VisualFlight: The Air Traffic Control Data Analysis System

by

Joyce C. Lin

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 24, 2002

© 2002 M.I.T. All rights reserved.
VisualFlight:
The Air Traffic Control Data Analysis System

by

Joyce C. Lin

Submitted to the Department of Electrical Engineering and Computer Science

May 24, 2002

In partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This paper introduces VisualFlight, a new system for recording and analyzing historical air traffic control data. VisualFlight permits queries about airspace, aircraft, carrier, and airspace temporal utilization to be posed, and results can be displayed as maps, tables, and graphs. VisualFlight is based upon the MIT Laboratory of Computer Science archival system for the real-time DOT/FAA Aircraft Situational Display to Industry feed. This archival system has been modified and enhanced for functionality and increased reliability as part of the VisualFlight system. In this paper we use VisualFlight to investigate the usage of our National Airspace System by examining aircraft density, airport operations, aircraft usage, and aircraft performance. This data analysis tool may be used for a wide variety of purposes, and its functionality is extensible beyond the scope of our initial applications.

Thesis Supervisor: David K. Gifford
Title: Professor of Electrical Engineering and Computer Science
Acknowledgements

Thanks to Professor David Gifford for being a terrific thesis supervisor in every way! He was always available and approachable, provided the right amount of guidance, and never assigned tasks that were unmanageable. His vision and enthusiasm for this project were contagious and really helped me to enjoy working on this thesis.

Thanks to Jeanne Darling for being extremely helpful in just about everything. She claims she’s “just a secretary” but she’s definitely the most knowledgeable Linux administrator / programmer / database repairperson type secretary that I have ever met. She always made herself available and would provide help at the drop of a hat.

Thanks to James Goode and Shang Chuankun at the MIT Oracle support office, who were extremely helpful in solving database problems.

Thanks to the Air Force Reserve Officer Training Corps (AFROTC) for allowing me to take an educational delay to receive a Master’s degree. I am excited to be working for the Air Force Office of Special Investigations (AFOSI) following this school year.

Thanks to my family and friends for their support, for always trying to make sure that I was alright and enjoying my last year at MIT in addition to completing all the necessary work.

And finally, thank you to Kathy, Monica, Sandy, Carolyn, Kristen, Sarah, Daniel, Ben, Lawrence, Russell, Chris, Brittany, and Dan for making up a very special group called the MIT Cross Products, and helping to make this past year one to cherish for the rest of my life.
Contents

Chapter 1: VisualFlight Introduction and Overview ................................................. 6
  1.1 Introduction ............................................................................................... 6
  1.2 Motivation ............................................................................................... 7
  1.3 Existing ASDI Technology ....................................................................... 8
  1.4 System Requirements and Expectations .................................................. 9

Chapter 2: Changes to the Database Archival System ............................................. 11
  2.1 Position Reports and Additional Database Fields .................................... 11
    2.1.1 Saving Position Reports in the Flight Database ................................ 11
    2.1.2 An Alternative Method of Saving Position Reports ......................... 15
    2.1.3 Additional Flight Table Columns .................................................... 16
  2.2 Modified Flight Resolution Algorithm ..................................................... 17
  2.3 Archival System Robustness and Reliability ......................................... 20
    2.3.1 Database Error Management .......................................................... 20
    2.3.2 Ensuring Collection of All Available Data ....................................... 21

Chapter 3: VisualFlight System Design ................................................................. 23
  3.1 System Overview ..................................................................................... 23
    3.1.1 User Specification Stage .................................................................. 24
    3.1.2 Flight Analysis Stage ....................................................................... 24
    3.1.3 Conclusion Stage .............................................................................. 25
  3.2 Basic System Functionality and Capabilities ......................................... 25
    3.2.1 Distance Computer .......................................................................... 25
    3.2.2 Airport Database ............................................................................. 27
    3.2.3 Map Creation .................................................................................... 28
    3.2.4 Graph Creation ................................................................................ 30
  3.3 Implemented Analysis Tools .................................................................... 30
    3.3.1 Map Related Analysis Tools ............................................................. 31
    3.3.2 Aircraft/Airport Operations Analysis Tools ..................................... 34
    3.3.3 Aircraft Performance Analysis Tools .............................................. 37
  3.4 Graphical User Interface .......................................................................... 40
    3.4.1 Filters: Data Specification ............................................................... 41
    3.4.2 Map Creation ................................................................................... 43
    3.4.3 Graph Creation ................................................................................ 47

Chapter 4: Graphical Results ............................................................................... 48
  4.1 Aircraft Route Maps ................................................................................. 48
  4.2 Aircraft Density Maps .............................................................................. 52
  4.3 Airport Usage ........................................................................................... 56
  4.4 Aircraft Usage .......................................................................................... 60
  4.5 Aircraft Performance ............................................................................... 63

Chapter 5: Conclusion ......................................................................................... 67
List of Figures

Figure 3.1: User specification stage .................................................................24
Figure 3.2: Flight Analysis Stage .................................................................24
Figure 3.3: Conclusion Stage .......................................................................25
Figure 3.4: VisualFlight Graphical Interface ...............................................41
Figure 3.5: Filter Parameter Selection Dialog .............................................42
Figure 3.6: Sample Data Filters .................................................................43
Figure 3.7: Map Selection Dialog ...............................................................44
Figure 3.8: Map Boundary Selection ...........................................................45
Figure 3.9: Newly Created Map Using User's Selected Boundaries ..........45
Figure 3.10: Selection and Specification of Map-related Analysis Tools ....46
Figure 3.11: Graph-related Analysis Tool Specification Example ..............47
Figure 4.1: Routes traveled by C550s on April 19, 2002 .............................49
Figure 4.2: Traffic at BOS on April 15, 2002 ...............................................50
Figure 4.3: Traffic arriving at DAY on April 15, 2002 ................................51
Figure 4.4: Density of Arriving/Departing Traffic over the U.S. ...............53
Figure 4.5: Density of Cruise Traffic over the U.S. .....................................54
Figure 4.6: Traffic Density over Washington D.C./Philadelphia Area .......55
Figure 4.7: Screen Shot of Airport Operations Data ..................................56
Figure 4.8: Air Traffic Levels of the Ten Busiest U.S. Airports ..................57
Figure 4.9: GA vs. Carrier Traffic Levels at Busiest 50 Airports ...............58
Figure 4.10: GA vs. Carrier Traffic Levels at Busiest 51-100 Airports ......58
Figure 4.11: GA vs. Carrier Traffic Levels at Busiest 101-150 Airports ......59
Figure 4.12: GA vs. Carrier Traffic Levels at Busiest 201-250 Airports ......59
Figure 4.13: Hours Flown vs. Day of Month by Ten Most Active Aircraft Types .61
Figure 4.14: Comparison of Hours Flown Between GA and Carrier Flights ....62
Figure 4.16: Average Speed vs. Distance Flown .......................................65
Figure 4.17: Cumulative Distribution of Distance Flown by Aircraft Type ....66

List of Tables

Table 2.1: Flight Table Columns .................................................................12
Table 2.2: Added Position Report Columns in the Flight Table ..................14
Table 2.3: Proposed Positions Table Columns ..........................................15
Table 2.4: Additional Flight Table Columns ..............................................17
Table 3.1: Distance Computer Comparisons ............................................27
Chapter 1: VisualFlight Introduction and Overview

1.1 Introduction

We have designed and implemented VisualFlight, the first general-purpose tool to render the MIT Laboratory of Computer Science (MIT LCS) archive of air traffic control data into a form that will make it useful to air traffic and aircraft researchers. MIT LCS is directly connected to the Department of Transportation and receives and archives the “Aircraft Situational Display to Industry” (ASDI) feed to an Oracle database. The ASDI feed is a real-time air traffic data stream provided as a joint project by the Federal Aviation Administration (FAA) and the Air Transport Association (ATA). The data stream is composed of ASCII text messages, where each message represents a “flight event”, such as a flight plan, arrival announcement, or position update. The details of the ASDI feed may be found in a description provided by the Volpe Center, an ASDI provider in Cambridge, Massachusetts [1].

Our graphical analysis tool (hereafter referred to as VisualFlight) is built upon the ASDI archival system (termed “ASDISaver”) developed by Micah Gutman [2]. The ASDISaver parses the ASDI data stream and tracks in-air flights. Once a flight arrives at its destination there is no more information to be saved for the flight, and the ASDISaver writes the flight’s information to a database. We have modified and enhanced this system in order to archive the necessary data for graphical analysis. The VisualFlight system presents users with a graphical interface that allows them to analyze a specified section of our data archive by producing maps, graphs, and/or tables. We have compiled a section of graphical results to demonstrate the capabilities of the VisualFlight with respect to analyzing the National Airspace System.
1.2 Motivation

Although many applications have been built to display real-time aircraft data, applications that archive the ASDI data stream and allow users to analyze historical data are rare. We do not know of an application that allows analysis of historical data beyond “replaying” data to observe the progress of flights and flight configurations at a past date.

It is important to be able to determine how our National Airspace System is being utilized. Analyzing historical flight data allows one to get the overall picture of many aspects of the National Airspace System, such as where planes are being flown, traffic levels at airports, and whether aircraft actually perform to published standards. An application that allows analysis of archived data may be used for a wide array of purposes, including the following:

- **Development of plans to relieve airspace congestion**
  
  The FAA spends an untold amount of money each year on developing better methods to relieve airspace and airport congestion as well as to improve safety [3,4]. Thus, it becomes necessary to determine whether these methods are actually being put into practice, and whether they make a difference.

- **Verification of airplane performance**
  
  Each type of aircraft comes with specifications describing the expected performance of the plane, such as the cruise speed of the airplane at given altitude, or the climb rate at specific altitudes. Analysis of archived data may produce the average cruise speed or rate-of-climb for each type of aircraft, which may then be compared to the values published by the manufacturers.
• **Determining the need for General Aviation**

   After September 11, 2001 the importance of General Aviation (GA) was questioned because of the concern for national safety. This leads to more questions, such as, “How many GA flights are flown per day?” or “What is the ratio of GA flights to commercial air carrier flights at each airport in this country?” To answer such questions one must be able to archive and analyze air traffic data.

1.3 **Existing ASDI Technology**

   Most, if not all of the interest surrounding the data provided by the ASDI feed involves tracking current in-air flights in order to determine whether flights will arrive/depart on time, or how flights should be (re)routed. The ASDI feed is commonly used to graphically display the status and position of aircraft that are currently in the air, as well as aircraft that will depart in the near future and aircraft that have recently arrived at their destinations. One well-known and comprehensive application utilizing the ASDI feed is Dimension International’s FlightExplorer [5]. FlightExplorer displays the real-time ASDI data graphically, allowing users to see the positions of aircraft on a zoomable and clickable map. An extensive set of filters may be used so that users may view only the set of airplanes that they are concerned about. Other examples of graphical interfaces that use the ASDI feed are Trip.com’s FlightTracker [6], and Sonalysts, Inc.’s FlightTrax [7].
1.4 System Requirements and Expectations

The following requirements are expected of a flight data analysis system:

1 - Data reliability

One must be able to determine whether the data archival system is collecting all the available data from the ASDI feed or whether the system is only collecting a subset of it. Furthermore we must be able to ensure that the data being saved is accurate. Data analysis is only as good as the data being analyzed.

2 - Ability to display data in a useful and intelligible manner

Data that is described by text only is usually difficult and time-consuming to try to understand. The system must have the ability to display flight data (such as flight routes) on maps. These maps must be drawn and colored in a way that can be understood by the user. The system must also be able to create intelligible graphs and tables to display flight statistics. In addition, maps and graphs should be created in a reasonable amount of time.

3 - Comprehensible user interface

Users should be presented with an intuitive graphical interface allowing them to specify what sections of the data archive they would like to analyze, as well as what types of tests they would like to perform on the data. A text-only interface is clumsy, and users should not need to have knowledge of SQL syntax in order to access the Oracle database.
4 - The system should be extensible

The system should be designed in such a way that desired tests or data analysis tasks may easily be added on at a later date. A programmer should not have to drastically reprogram the majority of the system in order to implement additional data analysis functionality.
Chapter 2: Changes to the Database Archival System

We are using the ASDI archival system (or “ASDISaver”) designed by Micah Gutman in order to save ASDI data to an Oracle database. The details of the design and implementation of this archival system may be found in Gutman’s thesis paper [2]. This chapter describes the ways in which the ASDISaver has been modified in order to make the system more robust, and to save all available and necessary data for flight data analysis.

2.1 Position Reports and Additional Database Fields

2.1.1 Saving Position Reports in the Flight Database

Each individual flight’s data is saved as a unique row in the flight data table (hereafter referred to as the “Flight Table”) located in the Oracle database. Reproduced from Gutman’s paper in Table 2.1 are the columns of each flight row as saved in the Flight Table.

Position report messages make up the vast majority of messages encountered within the ASDI data stream. Position reports are usually received for each flight every one to five minutes. Each of these position reports consists of a timestamp and the flight’s current altitude, speed and position (in latitude/longitude coordinate format). For effective flight analysis, position report information for each flight must be saved to the database, something that was not implemented in the original ASDI archival system. In designing a way to save flight position reports in the Oracle database, we chose the method that would be most optimal for both data retrieval time and database storage space.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Type</th>
<th>Allow Null</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aircraftid</td>
<td>flight/tail number</td>
<td>varchar2(11)</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>deppoint</td>
<td>departure point</td>
<td>varchar2(12)</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>dest</td>
<td>Destination</td>
<td>varchar2(12)</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>origidest</td>
<td>Original destination, if it changed in the air</td>
<td>varchar2(12)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>originaldeptime</td>
<td>The planned original departure time</td>
<td>date</td>
<td>Yes</td>
<td>7</td>
</tr>
<tr>
<td>actualdeptime</td>
<td>The time the plane actually departed</td>
<td>date</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>deptimevalid</td>
<td>Departure message received for flight</td>
<td>number(1)</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>etafromdep</td>
<td>ETA at time of departure</td>
<td>date</td>
<td>Yes</td>
<td>7</td>
</tr>
<tr>
<td>arrtime</td>
<td>Arrival time</td>
<td>date</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>arrtimevalid</td>
<td>Arrival message received for flight</td>
<td>number(1)</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>numberaircraft</td>
<td>Number of aircraft</td>
<td>number(2)</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>heavyindicator</td>
<td>Heavy indicator</td>
<td>varchar2(1)</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>aircrafttype</td>
<td>Aircraft type</td>
<td>varchar2(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>equipment-qualifier</td>
<td>Equipment Qualifier</td>
<td>varchar2(1)</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>assignspeed</td>
<td>Assigned Speed</td>
<td>varchar2(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>assignedaltitude</td>
<td>Assigned Altitude</td>
<td>varchar2(7)</td>
<td>Yes</td>
<td>7</td>
</tr>
<tr>
<td>highspeed</td>
<td>Highest observed speed</td>
<td>number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>highaltitude</td>
<td>Highest altitude observed</td>
<td>number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>avgspeed</td>
<td>Average observed speed</td>
<td>number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>avgaltitude</td>
<td>Average observed altitude</td>
<td>number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>proposedfirst</td>
<td>Whether proposed flight plan received for flight</td>
<td>number(1)</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>route</td>
<td>Route</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>Variable</td>
</tr>
<tr>
<td>timeentered</td>
<td>Time record written to database</td>
<td>date</td>
<td>No</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 2.1: Flight Table Columns
Each position report is saved as a string, with the following components:

- **Timestamp** – format is “ddhhmmss”, with a length of 8 bytes
- **Altitude** – format is “nnn” where each ‘n’ is an integer. “nnn” denotes hundreds of feet, such that “035” stands for 3500 feet. Altitude strings may come with a letter at the end denoting whether the aircraft is transitioning to a different altitude. The length of an altitude string will be at most 4 bytes.
- **Speed** – format is “nnn” where each ‘n’ is an integer. “nnn” denotes the speed of the aircraft in knots. The length of each speed report is 3 bytes.
- **Coordinate** – format is “xxxxXyyyyyY” where ‘xxxx’ is the latitude value in degrees/minutes, X is the latitude cardinal direction (N or S), ‘yyyyy’ is the longitude value in degrees/minutes, and Y is the longitude cardinal direction (E or W). The length of each coordinate report is 11 bytes.

A flight’s position reports are saved as a string of reports, where each report consists of “;timestamp/altitude/speed/coordinate”, such that the delimiter ‘;’ separates each position report and the delimiter ‘/’ separates each field in each position report. In this way, each position report will require 30 bytes of space.

An Oracle database saves strings as a “varchar2” datatype. Because the varchar2 datatype has a maximum length of 4000 bytes, we created five new columns in the Flight Table, each having the capacity to save 120 position report strings (see Table 3.2). The varchar2 datatype is useful because although we specify that each varchar2 is able to hold up to 4000 bytes, space is not actually allocated in the database unless it is needed. Thus we allow a flight to use up to 120*30*5 = 18000 bytes for position report information,
but a flight with 100 received position reports only uses $100 \times 30 = 3000$ bytes in the
database to save its reports.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Type</th>
<th>Allow Null</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>positions1</td>
<td>Position reports #1-120</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>variable</td>
</tr>
<tr>
<td>positions2</td>
<td>Position reports #121-240</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>variable</td>
</tr>
<tr>
<td>positions3</td>
<td>Position reports #241-360</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>variable</td>
</tr>
<tr>
<td>positions4</td>
<td>Position reports #361-480</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>variable</td>
</tr>
<tr>
<td>positions5</td>
<td>Position reports #481-600</td>
<td>varchar2(4000)</td>
<td>Yes</td>
<td>variable</td>
</tr>
</tbody>
</table>

**Table 2.2: Added Position Report Columns in the Flight Table**

An alternative datatype to varchar2 is the LOB (large object), which does not have the
4000-byte space restriction. However, we were unable to use LOBs because they may
not be used when Oracle databases are partitioned, and the ASDI database archival
system uses a partitioned database.

At an average rate of one position report each minute, we are able to save roughly ten
hours worth of position reports. While in the implementation stage, we observed that
some flights receive position reports at an astounding rate of one every five seconds.
Since it is not very useful to receive position reports at such a high rate, we limited the
system to gathering position reports at a rate of at most one every thirty seconds for each
flight, out of consideration for hard drive space. After monitoring this system for a
month, we found that no flights received more than 550 position reports.

On a daily basis we collect data on roughly 50,000 flights, and by saving position reports
in this fashion, we observe that the Oracle database archive grows at a rate of about
100MB each day, translating to about 36.5 GB each year.
2.1.2 An Alternative Method of Saving Position Reports

In designing a way to save position reports to the Oracle database we investigated one alternative method. In theory, it seems like a good idea to create a position reports table (hereafter referred to as the “Positions Table”) in the database. This table would have the following columns: aircraft ID, departure time, position report number, timestamp, altitude, speed, and coordinate (see Table 3.3), and each position report would be saved as a unique row in the table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Type</th>
<th>Allow Null</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aircraftid</td>
<td>Flight/Tail number</td>
<td>varchar2(11)</td>
<td>No</td>
<td>11</td>
</tr>
<tr>
<td>actualdeptime</td>
<td>Time the plane departed</td>
<td>varchar2(12)</td>
<td>No</td>
<td>12</td>
</tr>
<tr>
<td>reportnum</td>
<td>The position report # received chronologically</td>
<td>number(3)</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>timestamp</td>
<td>Time this report was received</td>
<td>date</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>altitude</td>
<td>Observed altitude</td>
<td>number(3)</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>speed</td>
<td>Observed speed</td>
<td>number(3)</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>coordinate</td>
<td>Observed position</td>
<td>varchar2(11)</td>
<td>No</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2.3: Proposed Positions Table Columns

We investigated this method because of the shortcomings of the method outlined in the previous section:

1) Limited number of saved position reports (600). As discussed above, the varchar2 datatype has a space limit of 4000 bytes. By saving each position report as a row in the Positions Table, we need not worry about receiving more than 600 position reports for each individual flight.
2) In the first method, upon retrieving a position reports string, the string must be parsed, and then each field must be placed into a specified datatype. With hundreds of reports per flight, parsing and formatting position report strings may become time-expensive, and it may save time if position report information is already parsed and placed in appropriate datatypes in the Positions Table.

The first two columns in the Positions Table would ensure that all rows in this table that share the same aircraft ID and departure time belong to the same flight. Thus, when position report data is required of a flight, the system should query the Positions Table for all rows that match the flight’s aircraft tail number and actual departure time. This method would require about 50 bytes per position report, which is about 1.66 times the amount of space taken by the method described in the previous section.

The Positions Table was implemented in the Oracle database and flight data retrieval times of the two methods were compared. Surprisingly, using the Positions Table was more time-intensive because of the time needed to retrieve all of the needed rows for each flight. We found that the database access time saved by storing a flight’s position report information with the rest of the flight information in the Flight Table outweighed the time saved by storing position reports in a parsed format in the Positions Table. Thus we decided to use the original method of saving position reports since it was found to be both space and time efficient.

2.1.3 Additional Flight Table Columns

In building a flight database archive, we consider that when creating maps from database information it is useful to know roughly where each flight traveled. For each flight in which position reports are received, we save the farthest observed north, south,
east, and west points traveled. When retrieving flight data we can then know the
“bounding box” for each flight without having to examine all of its position reports.
Thus when displaying flight information on a map, we can discard any flights whose
bounding box does not intersect with the map. We created four additional columns in the
Flight Table to save this information, as depicted in Table 2.4.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Type</th>
<th>Allow Null</th>
<th>Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>farnorth</td>
<td>Far north latitude traveled</td>
<td>Number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>farsouth</td>
<td>Far south latitude traveled</td>
<td>Number(4)</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>fareast</td>
<td>Far east longitude traveled</td>
<td>Number(5)</td>
<td>Yes</td>
<td>5</td>
</tr>
<tr>
<td>farwest</td>
<td>Far west longitude traveled</td>
<td>Number(5)</td>
<td>Yes</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2.4: Additional Flight Table Columns

2.2 Modified Flight Resolution Algorithm

When receiving position report messages from the ASDI data stream, it becomes a
challenge to match a position report message (referred to as a “TZ message” in ASDI) to
the flights being tracked by the ASDISaver, as outlined in subsection 5.4.4 of Gutman’s
paper [2]. The challenge comes from the possibility of there being two or more in-air
flights that share the same aircraft identifier (an “aircraft id collision”). The aircraft
identifier is the only flight distinguishing characteristic included in a TZ message, other
than the computer identifier (cid) given to that flight by the ASDI reporting station. The
cid is useful to a limited extent when flight resolution is required. Its usefulness is
limited for several reasons. First, ASDI reporting stations do not always assign a cid to
each flight. Secondly, each flight passes by multiple reporting stations on its journey,
and each reporting station gives a flight a different cid. Thus a cid is not always reliable
because it may change between two points in time.
Gutman created a TZ message flight resolution algorithm for the ASDISaver, reproduced here:

\[(\text{acid} = \text{aircraft id}, \text{cid} = \text{computer id})\]

\[
\begin{align*}
\text{if (more than one in-air flight with same acid)} \\
\quad \text{if (message contains cid and at least one in-air flight with same acid / cid)} \\
\qquad \text{if (exactly one in-air flight with same acid / cid) match with that flight} \\
\qquad \text{else match message with latest in-air flight with same acid / cid} \\
\text{else (if cid does not help)} \\
\text{\hspace{1em} match message with latest in-air flight with same acid}
\end{align*}
\]

The majority of aircraft id collisions are caused by ASDI report errors, such that one plane is seen as two different flights by the ASDISaver. This occurs chiefly with multiple-legged flights. Suppose a plane with aircraft id ABC1234 travels from point A to point B and there is no arrival message received when it lands at point B. If it takes off again in a short amount of time, the ASDISaver receives a flight plan and departure message for flight ABC1234 going from point B to point C. Since the ASDISaver decides whether a flight is unique from other flights by its (aircraft id, departure point, destination) information, there will now be two separate flights with the ABC1234 flight identifier. This type of aircraft id collision also occurs when flights change their destination in-air and no amendment message is put in the ASDI stream. This is why Gutman resolved aircraft id collisions by assigning a position report to the latest updated in-air flight. While this flight resolution works well much of the time, it fails when there are simply two different planes with the same aircraft identifier in the air at the same time. In this case the ASDISaver should compare the distance between the position report location and the last updated position of each flight involved in the aircraft id
collision. We say that such a distance is *plausible* if it is possible that a plane could have traveled that distance in the amount of time between the last updated time of the flight and the position report timestamp. The modified flight resolution algorithm is as follows:

if (more than one in-air flight with same acid)
  if (message contains cid and at least one in-air flight with same acid / cid)
    if (exactly one in-air flight with same acid /cid )
      match message with in-air flight with same acid/cid
    else (if more than one in-air flight with same acid/cid)
      if (collision-caused-by-one-plane)
        match message with latest updated in-air flight with same acid / cid
      else
        do distance-heuristic
  else (if cid does not help)
    if (collision-caused-by-one-plane)
      match message with latest updated in-air flight with same acid
    else
      do distance-heuristic
  else throw away this report, because system cannot intelligently assign it to any flight

procedure collision-caused-by-one-plane:
  if (departure point of one flight equals destination of another for all colliding flights)
    return true
  else if (destinations are same, or departure points are same for all colliding flights)
    return true
  else return false;

procedure distance-heuristic:
  if (distance is plausible for more than one flight) OR
    (distance is not plausible for any flight ) OR
    (distance is a mix of plausible and unavailable)
    throw away this report, because system cannot intelligently assign it to any flight
  else if (distance is unavailable for one and implausible for all others)
    match with flight whose distance is unavailable
  else if (distance is plausible for only one flight and implausible for all others)
    match with flight whose distance is plausible
The ASDISaver now keeps records of aircraft id collisions and their subsequent flight resolutions. The following is a sample excerpt of these records. Times are in Java millisecond format, and distance is in nautical miles.

acid collision:
OPT184/410 TEB NEW time:1021595192000 distance:0.9616
OPT184/null TEB MSY time:1021595205000 | distance comparison unavailable
flight chosen by time:OPT184/null with time 1021595205000

acid collision:
EGF639/null PLL RDU time:1021517715000 | distance:9.9575 |
   sec since last update:66
EGF639/420 DFW AMA time:1021517704000 | distance:1166.3760 |
   sec since last update:58
flight chosen by distance algorithm:EGF639/null with distance 9.9575

2.3 Archival System Robustness and Reliability

2.3.1 Database Error Management

As described in subsection 5.5.2 of Gutman’s thesis paper, the ASDISaver is designed to save flight records locally in the event of a database outage. However, it does not always save records locally in the event of other types of database errors. Originally, whether flight records would be saved in the event of a database write request error was conditional upon certain factors, such as whether the error included a database outage. Thus, some flight records were lost as a result of certain database errors. One example of such a situation is where the database runs out of datafile space in which to store flight records. In this example the database is still running but cannot store more flight records until the system administrator adds more usable datafile space. Here, we would desire the ASDISaver to store flight records locally until the database can once again handle write requests. Thus, we have modified the ASDISaver such that if, for whatever reason, a flight record is not successfully written to the database, it is saved locally. Later on the system administrator can then insert these records into the database if needed.
2.3.2 Ensuring Collection of All Available Data

One of the system requirements for the ASDISaver was that “message consolidation and entry into storage has to occur at least as fast as the data rate” [2]. In the year after the implementation of the ASDISaver, the position report transmission rate for each in-air flight was increased from roughly one every five minutes to one every minute. Thus the necessary speed for consolidating ASDI messages sharply increased. We found that the original computer running the ASDISaver was too slow to keep up with the increased influx of ASDI messages, and was only receiving about 30% of the ADSI data stream as a result. Thus, the ASDISaver was moved to a faster machine.

At peak times of the day, the ASDI feed delivers messages at the rate of roughly 70 messages per second. The current machine running the ASDISaver has a 400 MHz processor and 128 MB of RAM. It processes the ASDI messages at the same rate as they are received. Processing these messages causes the machine to use roughly 80% of its processing power.

We must be able to ensure that the ASDISaver is collecting all available data from the ASDI data stream rather than a subset of the available data. The ASDISaver is designed to save any flight record for which a departure, flight plan, or boundary crossing message was received. Thus, to guarantee the collection of all available data, we must 1) verify that the ASDISaver is receiving every message in the ASDI stream, and 2) demonstrate that for every departure, flight plan, or boundary crossing message encountered in the ASDI feed, a corresponding flight record exists in the database.

Each message in the ASDI data stream begins with a 4-digit hexadecimal number denoting where it belongs in the sequence of messages. Messages in the ASDI feed are
ordered according to this hexadecimal numbering system. To ensure that the ASDISaver was receiving all available ASDI messages, we added the functionality of asking the ASDISaver to print each message that it receives to a file. Within a few thousand messages one is able to tell whether the ASDISaver is receiving 100% of the ASDI data stream by determining whether there are any gaps between the hexadecimal numbers of adjoining received messages.

To demonstrate that there exists a corresponding flight record in the database for each departure, flight plan, or boundary-crossing message received, for an entire month we saved the raw ASDI data stream to files on a hard drive. From there we wrote a test program that would extract all received departure, flight plan, and boundary messages and query the database to determine whether a flight record with the required aircraft id existed within an appropriate departure time range.
Chapter 3: VisualFlight System Design

This chapter describes the design of the VisualFlight system, beginning with an overview to give the reader a general picture of how the system functions. We then move on to a detailed description of the necessary basic components used by VisualFlight, followed by an explanation of each specific analysis tool, and conclude with a brief guide to VisualFlight’s graphical user interface.

All code was written in Java version 1.4.0. The VisualFlight system is meant to run on a separate machine from those running the ASDISaver and the Oracle database.

3.1 System Overview


The FlightRetriever manages the VisualFlight system’s connection with the Oracle database. Given a SQL query, it queries the database and returns each database row of flight data in the form of a FlightRecord object. The FlightRecord object houses the data found in a single row of the Flight Table.

The AnalysisTool is actually an abstract term representing a family of tools that perform data analysis on FlightRecord objects. To be an AnalysisTool, an object must have the following functionality: 1) it must know what to do when asked to add a FlightRecord, and 2) it must know what to do when asked to conclude, or finish. To give an example, consider the AnalysisTool that creates a flight route map (termed “MapDisplayTool” in later sections). When asked to add a FlightRecord object, the MapDisplayTool will draw the FlightRecord’s route onto its map. When asked to finish, it will draw some kind of key/legend on the map and then write the map to a graphics file.
viewable by the user. (Section 3.3 provides a detailed explanation of the AnalysisTool family.)

The VisualFlight system is designed to work in three stages: 1) user specification, 2) flight analysis, and 3) conclusion.

3.1.1 User Specification Stage

To begin, the user specifies a section of data to analyze, as well as which AnalysisTool(s) to use in performing the analysis. Once this is done, a data request is sent to the FlightRetriever, which in turn sends the data request to the database. The database returns the requested records to the FlightRetriever.

![Figure 3.1: User specification stage](image)

3.1.2 Flight Analysis Stage

While there are records existing in the FlightRetriever, a request is sent to the FlightRetriever to obtain the next FlightRecord object. Once the FlightRetriever provides a new FlightRecord, each user-specified AnalysisTool is asked to add the current FlightRecord object.

![Figure 3.2: Flight Analysis Stage](image)
3.1.3 Conclusion Stage

When the FlightRetriever’s store of records is exhausted, each AnalysisTool is asked to conclude. Each AnalysisTool then outputs the appropriate result to the user.

![Diagram of Conclusion Stage](image)

**Figure 3.3: Conclusion Stage**

3.2 Basic System Functionality and Capabilities

The VisualFlight system depends on a basic set of tools to achieve its required functionality. The tools described in this section are not the AnalysisTools referred to above, but should rather be thought of as basic system capabilities that are incorporated into AnalysisTools.

3.2.1 Distance Computer

The VisualFlight system must have the capability of calculating the great-circle distance between two specified coordinates on the earth. The challenge comes with the fact that the earth is not a perfect sphere, but rather an imperfect ellipsoid. Many different models of the earth have been created, and distance computations vary slightly depending on the model being used. To compute great circle distances, I will be using the “Haversine formula” with consideration for the ellipsoidal curvature of the earth, because it is simple yet accurate enough for a graphical aviation analysis tool. The
The following explanation is a summarization of excerpts of a document provided by the U.S. Census Bureau [8].

The Haversine formula is as follows:

Given two coordinates with latitude/longitude values (lat1,lon1) and (lat2,lon2):

dlon = lon2 - lon1

dlat = lat2 - lat1

\[
a = (\sin(dlat/2))^2 + \cos(lat1) \cdot \cos(lat2) \cdot (\sin(dlon/2))^2
\]

\[
c = 2 \cdot \sin^{-1}(\min(1,\sqrt{a}))
\]

\[
distance = R \cdot c
\]

where \( R \) = radius of the earth

If the earth were perfectly spherical, knowing that one minute of arc of a great-circle earth is 1.15078 miles, \( R = 3956 \) miles. However, in order to take the ellipsoidal shape of the earth into consideration, the following formulas are used:

\( a \) = the equatorial radius of the earth (surface to center distance)

\( b \) = the polar radius (surface to center distance)

\( e \) = the eccentricity of the ellipsoid = \( \sqrt{1 - b^2/a^2} \)

Given the following values of \( a \) and \( b \), we can compute the eccentricity \( e \):

\( a = 3963 \) mi

\( b = 3950 \) mi

\( e = 0.081082 \)

The radius of curvature of an ellipsoidal Earth in the plane of the meridian at a specified latitude \( \text{lat} \) is:

\[
R' = a \cdot (1 - e^2) / (1 - e^2 \cdot (\sin(lat))^2)^{3/2}
\]

The radius of curvature in a plane perpendicular to the meridian and perpendicular to a plane tangent to the surface at a specified latitude \( lat \) is given by:

\[
N = a / \sqrt{1-e^2 \cdot (\sin(lat))^2})
\]

The computed radius of the earth at a specified latitude \( \text{lat} \) is then approximated as the geometric mean of these two radii of curvature:

\[
R = \sqrt{R' \cdot N} = a \cdot \sqrt{(1 - e^2) / (1 - e^2 \cdot (\sin(lat))^2})
\]

When compared with distance computers using varying ellipsoid-like models on the Internet, this distance computer is always within a 0.2% error margin. Table 3.1
compares the performance of our distance computer with two publicly available Internet distance computers [9,10]. Note that all distances (in statute miles) are rounded to the nearest integer, so the computed error becomes less reliable at small distances.

<table>
<thead>
<tr>
<th>Airport Pair</th>
<th>VisualFlight Distance Computer</th>
<th>Great Circle Mapper</th>
<th>Bali&amp;Indonesia Distance computer</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORD-HKG</td>
<td>7779 mi</td>
<td>7793 mi</td>
<td>7793 mi</td>
<td>0.18</td>
</tr>
<tr>
<td>BOS-TPE</td>
<td>7711</td>
<td>7725</td>
<td>7725</td>
<td>0.18</td>
</tr>
<tr>
<td>ZRH-BBY</td>
<td>2979</td>
<td>2973</td>
<td>2974</td>
<td>0.18</td>
</tr>
<tr>
<td>SEA-ORL</td>
<td>2545</td>
<td>2547</td>
<td>2547</td>
<td>0.08</td>
</tr>
<tr>
<td>LAX-BOS</td>
<td>2606</td>
<td>2611</td>
<td>2611</td>
<td>0.19</td>
</tr>
<tr>
<td>SEA-BOS</td>
<td>2492</td>
<td>2495</td>
<td>2496</td>
<td>0.14</td>
</tr>
<tr>
<td>LAX-ORL</td>
<td>2209</td>
<td>2213</td>
<td>2214</td>
<td>0.20</td>
</tr>
<tr>
<td>LAX-SEA</td>
<td>955</td>
<td>953</td>
<td>954</td>
<td>0.15</td>
</tr>
<tr>
<td>BOS-DCA</td>
<td>399</td>
<td>399</td>
<td>399</td>
<td>0.00</td>
</tr>
<tr>
<td>BOS-PVD</td>
<td>49</td>
<td>49</td>
<td>49</td>
<td>0.00</td>
</tr>
<tr>
<td>OQU-PVD</td>
<td>8.8</td>
<td>8</td>
<td>9</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**Airport Code Key**

<table>
<thead>
<tr>
<th>Airport</th>
<th>City, State/Country</th>
<th>Latitude/Longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOS</td>
<td>Boston, MA</td>
<td>42°21'52&quot;N 71°00'19&quot;W</td>
</tr>
<tr>
<td>BBY</td>
<td>Bambari, Central African Republic</td>
<td>05°30'00&quot;N 20°23'00&quot;E</td>
</tr>
<tr>
<td>DCA</td>
<td>Washington, D.C</td>
<td>38°51'07&quot;N 77°02'16&quot;W</td>
</tr>
<tr>
<td>HKG</td>
<td>Hong Kong</td>
<td>22°19'00&quot;N 113°55'00&quot;E</td>
</tr>
<tr>
<td>LAX</td>
<td>Los Angeles, CA</td>
<td>33°56'33&quot;N 118°24'29&quot;W</td>
</tr>
<tr>
<td>ORD</td>
<td>Chicago, IL</td>
<td>41°58'47&quot;N 87°54'16&quot;W</td>
</tr>
<tr>
<td>ORL</td>
<td>Orlando, FL</td>
<td>28°32'44&quot;N 81°19'59&quot;W</td>
</tr>
<tr>
<td>OQU</td>
<td>North Kingstown, RI</td>
<td>41°35'50&quot;N 71°24'44&quot;W</td>
</tr>
<tr>
<td>PVD</td>
<td>Providence, RI</td>
<td>41°43'26&quot;N 71°25'42&quot;W</td>
</tr>
<tr>
<td>TPE</td>
<td>Taipei, Taiwan</td>
<td>25°05'00&quot;N 121°13'00&quot;E</td>
</tr>
<tr>
<td>ZRH</td>
<td>Zurich, Switzerland</td>
<td>47°28'00&quot;N 08°33'00&quot;E</td>
</tr>
</tbody>
</table>

**Table 3.1: Distance Computer Comparisons**

### 3.2.2 Airport Database

In developing graphical tools for aircraft analysis, it is necessary to compile a database of airport information consisting of at least two columns: airport identifier and airport location (latitude/longitude coordinate). For example, in order to perform a simple task such as computing the distance between two airports, the VisualFlight system must first know where the airports are located.
Two websites were chosen to help build an airport database for the VisualFlight system. The first is the Airnav website, which contains official FAA information for every U.S. airport [11]. The second is a Great Circle computer website which stores coordinate information for most international airport codes [9]. Whenever a coordinate location for an airport code is needed, if the location information is not in the airport database, the Airnav website is queried first. If not successful, the Great Circle computer website is queried. If the necessary information is found at either website, it is then stored into the airport database so that the VisualFlight system will never have to query a website for that airport’s location information ever again.

3.2.3 Map Creation

The first step to map creation is finding a way to draw lines, shapes, etc., onto some kind of virtual canvas. Fortunately, Java 1.4.0 includes functionality for basic drawing techniques.

The second step is to decide whether to treat latitude and longitude lines as a grid where longitude lines are parallel to one another, or to acknowledge that in reality longitude lines become closer to one another as the degree of latitude increases. FlightExplorer treats the world map as a grid, where the distance between longitude lines is about \( \frac{2}{3} \) of the distance between latitude lines. As a result, political boundaries (especially nearer to the equator) look as if someone has taken an image of a map with correct width-to-height ratio and adjusted the width to be smaller without adjusting the height of the image. In comparison, U.S. sectional maps are drawn in such a way that the ratio of the distance between longitude lines to the distance between latitude lines varies with the degree of latitude. For example, on a New York sectional map, this ratio is
about .77 at 40 degrees latitude, which is the same as the mathematically computed ratio. For simplicity’s sake, VisualFlight will draw maps in the grid fashion, like FlightExplorer, and will assign the distance between longitude lines to be the same as that between latitude lines. The reasoning here is that the maps created by VisualFlight will be used to observe trends in flight routes and flight density, which do not require a completely accurate longitude vs. latitude distance ratio. Thus, whereas FlightExplorer maps look as if they have been longitudinally compressed, maps created by VisualFlight will look a bit longitudinally stretched, especially as latitude value increases.

The third step in map creation is to implement the ability to translate between latitude/longitude coordinate values and map image pixel values. First, the coordinate boundaries of the map must be defined. This is done by selecting coordinate values for the map’s top-left and bottom-right corners. Next, a map scale, or the number of pixels per nautical mile, must be defined. Once the map scale is known, simple algebra produces the pixel width and height of the map image, as well as the ability to translate between coordinate and pixel values. Now, it is simple to plot a flight’s route onto the map. One need only translate the coordinate value of each position report to a pixel value and draw lines between them, like a virtual connect-the-dots game.

Now, even with latitude/longitude boundaries defined for a map and the ability to plot flight information, it is difficult for a map to be useful without drawing some recognizable political boundaries. Fortunately, the U.S. Census Bureau makes U.S. State boundary information available to the public [12]. Boundary information is provided in three different types of file formats on the Census website. The files in raw ASCII format are sufficient for the needs of the VisualFlight system. The ASCII text files are
formatted such that U.S. boundary lines are treated as a collection of separate individual polygons. (For example, one polygon could be the boundary of one state or one island.) The coordinates of the vertices for each polygon are given in the files, and once those are translated to map pixel values, we need only connect the vertices with lines, and map boundaries will have been drawn.

The final step needed in creating a map is to save map information to an image file. Fortunately, Java 1.4.0 is the first-ever version of Java to include built-in functionality for creating images in popular image file formats. Thus, once a map image is finished on a virtual canvas, it is a simple task to write it to an image file.

3.2.4 Graph Creation

At first, we considered building our own graphing tools in Java, since Java 1.4.0 does not come with built-in graphing functionality. However, we soon realized that using Microsoft Excel could satisfy all necessary VisualFlight graph functionality. Instead of creating files in Excel file format, which is rather difficult, we found that Excel is able to incorporate delimited text files and place the data between delimiters into separate table cells in Excel worksheets. Thus, when in need of creating a graph, the VisualFlight system simply writes data to a text file in whatever format is necessary for the graph in an Excel worksheet. The only constraint is that Excel worksheets only allow up to 256 columns. Fortunately however, Excel worksheets allow up to 65,536 rows.

3.3 Implemented Analysis Tools

The VisualFlight system uses tools in the AnalysisTool family to carry out its data analysis. Every tool in the AnalysisTool family analyzes a body of data and presents a
finished product to the user in its own way. Just as each unique tool in a toolbox has a different function, each unique AnalysisTool performs a distinctive type of data analysis, which may result in its creating a map or graph for the user to view.

As described above, each tool in the AnalysisTool family is required to implement its own `addFlight` and `finish` methods. The VisualFlight system has been designed in such a way that its functionality is easily extensible. Programmers wishing to add different types of data analysis tests to the system need only implement new AnalysisTools, and the VisualFlight system will know how to use them to perform data analysis.

A number of AnalysisTools have been implemented to demonstrate the capabilities of the VisualFlight system, and are described in this section. Graphical data results are detailed in Chapter 4.

### 3.3.1 Map Related Analysis Tools

- **MapDisplayTool**

  **Description**

  A MapDisplayTool displays flight routes on a map. Users specify the boundaries of the desired map, as well as a color-coding scheme for the map. For each flight, a line representing the route flown by a plane will be drawn on the map. Users have the option of having position report markings drawn on the route line, depicting where each position report was received. Route lines and position report markings may be colored by any of the following schemes: aircraft type, altitude, speed, origin airport, destination airport, or transit (that is, whether entering, departing, or passing through the area depicted by the map).
AddFlight Method

When asked to add a FlightRecord, the MapDisplayTool determines what color(s) to use in drawing the flight’s route and report markings on the map. From there it draws the flight route (and position report markings, if requested) in the appropriate color(s) by “connecting the dots” between the flight’s position report coordinates on the map.

Finish Method

When asked to finish its task, the MapDisplayTool first draws the appropriate state boundary lines on top of all the route lines that have been drawn. It then adds a color key to explain how the map was colored, and adds appropriate text to record map boundary values, the number of flight routes drawn, and the parameters of the data being displayed, such as the time frame and the SQL statement used to retrieve the examined flights from the data archive. Finally, the map image is saved to a file in JPEG format under a filename specified by the user.

- MapDensityTool

Description

A MapDensityTool depicts the density of air traffic over an area represented by a map. This density map is displayed as a grid structure, such that each square tile of the grid will be painted a certain color based on the number of planes that travel over the area represented by the tile. Users must specify the boundaries of the map as well as an altitude range to be considered. Also, users must choose a resolution value for the density map. The resolution value is the pixel width of a single grid tile.
AddFlight Method

The MapDensityTool keeps count of how many flights have flown over each grid tile. When asked to add a FlightRecord, the MapDensityTool computes the route flown by the plane based on its position report coordinate information. It then increments the flight count for each appropriate grid tile, provided that the plane is within the specified altitude range when traveling through the area represented by each tile.

Finish Method

When asked to finish, the MapDensityTool examines the flight count for each grid tile and computes appropriate flight count ranges for each of its ten colors. Grid tiles that have a flight count of zero will be colored white. Every other grid tile will be colored according to its flight count as specified by the computed number ranges. Number ranges for each color will vary depending on the body of data being examined, in order to prevent any color from dominating the map. There should roughly be an equal amount of each color so that the map displays as much information as possible. Once each grid tile is colored, the appropriate state boundary lines are drawn on the map. A color key is drawn to record the flight count ranges for each color, and appropriate text is added underneath the map to record the map’s boundary values, time frame, altitude range, resolution value, and the SQL statement used to retrieve the examined flights from the data archive. Once this is done, the map is saved to a file under a filename specified by the user.
3.3.2 Aircraft/Airport Operations Analysis Tools

- **TimeFlownCodeshareTool**

  *Description*

  Every carrier, or commercial, flight in the ASDI data stream has a codeshare. The codeshare is usually the first three characters of the flight’s aircraft identifier. Flights are assigned codeshares based on the corporation owning the flight. For example, all United Airlines flights have the codeshare “UAL”. With a few exceptions, general aviation aircraft identifiers do not include codeshares. Most general aviation aircraft identifiers are simply the aircraft’s registered N-number.

  The TimeFlownCodeshareTool is designed to compute the number of hours flown each day by flights belonging to each codeshare. General aviation flights are grouped together as one “codeshare”.

  *AddFlight Method*

  The TimeFlownCodeshareTool keeps track of the number of hours flown for each encountered day and codeshare. When asked to add a FlightRecord, the TimeFlownCodeshareTool first determines the codeshare of the flight. If the flight is a general aviation flight, it is assigned the codeshare “N--”. It then determines which day the flight departed, the number of hours flown by the flight, and updates the sum of hours flown for that day and codeshare accordingly.

  *Finish Method*

  To finish, the TimeFlownCodeshareTool records its compiled data to a file under a filename specified by the user. It writes the data in a way such that the user can open the
file in Microsoft Excel and create a line graph from the resulting data to depict the number of hours flown for each codeshare versus each encountered day.

- **TimeFlownAircraftTool**

  *Description*

  The TimeFlownAircraftTool functions exactly like the TimeFlownCodeshareTool except that it computes the number of hours flown each day according to aircraft type instead of aircraft codeshare. The TimeFlownAircraftTool does not make a distinction between commercial and general aviation flights.

  *AddFlight Method*

  The TimeFlownAircraftTool keeps track of the number of hours flown for each encountered day and aircraft type. When asked to add a FlightRecord, the TimeFlownAircraftTool determines the flight’s aircraft type, the day that the flight departed, and the number of hours flown. It then updates the sum of hours flown for that day and aircraft type accordingly.

  *Finish Method*

  To finish, the TimeFlownAircraftTool records its compiled data to a file under a filename specified by the user. It writes the data in a way such that the user can open the file in Microsoft Excel and create a line graph from the resulting data to depict the number of hours flown by each aircraft type versus each encountered day.
• AirportOperationsTool

**Description**

The AirportOperationsTool is designed to count the number of operations (arrivals and departures) for each encountered airport and day.

**AddFlight Method**

When asked to add a FlightRecord, the AirportOperationsTool determines the flight’s day of departure, departure airport, day of arrival, and arrival airport. It then increments the number of operations at the departure and arrival airports for their respective days.

**Finish method**

To finish, the AirportOperationsTool records its compiled data to a file under a filename specified by the user. When the file is opened in Microsoft Excel, the user is able to create a line graph to depict the number of operations at each airport versus each encountered day.

• AirportGACarrierTool

**Description**

The AirportGACarrierTool is designed to determine the ratio of carrier, or commercial flights to general aviation flights at each encountered airport.

**AddFlight Method**

When asked to add a FlightRecord, the AirportGACarrierTool determines whether the flight is a carrier or general aviation flight by examining its aircraft identifier codeshare...
(see the description of the TimeFlownCodeshareTool). It then determines the flight’s departure and arrival airports and increments the flight count at each airport accordingly.

**Finish Method**

When asked to finish, the AirportGACarrierTool saves its compiled information to a file under a filename specified by the user. The user can then use the data in the file to create a bar graph depicting the number of operations at each airport, where each bar is colored to display the ratio of carrier to general aviation flights.

### 3.3.3 Aircraft Performance Analysis Tools

- **TimeDistanceTool**

**Description**

The TimeDistanceTool is designed to display the average time flown versus distance flown for each encountered aircraft type. Note that the distance flown by a flight is determined by the distance between the flight’s departure and arrival airports, rather than by the actual distance flown by the plane. This tool can be used to determine whether planes with faster cruise speeds actually have an advantage over planes with slower cruise speeds given a distance to be flown.

**AddFlight Method**

When handed a FlightRecord, the TimeDistanceTool first determines whether the departure and arrival times for the flight are reliable. Flights for which both a departure and arrival message was received are considered reliable. If a flight is not reliable, it is not considered in the TimeDistanceTool’s calculations. If a flight is reliable, the TimeDistanceTool determines the flight’s aircraft type, the distance between the flight’s
departure and arrival airports, and the flight’s time in the air. If either the departure or arrival airport location information cannot be located, the flight is not considered in the TimeDistanceTool’s calculations. If the distance flown is determined, the flight’s distance and time flown is added to the information stored for the flight’s aircraft type.

Finish Method

To finish, the TimeDistanceTool writes data to a file in such a way that the user is able to create a XY scatter graph where each encountered flight is plotted according to time flown (y-value) and distance flown (x-value) and colored by aircraft type. The user can then use Microsoft Excel’s trendline feature to add a best-fit line for each aircraft type in order to compare the average time vs. distance flown for each type of aircraft.

- SpeedDistanceTool

Description

The SpeedDistanceTool is similar to the TimeDistanceTool, except that instead of displaying the average time flown, it displays the average observed speed versus distance flown for each aircraft type. Again, the distance flown by a flight is determined by the distance between the flight’s departure and arrival airports, rather than by the actual distance flown by the plane.

AddFlight Method

When handed a FlightRecord, the SpeedDistanceTool first determines whether the flight received any position reports. Without position reports, a flight has no observed speeds, and thus these types of flights are not considered. If the flight has received position reports, the SpeedDistanceTool then computes the distance between the flight’s departure
and arrival airports. As with the TimeDistanceTool, a flight whose departure or arrival airport location information cannot be located is not considered. The SpeedDistanceTool then calculates the flight’s average speed simply by adding the observed speeds together and dividing by the number of received position reports. We considered calculating a flight’s average speed by weighting each observed speed according to the distance flown since the previous position report. This method was tested and found to vary from the simple average computation only by about .02%. Since we have no way of determining if any deviation in speed occurs between position reports, the simple method of computing a flight’s average speed was chosen.

**Finish Method**

The SpeedDistanceTool finishes in a similar manner to the TimeDistanceTool. Its data is written to a file under a filename specified by the user. The file is written in such a way that the user is able to create a XY scatter graph where each encountered flight is plotted by it’s average observed speed (y-value) and distance flown (x-value) and colored by aircraft type. Again, the user is then able to use Microsoft Excel’s trendline feature to add a best-fit line for each aircraft type in order to compare the average speed versus distance flown for each type of aircraft.

- **DistanceCDFTool**

  **Description**

  The DistanceCDFTool is designed to display a cumulative distribution function (CDF) of the observed distances flown by each type of aircraft. Like the other tools described in this section, the distance flown by a flight is determined by the distance between the
flight’s departure and arrival airports, rather than by the actual distance flown. This tool is used to graphically determine one aspect of the usage of each aircraft type.

AddFlight Method

For each FlightRecord, the DistanceCDFTool computes the distance flown between the flight’s departure and arrival airports. Flights for which airport information cannot be located are not considered. The distance flown is then stored according to the flight’s aircraft type.

Finish Method

When asked to finish, the DistanceCDFTool writes data to a file such that the user can create a graph displaying for each x-axis distance \( d \) the percentage of aircraft encountered for each aircraft type that flew a distance greater than or equal to \( d \).

3.4 Graphical User Interface

A graphical interface was created for the VisualFlight system in order to enable users to specify the necessary parameters for their desired data analysis as intuitively and efficiently as possible. Users first specify what body of data they want to analyze, and then specify which AnalysisTools they would like to use to analyze the selected data. Figure 3.4 displays a screen shot of the VisualFlight graphical interface as it appears when the VisualFlight program is initialized. The graphical interface is divided into three sections: data filters, map creation, and graph creation.
3.4.1 Filters: Data Specification

The user must first limit the desired body of data to a specific time frame. The ASDI data archive is partitioned by flight departure time, and when no time frame is specified, the VisualFlight system must search the entire database, a task that would take an unreasonable amount of time.
The time frame can be specified either by departure or arrival time. The user must specify a lower time bound but is not required to specify an upper time bound. Time bounds are given in YYYYMMDD hhmmss format, where Y = year, M = month, D = day, h = hour, m = minute, and s = second.

Next, the user decides which filters, if any, should be used to further specify a section of data within the chosen time frame. Clicking on the “Add Filter” button will display the dialog shown in Figure 3.5.

![Filter Parameter Selection Dialog](image)

**Figure 3.5: Filter Parameter Selection Dialog**

A filter encapsulates a description of the attributes of the desired data. The user indicates these attributes via the filter dialog shown in Figure 3.5. Parameters left as the wild-card character “*” will not be considered part of a filter. If more than one filter is specified by the user, the data selected from the database will meet the requirements of at least one of the filters (i.e., filters are joined together with logical ORs.) The exception to this rule is...
if the user selects the “AND” checkbox for a filter, as shown in Figure 3.6. Every flight selected from the database will meet the description of every filter whose “AND” checkbox is selected.

**Figure 3.6: Sample Data Filters**

In the sample set of data filters shown in Figure 3.6, the user would like to analyze all flights departing after April 5, 2002 at 12:00am and arriving before April 12, 2002 at 12:00am, for which the departure airport was BOS and the aircraft type was C550 or any of the Boeing 700 numbers. The ‘%’ character functions like the SQL wild-card ‘%’ character, denoting in this example that any aircraft type that begins with ‘B7’ will fulfill the requirements of the filter.

### 3.4.2 Map Creation

Users can use the map creation section of the graphical interface to specify the necessary attributes for the MapDisplayTool(s) and MapDensityTool(s) they would like to utilize. The user begins by selecting the desired pixel width and height of a map image. Then, the user must specify the map’s boundaries. Clicking on the “Select Map” button will display the map selection dialog shown in Figure 3.7. In this example, the user has specified a map width of 500 pixels and a map height of 400 pixels.
Figure 3.7: Map Selection Dialog

The user selects the map’s boundaries by dragging a rectangle on top of the U.S. map (see Figure 3.8), and then pressing the “Get Selected Map” button. The “Back” and “Forward” buttons function similarly to buttons of the same name found on Internet browser interfaces, and allow a user to navigate between previously created maps. In Figure 3.8 the user has dragged a boundary around a portion of the New England area. After selecting the “Get Selected Map” button, a new map of the selected area is created to fit within an image with the specified pixel width and height, as seen in Figure 3.9. Once the user is satisfied with the current map he may select “OK – Use This Map” and exit the map selection dialog.
Figure 3.8: Map Boundary Selection

Figure 3.9: Newly Created Map Using User’s Selected Boundaries
Once a map is selected, it may be added to the list of route maps and/or density maps to be created. This is done via the “Add Selected Map” button found in each map category’s section (see Figure 3.10.) Each time the “Add Selected Map” button is pressed, a new row appears in the adjoining table. Each row in the table represents a unique map tool on which the data specified in the Filters section will be displayed.

![Create Maps From Selected Data](image)

**Figure 3.10: Selection and Specification of Map-related Analysis Tools**

For the list of MapDisplayTools to be used, the user may select the coloring scheme and filename of each route map by clicking on and editing the appropriate table cells. For the list of MapDensityTools, the user may select the filename, altitude range, and resolution value parameters for each density map in the same manner. Note that there is no limit on the number of maps to be created. Users may continue to select new maps and add them to the list of route and/or density maps to be created, since they may find it useful to display the same section of data on maps of different scales, resolutions, altitude ranges, and/or coloring schemes.
3.4.3 Graph Creation

The final section of the graphical interface allows the user to specify which graph-related tools to use in analyzing the section of data specified in the Filters portion of the interface. Unlike the map creation section where an unlimited number of map tools could be specified and used on a single body of data, it is not useful to run more than one instance of a graph-related tool on a section of data. Thus, all possible graph-related tools are listed in a table, and the user indicates which tool(s), if any, should be utilized by checking the appropriate checkboxes. The filenames to be used for each tool may be edited at the user’s discretion.

![Table of Create Graphs](image)

**Figure 3.11:** Graph-related Analysis Tool Specification Example
Chapter 4: Graphical Results

This chapter demonstrates the capabilities of the VisualFlight system by displaying some of the maps and graphs created by VisualFlight in analysis of one month of archived data. The ASDISaver was run continuously for the month of April, 2002. The data reliability tests described in subsection 2.3.2 were run throughout the month to make sure that all possible flights were being collected from the ASDI data stream.

4.1 Aircraft Route Maps

Flight route maps may be used for many different purposes. This section gives just a few examples.

With so many types of aircraft flying within the National Airspace System, it may be useful to know where each type of aircraft flies. Prospective airplane buyers, for example, may want to know where aircraft of a certain type are flown in order to compare those routes with the ones that they will be flying. Figure 4.1 displays the routes traveled by Citation 550 jets that departed on April 19 between 0000Z and 2359Z (Z denotes Zulu, or Universal Coordinated Time), with route lines colored by aircraft speed. We observe that the majority of Citation 550s cruise between 300-400 knots.

Viewing airport traffic is another use of aircraft route maps. Figure 4.2 displays the traffic arriving and departing at Boston’s Logan airport between April 15 at 0000Z and April 16 at 0000Z, with routes colored by arrival/Departure. With this map one can see how arriving and departing planes are routed so as to promote collision avoidance. Figure 4.3 displays the traffic arriving at Dayton, OH on the same day, with routes colored according to departure airport.
Figure 4.1: Routes traveled by C550s on April 19, 2002
Figure 4.3: Traffic arriving at DAY on April 15, 2002
4.2 Aircraft Density Maps

Density maps are useful in that they may be used to effectively analyze the usage of our national airspace and get the “big picture” of air traffic.

Figure 4.4 displays the density of aircraft flying below 10,000 feet Mean Sea Level (MSL) over the entire United States. The altitude 10,000 was chosen so as to display mostly departing and arriving traffic, as the vast majority of airports in the United States reside at altitudes well below 10,000 feet MSL. From this density map we can conclude that airport traffic in the United States is concentrated mostly east of the Mississippi River and along the Pacific coast.

Figure 4.5 displays the density of aircraft flying above 10,000 feet MSL over the United States. This map shows predominantly aircraft in cruise, demonstrating that the heaviest flow of air traffic over the United States exists between a handful of locations: San Francisco/Bay area, Los Angeles, Denver, Dallas, Chicago, Atlanta, and the entire New England area.

Figure 4.6 shows a map of the Washington D.C./Philadelphia area. Aircraft are supposed to avoid flying over the nation’s capital, and this map verifies that pilots route their planes around Washington, D.C. It is also interesting to note the well-traveled routes taken by the majority of air traffic.
Figure 4.4: Density of Arriving/Departing Traffic over the U.S.
Figure 4.5: Density of Cruise Traffic over the U.S.
Figure 4.6: Traffic Density over Washington D.C./Philadelphia Area
4.3 Airport Usage

Graphs displaying the number of operations (arriving/departing traffic) at each airport may be used to determine how busy airports are in relation to one another. The AirportOperationsTool was used to generate the number of operations per day for every airport included in the ASDI data stream in the month of April. A screen shot of the Excel data file produced by the AirportOperationsTool is displayed in Figure 4.7.

Figure 4.8 displays the traffic levels of the nation’s ten busiest airports. From this graph one can observe that air traffic levels drop significantly every Sunday.

Carrier, or commercial airlines visit the majority of U.S. airports, but are concentrated among the busiest fifty or so airports. Figure 4.9 shows that the fifty busiest airports receive small numbers of General Aviation (GA) flights. However, Figures 4.10 through 4.12 demonstrate that at airports that rank beneath the 100 busiest airports in the nation, GA flights compose on average roughly half of the landing and departing traffic.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ORD</td>
<td>2220</td>
<td>2532</td>
<td>2488</td>
<td>2531</td>
<td>2368</td>
<td>2402</td>
</tr>
<tr>
<td>3</td>
<td>ATL</td>
<td>2511</td>
<td>2487</td>
<td>2547</td>
<td>2536</td>
<td>2361</td>
<td>2497</td>
</tr>
<tr>
<td>4</td>
<td>DFW</td>
<td>2015</td>
<td>2210</td>
<td>2224</td>
<td>2202</td>
<td>2001</td>
<td>1637</td>
</tr>
<tr>
<td>5</td>
<td>LAX</td>
<td>1750</td>
<td>1815</td>
<td>1929</td>
<td>1932</td>
<td>1933</td>
<td>1792</td>
</tr>
<tr>
<td>6</td>
<td>PHX</td>
<td>1362</td>
<td>1455</td>
<td>1457</td>
<td>1443</td>
<td>1356</td>
<td>1246</td>
</tr>
<tr>
<td>7</td>
<td>MSP</td>
<td>1260</td>
<td>1439</td>
<td>1410</td>
<td>1364</td>
<td>1231</td>
<td>1179</td>
</tr>
<tr>
<td>8</td>
<td>DTW</td>
<td>1346</td>
<td>1421</td>
<td>1457</td>
<td>1444</td>
<td>1330</td>
<td>1314</td>
</tr>
<tr>
<td>9</td>
<td>DEN</td>
<td>1266</td>
<td>1363</td>
<td>1401</td>
<td>1382</td>
<td>1300</td>
<td>1216</td>
</tr>
<tr>
<td>10</td>
<td>CLT</td>
<td>1286</td>
<td>1368</td>
<td>1367</td>
<td>1333</td>
<td>1172</td>
<td>1135</td>
</tr>
<tr>
<td>11</td>
<td>PHL</td>
<td>1350</td>
<td>1364</td>
<td>1413</td>
<td>1383</td>
<td>1134</td>
<td>1121</td>
</tr>
<tr>
<td>12</td>
<td>CVG</td>
<td>1278</td>
<td>1308</td>
<td>1335</td>
<td>1321</td>
<td>1202</td>
<td>1209</td>
</tr>
<tr>
<td>13</td>
<td>STL</td>
<td>1202</td>
<td>1268</td>
<td>1279</td>
<td>1265</td>
<td>1119</td>
<td>1007</td>
</tr>
<tr>
<td>14</td>
<td>IAH</td>
<td>1153</td>
<td>1253</td>
<td>1269</td>
<td>1264</td>
<td>1065</td>
<td>1064</td>
</tr>
<tr>
<td>15</td>
<td>PIT</td>
<td>1168</td>
<td>1232</td>
<td>1227</td>
<td>1219</td>
<td>1131</td>
<td>1049</td>
</tr>
<tr>
<td>16</td>
<td>MEM</td>
<td>1191</td>
<td>1213</td>
<td>1233</td>
<td>1186</td>
<td>987</td>
<td>724</td>
</tr>
<tr>
<td>17</td>
<td>EWR</td>
<td>1202</td>
<td>1211</td>
<td>1285</td>
<td>1270</td>
<td>1092</td>
<td>1067</td>
</tr>
<tr>
<td>18</td>
<td>IAD</td>
<td>1084</td>
<td>1123</td>
<td>1136</td>
<td>1141</td>
<td>972</td>
<td>980</td>
</tr>
<tr>
<td>19</td>
<td>BOS</td>
<td>1069</td>
<td>1080</td>
<td>1149</td>
<td>1165</td>
<td>949</td>
<td>954</td>
</tr>
</tbody>
</table>

**Figure 4.7: Screen Shot of Airport Operations Data**
# Airport Operations vs. Day of Month - Top 10

Day of Month (April)
Figure 4.9: GA vs. Carrier Traffic Levels at Busiest 50 Airports

Figure 4.10: GA vs. Carrier Traffic Levels at Busiest 51-100 Airports
Figure 4.11: GA vs. Carrier Traffic Levels at Busiest 101-150 Airports

Figure 4.12: GA vs. Carrier Traffic Levels at Busiest 201-250 Airports
4.4 Aircraft Usage

The graphs in this section display the number of hours flown by certain aircraft types for each day of the month. The TimeFlownCodeshareTool and TimeFlownAircraftTool were used to compute the number of hours flown each day for every codeshare and aircraft type, respectively, which appeared in the data archive during the month of April. This allowed us to gain an understanding for how much different types of aircraft are flown in relation to one another.

The ten aircraft types that flew the most hours during the month of April are displayed in Figure 4.13. As with the ten busiest airports, we see a decrease in aircraft usage on Sundays.

Figure 4.14 compares the number of hours flown by GA flights with the number of hours flown by the four air carrier companies that flew the most hours in the month of April. This graph demonstrates that general aviation flights play a significant role in our National Airspace System. It is also interesting to note that the least number of GA hours are flown on Saturdays, and that the number of hours steadily increases and peaks between Wednesdays and Fridays before sharply dropping each Saturday.
# Hours Flown vs. Day of Month (Top 10)

![Chart showing hours flown vs day of month for different aircraft types.](image-url)
Hours Flown vs. Day of Month
GA vs. Top 3/Top 4 Air Carriers

Day of Month (April)

- GA Total
- Top 3 Total
- Top 4 Total

Figure 4.14: Comparison of Hours Flown Between GA and Carrier Flights
4.5 Aircraft Performance

The graphs in this section analyze different aspects of aircraft performance. They are particularly useful in comparing expected or advertised performance with actual collected data. For prospective aircraft buyers, these types of graphs may be of interest when comparing observed performance advantages to aircraft prices. We chose to analyze the differences in performance between five types of Cessna aircraft.

Figure 4.15 displays the average time flown versus distance flown for each type of aircraft. Each time/distance pair for each aircraft is plotted on the graph, and a linear best-fit line was applied to each data series. The graph demonstrates that when flying distances of under 200 nautical miles, the difference between the different aircraft is only a matter of a few minutes. However, when flying thousands of miles the difference may stretch to a number of hours.

The average observed speed flown versus distance flown by each aircraft is displayed in Figure 4.16. Logarithmic best-fit lines were used instead of linear ones because the average speed of a plane will not change very much above a certain distance to be traveled. When flying a distance of 2000 miles, the C750 flies quite a bit faster than the other Cessna aircraft.

The cumulative distribution of the distance flown by each aircraft type is shown in Figure 4.17. 50% of the leg lengths flown by the C550, C525, and C560 are less than 400 nautical miles, and 90% are less than 800 miles. On the contrary roughly 50% of the leg lengths flown by the C750 are greater 700 miles, and 10% of C750 flight distances are greater than 1800 miles.
Figure 4.15: Average Time vs. Distance flown
Avg Observed Speed vs. Distance Flown
April 27&28, 2002
Distance Cumulative Distribution
April 27&28,2002

Figure 4.17: Cumulative Distribution of Distance Flown by Aircraft Type
Chapter 5: Conclusion

The VisualFlight system makes it possible to analyze numerous characteristics of the National Airspace System. It provides an important research tool that provides a window into both airspace and aircraft utilization. The system has been designed to be modular, and to admit new analysis methods. We expect that it can be readily adapted to a wide variety of purposes. The ASDI archival system has been proven to be reliable, and continues to collect and archive historical flight data. The wealth of information found in the database may be mined for many other types of analysis other than those found within the pages of this paper. The VisualFlight system has been built in such a way that it can be easily extensible to other types of analysis.

As a whole, the VisualFlight and ASDISaver systems allow powerful and compelling analysis of our national airspace that is as yet largely unpublished. It is our hope that these systems may be used in the future to further air traffic research.
Bibliography


68