Glider Pilot Trainer

Final Report

Submitted for the MEng Computer Science with Games Development

May 2018

by

Rhys David Blackett

Word Count: 8479

Table of Contents

1. Introduction

The intention of this report is to elaborate on the design process, aims and objectives of this project – the domain of which is the creation of a simulation of manned vehicular flight. The intended purpose of this is to have something that could be used to help novice pilots with no past experience to understand the basic principles of the operation of an unpowered glider. Such simulators are used both recreationally and in the training of future pilots. Throughout the development process, particular attention has been paid to the level of realism of the flying experience, with the intention of being able to compare data gathered throughout a simulated flight with data gathered through real world flights and wind tunnel tests and see a close correlation. The reasoning behind this is that if a simulator is to prove useful as a training tool it must be as faithful a representation of an actual flight as possible – it is as important to make pilots aware of what to expect during certain manoeuvres as it is to avoid training pilots to deal with problems that will not exist in a real flight.

This report outlines the various principles and materials considered applicable to the project's subject area as well as a breakdown of the reasoning behind various design decisions over alternative solutions in addition to an analysis of similar solutions already implemented in other software.

Furthermore, aims and objectives as well as the end goals and potential further additions onto the project that would be implemented if time allows will also be presented in this report. These same objectives will be further broken down in the tasks section and the development processes documented.

2. Background

2.1. Problem Context

There are various approaches to simulating flight, however the main principle and object of a flight simulator is always “to reproduce on the ground the behaviour of an aircraft in flight” (Rolfe & Staples, 1986).

Simulation itself has a variety of possible uses. One of these is in the research and design process of creating whole new aircraft. “Flight simulators allow designers to explore the implications of different design options” (Rolfe & Staples, 1986). This purpose, however, is beyond the scope of this project. The intended use of the final delivered software developed during this project is the training of pilots in the successful operation of pre-existing aircraft – focusing mainly on the accurate portrayal of unpowered lightweight glider aircraft, with the possibility of introducing propeller powered craft as stretch goals.

Emphasis should be added on the recreation of real world flight characteristics, basic lift and drag should be modelled so that the aircraft handles realistically with convincing behaviours such as stalling, wind direction, energy retention through manoeuvres and landing procedures as well as audio cues like pre-stall noise and touch down sounds – all of which are important to the creation of an immersive and useful training simulator. Unlike conventional videogames, simulators have a completely different design philosophy. Where videogame projects would put emphasis on user friendly but fun gameplay and could get away with hardcoding behaviours for vehicles, etc., a flight simulator does not have this luxury. The simulator must focus on the way that forces act upon a vehicle and the accurate modelling of these forces is what leads to the vehicle having satisfying and realistic behaviour, even if for most users it may be too difficult to control.

The obvious application of a flight simulator is the replacement of time spent training in real aircraft, to reduce safety risks and expenses. *Training Transfer* is a term used to refer to the effectiveness of a training simulator (Rolfe & Caro, 1982). A transfer effectiveness ratio (TER) is given by

$$TER = \frac{Tc - Te}{Xe}$$

where Tc is the amount of airborne time needed by a control group and Te is the amount of time needed by an experimental group to reach a certain criterion of perceived flight ‘skill’ and Xc is the time spent in the training device. The following table shows a practical application of this principle;

Table 1.1 Results of a training transfer experiment

Group	Time to first solo (hours)	Simulator training (hours)	Training Transfer (%)
1	14.94	0	-
2	11.60	3	22.15
3	8.24	7	44.70
4	8.43	11	43.42

(Allerton, 2009)

Ideally it would be possible to measure the Training Transfer of this project, to see if it has a positive TER, indicating it to be a successful simulator in the training of pilots, or a negative TER, indicating it to be unsuccessful. However measuring this is beyond the scope and means of such a project.

2.2. Pre-existing solutions

The kind of simulator designed and implemented in this project should be similar in usage to that of already existing simulators such as Microsoft Flight Simulator – meaning it should be a faithful representation of the handling properties and characteristics of real aircraft, being accessible to both novice and experienced pilots alike. This approach is one of emphasising recreation of real world craft in as much intricate detail as possible. Simulators such as this are often used in training not only recreational pilots but also in training commercial airline pilots, a testament to their accuracy. Not only are such simulators present in the videogames industry, they are also often commissioned by aerospace companies, for example Lockheed Martin purchased the intellectual property for Microsoft Flight Simulator in 2010 for development of their own simulator. (Johnson, 2010)

A simulator with a similar training role is the Cockpit Procedures Trainer (CPT), however this style of simulator puts a much larger emphasis on the procedures and intricacies of working within a cockpit and learning the instruments, with very little in the way of actual aerodynamic simulation. (U.S Navy, n.d.)

There also exists a wide range of games which portray flight in varying degrees of realism. One such game is Kerbal Space Program – a game with a surprisingly accurate portrayal of Newtonian physics, it also allows for the design of aircraft with the goal of eventually designing space-flight capable 'space planes'. Because of the nature of this game being an emphasis on user designed vehicles it is easy for any player to put together an aircraft using various parts – the physics being calculated for each individual part added to the aircraft (mass, surface area, etc.). If a system similar to this were implemented within this project it would allow for the quick implementation of a variety of different aircraft, though it might sacrifice on the realism factor of the way they handle when compared to real life counterparts. However, it would also allow for the implementation of a custom aircraft designer – while not necessarily important to the intended goals of the project, it would make the simulator more appealing to a wider range of users and also could help would-be pilots develop a more intimate understanding of the fundamental physics involved with flight.

2.3. Comparison of Technologies

Game engines make the basis for almost any game or simulator design project. Some existing flight simulators use their own game engine, such as Microsoft Flight Simulator. Flight Simulator World uses this same engine. In addition to game engines, various middleware solutions should also be considered.

Havok is a middleware physics engine, used in over 600 games (Havok, 2017). The features that it offers are similar to those offered by the Unity engine, however it does not include the convenience of already being fully implemented within a game engine. A major downside of Havok Physics would also be its price point while the aforementioned Unity engine is free to use. There does exist a wide variety of other physics solutions, such as Phys-X, Bullet or Vortex, all with their own pros and cons.

For the sake of time constraints this project will use a pre-existing engine. Unreal and Unity stood out as the most obvious choices – with the previously mentioned Kerbal Space Program being built on Unity. In the end it was decided that Unity would be the best choice largely due to pre-existing experience with that engine as well as its generally high level of usability and the prospect of being able to quickly make working prototypes, allowing the main focus of the project to be placed on the implementation of the physics and flight. A particular advantage of Unity is the fast and effective creation of expansive terrain through the utilisation of its terrain systems. This also presents the possibility of quick creation of trees and other such terrain features which can prove important to the user, allowing speed to be perceived by acting as

landmarks and effectively conveying the movements of the aircraft to the user (Unity Technologies, 2017).

The problem of how to simulate real world physics within the constraints of a game engine is popular and well-documented within the games design industry. In this project it was decided that Unity's inbuilt physics engine would be used as little as possible when it comes to the creation of the actual flight model, for the sake of both learning as well as to grant greater control over the way physics calculations are made, since Unity lacks any sort of native support of lift producing surfaces. Unity's system of rigid bodies and colliders will be utilised to its full potential to allow greater focus on the modelling of realistic aircraft handling. Other objects within the simulation that may require physics but are not vital to the operation of the aircraft could fully utilise Unity engine's built in systems as this is quick and effective for creating objects that behave in a realistic way (Unity Technologies, 2017). Furthermore, these objects will also be able to interact with the aircraft being simulated without the need for any extra coding work.

2.4. Physics of Flight

There is extensive scientific research into aerodynamics and the physics of flight as subsections of fluid dynamics, but for the purposes of this project only the basics must be present. There are 4 fundamental forces involved with making an aircraft fly, these are its weight – or gravity, lift, thrust and drag (Smithsonian National Air and Space Museum, n.d.). Being primarily a simulator of unpowered glider flight, thrust is mostly irrelevant.

Weight is one of the simplest forces involved in simulating an aircraft in flight as it is effectively a constant, non-changing force. In reality gravity weakens very slightly at altitude – as well as geographical location – however at the height at which a glider flies this is negligible and does not need to be included in the simulation. For example at 3,000 meters weight would reduce by around 0.1% (National Physical Laboratory, 2010). Unity's inbuilt physics engine can easily handle basic weight and allows object mass to be quickly tweaked.

Lift is one of the more difficult forces to properly simulate within a game engine. A common expression used to calculate the lift of an aerofoil (the standard shape of most fixed-position wings) is as follows:

$$L = C_L \frac{1}{2} \rho V^2 S$$

where C_L is the lift coefficient, V is velocity, ρ is air density (which varies with altitude) and S is wing area (Torenbeek & Wittenberg, 2009). Lift coefficient changes with angle of attack and is responsible for what is known as a 'stall', where the aircraft's angle of attack has exceeded that of normal operation, resulting in the wings no longer providing lift. Recovering from such stalls is a commonly trained manoeuvre for pilots and therefore must be present within the simulation if it is to fulfil its intended purpose as a training aid.

The graph below is an example of how angle of attack may affect the lift coefficient of an aerofoil type wing:

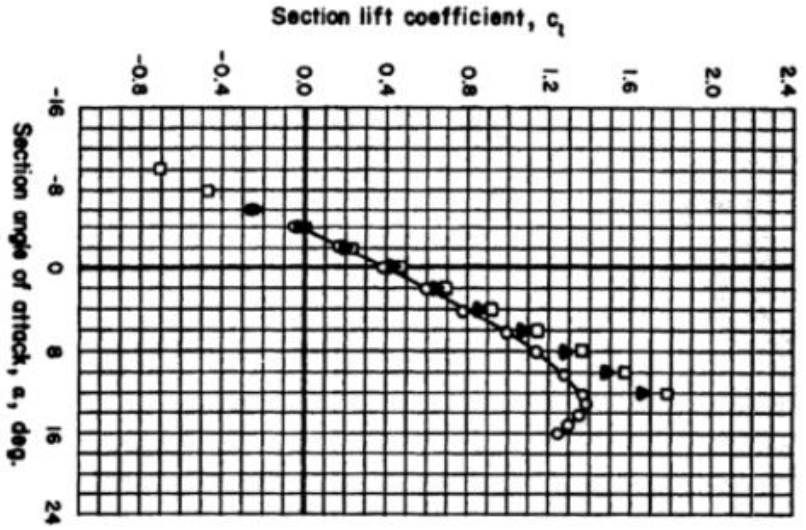


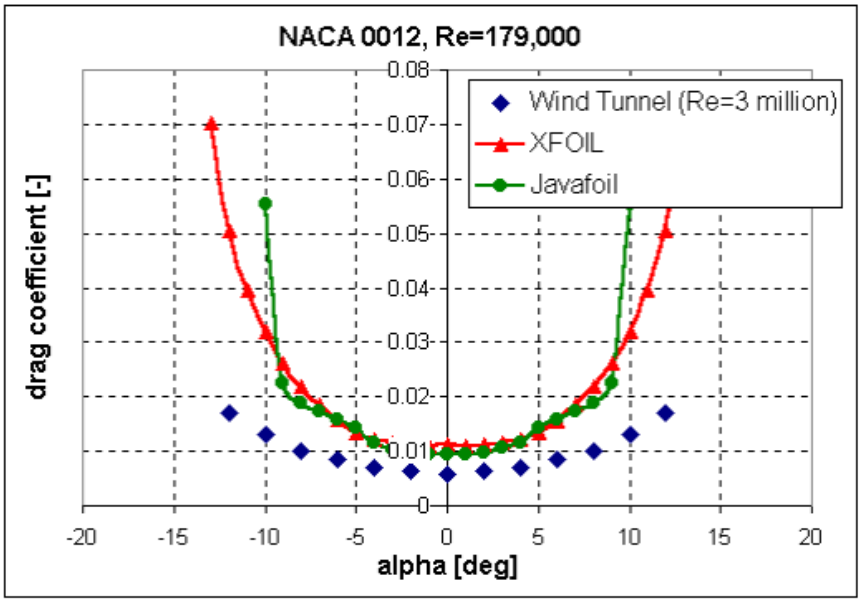
Figure 1 Section Lift Coefficient of Aerofoil Wing (Smetana, 2001)

Stalling also leads to further behaviours that pilots must train to deal with – principle amongst these is 'wing drop'. A number of aircraft have the tendency to drop a wing or roll at the stall which could result into a spin if left unattended (Experimental Aircraft Info, 2006). These characteristics will be covered further in section 2.6.

Drag has the potential to be very complicated if calculated as accurately as possible, however such tiny attention to detail will have little impact on the behaviour of the aircraft. Generally for aircraft drag is calculated as follows:

$$D = Cd\frac{\rho V^2}{2}A$$

Where Cd is the drag coefficient, ρ is the density of the air, V is velocity and A is the reference area. (NASA, 2015) Calculating the drag coefficient would be even more complicated than it is for calculating the lift coefficient. As such a similar approach should be taken, with the value being taken from an array, varying depending on the surface's angle of attack. An example of how this may look in graph form is given below:



NACA 0012 drag coefficient at a Reynolds number of 179,000

Figure 2 Section Drag Coefficient of Aerofoil Wing (Scott, 2006)

2.5. Characteristics of Glider Flight

The forces and physics described previously all culminate to form various distinct characteristics familiar to glider pilots, as well as enabling various features to be easily implemented; this is because if the fundamental forces are all correct, then the behaviours will manifest themselves as a result of this with no need for hard-coding or 'hacking' solutions. The plane being modelled within this project is a Grob-102-Astir. Further details of this craft can be found in Appendix A.

2.5.1. Launch Methods:

Aero-tow and Winch are the two most common methods for launching an unpowered glider (Piggot, 1990). The project will focus on winch launching as the main method to get the craft airborne. This involves a stationary ground based winch, usually mounted on a heavy vehicle, which pulls the aircraft along the runway strip. After the minimum safety speed of 50 knots is attained, the glider will be "rotated" into its normal climb attitude, which is at an angle of about 45° to the ground (Mendip Gliding Club, n.d.).

2.5.1. Energy Management:

An aircraft's energy condition is a function of various parameters, these include airspeed, altitude, drag caused by airbrakes, landing gear, flaps, and thrust (SKYbrary, 2017). Energy management is especially important for all aircraft types during take-off and primarily during landing and final approach. For gliders it is also incredibly important, especially when performing aerobatics. Essentially energy management in a glider is the transfer of kinetic potential energy in the form of altitude into kinetic energy in the form of airspeed, and vice versa. Poor energy management can easily lead to situations where it is no longer possible to reach the runway to land and the craft having to be ditched in a suitable field, possibly damaging it in the process (Piggot, 1990).

2.5.2. Soaring and Cruising:

Gliders can fly for surprisingly long periods of time when proper 'soaring' techniques are employed. In ordinary cruising conditions the majority of gliders will cover around four miles for every 1,000 feet of altitude lost (Piggot, 1990). A common soaring technique is the usage of thermals. These are masses of rising air formed as a result of water vapour or warm air caused by the sun's heating effect on the ground below (Bradbury, 2004). These thermals push the aircraft up with them and allow the pilot to gain sufficient altitude to locate and move towards the next thermal, often these can be located by the formation of cumulus clouds, though this is not always possible. Other means to soar and gain altitude include wind phenomenon such as ridge lift, where wind is directed upwards by hills and other terrain features (Eckey, 2009) as well as wave lift, analogous to the ripples on the surface of a stream. In terms of project implementation, thermals will take the biggest priority as these tend to be the most common and new-pilot friendly method of climbing and so are valuable for training purposes.

2.5.3. Adverse Yaw:

If bank (Aileron input, rolls the aircraft) is applied without any usage of the rudder, the nose of the aircraft will swing towards the opposite direction which the pilot is intending to bank towards (Piggot, 1990). The reason for this is partially because by definition, lift is always perpendicular to oncoming airflow (Hage & Courtland, 1965), this is demonstrated in the diagram below. Additionally it is also related to an increase in drag produced by the downward going wing's deflected aileron. Though the effect of this may seem subtle, it is important that it is present within the simulation as it is something that pilots must constantly keep in mind. Additionally these forces are partly responsible for the early stages of most aircraft stalls.

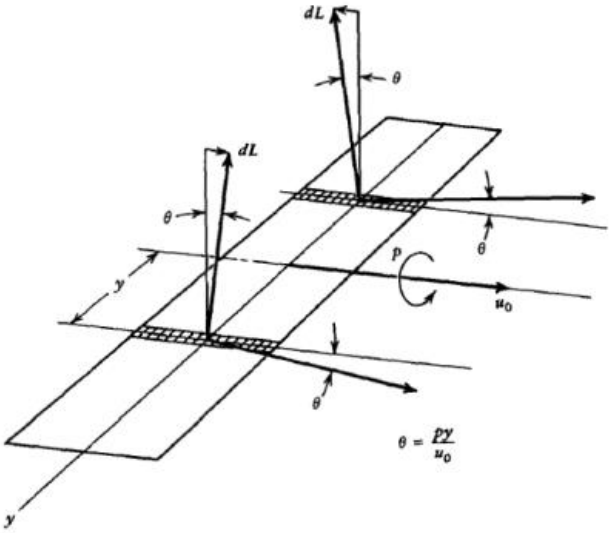


Figure 3 Adverse Yaw (Vitsas, 2016)

2.5.4. Stall Behaviour:

When the nose of the glider is raised above the normal gliding attitude the craft will experience a temporary increase in lift and drag – this may cause the aircraft to climb. However, this extra drag will quickly slow the craft. Initially the craft will attempt to nose down, but if further elevator is applied a stall may be induced – perceived by the pilot as rapid descent and the nose of the craft trying to point downwards as lift over the wings is lost, as described previously in the *Physics of Flight* section. In terms of training potential, realistic stall behaviours are of utmost importance as stalls can be potentially deadly if not handled correctly, largely due to the fact that they most commonly occur during take-off and landing where there is insufficient altitude to be able to recover from a stall (Shih, et al., 1992). If there is lateral motion when stall conditions are met it is likely that one wing will stall before the other, leading to wing drop occurring. This downward motion of one wing often then leads to the wing moving further into the stall, causing an even more sudden drop. This often prompts the pilot to react using aileron – however depending on what stage of the stall the wing is currently experiencing this aileron input can make the situation even worse by increasing the stalled wing's effective angle of attack and inducing further stall, sometimes developing into a spin (Experimental Aircraft Info, 2006).

Spins are caused by autorotation, which is linked closely with adverse yaw and describes the state when the aircraft is rotating, be it either roll or more commonly yawing in a spin, without the user's intent. Autorotation spins and spirals are generally easy to come out of in glider flight, requiring just a little opposite rudder and a light touch on the elevator and aileron controls (Piggot, 1990).

2.5.5. Side Slipping and Landing:

Judging the rate of descent and speed leading up to a landing can be surprisingly challenging, especially in an aircraft that lacks effective airbrakes or spoilers. One method used to reduce altitude without gaining airspeed is known as 'side slipping'. This involves applying roll to one direction, and purposefully applying rudder in the opposite direction to counteract the turning effect of the roll – this leaves the aircraft flying with a high amount of 'side slip' – the nose is either pointing to the left or right of the actual heading of the aircraft – as a result there is a marked increase in drag produced by the fuselage or body of the craft (Piggot, 1990). Similar to stall behaviours, this is valuable as a tool for training pilots as landing often proves to be the most dangerous and most difficult part of flying an aircraft.

3. Aims and Objectives

This project aims to produce a flight simulator for training glider pilots. It should implement a simplified aerodynamic model of glider flight as well as various visual components including cockpit instrumentation, the aircraft itself as well as various terrain elements including a landing strip. The end result should be the ability to start a flight from cable launch, circuit of the field and surrounding terrain and ultimately performing the final approach and landing of the glider on the runway. More advanced features could be added if the project is ahead of schedule, such as the simulation of wind and turbulence and how they affect flight.

If time permits, with some additional work powered flight could also be implemented. Simple single or double engine propeller propulsion aircraft, with their handling and characteristics tweaked appropriately to simulate how their real world counterparts might handle. In addition to this various challenges of pilot skill could be added – specific scenarios such as varied weather conditions or waypoints and manoeuvres to follow.

Dependent on time constraints, thanks to the nature of the unity engine it may be possible to make the aircraft in the simulator somewhat modular, allowing users to change out various parts before their flight to create something unique and have the flight characteristics also affected by their changes, in theory deepening their understanding of how aircraft operate.

3.1. The Problem of "Realism"

A large part of creating a simulation is attempting to have the aircraft behave "realistically", meaning it handles and behaves the same as it's real life counterpart. This can prove problematic however when one wishes to evaluate progress towards this goal. It is easy for something to seem realistic on first glance when in actuality it is not, and under certain conditions this may become obvious. Therefore it is important to be able to quantify "realism" through rigorous tests and evaluation of raw data that can be compared to aerodynamics data collected by real world aerodynamicists and scientists. This in its self is also not without issues however, as it is very possible to delve too deeply into how the physics of flight actually works and end up with a simulation incapable of running in real time that has taken many years to create. To this end it has to be identified when collecting data to compare against how deep into flight physics the project should go as at some point the differences made may have no observable impact on the user experience.

3.2. Objective 1 – Simple Rendered Environment and Glider in Unity Engine

The first step is to get basic visuals nailed down so that work can begin on the physics involved in making the glider fly – doing so without any visuals would make testing markedly more difficult. This objective could be expanded upon to include visual movement of parts of the aircraft based upon control inputs.

3.3. Objective 2 – Basic Forward Movement, Manoeuvres and collisions.

Gravity and drag should affect the glider, with appropriate centre of mass and a simple way to generate lift at a single point on each wing. Unlike the final product this lift could be linearly related to the speed of the aircraft and as such have no realistic stall characteristics, but allow the user to "fly" the aircraft around the world space in an unrealistic manner. Drag can make use of Unity engine's built in drag model for the purposes of this stage of the project – though this would prove insignificant for the final product. Implementation of basic collisions is a simple task thanks to Unity's built in hitbox functionality.

3.4. Objective 3 – More Advanced Modelling of Lift

One of the two areas that require the most attention in order to meet the intended purpose of the program is the accurate calculation of lift. This means faithfully recreating the real physics equations for calculating the lift of a fixed wing using variables acquired from within the simulation. The end result should be a script that can be used to generate forces at any point on the aircraft wing, with an accurately modelled lift coefficient variable – this variable in combination with simpler variables such as velocity and angle of attack will result in a sufficiently realistic stall behaviour. Velocity should also be calculated based on the local velocity of the point of the aircraft that is making the lift calculation. For example a point near the tip of one of the main wings may be moving at a different velocity due to the aircraft rotating relative to the world around it than the tip of the opposite wing.

3.5. Objective 4 – Advanced Modelling of Drag

The second of the two main development areas is the modelling of the drag created by various points on the aircraft. As with lift this must use local velocity relative to the world in addition to accurately modelled drag coefficients – based upon real world scientific data gathered through wind-tunnel tests of aerofoil type wings. Ideally the drag coefficient should also change as control surfaces are manipulated as this has a direct effect on airflow over the wing and by extension can increase or decrease drag depending on the angle of attack of the wing.

3.6. Objective 5 – Realistic Stall Behaviour

This objective is more of an amalgamation of objectives 3 and 4 than a specific task in and of its self. The aircraft should behave realistically at the point of stall, with characteristic behaviours such as adverse yaw, wing drop and spins caused by autorotation being present.

3.7. Objective 6 – Full Flight Possible – Take-off, Soaring and Landing

This would mean the near completion of the main goal of the project – the ability to start a flight from cable launch, perform manoeuvres, use thermals or other means to soar to higher altitudes and then successfully execute final approach and landing of the vehicle on the landing strip.

3.8. Objective 7 – Additional Features

Non-essential features such as the implementation of additional aircraft or powered flight as well as challenges. This could also include features such as dynamic wind direction, varying air densities with altitude and temperature, do-not-exceed speeds or calculations of G-forces and their potentially damaging effects on the craft as well as sound effects.

4. Technical Development

4.1. System Architecture and Design

A key part of the design philosophy of this project is that the real-world physics of flight should be recreated as faithfully as possible. When the aircraft behaves in certain ways, for example when it stalls, it should be because the wings stopped producing lift due to a lack of speed or an excess in angle of attack and not because a preconceived trigger was set.

The simulator has been built using Unity's component based capabilities. The aircraft is a single entity which has multiple child objects, each representing an area of the plane which will need to make lift and drag calculations.

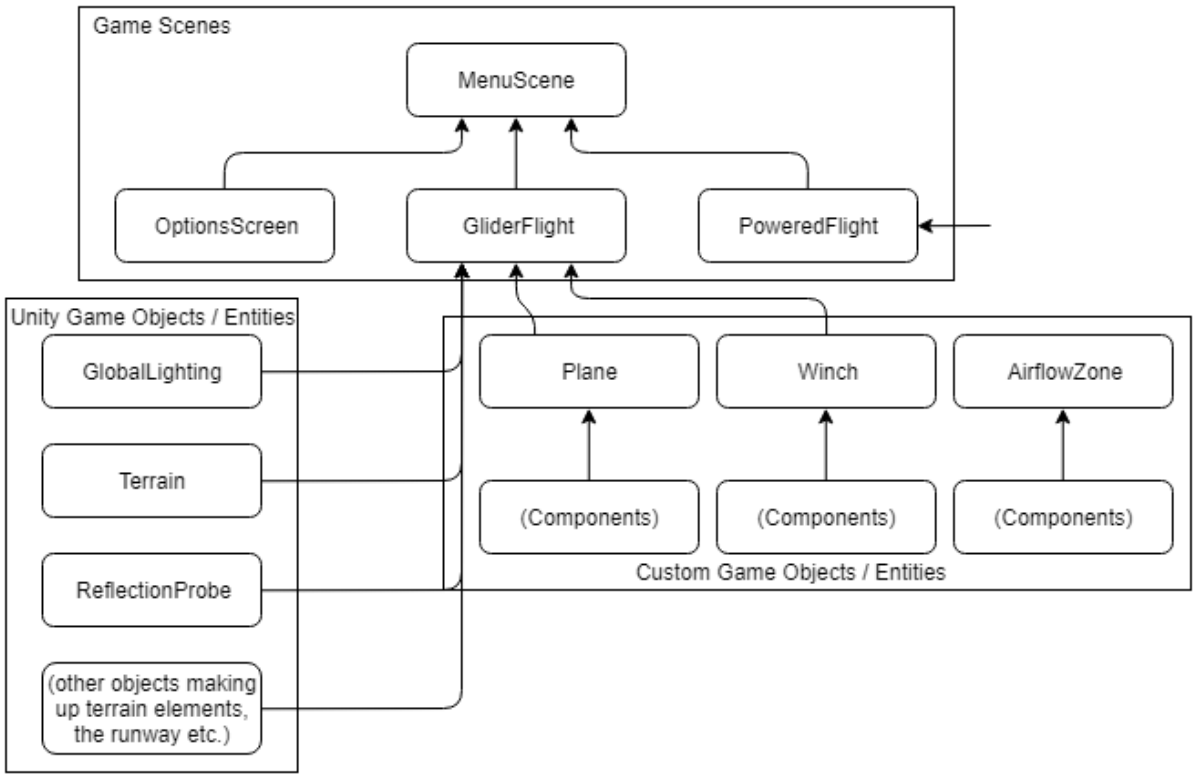


Figure 4 Overall Object Structure

Unity engine uses game scenes to handle the current state of the game. These scenes contain the various environments and menus of a game (Unity Technologies, 2017). The main menu of the simulator is implemented within its own scene and allows the user to choose between an options / settings screen which will include simple configuration options, as well as two versions of the main meat of the program – GliderFlight is a game scene that includes the actual simulation of flight, putting the user in a glider parked on the runway and hooked up to a winch ready for launch. An almost identical scene called PoweredFlight allows the user to start in a powered aircraft, without the winch launch, although this section of the simulation is quite rudimentary and not fully implemented.

Most visual elements of the project are handled largely by Unity engine's built in objects. For example the terrain is handled by a single object which allows the quick creation of an environment for the glider to be flown in. This allows the quick painting of terrain height in addition to features such as trees, rocks and foliage. Unity also handles lighting and reflections through a global lighting entity and reflection probes – with some minimal configuration this results in lighting that is more than sufficient for the scope of the project. Other entities such as basic shapes are used to make an area suitable for the launch and landing of the aircraft.

The real meat of the project is found within the section of figure 4 labelled as Custom Game Objects / Entities. This is where most of the development time is fit into the overall structure of the game engine, consisting most notably of the plane object.

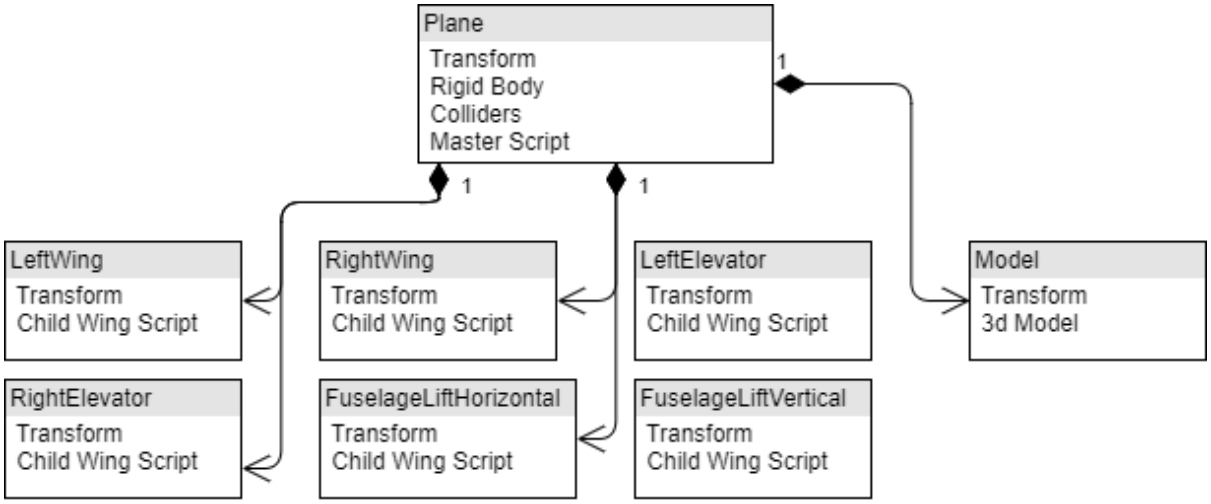


Figure 5 Example Plane Entity Structure

As can be seen in the diagram, the plane does not consist of a single entity, rather it has multiple children, this makes it easier to tweak the exact location of where each 'wing' is, including two objects which have been set up to simulate the small amount of lift and drag provided by the plane's fuselage near the cockpit. The only object within this hierarchy to actually contain a rigid body and any colliders is the parent 'Plane' object. This means that this object alone can use Unity engine's inbuilt physics calculations. It has its mass set to the appropriate amount for the entire aircraft and has its drag values set to 0, as these will be simulated manually by the various Child Wing Scripts.

'Master Script' is a simple script, with the express purpose of relocating the Rigid Body's centre of mass to the appropriate spot (near the wing leading edge (Grob Aircraft, 1982)) for the aircraft being simulated, in this case a Grob-102-Astir.

In the final version, this diagram is not 100% accurate, as each wing now consists of multiple entities going along the span of the wing, though the principal is the same. Variables such as the surface area of the wing and what control surfaces may be present are all able to be set from within Unity editor as they are declared as 'public' variables within the Child Wing Script.

While all the child objects have their own Transform component, it is directly affected by the parent 'Plane' object's Transform component. The main purpose to having multiple separate entities is that the wing script that they contain can easily grab their current location when applying forces to the parent entity's rigid body. This is important as it was decided to avoid using Unity's 'Torque' feature when adding the controls of the aircraft, the reason being that simply applying torque on top of the usual lift provided by any of the wings in order to apply pitch, yaw or roll can result in unrealistic handling of the aircraft, effectively creating energy out of nowhere.

4.2. Test Design

Early testing was mainly limited to observation of various statistics printed out by the wing script into the console for use in rudimentary unit testing. Things such as the aircraft's speed, angle of attack, current lift and drag. These are then compared with what the expected results, calculated using the lift and drag equations outlined in the background section of this report. Additionally, stall characteristics have been a central focus of testing thus far and have proven to be satisfactory for the current stage of development, but more nuanced behaviours such as deep-stalls and spins do not occur, due mainly to the simplicity of the current flight model.

4.2.1. Unit Testing

Unit testing is particularly useful in testing the accuracy of the implementation of mathematical formulas. It essentially involves using code to automatically run various other sections of the program with pre-set variables to see if it gets back the expected result. For example with lift:

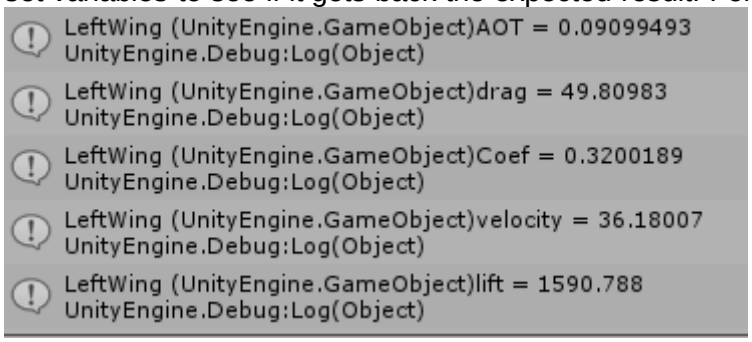


Figure 6 Output Window

The image above is a screen capture of the console showing the current statistics given by a main wing in level flight. Assuming all the input variables are correct, some of which are pre-set, such as the wing surface area and others which are calculated on the fly, we can solve the same calculation using the lift equation given in the background section of this report and see that it yields the exact same result of 1590.788.

The exact same test can performed to test the accuracy of the drag equation. Some example tests could be run within the code to check the equation gives expected results by temporarily setting the observed velocity, angle and appropriate coefficient and then running the lift calculation method.

4.2.2. Usability Tests

Usability testing usually relates to the evaluation of the effectiveness of User Interface elements as well as control inputs, it aims to identify whether users are able to complete specified tasks successfully and identify areas which prove problematic for most users (U.S Department of Health & Human Services, n.d.). Due to the nature of flight simulators the majority of UI elements will only be present within menus, with most of the information during a flight being displayed on the instruments within the cockpit of the aircraft – therefore this should be made to resemble the real world counterpart even if this isn't necessarily the most user friendly way to display information such as velocity, angle of attack, altitude, etc.

For the elements that do not need to be direct recreations, such as keyboard and mouse input, simple tests such as Hallway Testing should provide sufficient feedback. This involves randomly selecting individuals and observing as they attempt to use the program.

4.2.3. Aerodynamics / Physics Testing

In order to ensure a high degree of realism as outlined within the objectives a large focus must be put on the testing of specific areas within the aerodynamic modelling. Due to the nature and complexity of the intricate workings of the physics engine in addition to the wide variety of variables that can affect the aircraft it is important that a reliable system is developed for the presentation of data in a readable form.

- Change over time:

  Tracking certain variables and printing their values to a file or window at regular intervals is extremely valuable in the creation of graphs. Said graphs can then be compared side by side with similar sets of data that have been collected by aerodynamicists using scientific methods. This gives a quantifiable approach to determining realism within the simulation as discussed within the Aims and Objectives section.

- Data during stall

  Similar to collecting data over time, it will be important to collect data from specific events during a flight – one of the most important of these is during a stall. It is vital to collect as much data as possible just before, during and after a stall to be able to see how the variables affect one another and also to compare against real data as before. Lift and Drag coefficients stand out as two important variables that both require the largest amount of time to get correct as well as have one of the largest impacts on the amount of lift and drag produced. As such these will prove highly valuable for assessment.

The philosophy to keep in mind during the testing of the aerodynamics is the provision of data that can be easily compared to data collected during real world experimentation. This comes back to the problem of how to measure the degree of realism within a simulation. In a way it can be considered similar to general scientific processes – the simulation can be considered the experiment, and the equivalent real world data can be considered as being the hypothesis and, if the program was developed correctly, the two will be similar. The extent of these similarities is what will determine how realistic the simulation actually is.

4.3. System Implementation

4.3.1. Prototype Wing Script

When calculating lift and drag, care has been taken to implement real world physics equations as closely as possible, giving a result in newtons that is then immediately applied to the parent rigid body.

Child Wing Script

```
// Public variables which can be changed from within the Unity editor.
public RigidBody parentRB; (set as the rigidbody of the Plane entity)
public string wingType;
public float wingSurface;
public GameObject movable;
private float [,] liftCoef;
private float[,] dragCoef;

// Start method runs once upon starting the game.
void Start();
{
        if (wingType == rightWing or leftWing)
                Populate coefficient arrays with appropriate values for main wing
        else if (wingType == rightElevator or leftElevator)
                Populate coefficient arrays with appropriate values for elevators
        else
                Populate coefficient arrays with generic, symmetrical values for use on
                rudder and fuselage
}
// Fixed update method runs a set amount of times per second ingame.
Void FixedUpdate();
{
        Convert current velocity from world space into local space
        Calculate angle of attack based on local velocity
        CheckControls() // This method checks if any controls are being pressed, and if
        the wing type is appropriate to the controls being pressed manipulates the current
        angle of attack value before it is used to generate lift
        liftCoef = GetCoefficient(angle of attack) // This method gets a value from the
        array of coefficient values, interpolating between them when needed.
        ComputeLift() // This method calculates how much lift the wing is producing and
        applies it to the parent rigid body.
        dragCoef = GetDragCoefficient(angle of attack)
        ComputeDrag() // This method calculates the current drag this wing should be
        generating and applies the force onto the parent rigid body.
        MoveSurfaces() // This method will use quarternions to rotate the "movable" game
        object, this would be the appropriate control surface to provide visual feedback to
        the user's control inputs.
}
```

Figure 7 Early Version Wing Script Pseudocode

The pseudocode above is a simplified design of the script run by each wing entity as described in section 4.1. System Design. It outlines the basic functionality of the generation of lift and drag forces – When the simulation is started all appropriate coefficients are loaded into an array from a file, this is due to the complex nature of the coefficients plotted against angle of attack. FixedUpdate is a Unity engine method that runs every set amount of time and in this case is responsible for the generation of forces on the craft.

During development, there were two ways in which control inputs were made to affect the aircraft, the early stage method was reminiscent of the final version in the way it behaved except for the fact that it caused incorrect behaviours when flying at near-stall speeds. Effectively the entire wing surface is slightly tilted and the resultant change in the wing's angle of attack causes increased or decreased lift, in turn causing the craft to rotate. For instance, if the S key is pressed – with the intention of pitching the aircraft upwards – both the elevator entities at the rear of the craft will detect the S key being pressed and subsequently subtract a small amount from the effective angle of attack (effectively the same as angling the wing downwards). Then, when the lift is calculated the result is a downward force being applied towards the rear of the craft and the entire glider tilts back. Shortly after this the other child objects will all temporarily create more lift due to the increase of the aircraft's angle of attack and the entire plane begins to fly upwards. The issue with this method is that it can cause an entire wing to rapidly stall when it is given control inputs at low speeds – though this in its self isn't entirely unrealistic it is the manner in which it happened which was determined to not be sufficiently accurate and therefore not in-line with the project aims of emphasis on faithful recreation of real world physics and aircraft handling characteristics. To solve this issue the number of scripts running per wing was increased, with 3 points along each main wing making calculations and only one of these handling control inputs (the section near the wing tip where the ailerons are located).

Figure 8 Glider with Individual Wing Objects Highlighted

Additionally, it was found that forces were being applied incorrectly – particularly lift. There were also issues with the way that velocity was read from the game, reading from the parent rigid body caused issues where the rotation of the craft would not affect wing lift and drag. These issues will be covered further in section 4.3.3.

In order to move the control surfaces there must be a public game object, set within unity to be the appropriate bone within the plane's 3d model. This bone can then be manipulated from within the Child Wing Script using the Quaternion.Euler function, which takes in a 3d vector of angles to rotate the object by.

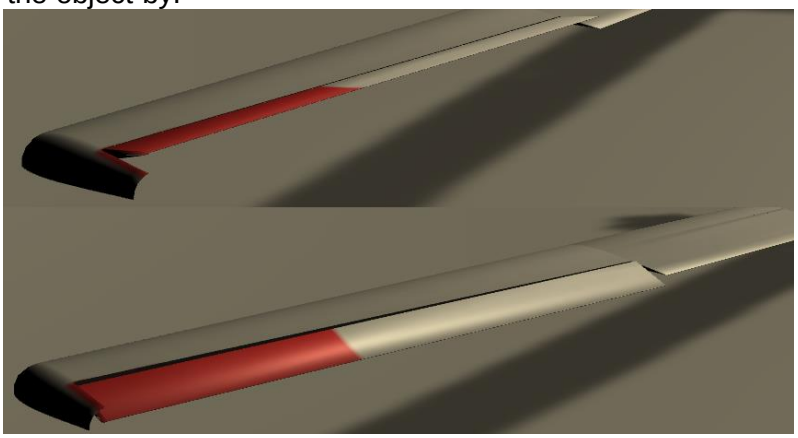


Figure 9 Aileron Deflection

4.3.2. Master Script

This script runs on the parent Plane object, and so only one of this script is present per aircraft. Its function is mainly to ensure that the centre of mass of the aircraft is set appropriately as well as the handling of current airflow.

Using OnTriggerEnter and OnTriggerExit methods the master script can detect when the aircraft enters an airflow object in the game world. It then gets the airflow value stored within these airflow trigger areas, and calculates their sum total. This is then used by all the child wing scripts when they make their respective calculations, giving the possibility of soaring to greater altitudes as specified within objective 6.

4.3.3. Wing Script Finalised

Figure 10 is pseudocode of the finalised version of Child Wing Script. Important methods and objects will be covered in further detail.

```
…
void Start()
{
        liftCoefficient = new Coefficient(liftCoefficientFileName)
        dragCoefficient = new Coefficient(dragCoefficientFileName)
        // Coefficients are now stored in a Coefficient object, to improve coding practice
by cutting down on duplicate blocks of code.
        Set rotation angles for moving parts, depending on type of wing.
}
void FixedUpdate()
{
        Vector3 currentAirflow = parent airflow // The current airflow the craft is
experiencing is stored within the main script running on the plane's parent object.
        Vector3 velocity = (currentTransformPosition – previousPosition) / deltaTime //
because the child wing entities do not have their own rigidbodies, they do not have a
velocity variable – therefore we must calculate this ourselves.
        velocity = velocity + currentAirflow // This accounts for aiflow and allows soaring
        vector3 localVelocity = Convert velocity to local space
        previousPosition = currentTransformPosition

        Calculate current air density linearly based on altitude // A simple solution not fully
realistic but better than not having any affect at all for the purposes of training.
        Calculate angle of attack based on local velocity
        CheckControls() // This method checks if any controls are being pressed, and if
        the wing type is appropriate to the controls being pressed manipulates the current
        angle of attack value before it is used to generate lift
        ComputeLift()
        ComputeDrag()
        MoveSurfaces() // This method uses quarternions to rotate the "movable" game
        object, this would be the appropriate control surface to provide visual feedback to
        the user's control inputs.
}
…
```

Figure 10 Finalised Wing Script Start and Update Methods Pseudocode

```
5 references
public class Coefficient
{
    private float[,] CoefArray;
    private List<string> coefficientList;
    2 references
    public Coefficient(string fileName) {
        coefficientList = new List<string>();
        // Attempt to use stream reader to fill CoefArray variable with appropriate
        //coefficient values. [Angle of Attack, Coefficient]
        try...
    }
    3 references
    public float GetCoefficient(float x)
    {
        int length = CoefArray.Length / 2;
        int offset = length / 2;

        int min = (int)Mathf.Clamp(Mathf.Floor(x / 5 - 0.00001f) + offset, 0, length - 1);
        int max = (int)Mathf.Clamp(Mathf.Ceil(x / 5) + offset, 0, length - 1);

        if (max == 0)
        {
            return CoefArray[0, 1];
        }
        if (min == offset * 2)
        {
            return CoefArray[length - 1, 1];
        }
        return Mathf.Lerp(CoefArray[min, 1], CoefArray[max, 1], Mathf.Clamp((x -
            CoefArray[min, 0]) / (CoefArray[max, 0] - CoefArray[min, 0]), 0, 1));
    }
}
```

Figure 11 Coefficient Object within Child Wing Script

Because every wing requires at least 2 coefficients to function correctly – lift and drag (each consisting of a coefficient value for every 5 degrees, ranging from -180 to 180) – it was logical to use an object to handle the operations of loading from file as part of the object's constructor as well as having a method for performing linear interpolation between points in the array to get the current coefficient.

```
1 reference
void ComputeLift(float speed)
{
    // compute lift coef
    liftCoef = LiftCoefficient.GetCoefficient(angleOfAttackDeg + currentOffset);
    float speedSquared = speed * speed;

    // compute lift equation
    lift = liftCoef * ((density * speedSquared) / 2) * wingSurface;

    // Account for deep stalling at wing roots caused by fuselage blocking air flow
    if (leftSideRoot == true || rightSideRoot == true)
    {
        if (leftSideRoot == true)
        {
            if (sideSlipDeg < 0)
            {
                float mod = sideSlipDeg / 25;
                mod = 1 - mod;
                if (mod < 0) { mod = 0; }
                if (mod > 1) { mod = 1; }
                lift = lift * mod;
            }
        }
        else
        {
            if (sideSlipDeg > 0)
            {
                float mod = sideSlipDeg / 25;
                mod = 1 - mod;
                if (mod < 0) { mod = 0; }
                if (mod > 1) { mod = 1; }
                lift = lift * mod;
            }
        }
    }

    // Calculate direction to apply force in - perpendicular to airflow
    Vector3 dir = transform.TransformDirection(new Vector3(0, 1, 0) * lift);
    Vector3 rotatedVector = Quaternion.AngleAxis(angleOfAttackDeg, axis: transform.right) * dir;

    parentRB.AddForceAtPosition(rotatedVector, transform.position, ForceMode.Force);

    // Output information to console window if debug is enabled for this wing section
    if (debug == true)...
}
```

Figure 12 Compute Lift Method of Finalised Wing Script

This method applies the appropriate amount of lift for the aircraft's current speed, altitude, angle of attack, etc. based on the lift equation discussed in section 2.5. Physics of Flight. Rather than applying the force directly upward or downwards relative to the wing surface as in earlier prototypes of the Compute Lift method, the force needed to be applied perpendicular to the incoming airflow. Doing so improves adverse yaw and close-to-stall behaviour remarkably. In addition, each wing object makes its calculations based on the local velocity of that particular wing section. For example, if the aircraft is yawing right violently, for example during the start of a stall, the left wing will be moving forwards faster than the right, exacerbating the yawing motion and simultaneously causing wing drop, important for the achievement of objective 5.

```
void ComputeDrag (Vector3 localVelocity)
{
    // Get drag coefficient
    float absAOTD = Mathf.Abs(angleOfAttackDeg);
    float absCO = Mathf.Abs(currentOffset) / 2;
    float absSS = Mathf.Abs(sideSlipDeg);
    if (absSS > absAOTD)
    {
        dragCoef = DragCoefficient.GetCoefficient(sideSlipDeg + absCO);
    }
    else
    {
        dragCoef = DragCoefficient.GetCoefficient((absAOTD * 0.98f) + (absSS / 10) + absCO);
    }

    float speed = localVelocity.magnitude;

    // Compute drag equation
    drag = dragCoef * (( density * (speed * speed)) / 2) * wingSurface;
    drag = Mathf.Abs(drag);
    Vector3 dir = transform.TransformDirection(localVelocity);
    dir = -dir.normalized;
    dir = dir * drag;

    // Output information to console window if debug is enabled for this wing section
    if (debug == true)...

    // Add force in opposite direction to incoming airflow
    parentRB.AddForceAtPosition(dir, transform.position, ForceMode.Force);
}
```

Figure 13 Compute Drag Method of Finalised Wing Script

The Compute Drag method applies drag forces in the opposite direction to the airflow, based on the drag equation discussed previously using either the current angle of attack or current side slip for getting the coefficient depending on which is largest, though more often than not this is the angle of attack.

4.3.4. Winch Script

The script in charge of the winch launching of the aircraft is relatively straightforward:

```
void FixedUpdate()
{
    if (connected == true)
    {
        lineRenderer.SetPosition(0, transform.position);
        lineRenderer.SetPosition(1, targetTransform.position);
        Vector3 forceDirection = (targetTransform.position - transform.position).normalized;

        if (Input.GetButton("Fire1") == true && connected == true)
        {
            reeling = true;
        }
        if (reeling == true)
        {
            targetRigidBody.AddForceAtPosition(forceDirection * -winchForce * 2, targetTransform.position, ForceMode.Force);
            Debug.DrawLine(targetTransform.position, targetTransform.position + forceDirection.normalized * 10.0f, Color.blue);
            //rigidBody.AddForce(forceDirection * winchForce * 2);
            if (Input.GetButton("Fire1") == false)
            {
                reeling = false;
                connected = false;
            }
        }
    }
    else
    {
        lineRenderer.enabled = false;
    }

}
```

Figure 14 Winch Script Update Method

This script renders the cable and pulls the aircraft towards it's self with a constant force, at a pre-specified point while left click is held, this point can be set as a game object placed under the belly of the aircraft to simulate the winch attachment point. A possible improvement to this design would be to have the power increase the longer it is pulling the aircraft, to simulate standard launching procedure (Piggot, 1990). For the same reason it would be realistic to have the cable disconnect from the winch machine end if the user doesn't release at the correct time.

4.3.5. Other Scripts

Various other scripts were created for specific purposes, such as a Camera Controller script which handles camera movement and rotation as well as switching between third and first person views. When in third person view the camera moves back and forth slightly depending on the current speed of the aircraft to give the user some visual feedback.

Airflow Effector is a simple script which has the express purpose of storing data of local airflow – this is used to simulate the presence of thermals, updraft due to terrain, etc. The vector that is stores is used by the Master Script which is then passed on to all the Child Wing Scripts as described previously

A simplified version of an engine script was added as prototyping for objective 7. It works by adding constant forward force at its location as well as some simulation of torque, based on the current throttle setting. However this was not sufficiently realistic for having a working powered aircraft with any training value as there are a whole host of effects that a propeller has on the aircraft other than thrust and propeller torque. For example the sections of wing immediately behind the propeller all experience an increase in airflow over them depending on the throttle setting of the engine, and this needs to be modelled as it noticeably changes the handling characteristics of the craft.

5. Evaluation

5.1. Project Achievements

Determining the exact level of “realism” within the simulation has been done by comparing statistics of both individual elements within the flight model such as the lift and drag produced under certain circumstances, compared to the same results one might find by inputting the same circumstances into mathematical models of fluid dynamics. Additionally, the overall “feel” and characteristics of the plane being simulated can be compared to the plane on which it is based and the stated characteristics within its flight manual as well as general behaviours described in various gliding manuals.

5.1.1. Altitude vs Speed

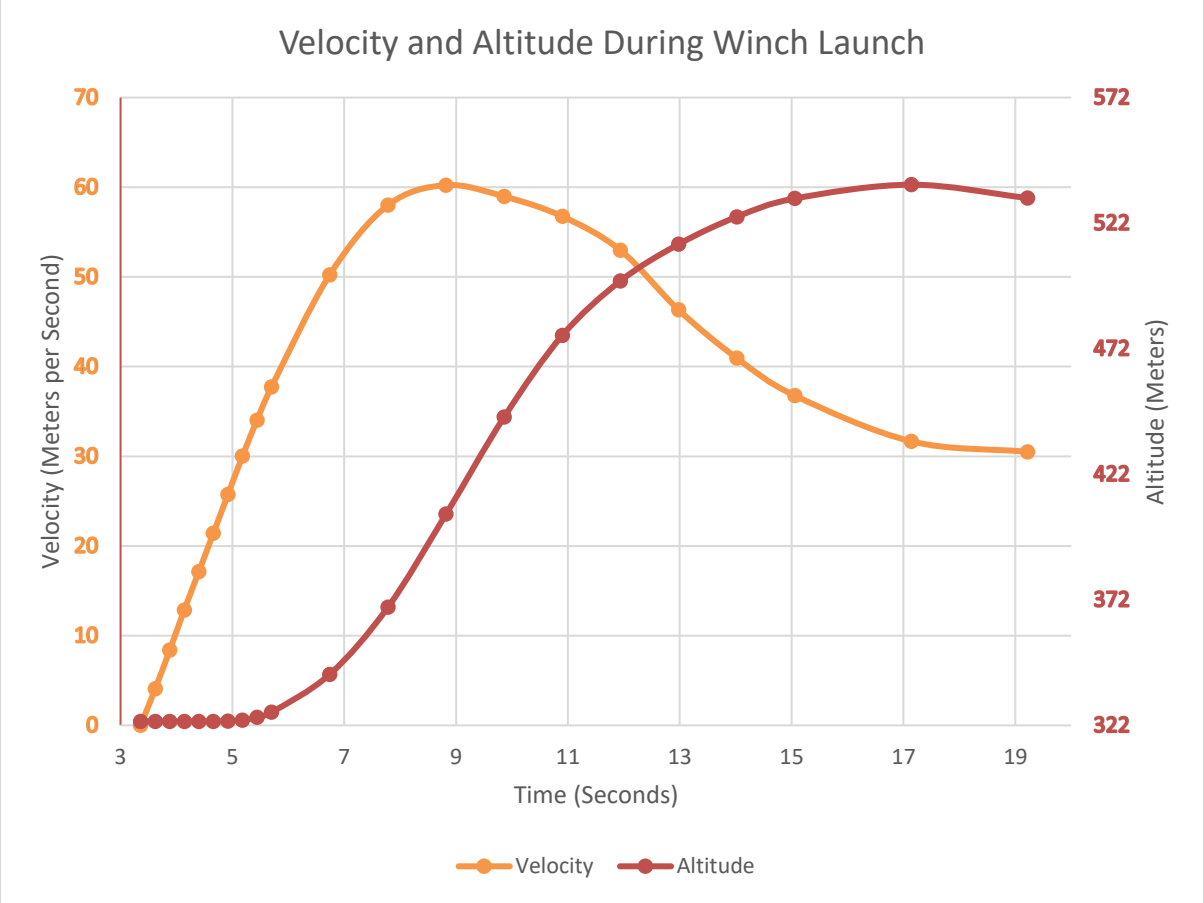


Figure 15 Velocity and Altitude during Winch Launch

In figure 15 we see the velocity and altitude of the glider changing over time during a winch launch from the runway. The cable is released between the 12 and 13 seconds mark. The results are satisfactory – though the cable is quite short, with the winch standing merely 477 meters at the beginning of the simulation – the height gained during the launch is sufficiently accurate to the general rule of thumb of the cable being released at a height of about 35% of its initial length (Piggot, 1990). At the point of cable release the craft has climbed about 180 meters, compared with the cable length this is approximately 37%, this is in line with expectations. This shows achievement of the take-off part of objective 6.

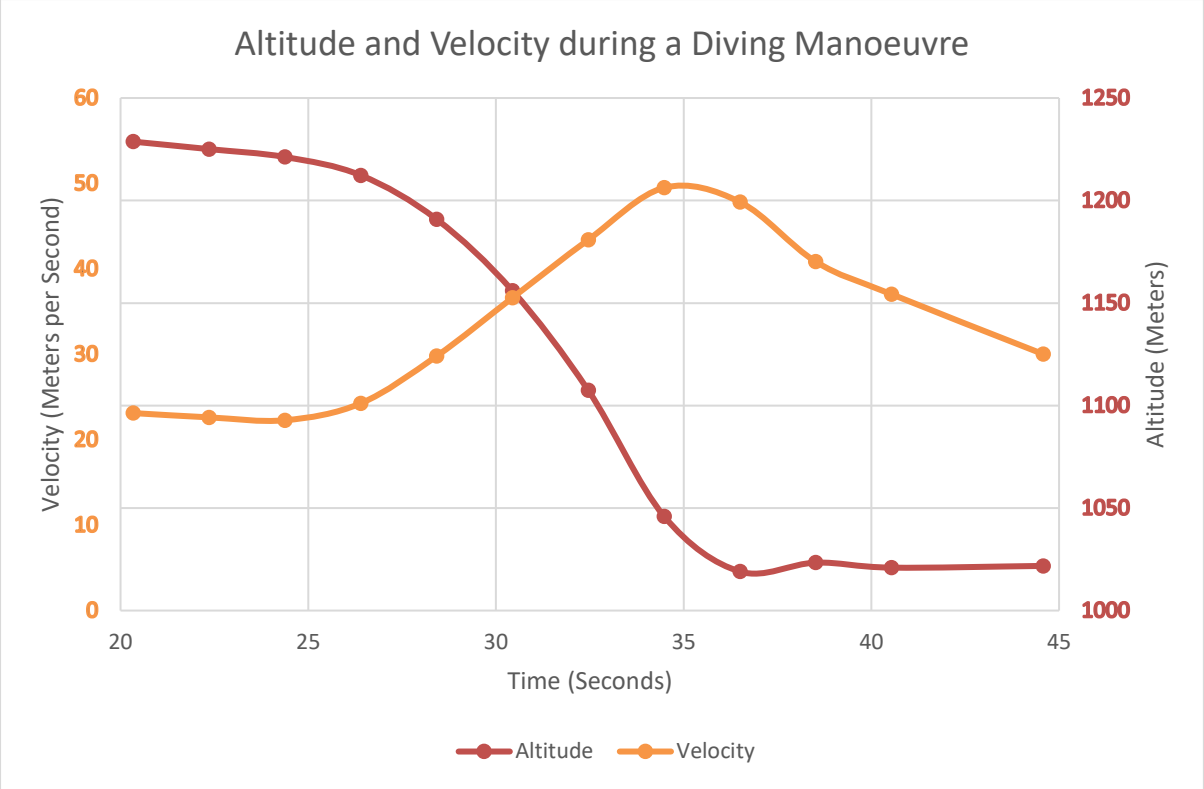


Figure 16 Altitude and Velocity during a Diving Manoeuvre

Figure 16 demonstrates the principle of energy transfer being present within the simulation. As the craft begins slowly pitching down around the 24 second mark it begins to trade altitude for speed. Then at the 35 second mark the craft is pitched up again and speed is lost as a result of this manoeuvre, shortly then after the aircraft climbs slightly, trading speed for altitude before continuing to slowly lose speed as a result of drag as it goes into level flight. This partly shows success of objectives 3 and 4.

5.1.2. Main Wing Behaviour during Stall

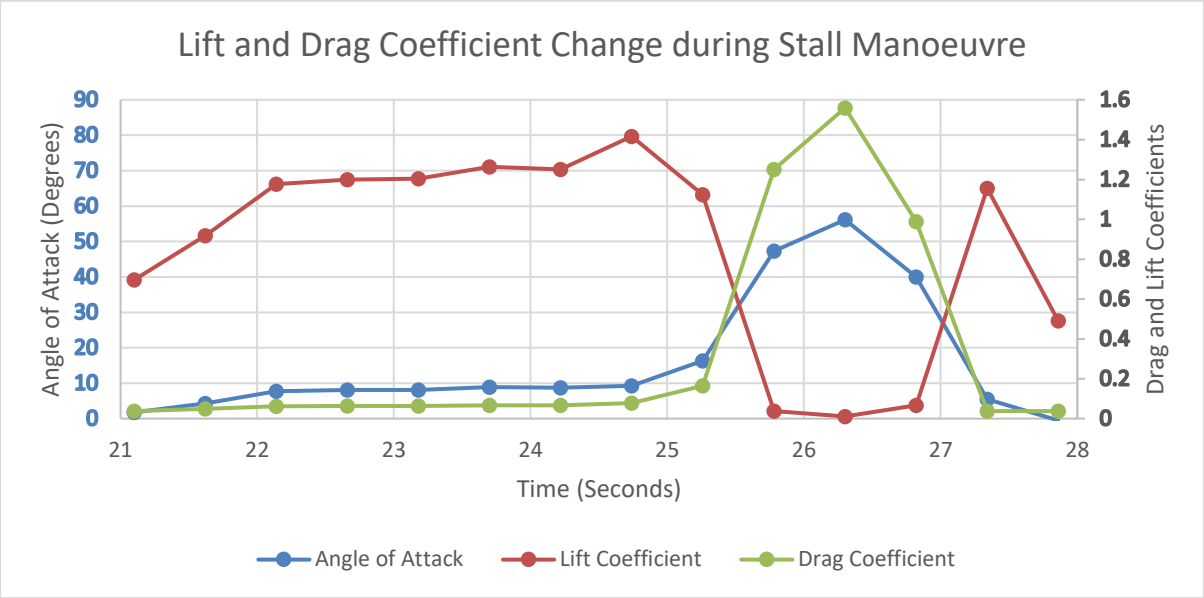


Figure 17 Lift and Drag Coefficient Change during Stall Manoeuvre

The chart above shows the angle of attack's effect on the lift and drag coefficients of a section of wing close to the tip during a stall. It shows how the lift coefficient increases as the angle of attack increases, up to around 14 to 15 degrees where it then drops off dramatically until the wing is in a complete stall – that is, producing no lift. When this is compared with figures 1 and 2 (Section Lift Coefficient of Aerofoil Wing and Section Drag Coefficient of Aerofoil Wing), in the Background section of this report, we see a more than satisfactory relationship between angle of attack and the lift and drag coefficients. Any discrepancies can be easily fixed through simple tweaking of the text files from which the coefficients are loaded.

In the following graph we see how these coefficients manifest themselves as forces, applied in newtons. These are compared with the velocity of the craft, both the forward speed and overall velocity, to show how speed also plays an important part in determining the magnitude of the force. These characteristics, along with a few others, contribute greatly to the completion of project objectives 3 and 4.

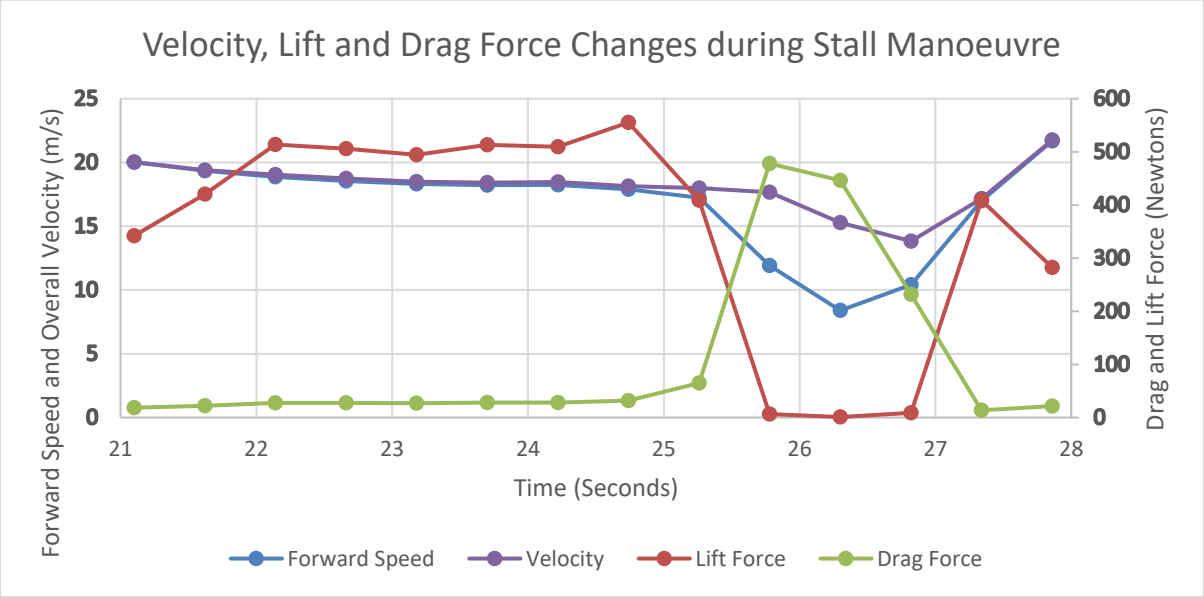


Figure 18 Velocity, Lift and Drag Force Changes during Stall Manoeuvre

Effects of Local Velocity during Stall.

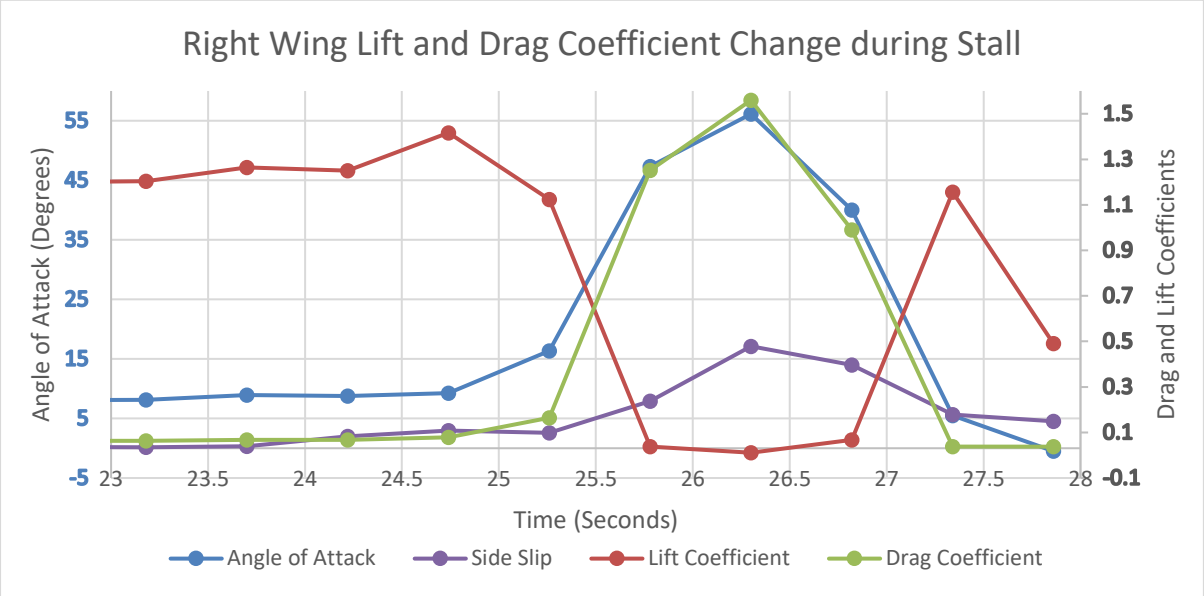


Figure 19 Right Wing Lift and Drag Coefficient Changed during Stall

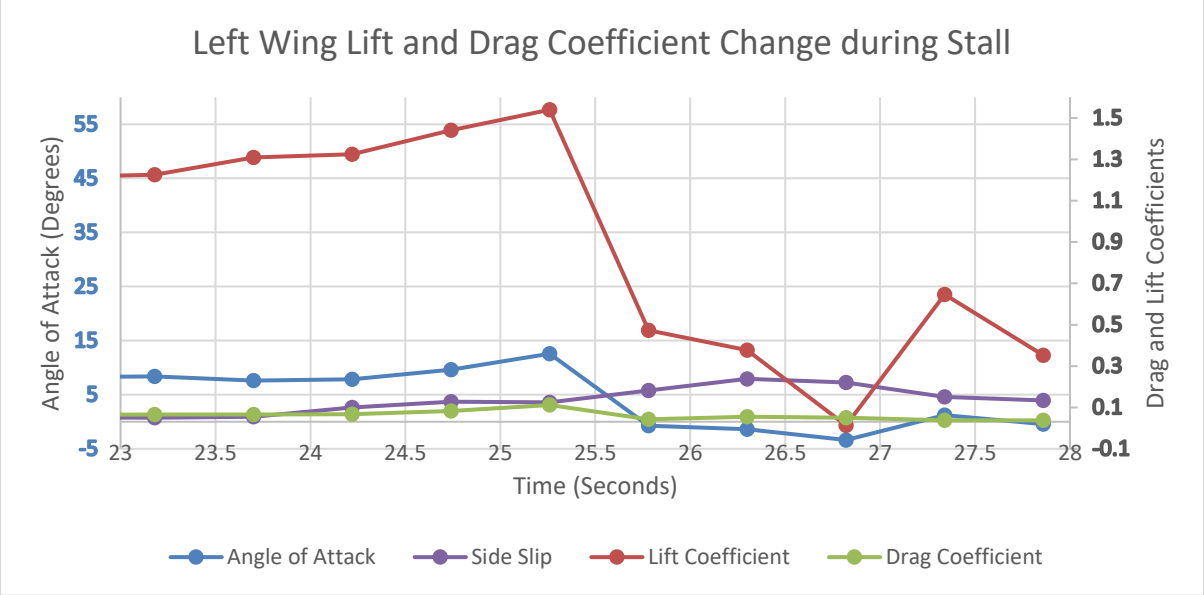


Figure 20 Left Wing Lift and Drag Coefficient Change during Stall

In figures 19 and 20 we see the effect of using velocity local to the particular wing rather than the craft's overall velocity. The data for both these graphs was collected from the same flight as previously shown in figures 17 and 18, with figure 19 showing the right wing's local variables with the addition of side slip and figure 20 the left wing. While the right wing tip goes into a deep stall, experiencing up to 55 degrees angle of attack as it drops, the left wing tip in fact moves upwards slightly for a brief moment, experiencing negative angle of attack, and not stalling at all, the reason for the drop in lift coefficient is the angle of attack going briefly into the negative (wind flowing downwards onto the wing). In motion this takes the form of wing drop. Adding left stick control to try to counter the rolling motion of the craft would have caused the right wing tip to stall for even longer. In regards to objective 5 this can be considered very successful behaviour.

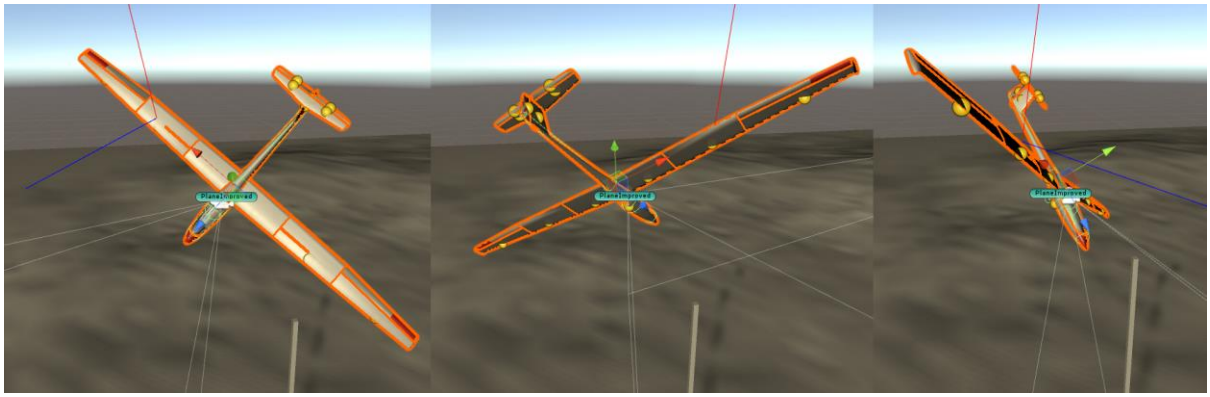


Figure 21 Aircraft in a Stall Spin / Spiral

Figure 21 depicts the glider experiencing autorotation in a stall spin or spiral, due to the user holding full elevator in an attempt to pitch up. The red line indicates the direction in which drag is being applied for the mid-section of the right wing (this is always opposite the direction of movement) and so shows us that the craft is falling. The blue line coming from the wing indicates the direction in which lift force is being applied – though because the wing is deep in a stall this force is actually negligible. This autorotation and stall spiralling is further evidence of the achievement of objectives 3 and 4, and by extension objective 5.

5.1.3. Environmental Effects on Aircraft Behaviour

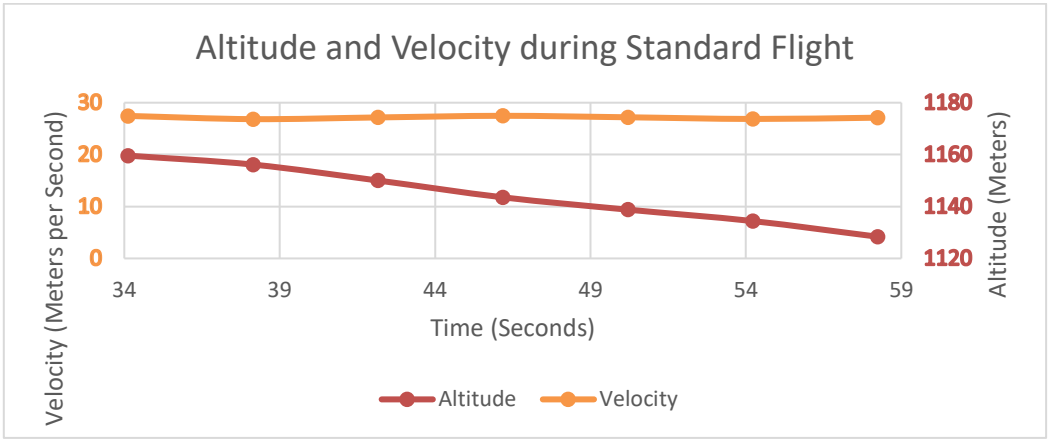


Figure 22 Altitude and Velocity during Standard Flight

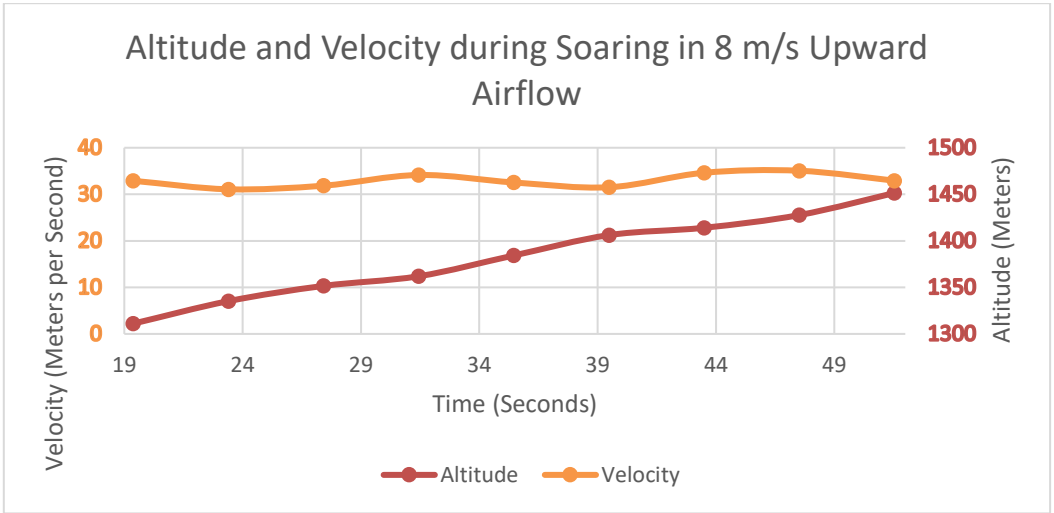


Figure 23 Altitude and Velocity during Soaring in 8 m/s Upward Airflow

Figure 22 is the altitude and velocity during a standard, near level flight. It shows a 1.295 meters per second rate of descent while at an altitude of 1000+ meters. In figure 23 we see the altitude and velocity while circling slowly within a thermal, with an upward airflow of 8 meters per second. We see that the craft maintains almost constant speed while climbing significantly. The overall rate of climb is around 4.4 meters per second which is well in line with rates discovered during background research and as such this can be considered a successful contribution to objective 6.

5.2. Further Work

As mentioned previously, a script for simulating the power of a propeller engine was prototyped. The next logical step for this project would be the full implementation of some powered aircraft which currently are present within the project, but prove lacklustre in their realism.

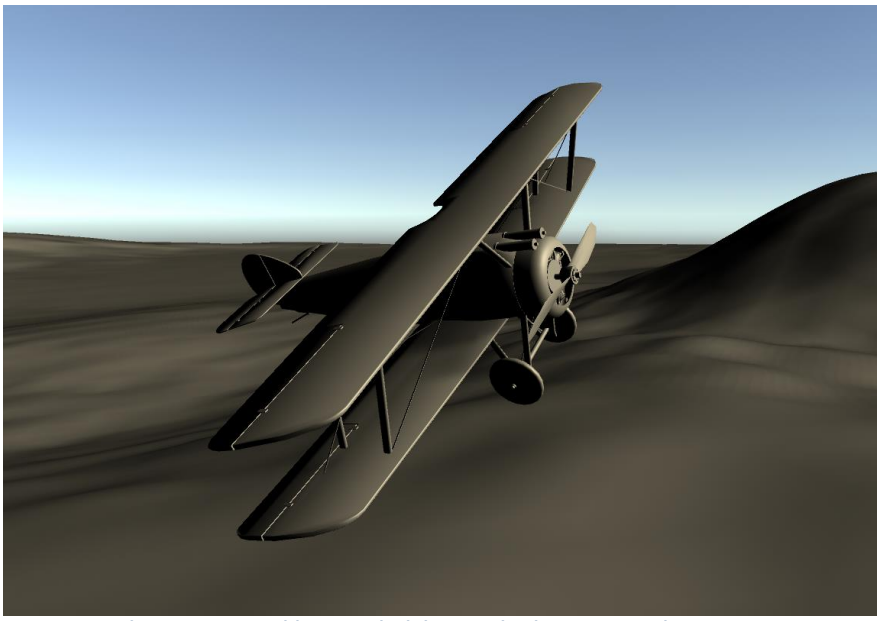


Figure 24 Sopwith Camel Flying Under its Own Engine Power

Landing gear control is a fairly important feature that has not been implemented due to time constraints. Similarly, airbrakes should also be added – both of these features should have their own effects on the aerodynamics of the plane as well as visibly moving on the model when they are actuated. Currently a landing gear is partially simulated – the craft collides with the ground as if there is a wheel present below the fuselage, however this wheel generates no drag and is always ‘deployed’.

In its current state the simulator lacks any sound effects whatsoever. These are of some importance as they can provide pilots with good indication of what the aircraft is currently doing, such as when a wing is beginning to stall.

In addition, the terrain and overall visual fidelity of the simulation could use some work. Features such as trees and texturing of the terrain would not go amiss.

6. Conclusion

Overall a working and useable solution, fit for purpose, has been developed. The majority of objectives have been met with a generally high degree of success. The user can expect to be able to launch his aircraft, make an extended duration flight with the possibility of taking advantage of thermals and then come around for final approach and landing.

However there are various areas that are incomplete or not implemented, particularly with regard to objective 6 – there is no drag modelled for the landing gear although it is possible to land on what is essentially an invisible wheel. Additionally the lack of airbrakes is problematic to being able to use the program as a genuine training device.

The viability of the project is thoroughly proven though, and with additional work it could become a competent training device to be used by total novice pilots.

Appendix A: Grob-102-Astir Technical Data

Glider Flight Manual GROB G102

1.6. Description

The CLUB ASTIR III and IIIb a single seat performance glider for the club class with a T-tail and airbrakes on the upper wing surface. The STANDARD ASTIR III is the equivalent high performance glider for the standard class, with retracting undercarriage and ballast tanks in the wings. The glider incorporates the most modern fibre reinforced plastic technology. The fuselage stringers are fabricated from Carbon fibre; all other surfaces and shells are glassfibre,

Technical Data

Wingspan	15,0 m	(49,2 ft)
Length	6,75 m	(2291 ft)
Height	1,26 m	('4,1 ft)
Aspect ratio	1892	(18,2)
Wing area	12,4 m 2	(133,5 sq.ft.)
Maximum flying weight		
with waterballast	450 kg	(992 lbs)
without waterballast	380 kg	(838 lbs)

Maximum wing loading 36,3 kg/m 2 (7,4 lbs/sq.ft)

December 6, 1982 Approved by LBA

II. operating limitations

II.1 Category of airworthiness:

U (Utility) according to JAR 22

Certification Basis: 14 CFR Sections 21.23 and 21.29 effective 1 February 1965; and Joint Airworthiness Requirements for Sailplanes and Powered Sailplanes (JAR-22), dated 1 April 1980.

II.2 Permitted operations:

1. VFR day
2. *Simple aerobatics (loop, stall turn, lazy eight, chandelle, spin)

*NOTE: ASA does not permit aerobatics in club ships

II.3 Minimum equipment

1. Air speed indicator reading to 300 km/h (162 knots, 187 mph)
2. Altimeter
3. Four part safety harness
4. Back cushion of at least 3" depth when compressed, or parachute
5. Loading limit placard
6. Flight limits placard
7. Flight Manual

II.4 Airspeed limitations

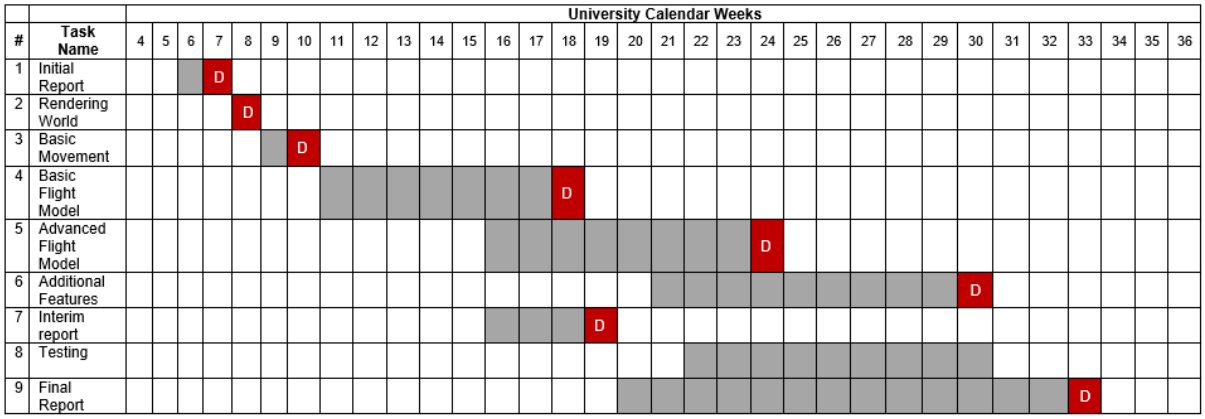
Never exceed	VNE 250 km/h	(135 kts,155mph)
Maximum Rough Air	VB 250 km/h	(135 kts,155mph)
Manoeuvring speed	VM 170 km/h	(92 kts,105mph)
Maximum on winch launch	VW 120 km/h	(65 kts, 74mph)
Maximum on aerotow	VT 170 km/h	(92 kts,105mph)
Maximum for operating landing gear, and L.G. extended	VT 250 km/h	(135 kts,155mph)

(Grob Aircraft, 1982)

Appendix B: Risk Analysis

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Data loss	H	M	HM	Keep Backups	Reinstate from backups
Loss of backups	H	L	HL	Multiple Backups	Use alternate
Environment crash during data gathering	L	M	LM	Ensure good level of testing throughout the whole project development	Restart test session – Fix bug that caused crash if possible
Unity update breaks code	H	L	HL	Ensure code is written clearly, making it easier to fix if any functions that may need changing	Rewrite effected code
Unity not launching	H	L	HL	Ensue Unity engine on own system has all necessary files to run	Re-install Unity engine
Hardware limitations	M	L	ML	Only use systems with known specifications (E.G in university)	Switch machine

Appendix C: Time Plan



References

Allerton, D., 2009. *Principles of Flight Simulation.* s.l.:AIAA.

Bradbury, T., 2004. *Meteorology and Flight: Pilot's Guide to Weather.* 3rd ed. s.l.:Bloomsbury Publishing Plc.

Eckey, B., 2009. *Advanced Soaring Made Easy: Success is a Journey - Not a Destination.* 2nd ed. s.l.:Bernard Eckey.

Experimental Aircraft Info, 2006. *Lift & Stall Effects.* [Online]
Available at: https://www.experimentalaircraft.info
[Accessed 17 04 2018].

Grob Aircraft, 1982. *Glider Flight Manual GROB G 102.* s.l.:s.n.

Hage, R. & Courtland, P., 1965. *Airplane Performance, Stability and Control.* s.l.:Wiley.

Havok, 2017. *About Havok.* [Online]
Available at: https://www.havok.com
[Accessed 28 December 2017].

Johnson, D., 2010. *Lockheed Martin Announced Prepar3D, archived from the original at www.lockheedmartin.com.* [Online]
Available at:
https://web.archive.org/web/20100713125615/http://www.lockheedmartin.com/news/press_releases/2010/05172010Prepar3d.html
[Accessed 18 January 2018].

Mendip Gliding Club, n.d. *Launching Methods.* [Online]
Available at: www.mendipgliding.co.uk
[Accessed 25 April 2018].

NASA, 2015. *The Drag Equation: Glenn Reserach Center.* [Online]
Available at: https://www.grc.nasa.gov
[Accessed 20 January 2018].

National Physical Laboratory, 2010. *FAQ - Mass & Density.* [Online]
Available at: www.npl.co.uk
[Accessed 19 January 2018].

Piggot, D., 1990. *Gliding: A handbook on soaring flight.* 6th ed. London: A & C Black (Publishers) Ltd.

Rolfe, J. & Caro, P., 1982. Determining the training effectiveness of flight simulators: some basic issues and practical development. *Applied Egonomics,* 13(4), pp. 243-250.

Rolfe, J. & Staples, K., 1986. *Cambridge Aerospace Series: Flight Simulation.* Cambridge: Cambridge University Press.

Scott, J., 2006. *NACA 0012 Lift Characteristics.* [Online]
Available at: www.aerospaceweb.org
[Accessed 20 January 2018].

Shih, C., Van Dommelen, L. & Krothapalli, A., 1992. Unsteady flow past an airfoil pitching at a constant rate.. *AIAA Journal,* Volume 30.

SKYbrary, 2017. *Energy Management during Approach.* [Online]
Available at: https://www.skybrary.aero
[Accessed 06 05 2018].

Smetana, F. O., 2001. *Flight Vehicle Performance and Aerodynamic Control.* Ohio: American Institute of Aeronautics and Astronautics, Inc..

Smithsonian National Air and Space Museum, n.d. *The Four Forces.* [Online]
Available at: http://howthingsfly.si.edu
[Accessed 19 January 2018].

Torenbeek, E. & Wittenberg, H., 2009. *Flight Physics: Essentials of Aeronautical Disciplines and Technology, with Historical Notes.* 2009 ed. s.l.:Springer.

U.S Department of Health & Human Services, n.d. *Usability Testing.* [Online]
Available at: https://www.usability.gov
[Accessed 01 04 2018].

U.S Navy, n.d. *Naval Air Systems Command.* [Online]
Available at: www.navair.navy.mil
[Accessed 18 January 2018].

Unity Technologies, 2017. *Unity Manual.* [Online]
Available at: https://docs.unity3d.com
[Accessed 13 January 2018].

Vitsas, P. A., 2016. *Yaw due to roll rate derivative: aeroscience.* [Online]
Available at: https://aeroscience.wordpress.com
[Accessed 17 04 2018].