OLANdroid

Final report for CS39440 Major Project

Author: Gideon MW Jones (gij2@aber.ac.uk)

Supervisor: Neal Snooke ([nns@aber.ac.uk](mailto:nns@aber.ac.uk))

Date (fixed)

Version 0.1 (draft)

This report is submitted as partial fulfilment of a BSc degree in Computer Graphics, Vision and Games (G451)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

In signing below, I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the sections on unfair practice in the Students’ Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
* I understand and agree to abide by the University’s regulations governing these issues.

Signature ……………………………………………. (Gideon MW Jones)

Date …………………………………………………

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ……………………………………………. (Gideon MW Jones)

Date …………………………………………………

Acknowledgements

I am grateful to… I’d like to thank…

Abstract

The acrobatic flight of aeroplanes

Table of Contents

[Background, Analysis & Process 6](#__RefHeading__2068_1315851627)

[Background 6](#__RefHeading__2070_1315851627)

[Analysis 6](#__RefHeading__2072_1315851627)

[Process 6](#__RefHeading__2074_1315851627)

[Design 7](#__RefHeading__2076_1315851627)

[Overall Architecture 7](#__RefHeading__2078_1315851627)

[Detailed Design 7](#__RefHeading__2080_1315851627)

[Even More Detail 7](#__RefHeading__2082_1315851627)

[User Interface Design 7](#__RefHeading__2084_1315851627)

[Other Relevant Sections 7](#__RefHeading__2086_1315851627)

[Implementation 8](#__RefHeading__2088_1315851627)

[Testing 9](#__RefHeading__2090_1315851627)

[Overall Approach to Testing 9](#__RefHeading__2092_1315851627)

[Automated Testing 9](#__RefHeading__2094_1315851627)

[Unit Tests 9](#__RefHeading__2096_1315851627)

[User Interface Testing 9](#__RefHeading__2098_1315851627)

[Stress Testing 9](#__RefHeading__2100_1315851627)

[Other Types of Testing 9](#__RefHeading__2102_1315851627)

[Integration Testing 9](#__RefHeading__2104_1315851627)

[User Testing 9](#__RefHeading__2106_1315851627)

[Critical Evaluation 10](#__RefHeading__2108_1315851627)

[Appendices 11](#__RefHeading__2110_1315851627)

[Third-Party Code and Libraries 11](#__RefHeading__2112_1315851627)

[Code Samples 12](#__RefHeading__2114_1315851627)

[Annotated Bibliography 13](#__RefHeading__2116_1315851627)

# Background, Analysis & Process

## Aerobatic flight

The acrobatic flight of aeroplanes is the practice of flying in patterns not commonly used in flight. There's a number of different motivations behind this, including recreation, entertainment, sport and training. A range of flying vehicles can participate in aerobatics, while primarily aeroplanes are used, gliders and helicopters can also perform, the latter with its own more specific set of movements available.

Model aircraft – of the powered and controllable variety – are quite commonly used for performing aerobatics in the hands of hobbyists. They come in many different shapes and sizes, with different flight characteristics. The smaller nature of these aircraft means that forces exerted on them are far less significant than those on a full-sized plane, and that there's no worry about the *in situ* G-force a pilot would have to endure. As a result, far more intense flights can be undertaken with model aircraft than full-sized ones, though that doesn't rule them out from being able to perform as well.

An aerobatic flight can be broken down into manoeuvres, each representing a pattern of movement for the aircraft to follow. An commonly known example manoeuvre would be a “loop-the-loop”, consisting of flying a plane round in a vertical loop. A wide range of these manoeuvres exists, describing many different patterns involving movements on every axis – pitch, yaw and roll – as well as even backwards movement (tailslides). Manoeuvres can be described in two main different ways – the Aresti and OLAN notations.

The Aresti system was developed by Spanish aerobatic pilot José Louis Aresti Aguirre, and is now adopted by the Fédération Aéronautique Internationale (FAI) as a standard. It represents manoeuvres using various diagrams and numbers (catalogue references). The prior is pretty simple to understand, but the latter relies on an official catalogue which is only available in a physical format for a price.

Alternatively, OLAN (One Letter Aerobatic Notation), developed by Michael Golan, provides a more concise and independent method of representing manoeuvres. It reduces manoeuvres instead to a simple letter description, with several modifiers available, such as extended entry/exit and scaling to allow for more complex flights.

## Problem definition

Within the realm of this subject, the task this project is undertaking is the visual modelling of acrobatic flight. This is only a small section of a field which has many different modellable variables. The visualisation provided needs to effectively demonstrate the movement of an aircraft during the flight. Users also need to be able to define these flights themselves, using an intuitive, simple system.

This project aims to go a bit further than that definition of the task, as it's quite a wide description. Concision is added by focusing the brief with explicitly stating the target user group to be people who wish to learn how to fly these flights. This clarification adds some extra constraints – simplicity and intuitiveness – as well as opens the project up to features not necessarily directly tied to the visualisation aspect.

There's a number of reasons for this extension to the task brief. Mainly, this is to overcome the breadth of the original brief. A second reason is to build a more complete experience, by defining what extra features would be valuable in the final product.

## Background and Motivation

Developing a 3D understanding of an aerobatic manoeuvre is a complicated process, involving a good degree of spatial visualisation thinking, imagining the way aircraft move. Beginners are often taught it by another person, with the aid of a stick plane, (essentially a tiny model plane on a stick you can move around to emulate manoeuvres), by studying Aresti diagrams and interpreting them or using online resources like videos. Neither of these methods are perfect or ideal, with either requiring another person, an Aresti catalogue or successfully understanding (potentially unreliable) online resources.

My experience of being taught by someone else, understanding Aresti diagrams (from OpenAero, detailed below) and watching videos of flight simulators was not an ideal learning process. It required detailed studying and use of imagination to be able to build up a coherent model of what the aircraft was doing.

This project aims to provide a functionality useful to people learning to fly these manoeuvres and build up a picture of how the plane moves without the facilities suggested above. Part of the motivation for this was the fact that there were no systems available providing functionality anything like this.

As preparation for embarking on this project, a few system were looked into. Principally OpenAero and OLAN.

OpenAreo is an extensive tool for designing aerobatic sequences built by Ringo Mass. It takes the form of an open-source project web application. Among many more complex features, it primarily provides OLAN interpretation and translation of OLAN into Aresti, with diagrams and catalogue references of manoeuvres. It is not so much a tool for visualising acrobatic flight as a tool for people who already understand the manoeuvres and want to design flights combining them, but it offers some useful hints about the shapes of manoeuvres and the OLAN system.

The OLAN tool is a Windows application developed by Michael Golan and is the point of origin for the OLAN system.

On a personal level, I have an interest in all three of the main components of the application – aviation, 3D graphics and Android development.

My interest in planes stems from spending a lot of time playing flight simulators, an intrigue in aviation and my dad sharing his interest in model planes (Airfix and of the rudimentary flying variety) with me in my youth.

Multiple courses at university leads me to be familiar with rendering 3D graphics, with experience with the fairly high level Java3D and the much lower level OpenGL and GLSL, both of these built knowledge useful for developing applications with 3D rendered elements.

Android development is an area I have spent a lot of time sitting on the sidelines of, with having rooted and installed custom ROMs on many devices, and getting familiar with the OS and the main Android-PC interaction tool – Android debug bridge (adb). Having experience with Java meant the barrier of entry to developing on this new platform was lower than with other alternative technologies.

What was your background preparation for the project? What similar systems did you assess? What was your motivation and interest in this project?

## Platform and Technology

This project is aimed at the Android platform. The decision was made based around several main factors:

The final product of this project needs to be accessible in as many ways as possible. Physically

The other side of the background research was looking into the Android platform and development.

It was first important to ascertain that the platform could support the requirements of the project – the most demanding being the need for rendering 3D graphics. Thankfully, the platform has various different ways of doing this, including OpenGL ES 2.0 (being familiar with WebGL, this was convenient).

Higher level libraries for 3D rendering and scene building were explored, such as {ANDROID GRAPHCS LIBRARIES I PRETENEDED TO LOOK INTO HERE}. However, none were found to offer any real incentives for what will be quite simple planes (surfaces), as compared to the fairly basic direct OpenGL ES and GLSL interaction.

The Android platform posed some different challenges to what one would find when developing as a PC application. The hardware gamut of Android devices is different to that of computers, primarily with greater variation in screen sizes (both physical and resolution) due to tablets. General power was across the board fairly low in comparison to PCs, so that needed to be taken into account.

Performance issues could be somewhat mitigated by restricting access to the application by giving it a minimum API level. Newer devices tend to more power and newer API levels, so an application can sort of be targeted in this way.

The performance/computing power of Android devices tends to be lower than those of PCs, and there is also a much wider range of screen sizes. Both the UI and the complexity of the visualisation should take these factors into account.

This section should discuss your preparation for the project, including background reading, your analysis of the problem and the process or method you have followed to help structure your work. It is likely that you will reuse part of your outline project specification, but at this point in the project you should have more to talk about.

**Notes:**

* **All of the sections and text in this example are for illustration purposes. The main Chapters are a good starting point, but the content and actual sections that you include are likely to be different.**
* **Look at the document on the Structure of the Final Report for additional guidance.**

## Analysis

The project, at a rudimentary level, can be broken down into three main components:

* The internal representation of the manoeuvres.
* The translation of fights descriptions into that model.
* The visual modelling of that flight.

While these three components cover different aspects of the system, it should be noted that they don't necessarily have to be distinct in the actual implementation. A prime candidate for this is cohesion between the representation of manoeuvres and their visualisation component.

### Internal representation of manoeuvres

The first task highlighted by the research was that of developing a representation of the manoeuvres in-application. There are, of course, a number of different ways of doing this, including hard coding every manoeuvre.

The most logical approached was to analyse each manoeuvre and find effectively a set of the highest common factors between them all. As the standard catalogue consists of a lot of different manoeuvres, this is an extensive task, but there are definitely some common features between various movements, for example: a straight line forwards and a turn upwards by a few degrees. In this process, some parameters, such as the aforementioned *few degrees* need to be derived. For example, with a manoeuvre set consisting of a straight up vertical movement, and one at a diagonal climb, a component part's upward turn angle should not be ninety degrees.

Alongside this, another angle which could be taken was interpretation manoeuvres in the context of being flown by an aeroplane. In this context, it is obvious that manoeuvres can only exist if they can actually be performed, and thus the movements of a manoeuvre are limited by the mechanics of flight. From this, we can distil that that components of the manoeuvre can only move in the ways a plane can move – a plane cannot suddenly switch from flying straight flying upside-down – so a series of small sections can be built up, using operations such as a realistic change in pitch, yaw, roll and movement forwards (or backwards, in some cases). A component can then be considered to be the plane's velocity at a point.

Further on this, each piece of the aircraft's movement and progression through a manoeuvre is reliant on its previous position and orientation. Any set of simple components, representing an aircraft's movement, needs to be a cumulative progression.

However, this cannot be completely accurate, as the ability for an aircraft influence its position and orientation while flying straight line forwards is different to doing the same thing but upside down. The influence of both gravity and the thrust generated by the wing are two large factors which need to be considered, or ignored. Other factors worth thinking about how movement in each axes is different – generally, aircraft can adjust their yaw to less effect than its pitch and roll – and how the characteristics various aircraft differ, such as how much influence its ailerons exert on its direction – how quickly it can turn etc.

These issues can probably be largely ignored through, as the aim is not to build a realistic flight simulation – a much more complex task, of which the issues noted above barely scratch the surface.

It is an important to recognise that the manoeuvres in the manoeuvre catalogue are data, and thus should not be hard-coded into the application, and should be stored separately, as a resource data file. Making this catalogue file recognisable – using a fairly standard data format such as XML – allows for simple extensibility and allows for building on the catalogue and adding extra manoeuvres.

Having this internal representation of all of the manoeuvres available is reminiscent of the Aresti catalogue. Making this available to users and a resource for building flights is a sensible feature idea.

### Flight description interpretation

The interpretation of the flight description relies first on deciding which flight encoding method to use – Aresti, OLAN or something new. After a bit of research, including looking on {THATMODELFLYINGSITE}, some ideas of the pros and cons of each system were developed. Weighing these up is important:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Aresti** | **OLAN** |
| Brevity | Diagrams or numbered catalogue references | Letters with various modifiers |
| Familiarity to users | Standardised by the FAI | Fairly new system |
| Openness | Catalogue must be bought | An open system |

Despite the advantages Aresti offers, the positive aspects of the OLAN system are more relevant to the project, thus the decision was made to go with that system. The ease of providing user input – simple strings of characters, instead of diagrams/catalogue references – and open availability were strong factors in this decision. A compromise can however be struck, by providing the Aresti catalogue references alongside the OLAN figures (an incomplete set can be found via OpenAero) in the manoeuvre catalogue, and therefore allowing users to find manoeuvres by Aresti as well as OLAN.

Parsing this string into the program's internal representation of a flight is fairly simple task, despite the fact that there are quite a few different modifiers available in the system. Figure shows some example OLAN with modifiers.

ta ia

d vd v z 1 imm

6% ++```id+ 3 v``

Figure 1: example flights encoded using the OLAN system

(\\d)% ([\\+,-]\*)([`]\*)(\\w\*)([`]\*)([\\+,-]\*)

{number, full scaling}% {pluses, entry length}{backticks, inverse first group scaling}{OLAN figure}{backticks, inverse second group scaling}{pluses, exit length}

Figure 1: regular expression for extracting parts and an explanation of the OLAN system

In rough, an OLAN figure can be explained with:

Most of the OLAN figures can be quite easily derived by splitting the string up into each figure and using regular expression groups to extract the various components of the manoeuvre (if the modifiers exist). Figures can be split up with spaces. However, there exists the case of full manoeuvre scaling, which is separate from the figure, so this needs to be handled slightly differently.

The representation of a manoeuvre needs support for these modifiers. Considering the representation suggested above, some tasks are fairly trivial – full feature scaling and entry/exit scaling. On the other hand, the group scaling is a bit more complex, with certain parts of the manoeuvre being scaled. These groups vary between manoeuvre, and really needs to be tied into their representation.

### Visual modelling of a flight

The visualisation aspect of the project is arguably the most complex part, as it's not really constrained by any hard and fast rules, like previous components. It allows room for creativity, and can be done in a number of different ways. Most importantly, the way the visualisation expresses the flight should be easily understood and offer value for its performance demands (there's little point in having something pretty which is unclear to the user).

Looking at an aircraft flying an aerobatic procedure, as stated before, does of course offer some value in terms of educating viewers on how to do the manoeuvre. This can be replicated in the application as a method of visualising the flight. In the real world, there isn't many more options, other than a technique occasionally used by stunt pilots – smoke trails. These leave a – comparatively – lasting impression of the tracks of the plane and, while they're not used to bring about an understanding of the manoeuvres, they can be a useful point of inspiration for developing a visual solution.

There is no real reason why the visualisation should go for a realistic look, as it's not bounded by the same rules. {TRY AND FIT A MATRIX QUOTE IN HERE}. Instead, a more abstract approach can be taken. Using the level of the aircraft's wing as a (surface) plane, a full flight can be thought of as a ribbon, with the area behind the aircraft that its been through filled in, similar to the smoke trails. There is an issue that this ribbon may be too abstract, and with no reference points in a 3D space, it can be difficult to understand. Adding a reference (surface) plane would be a good way to tackle this issue. There also exists the problem of a fully completed flight potentially being tight and complex, and thus hard to follow, running into issues like not being able to tell which direction it is flown etc.

Animation is a tool which can be used to combat these issues. This is a logical idea, considering that it harks back to the subject matter – (most) planes are continually in a state of motion during flight. Showing the progression of the flight from the beginning to the end is a good way of avoiding becoming too abstract, and makes sense to users. There are, of course different ways of doing this too, but following on from the smoke trail ribbon idea explored above is a sensible method. Having a full flight drawn behind the aircraft as it progresses through the flight is an option – smoke ribbon lasting forever – and so is having a flying wing look – smoke ribbon only lasting so long.

As a 3D rendering is derived from a 3D environment, it would also be beneficial to offer some control to the user to be able to manipulate the perspective. This can come in the format of rotation, translation and zooming. This would vastly improve the visualisation of the flight, giving it a truly 3D feel and allowing the user to build a better comprehension of the manoeuvres.

Changing the position and orientation of the view will require a user input. Due to the platform of choice being Android, the application will be running on mainly touch screen devices. This constraints the inputs available, and possibly needs to be considered when developing these ideas.

Any implementation of manoeuvres needs a related drawable part which supports the style of visualisation put forward.

how the visualisation is going to look, stylistically

exploring the flight in 3D

GIVING USERS

DO NOT GO TOO TECHNICAL IMPLEMENTATION DETAILS

### User Experience Features

Besides the basic core of the project, features which make it more accessible and usable are important for ensuring that it is actually a useful application.

The vast majority of accessibility and usability based features make up part of the user interface in general. This UI should be logical and intuitive, so users are never confused and can predict with accuracy what each UI element will do when used. Another way of improving this accessibility is to provide a series a set of documentation, accessible in-app, which describes how to use the UI. This is an unobtrusive way of explaining features to new users, but allowing more experienced users to work uninhibited.

Useful user experience focused features.

Having a store of previously written flights is a highly beneficial feature to the user, preventing them having to rewrite flights every time they open the application. Being able to edit them by tweaking the OLAN encourages the development of manoeuvre sequences.

There are various parts of the project which do not have a clear, right answer. In some cases, it is true that the user might find one of a few different options useful – things like whether or not to correct potentially invalid OLAN, a speed for the animation etc. A solution to this is providing a settings menu for users to selection their preferences.

Prioritising features

Features we need vs features we like

functional features, for the core of the program, vs features for usability – actual visualisation vs things like flight saving etc.

Taking into account the problem and what you learned from the background work, what was your analysis of the problem? How did your analysis help to decompose the problem into the main tasks that you would undertake? Were there alternative approaches? Why did you choose one approach compared to the alternatives?

There should be a clear statement of the objectives of the work, which you will evaluate at the end of the work.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

## Process

The process chosen for the development of this project took into account attributes of both the project's technology and its developer.

### Preparation

Before any development work could begin,

### Supervision

While it was initially proposed that there would be weekly meetings with the client (supervisor) to discuss progress that week and clarify the next round of features, this was deemed unnecessary after a few weeks. Due to the level of up-front planning and progress thus far, the client was satisfied with the interpretation of the task, as well as the implementation of the work already carried out, and so was confident that less frequent meetings were sufficient.

### Iteration

Evolutionary design

Not TDD, test backed up

Feature driven

XP-ish

well documented code, Coding style

though kept simple with the use of merciless refactoring.

weekly development diaries & demos

WRITING DOWN NOTES, physical development diary & working out of ideas

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

# Design

Due to the process by which this project was developed, the design and architecture was inherently an evolutionary one. This means that at any one point in the development of the project, the overall design was subject to change. Even at this completion stage of the project, there is still an element of flux with the design going forward with any further developments. However, that said, the pace of large-scale changes to the overall design slows down during development, with the arrival at a (somewhat) simple, yet extensible design.

The designs used across the project at various levels of architecture were inspired by well-established design patterns and frequently used Android development practices. The latter was particularly important in its adoption, considering that this is the first time I have programmed for the platform.

During the background research and problem analysis, some ideas came with clear suggestions to the design of its respective component (take the manoeuvre components, for example). This research also uncovered some possible use cases for design patterns and other various solutions.

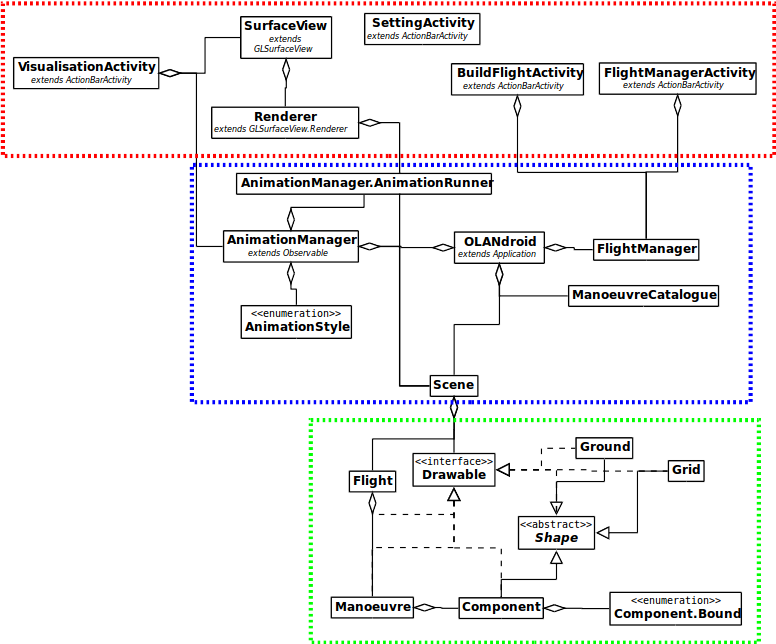
Another factor in the design of the application was the choice of technologies used (besides the platform).

The design is also based around the fundamental concepts of the implementation language. Object orientated program prizes – amongst other concepts – abstraction, encapsulation and polymorphism. These features were made use of to great extent throughout the design.

HIGH COHESION, LOW COUPLING

## Overall Architecture

Simply put, the overall architecture for this project is inspired by the model view controller design pattern, with the majority of classes fitting into these three categories as Figure shows.

Figure 1: architecture overview demonstrating the model view controller, red for view, blue for controller and green for model

### Model

The model part of the design mainly encapsulates the representation of the flights and their manoeuvres.

For the flight, the “Drawable” interface is a key part of this design. It details the functionality required of a flight. The methods which it specifies are designed to be highly reusable, and valuable to the lower level implementation of the flight as well. This is particularly noticeable when comparing the design of the “Flight” and “Manoeuvre” classes, as both contain lists of constituent parts (manoeuvres and components respectively).

The interface is used again to a lesser extent for the planes - “Ground” and “Grid” classes.

The “Shape” abstract class is where the flights meet the visualisation, and so it generally goes hand-in-hand with the “Drawable” interface, being extended by “Component”, and the two (surface) plane classes. The class encapsulates the interaction with rendering part of the program – OpenGL – so any extensions to it do not need to manage this themselves. In place of that direct interaction, it provides an interface which supports the drawing of planes (or empty polygons) through a list of vertices and a draw order, with a few extra optional features such as

“Shape” relies on the “Render” to have an compiled and initialised OpenGL program, which is accesses through a public reference. As draw calls are only made once the “Renderer” is initialised, which is done through the “SurfaceView” and “VisualisationActivity” classes, this value can be relied upon.

### View

### Controller

The centre of the project is the OLANdroid class.

Use of design patterns

Compromise of MVC

Class diagrams

Reusability of components, design of components to match the potential movement of aeroplanes

hierarchy of classes, shape, opengl stuff, shader descriptions

animation – DIRTY DRAWING & THREADING COMPONENT ADJUSTMENT

technology – java, gradle (build automation), android sdk, emulator, devices

Developing a working environment – IDE & use of xml-generating ui tool

relationship between the model of the application – manoeuvre representation & class diagram

AVOIDING FUNCTIONAL CODE IN ACTITIVITIES, LIMIT ACTIVITY CODE TO VIEW AND A TINY BIT OF CONTROLLER

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.

The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

Some example sub-sections may be as follows, but the specific sections are for you to define.

## Overall Architecture

Android applications are written using Java, a strictly object-oriented language, therefore the design can be summed up with a class diagram:

UML CLASS DIAGRAM HERE YO

(It should be noted that this diagram is incomplete for the sake of brevity, omitted instance variables/methods do not contribute to the description of the overall system well).

## Detailed Design

### Even More Detail

## User Interface Design

The evolutionary design of the project also affected the user interface, with various iterations catering for different features and handling them in different ways.

In Android, application user interfaces are split up into “activities”, which can be thought of as pages of an application. The layout of each activity consists of user interface elements like buttons and text input, these are “views”. Layouts can be constructed programmatically in the initialisation of an activity or with the interpretation of an XML layout.

There are also user interface elements which are not directly part of an activity. This includes things like menu bars and dialogue boxes.

The final user interface makes use of a number of these features.

My experience with using Android devices, having used them for over five years, was useful in designing a user interface an Android user would expect – utilising common UX patterns like the menu bar, settings menu and heavy use of standard iconography. However, it was important also to recognise that not all users will have this experience, and so to create a user interface which was accessible to all

From a stylistic perspective, the UI is simple, with no distinct style of its own – making use of the system “look and feel” to look native.

Various iterations, in terms of flow dependant on features being added

CONSIDERING THE PRINCIPALS OF USER INTERFACE DESIGN

## Other Relevant Sections

# Implementation

HOW TO BUILD THE VISUALS – VERTICES ETC.

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

# Testing

Testing visual things is a complex matter

can only really thoroughly test non-visual features

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on ’real users’? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

## Automated Testing

### Unit Tests

### User Interface Testing

Monkey

### Stress Testing

monkeyrunner

### Other Types of Testing

Various devices

## Integration Testing

Play store beta testing

## User Testing

Got family to use it to raise issues

crash reports & ANR ones too

# Critical Evaluation

Failure to incorporate all possible OLAN modifiers

recognition of the value of merciless refactoring & why its so often not kept to high regard

earlier testing

ISSUES: FLOATING POINT ERRORS WITH ANIMATION THREAD, DEGREES/RADIANS issues

Examiners expect to find in your dissertation a section addressing such questions as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

# Appendices

Third-Party Code and Libraries

Openaero

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation Apache Software Foundation (2014) “Apache POI - the Java API for Microsoft Documents” (Online) Available at: http://poi.apache.org Accessed: 14th March 2014.. The library is released using the Apache License Error: Reference source not found. This library was used without modification.

Android SDK –

Code Samples

Regex for OLAN interpretation perhaps?

Manoeuvre catalogue xml representation of a manoeuvre

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

# Annotated Bibliography

# 

This final section should list all relevant resources that you have consulted in researching your project. Each reference should also include a brief annotation.

1. Sylvia Duckworth. A picture of a kitten at Hellifield Peel. <http://www.geograph.org.uk/photo/640959>, 2007. Copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.  
     
   This is my annotation. I should add in a description here.
2. Mark Neal, Jan Feyereisl, Rosario Rascunà, and Xiaolei Wang. Don’t touch me, I’m fine: Robot autonomy using an artificial innate immune system. In *Proceedings of the 5th International Conference on Artificial Immune Systems*, pages 349–361. Springer, 2006.   
     
   This paper…
3. W.H. Press et al. *Numerical recipes in C*. Cambridge University Press Cambridge, 1992.  
     
   This is my annotation. I can add in comments that are in **bold** and *italics*and then further content.
4. Various. Fail blog. <http://www.failblog.org/>, August 2011. Accessed August 2011.  
     
   This is my annotation. I should add in a description here.
5. Apache Software Foundation (2014) “*Apache POI - the Java API for Microsoft Documents*” (Online) Available at: [http://poi.apache.org](http://poi.apache.org/) Accessed: 14th March 2014.   
     
   Annotation...
6. Apache Software Foundation (2004) “Apache License, Version 2.0” (Online) Available at: <http://www.apache.org/licenses/LICENSE-2.0> Accessed: 14th March 2014.   
     
   Annotation...