B.Eng. Dissertation

3D PLOTTING OF AIRCRAFT ATTITUDE

Submitted by

MUHAMMAD OMER IQBAL

[November 5, 2013 at 23:45 – classicthesis version 2.0]

ABSTRACT

Air accident investigations usually involve reading and interpreting large volumes of flight data. Visualising this data, especially in a geographic context makes its analysis easier. Therefore the Air Accident Investigation Board (AAIB) needed a tool that would render a 3 dimensional model of flight data from their datasets. The data includes measurements like latitude, longtitude, altitude, heading, roll, thrust etc over time. To meet this criteria I designed a Clojure based application that takes a dataset in csv format and renders aircraft attitude over time on a map. The application generates a KML (Keyhole Markup Language) file which can be opened on Google Earth and displays a 3d model of the data.

SUBJECT DESCRIPTORS:

- Specialized information retrieval
- Information visualization
- Geographic visualization

KEYWORDS: Clojure, KML, Visualisation, Aircraft Attitude, XML generation, 3d Modelling

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Put your publications from the thesis here. The packages `multibib` or `bibtopic` etc. can be used to handle multiple different bibliographies in your document.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— **?** [**?**]

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Colin Tan for recommending this project to me and for giving me full independence in the implementation. I would also like to thank Mr. Steven Teo of AAIB for his prompt and helpful feedback over the course of the project.

I also would like to extend my thanks to the clojure community in general for releasing such a magnitude of powerful open source libraries that made this project incredibly fun to work on.

Lastly, I would like to thank Mina, my girlfriend, for being patient.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

DRY	Don't Repeat Yourself
API	Application Programming Interface
UML	Unified Modeling Language
KML	Keyhole Markup Language
AAIB	Air Accident Investigation Board

Part I

THE PROBLEM

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* anglo-romanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

1 INTRODUCTION

1.1 BACKGROUND

Aircraft systems record large quantities of data during flight. A lot of this data is necessary in the system 's proper functioning. It takes an even more significant role when investigating air accidents. As this data establishes the causes behind those accidents, it aids measures to prevent such accidents from occurring. That is the primary reason for why recovery of Flight Data Recorders [1] is usually a very high priority for aircraft investigations.

The data itself features many components tracked over time, including but not limited to the following[2]:

- Latitude
- Longtitude
- Altitude
- Heading
- Pitch
- Roll
- Engine Thrust
- Throttle Lever Position

As is probably evident, analyzing this largely numeric data by itself would be a painstaking task. Primarily because finding patterns in numbers through observation is inherently difficult. Also because the data set is of a substantially large size [3].

Therefore the need for a tool to visualise this data becomes immediately obvious.

The Air Accident Investigation Board (AAIB), a branch of Singapore's Ministry of Transport performs analysis on such flight data, and requested an application that would render relevant flight data quickly and in a portable manner.

1 Popularly referred to as 'Blackboxes'
2 The following components were part of the sample dataset I was given for testing.
3 I will be cautious before calling it "Big Data".

1.2 SYSTEM REQUIREMENTS

AAIB needed the application to meet certain criterias and contain certain core features. These included:

1. GOOGLE EARTH Compatibility. Google Earth is a popular "virtual globe, map and geographic information program". It provides detailed 3d projections of Google's satellite images on a spherical globe, overlaid with huge quantities of textual and pictorial data varying from international boundaries to streets. There are many good reasons for this requirement including:
   - Cross Platform. It works and is supported on Windows, OSX and Linux [4]
   - Extensible through the Keyhole Markup Language (KML) [5]
   - Free[6]
   - AAIB's familiarity with it
   - Offline use. This is quite important as internet access is not guranteed on accident sites
2. CSV[7] based input. The system should be able to use large datasets encoded in CSV.
3. The system should display the data in its geographical location as vectors. The visualization should be able to reflect the actual data in an intuitive way. For instance, vectors at each data point should represent roll, heading, engine throttle
4. Users should be able to select which vectors they want to view.
5. The data points should be manually adjustable from within google earth. This is particularly relevant for data points near landing strips, where measurement errors may cause an offset in the rendered visualization

4 Atleast the desktop version is supported on all three major Operating Systems.
5 A lot more on this later
6 As in beer, not speech
7 Comma Separated Values

Part II

DESIGNING A SOLUTION

2 RENDERING STRATEGIES

As with any project, the first few stages of the project involved researching on existing technologies that either implemented the features specified in by AAIB in their requirements, or aided in developing those features. As Google Earth compatibility was a major requirement for the project, my research started with exploring methods to render custom data on Google Earth's platform.

2.1 KEYHOLE MARKUP LANGUAGE

The Keyhole Markup Language (*KML*) is a subset of XML, used to display geographic data in an Earth Browser, like Google Earth. It uses a tag based structure with nested elements that represent certain geometric and descriptive constructs. The KML reference is very reasonably documented with sample code demonstrating it's usage.

KML supports rendering the following features:

1. Placemarks
2. Descriptions
3. Ground Overlays
4. Paths
5. Polygons

KML uses latitudes and longtitudes as a coordinate system for positioning elements, which works very well with geographic data.
However KML is primarily designed as a means to display data, as opposed to performing interactions with the data. The only interactions it allows are those that Google Earth features, i.e. zooming, panning and moving to a certain point on the globe. Clicking on placemarks with descriptions opens a modal that renders the description, which can be written in a subset of HTML.

2.2 GOOGLE EARTH'S BROWSER PLUGIN

With web browsers becoming increasingly powerful and technologies like WebGL that allow efficient 3d rendering on the browser, Google launched the Google Earth Browser Plugin. The browser plugin by itself allows pretty much the similar core functionality as Google

Earth's Desktop Application. The primary difference is that the virtual globe can be embedded within an HTML web page.

By itself the browser plugin may not seem to accomplish much. However Google also released the Google Earth API, a javascript library that provides access to various features in Google Earth. The browser plugin when combined with Google Earth's API becomes a very powerful tool to produce complex applications that render and allow interaction with geographic data.

For instance one of the sample toy applications provided as documentation for the API is a car simulator, where you drive a car over the virtual globe's surface.

Such user interactions are possible by combining interactions provided by the DOM, and using them to render models using the Google Earth API.

However, the browser plugin is not a silver bullet. One of it's most lacking features is offline usage. The browser plugin does not work offline, unlike the desktop application which caches data for offline use.

2.3 CHOOSING A RENDERING STRATEGY

After researching on rendering methods available I had to choose which method was a better fit for this project. From my research the following rendering strategies seemed initially viable:

1. Generate KML and use it's Path and Polygon constructs to render the various vectors that AAIB requested
2. Use Google Earth's javascript API to render the same data.

After revisiting AAIB's application requirements, I decided that between producing KML and using Google Earth s Browser Plugin, generating KML was a significantly better option. The rationale guiding the decision was based on the following factors:

- Offline usage. As mentioned before, the browser plugin does not feature offline usage. This is in contrast to rendering KML documents on Google Earth's Desktop client which works well offline. Considering offline usage was an important part of the application requirements, this was a major reason.
- Portability. KML is a well defined format that can be rendered on many other earth browsers. Furthermore, the browser plugin is only supported for Windows and Mac OS. Linux [1] was not

1 Which I am a huge fan of

yet supported, and I did not want to add that constraint to the application without good reason. In contrast, Google Earth is supported on Windows, Mac OS and Linux.

3 CHOOSING THE STACK

This decision took a while. And rightfully so, as it would affect all decisions from that point. Choosing the right language and framework could have more than substantial effects on the system's architecture.

After quite a bit of research and debate, I decided to implement the system using CLOJURE, a relatively new lisp class language than runs on the JVM.

3.1 CLOJURE

The choice of implementing this project in clojure may seem peculiar. However there are a number of features that clojure and it's ecosystem that fit very well with this particular project. Before jumping to those features, a quick overview of clojure is in order.

Clojure is a lisp dialect, created by Rich Hickey. It is a general-purpose, functional programming language that runs on the Java Virtual Machine (JVM) [1]. Clojure focuses on programming with immutable data structures and provides a number of persistent hash-trie based data structures that allow efficient functional programming.

Clojure was designed to be a real world language, as opposed to a purely academic one. Therefore it integrates seamlessly with it's host environment. For the JVM that means java interoperability is very straightforward. This also means that all existing libraries and frameworks for Java can be used with Clojure.

The decision to choose Clojure as the main language for implementing this project was based on the following features of clojure

3.1.1 *Functional Programming*

Clojure is foremost a functional programming language. That means the core logic of any application is implemented as a series of pure functions that return immutable values. Pure here means that functions do not return different values for the same set of inputs. The paradigm also involves treating functions as first class members that

1 Clojure runs in other environments like Microsoft's Common Language Runtime (CLR) and Javascript engines as well

can be passed around as values.

As mentioned before, my strategy to rendering the data was to convert it into an appropriate form in KML. This operation in essense is a pure function that consumes a set of inputs and performs several transformations on it to return a KML document.

In other words, there is not state involved. For a given set of data, the output never changes. The transformation operation does not need any side-effects like doing IO or mutating variables.

Furthermore clojure idioms encourage the use of higher order functions that make code succint. I personally enjoy writing programs in a functional style, as opposed to an object oriented style, which despite it's merits can tend to overuse mutation.

3.1.2 *Libraries*

Library support is probably one of the most important factors in deciding the stack for any project. Since part of the system involved generating xml, a fast, xml processing library was a must. Similarly, on the parsing side, even though parsing csv files is trivial, the parsing needed to be done in a performant manner, especially when the data sets might be very large. Also, the input data format could change in other scenarios, and having a decent set of parsing tools for more advanced grammars (JSON, XML) makes a lot of sense.

On both counts clojure scores very well. In terms of xml processing, clojure.data.xml provides a decently documented, clean api which can lazily parse and generate large xml documents. Under the hood its uses Java's STAX library.

But in addition to xml generation and parsing, clojure also provides a very neat library called clojure.data.zip for performing queries and transforms on generic tree like structures. clojure.data.zip, as the name suggests uses Zippers to functionally traverse and modify trees.

3.1.3 *Domain Specific Languages*

Building Domain Specific Languages (DSLs) is where lisp like languages are reputed to shine. Lisp's homoiconicity, and the ability to treat code as data helps quite a bit here. Clojure is no exception to this rule, and features many rich DSLs.

The need for a DSL particularly arose when dealing with XML generation. As the application required generating KML, which is a subset of XML, finding a tool that eases it's generation was paramount. XML by itself is not "designed for humans". The syntax is verbose, repetitive and prone to errors.

Representing XML data as raw strings can be incredibly messy and error prone, as there is no way to gurantee the validity of document structure. Forgetting or misplacing closing tags can lead to bugs, that would be very difficult to find if the generated XML document is complex. Considering the application was meant to generate reasonably complex KML documents, using raw strings was to be avoided at all costs.

Enter Hiccup, a very popular DSL that represents XML using clojure's vectors and hash-maps. Although originally just designed for HTML, the DSL became so popular that most of clojure's XML libraries support it.

The following snippet compares raw KML with the equivalent Hiccup representation:

Listing 1: A simple placemark in raw KML

```
<Placemark>
  <name> Simple Placemark </name>
  <Polygon>
    <extrude>0
    </extrude>
    <outerBoundaryIs>
    </outerBoundaryIs>
  </Polygon>
</Placemark>
```

Listing 2: The equivalent placemark with hiccup. Notice the brevity and lack of repetition

```
[:Placemark
  [:name ''Simple Placemark'']
  [:Polygon
    [:extrude ''0'']
    [:outerBoundaryIs]]]]
```

As is evident, the sample using hiccup is far more succint. This becomes even more evident when the size of the document increases.

However hiccup is not only useful because of it's brevity. It's killer feature is the fact that the representation is simply a clojure vector, which can be manipulated as a proper tree like data structure. This reduces XML generation to manipulating clojure vectors, which is signifigantly easier.

Another major feature that hiccup brings is composibility. Because I just have to deal with clojure vectors, I can make small functions that produce base components of the document and compose them together to form more advanced forms. This is very powerful and much more elegant than the naive approach of concatenating XML strings together.

4 MODELLING THE PROBLEM

As mentioned before, the core function of the application was to convert periodic flight data into a renderable 3d model. This section will detail the approach taken to model the various aspects of the flight data.

4.1 INPUTS

The dataset AAIB provided, contained the following measurements, recorded over time t.

1. Magnetic Heading $\theta_m(t)$. Measures the direction of the aircraft with respect to the north pole. Measured in degrees
2. Pressure Altitude $a(t)$. Indicated altitude, measured in feet
3. GPS Latitude $\phi(t)$. Measured in degrees
4. GPS Longtitude $\lambda(t)$. Also measured in degrees
5. Roll Angle $\theta_r(t)$. Also measured in degrees
6. Engine Thrust $e_i(t), 0 < i \leqslant 4$. Measured as a percentage

The dataset had other fields as well, but they were not used for the model implemented

4.2 CONSTRAINTS

A constraint that choosing KML placed on the model was that any coordinate could only be represented withe following dimensions:

1. Latitude (ϕ)
2. Longtitude (λ)
3. Altitude a

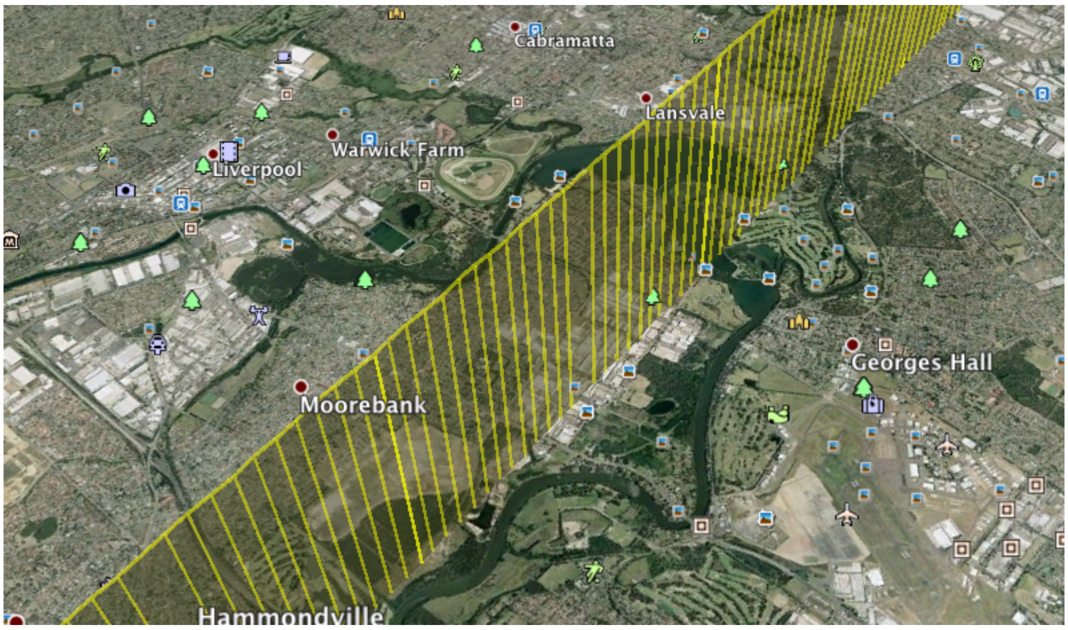
This implied that plotting polygons with KML would require converting to and fro between cartesian and spherical coordinate systems.

4.3 FLIGHT PATH

Plotting the flight path was the most straightforward operation. The flight path was to be modelled using KML's Line type, which just required sets of Latitude, Longtitude and Altitude readings

$$\mathrm{FlightPath} = \{\{a(t), \phi(t), \lambda(t)\} | t_{start} < t \leqslant t_{end}\}$$

Figure 1: Sample Flight Path, displaying Latitude, Longtitude and Altitude



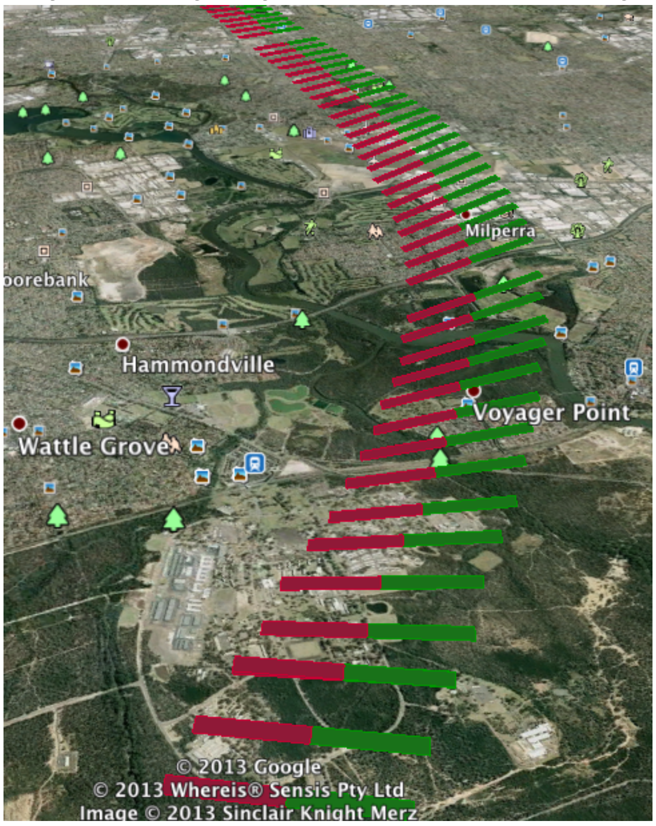
4.4 ROLL AND HEADING

Representing the roll and heading with KML was more complex. The central idea was to model the aircraft's "wings", which would give a visual representative of roll by assessing the roll angle of the wing. Modeling the wings would also give an indication of the heading, if the wings are assumed to be straight lines. The heading would always be perpendicular to the wings.

The following figure should make this idea clearer:

The wings could be modelled using KML's Polygon type. However, as mentioned before, we can only use latitude,longitude and altitude as coordinates for KML's types. Therefore to make the modelling easier, the following concepts are necessary:

Figure 2: Using wings to visualize roll and heading



4.4.1 *Distance*

Finding the distance between two latlong points is an essential operation. For this application I used the Haversine Formula, which works was follows:

$$a = \sin^2(\Delta\phi/2) + \cos(\phi_1)\cos(\phi_2)\sin^2(\Delta\lambda/2) \tag{1}$$

$$d_H = 2R_{earth} \arctan(\frac{\sqrt{a}}{\sqrt{1-a}}) \tag{2}$$

Here d_H refers to the Haversine distance between two latlong points $\{\phi_1, \lambda_1\}$ and $\{\phi_2, \lambda_2\}$ and R_{earth} refers to the Earth's radius.

The Haversine distance gives the shortest distance between both points over the Earth's surface. It can be thought as being equivalent to the "as the crow flies" distance between two points.

4.4.2 *Bearing*

Bearing θ between two latlong points $\{\phi_1, \lambda_1\}$ and $\{\phi_2, \lambda_2\}$ can be calculated with the following method:

$\theta = \arctan(\frac{\sin(\Delta\lambda)\cos(\phi_2)}{\cos(\phi_1)\sin(\phi_2)-\sin(\phi_1)\cos(\phi_2)\cos(\Delta\lambda)})$
Bear in mind, this bearing is the initial bearing, also referred to as the forward azimuth, which if followed in a straight line would take one from the first point to the second. By straight line I mean a stright line over the Earth's surface, also referred to as a great circle.

4.4.3 *Destination Point, given distance and bearing*

Probably the most useful concept in terms of generating KML. Representing any polygon in KML would require calculating a point's latlong coordinates when given a distange range and bearing.

To calculate the coordinates $\{\phi_2, \lambda_2\}$ located a distance d from point $\{\phi_1, \lambda_1\}$ at a bearing of θ, the following can be used:

$$\phi_2 = \sin(\phi_1)\cos(d/R_{earth}) + \cos(\phi_1)\sin(d/R_{earth})\cos(\theta) \quad (3)$$

$$\lambda_2 = \lambda_1 + \arctan(\frac{\sin(\theta)\sin(d/R_{earth})\cos(\phi_1)}{\cos(d/R) - \sin(\phi_1)sin(\phi_2)}) \quad (4)$$

One thing to note though is that these equations assume altitude from the earth's surface to be negligible when compared to the Earth's radius. Since this particular application mostly deals with commercial aircraft which do not cruse above 18 km, which is significantly lesser than the Earth's Radius (6,371 km)

4.4.4 *Wings as KML Polygons*

With equations 3 and 4 above, modelling wings via KML becomes easier. From a given point, it should be easy to obtain the coordinates of a rectangle of a particular width and height.

To illustrate this further, lets define the following function

$$destinationPoint(point, \theta, d, a) \rightarrow \{\phi, \lambda, a\} \quad (5)$$

It takes a latlong coordinate point $point$, an initial bearing θ, a distance d and altitude a and returns a destination coordinate using equations 3 and 4.

To define a wing, lets assume we have a starting coordinate $\{\phi, \lambda, a\}$, a width of $2w$, a height of $2h$, a heading $\theta_{heading}$ and a roll angle θ_{roll}

$$\beta_1 = \arctan(w/h) \quad (6)$$

$$\beta_2 = 2(\pi/2 - \beta_1) \tag{7}$$

$$r = \sqrt{w^2 + h^2} \tag{8}$$

$$\delta a = \sqrt{2} r \sin(\theta_{roll}) \tag{9}$$

δa measures the change in altitude required to represent the roll

Finally, lets define an array $Wing$ which consists 4 vertices to plot the rectangle

$$Wing[0] = destinationPoint(\{\phi, \lambda\}, \beta_1 + \theta_{heading}, r, a + \delta a) \tag{10}$$

$$Wing[1] = destinationPoint(\{\phi, \lambda\}, \beta_1 + \beta_2 + \theta_{heading}, r, a + \delta a) \tag{11}$$

$$Wing[2] = destinationPoint(\{\phi, \lambda\}, \beta_1 + \pi + \theta_{heading}, r, a - \delta a) \tag{12}$$

$$Wing[3] = destinationPoint(\{\phi, \lambda\}, \beta_1 + \beta_2 + \pi + \theta_{heading}, r, a - \delta a) \tag{13}$$

After this, $Wing$ can easily be rendered as coordinates of a KML Polygon

4.5 THRUST

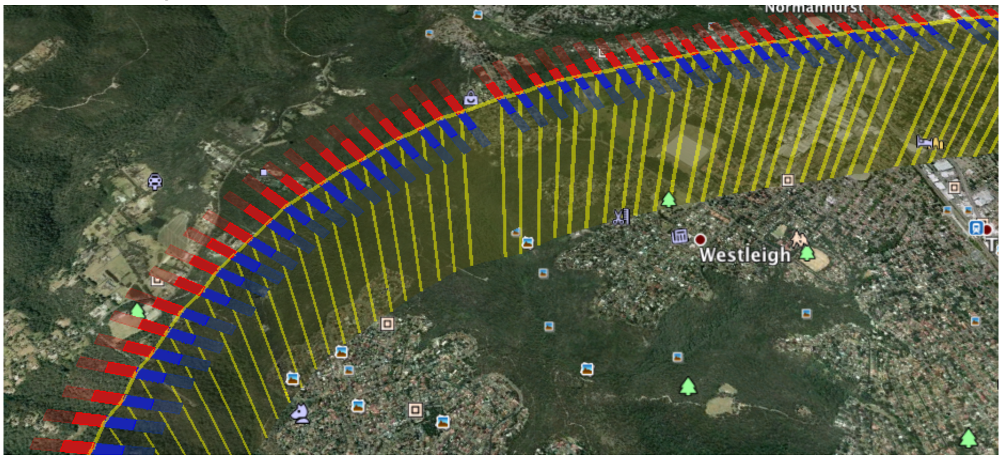
Visualising thrust is relatively much easier. Since we already have a method for rendering wings of a certain width and height, thrust can be visualized by simply adjusting the width and height of the wing and making it proportional to thrust of engines on that wing.

The only change required would be to represent left and right wings as two separate rectangles. The reason for this is because many commercial aircraft have engines on both wings. The dataset provided by AAIB had 4 engines, two on each wing, which may have differing thrust values.

The figure below should make this idea clearer. The rectangles with the darker color represent thrust on both wings. They are positioned over rectangles of a lighter color, whose size would correspond to 100% thrust.

This allows the user to get a better idea of what the thust level at that point in the flight was.

Figure 3: Using several wings to indicate thrust levels on left and right wings



Part III

APPENDIX

A

APPENDIX TEST

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

A.1 APPENDIX SECTION TEST

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

More dummy text

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse

LABITUR BONORUM PRI NO	QUE VISTA	HUMAN
fastidii ea ius	germano	demonstratea
suscipit instructior	titulo	personas
quaestio philosophia	facto	demonstrated

Table 1: Autem usu id.

Listing 3: A floating example

```
for i:=maxint to 0 do
begin
{ do nothing }
end;
```

viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

A.2 ANOTHER APPENDIX SECTION TEST

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

COLOPHON

This document was typeset using the typographical look-and-feel classicthesis developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". classicthesis is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of classicthesis usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

DECLARATION

Put your declaration here.

Singapore, November 2013

Muhammad Omer Iqbal,
November 5, 2013