are determined. The acronyms are as follows: the first letter denotes descending, level or climb by D, L or C respectively. The second letter stands for decelerating, constant or accelerating, denoted by D, C or A respectively.

Table 3.2: Possible phases for flight segments

		FPA and RoC		
		-	0	+
Speed	-	DD	LD	CD
	0	DC	LC	CC
	+	DA	LA	CA

Using aforementioned logic, each flight segment was assigned the corresponding phase. The segments were a result from the algorithm explained in Section 3.4.3.

3.4.6 Thrust setting

The most important step in the preprocessing was the determination of the thrust settings. Meaning, how are the thrust levers set as function of the maximum available thrust. Independent of when which setting was to be used (regarding the new logic, discussed in Section 3.3), a methodology had to be set up to determine the value of the setting itself. In this process, three parameters were essential. These were the mass of the aircraft, the drag and the total thrust available. The latter two had to be calculated at each data point. One important note is that a ballpark value of the mass was required, compared to the real flight, but no exact value. This was simply not possible and not within the scope of this research. Moreover, the mass would be set equal for all simulations. Thus, it would not cause a discrepancy among the simulations relatively. There would only be a difference with the actual ADS-B data.

Determination of mass was done by using the OpenAP module earlier discussed in Section 2.1.6. For the simple reason that BADA does not have a standalone module. Moreover, as BADA becomes more proprietary, it was deemed wiser to lay focus on OpenAP. OpenAP contains a large amount of data on all kinds of aircraft, including different weights, dimensions and specifications. Each ICAO was used to get the specific data of that particular flight, thanks to the classification discussed in Section 3.4.1. The Maximum Take-Off Weight (MTOW), the Operating Empty Weight (OEW), length and width were taken from the data sets. The OEW including passengers was determined using Equation (3.10). Note that both OEW and MTOW are in kilograms, rather than being an actual 'weight'.

$$OEW_{+PAS} = OEW + 1.35 \cdot 100 \cdot \text{length} \cdot \text{width} \tag{3.10}$$

In Equation (3.10), two constants stand out. These two parameters have been determined by evaluating several aircraft its passenger to length/width ratio. The number 1.35 can thus be seen as a metric for passenger per length per width (in meters) of the aircraft. Note that this is

done with aircraft supported by BlueSky. This is an average value and of course the exact value differs per type of aircraft. Moreover, the number 100 is the average weight of a passenger. That is, a passenger including its luggage and other weights [18].

Then, depending on whether the aircraft was in descent or climb, the fuel fraction was determined. These fractions were calculated to be for 0.3 and 0.42 for medium and long haul respectively [19]. The distinction was made using the FAA weight classification.[5] Aircraft over 115,000 kg were considered long haul, and under 115,000 kg medium haul.
It was assumed that climbing aircraft just left the airport and therefore were fuelled up. On the other hand, descending aircraft were assumed to be at the final stages of the flight, thus carrying far less fuel. The fuel mass was therefore multiplied by either 0.1 or 0.9 to account for an almost empty or full fuel tank.
The total mass was calculated by adding this calculated fuel mass to the OEW_{+pas}.

Next to the thrust calculation itself, the determination of drag also required the value of mass. OpenAP contains drag polars set up with historical data, from which the drag value is determined in combination with the inputs aircraft type, mass, TAS, altitude and flight path angle. The angle used was as described in Section 3.4.4. This was done for each data point.
Again, using OpenAP the maximum available thrust was calculated for each data point. This required the aircraft type, phase (as presented in Table 3.2), TAS, altitude and rate of climb.

When looking at basic physics, a force (and subsequently the fraction of maximum available force) is determined by multiplying the acceleration times the mass. Forces would be difficult to determine as the acceleration would be difficult to determine. Instead, the concepts of energy and work were used, which was clearly demonstrated in Equation (3.6). The change in energy is a function of force and speed (power), but so is the total energy a function of work done. Potential and kinetic energy were summed at all data points, resulting in the energy difference between each data point (E_1 and E_2). Moreover, the drag would be acting at all of these data points as well (D_1 and D_2). Visually, this can be seen in Figure 3.10, where L denotes the euclidean distance between the points and T is the net thrust.

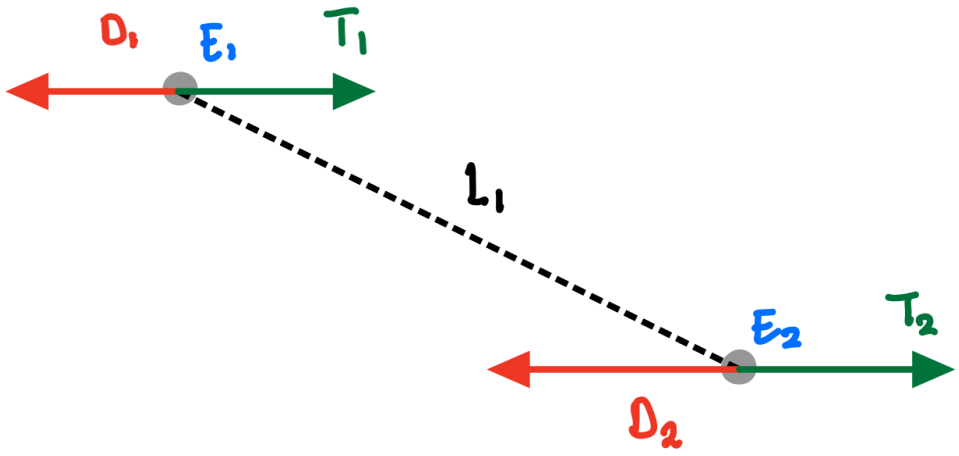

Figure 3.10: Energy visualisation to determine thrust setting

Figure 3.10 exaggerates the flight path angle for clarity. However, to calculate the work done by the drag, the drag force was assumed to be in line with the flight path as flight path angles were considerably small. This drag force was added to the energy that had to be 'overcome' by the thrust force, for the case of increasing total energy. In case of a decrease in energy, drag played a large role and the thrust setting was low. On the other hand, if the decrease was small,

[5] https://aspm.faa.gov/aspmhelp/index/Weight_Class.html [Cited: 20-02-2022]

there had been a significant thrust setting to compensate the energy loss due to work done by the drag force. Summarising, thrust setting *Ts* was determined as seen in Equation (3.11).

$$T_s = \frac{\Delta E + L_1 \cdot D_1}{L_1 \cdot T_{available}} \tag{3.11}$$

Note that D_2 was not used in this calculation. It was assumed that the drag D_1 was present during the segment with distance L_1. Taking an average between the two values for drag would not be representative as the segments could be two whole different phases, indicating significant different drag values.

These thrust values required one further transformation. A lot of values turned out be either negative, whilst some were larger than one. The former can be attributed to the fact that a lot of idle segments have an energy decrease due to drag rather than thrust. The latter can be attributed to simplifications (e.g. no wind) made in the process. To attain a workable result, all results below 0.07 were set to 0.07, and all values above 1 were set to 1. The 0.07 value comes from OpenAP literature, which represents idle thrust. Meaning, if the engines of the aircraft are put into idle mode, they will still generate around 7% of the total thrust available. The result of all thrust settings can be seen in Figure 3.11. These values have been determined after the reduction of data points, therefore this plot contains the thrust values for around 3000 data points.

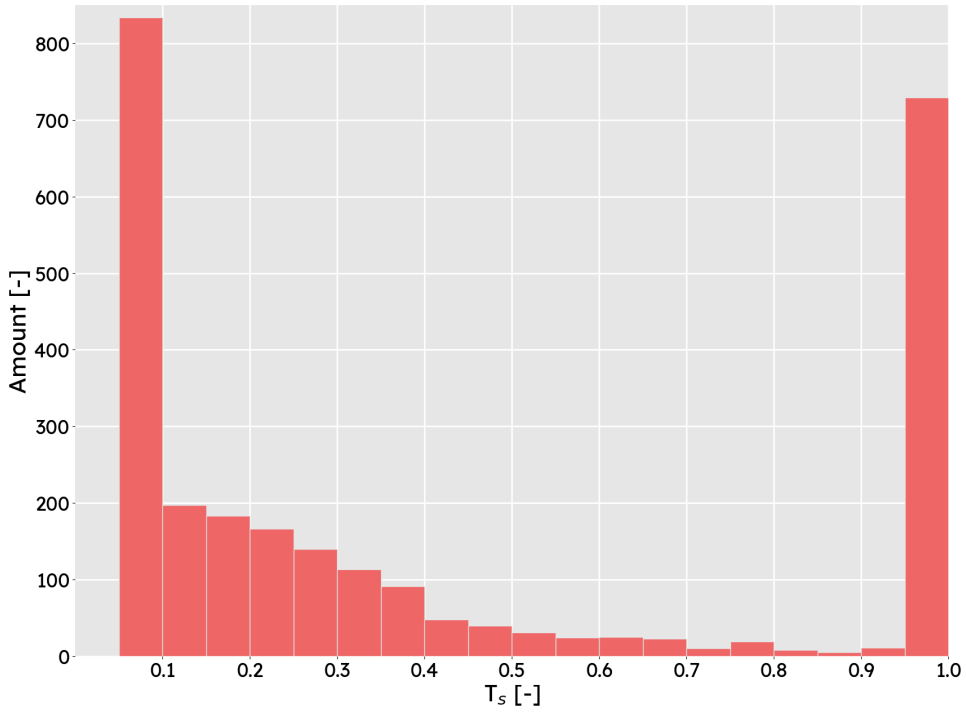

Figure 3.11: Spread of thrust settings for 3000 data points

The histogram shows a comparable amount of descent settings, as well as climb settings. This does not mean that the amount of climb and descent flights is similar, it simply means that the amount of settings is similar. Furthermore, it demonstrated that the used methodology to filter climb and descent flights was correct, as the thrust settings showed corresponding values to these kind of flights. This meant that the data was ready to be used by the new version of BlueSky, discussed in Section 3.5.

3.5 Changes to BlueSky code

This section will conceptually (and partly) technically explain what code changes have been made to BlueSky. It is not expected of the reader to be full aware of the BlueSky code, neither is it necessary. The idea of this section is merely to explain what the structure is of the implemented code. However, basic Python knowledge is necessary to understand code snippets. The reader can then conceptually understand how it functions. Therefore, it cannot be seen as a full guide as much of the smaller code changes are left out. This section will also use a lot of ideas explained in Section 2.3, with an emphasis on Figure 2.8.

As Section 3.3 explained, commands had to be saved in order to determine the behaviour. This was achieved by adding a small function to the autopilot module seen below.

```
def cmdchg(self, idx, cmd) :
#Dictionary to transform commands into characters
dict=['spd':'S', 'alt':'A', 'thr':'T']

#Check if the command is not equal, if so, move both one position
if self.lastcmd[idx][1]!=dict[cmd]:
    self.lastcmd[idx][0]=self.lastcmd[idx][1]
    self.lastcmd[idx][-1]=dict[cmd]
elif self.lastcmd[idx][1]!=dict[cmd]:
    return False
```

Whenever a command would be similar it would be discarded. This meant that the self.lastcmd list became a list containing two of the following three parameters (in any combination): 'S', 'A' or 'T'.
This list was subsequently used in the traffic module. In this module, the basic variables like speed, vertical speed and altitude were calculated. Of course, whenever a command sequence would contain a combination of 'S' and 'A', regular logic was used. However, if any other of these combination would be the case, different array calculations would take place. An example of which can be seen in the code below. Note how variables from the 'perf' class are the result of performance model calculations.

```
self.vs = np.where(self.swats,
              np.where(need_az, self.vs+self.az*bs.sim.simdt, target_vs),
              np.where((indexSpd),
                       ((self.perf.thrust - self.perf.drag) * self.tas-self.perf.mass*self.tas*self.ax) /
                       (self.perf.mass * g0),
                       (np.where(need_az, self.vs + self.az * bs.sim.simdt, target_vs))))
```

This 'numpy.where' cascade was the general idea behind the determination of the other variables like acceleration as well. Initial selection happened with self.swats, which indicated whether autothrottle was on or off. Throughout these command lines, indexSpd, indexAlt or indexATS would determine whether the aircraft would have thrust, speed or altitude free, respectively. An example of an index line can be seen below.

```
indexATS = np.all(np.asarray(cmd)==['A','S'] , axis=1) | np.all(np.asarray(cmd)==['S','A'], axis=1)
```

Calculating the variables required values for thrust, drag, mass and other performance parameters. As explained in Figure 2.6, there exists a general base performance class. This class connects any performance model to the traffic module. As thrust and acceleration were initially differently calculated, these were changed for both BADA and OpenAP. Thrust became a fraction of the maximum thrust available, and acceleration became a function as in Equation (3.5).

BlueSky had been extensively tested by this point, and was rendered to be able to simulate the new created scenarios. These scenarios were a results of the preprocessing discussed in Section 3.4. The next step was to set up the validation, which would show the accurateness of the implemented code.

3.6 Validation set-up

Validation brought the research back to the aim. What was to be shown, in order to demonstrate the effectiveness of the research? The research, simply put, had to show that rules of thumb were outperformed and that extra possibilities were added to BlueSky. This meant that BlueSky had to be tested quantitatively, and qualitatively had to be shown that the logic actually worked.

Several metrics in terms of energy and travelled distance were attempted. The cumulative energy was used as parameter to compare energy profile, and thus whether the overall flight profile would be comparable in combination with the altitude profile. Solely testing for altitude profile would namely not prove whether the correct speeds were adhered to as well. Moreover, level segments were used to demonstrate the improved accuracy of the model. By comparing the travelled distance on level parts, it could be demonstrated that the standard acceleration used in BlueSky was bested by the new model. Doing this for the large amount of flights available would prove its robustness.

However, results showed significant differences in flight profiles, which led to the question what was the cause of this deviation. Indications quickly led to the thrust setting, as almost all aircraft in climb would not gain altitude. They would hardly ascend and quickly lose speed, crashing as result. The original quantitative analysis was therefore rendered not possible, leaving a qualitative focused analysis.

By this point, both the idea of improving VNAV together with the quantitative analysis were cancelled, but still the basic logic had to be proven to work. The qualitative analysis was split into two parts. One of which was the demonstration of overcoming the limitations as posed in Section 3.2.3. This would be a proof of concept, but would not prove its validity for entire flight profiles.
Therefore, several climb and descent flights were selected to show the workings of the model. For descent flights, this selection was done based on path angles present in the flight profile. Reason being, the thrust setting seemed to pose the problem. Reducing the dependency on said parameter, by selecting flights with as many idle segments as possible, would reduce the chance of a failing flight.
Similarly, the climb flights were adjusted so have a thrust setting of 1 in each segment. As this would still not provide sufficient result, exclusively flights with shallow path angles were selected for this part. The total fuel quantities were also put in a table to further quantify the performance of these flights.

The results were created making use of four different data sets, being: the ADS-B data (reduced using RDP) BlueSky using autothrottle (BADA), and using thrust settings using OpenAP and BADA. During the simulations, OpenAP showed to have issues with the net thrust and maximum thrust. When using autothrottle, the net thrust would often exceed the maximum available thrust. This indicated that there was an error in the thrust model of OpenAP. Therefore, it was decided to use the BADA model for autothrottle simulations. To compare the differences between BADA and OpenAP, both models were used to simulate the scenario using a thrust setting. Note that the preprocessing was still done using the OpenAP model, as there was no BADA alternative at hand.
The profiles to be created were those of altitude, speed, vertical speed and work. Work for both the old and new situation has been derived by multiplying the travelled distance by the thrust setting, and maximum available thrust. The results can be seen in Chapter 4.

3.7 Sensitivity analysis

The inconsistency in model between preprocessing and simulation introduced an error. The sensitivity analysis was carried out to demonstrate the relevance and impact of such an error. To do so, both mass and maximum available thrust were taken as parameters. These two parameters had a significant influence on the outcome of the simulation. However, mass was determined empirically and the thrust was calculated using the OpenAP module and energy balance. As this module showed odd behaviour mentioned in Section 3.6, it required further inspection.

The set up for the sensitivity analysis was as follows. To prevent falling short of any data due to thrust deficiencies, solely descent flights were chosen to be sure of a (almost) full flight profile. Both were varied from -10% to +10% in steps of 5%. These values were chosen to ensure a significant enough difference from the standard value. Moreover, especially when considering the mass, the uncertainty was predominantly in this range.

As stated, both mass and available thrust were chosen as varying parameters. Note that there explicitly has been chosen for the maximum available thrust.
Thrust was calculated using the methodology explained in Equation (3.11). The available thrust is a parameter taken from OpenAP. Therefore, it can (mainly) be seen as a black box calculation. Taking the available thrust actually test how much that external variable impacts the simulation, rather than multiplying all variables in Equation (3.11) by a certain factor.

A short explanation follows to help understand what changing these two variables theoretically (and practically) could mean.
Firstly, using the equation mentioned in Equation (3.5) and repeated here in Equation (3.12) for clarity, the mass will impact the following.

$$W \cdot \frac{dh}{dt} + mV \cdot \frac{dV}{dt} = (T - D)\, V \tag{3.12}$$

In case of level flight, increasing the mass means increasing the required deceleration or acceleration to come to a certain speed. Level parts with changing speeds will (theoretically) require more distance to come to the same result. Moreover, when assuming $\frac{dh}{dt}$ to be set, means that the $\frac{dV}{dt}$ will be smaller. It cannot be said that this leads to better or worse performance, as extra mass can actually lead to a sign change in acceleration. Which can result in achieving or not a next 'AT' command.

Secondly, increasing the total amount of available thrust makes a difference in what thrust is set. Again, using the equation mentioned in Equation (3.11) and repeated here in Equation (3.13) for clarity, the thrust will impact the following.

$$T_s = \frac{\Delta E + L_1 \cdot D_1}{L_1 \cdot T_{available}} \tag{3.13}$$

A larger available thrust will in fact lead to a smaller net thrust, vice versa. Therefore, an increase of 10% in available thrust can cause thrust settings to be too low to achieve a certain acceleration. On the other hand, it can also mean that certain accelerations are achieved that were not initially.

Therefore, it cannot be said whether it provides better results whether one increases or decreases parameters, but it does explain what happens with the variables conceptually. Together with the flight profiles that will be produced, according tables will show the travelled distances to quantify the results. The result of the sensitivity analysis can be seen in Section 4.3.

4

Results

This chapter discusses all the results following the method described in Chapter 3. These results are sorted in a couple of sections. The first section will cover the proof of concept. This will be used to show that implementations are working fundamentally. To actually prove that these concepts lead to a successful flight profile, a total of four flight have been chosen to display both descent and ascend performance.

4.1 Proof of concept

To demonstrate both the thrust setting and flight path angle, the mentioned flights in Section 3.2.3 (Figure 3.2 and Figure 3.3) are repeated in Figure 4.1 and Figure 4.2. These graphs were shown to illustrate what kind of behaviour was to be expected. To actually see if these ideas have worked, the first flight has been tasked with the same commands, but this time with an idle thrust setting. The result can be seen in Figure 4.1.

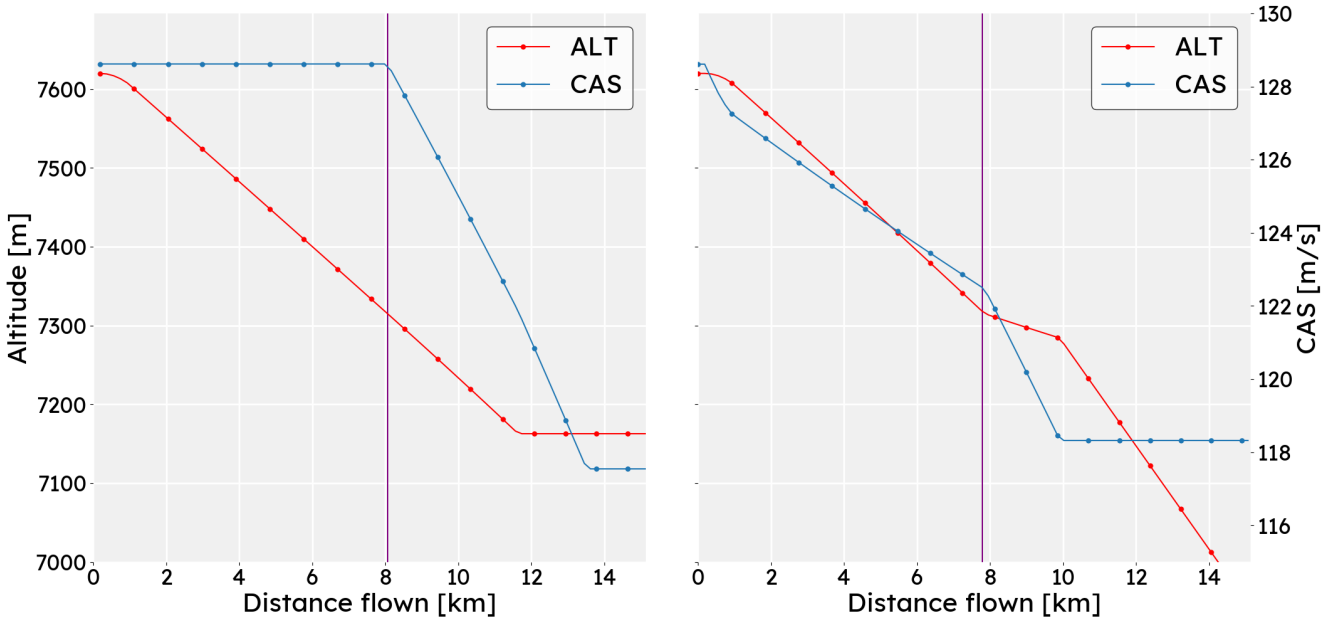

Figure 4.1: Flight profile comparison between using autothrottle and thrust setting

The left graph shows BlueSky in its old format, the right graph shows the flight using a idle thrust setting at the start. The altitude shows a changing altitude profile based on the speed setting. The nose is tilted up, essentially decreasing the altitude change, to ensure a quicker speed change. The moment the speed reaches the desired value of 230 knots, the altitude

drops quicker to maintain speed. Also note the decreasing speed seen in the new situation, even with a decreasing altitude.

The profile for the flight path angle shows similar behaviour, which can be seen in Figure 4.2. In this scenario, the constant path angle has been added to show what the behaviour would be. The essential difference is visible in the altitude figure. The ratio between altitude change and flown distance shows a constant angle, whereas the constant vertical speed can result in falling short in terms of distance to approach.
Furthermore, note that the path angle of the vertical speed case reduces as speed decreases. Which was expected, according to Equation (3.1).

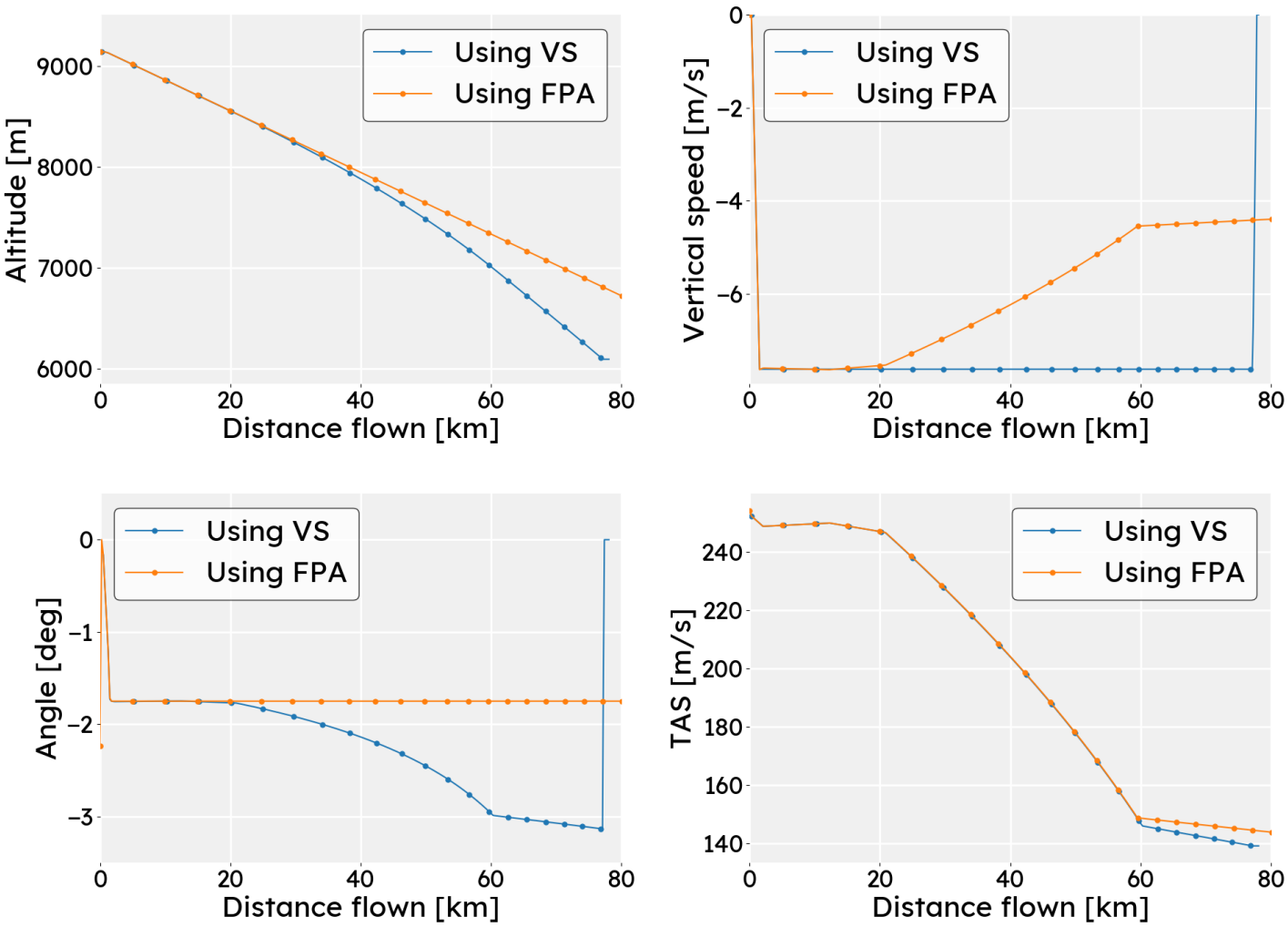

Figure 4.2: Flight profile comparison between using a constant vertical speed or flight path angle

4.2 Qualitative analysis

This analysis consisted of comparing the climb and descent flights as set up according to the explanation given in Section 3.6. A comparison was made using the altitude, speed, vertical speed and work output. Up first are the climb flights, after which the descent flights are

discussed. Note that the legend throughout these subsections describe the four different simulations described in Section 3.6. Throughout this section the following holds: 'new' stands for BlueSky using thrust settings and autothrottle, whereas 'old' indicates solely autothrottle.

4.2.1 Ascending flights

As stated earlier, the climb flights posed the most problems as they did not give a workable result. Given the measures taken explained in Section 3.6, several ascending flights have been set up to at least demonstrate the concept. The altitude profile can be seen in Figure 4.3.

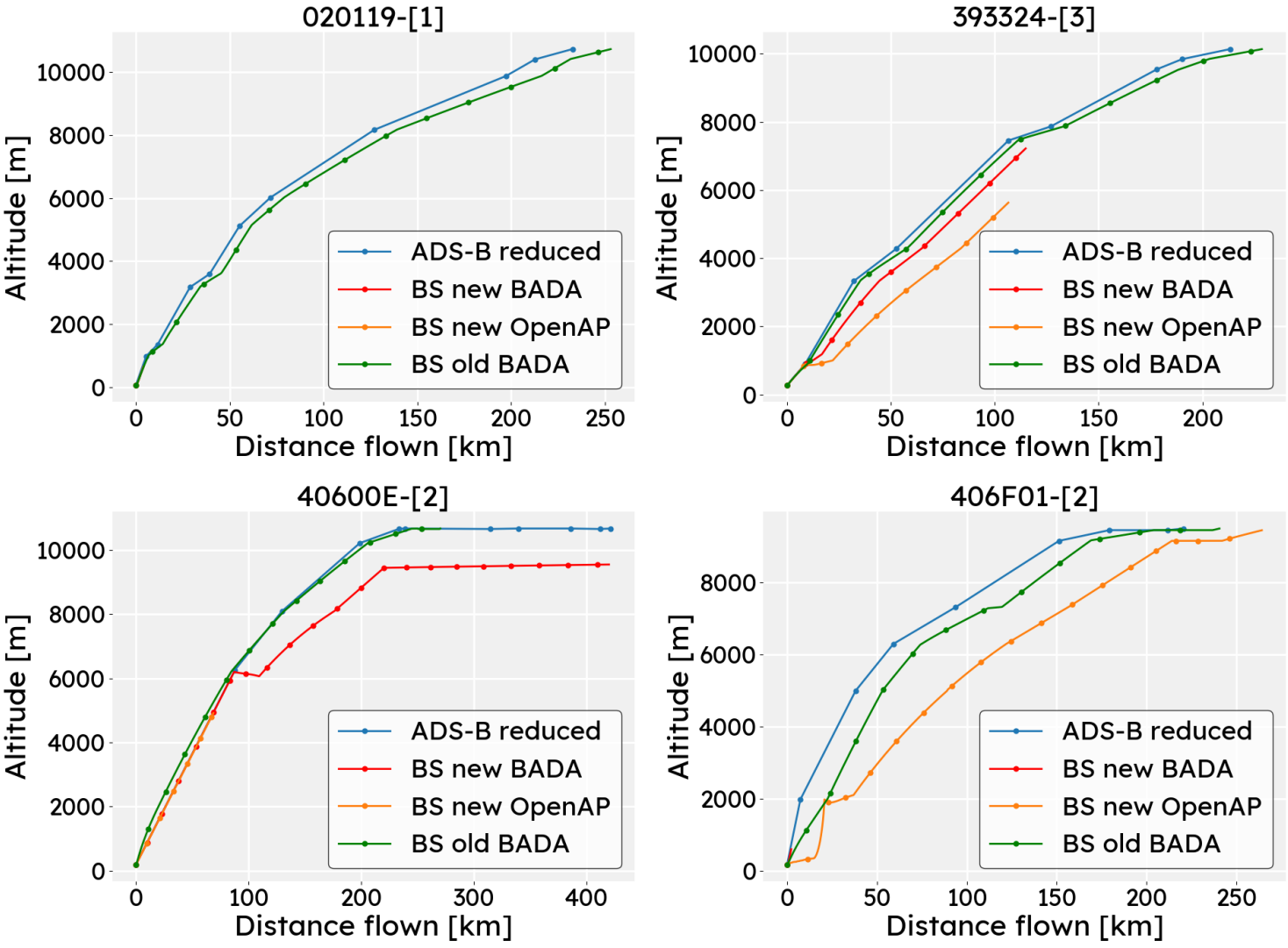

Figure 4.3: Altitude profile for four climb flights

The titles of the subplots represent the ICAO of the aircraft, with their corresponding number of flight for that day. It can be seen that for flight 020119-[1] neither of the performance models manage to properly simulate the aircraft. Flight 393324-[3] shows interesting behaviour, as both performance models are able to simulate the flight but end abruptly. For 40600E-[2] and 406F01-[2], OpenAP and BADA are not able to simulate, respectively.

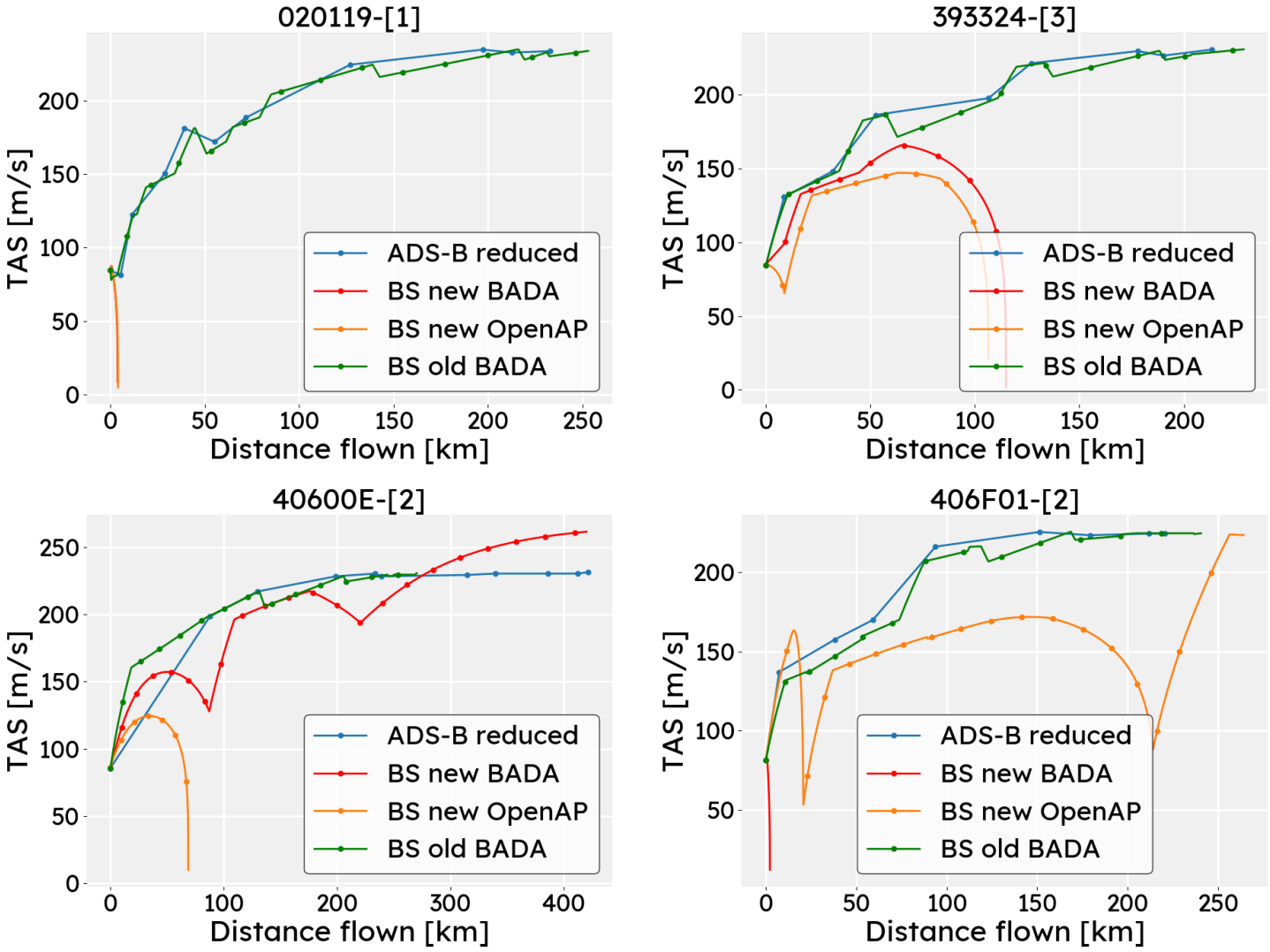

Figure 4.4: Speed profile for four climb flights

Reason for said behaviour can be seen in Figure 4.4. Both the OpenAP model as the BADA model show a speed decreasing to zero. Meaning, there is not sufficient thrust to keep the aircraft in the air. Similar behaviour can be observed for other flights which are not able to simulate the altitude profile: all respective speeds go to zero. Furthermore, it can be seen that flights which do actually manage to stay in the air, still experience large speed dips. Seen in both the 'BS new BADA' model for 40600E-[2] and the 'BS new OpenAP' model for 406F01-[2].

The vertical speed graph also shows the rate of altitude change seen in Figure 4.5. These flights have all been given mainly an altitude and path angle command. Meaning, the vertical speed becomes a function of the speed, as the speed is the free parameter. Therefore the vertical speed graphs are in line with what is seen in Figure 4.4. It can be observed that the default vertical speed used by 'BS old BADA' (1500 feet per minute) is a value in the same order of magnitude of the other simulations.

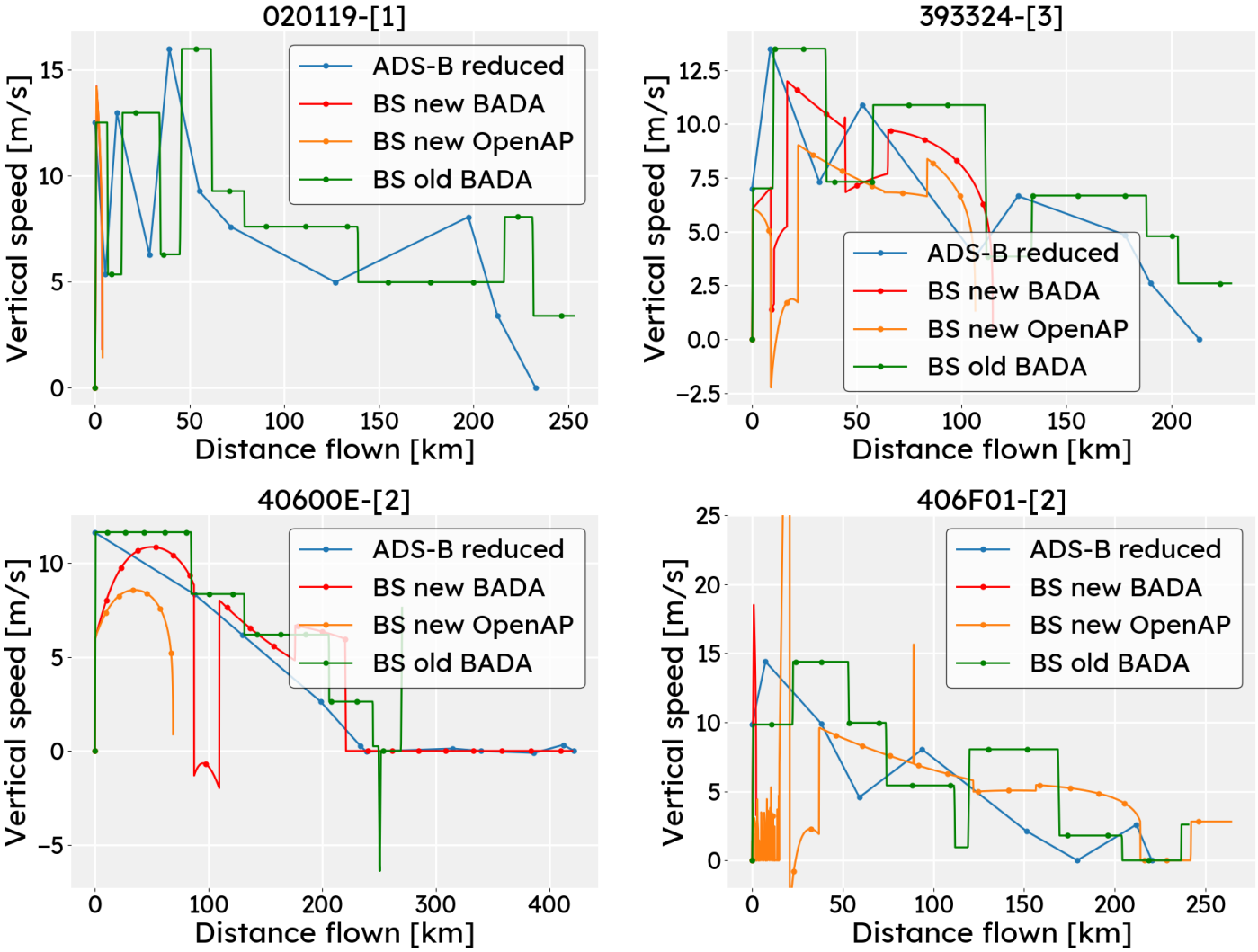

Figure 4.5: Vertical speed profile for four climb flights

Figure 4.6 shows the work performed by each of aircraft. Again, lines that end prematurely are flights that did not manage to climb to the final altitude. What can be observed is that the work performed for each of the flights is rather similar when it comes to the new and old model. Neither of the BlueSky models comes close to the work performed by the real aircraft data.

The total work for each flight and model is presented in Table 4.1. The error percentages are the absolute error relative to the ADS-B data. Results that represent incomplete flights are coloured red. It can be observed that the old situation has not been improved in terms of total work.

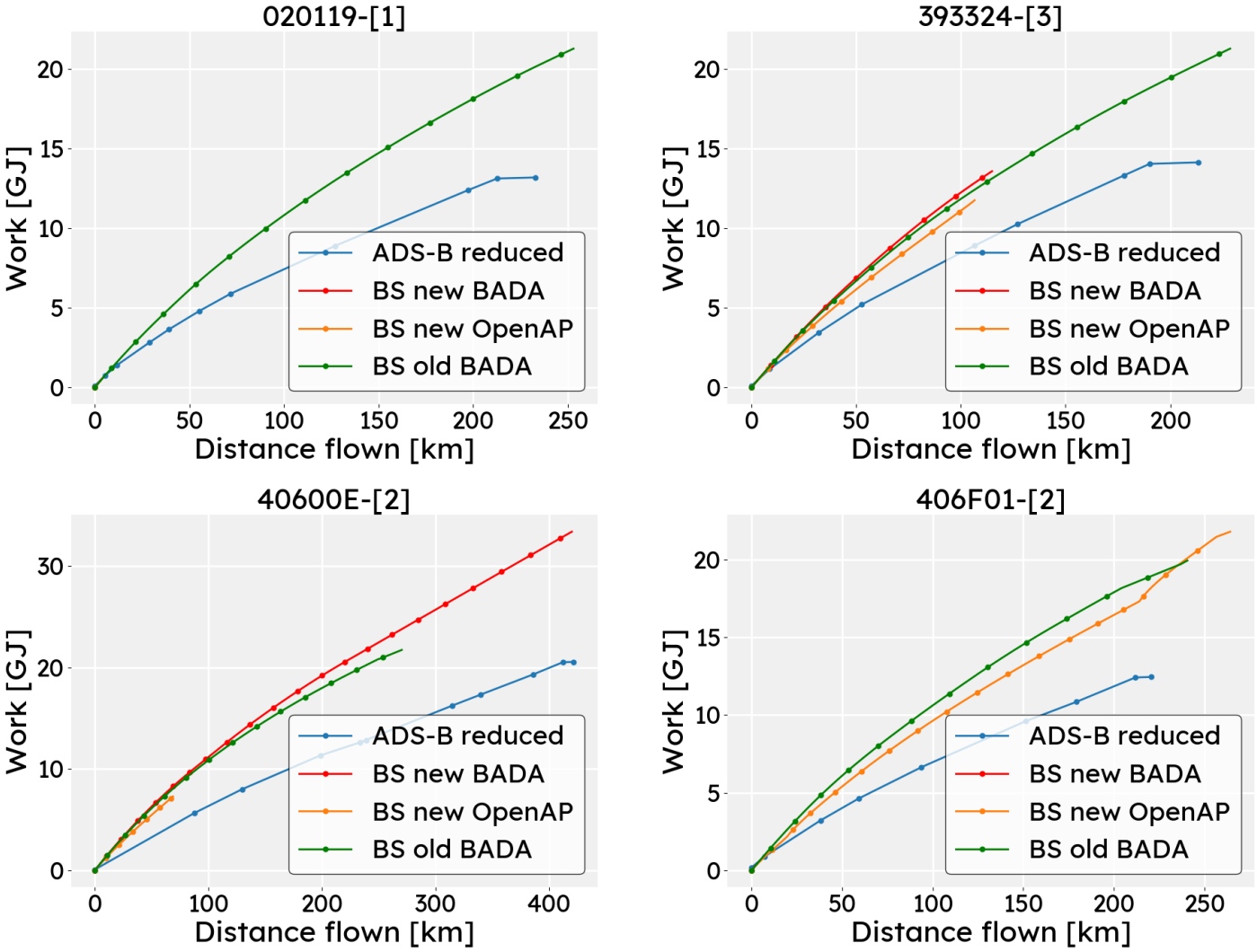

Figure 4.6: Work profile for four climb flights

Table 4.1: Total work quantities with their relative error for four climb flights

	Work [GJ]						
ICAO	ADS-B	Old BADA	err.	New BADA	err.	New OpenAP	err.
020119-[1]	13.2	21.3	61%	0.54	96%	0.57	96%
393324-[3]	14.1	21.3	51%	13.6	4%	11.7	17%
40600E-[2]	20.6	21.7	6%	33.4	60%	7.3	65%
406F01-[2]	12.4	19.9	60%	0.32	97%	21.8	75%
			45%		64%		63%

Next to the total work, the total distance flown is presented in Table 4.2. It can be observed that the old model performs even better than the total work shown in Table 4.1.

Table 4.2: Total distances flown with their relative error for four climb flights

	Distance flown [km]						
ICAO	ADS-B	Old BADA	err.	New BADA	err.	New OpenAP	err.
020119-[1]	232	253	9%	3.9	98%	4.1	98%
393324-[3]	213	228	7%	115	46%	107	50%
40600E-[2]	421	270	36%	421	0%	69	84%
406F01-[2]	220	240	9%	2.3	99%	264	20%
			15%		60%		63%

Lastly, the fuel data can be seen in Table 4.3. The fuel values for flights using the new model are significantly higher than in the old model. Note that autothrottle using OpenAP has been added for reference, but is not used in the simulations.

Table 4.3: Total fuel quantities for the climb profiles

	Total fuel [kg]			
	BADA		OpenAP	
ICAO	Old	New	Old	New
020119-[1]	1650	64	461	181
393324-[3]	1818	974	496	525
40600E-[2]	1613	2186	464	357
406F01-[2]	1502	32	420	916

4.2.2 Descending flights

The other comparison made is using descending flights. These have been more successful as descending is far more concerned with losing energy rather than gaining energy. The altitude profile for the descending flights can be seen in Figure 4.7.

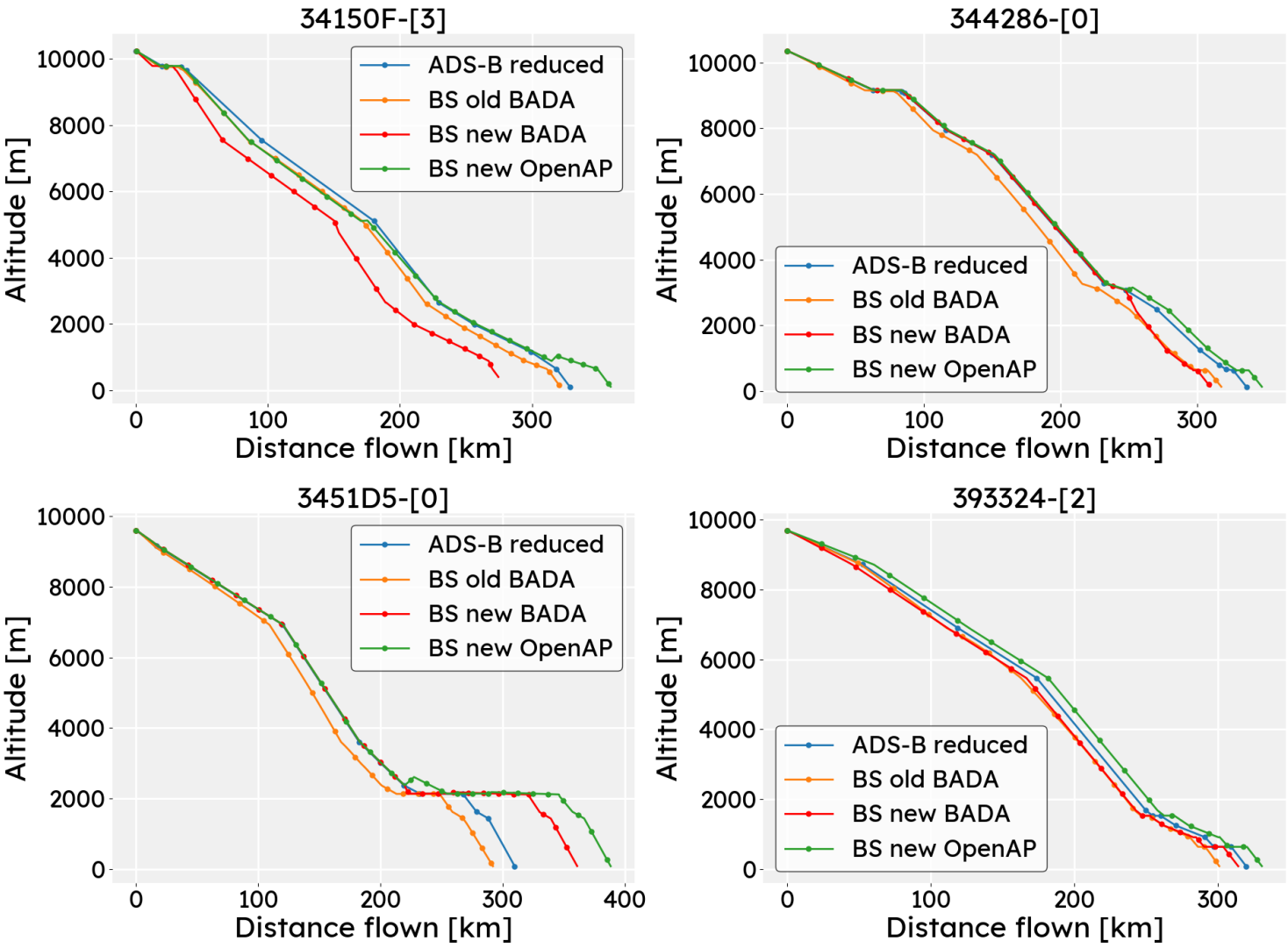

Figure 4.7: Altitude profile for four descent flights

The altitude profiles do not show too much deviating behaviour. For flight 3451D5-[0] stands out that the level segment at 2000 meter is significantly longer for the models using thrust settings than for autothrottle. Closer inspection indicated a shallow descent, which creates a challenging scenario. Whether it is a level or descending segment makes a difference in the outcome of the equations of motion. This boils down to ensuring a proper identification using the Fuzzy logic.

Results in the form of the speed can be seen in Figure 4.8. What stands out is that the BADA model using thrust settings experiences the largest decrease in speed. It even leads to a premature end of the flight for 34150F-[3].

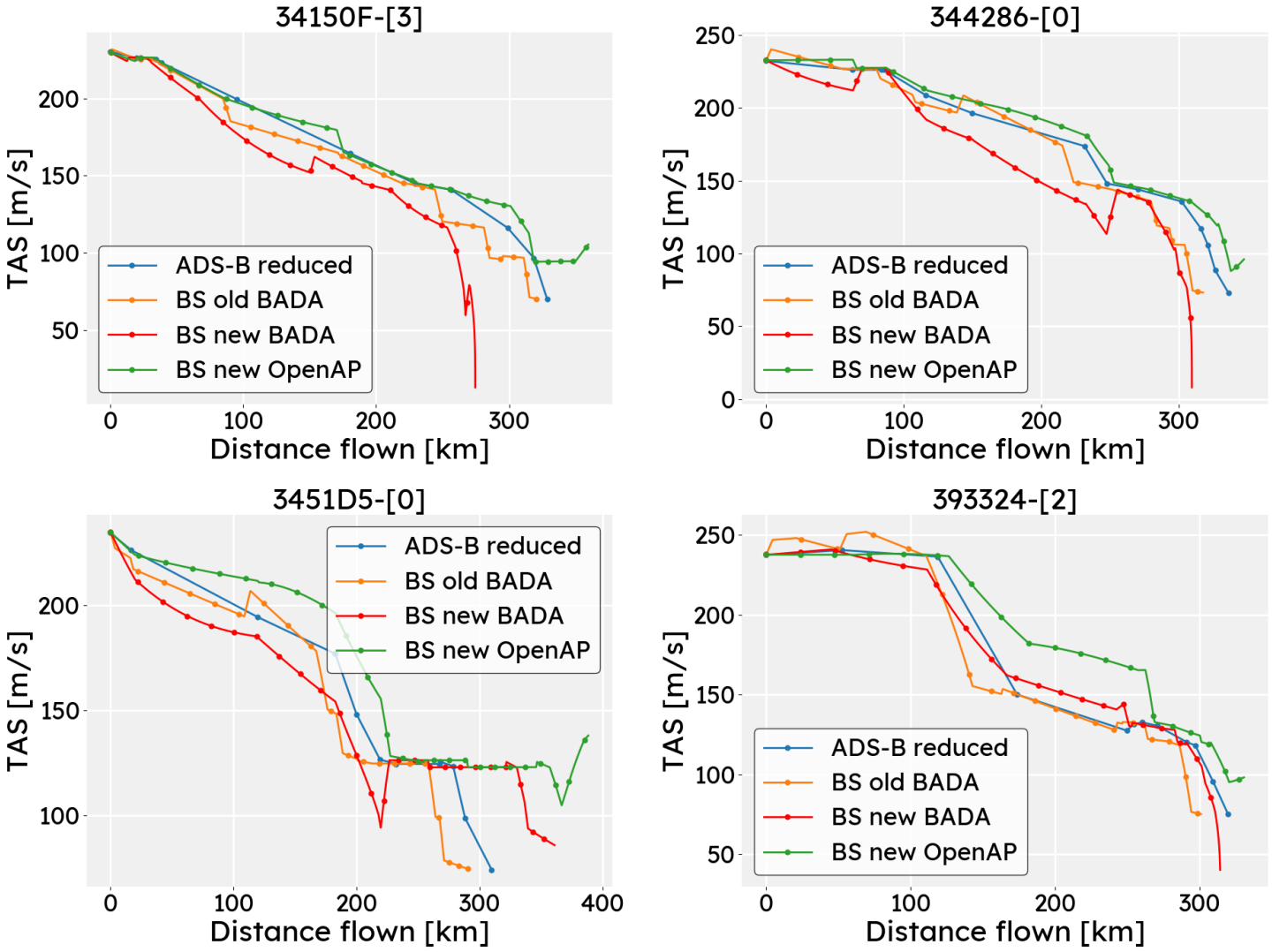

Figure 4.8: Speed profile for four descent flights

Figure 4.9 shows the vertical speed profile of the descending flights. Apart from a few spikes in the thrust setting models, the vertical speeds do not show particular behaviour. The thrust models follow the general ADS-B speed rather well.

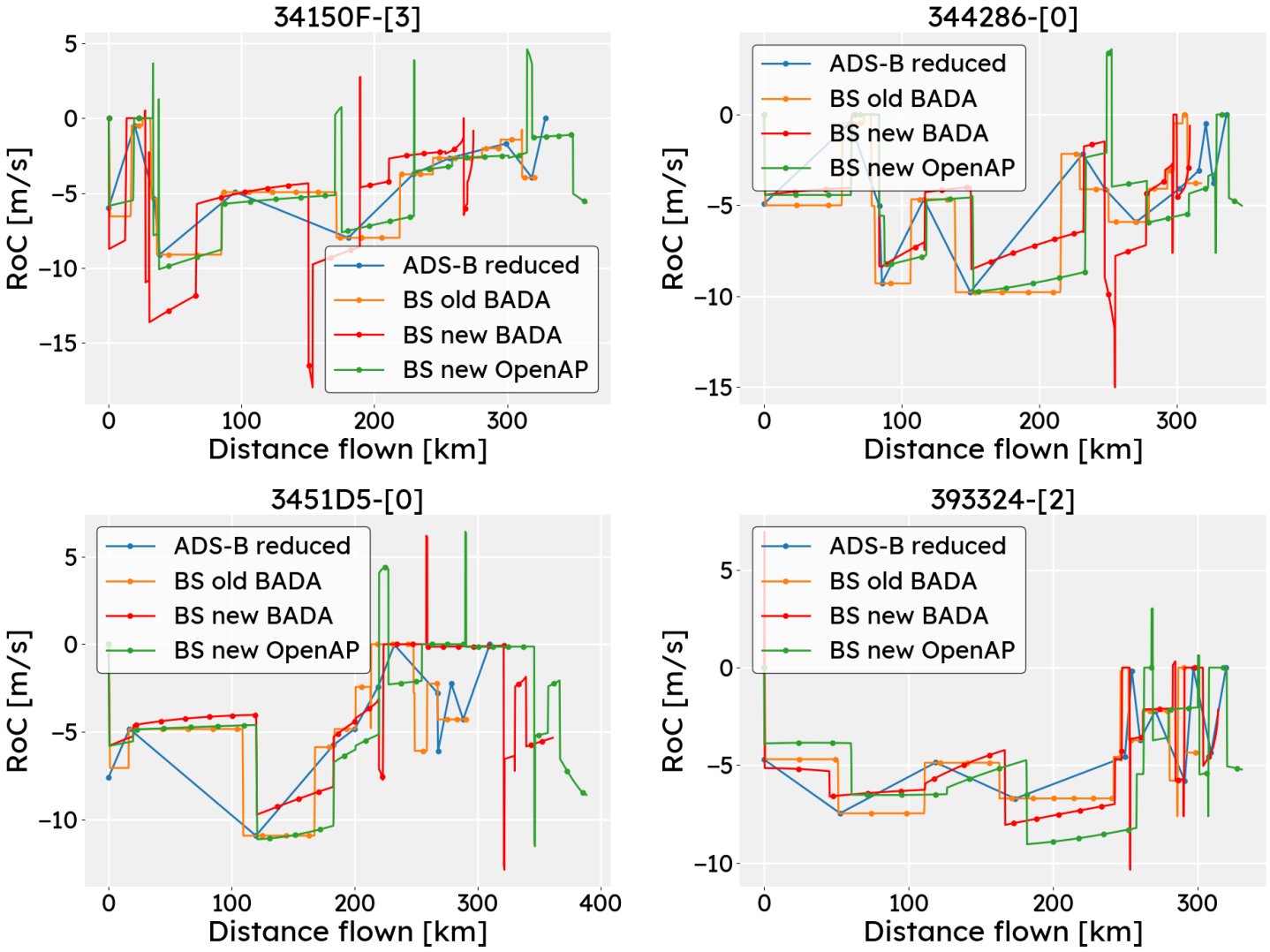

Figure 4.9: Vertical speed profile for four descent flights

Figure 4.10 shows the work performed by each of aircraft. Again, lines that end prematurely are flights that did not manage to climb to the final altitude. What can be observed is that the work performed for each of the flights is rather similar when it comes to the new and old model. Neither of the BlueSky models comes close to the work performed by the real aircraft data.

The total work for each flight and model is presented in Table 4.4. The error percentages are the absolute error relative to the ADS-B data. Results that represent incomplete flights are coloured red. It can be observed that for both performance models, the new situation has performed less work.

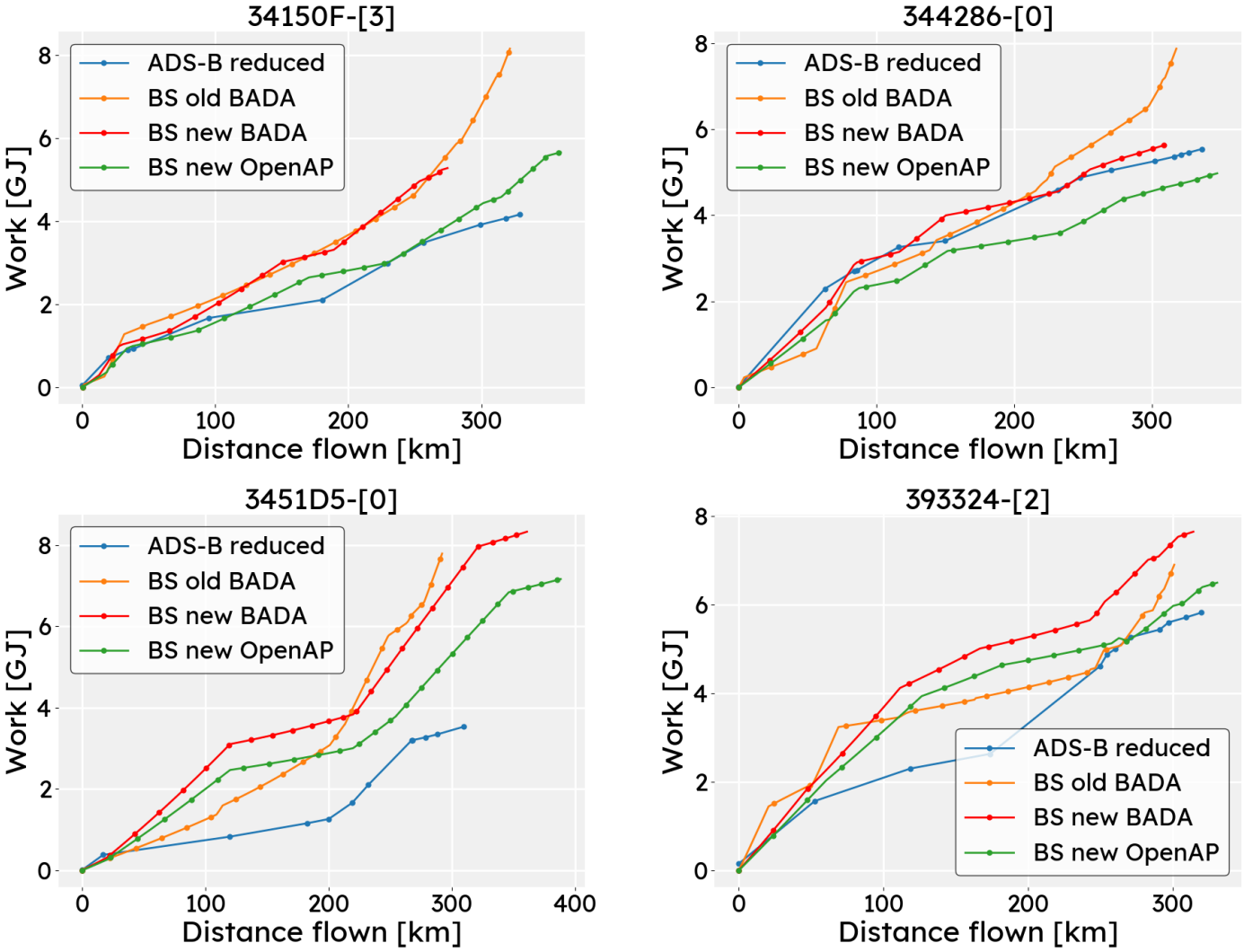

Figure 4.10: Work profile for four descent flights

Table 4.4: Total work quantities with their relative error for four descent flights

	Work [GJ]						
ICAO	ADS-B	Old BADA	err.	New BADA	err.	New OpenAP	err.
34150F-[3]	4.2	8.2	96%	5.2	27%	5.7	36%
344286-[0]	5.5	7.9	42%	5.6	2%	5.0	10%
3451D5-[0]	3.5	7.8	120%	8.3	136%	7.2	102%
393324-[2]	5.8	6.9	19%	7.6	31%	6.5	11%
			70%		49%		35%

Table 4.5 shows the total distance flown. It can be observed that both performance models in the new model have a larger error than in the old model.

Table 4.5: Total distances flown with their relative error for four descent flights

	Work [GJ]						
ICAO	ADS-B	Old BADA	err.	New BADA	err.	New OpenAP	err.
34150F-[3]	328	321	2%	274	16%	359	9%
344286-[0]	336	317	5%	310	8%	347	3%
3451D5-[0]	309	292	6%	361	16%	388	25%
393324-[2]	320	301	6%	314	2%	330	3%
			5%		11%		10%

To further examine the characteristics of the descent flights, Table 4.6 shows the total fuel quantities for these flights. These quantities show remarkable values: using thrust settings in no case provides a reduction in fuel consumption. Note that the autothrottle situation for OpenAP has been included here for reference as well.

Table 4.6: Total fuel quantities for the descent profiles

	Total fuel [kg]			
	BADA		OpenAP	
ICAO	Old	New	Old	New
34150F-[3]	445	1170	600	602
344286-[0]	327	771	505	581
3451D5-[0]	500	1035	553	738
393324-[2]	426	921	574	651

4.3 Sensitivity analysis

As discussed in Section 3.7, both available thrust and mass were varied for four different flights, for altitude, speed and vertical speed. The varying of thrust can be seen in Section 4.3.1, and the varying of mass can be seen in Section 4.3.2.

4.3.1 Sensitivity to thrust

Varying the maximum available thrust resulted in the altitude graph seen in Figure 4.11.

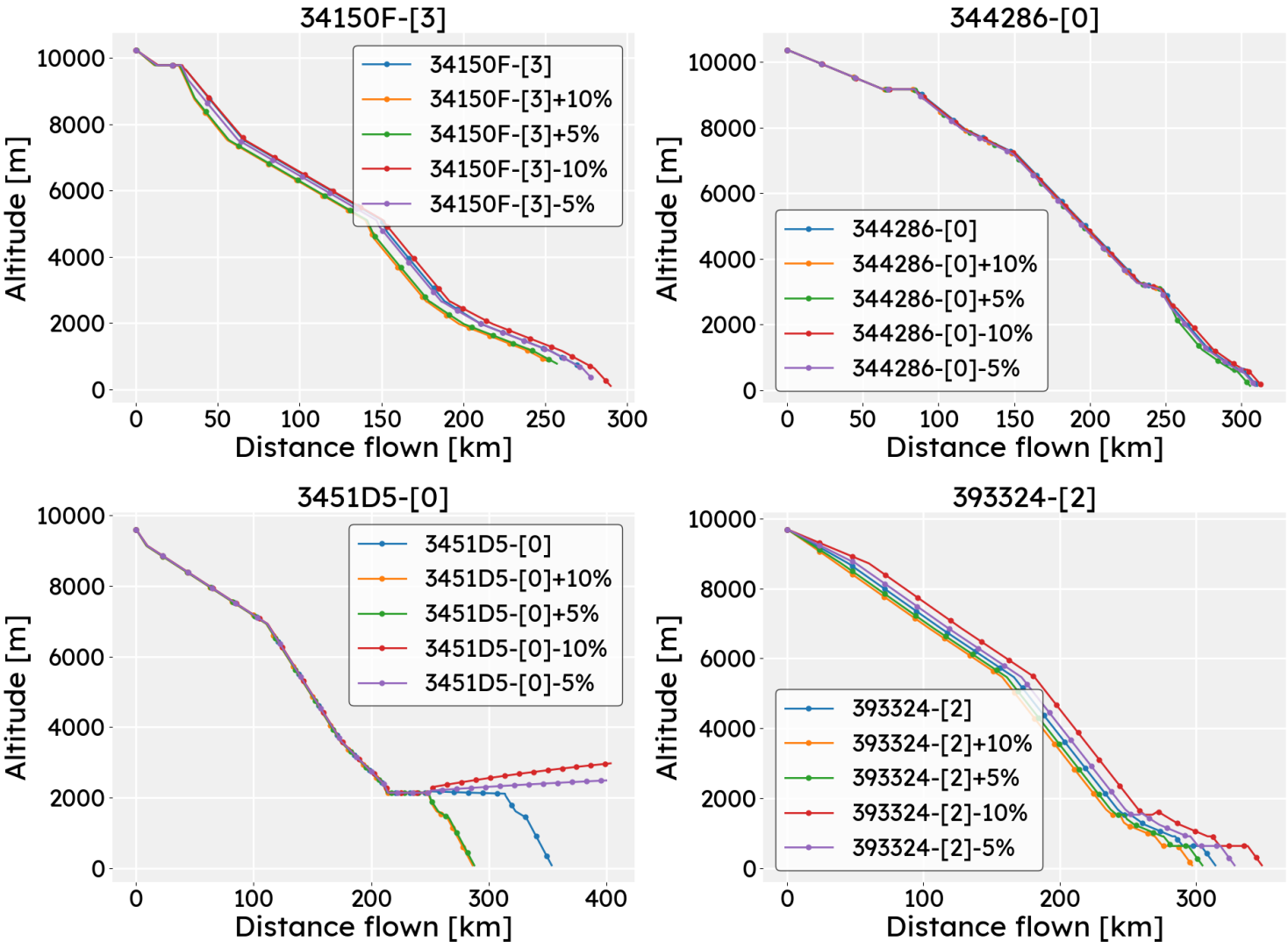

Figure 4.11: Sensitivity analysis regarding altitude with varying available thrust

Both flights 34150F-[3], 344286-[0] and 393324-[2] show varying altitude profiles without any anomalies. However, 3451D5-[0] shows an interesting phenomenon. Flight with a larger available thrust, and therefore smaller thrust settings, show a *full* flight profile whereas the other flights remain at a certain altitude. This indicates that the 'AT' command is most likely not fulfilled, therefore remaining stuck at a certain altitude. It can even be observed that the two flights with decreased available thrust (-5% & -10%) gain altitude after a while.

This analysis can be further expanded by looking at the distance flown, which can be seen in Table 4.7. Note that values in red indicate premature ended flights. What becomes clear from this graph is that for both 3451D5-[0] and 393324-[2] distances significantly differ when changing the maximum available thrust. It cannot be said that these distances scale with said parameter, as that depends on the aforementioned 'AT' situation.

Table 4.7: Distance flown for the flight regarding sensitivity to available thrust

	Distance flown [km]				
ICAO	-10%	-5%	+0%	+5%	+10%
34150F-[3]	277	289	272	257	251
344286-[0]	313	309	310	305	245
3451D5-[0]	403	399	353	287	286
393324-[2]	348	328	314	304	297

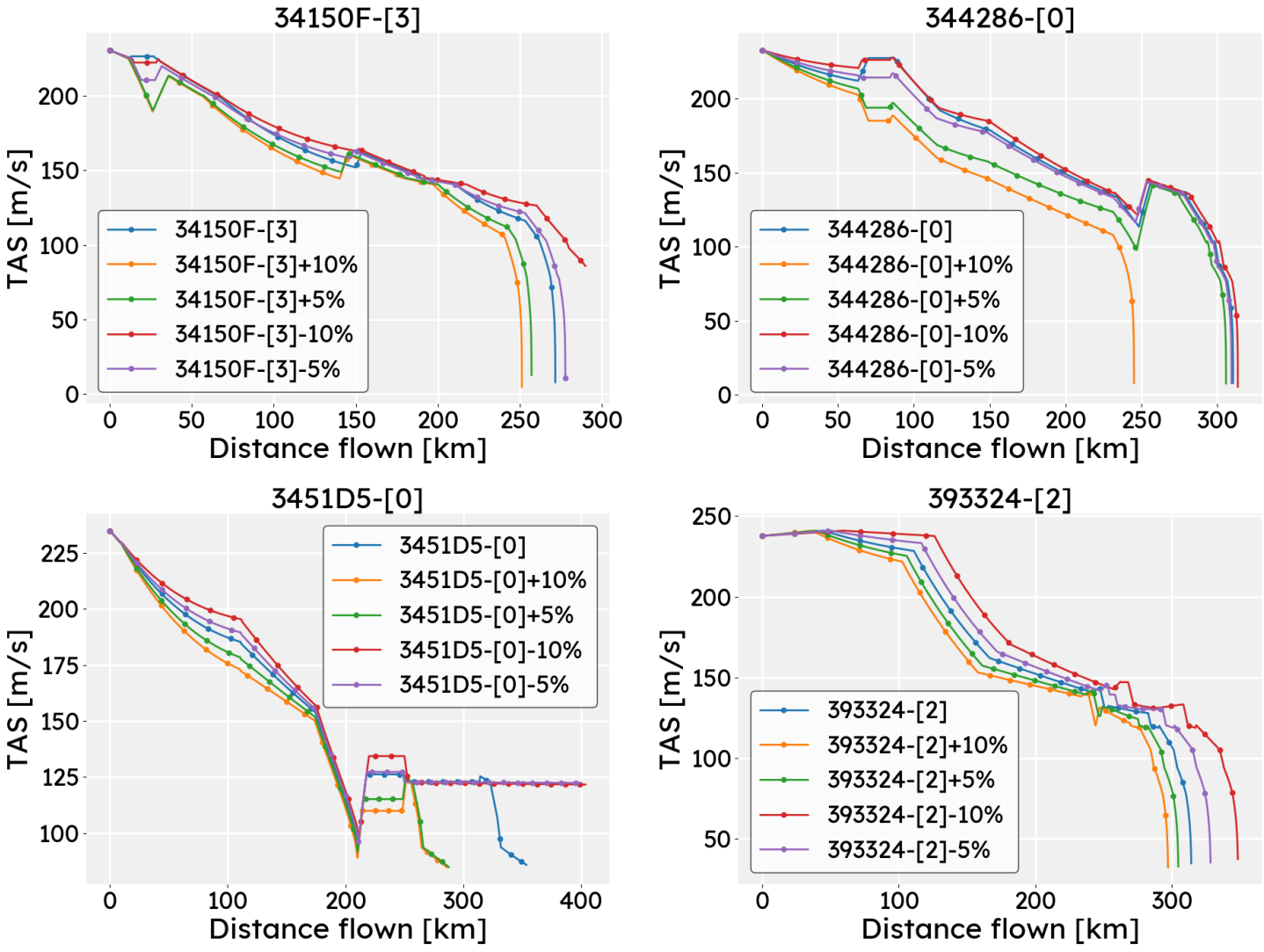

Figure 4.12: Sensitivity analysis regarding speed with varying available thrust

Figure 4.12 shows a more clear what happens in each of the profiles of Figure 4.11. If looking at 3451D5-[0] again, it can be observed that the speeds do coincide. The major level part flown by 3451D5-[0] is a level part with the autothrottle on, after which a THR and SPD command are given again. The next step is a lower altitude, which clearly, shown by Figure 4.11 is not achieved by some.
Further speed profiles in Figure 4.12 show behaviour as expected. Flights with higher available thrust and thus lower thrust settings cause quicker diminishing speeds. This even causes flight 344286-[0] to fall from the sky as the setting is too low.

Observations done are further confirmed by Figure 4.13. There is a constant exchange between vertical speed and acceleration, following the formula given in Equation (3.12).

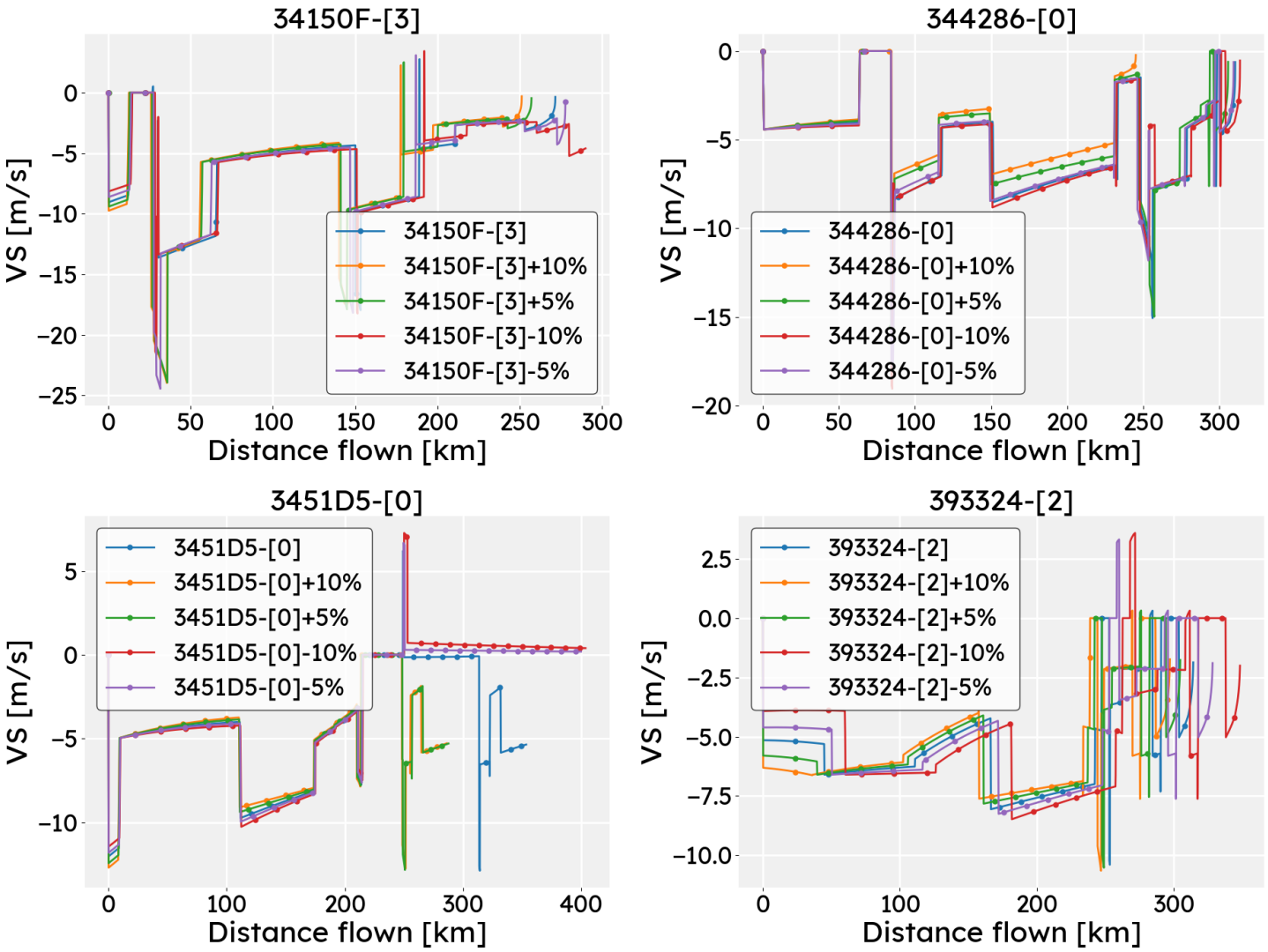

Figure 4.13: Sensitivity analysis regarding vertical speed with varying available thrust

4.3.2 Sensitivity to mass

The second analysis was concerned with a varying mass. In Figure 4.14, solely the bottom left graph shows deviating behaviour compared to the other three. The flights with increased mass

manage to continue their flight path, whereas the lighter aircraft stay at a certain altitude.

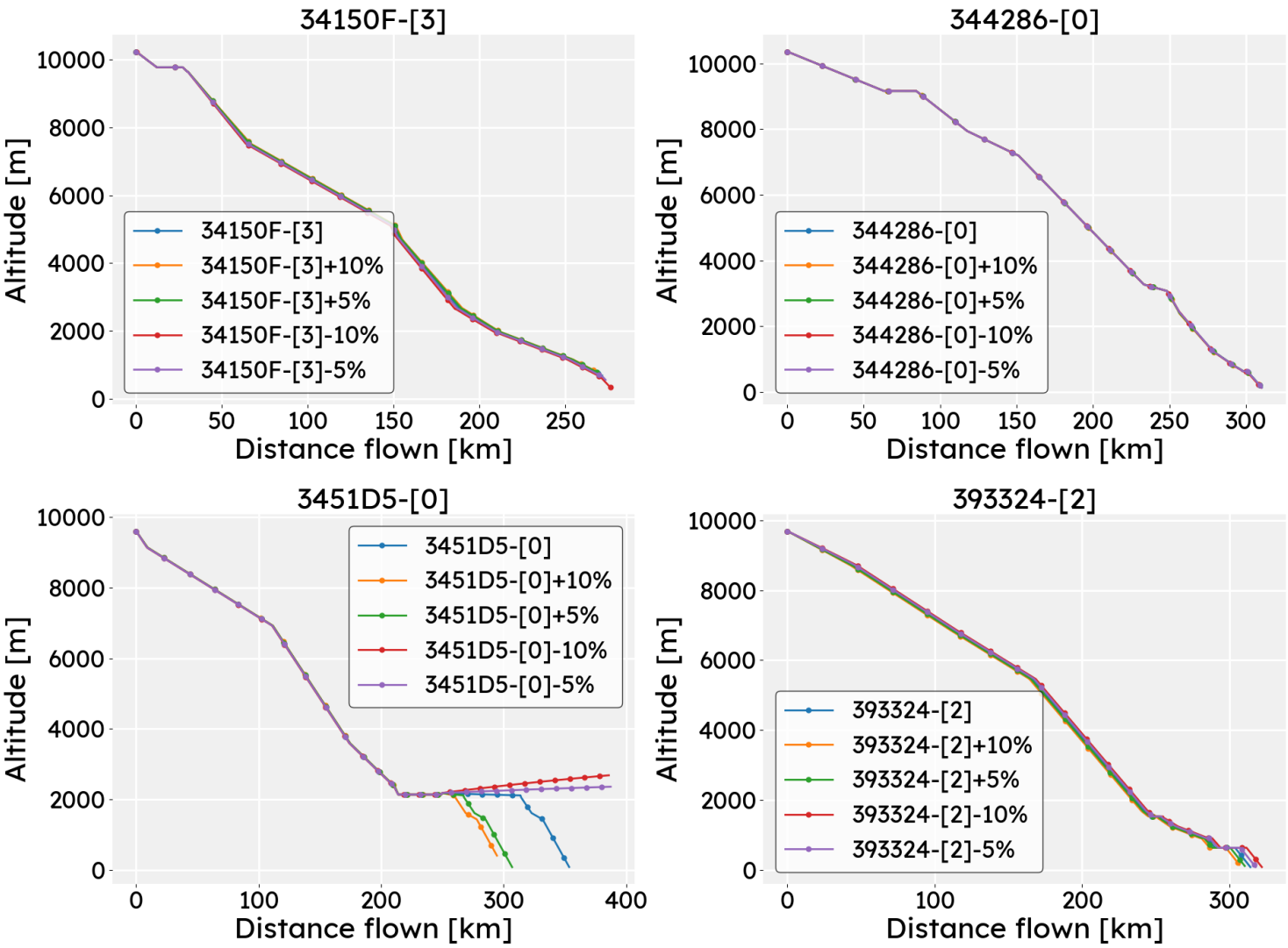

Figure 4.14: Sensitivity analysis regarding altitude with varying mass

Quantifying the distances is done in Table 4.8. Note that here no flight is regarded as ended prematurely, therefore all values are considered to be representative. The entire table shows rather insensitive behaviour to mass changes, with the exception of 3451D5-[0]. The same phenomenon occurs as discussed in Section 4.3.1.

Table 4.8: Distance flown for the flight regarding sensitivity to mass

	Distance flown [km]				
ICAO	-10%	-5%	+0%	+5%	+10%
34150F-[3]	276	273	271	269	266
344286-[0]	309	310	310	309	308
3451D5-[0]	385	387	353	306	294
393324-[2]	321	317	314	310	306

Figure 4.15 follows a similar observation as for the altitude. Speeds drop for 3451D5-[0] as it approaches ground level. The other graphs show no significant relative phenomena.

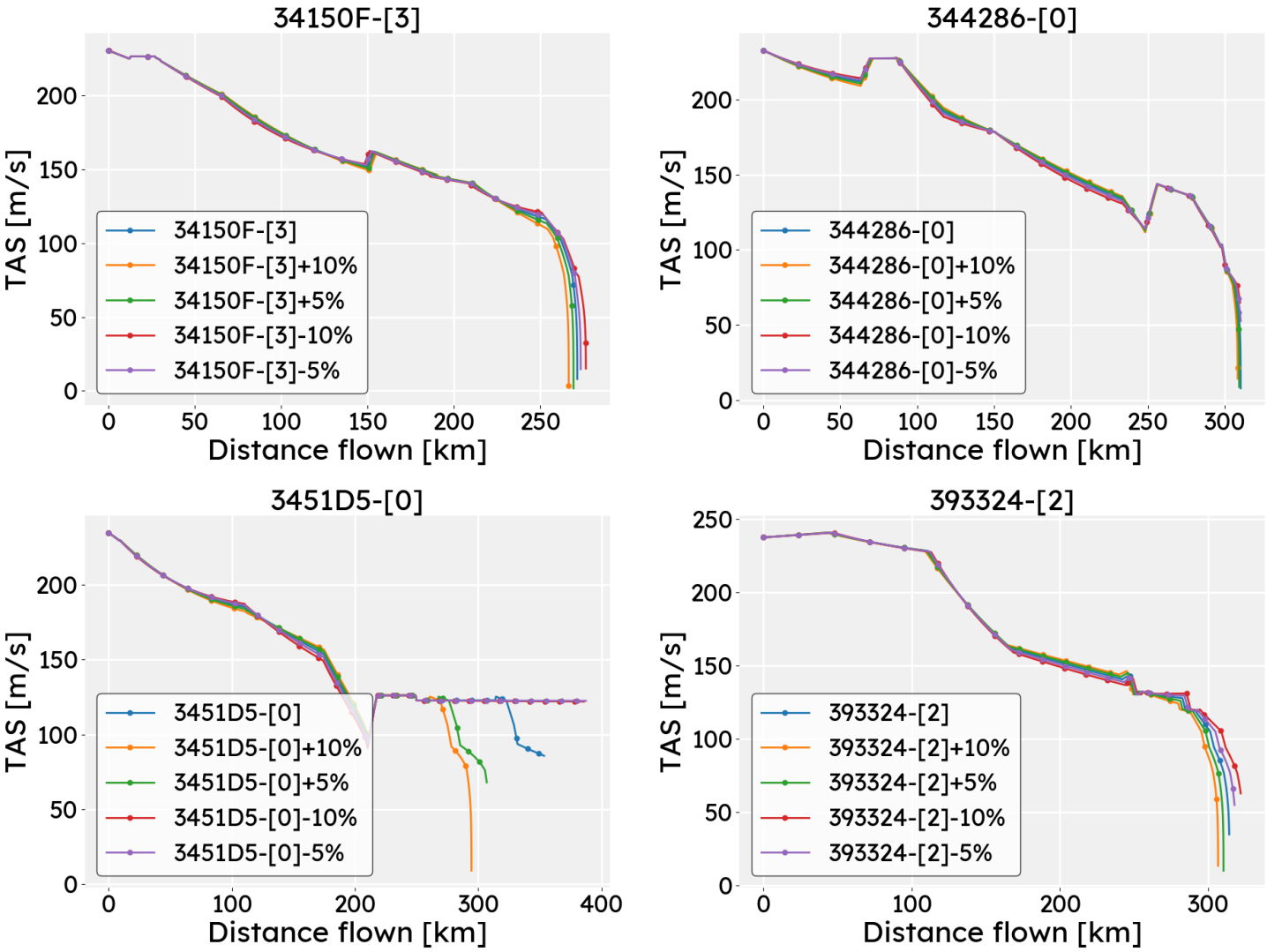

Figure 4.15: Sensitivity analysis regarding speed with varying mass

Following from the previous two graphs, the vertical speed profile seen in Figure 4.16 neither contains any deviations from what has been observed already. The vertical profile of 345D5-[0] shows that for some flights the descent is continued, whereas for others it is not.

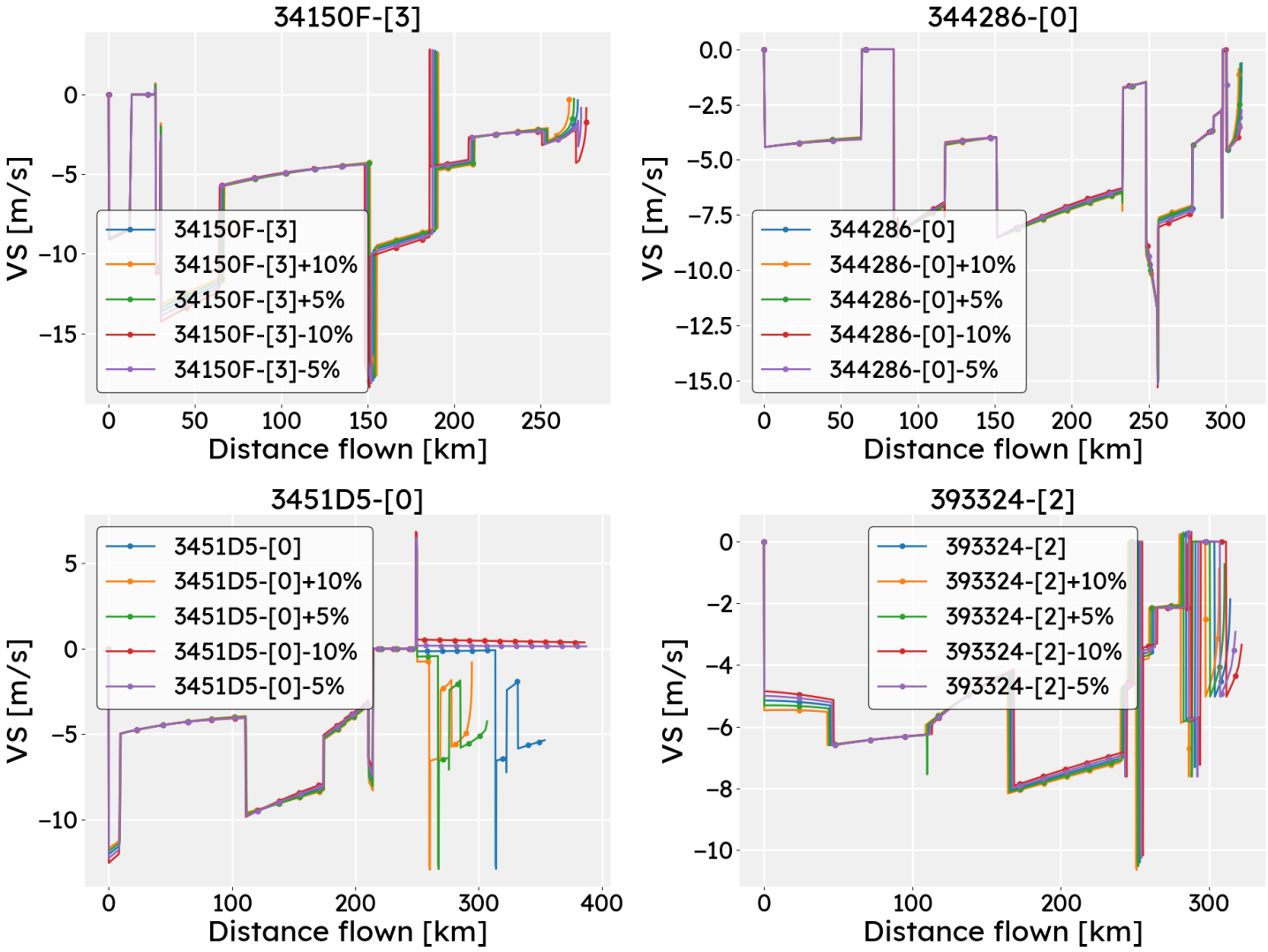

Figure 4.16: Sensitivity analysis regarding vertical speed with varying mass

5 Discussion

This chapter will discuss the results of Chapter 4. It will treat observed behaviour and come up with possible explanations, but will also separately treat the research as a whole.

5.1 Regarding results

The results shown in the previous sections have had an observational explanation, and what follows is a explanation of possible causes. The proof of concept is discussed, after which the flight profiles follow, and is concluded with results from the sensitivity analysis.

Proof of concept

The basic principles are demonstrated using Figure 4.1 and Figure 4.2. Figure 4.2 requires little analysis as behaviour is as expected. What can be seen in Figure 4.1 is that, in the case of a decreasing altitude, the speed also decreases. Thrust has been set to idle, and the altitude is decreased with the standard value of 1500 feet per minute. This means that in the case of an idle thrust, and the maximum descent rate *in this situation*, speed in any case will decrease. This behaviour is further validated by FlightGear, although not to the extent of correct slope.

Climb Flights

Some climb flights do in fact manage to climb to certain cruise altitudes, but they clearly struggle maintaining speed. This is indicated by the large dips in the speed graphs seen in Figure 4.4. It is difficult to appoint either performance model as worse performing. Both BADA and OpenAP struggle with their respective flight profiles as seen in Figure 4.3. Striking in each of these results is the role of the speed. The speed repeatedly tends to drop to stall speeds, indicating a lack of thrust.

Figure 4.5 shows that regular vertical speeds are achieved for all flights, therefore ruling out the option of unrealistic rates of climb. Moreover, even a thrust setting of 1 on all segments is often not enough to obtain a full flight profile. This indicates that the drag and thrust calculation is off, which is further supported by findings in Section 5.2. But, further adding to the previous finding, Figure 4.6 shows that a similar amount of work is performed for all three BlueSky models. However, regardless of equal amounts of work, the new model is not able to sustain flight. Therefore, with similar vertical rates and performed work, it is highly likely that Equation (3.4) contains an incorrect drag value in these situations.

Descent flights

The descent graphs seen in Figure 4.7 show an interesting contrast with the climb flights in Figure 4.3. The difference between descent and climb performance is gaining and losing energy. Therefore, deceleration and altitude decrease is a result of drag with minimal thrust.

Moreover, the phenomena seen in Figure 4.4 regarding the speed drops is far less common in Figure 4.8. Speed drops in Figure 4.8 are mainly BADA based. The most likely explanation is the earlier mentioned discrepancy in using OpenAP and BADA.

Work done showed contrasting results, seen in Figure 4.10. Both Table 4.4 and Table 4.5 show that the average error in terms of work is worse for the old situation. On the other hand, the distance flown shows better results for the old situation. Therefore, the total amount of work can be seen as better estimated by the new model, at the cost of an accurate distance.

Sensitivity analysis

Lastly, and this topic will be touched upon in Section 5.2 as well, are the results from the sensitivity analysis.

In Figure 4.11, flight 3451D5-[0] showed odd behaviour when it came to the old model. Further inspection of the command structure showed that the flight was classified as descending in this seemingly level part. This led to ATALT commands in the code, but ATSPD commands would have been appropriate here. It might not significantly have influenced this flight, but it does show the significance of correct functioning Fuzzy logic.

The sensitivity analysis also showed a rather insensitive behaviour to changing of the mass Figure 4.14. This is especially helpful, as it indicates that an accurate mass model is not as significant as the thrust setting. Still, flights can differ completely in behaviour due to a mass change as seen in Figure 4.14, but these can be regarded as exception. Other flights from Figure 4.14 show almost spot on profiles, indicating little sensitivity. However, the results are from only four flights, which is limited to provide sufficient justification. Note that this sensitivity analysis is only applicable for descent flights. Thrust settings for climb flights have been changed so significantly that no conclusion can be drawn regarding these profiles.

5.2 Regarding the research

Research requires one to be critical on steps taken. This means investigating how certain choices have impacted performance, what could possible be done differently and how would that have influenced the research.

Preprocessing

The calculation of the FPA and execution of RDP has been crucial for the Fuzzy logic to determine phases, whereas the Fuzzy logic and energy method subsequently were essential for the calculation of phase thrust. These steps following each other cause simplifications and thus come to a result that is off from reality to a certain extent. Adding this uncertainty on top of the fact that the implemented logic in BlueSky is also an experiment, it has been difficult to determine what parameters have had the largest influence in creating a somewhat untruthful model.

When considering Fuzzy logic, it has been shown (Section 4.2.2) that it has had influence on the profiles as well. The discussion of the sensitivity results has shown how this turned out. It has been underestimated how important it is what probability functions have to be set up for this. That has not been due to a lack of time, it is simply very difficult to determine as each research using Fuzzy logic is different. Finding sources that justify certain functions are thin spread and often do not provide a full answer.

One way to avoid situations in which these simplifications are stacked is the separate validation of both models. Meaning, that the input file is representative for real aircraft behaviour. On the other hand, more extensive aircraft data could also be used, although data like thrust settings are incredibly hard to come by.

BlueSky and performance models

The second challenge has been the way BlueSky, and its performance models, have been coded into the structure. The first signs of deviating behaviour within OpenAP became apparent after running several simple scenarios, with no assigned thrust setting. When climbing and accelerating, the net thrust would sometimes exceeded the maximum available thrust. The issue causing this, is the way OpenAP and BADA work. Due to the kinematic approach, thrust is an result rather than an input. This is further demonstrated by how acceleration and thrust are determined. Acceleration is calculated using Equation (5.1) and the net thrust is the result of Equation (5.2).

$$ax_{max} = \frac{T_{max} - D}{m} \qquad (5.1)$$

$$T_{net} = D + m \cdot ax_{max} \qquad (5.2)$$

Surprising in Equation (5.1) is the use of maximum thrust, and the use of net thrust in Equation (5.2). This does explain why the maximum thrust is exceeded from time to time. The acceleration is the only value where this odd method has influence, as speeds and altitudes are calculated using the kinematic model using its defined limits. However, when using a thrust setting, and using that as a fraction of the maximum available thrust, the net thrust will often be too small.
Moreover, during the autothrottle phases (classified as LC segments), the logic would resort back to specific autothrottle logic. It has been adjusted to use the net thrust, in order to refrain from using the acceleration as in Equation (5.1). However, it is not possible to tell if more logic changes would be necessary. The combination of the expanded logic using current logic is an aspect which requires more looking into to be sure of a similar method in both cases.

The result of all this was to change to BADA as performance model, as BADA seemed not to have this behaviour. However, this is were the discrepancy between preprocessing with OpenAP and simulating with BADA was created. This has been illustrated in Table 5.1. This table demonstrates that for four different altitudes and speeds BADA and OpenAP on average differ 158% in value regarding maximum available thrust. This value is on average 118% for drag, respectively.

Table 5.1: Absolute and percentual differences between BADA and OpenAP

		Maximum available thrust (T_{avail}) [N]		Drag (D) [N]			Percentual of OpenAP [-]	
Flight level [ft]	CAS [kts]	OpenAP	BADA	OpenAP	BADA		Maximum thrust	Drag
FL300	300	41,473	67,769	43387	46901		163%	108%
FL200	200	57,914	92,967	32524	35252		161%	108%
FL100	200	73,830	119,911	32561	35291		162%	108%
FL050	150	90,878	134,037	36378	53923		147%	148%
						Avg.	158%	118%

Lastly, next to the fact that these models had differences in calculations, also surprising behaviour was observed in the calculation of the fuel quantities. In every case of using a thrust setting, and therefore an optimal use of idle setting, the consumption in Table 4.6 and Table 4.3 was significantly higher. Although the fuel model is not questioned and it has not been possible to investigate this claim, it did increase scepticism to believe that the performance models were not outfitted to handle thrust settings.

Other climb and descent parameters

Thirdly, it can be argued that calculating thrust for especially climb and descent is more complicated than assumed in this research. Aircraft not only use thrust as method to control the descent, but also all kinds of control surfaces.

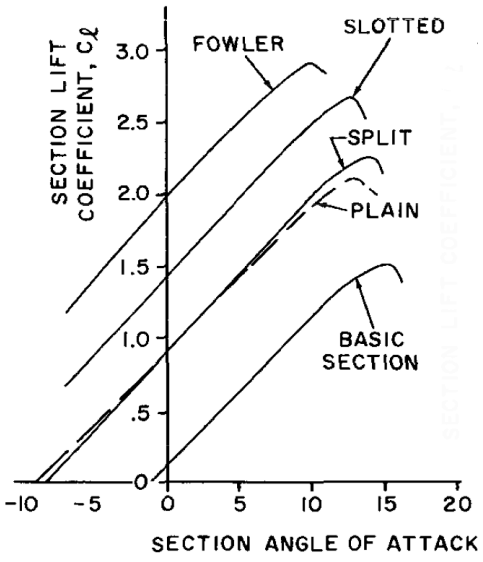

Figure 5.1: Impact of high lift devices on lift coefficient

Figure 5.1 illustrates that a change configuration can have significant influence on the aircraft its performance [20]. Especially speed is influenced by this a change in lift coefficient, making it able for the aircraft to attain lower speeds. In this research, it is impossible to tell what each aircraft did at what stage, there it was simply left out. Knowing, or at least taking lift devices into account, can already have a significant impact on the limits of the aircraft and thus its performance.

Small number of flights

The entire validation has been carried out on a small number of flights. This was chosen to to demonstrate the concept and logic, but does not prove the robustness of the model. Testing for a large number of aircraft is essential and proves that it works for many scenarios. Section 6.2 provides ideas on how this can be performed.

VNAV

In the end, the research has not been able to address the concept of VNAV which was caused by several reasons.

Early on in the literature it became clear that VNAV used waypoints. This meant defining latitude and longitude positions. The drawback of this is that BlueSky would still be very much helped in what it would have to do. Way points could be further specified by imposing speed or altitude constraints. This could be done in combination with the data that was used as input, but missing a way point would be very likely with the concept version of the input data. This would cause a far more hectic flight profile which would provide less basis to draw conclusions on. Therefore, early in the methodology it was chosen to make the basic logic without VNAV work first. As it turned out that the basic logic already had some major challenges, as described in the Section 3.6, VNAV was regarded to be too far of a stretch to improve as well.

Changing the steepness as defined in Section 3.2.2 would be a first step in making a more true VNAV. The current steepness is defined using a basic logic of 3000 feet per 10 nautical miles. This is based on a speed of 300 knots. Generally, VNAV tries to achieve a 2 degree flight path angle, however this can be exceeded up to 3000 feet per minute.[1] This top of descent point is determined using the speed, altitude difference and distance between waypoints. As seen in the results, Figure 4.8 and Figure 4.4 the speed profile showed too much unstable behaviour. Ensuring a more stable speed profile would create the opportunity to make VNAV dependent on the speed rather than a fixed value. Whenever the aircraft would be high on profile and struggling to follow the path, drawing back the thrust would be essential. Validation would be possible by comparing the position of the ToD points between flights.

The discussion has given an analysis of the results, but also looked at the research as a whole in a critical way. The latter was necessary to conduct before rather then after the conclusion, as it influenced the conclusion as well. Based on the results and analysis of the research, the conclusion is presented in Chapter 6.

[1] https://infiniteflight.com/guide/flying-guide/descent-to-landing/vertical-navigation-(vnav) [Cited: 17-02-2022]

6

Conclusion & Future work

This chapter concludes the research. Within the conclusion the key take-aways are presented which have been derived from the work performed. The conclusion is written to be understandable on its own, though its recommendations for future work make use of terms used within the research.

6.1 Conclusions

The research started off with an introduction to the topic, where its main question was formulated as follows:

Research question
”How can BlueSky and its performance models be expanded to ensure that the simulator itself, during climb and descent phases, is able to simulate representative flight behaviour while refraining from using full autopilot logic?”

A methodology was set up to answer this main question. This methodology entailed transforming real aircraft data to a new kind of input for BlueSky. This new kind of input was based on a expanded logic containing new functionality implemented in BlueSky. This would show whether using thrust settings and defined flight path angles could simulate more true to reality behaviour, as BlueSky currently only possessed the option to perform flights using autothrottle.

The result of which were the following conclusions:

- The *how* is answered by using thrust settings and the option to provide flight path angles. Basic functionality of these options showed behaviour according to reality, and can therefore be concluded to be working. This was proven with a simple example in the results, therefore providing a proof of concept.
- The new logic (being the command structure) has resulted in the model with thrust settings and FPA, which in itself follows basic flight profiles, therefore proving the validity of the logic. That is, on the condition that Fuzzy logic has satisfactory membership functions, as it significantly can influence the result. Moreover, its intuitiveness has been proven using real world examples of flying behaviour.
- Validation of the entire model has been attempted, but has resulted in a partial success. Due to the combination of two different performance models for both preprocessing as execution, a certain discrepancy has been introduced which is most likely a significant contributor to the observed results.
  Moreover, the sensitivity analysis has shown the relevance of correct input values for descent flights, therefore further proving the impact of this double use of performance models. This also hindered the possibility for large scale testing, as single workable result were already difficult to find. This was mainly concerning climb flights, as descent flights have been far more successful due to the relative smaller possibility of stalling. Climb flights

have given unsatisfactory results even when changing thrust settings to the maximum value. Thus, basic validation has taken place, but a comprehensive validation has not been possible. This eliminates the possibility of drawing a decisive conclusion about the *entire* model.

- Regarding the simulator itself: BlueSky is built using two performance models that use a flight phase logic to determine aircraft performance. Following the determination of the phase logic, aircraft movement is calculated from which forces follow. Therefore, implementing an idea which uses forces as an input and calculates movement has shown significant challenges. This was also seen in the results for work. Performed work was similar in old and new situation, indicating a similar applied force. However, this still yielded very different flight profiles. The research has been unable to prove to what extent mentioned challenges caused the observed behaviour, or can be attributed to other mentioned factors. In any case is certain that this frequently caused flights to be unable to accelerate in climb scenarios.
- Making alterations to VNAV has not been possible as the research could not progress further than enabling the basic functionality to work. Concluding whether VNAV works seamlessly and also profits from this new functionality is therefore not possible.

Thus, summarising the answer to the main question: to provide users requiring the possibility to simulate advanced use cases, thrust settings and flight path angle functionality enable such. That is, if rudimentary representative flight behaviour is sufficient, as basic rule of thumbs present in BlueSky have not been significantly improved by this new model. This does create enough opportunity to provide suggestions for future work, presented in Section 6.2.

6.2 Suggestions for future work

Although the research might not have given the desired result, it did open up opportunities for a possible scenario in which the model could actually work. By providing suggestion for future work the possibility is created to attain more accurate results in future.

Flight phase logic

One major unknown in the research has been the flight phase logic. For kinematic performance models holds that they use data from which forces are derived. The used data is chosen based on the flight phase that the aircraft currently is in. The determination of this phase and thus the subsequent assumptions were not within the scope of this research. However, it did most likely influence the outcome. Each phase has certain assumptions and methods, deriving according parameters (e.g. drag force) which may not be valid for the new methodology.

Moreover, this is further accentuated in autothrottle phases. Whenever the new model would use the autothrottle in level, steady phases, the model would essentially resort back to the old model (thus rudimentary acceleration calculations). The consequence of this blend has not been investigated.

A suggestion would be to dive into either performance model to ensure its compatibility with thrust settings. Furthermore, it should simultaneously be ensured that it is capable of switching seamlessly between the two kind of logic schemes without sacrificing model accuracy.

Continuity in methodology

One of the major setbacks in this research has been that continuity has not been guaranteed as different models were used for the same process. The solution to this problem is twofold, and can be resolved by following either suggestion.

One of which is ensuring that OpenAP attains a improved thrust model, in which thrust at least stays within maximum boundaries. It was taken out of the process because of this sole, but essential, issue.
Another option would be to develop a stand-alone BADA model for the preprocessing of data. As this has been researched and found in a rudimentary phase, it is very likely that this option can be chosen to further progress this research.[1] One note is though that using BADA 3.0 will lose its relevance over time, as BADA 4.0 is the newer version but not included in BlueSky.

Separate validation of models

Another suggestion is change the workflow of the current research. The current validation of the model implemented in BlueSky is done with an input file created in a data formatting process. It is difficult to determine whether one of the two steps have introduced significant deviations. Though, when using the same performance models for both steps, these deviations would be relative. Still, to ensure a realistic behaviour as stated in the research question, a separate validation of both concepts would ensure more robustness.

Validation

The discussion addressed the fact that the model has not been validated for a large number of flights. As this is essential to obtain a robust model, validation ideas that were thought of are presented as suggestion for future work.

Flights can be compared by using a total energy or work model. Solely comparing for altitude or speed leaves out one of the two, therefore not providing sufficient information whether a flight profile is properly followed. To illustrate how this would look like, work has been shown as function of the normalised distance in Figure A.2 and Figure A.1. This normalisation would be required to compare using a similar distance. This would of course require the total distance difference among the models to be negligible.
Another idea would be to compare level segments to compare accelerations. Whenever the aircraft would accelerate or decelerate on level parts, the required distance can be compared. It can then be measured whether accelerations are more true to reality, instead of using rules of thumb. In any of the cases holds that these results can be presented in a density graph or boxplot, in order to attain a statistical conclusion.

[1] Please refer to https://github.com/tabassco/badapy for the source code [Cited:17-02-2022]

A

Appendix A - Normalised distance

A.1 Descent

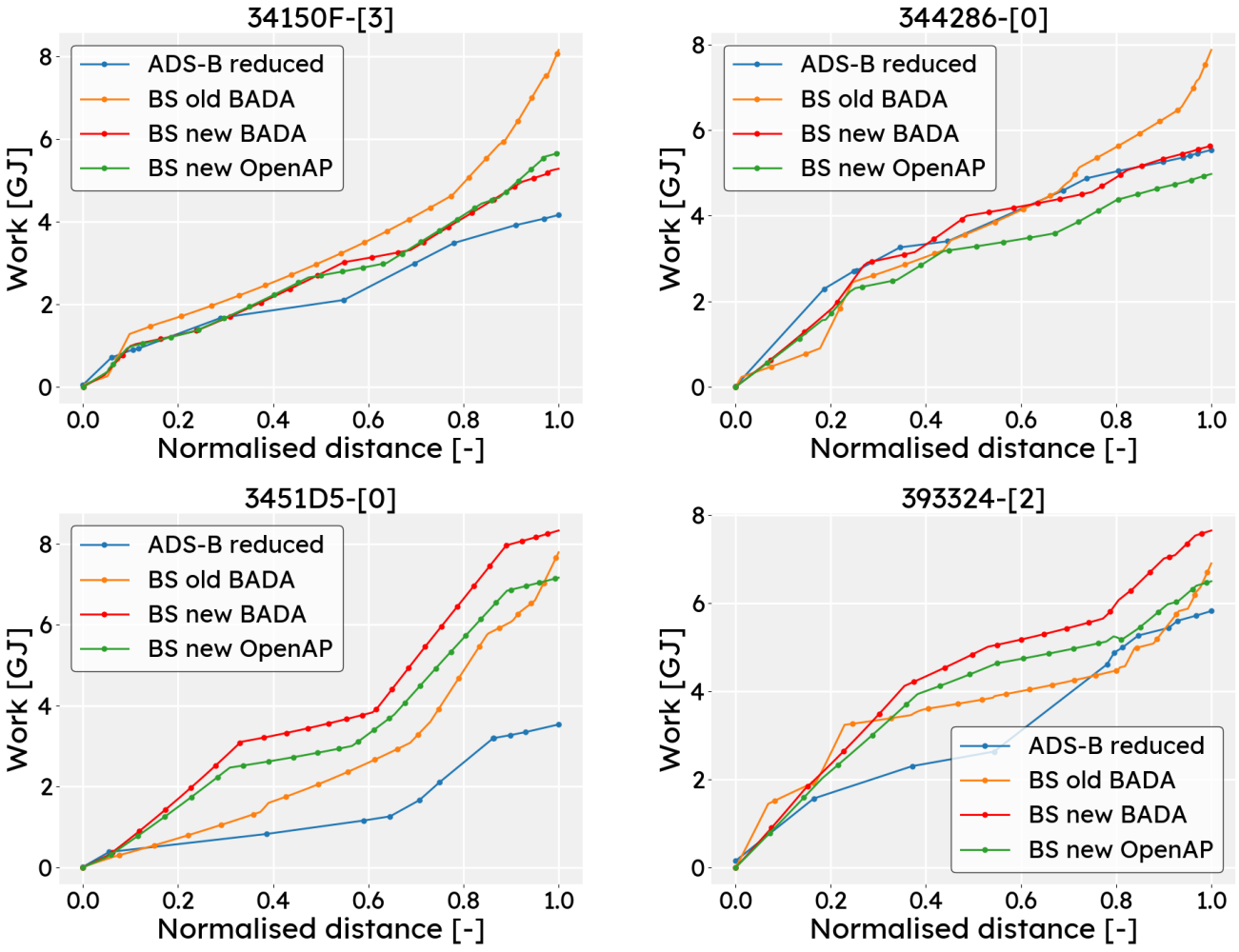

Figure A.1: Work profile for four descent flights using normalised distance

A.2 Ascend

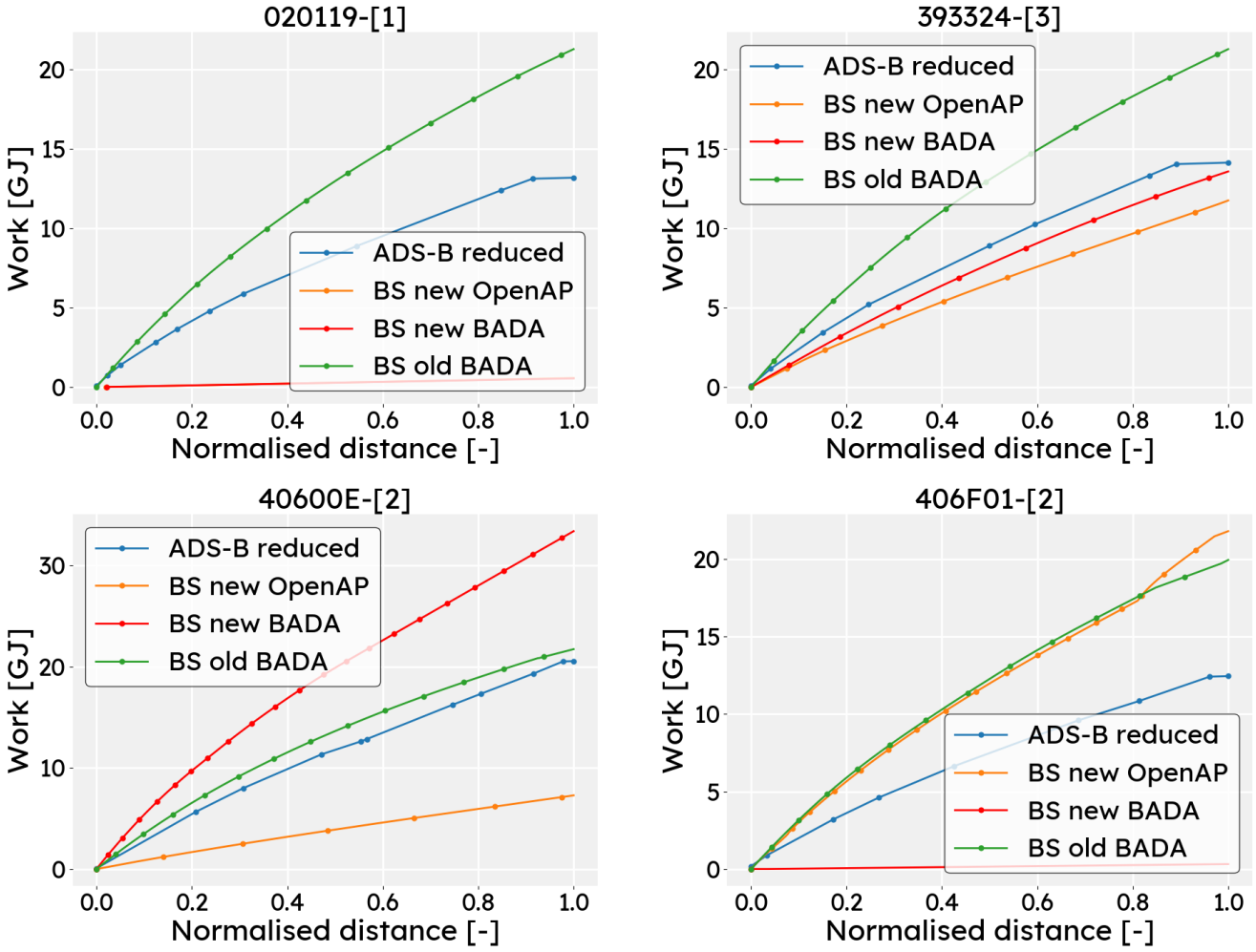

Figure A.2: Work profile for four climb flights using normalised distance

Bibliography

[1] Jacco Hoekstra and Joost Ellerbroek. BlueSky ATC Simulator Project: an Open Data and Open Source Approach. In *7th International Conference on Research in Air Transportation: Philadelphia, USA*, 06 2016.

[2] Junzi Sun, Jacco M. Hoekstra, and Joost Ellerbroek. OpenAP: An Open-Source Aircraft Performance Model for Air Transportation Studies and Simulations. *Aerospace*, 7(8):104, 07 2020. ISSN 2226-4310. doi: 10.3390/aerospace7080104.

[3] Angela Nuic, Damir Poles, and Vincent Mouillet. BADA: An advanced aircraft performance model for present and future ATM systems. *International Journal of Adaptive Control and Signal Processing*, 24(10):850–866, 2010. doi: https://doi.org/10.1002/acs.1176.

[4] Neno Ruseno and Mahardi Sadono. Body of Knowledge in Research of Air Traffic Management: Case Study in Indonesi. *Angkasa: Jurnal Ilmiah Bidang Teknologi*, 11:9, 05 2019. doi: 10.28989/angkasa.v11i1.400.

[5] David G. Hull. *Fundamentals of Airplane Flight Mechanics*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 3540465715.

[6] Mark Peters and Michael Konyak. The Engineering Analysis and Design of the Aircraft Dynamics Model For the FAA Target Generation Facility. Technical report, FAA, 2012.

[7] Muhammad Ahmad, Zukhruf Liaqat Hussain, Syed Irtiza Ali Shah, and Taimur Ali Shams. Estimation of Stability Parameters for Wide Body Aircraft Using Computational Techniques. *Applied Sciences*, 11(5):2087, Feb 2021. ISSN 2076-3417. doi: 10.3390/app11052087.

[8] Junzi Sun, Jacco M. Hoekstra, and Joost Ellerbroek. Estimating aircraft drag polar using open flight surveillance data and a stochastic total energy model. *Transportation Research Part C: Emerging Technologies*, 114:391–404, 2020. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2020.01.026. URL https://www.sciencedirect.com/science/article/pii/S0968090X19307764.

[9] *User Manual for the Base of Aircarft Data (BADA) Revision*, 2014.

[10] Junzi Sun, Joost Ellerbroek, and Jacco M. Hoekstra. WRAP: An open-source kinematic aircraft performance model. *Transportation Research Part C: Emerging Technologies*, 98: 118–138, Jan 2019. ISSN 0968090X. doi: 10.1016/j.trc.2018.11.009.

[11] Gerard T. Fairley and Seamus McGovern. A Kinematic/Kinetic Hybrid Airplane Simulator Model. In *Volume 1: Advances in Aerospace Technology*, page 11–20. ASMEDC, Jan 2008. ISBN 9780791848623. doi: 10.1115/IMECE2008-66430. URL https://asmedigitalcollection.asme.org/IMECE/proceedings/IMECE2008/48623/11/334802.

[12] Damir Poles, Angela Nuic, and Vincent Mouillet. Advanced aircraft performance modeling for ATM: Analysis of BADA model capabilities. pages 1.D.1–1, 11 2010. doi: 10.1109/DASC.2010.5655518.

[13] Gary Slater. Study on Variations in Vertical Profile for CDA Descents. In *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*. American Institute of Aeronautics and Astronautics, 09 2009. ISBN 9781600869778. doi: 10.2514/6.2009-6912. URL https://arc.aiaa.org/doi/10.2514/6.2009-6912.

[14] L. Sherry, M. Feary, P. Polson, R. Mumaw, and E. Palmer. A Cognitive Engineering Analysis of the Vertical Navigation (VNAV) Function. 2013.

[15] PMDG 747-400 Type Rating Course - Introduction, 2006. https://pmdg.com/pmdg-747-400-queen-of-the-skies-ii-base-package-for-prepar3d/#product-tab-description.

[16] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(65)90241-X. URL https://www.sciencedirect.com/science/article/pii/S001999586590241X.

[17] Junzi Sun, Joost Ellerbroek, and Jacco Hoekstra. Large-Scale Flight Phase Identification from ADS-B Data Using Machine Learning Methods. 06 2016.

[18] Damien J. Melis, Jose M. Silva, Miguel A. Silvestre, and Richard Yeun. The effects of changing passenger weight on aircraft flight performance. *Journal of Transport & Health*, 13:41–62, 2019. ISSN 2214-1405. doi: https://doi.org/10.1016/j.jth.2019.03.003. URL https://www.sciencedirect.com/science/article/pii/S221414051930009X.

[19] Jefry Yanto and Rhea P. Liem. Aircraft fuel burn performance study: A data-enhanced modeling approach. *Transportation Research Part D: Transport and Environment*, 65:574–595, 2018. ISSN 1361-9209. doi: https://doi.org/10.1016/j.trd.2018.09.014. URL https://www.sciencedirect.com/science/article/pii/S1361920917309288.

[20] Hugh Harrison Hurt. *Aerodynamics for naval aviators*. Naval Air Systems Command, California. ISBN 9781619540170. URL https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/00-80t-80.pdf.