

# AeroVision

Aircraft Trajectories  
Optimization and Visualization





# AeroVision

Aircraft Trajectories  
Optimization and Visualization

by

Elvan Kula & Hans Schouten

in partial fulfillment of the requirements for the degree of

Bachelor of Science

in Computer Science

to be defended publicly on Friday July 1, 2016 at 14:00 PM.

Project duration: April 19, 2016 – July 1, 2016  
Thesis committee: Dr. Ir. N. Dintzner, TU Delft, supervisor  
Dr. ir. S. Hartjes, Client  
Dr. ir. M. Larson, Bachelor Project Coordinator

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.





# Summary

Nowadays, major airports are dealing with more than a thousand flights a day leading to considerable impact on the surrounding populated areas in terms of noise. Recent studies show that the most effective way of minimizing noise is the calculation of noise minimal approach routes. The department of Air Transport & Operations (ATO) at TU Delft's Faculty of Aerospace Engineering is a research group that focuses on the understanding of noise impacts, identifying solutions to reduce those impacts and educating people on the issues. To understand flight navigation concepts and support their research findings at conferences, the researchers at ATO need a standalone application for model-based trajectory optimization and visualization of aircraft noise.

The challenge set out by ATO is to build a program that represents an innovative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. There are no existing programs that offer both optimization and visualization of aircraft noise. Therefore, the research department would benefit greatly by an automated and pipelined execution of these processes in a standalone application.

To remedy this issue, we designed and implemented *AeroVision*: an integrated solution to the computation, optimization and visualization of (noise) contours. This includes (a) a generalised model for contour calculation requiring lower computational effort as compared with the current tool used by ATO; (b) a new multi-objective approach to trajectory optimization based on contour areas and population annoyance; (c) an extensible visualization component for 2D and 3D animation in Google Earth.

The final version of *AeroVision* meets the functionality and quality required by ATO. We implemented all requirements despite of one could-have requirement. For trajectory optimization we were able to achieve a performance improvement of 84% by using a multi-core and multi-threaded genetic algorithm based on crossover operations. This led to a major improvement on the workflow management and automation of ATO.



# Contents

<b>Summary</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context . . . . .	1
1.2 Problem Statement . . . . .	1
1.3 Main Contributions. . . . .	2
1.4 Structure . . . . .	2
<b>2 Background</b>	<b>3</b>
2.1 The Need for an Integrated Solution . . . . .	3
2.2 Systems Used by ATO . . . . .	4
2.2.1 INM for Noise Calculation . . . . .	4
2.2.2 NoiseLAss for Trajectory Optimization. . . . .	4
2.2.3 Google Earth for Visualization . . . . .	4
2.3 Related Work . . . . .	5
2.3.1 FlightSight for Trajectory Visualization. . . . .	5
2.3.2 DIRCOL for Trajectory Optimization. . . . .	5
2.3.3 VirtualAir for Trajectory Simulation . . . . .	5
<b>3 Problem Analysis</b>	<b>7</b>
3.1 System Requirements. . . . .	7
3.1.1 Requirements Prioritization . . . . .	7
<b>4 Design of AeroVision</b>	<b>11</b>
4.1 Logic Models . . . . .	11
4.2 Required Input Files . . . . .	11
4.3 System Modularity . . . . .	14
4.4 Pipelined Architecture . . . . .	15
4.4.1 Optimization Pipeline . . . . .	15
4.4.2 Visualization Pipeline . . . . .	16
4.5 User-System Interaction . . . . .	17
4.5.1 The .NET Framework . . . . .	17
4.5.2 Model-View-Presenter Interface . . . . .	17
4.5.3 MDI Forms. . . . .	18
4.6 Noise Calculation . . . . .	19
4.7 Contouring . . . . .	19
4.7.1 The Contouring Algorithm. . . . .	19
4.7.2 Implementation . . . . .	20
4.8 Optimization . . . . .	21
4.8.1 Genetic Algorithm . . . . .	21
4.8.2 Genetic Algorithm description. . . . .	21
4.8.3 Flight Simulation. . . . .	23
4.9 Visualization . . . . .	25
<b>5 Quality Assurance</b>	<b>27</b>
5.1 Unit Testing and Continuous Integration . . . . .	27
5.1.1 Testing Policy . . . . .	27
5.1.2 Testing Framework. . . . .	28
5.1.3 Continuous Integration: AppVeyor. . . . .	28

5.2	Code Analysis . . . . .	28
5.2.1	CoverAlls. . . . .	28
5.2.2	Software Improvement Group . . . . .	28
5.2.3	Code Metrics in Visual Studio . . . . .	29
5.2.4	SourceMonitor. . . . .	30
<b>6</b>	<b>Research, Design, and Development Process</b>	<b>31</b>
6.1	Project Management Processes . . . . .	31
6.2	Research Processes . . . . .	32
6.2.1	User Study . . . . .	32
6.2.2	Feasibility Study . . . . .	32
6.2.3	Requirement Engineering . . . . .	32
6.3	Design Process . . . . .	33
6.3.1	Changed Requirements . . . . .	33
6.4	Development Process . . . . .	34
6.4.1	Trello for Project Management . . . . .	34
6.4.2	Version and Quality Control . . . . .	34
6.4.3	Division of Labour . . . . .	34
6.5	Reflection . . . . .	34
<b>7</b>	<b>Product Validation</b>	<b>37</b>
7.1	Critical Requirements. . . . .	37
7.2	Non-critical Requirements . . . . .	39
7.3	Client Reflection . . . . .	40
<b>8</b>	<b>Ongoing and Future Work</b>	<b>41</b>
8.1	New Functionality . . . . .	41
8.1.1	Reverberation correction in INM. . . . .	41
8.1.2	Integration with another noise model . . . . .	41
8.1.3	Addition of graphs . . . . .	41
8.1.4	Optimization based on weather data. . . . .	41
8.2	Publication of the AeroVision Code . . . . .	42
8.3	Process Improvements . . . . .	42
<b>9</b>	<b>Conclusion</b>	<b>43</b>
<b>A</b>	<b>Research Report</b>	<b>47</b>
<b>B</b>	<b>SIG Evaluation</b>	<b>49</b>
B.1	Initial Evaluation . . . . .	49
<b>C</b>	<b>Product Planning</b>	<b>51</b>
C.1	Introduction . . . . .	52
C.2	Product . . . . .	52
C.2a	Program Overview . . . . .	52
C.2b	Roadmap . . . . .	54
C.3	Release Plan. . . . .	55
C.3a	Initial release plan (milestones, MRFs per release) . . . . .	55
C.3b	Definition of Done . . . . .	57
<b>D</b>	<b>Original Project Description</b>	<b>59</b>
	<b>Bibliography</b>	<b>61</b>

# 1

# Introduction

## 1.1. Context

Nowadays, major airports are dealing with more than a thousand flights a day leading to substantial impact on the surrounding populated areas in terms of noise. Consequently, plans for the expansion of existing airports or the construction of new airports are strongly opposed by the local population. Noise control measures include noise reduction at the origin, soundproofing buildings near airports, practical flight control measures and land use planning strategies [17]. Recent studies show that the most effective way of minimizing noise is the calculation of noise minimal approach routes [7]. Thanks to modern navigation it is possible to define curved approach and departure procedures instead of the conventional straight flight tracks. This allows the definition of environmentally safe procedures that minimize the noise impact on ground.

The department of Air Transport & Operations (ATO) at TU Delft's Faculty of Aerospace Engineering is a research group that focuses on the optimization of aircraft operations for efficiency, safety, cost and environmental impact [5]. ATO'S primary mission is aimed at increasing the understanding of noise impacts, identifying solutions to reduce those impacts and educating people on the issues. To understand flight navigation concepts and support their research findings at conferences, the researchers at ATO need a standalone application for model-based trajectory optimization and visualization of aircraft noise.

## 1.2. Problem Statement

The challenge set out by ATO is to build a program that represents an innovative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. There are no existing programs that offer both optimization and visualization of aircraft noise. At the moment ATO uses multiple tools separately for this purpose, but this requires a lot of user input and lacks automation. The researchers manually transform the output of the noise calculation tool Integrated Noise Model (INM) into the right format with Matlab, and then enter this into the optimization tool NoiseLASS [8]. The research department would benefit greatly by an automated and pipelined execution of these processes in a standalone application.

One of the main goals of this project is to develop a system that is able to calculate noise contours and visualize these contours produced along a trajectory through a 3D animation pictured on a real map. A contour line of a function of two variables is a curve along which the function has a constant value [29]. They are 'footprints' which usually indicate areas of constant noise, but they could also be used to depict other environmental factors, such as gas emissions. Since ATO's research department also focuses on other environmental effects than noise, the desired program should be generic enough to help engineers in assessing the effect of noise and other kinds of pollution in populated areas.

Another project goal is the implementation of trajectory optimization: the computation of noise minimal approach routes for populated areas surrounding airports. The fitness of a flight route needs to be based on a weighted sum of noise contour areas and population annoyance for multi-objective optimization. Next to the course of the actual flight path, the height and speed profiles of the airplane are also being considered for optimization. The client would like the optimization algorithm to especially take contour areas into account. Noise contours are a new subject to ATO'S research group and have not been implemented in relation with trajectory optimization before [35]. Our extension of the algorithm led to a significant improvement on the art of trajectory optimization.

The goals set by ATO lead to the following research questions:

1. How to visualize contours produced over time along a flight trajectory in a 3D animation?
2. How to effectively optimize trajectories for minimum noise and/ or other environmental effects of aircraft using contour areas?

### **1.3. Main Contributions**

This thesis presents the AeroVision project, an effort to improve and automate the contour calculation, optimization and visualization of the environmental impacts of aircraft.

The main contributions of this thesis are:

1. AeroVision, a generic software system to minimize and visualize aircraft pollution along simulated and real flight routes. This includes: (a) a generalised model for contour calculation requiring lower computational effort as compared with the current tool used by ATO; (b) a new approach to trajectory optimization based on contour areas using a genetic algorithm; (c) an extensible visualization component for 2D and 3D animation in Google Earth; (d) an integrated solution to the computation, optimization and visualization of (noise) contours.
2. An evaluation of AeroVision's performance compared to related applications. Experimental evaluation is used to measure its response times and robustness in real-world use cases.

### **1.4. Structure**

This thesis is structured as follows: Chapter 2 provides background information and presents related work that is needed to understand the context. Chapter 3 gives a more detailed analysis of the problem as well as a description of the system requirements. The design of AeroVision is covered in Chapter 4 and the measures taken to increase the quality of the software are described in Chapter 5. The processes used in this project, such as the usage of the Scrum methodology, are described in Chapter 6. In order to validate the final product, a last acceptance test with the client has been done. The results of this test can be found in Chapter 7. Finally, the future of AeroVision is described in Chapter 8.

# 2

## Background

This section provides additional information needed to understand the details of the project. First, Section 2.1 describes the need of ATO for an integrated solution to the computation, optimization and visualization of contours. Section 2.2 provides information on systems that are currently being used by ATO for this purpose. Finally, Section 2.3 describes related work and familiar systems in aircraft visualization.

### 2.1. The Need for an Integrated Solution

The client of the project is Dr. ir. Sander Hartjes. He is employed as an Assistant Professor in the field of airline operations at the chair of Air Transport and Operations [6]. Through our interviews with the client we learned that ATO lacks a visualization tool for its research findings at conferences. This tool needs to implement model-based optimization and 3D visualization of contours in a pipelined manner.

#### **The need for 3D contour animation**

The current visualization used at ATO consists of simple animations of an airplane following a particular flight path in Google Earth. Sander told us that they would like to be able to visualize noise contours over time in a 3D animation. Currently, the ATO researchers create static visualizations of noise contours with 3D model simulation software such as AutoCAD and Matlab, but they have no program to animate noise contours that are produced along a particular trajectory. To increase their understanding of aircraft noise and flight navigation concepts, ATO needs a program that implements dynamic visualization of contours. This way the user has a freedom to select different flight scenarios and compare their results by visualizing them.

#### **The need for trajectory optimization based on contour values**

After consultation with other researchers of the department, Sander told us that they would like our program to implement iterative optimization of flight trajectories for minimum noise. Therefore, an optimization algorithm needs to be implemented that should not only take the actual measured noise volumes into account but also the area of the noise contours. The trajectories would then be optimized based on the volume and spreading of the noise. Recent studies show that the timing and spreading of noise affect the population annoyance greatly and thus they are potentially important factors in the minimization of noise [33]. This way of minimizing aircraft noise and visualizing noise contours is not only new to the ATO department, but also to the research field itself. Noise contours are normally used to regulate sound and not as an indication for the optimization of trajectories [17].

#### **The need for automation**

When Sander showed us the tools that are currently used by the ATO department for trajectory optimization, we noticed that these tools lack automation. They use the tool INM for noise calculations and NoiseLAss for optimization, which are further described in Section 2.2. To exchange the data between INM and NoiseLAss, the researchers manually transform the output of the noise model in the right format and then enter this into the optimization model. For visualization the researchers still manually transform Matlab files containing trajectory coordinates into KML files and enter this into Google Earth for visualization. Sander told us that they increasingly need this process to be automated when dealing with large trajectories. This would spare a lot of their time that goes into manually converting flight data from Matlab into KML files. The research department would benefit greatly by an automated and pipelined execution of these processes in a standalone

application.

## 2.2. Systems Used by ATO

In this section the tools that are currently separately used by ATO for noise calculation, trajectory optimization and visualization are described.

### 2.2.1. INM for Noise Calculation

The Integrated Noise Model (INM) is a noise calculation tool provided by our client. The general noise model is implemented in this tool created by respectively the TU Delft and the Nederlands Lucht- en Ruimtevaartcentrum (NLR) [8]. It implements the FAA's standard methodology for noise assessments. Since 1978 this has been the standard Noise Model in over 65 countries. INM computes noise levels expressed in six time-based metrics at a user-specified regular grid based upon finite flight-segment data. It requires two user-provided text files: one describing the full trajectory expressed in a number of points in space (coordinates) and one defining a two-dimensional (no elevation data) or three-dimensional grid for noise calculation.

The calculations in this tool are conform with world standards and are performed below one second for 25,000 noise grid points. This is done with a precision of about  $\pm 5$  dB directly beneath an airplane's flight path. The tool being compliant with standards and suitable for noise computation, the client decided that the INM tool should be part of the integrated solution of our final program. AeroVision calls INM in case the user does not provide his own noise values as input and wants the program to calculate it for him.

### 2.2.2. NoiseLAss for Trajectory Optimization

NoiseLAss is a noise level assessment tool which implements the contouring algorithm and optimization model [10]. NoiseLAss calculates the noise levels in the enforcement points of a flight trajectory by interpolating the user-supplied noise input data on a specific point. Enforcement points are locations in the vicinity of the airport at which a maximum value for a specific noise metric is defined. This allows the user to define other points at which the noise level is of interest. Given the noise input, NoiseLAss can extrapolate user specified assessment criteria from the calculated noise contours and flight path.

Additionally, the tool contains a number of noise exposure-response relationships to assess the direct impact on population for six different noise metrics. These relationships can be used to quantify the impact of commercial aviation on near-airport communities for daytime and night operations, e.g. the number of expected awakenings due to a single night-time flyover and due to cumulative noise metrics. However, in contrast to AeroVision, NoiseLAss does not support optimization based on contour areas. This is a feature with which AeroVision aims to distinguish itself from NoiseLAss and other optimization tools. As our client let us know, trajectory optimization regarding to contour areas has not been implemented before.

### 2.2.3. Google Earth for Visualization

The client informed us that they prefer Google Earth as a plugin for the visualization since it is commonly used in their research field [9]. Google Earth also provides a lot of options for satellite imagery of the entire earth in an interactive format. Compared to all its alternatives like Marble and Here Maps, Google Earth offers the most complete satellite imagery in 3D. One disadvantage of Google Earth is that the resolution of the image varies depending upon the location. This may make it more difficult for the user to view certain areas that are isolated or uninhabited. But since ATO will use AeroVision mainly for airports located in the Netherlands and in populated areas surrounding airports, this should not be a problem for their application.

KML (Keyhole Markup Language) is a file format used to display geographic data in Google Earth [15]. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard. It enables the user to store animations in Google Earth. Since KML meets all the requirements of our flight animation and since the ATO department is already familiar with this file format, KML seemed a perfect fit for our project. Next to KML we also considered alternatives like CityGML and Geography Markup Languages [22] [30]. Although these alternatives are fine, KML is better integrated with Google Earth. Therefore we decided to use KML files for visualization in our program.

## 2.3. Related Work

Both optimization and visualization have grown in the field of aviation. Researchers and airline companies are interested in minimizing the effects of aircraft on the environment by optimizing their flight trajectories. In the last years several systems related to AeroVision have been developed. In this section some related work will be described and compared to AeroVision. There is no existing program that offers both trajectory optimization and 3D visualization, but there are tools that provide one of the options.

### 2.3.1. FlightSight for Trajectory Visualization

SpaceWorks Enterprises offer a commercial application called FlightSight that supports three-dimensional visualizations and dynamic animations of flight trajectories in Google Earth [16]. It is compatible with path data from different data sources. When the data is ready for post-processing, the user can control the appearance, content and behavior of the visualization. The user can also alter the colors, data overlays, integrated 3D models, camera angles, and more. The program is based on KML importing utilities.

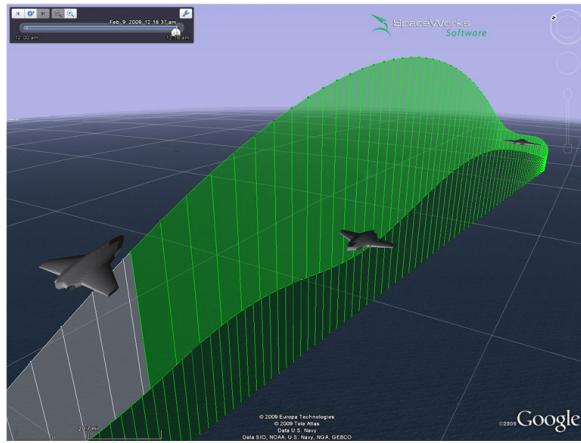


Figure 2.1: A visualization of jets in FlightSight.

While the program offers some interesting visualization options, it does not provide visualization of contours and population annoyance. FlightSight is also one of the most expensive tools in the field. A 1 year commercial license costs \$499.00, which would be a large investment for a research department.

### 2.3.2. DIRCOL for Trajectory Optimization

DIRCOL is a general-purpose trajectory optimization application based on direct collocation [13]. In mathematics a direct collocation method is a method to numerically solve ordinary differential equations, partial differential equations and integral equations [36]. The idea is to choose a finite-dimensional space of candidate solutions and a number of points in the domain (called collocation points), and to select that solution which satisfies the given equation at the collocation points [36]. DIRCOL applies a direct collocation method for trajectory optimization of air and space vehicles. The tool consists of a set of Fortran subroutines designed to solve optimal control problems of systems described by (first order) ordinary differential equations. The user must provide subroutines that define the objective and the constraint functions. Subroutines for derivatives of the problem functions are not required. DIRCOL provides a simple interface with a script editor and is only compatible with Unix systems.

While this tool provides a practical solution to trajectory optimization, it is not that user friendly since it requires the user to program its own optimization profile in a Fortran like scripting language.

### 2.3.3. VirtualAir for Trajectory Simulation

VirtualAir is a framework for distributed air traffic simulation [34]. The framework integrates various third party simulation and visualization software and several original components. It supports the visualization of map symbols, scalar fields, aircraft trajectories and airspaces for research purposes. The software allows plug-and-play integration using the Python programming language.

In contrast to AeroVision, VirtualAir requires a lot of user input for the visualization. First of all, the user needs to download five external software packages in order to be able to run flight simulations. Next, the user

can only run the simulation by giving instructions in an object-oriented scripting language.

# 3

## Problem Analysis

To support their research findings at conferences, the ATO research department needs a standalone application for model-based optimization and visualization of aircraft noise. The goal of the project is to design a program that represents an innovative and efficient way to minimize and visualize aircraft noise and other kinds of pollution along simulated and real flight routes. To visualize the noise produced along a trajectory, a program will be built to calculate and depict contours in a 3D animation on a real map. This should also show the effects of the produced aircraft noise on population annoyance.

### 3.1. System Requirements

Throughout the project, system requirements have been identified before and during development. These requirements are either derived from the research questions listed in Section 1.2, or obtained through interviewing our client. A few requirements were changed during the project because they turned out to be unnecessary or unfeasible. All the other requirements are fulfilled in our final product. In this section only the up-to-date requirements are described. For initial requirements see the Research Report in Appendix A. For requirements that have been changed or discontinued see Section 6.3.1. Table 3.1 provides an overview of the system requirements. The remainder of this section describes the required evolution and execution qualities in detail.

#### 3.1.1. Requirements Prioritization

The way the requirements were gathered using requirements engineering is described in Section 6.2.3. Their priority, expressed as 'critical' or 'non-critical', is based on their contribution to the execution and evolution qualities desired by the client.

##### Execution Qualities

The following execution qualities, which are observable at run time, were required by the client:

- **Compatibility**

The program should be compatible with the noise calculation tool INM that is currently being used by ATO's research department to assess noise values along a trajectory. For future use, the program should also be able to handle the input of other noise tools and other input forms representing alternative environmental effects of aviation. The program should be generic enough to ensure compatibility with numerical output of other calculation tools, but remain flexible to perform contours calculation and visualization in a short period of time.

- **Usability**

To support the tasks that the user would like to perform, AeroVision needs to be able to execute all processes in an automated manner as part of its workflow. It should be able to support the user with two main goals: optimization and visualization. These goals can be achieved by executing any combination of the operations: computation of noise counter, visualization and optimization. The program should be able to operate any combination of these operations based on user input.

Table 3.1: System requirements

#	Critical	Type	Requirement
1	YES	Functional	The program can read in a file specifying the input trajectory and grid
2	YES	Functional	The program can calculate noise contours produced along a particular trajectory
3	YES	Functional	The program can save the calculated contour values as a file
4	YES	Functional	The program is capable of trajectory optimization based on noise contour areas and population annoyance of areas that are affected
5	YES	Non-functional	The program is able to evaluate 2500 trajectories for optimization in under 30 minutes
6	YES	Functional	The program is capable of trajectory visualization in a 3D animation in Google Earth
7	YES	Functional	The program is able to compute and visualize the effects of the aircraft on population annoyance
8	YES	Functional	The program can save the generated animation as a KML file
9	YES	Non-functional	The program is compatible with Windows 7 and later
10	YES	Non-functional	The program executes all processes in an automated and pipelined manner
11	NO	Non-functional	The program should be compatible with input from other (noise) calculation tools
12	NO	Functional	The program can visualize multiple trajectories with animation and static visualization
13	NO	Functional	The program can read in KML files and directly visualize it in Google Earth
14	NO	Non-functional	The program can cancel each (calculation) process while it is running
15	NO	Non-functional	The program offers all tasks in a graphical user interface
16	NO	Functional	The program should show the estimated calculation time and current progress during trajectory optimization

- **Performance**

Since the program will be made for daily use by the ATO researchers, it is critical that the calculations carried out by AeroVision are performed within a short period of time. The noise contour calculations should be done within a second. An optimization run of 2500 evaluations should be done in under 30 minutes.

- **Scalability**

Scalability was prioritized by the client as an evolution quality, which is embodied in the static structure of the system. The program should be able to process large data sets with sizes in the gigabyte range, containing trajectories with a duration up to 30 minutes.

Together with the client we prioritized and arranged the requirements using the MoSCoW method [20]. This is a prioritization technique used in software development to determine the importance the stakeholder places on the fulfillment of each requirement. The term MoSCoW itself is an acronym derived from the first letter of each of four prioritization types (Must have, Should have, Could have, and Would like but won't get). In Table 3.1 the must have requirements are labeled as 'critical' and should have requirements are marked as 'non-critical'. Could have and won't have requirements are described in our Research Report that can be found in Appendix A.



# 4

## Design of AeroVision

The main requirements of AeroVision, as described in Section 3.1, express the need for an innovative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. This requires the implementation of three logic models, which will be described and motivated in Section 4.1. The input files that are required for AeroVision are described in Section 4.2. To support the user with different use cases, AeroVision uses a modular design and pipelined architecture. Both designs are further described in Section 4.3 and Section 4.4, respectively. In Section 4.5 all design choices regarding the graphical user interface are explained. The way noise values and (noise) contours are calculated by AeroVision is described in Section 4.6 and Section 4.7. Our innovative approach to trajectory optimization is made clear in Section 4.8 by explaining the used genetic algorithm and flight simulation model. Finally, the visualization component of AeroVision is described in Section 4.9.

### 4.1. Logic Models

To fulfill the requirements three models needed to be implemented:

1. *Contouring Model*: for the computation of contours. This model will be deployed by ATO's research team to predict aircraft noise and other environmental effects along a particular trajectory.
2. *Optimization Model*: for the optimization of aircraft trajectories. This model will be deployed by ATO to update the trajectory design in an iterated manner to minimize the produced noise and/or other environmental effects over populated areas around airports. The parameters that take part in the optimization of aircraft trajectories include the criterion of contour areas and a number of site-specific criteria based on population density.
3. *Visualization Model*: for the visualization of the calculated contours and their effect on population annoyance. This model will work with the Google Earth plugin for 2D and 3D static and dynamic visualization.

### 4.2. Required Input Files

In this section the input files that are required for the use of AeroVision are described. The trajectory and grid with noise input files are needed for all operations that can be executed with AeroVision. The population file is only required for the calculation and visualization of population annoyance.

- **Trajectory**

A trajectory consists of points representing the positions along the flight path. Each model in AeroVision requires a trajectory as input. A trajectory can be specified with any input file similar to the example in Figure 4.1. Each line in the file corresponds to one trajectory point and each line consists of six data elements indicating the x-coordinate, y-coordinate, z-coordinate, speed and thrust level at the corresponding trajectory point. The x, y, z-coordinates are the distances measured in the xy-plane and height from the reference point of the user specified coordinate system. At the bottom of the file the aircraft model and engine mount can be specified.

- **Grid and noise input**

The Contouring model can read any noise input file which complies with the input template in Figure 4.2. Here x, y and z specify the exact location of the grid point, and the other columns specify the noise metrics. Note that the z-coordinate in this model is not used, and therefore can be left out. The minimum input to the Contouring model is at least three columns consisting of a x- and y-coordinate and one metric. Since the program uses the header of the file to recognize the input, it is important to ensure the correct order and definition of the columns is provided. The two metrics not available in this example can simply be defined in the header as 'Lden' and 'Lnigh'. The noise values should be provided on a regular grid. The cells are not required to be square, but to allow for a proper refinement of the grid it cannot be irregular. Although the contouring model can handle this input template, the model is directly compatible with the output files of INM. This allows the user to provide values derived from any noise model or to let AeroVision calculate the noise values by calling INM.

- **Population data**

The optimization model needs to be supplied with an additional set of data to provide site-specific information on population density. This file consists of three columns in which the first two columns specify the x,y coordinates of addresses and the third column represents the number (or density) of people living at those addresses. An example file is shown in Figure 4.3.

x	y	h	v	t	m
<hr/>					
110658	478103	10.668	400	50000	2
110541	478031	35.34622902861065	400	50000	2
110482	477995	47.70313208377475	400	50000	2
110364	477924	72.43306301161373	400	50000	2
110305	477888	84.80616920697339	400	50000	2

nois_id / engine mount
<hr/>
GP7270
wing

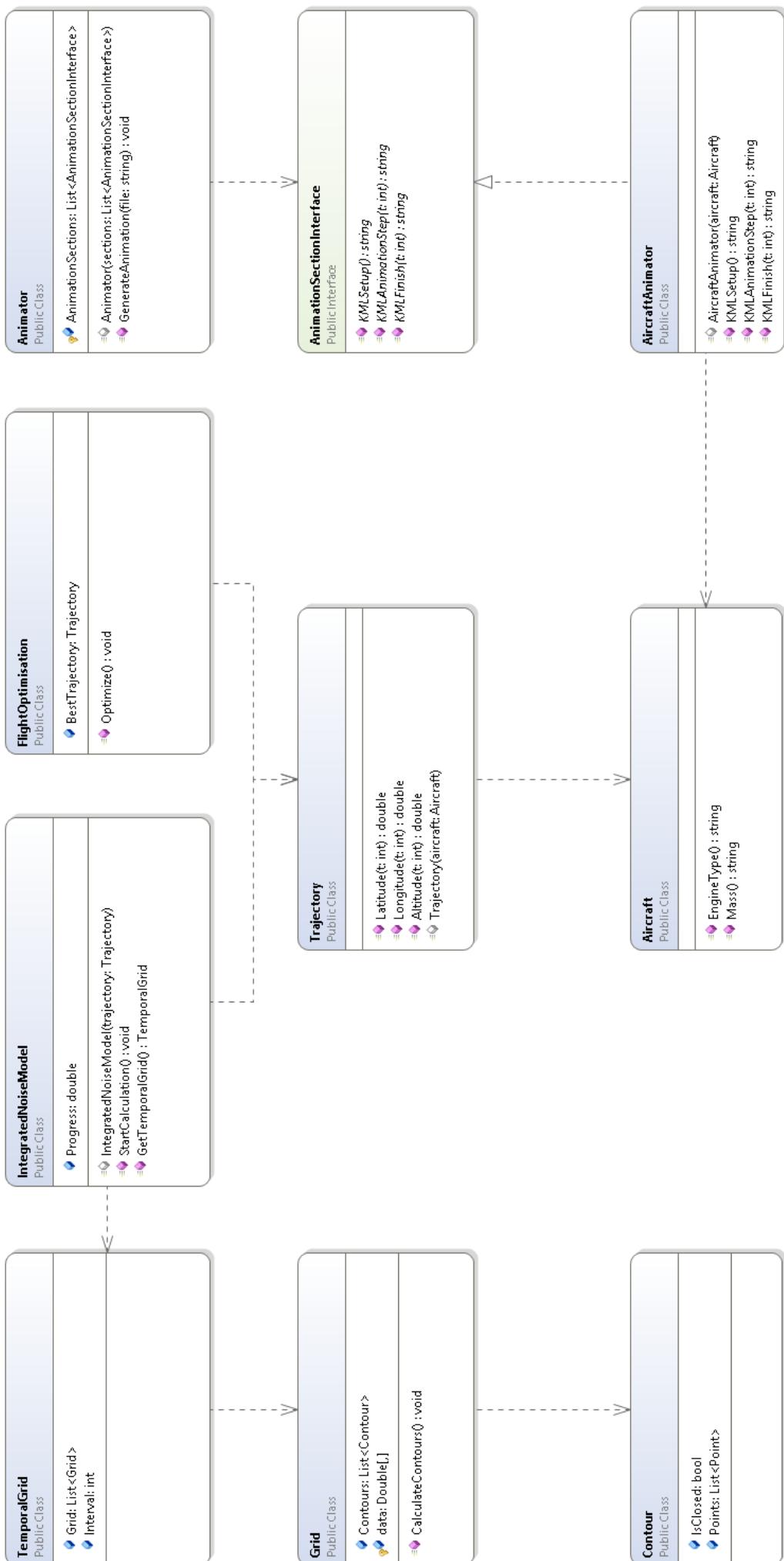
Figure 4.1: The required format for an input trajectory.

x [m]	y [m]	z [m]	SEL [dBA]	LAm <sub>ax</sub> [dBA]	EPNL [EPNdB]	PNLTM [TPNdB]
-5000.00	-5000.00	0.00	36.54	21.85	37.21	27.24
-5000.00	-4875.00	0.00	36.67	22.07	37.69	27.51
-5000.00	-4750.00	0.00	36.80	22.29	38.23	27.79
-5000.00	-4625.00	0.00	36.94	22.51	38.51	28.06
-5000.00	-4500.00	0.00	37.08	22.73	38.11	28.35
-5000.00	-4375.00	0.00	37.22	22.96	37.89	28.63
-5000.00	-4250.00	0.00	37.37	23.19	37.79	28.91
-5000.00	-4125.00	0.00	37.52	23.42	37.76	29.20

Figure 4.2: The required format for noise input.

x [m]	y [m]	pop. density
70001	4.4729e+005	0.9786
70003	4.4681e+005	2.1079
70003	4.4671e+005	2.1079
70005	4.4672e+005	2.1079

Figure 4.3: The required format for the population file.



### 4.3. System Modularity

Because AeroVision needs to support the user with two distinct use cases, optimization and visualization, it uses a modular design. This is a design approach that divides the system into smaller parts, that can be independently created and then used in different systems. Each module knows about all the classes that are necessary to execute one aspect of the desired functionality. These modules are independent, which means that one does not know about the presence of the others. An overview of this design is shown in a UML diagram. Corresponding to the two use cases, AeroVision offers an optimization pipeline and a visualization pipeline. These pipelines are actual paths consisting of modules that are connected in such a way that the right user input leads to the desired output.

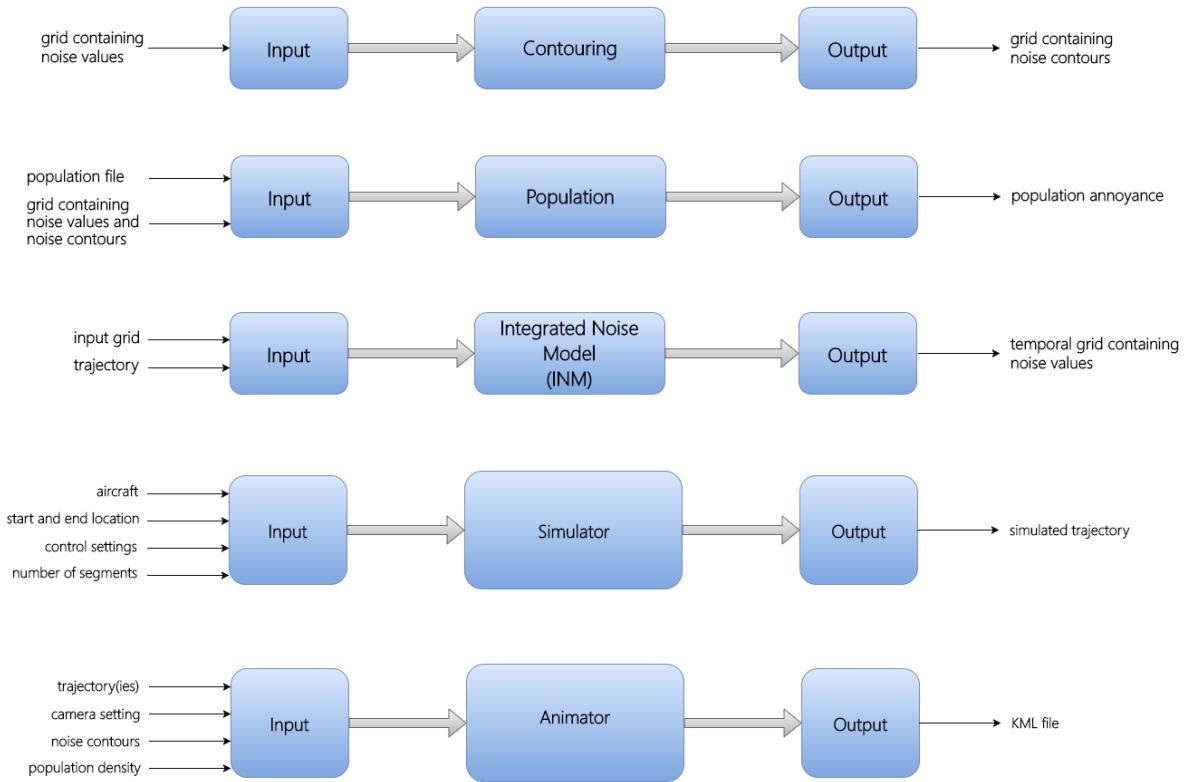


Figure 4.5: Block diagrams of the modules.

AeroVision consists of a number of modules that are called from within the two pipelines:

1. The INM submodule is responsible for calculating the noise values along a given trajectory. In order to calculate aircraft noise the submodule makes use of the Integrated Noise Model, which is the international standard computer model for assessing aircraft noise impacts in the vicinity of airports and over National Parks. The output of the INM submodule is a temporal grid, containing a grid with perceived noise values for each moment in flight time.
2. The Contouring submodule is responsible for the calculation of noise contours, which are collections of connected points with an equal noise value. The input of the Contouring submodule is a two-dimensional grid containing noise values. The submodule calculates all connected points at which the noise value is equal for every integer between the lowest and highest value. Since the input of the contouring module can be any two-dimensional grid, the user is not restricted to the use of INM. He can also provide values derived from other noise calculation tools.
3. The Animator submodule is responsible for generating animations by creating KML files. This submodule receives one or more trajectories, the noise along the trajectories and a number of visualization settings (among which a definition of the camera position).

4. The Simulator submodule is responsible for simulating a trajectory based on a predefined start and end location and a set of control settings coming from the optimization algorithm.
5. The Population submodule is responsible for processing population data, for example the population density. It offers functionality to calculate the experienced annoyance given noise values in the area surrounding a trajectory.

Next to the two-fold functionality support, there are other benefits to a modular design. First, it contributes to code reusability. This way the optimization and visualization modules could be reused in future development projects. They could also function as standalone applications. Second, modular programming makes a software project more manageable and easy by breaking the project down into modules. These individual modules are easier to design, implement and test. Additionally, a modular design can also help with future extensions. It makes the program easier maintainable and expendable.

## 4.4. Pipelined Architecture

AeroVision offers two possible paths for the user, each following another pipeline of modules. These paths are shown in Figure 4.6.

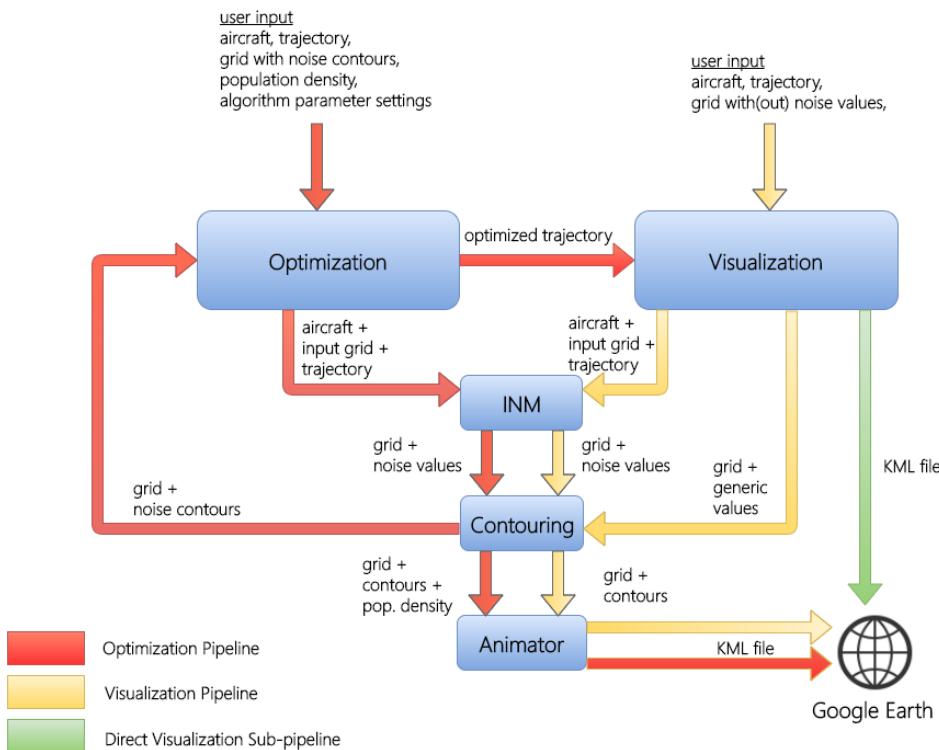


Figure 4.6: Block diagram of the three pipelines.

### 4.4.1. Optimization Pipeline

The optimization model evaluates many possible trajectories by using a genetic algorithm. Each trajectory is decoded by a number of genes on a chromosome. Chromosomes use mutation in order to generate new trajectories. In order to select which trajectories should be part of the next generation, the fitness of each trajectory is evaluated. This is done in a pipelined fashion. First the trajectory is passed to the INM submodule which calculates the noise generated along the trajectory. The output of the INM submodule is a temporal grid describing the noise perceived in a grid underneath the trajectory for each moment in time. Next, the Contouring submodule calculates the noise contours across the grids. The contours are used in combination with population data for predicting the number of people being awoken by the noise of the aircraft. The lower the number of awakenings, the more favorable the trajectory is. The characteristics of the updated trajectory are then entered into the Simulator module to compose the simulated trajectory. This process is repeated until the fitness converges or a maximum number of trajectories is reached.

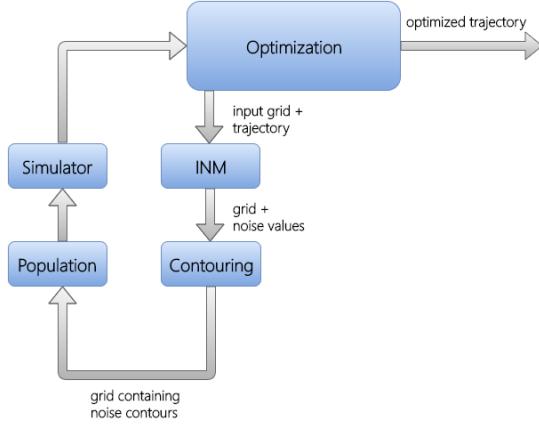


Figure 4.7: Block diagram of the optimization pipeline.

#### 4.4.2. Visualization Pipeline

The visualization pipeline starts off with defining a trajectory (or a number of trajectories) that should be visualized. For each trajectory the noise is calculated by passing the trajectory to the INM submodule. Next, for each resulting grid the noise contours are calculated by the Contouring submodule. The resulting contours are passed to the Animator submodule together with the trajectory(ies). The animator creates a KML file describing the actual animation while taking the user preferences into account. Finally, the KML file is loaded into the Google Earth plugin and shown to the user.

If the user has previously created a visualization, the resulting KML file can be reloaded into the program to start the visualization directly without the need of recomputing all positions and noise values. The option for direct visualization is seen as part of the visualization pipeline.

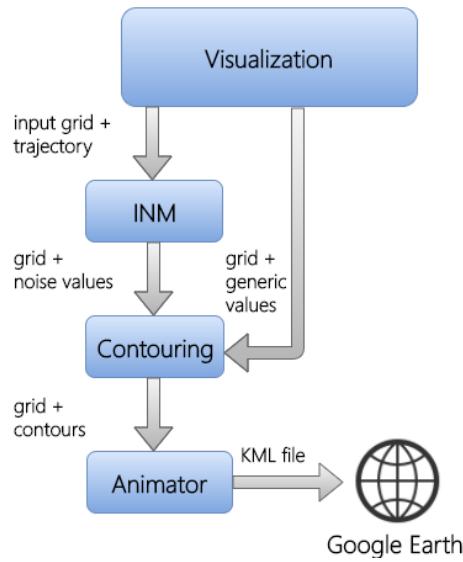


Figure 4.8: Block diagram of the visualization pipeline.

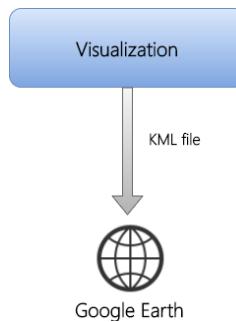


Figure 4.9: Block diagram of the direct visualization subpipeline.

## 4.5. User-System Interaction

### 4.5.1. The .NET Framework

After we decided to choose C# as the main language for our code, it was an obvious choice to develop our GUI with the .NET framework. The .NET Framework has become the mainstream environment for new Windows applications. It offers advanced graphical user interface components in case we decide to use innovative UI objects. Besides, the .NET framework contains a lot of predefined libraries for graph rendering and the creation of PNG's and color maps (which we are actually going to need for the visualization model). This helps to eliminate the amount of unnecessary codes and involves less coding for the developers. The framework also supports System.XML, which lets applications access data in XML files, including using XSLT and XPath. This comes in handy for us since we are going to use KML objects that are based on XML-defined data for the visualization model of our program.

**Windows Forms Application** But we still had to decide between using Windows Forms or WPF. The single most important difference between WinForms and WPF is the fact that while WinForms is simply a layer on top of the standard Windows controls (e.g. a TextBox), whereas WPF is built from scratch and doesn't rely on just the standard Windows controls. This might seem like a subtle difference, but it really isn't, which you will definitely notice if you are either creating a very complex or a relatively easy GUI. Since WinForms restricts users to a collection standard windows controls, a GUI that requires control elements that deviate from this ask for a significant amount of customization. By default WinForms for example does not support adding a background image to a ListView component. A programmer who really wants to seek the limits of what is graphically feasible will stumble upon a vast number of restrictions in WinForms. With WPF on the other hand control elements can be created by the designer in a container like fashion. In WPF for example a button with background image is created by adding a panel containing the image to the button. This is a great strength of WPF, however the current WPF form designer makes designing forms in WPF more tedious since you have to do significantly more work yourself. To give an example, the alignment of control elements requires more effort and is less obvious in WPF compared to WinForms. Since our GUI will mainly focus on requesting parameter settings, no complex control elements are necessary. Also, the visualization will be handled in the Google Earth plugin. So the user interface does not need to support heavy graphics. Therefore, for our Windows application WinForms will suffice and the currently superior design environment will save us precious time.

### 4.5.2. Model-View-Presenter Interface

For the user interface the architectural pattern Model-view-presenter (MVP) was used. In MVP all presentation logic is pushed to the presenter and the presenter assumes the functionality of the "middle-man". This gives a few benefits that are useful for us.

First, MVP improves the separation of concerns in the presentation logic. Business logic implemented in the Presenter can be retargeted to any number of UIs. Once the business logic is implemented and tested, the original View can be converted from a Win Form to Web Form or a Mobile Web Form, etc without impacting the Presenter as long as the new Views continue to implement the same interface. This also supports the Interface Segregation principle which makes the user independent of methods it does not use.

Second, MVP simplifies automated unit testing. The Presenter can be tested without having to worry about the database or the UI. Both the Model and View can be replaced with "Mock" objects designed to

simplify testing.

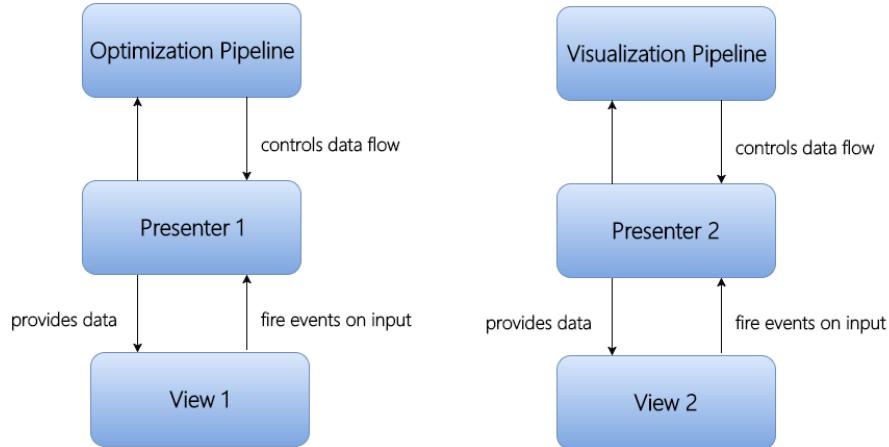


Figure 4.10: The application of MVP in AeroVision.

In our program the pattern was implemented as represented in Figure 4.10.

- **Model** In our case the model exists of the optimization and visualization models which are implemented through pipelines of modules. They define the data to be displayed.
- **Presenter** Our program contains two presenters: one for the Optimization pipeline and one for the Visualization pipeline. Both Presenters act upon the View and their corresponding module.
- **View** The View is a passive interface that displays data and routes user commands and events to the presenters to act upon that data. The optimization and visualization models have their own views.

Both presenters get and set information from/to the view through an interface that can be accessed by the interface (view) component. They handle the user events from the View and uses the output from one subsystem as input for the next subsystem. The .NET environment offers full support of the MVP pattern and the option to support the same model and presenter class with multiple interfaces.

#### 4.5.3. MDI Forms

For the interface of AeroVision a multiple document interface (MDI) is used. MDI is a graphical user interface in which multiple windows can be maintained within a single parent window. Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. This contrasts with single document interfaces (SDI) where all windows are independent of each other. MDI is a standard component within the Toolbox of WinForms.

The reason why we decided for MDI is because we wanted to separate the windows for specifying user settings from the windows showing the results of optimization and visualization. MDI allows this by providing a single menu bar for the three different pipelines AeroVision offers. The menu bar is shared between all child windows (tabs), reducing disorder and increasing efficient use of screen space. MDI allows us to display multiple documents at the same time, with each document displayed in its own window. Documents or child windows are contained in a parent window, which provides a workspace for all the child windows in the application. This way child windows can be hidden, shown and maximized as a whole independently of the main frame.

In case we would like to extend AeroVision to other platforms, MDI provides us with consistent application behaviour between platforms. It also supports modularity: windows can be upgraded independently of the application. This increases the maintainability of the program.

## 4.6. Noise Calculation

AeroVision offers the ability to calculate ground noise values along a trajectory by executing the Integrated Noise Model (INM). INM is the international standard computer model for assessing aircraft noise impacts in the vicinity of airports and over National Parks [24]. INM performs the calculations based on two user input files, a file defining a grid and a file defining the trajectory. The trajectory file contains the exact location, speed and thrust level for a number of positions along the trajectory. The grid file contains the boundaries of the grid and the size of a single grid cell in which the noise will be calculated. Besides the user inputs, INM uses a database of recorded noise values for a large collection of engine types. The database contains paired distance and thrust values for each engine and operational configuration (landing, takeoff, overflight). For each combination of distance and thrust, the perceived noise level is stored. By using bi linear interpolation, the noise value at any distance and delivered thrust value can be computed. For every time step the INM algorithm interpolates the current aircraft thrust and position. Given the aircraft thrust and distance, the noise is interpolated using the INM database. For further explanation on the methodology of INM we refer you to its technical manual [8].

## 4.7. Contouring

The Contouring Module implements an algorithm for taking gridded data and turning it into contour lines. The custom way to display noise and other environmental data on a map is by using contours. The algorithm basically plots the data onto a grid, searches for adjacent points with the same value and connects them into a collection of points [2]. This collection of points forms the so called contour.

### 4.7.1. The Contouring Algorithm

Below the general contouring algorithm will be explained step-by-step [10]. The input of the contouring algorithm is a grid containing values defined by the user. The grid can contain noise values calculated by the INM submodule or input values that are provided by the user. This should be delivered in a certain pre-defined format, representing the values on the specified grid for each time step (as mentioned in Section 4.2). Each grid can then be passed on to the contouring algorithm in order to calculate the contours. Instead of grid cells, the algorithm uses values at the intersection points between grid cells. Therefore, the algorithm places the value of every grid cell in its upper left corner. This is based on the assumption that the value corresponds to a specific location and that values in between can be calculated by linear interpolation.

#### 1. Step 1: Find switch points

Switch points are adjacent points (grid cell corners) in between which an integer value is crossed. The first step of the algorithm loops over all the points between grid cells and determines whether an integer value is passed vertically or horizontally. If an integer is crossed over, interpolate between both grid cell corners to find the exact coordinates at which an integer value is reached. Between adjacent points multiple integers may be crossed. This results in multiple switch points.

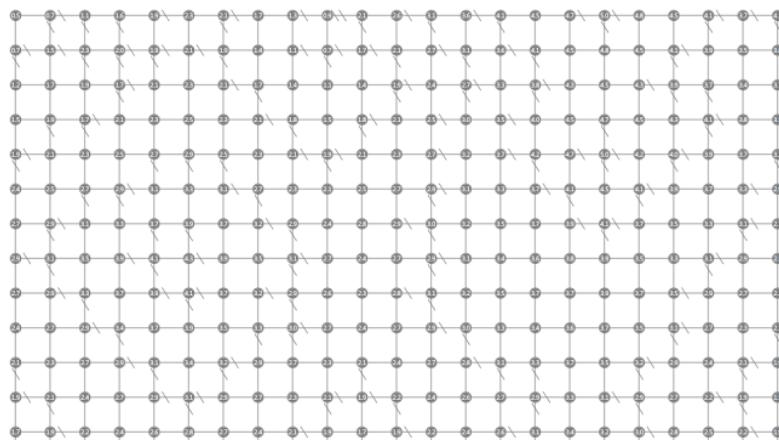


Figure 4.11: Find adjacent points in between which an integer value is crossed [2].

## 2. Step 2: Identify the direction of each switch point

In order to accumulate switch points together into contours, the direction of each switch point needs to be determined. All contours will be constructed clockwise, so if the higher point is to your east, then the switch point's direction will become north. If it is north, then the direction will be west, and so on.

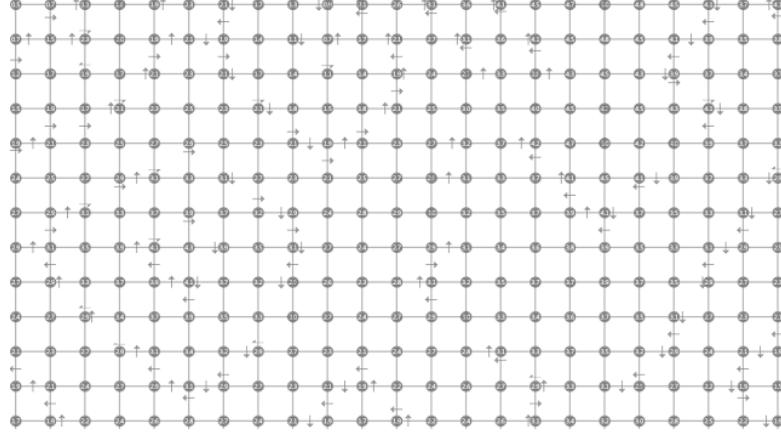


Figure 4.12: Determine the clock wise direction of each switch point[2].

## 3. Step 3: Construct the contours

In the final step of the algorithm it is assumed that for every switch point pointing into a grid cell at least one other switch point points outwards and has the same value [2]. Additionally, every contour starting at a border is open while all other contours are closed. With these assumptions we only need to traverse the switch points at the border that point into the grid. An open contour is created by following these switch points. Stop following when you either reach the border of the grid or when you encounter a switch point that is already selected for the current contour. Next, loop through the inner points of the grid and identify switch points which do not take part in any contour. These points should be followed to create a closed contour.

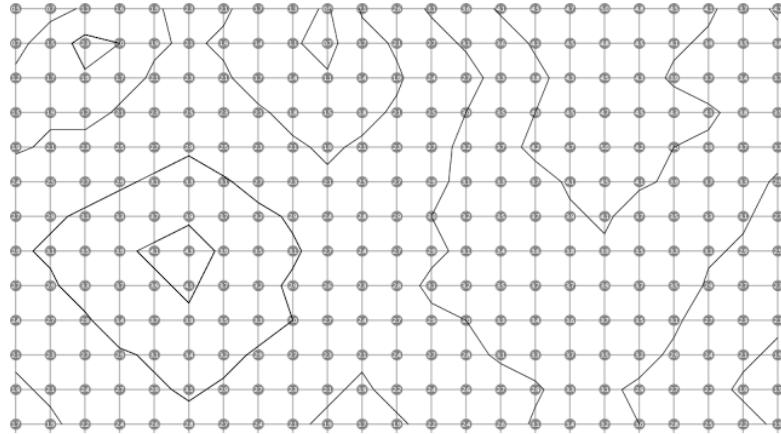


Figure 4.13: Follow the incoming switch points located on the borders for the construction of open contours. Follow the inner points for the construction of closed contours [2].

### 4.7.2. Implementation

The input to our implementation is a two-dimensional array of double values, representing the grid with noise values. To store the points on the grid where a contour intersects with a grid line (step I), we used the class `ContourPoint`. A `ContourPoint` holds a field for its value and two types of coordinates indicating its exact location and index on the grid line. For step I and II we use a method called `CreateContours` that creates two grids of lists of `ContourPoints`: `hgrid` and `vgrid`. They hold the horizontal and vertical `ContourPoints` respectively. For Step III `CreateContours` passes on the data constructed in the two grids to a method called

GenerateContours which iterates over the grids. For each unprocessed ContourPoint it creates a new contour. The Contour class tracks all the ContourPoints found on the resulting path. The last step is handled by the FindNext method in the ContourPoint class, which finds the next ContourPoint as described in Step III.

## 4.8. Optimization

The optimization model contains all logic regarding the optimization of aircraft trajectories. It offers the user the ability to find an optimal trajectory between a predefined start and end point. The optimization can be executed using different modes, depending on which parameter the user wants to optimize. AeroVision supports two optimization modes, which are: fuel optimization and annoyance optimization. In the first optimization mode, the software minimizes the total amount of fuel consumed by the aircraft's engines from the moment of takeoff until it reaches the predefined end point. In the second optimization mode, the software aims to minimize the amount of annoyance experienced by the population due to aircraft noise. This includes the population living in the vicinity of possible approach or departure trajectories. In this optimization mode, the software calculates noise contours and assigns each house to the closest contour. The contour area is used to determine which people and to which extent they are annoyed by the noise. This is the innovative part of our algorithm. Previous trajectory optimization algorithms, including NoiseLAss, only use the computed number of people that are annoyed or awoken by the noise (the number of awakenings) [10]. This way only the volume of the noise is optimized. By also taking the contour area into account, both the volume and spreading of noise are optimized. The optimal solution that results from the optimization consists of a number of aircraft control settings (i.e. for any moment in time the optimal throttle setting and flight path angle).

### 4.8.1. Genetic Algorithm

Finding the optimum solution in minimizing annoyance is not trivial as it has a very complex fitness landscape. The best option might seem to let the aircraft climb as fast as possible in order to minimize annoyance, however the option to fly low over a city using very little power (and thereby producing little noise) could even be better. As a third option, using a more complex ground path and avoiding densely populated areas altogether should also be considered. Since population density distributions can have any shape, there is a high chance of the optimization algorithm reaching a local optimum [38]. A traditional hill climbing algorithm starts with a random initialization and tries to find the best solution by updating parameters in the direction that yields an increase of the fitness. Since in our problem domain the chances of getting stuck in local optima are very high, hill climbing algorithms would not suffice. Using a genetic algorithm instead is a much better choice. In a genetic algorithm solutions are encoded in the form of chromosomes consisting of elementary building blocks, i.e. genes [14]. Each gene consists of a single value, which represents a part of the final solution and could be interchanged by other genes to result in a different solution. The algorithm starts with a population of random solutions (or chromosomes). For each of those possible solutions the quality (or fitness) is calculated. The best chromosomes are used for creating the next generation of solutions, but random mutations are applied to them in advance. In those random mutations, chromosomes are able to randomly interchange genes in any point of the chromosome. This effectively spreads the optimization throughout the whole search space and therefore makes genetic algorithms less prone to reaching a local optimum.

### 4.8.2. Genetic Algorithm description

To optimize aircraft trajectories using a genetic algorithm, AeroVision uses the GeneticSharp library, which is a fast, extensible, multi-platform and multithreading C# Genetic Algorithm library that simplifies the development of applications using Genetic Algorithms. The algorithm needs a number of settings: the method of selecting the best chromosomes, the mutation method, the stop criteria, the size of a population, the chromosome definition and a fitness function. The library supports a number of selection methods, of which the most basic option (elite selection) is recommended. The mutation method used by AeroVision is crossover, which is a commonly used setting. The stop criteria can be a fixed number of generations, which can be specified by the user. The default setting is 100 generations. Additionally, the user is free to define the number of chromosomes that are part of one population. The default population size is 70 chromosomes, each representing a possible trajectory in our case.

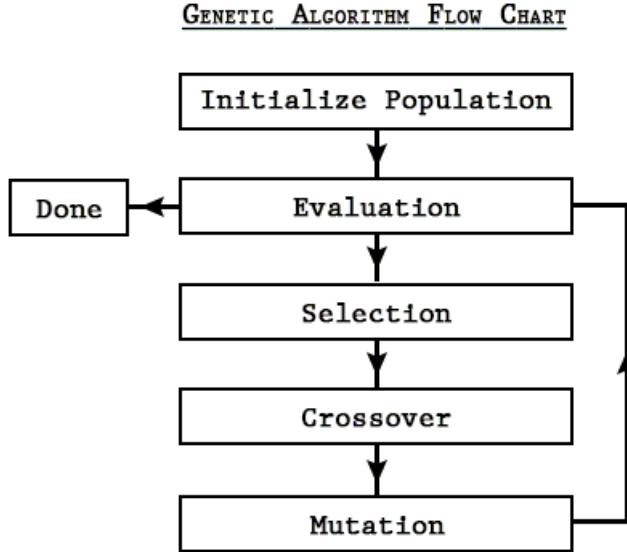


Figure 4.14: The general process of a genetic algorithm.

Parameter	Optimal Value
Mutation method	crossover
Number of generations	100
Population size (per generation)	70

Table 4.1: The optimal settings for AeroVision's genetic algorithm.

**Chromosome description** As explained earlier, a chromosome contains all information about one possible solution of the problem. So, in the case of trajectory optimization a chromosome should contain all data that is needed to describe a full possible trajectory. The trajectory is split up in a user-specified number of segments. In order to allow different ground paths, we decided to alternate between straight and curved segments. For each of those segments the solution needs to contain four control settings of the aircraft. For curved segments the chromosome should contain a value for the throttle setting, flight path angle, bank angle and a value indicating the amount of heading change. For a straight segment the chromosome contains the throttle setting, flight path angle, bank angle (which equals zero) and the length of the segment. We decided to encode the control settings by using values between 0 and 1. Each chromosome sequentially describes the control settings for a number of alternating straight and curved segments, always starting with a straight segment. For each segment the chromosome contains three or four genes representing values between 0 and 1 that describe the corresponding control settings. Those values will be used as factors between the minimum and maximum allowed settings at any given moment during the flight. This way we enforce that all randomly generated solutions are feasible according to the laws of physics.

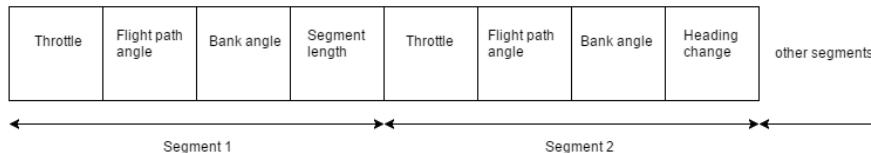
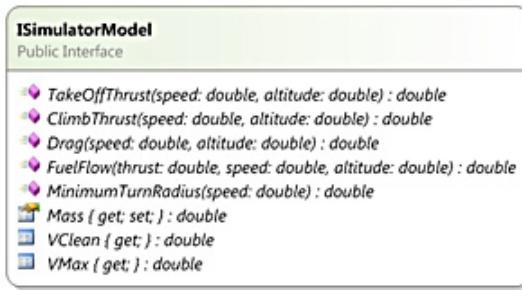


Figure 4.15: Set-up chromosome structure.

**Fitness function** To evaluate the quality of a chromosome we need to implement a fitness function for both optimization modes. To offer a combined optimization of the fuel consumption and experienced annoyance, we need to simulate the flight based on the control settings that are encoded in the chromosome. In the simulation we keep track of the fuel consumption which is used as an objective for minimizing the total fuel consumption. To minimize the population annoyance, we keep track of the amount of thrust delivered by the engines at any given moment during the flight. The delivered thrust together with the aircraft's position enable us to calculate the noise level for any position. After calculating noise contours throughout the grid, houses will be assigned to the noise contour that is closest to them. By doing this the amount of people and the extent to which they are annoyed can be estimated. The next subsection explains in detail how the flight simulation works.

#### 4.8.3. Flight Simulation

In order to evaluate the fitness of a chromosome, we need to simulate a flight with the control settings as encoded in the chromosome. Since fuel consumption and the delivered thrust depend on the type of engine and the number of engines the aircraft has, the flight simulation requires an object with the characteristics of the aircraft that is used for optimization. In order to take these aircraft characteristics into account, we defined an interface containing a number of methods and properties that are required by the simulation. AeroVision comes with a detailed model of the Boeing 747-400, however additional aircrafts can easily be added by implementing this interface [32]. Besides the aircraft details, the simulator receives the exact start



and end location, and the control settings for each of the segments that the flight consists of [3]. The simulation works by maintaining a number of state parameters in memory and updating them every second until we reach the end point of the trajectory. The following state parameters are used:

$x$  : x coordinate [m]  
 $y$  : y coordinate [m]  
 $z$  : altitude [m]  
 $\chi$  : heading [ $^{\circ}$ ]  
 $\mu$  : flight path angle [ $^{\circ}$ ]

The position of the aircraft is updated using the following control parameters:

$\Gamma$  : throttle setting  
 $\gamma$  : flight path angle [ $^{\circ}$ ]  
 $\mu$  : bank angle [ $^{\circ}$ ]

**Typical departure** To explain the implementation of the simulation process, one must first get acquainted with a typical aircraft departure. During a takeoff the aircraft accelerates towards the end of the runway. Once the aircraft's speed reaches a speed called  $V_1$  the speed is too high to safely abort the takeoff attempt. The aircraft accelerates towards a speed called  $V_r$ , which is the speed at which the pilot will pitch the nose up to let the aircraft takeoff. Soon after the aircraft clears the ground, the takeoff safety speed  $V_2$  is reached before it reaches 35ft altitude. This is the minimum speed that needs to be maintained for a safe climb in case of a single engine failure.

**Takeoff thrust** In order to respond to possible failures it is important to reach a safe altitude as quickly as possible. Therefore, soon after takeoff, the pilot will use all the thrust delivered by the engines to gain altitude instead of using it for accelerating to a higher speed. The thrust setting is also fixed to a setting called takeoff thrust. The pilot continues using takeoff thrust to climb until the aircraft reaches an altitude of 1500 ft.

**Climb thrust** Once it reaches this altitude the thrust is decreased from takeoff thrust to climb thrust. This setting prolongs the lifetime of the engines and at this point the aircraft has already reached an altitude that offers some diversion possibilities in case of an emergency. On climb thrust the aircraft ascends to an altitude of 3600ft or until it reaches  $V_{clean}$ , the speed at which the landing gear and flaps are fully retracted [32]. The control settings in those two segments are fixed. The throttle is set to a value that delivers takeoff thrust or climb thrust in the first and second vertical segment respectively. The flight path angle for both segments is maximal, which means that the aircraft ascends with a constant velocity as all engine thrust is being used. Climbing with a steeper angle is not possible, since the aircraft will then slow down and eventually stall.

**Segments with free parameters** From  $V_{clean}$  or 3600ft the aircraft enters a state in which the pilot is able to decide what climb angle and throttle setting to use [11]. Also, the pilot is now free to choose different ground paths. Therefore, this is the moment the simulation starts using the actual control settings as encoded in the chromosome. Once the aircraft reaches the free movement state, the trajectory always starts with a straight segment. This segment is followed by an amount of turn or by straight segment pairs. To ensure that the aircraft will always arrive at the prescribed end point, the aircraft is forced to continue turning towards the end point during its last turn. The final straight segment simply connects the end point of the last turn with the end point of the trajectory. This ensures that no matter what ground path is chosen, the aircraft will always end up at the predefined trajectory end point.

**Updating state parameters** All state parameters are updated for each second of the optimization process. The flight path angle  $\gamma$  is calculated by interpolating between 0 and the maximum flight path angle  $\gamma_{max}$  the aircraft can achieve without slowing down with the current amount of thrust and drag. Since the minimum flight path angle equals 0, the actual flight path angle based on the gene setting  $s$  is computed by:

$$\gamma_{new} = s \gamma_{max}$$

Given a current delivered total thrust  $T$ , current total aircraft mass  $M$  and currently exerting drag force  $D$ , the  $\gamma_{max}$  for this moment in the simulation is computed by:

$$\gamma_{max} = \arcsin \frac{T - D}{M g_0}$$

Note: from takeoff until 3600ft or  $V_{clean}$ , no gene setting  $s$  exists since the actual flight path angle equals the maximum flight path angle.

After updating the flight path angle, the speed will be updated, by computing:

$$v_{new} = v_{current} + \left( \frac{1}{M} * (T - D - (M g_0 \sin \gamma)) \right)$$

Once the aircraft passed the 3600ft or  $V_{clean}$  mark, the bank angle given gene setting  $s$  and the maximum allowed bank angle  $\mu_{max}$  is computed by applying:

$$\mu_{new} = 2 \mu_{max} s - \mu_{max}$$

The other position state variables are updated by computing:

$$\begin{aligned} \chi_{new} &= \frac{g_0}{v} * \tan \mu \\ x_{new} &= x + v * \cos \gamma * \sin \chi \\ y_{new} &= y + v * \cos \gamma * \cos \chi \\ z_{new} &= z + v * \sin \gamma \end{aligned}$$

**Switching between segments** Besides the position and altitude state parameters, the horizontal state should also be updated. This means that every second, the simulator needs to check whether the aircraft is still in the same segment. Any time the aircraft switches between two segments, the current state is stored as the segment start state. For each straight segment, the chromosome contains a gene  $s$  indicating the segment length. The actual used segment length  $l$  equals  $l = s * l_{max}$ , in which  $l_{max}$  is set to 8 kilometers by default. Every second the simulator checks whether the end of a straight segment is reached by comparing the distance between the current position and the segment start position with the actual segment length  $l$ . Once the distance exceeds  $l$ , the aircraft switches from a straight segment to a turn. In a turn the bank angle will be updated by the formula for  $\mu$  as presented above. Instead of the segment length, the fourth gene now represents a factor describing a heading change  $\Delta\chi$  between  $-90^\circ$  and  $90^\circ$ . At the start of the segment the current heading is stored. During every second of the simulation, the angle between the segment start heading and the current heading is compared to the target heading change  $\Delta\chi$ . If the target heading change is reached, the aircraft switches back to a straight segment. This process repeats until the predefined number of segments is reached.

## 4.9. Visualization

The visualization pipeline is responsible for the visualization of trajectories, noise levels produced along the track and the annoyance experienced by the population living in the vicinity of the trajectory. For visualization AeroVision contains a Google Earth integration using the Google Earth Plugin and Google Earth Server DLLs. This enables AeroVision to show 3D visualizations on the high resolution maps of Google Earth. The visualizations produced by AeroVision are based on KML files generated by the Animator submodule. This submodule receives one or more trajectories, the noise contours along the trajectories and a number of visualization settings (among which a definition of the camera position). Animator composes each KML file out of three components, namely the set-up, animation step(s) and finish. The content of each component can be described as follows:

- **KML Set-up** The set-up consists of all pre-defined animation definitions that are required for the to be visualized object. This includes the camera settings and the origin of the map on which the animation is depicted.
- **Animation Step(s)** The animation steps contain all updates for the visualized object at the given moment in time (time step)
- **KML Finish** The finish consists all post-defined animation definitions that are required for the to be visualized object.

Each object in the animation, such as the aircraft and flight path, has its own Animator that produces the three KML components specifically for that object. The KML files are loaded via the Google Earth Server and displayed using the Google Earth Plugin. AeroVision offers 2 visualization modes: 2D and 3D.

**2D visualization** In the 2D visualization mode, AeroVision visualizes trajectories on the maps of Google Earth using a static camera position directly above a user defined point. The visualization either contains one trajectory along which AeroVision visualizes the noise at any moment during the flight, or contains the visualization of noise integrated over multiple trajectories. Prior to loading the visualization, the user is able to define a custom map that should be loaded underneath the trajectory. This is useful when the user wants to create images in which trajectories are more in contrast with respect to the background map.

**3D visualization** In the 3D visualization mode, AeroVision visualizes a single trajectory on the maps of Google Earth using one of a number of supported camera positions. The supported camera positions are:

1. **Flyby**, in which the camera is positioned at a fixed point along the trajectory, pointing towards the aircraft as it passes by.
2. **Follow**, in which the camera follows the aircraft at a fixed distance and from a fixed angle with respect to the aircraft.
3. **Manual**, in which the user can manually move the camera to any position and point it in any direction.



# 5

# Quality Assurance

AeroVision has been designed to be modular and compatible with other noise calculation tools now and in the future. As is the case for any project of this scale, code quality is important to guarantee maintainability and readability. As there is a possibility that AeroVision will be further developed by us or other engineers, measures have been taken to increase code quality. To guarantee this code quality, AeroVision's source code is thoroughly tested with the help of unit testing and continuous integration through AppVeyor. These measures are described in Section 5.1. The code quality was also regularly checked by Visual Studio's Code Metrics Tool and SourceMonitor. More about software measurements in Visual Studio and other code analytics performed that helped to improve AeroVision source code can be found in Section 5.2.

## 5.1. Unit Testing and Continuous Integration

As a project grows, new features are added, leading not only to an increase in functionality, but also a decrease in readability and maintainability. A countermeasure that can be taken to help maintainability is the development of a proper test suite containing automated tests. When new features are added, these tests can show whether any previous functionality is broken by the code modifications. Section 5.1.1 describes the policy the team applied during the project. Sections 5.1.2. and 5.1.3 detail what testing framework has been used and how continuous integration was set-up to automatically run the tests.

### 5.1.1. Testing Policy

The research report describes Test-Driven Development (TDD) and how it was applied to this project. In TDD the policy is to first write a suite of test-cases with stub implementations, only so let the test cases fail. Then, after the full test suite has been finished, the actual implementation is allowed to start. For this project we applied a similar strategy, with one person writing test code for a certain feature and the other person independently implementing it. This policy worked very well for us since it also allowed us to

Testing in general was included in the process to a point where all modules have an extensive test-suite. Especially the optimization module is thoroughly tested, since it forms the more 'unpredictable' part of our project. For validation we used the NoiseLAss tool to confirm that our program actually found the optimal trajectory. If new features were to be added to this module, the tests could clearly show if any of the old functionality had been broken. For the computation of the contours we received a validation set of the client. By comparing our results to this set we were able to confirm if the code worked correctly.

The only part of our project that is currently lacking a thorough test-suite is the visualization module. Simply running the code and entering the generated KML file in Google Earth will already expose any bugs relatively quickly. The amount of time that would have gone into testing the KML file line by line and setting up a proper testing environment in combination with Google Earth was better spent testing the mathematical modules. The mathematical modules, such as contouring and optimization, form the basis of both our application and visualization.

Whereas many different unit tests have been written, we have also written automated system integration tests. The integration tests that currently exist test the communication between different components. Although a very time-consuming operation, this could have been expanded upon to feature a test that would

test everything from creating an application to launching a visualization in Google Earth. Instead we tested this manually before merging new additions with the master branch.

### 5.1.2. Testing Framework

To thoroughly test the AeroVision source code the testing framework NUnit has been used. In the .NET world the two most popular unit testing frameworks are the open source NUnit and the commercial MS Test [25]. Out of the two options we decided to go with NUnit, since it is most similar to JUnit which we both are the most familiar with. Additionally, NUnit integrates seamlessly with Visual Studio, the editor we used [31]. In contrast to MS Tests, NUnit also fully supports test class inheritance and parametrized tests. MsTest has slightly more features for test logging, but we still chose for NUnit due to its native support in Visual Studio and our previous positive experience with JUnit.

### 5.1.3. Continuous Integration: AppVeyor

The creation of test code alone is not enough to have simple checks whether or not new functionality breaks older code. For this purpose a continuous integration server deploying AppVeyor has been set-up [4]. The git repository used for AeroVision is configured to notify AppVeyor every time new code enters the repository, triggering a build in AppVeyor. This ensures that every time new code enters the repository, *regression testing* in the form of the unit tests is automatically run and results can be seen on the server [18]. Installation of the AppVeyor server in itself is simple, as this is all well documented on their own website.

In the orientation phase of our project we also considered popular alternatives like Travis and Jenkins, which also support continuous integration for C#.

## 5.2. Code Analysis

Whereas an extensive test suite can help to quickly see if new code breaks old features, static code analysis can help to improve maintainability and more easily expand existing code. Not only will it help to give the programmer a clear overview of the current state of the code, it can also help to find issues in the code. For this purpose a CoverAlls installation has been used to track our code coverage, which we describe in Section 5.2.1. In addition we used a built-in software analysis tool in Visual Studio and SourceMonitor to measure the code metrics, as described in Section 5.2.3 and Section 5.2.4 respectively. In addition every Bachelor thesis group has been permitted to submit their code for analysis by the Software Improvement Group (SIG), of which we describe the comments and feedback in Section 5.2.3.

### 5.2.1. CoverAlls

To keep track of our test coverage we deployed a tool called CoverAlls [12]. It works with our CI server and with every new build it recomputes your code's coverage statistics. It does not only show the total percentages and lines covered, but it also offers individual file line-by-line coverage reports. For C# there are a few other free alternatives for code coverage, such as NCover and OpenCover, but these tools do not support continuous testing. Therefore, our choice for CoverAlls was quickly made.

All software developed for this project has been submitted to regular coverage analysis, the results of which are represented in Figure 5.1. Because the amount of production code written during the first two weeks was low, these weeks were merged into one sprint in the graph. During this phase of the project we set-up the work and test environment and did not spend much time testing, which explains the low achieved coverage of 33%. Furthermore, the line graph shows an overall increase in test coverage throughout the entire project. In the final version of our product we achieved a line coverage of 93%. The full 100% was not achievable because of the visualization module. The generated KML files were tested manually in Google Earth, otherwise this would have been very time consuming.

### 5.2.2. Software Improvement Group

The Software Improvement Group (SIG) is a company that translates detailed technical findings concerning software systems into actionable advice for upper management [28]. SIG performs static analysis on the AeroVision source code two times. The first time to provide feedback to the development team on what can be improved. The second time to confirm that this feedback is taken in consideration and to evaluate the final quality of the code. Feedback from SIG can be found in Appendix B. SIG awards points on a ranking in which five stars is the maximum. The points awarded are based on items such as the presence of testcode, the amount of code duplication, module coupling and other metrics indicating good readability, maintainability

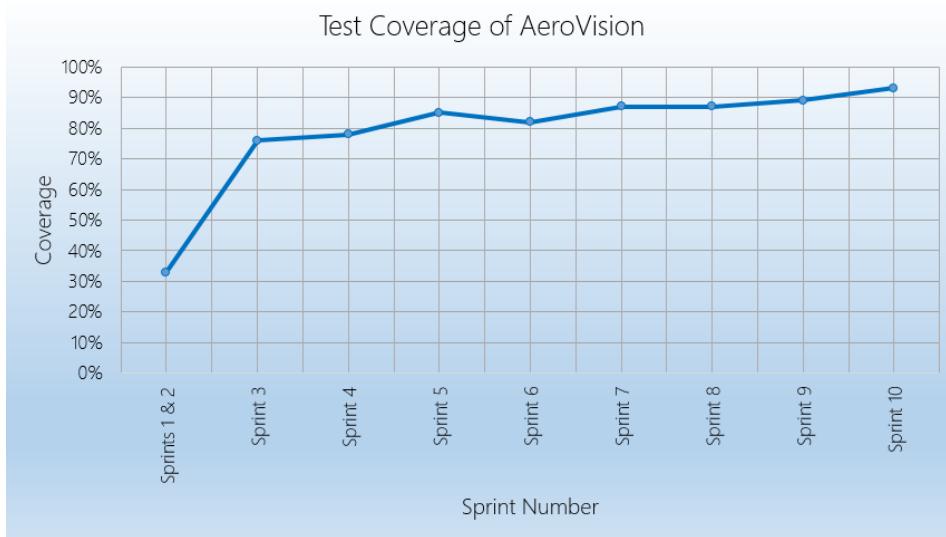


Figure 5.1: A line diagram showing the achieved test coverage in AeroVision during all sprints.

and testability. In the initial feedback, the implementation scored 3.7 out of five starts, indicating the code was already up to high standards. The only issue reported by SIG was related to the unit size and unit complexity, indicating that some methods were too long and contained multiple functionalities. This issue has been resolved by splitting longer methods into multiple shorter methods. If a method had various responsibilities, we split it up into methods with distinct tasks. As soon as we receive the second round of feedback from SIG, this will be processed in the final report.

### 5.2.3. Code Metrics in Visual Studio

To keep track of our software quality we used Visual Studio's Code Metrics Tool. This is a built-in command line utility that calculates code metrics for your managed code [26]. The code metrics calculated are:

- **Maintainability Index (MI)** An index value between 0 and 100 that represents the relative ease of maintaining the code. A high value means better maintainability.
- **Cyclomatic Complexity (CC)** The structural complexity of the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable.
- **Depth of Inheritance (DoI)** The number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy the more difficult it might be to understand where particular methods and fields are defined or/and redefined.
- **Class Coupling (CO)** The coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Good software design dictates that types and methods should have high cohesion and low coupling.
- **Lines of Code (LoC)** The approximate number of lines in the source code (excluding test code).

During our project we did two measurements with the Code Metrics Tool: around the first and second SIG deadline. The results of these measurements are summarised in Table 5.1. Note that the lines of code mentioned in the table do not include the test code. From the table it becomes apparent that in the second half of the project, we improved the maintainability of the project while the the number of lines of code increased with 45%. However, the other metrics (Cyclomatic Complexity, Depth of Inheritance and Code Coupling) of our code slightly increased. But this is understandable considering the 45% growth in production code.

SIG Deadline	MI	CC	DoI	CO	LoC
1	79	298	1	80	785
2	80	450	2	97	1,141

Table 5.1: Code metrics calculated by Visual Studio's Code Metrics Tool

SIG Deadline	M/C	S/M	MaxC	MaxD	Docs	LoT	LoC
1	5.07	5.41	22	6	24.9	1,158	785
2	6.29	4.17	12	5	25.5	1,324	1,141

Table 5.2: Code metrics calculated by SourceMonitor

#### 5.2.4. SourceMonitor

For code metric measurement we also used a tool called SourceMonitor. This tool provides different metrics than Visual Studio's Code Metrics Tool that are worth exploring for the quantification and qualification of AeroVision's source code. SourceMonitor was specifically used to measure the following code metrics [1]:

- **Methods per Class (M/C)** A useful indication of how big each class is. A number between 5 and 8 methods per class is recommended by SourceMonitor.
- **Statements per Method (S/M)** A useful indication of how big each method is. A number below 20 statements per method is recommended by SourceMonitor.
- **Max Complexity (MaxC)** The maximal cyclomatic complexity that is measured. A maximum between 10 and 20 is recommended by SourceMonitor.
- **Max Depth (MaxD)** The maximal depth of inheritance that is measured. A maximum between 5 and 10 is recommended by SourceMonitor.
- **Percentage of Documentation (Docs)** The percentage of source code that exists of documentation and comments.
- **Lines of Test Code (LoT)** The approximate number of lines in the test code.

With SourceMonitor there were also two rounds of measurements: once around the first SIG deadline and once around the second deadline. The results of these measurements are summarised in Table 5.2.

In the second half of the project the unit complexity has clearly been improved, based on the decrease in maximum complexity, maximum depth and statements per method. However, the number of methods per class has increased, which is caused by the refactoring that was done in response to SIG's first feedback round. To decrease the unit size we split long methods into shorter methods. Additionally, in the second part of the project we added the FlightSimulation class as part of the trajectory optimization pipeline. At the moment this is AeroVision's largest class containing 16 methods of  $\pm$  15 lines each. We were not able to split this class further since the methods it currently contains are necessary for the simulation of an entire trajectory. A FlightSimulator object reproduces each line segment of the simulated trajectory separately. If the reproduction of the segments would be handled by multiple classes, this would complicate the final set-up of the simulated trajectory and would also make the code harder to follow for us.

As can be seen, the number of comment lines has been relatively constant throughout the project. SIG recommends a documentation percentage between 20% and 40%, indicating that AeroVision's source code was well commented when committed to the server [27].

Another metric worth exploring is the ratio of test code compared to production code. When excluding commented lines, a rough estimate can be made by calculating the ratio production-code:test-code, which yields 1: 1.16. SIG recommends a ratio of at least 1:1.

# 6

## Research, Design, and Development Process

This section describes the process of AeroVision's development, which can be divided into three main categories. First, Section 6.1 describes some of the methods applied through the entirety of the project, such as the application of the Scrum methodology. Second, Section 6.2 describes the research processes utilized throughout this project, such as a feasibility study. Third, Section 6.3 describes the process of designing for the system. In addition, this section details the requirements that were changed during the course of this project and how these changes were handled by the team. Fourth, Section 6.4 outlines the different processes that concern development, such as the tools used to plan the individual sprints, during the implementation phase of this project and the tools used to ensure that both team members had a good overview of all the code that was written. Finally, Section 6.5 reflects on the processes used and lists new insights gained from using these processes.

### 6.1. Project Management Processes

Throughout this project the team has applied the Scrum methodology, in which teams work in intervals, known as sprints, with deliverables finished after every interval. This model of sprints contrasts with the more traditional waterfall method in which different stages of the product cycle can be clearly defined. A project using the waterfall methodology starts with a design phase in which the whole system is designed, followed by a production phase and ending with a validation phase. Scrum, as a form of agile development, promotes the use of short cycles consisting of these three phases instead. This is visually presented in Figure 6.1. When using Scrum validation plays a more important role in the production process, ensuring that the final product matches the customer's expectations and needs.

For agile development to handle the frequent changes that are associated with it, communication is a fundamental part of the process. Three different types of meeting were held on a frequent basis. First, both team members met on a daily basis to inform each other about their current tasks and discuss what other tasks needed to be picked up. Second, meetings with the client took place every two weeks to receive his feedback and requests. Program features that consisted of a major mathematical or aerospace engineering part were also explained by him to us in extensive meetings. Finally, meetings with the TU coach also took place every two weeks. During the orientation phase we met more frequently to discuss the set-up of our code.

The main timeline of this project can be summarised in a total of eight sprints, which are represented in Figure 6.2. Because the activities in the first two weeks of the project overlapped, these weeks were merged into one sprint. A roadmap and release plan can be found in the Product Planning that can be found in Appendix C. The sprint plans and reflections can be found in Appendix D. During the second week of the first sprint we started working on the design process. This sprint was also used for the set-up of code analysis tools like AppVeyor. Sprints 2 through 7 consisted of design and development iterations. Sprint 8 moved the focus to the evaluation of both code and design. This resulted in removal of multiple code issues and cleaner code in general. This sprint was also used to do the last acceptance tests with the client. Sprint 9 was the last sprint and focused on creating the final deliverables.

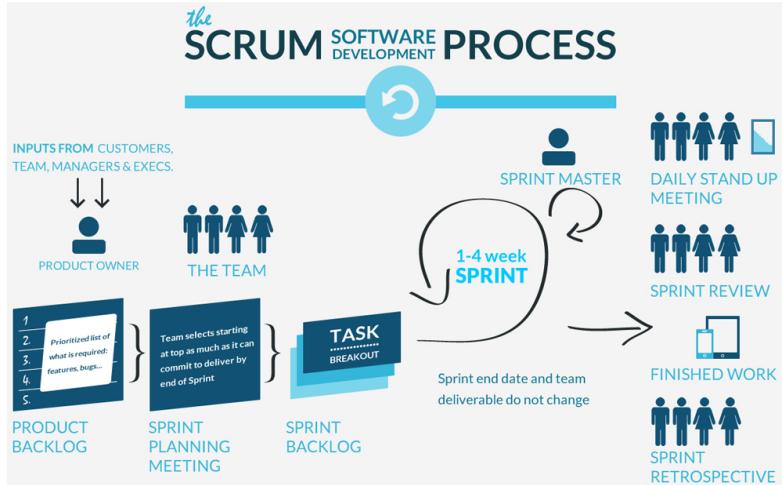


Figure 6.1: The Scrum methodology visually represented.

## 6.2. Research Processes

In preparation of this project an orientation phase of the current state of affairs was done on two levels. The first was a small study on the customer needs. This process, as well as the results, is detailed in Section 6.2.1. The second was a feasibility study in trajectory optimization, which is described in Section 6.2.2.

### 6.2.1. User Study

To establish a firm understanding how ATO currently manages the processes of trajectory optimization and visualization and what their current tools look like, we conducted a small user study through interviews with the client. Two extensive interviews of two hours each were conducted in person and featured both specialised and general questions related to broader concepts. Dr. Ir. Sander Hartjes is employed as an Assistant Professor in the field of airline operations at the chair of Air Transport and Operations [6]. His research mainly focuses on optimal aircraft performance regarding the reduction of noise and pollutant emissions. He has already been working at ATO since 2008 and therefore he is equipped to inform us on the way the department is would like to use our program and what features the department will benefit most from. The results of these interviews have formed the basis of the design as presented in Chapter 4, and are further detailed in the research report that can be found in Appendix A.

### 6.2.2. Feasibility Study

To determine the feasibility of basing trajectory optimization on contour areas and making it multi-core for potential speed-ups, we did a small study about the desired functionality. The study was focused towards goals of the project. We analyzed whether the desired product can be practically materialized in terms of implementation and contribution. For this purpose the client provided us the technical manuals of INM and NoiseLASs, and multiple theses on the concepts of contouring and trajectory optimization. By exploring these documents we were able to conclude that we had sufficient understanding of mathematics that is needed for the concepts. Besides the interviews described in Section 6.2.1, other meetings were also held with the client to discuss and clarify the academic part of the project. This was necessary since we did need some extra guidance with trajectory optimization.

### 6.2.3. Requirement Engineering

Since our feasibility study was positive towards undertaking the project, the next phase started with gathering requirements from the client. To know the ideas on what the final product should provide and which features the software should include, we used the requirement elicitation process divided in five steps [21].

- **Step 1: Requirement Gathering**

By interviewing the client and researching similar products we were able to understand the user needs and elicit the requirements.

- **Step 2: Requirements Organisation**

Together with the client we prioritized and arranged the requirements in order of importance. In the orientation phase of our project the client prioritized the visualization component of the program as a must have requirement. Trajectory optimization was seen as a "should have" requirement, since it was not clear to us if we were actually able to achieve a major speed-up in optimization. However, during the course of the project, when we started implementing the optimization module and achieved a speed-up, the priority of trajectory optimization was changed to a must have by the client.

- **Step 3: Software Requirement Specification**

In this step of the process we defined how the intended software will interact with hardware, interface, speed of operation, response time of the system, maintainability and compatibility of the software with other systems. The requirements were received from the client in natural language, next the requirements were documented in the form of user stories so that we could explore the different scenarios.

- **Step 4: Software Requirement Validation**

After the requirement specifications were developed, we validated the requirements in consultation with the project coach. We checked all requirements against following conditions:

- If they are complete
- If there are any ambiguities
- If they can be practically implemented

If requirements were ambiguous or conflicting, we negotiated and discussed this with the client. For example, during our first meetings, the client used the term 'real-time' for the computations of contours. Real-time technically means that the program reacts within 100 ms to a real event that is happening. However, our program will do computations on historic or simulated trajectories, not on real flights at that moment. When we asked the client what he exactly meant with the term real-time, he explained that he wanted the contours to be calculated within a second. Therefore, we adjusted this requirement to a performance constraint and required the contours to be calculated over time along a trajectory. As a result of this validation step, requirements were reasonably compromised and prioritized.

- **Step 5: Documentation**

The final step in this process was to document functional and non-functional requirements for next phase processing. A detailed overview of the initial requirements and user stories can be found in the research report in Appendix A.

## 6.3. Design Process

### 6.3.1. Changed Requirements

The enhanced functionality of the program was caused by changes in the requirements of the final product. This section describes why these changes occurred, how these changes were addressed and how they impacted the design of the system.

#### Added support for 2D visualization

AeroVision's initial requirements only included 3D visualization, allowing the user to animate the noise contours along a particular trajectory. During the project the client realized that 2D visualization would also come in handy, for example for an overview of the noise produced along an entire trajectory in one shot. This is a feature that will often be used to produce figures for papers and other academic articles. For this use case it is important that the visualized data is clearly depicted without any distractions. Therefore the client required the program to offer 2D visualization with the option to add a custom made map. This way the user is able to depict trajectories and contours on different maps for different purposes.

#### No longer integrated with FlightRadar24

In the orientation phase of the project we came up with the idea to let our program support real flight routes from the flight-tracking app FlightRadar24. We thought that it would be an interesting feature to show at our final presentation. It would enable the user to select an airplane from FlightRadar24 and visualize its

noise contours real-time in our program. The client approved the feature, but he let us know that it was not necessary for him since he would not use this extension himself. ATO's research is primarily focused on optimization of an entire trajectory, not of partial observations. Hence, we addressed the feature as a could-have requirement at the time.

During the course of the project we found out that the API of FlightRadar24 was not public. When looking for alternative APIs that offer tracking data of flying aircraft, we found just one free option: the OpenSky Network. However, this API does not offer real-time data because of a 15 minutes delay. The whole idea behind the integration with FlightRadar24 was to visualize contours real-time for a particular flight. Since the use-case for this feature at the ATO department was limited, it was discontinued in favour of more advanced visualization like support for multiple trajectories.

#### **Added support for multiple trajectories**

AeroVision was initially built to provide visualization for a single trajectory. However, the client noticed that he would also like to feature multiple trajectories in the same visualization. This would enable the user to generate overviews of noise produced on airports over a longer period of time. Since this feature was added last-minute, we addressed it as a should-have requirement.

## **6.4. Development Process**

### **6.4.1. Trello for Project Management**

During our bachelor courses we had become familiar with task management in Trello. Trello is a software development workflow, in which tasks and user stories move through a pipeline of states. Examples of these states are "To Do", "In Progress", "Task Completed", "To Be Reviewed" and "Done." We complemented the use of Scrum with the use of the Trello workflow.

At the start of every sprint, new tasks would be created in Trello, split into smaller tasks that would then be assigned to one person or to us both (pair programming). Next, these tasks go through the states of the Trello workflow. The "To Be Reviewed" state was handled every two sprints by showing the new functionality to our client and discussing its impact.

### **6.4.2. Version and Quality Control**

AeroVision uses Git for its version control. Git offers branching as a built-in feature. This allows for a main branch that contains the stable version of the software, and multiple other branches that contain new features that are being developed. The initial plan on how to use version control included a merge request policy with the goal to increase code quality. This policy meant that only code that had been reviewed by the other team member may be merged into the master branch. Other methods that were used to increase code quality included *Coveralls* and code evaluation by SIG. These methods are described in Section 5.2.1 and Section 5.2.2 respectively.

In practice, merge requests were only used when important features or critical changes were introduced in the master. Otherwise, features could be silently added to the master branch as long as it was tested for at least 75% and it did not break the master branch. Tests could always be silently added to the master branch. The main reason for deviating from the original policy is because merge requests are time consuming, and most of the time we were pair-programming or sitting next to each other while working on the project.

### **6.4.3. Division of Labour**

During the project we worked together on most of the important features, such as the contouring and visualization models. This was primarily done to speed up the completion of larger tasks and improve the code quality. Besides these major tasks, each team member also took on a task as identified in Section 6.1. Hans worked on the GUI and optimization model. Elvan worked on the population class and the compatibility of the program with other (noise) calculation tools.

## **6.5. Reflection**

Although some deviations from the original plan have already been described in this chapter, this section focuses on the evaluation of the final method of execution of the described processes. Firstly, the Scrum methodology allows for more frequent feedback from the project coach to keep on track. This made it easier to focus on the requirements that matter to ATO. It was through one of these feedback rounds that the client

mentioned ATO prefers a properly working visualization tool over a system with half-implemented optimization and visualization features.

Something that could be improved upon was the use of code evaluations. In the current set-up, merging code through the strict policy of merge requests and code reviews proved to be too time consuming when the development team consists of two people. For future projects, better tools could be used to accomplish efficient code reviews. Due to the low number of formal code reviews that took place during the project, it was sometimes hard to make changes in code that was written by the other team member. This occurred most often with smaller system components, which were usually assigned to the person who either had the most time to spend on the task or was more skilled in performing that particular task. Code reviews would have ensured that we both had seen all of the code improving code quality and extensibility.

Sprint	Week(s)	Main Activities	Focus
1	1 - 2	Orientation in subject matter, client interviews, requirements gathering	Orientation, Research
	2	- Set-up AppVeyor + CoverAlls - Analysis of the context delivered in Research Report + Product Planning - Analysis of the design delivered in Emergent Architecture	Orientation, Design
2	3	Contouring calculation + GE overlay, basic aircraft animation	Development
3	4	Basic implementation trajectory optimization (point-mass calculation)	Development
4	5	3D visualization of contours	Development
5	6	2D visualization of multiple trajectories, speed-up optimization	Development
6	7	First version GUI, calculation population annoyance	Development
7	8	Full implementation trajectory optimization (operational constraints), Final version GUI, Visualization population annoyance	Development
8	9	Write final report and prepare end presentation	Evaluation

Figure 6.2: A roadmap summarizing the different sprints in the project, in which week one corresponds to the week starting on the 19th of April 2016.

# 7

## Product Validation

Every two sprints newly added functionality was discussed with the client to assess whether the new features meet test objectives, including correct implementation, quality verification, error identification and other valued detail. This was done through acceptance tests based on user stories, which can be found in our Research Report in Appendix A. In order to confirm whether the final version of AeroVision meets the requirements as described in Section 3.1, a last acceptance test was conducted with the client in the eighth sprint. This section describes the results of the last acceptance test by evaluating the implementation of the requirements. Requirements that are trivial, such as 'The program should be compatible with Windows 7 and later', are not mentioned in this section since they are indirectly covered by other requirements.

All performance tests were done on a test platform containing an Intel Core i7 processor, a memory of 4GB and Microsoft Windows 7 as the operating system.

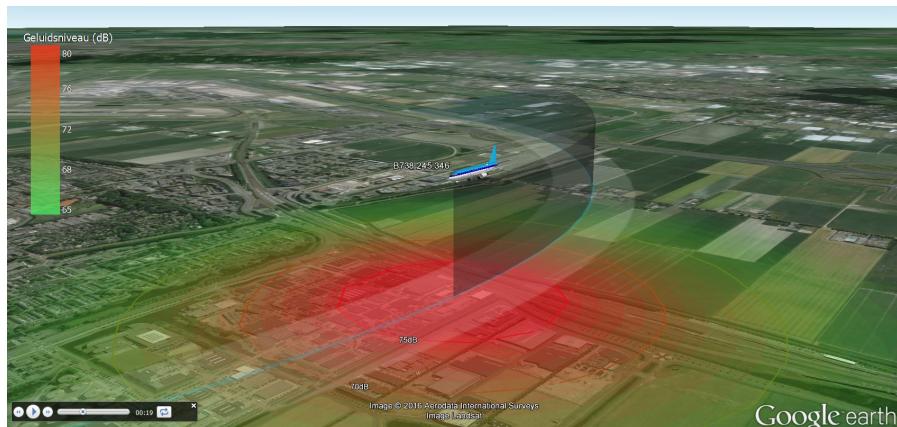


Figure 7.1: A 3D visualization of an example trajectory

### 7.1. Critical Requirements

#### 1. The program can calculate noise contours produced along a particular trajectory

AeroVision is able to calculate the contours along a typical trajectory of 20 kilometers (25,000 noise grid points) in the order of milliseconds. NoiseLAss does this in around a second.

#### 2. The program is capable of trajectory optimization based on noise contour areas and population annoyance of areas that are affected

The program is capable of multi-objective trajectory optimization based on a weighted sum of noise contour areas and population annoyance. When the optimization is finished, some characteristics are given on the optimal trajectory and a photo is shown displaying all evaluated routes. The user is able to visualize the optimized trajectory. Next to the course of the optimized flight path, the height and speed profiles can also be visualized.

**3. The program is able to evaluate 2500 trajectories for optimization in under 30 minutes**

AeroVision takes nineteen minutes to execute a typical run of its genetic algorithm on default settings (100 generations and 70 chromosomes). This means that Aerovision is able to evaluate 2500 trajectories in around eight minutes. NoiseLAss does 2500 evaluations in 50 minutes, so we achieved a performance improvement of 84%!

**4. The program is capable of trajectory visualization in 3D animation in Google Earth**

AeroVision offers the user the option to visualize single trajectories with produced contours in 3D in Google Earth. The client found the visualization clear and nice looking. He explained us that it was "above expectations".

However, during the last acceptance test, the client told us that he would like the program to offer two choices of contour visualization. One with filled contours using a color map as shown in Figure 7.1 and one with contour lines combined with population annoyance. At the time of the acceptance test we only offered a combined visualization of color mapped contours and population annoyance. The client really liked this approach, but in some cases he preferred the contour lines for a better overview. Therefore, we added the option to visualize contour lines last-minute to the program. Now, after choosing two-dimensional or three-dimensional visualization, the user is asked to choose between filled contours and contour lines visualization.

**5. The program is able to compute and visualize the effects of the aircraft on population annoyance**

The program is able to visualize population annoyance using default or user-supplied icons representing houses that change color and/or size when the people living there are annoyed by the produced noise levels. Two examples are shown in Figure 7.2 and Figure 7.3. The population density can be visualized with heat maps and user-specified shapes and color.

**6. The program can save the generated animation as a KML file**

AeroVision enables the user to save the generated animation as a KML file. We also added a screenshot button, allowing the user to save the produced animation as a .PNG file with one mouse click.

**7. The program executes all processes in an automated and pipelined manner**

The client approved the workflow of our program and was satisfied with the way visualization and optimization can be executed with just a few mouse clicks. He said that it was a major improvement on the workflow management and automation of ATO.

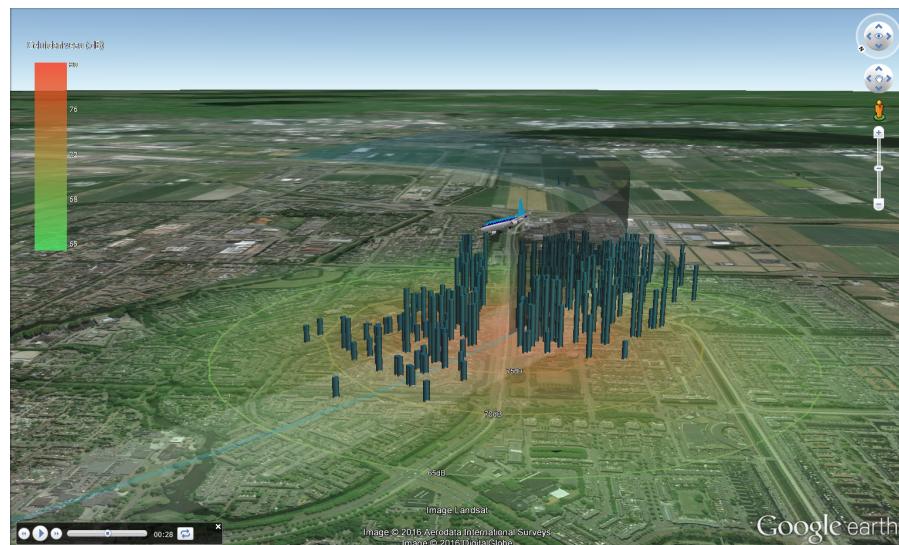


Figure 7.2: A 3D visualization of an example trajectory with bars that represent the people who are annoyed by the aircraft noise. The height of the bars correlates to the number of people being annoyed on that location.



Figure 7.3: A 3D visualization of an example trajectory with icons that represent the local population. The red icons show the people that are annoyed by the aircraft noise.

## 7.2. Non-critical Requirements

### 1. The program should be compatible with noise input from other (noise) calculation tools

The program is able to read in any file which complies with the input template in Figure !!!. This file format was chosen by the client since he found this format the easiest to generate.

### 2. The program can visualize multiple trajectories with animation and static visualization

The program can visualize multiple trajectories with both static and dynamic visualization. The client noted that he missed the visualization of the Day-Evening-Night Average (LDEN) noise level for this functionality. LDEN is the most meaningful level when assessing multiple trajectories over a duration of a day since it is a cumulative metric and therefore takes departure and landing times into account. After the acceptance test the LDEN noise level was added to this feature.

### 3. The program offers all tasks in a graphical user interface

The graphical user interface was approved by the client. He described its look as "intuitive and very clean".

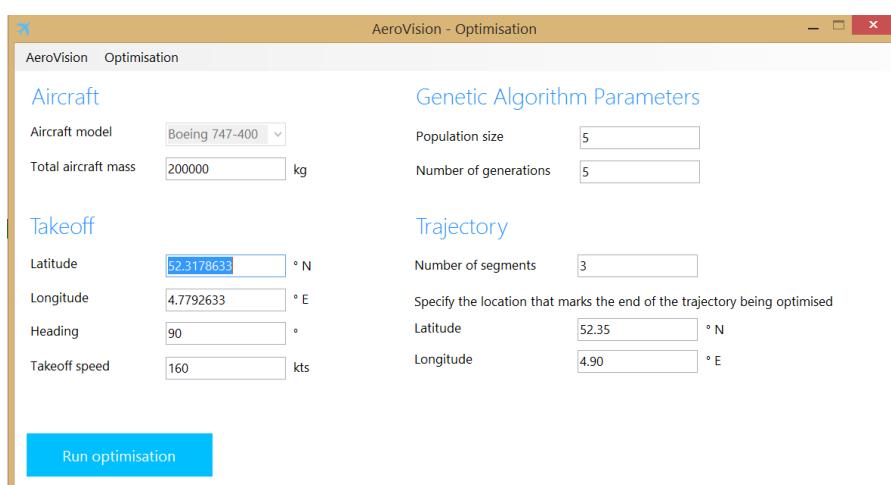


Figure 7.4: AeroVision's optimization window in which the user is asked to set the algorithm parameters.

#### 4. The program should show the estimated calculation time and current progress during trajectory optimization

During trajectory optimization and visualization the progress and estimated calculation time are shown with a progress circle. The client found this very helpful to manage time and to get a better overview of the calculation.

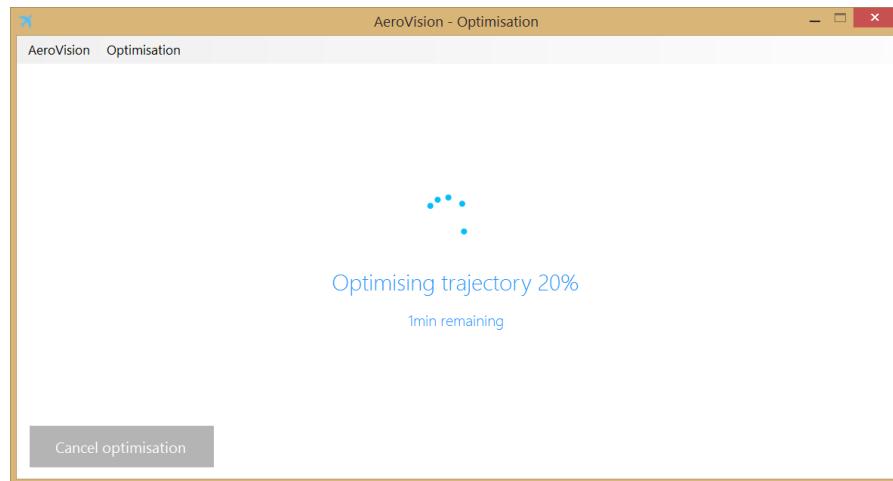
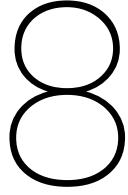


Figure 7.5: AeroVision's progress window is shown when the optimization is being executed.

### 7.3. Client Reflection

From the last acceptance test with the client we can conclude that the final version of AeroVision meets the functionality and quality required by ATO. All requirements are implemented except the integration with FlightRadar24. However, this was a could have requirement and had a limited use case for ATO. The client is very content with the performance achieved by the implementation of the optimization pipeline. He finds AeroVision to be a major improvement on the workflow management and automation of ATO. He let us know that the program will definitely be used by the research department and that he is open for future extensions.



# Ongoing and Future Work

This section is split into three major parts. First, Section 8.2 describes new functionalities that could be implemented in future versions of AeroVision. Second, Section 8.3 describes how the AeroVision source code will be open sourced, and a community can be formed to further develop AeroVision. Third, process improvements are described for future projects in Section 8.4.

## 8.1. New Functionality

The AeroVision project is designed to be potentially extended in the future. There are a few functionalities that we came up for future extensions.

### 8.1.1. Reverberation correction in INM

The noise calculations could be made more accurate by adding support for reverberation in mountain landscapes. Acoustic waves are reflected by walls or mountains. This can be heard when the reflection returns with sufficient magnitude and delay to be perceived distinctly. When sound, or the echo itself, is reflected multiple times from multiple surfaces, the echo is characterized as a reverberation [19]. In this kind of landscapes the noise level produced by an aircraft can be much higher than in flat landscapes. Currently, reverberation is not supported in INM [8]. For ATO this is not a problem since they will use AeroVision only for the study of airports in the Netherlands. Therefore, we decided not to spend time on this extension.

### 8.1.2. Integration with another noise model

As a future extension it would be possible to offer the user the option to enter a noise model other than INM as the standard noise model within AeroVision. This way the user would not be required to provide his own noise input when using an alternative noise model. Because of AeroVision's extensibility this would not be difficult to add. It would come down to replacing the INM executable and linking this with an input form in the user interface. We did not add this feature since it is unnecessary for the client. INM has been ATO's standard model for noise calculations since 2008.

### 8.1.3. Addition of graphs

The feature to generate graphs could be added to the visualization component of AeroVision. These graphs could show the evolution of certain conditions, like fuel consumption and noise production, during a flight route. This would allow the user to simulate any flight scenario and analyze the correlation between flight conditions. ATO did not have the need for the addition of graphs since they already use Matlab for this kind of functionality.

### 8.1.4. Optimization based on weather data

AeroVision's implementation of trajectory optimization could be improved by adding atmospheric, wind and earth models [37]. This way trajectories would be optimized by also taking the influence of wind and weather conditions into account. The system would assume that the user has this data in possession or the data could be retrieved online for instance from the National Climatic Data Center (NCDC), the world's largest active archive of weather data [23].

However, the addition of this feature would create the necessity of a dynamic optimization [7]. Various analytical methods are available to obtain a numerical solution in optimal control problems. A dynamic programming approach is the most suitable to solve a discrete deterministic process to achieve optimal flight trajectories in presence of winds. This would require some changes to AeroVision's current approach to trajectory optimization.

## 8.2. Publication of the AeroVision Code

Although the work the team has done on AeroVision for the thesis is now finished and a working final product has been delivered, the possibility for AeroVision to gain new functionality remains. It has been agreed with ATO that the software written for AeroVision will be open sourced through the popular open source code portal GitHub. The final version of the system will be released there with accompanying documentation and a reference to this thesis. The documentation will describe how AeroVision can be installed and used by providing a step by step user guide.

With the code open sourced other developers can provide new functionality to AeroVision, which will go through a pull request mechanism supported by GitHub. Developers will submit their new code for examination by a contributor (in this case the team members), after which the code can either be merged into the main branch, or be rejected specifying why it is not good enough to be merged into the project. This way quality of code can be maintained, whilst also providing sufficient opportunity for other developers to contribute to the project.

## 8.3. Process Improvements

In general, the processes described in Chapter 6 proved to be effective. There are no changes that we would like to introduce for the development process. For future projects, however, we will make sure to start the performance measurements at an earlier stage. In this project, the measurements started in the 8th week after we validated the output of all calculation modules. Because of this we still had to work on finding speed-ups and the optimal settings for the genetic algorithm of the trajectory optimization. we overcame this issue during the final weeks of the project. Luckily, the planning was constructed in such a way that other tasks could be rescheduled and time was made available to improve the performance of the optimization pipeline.

# 9

## Conclusion

This thesis focused on the main aspects related with the development of AeroVision, a software tool to optimize and visualize aircraft trajectories. The necessity of this kind of tools is supported for the new and future challenges introduced in aviation. Major airports are dealing with more than a thousand flights a day leading to considerable impact on the surrounding populated areas in terms of noise. Recent studies show that the most effective way of minimizing noise is the calculation of noise minimal approach routes. Due to modern navigation it is possible to define curved approach and departure procedures instead of the conventional straight flight tracks. To support their primary mission of understanding and reducing noise impacts of aviation, the ATO research department needed a standalone application for model-based optimization and visualization of aircraft noise. The goal of the project was to design a program that represents a creative and efficient way to minimize and visualize aircraft noise and other kinds of pollution along simulated and real flight routes.

The research questions posed at the start of this thesis, can now be answered as follows:

1. As described in Section 4.7 the contours were calculated using an algorithm that is slightly different in the way switch points are identified, and thereby more efficient than the one implemented in NoiseLAss. Instead of calculating and selecting the smallest clockwise angle between incoming and outgoing switch points, our algorithm only looks at the borders containing an incoming switch point. This is based on the assumption that for each incoming switch point an outgoing switch point with the same value exists. By traversing the borders with an incoming switch point all open contours can be constructed. The closed contours can be easily constructed from the remaining inner grid points.

According to the client's preferences AeroVision contains a Google Earth integration for the visualization of contours. The visualizations are based on KML files generated by the Animator submodule. This submodule receives one or more trajectories, the noise contours along the trajectories and a number of visualization settings (among which a definition of the camera position). An Animator composes each KML file out of three components, namely the set-up, animation step(s) and finish. Each animated object, such as the aircraft and flight path, has its own Animator object that produces the three KML components specifically for that object. The use of object specific Animators enabled us to automate the process of visualization by using a main method that calls all Animators to generate their corresponding KML files. The contours are visualized by the ContourKMLAnimator that creates a separate KML file containing all contour values of interest. The KML files are loaded via the Google Earth Server and displayed using the Google Earth Plugin. This approach allowed us to produce 3D visualization based on user defined settings and focused on user supplied contours of interest.

2. An effective way to tackle the problem of trajectory optimization is by using multi-objective optimization. The fitness of routes should not only be based on the actual noise values and population annoyance, but also on the noise contour areas. The contour area can be used to determine which people and to which extent they are annoyed by the noise. This is the innovative part of our algorithm. Previous algorithms only optimized trajectories for the volume of the noise. But by also taking the noise contour area into account, the trajectories are optimized for both the volume and spreading of the noise.

To efficiently implement trajectory optimization for minimum noise and/ or other environmental effects a genetic algorithm can be used. Finding the optimum solution in minimizing annoyance is not trivial as it has a very complex fitness landscape. Because population density distributions could have any shape, there are lots of local optima optimization algorithms could reach. Since in our problem domain the chances of getting stuck in local optimum are very high, hill climbing algorithms would not suffice. Using a genetic algorithm instead is a much better choice. A genetic algorithm is the best solution in this case since it applies random mutations to candidate trajectories. This effectively spreads the optimization throughout the whole search space and therefore makes genetic algorithms less prone to reaching a local optimum. A genetic algorithm with crossover as mutation method gave us the best optimization results. By implementing this algorithm multi-core and multi-threaded we achieved great performance results: 2500 evaluations can be done in under 10 minutes. Our implementation improved the response times of the NoiseLAss tool by 84%.

From the last acceptance test with the client we can conclude that the final version of AeroVision meets the functionality and quality required by ATO. The client is very content with the performance achieved by the implementation of the optimization pipeline. He let us know that the program will definitely be used by the research department and that he is open for future extensions.

Table 9.1: System requirements

#	Critical	Requirement Met	Description
1	YES	YES	The program can read in an arbitrary data file or text file specifying the input trajectory and grid
2	YES	YES	The program can calculate real-time noise contours produced along a particular trajectory
3	YES	YES	The program can save the calculated contour values as an arbitrary data file
4	YES	YES	The program is capable of trajectory optimization based on noise contour areas and population annoyance of areas that are affected
5	YES	YES	The program is able to evaluate 2500 trajectories for optimization in under 30 minutes
6	YES	YES	The program is capable of trajectory visualization in a real-time 3D animation in Google Earth
7	YES	YES	The program is able to compute and visualize the effects of the aircraft on population annoyance
8	YES	YES	The program can save the generated animation as a KML file
9	YES	YES	The program is compatible with Windows 7 and later
10	YES	YES	The program executes all processes in an automated and pipelined manner
11	NO	YES	The program should be compatible with input from other (noise) calculation tools
12	NO	YES	The program can visualize multiple trajectories with animation and static visualization
13	NO	YES	The program can read in KML files and directly visualize it in Google Earth
14	NO	YES	The program can cancel each (calculation) process while it is running
15	NO	YES	The program offers all tasks in a graphical user interface
16	NO	YES	The program should show the estimated calculation time and current progress during trajectory optimization



A

# Research Report

# Research Report

---

Elvan Kula & Hans Schouten

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Documentation on Calculation Models . . . . .	4
2.1.1	Noise Model . . . . .	4
2.1.2	Noise Contouring Model . . . . .	4
2.1.3	Trajectory Optimization for Minimum Noise . . . . .	4
2.2	Related Work . . . . .	5
2.2.1	INMTM v3.0 . . . . .	5
2.2.2	NoiseLAss v2.0 . . . . .	5
2.2.3	Visualization with Google Earth KML Objects . . . . .	6
<b>3</b>	<b>Requirements</b>	<b>7</b>
3.1	Interviews . . . . .	7
3.2	Requirements . . . . .	8
3.3	User stories and scenarios . . . . .	8
3.3.1	Must have features . . . . .	9
3.3.2	Should have features . . . . .	10
3.3.3	Could have features . . . . .	11
3.3.4	Won't have features . . . . .	12
3.3.5	Scenarios . . . . .	12
<b>4</b>	<b>Approach</b>	<b>14</b>
4.1	Development Methodology . . . . .	14
4.2	System Overview . . . . .	14
4.3	Programming Languages & Libraries . . . . .	16
4.4	Tools usage . . . . .	17
4.4.1	Functionality Tools . . . . .	17
4.4.2	Development tools . . . . .	18
4.4.3	Process tools . . . . .	18
4.5	API . . . . .	19
<b>5</b>	<b>Quality Guarantees</b>	<b>20</b>
5.1	Documentation . . . . .	20
5.2	Testing . . . . .	20
5.3	Evaluation . . . . .	21

## 1 Introduction

**Target Customers** Aircraft and airport noise are complex subject matters which have been studied for decades and are still the focus of many research efforts nowadays. Also at the department of Air Transport & Operations (ATO) at TU Delfts Faculty of Aerospace Engineering. ATO has three research aims: 1) To develop radical new ways to optimize aircraft operations for efficiency, safety, cost and environmental impact; 2) To extend the analysis to an airline fleet and network level to include capacity and resilience; 3) To synthesize these to include operational safety at an airline and ATM level. To support their research findings at conferences, the researchers at ATO need a standalone application for model-based optimization and visualization of aircraft noise. This report summarizes the research phase of this project, during which we familiarize ourselves with the field and investigate what programming language, libraries, tools, etc. are best suited for our product.

**Customer Needs** We will provide a product that represents a creative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. This requires the implementation of two mathematical models: one for the computation of noise contours and one for the iterative optimization of aircraft trajectories for minimum noise and population annoyance. The models will be deployed by the research team to predict aircraft noise along a particular trajectory (flight route) and to update the trajectory design in an iterated manner to minimize the produced noise over populated areas around airports. Therefore, the parameters that take part in the optimization of aircraft trajectories range from the generic criterion of contour areas to a number of site-specific criteria based on the impact on population

Additionally, the program should be able to visualize the noise produced along the trajectory through a 3D animation pictured on a real map. This requires a visualization of noise contours, which are noise footprints whose shape indicate areas of constant noise. Noise contours are a new subject to the research group and have not been implemented in relation with noise minimization before so this will be a challenging topic for us. The visualization should also show the effects of the produced aircraft noise on population annoyance.

In this orientation report we summarize the results of the research phase of this project as follows: first in Section 2 we give some more detailed information on the applications that we will extend and on similar products in the field. Section 3 focuses on the requirements gathering for this project, including a description of the various interviews we have conducted as well as the main requirements we identify. Section 4 then describes our chosen approach based on these requirements, including what programming language and tools we will use. Finally, Section 5 details what quality guarantees we will provide and how these are verified.

## 2 Background

The functionality addressed by our client requires the implementation of four models: noise model, noise contouring model, trajectory optimization model and the visualization model. For our project, we have investigated the documents with background information on these models that were provided by the client.

In the first section we will further describe the way these models work and the way our product will incorporate them. We also compare the functionality of our product with existing work. The Noise Model and Noise Contouring Model are further detailed in Sections 2.1.1 and 2.1.2 respectively. Then we will describe the original approach to the trajectory optimization and compare this with the new approach suggested by the ATO department in Section 2.1.3. The second Section covers existing software on related work. We will look into software provided by the client and visualization in Google Earth with KML objects.

### 2.1 Documentation on Calculation Models

#### 2.1.1 Noise Model

The general algorithm of noise calculation is based on empirically obtained noise-power-distance (NPD) tables. The first step consists of interpolating for the current thrust level and slant range (observer aircraft) to find the uncorrected noise metric. Since the NPD-tables are created under the assumption that the aircraft flies on an infinitely long segment at a given reference speed a number of corrections need to be applied. For more information on the methodology you are referred to the INM7.0b Technical Manual.

#### 2.1.2 Noise Contouring Model

The methodology for calculating the contour areas partly follows the standard closest point search common in contouring algorithms, but differs in the fact that It will avoid incorrectly forming multiple contours when in reality only one single closed contour exists. In the case that contour areas are required the spline representation is used to refine the grid to cells of 125x125 m. After refining the grid, all the switch points in the new grid are located. Switch points are points on the axes of the grid cells in which the noise value changes from a value higher than to a value lower than the requested contour value. After gathering all switch points within the grid, the points need to be ordered to form a continuous string. For this purpose, starting in point  $i$ , candidate points for  $i + 1$  are located and evaluated for feasibility. This process is further explained in detail in documentation about the NoiseLAss tool v2.0.

#### 2.1.3 Trajectory Optimization for Minimum Noise

In general, an optimization of the noise impact requires criteria that range from the generic criterion of contour areas to a number of site-specific criteria based on the impact on population or on enforcement points. Enforcement points are locations in the vicinity of the airport at which a maximum value for a specific noise metric is defined. This value is then used for regulatory purposes to control the noise exposure near the airport.

The optimization algorithm first gathers the noise input on an arbitrary grid with cell size  $gx$  and  $gy$ , where  $gx$  is not necessarily equal to  $gy$ , allowing for a grid consisting of non-square cells. The original grid is then used to define a set of b-spline coefficients, which form the basis for a b-spline interpolation of the data. For all site-specific criteria the spline coefficients are directly used to find the noise values in the requested observer or enforcement points by evaluating the b-spline in the required coordinates.

## 2.2 Related Work

Software tools that implement the four models already exist and are already being used in the aerospace industry and research field. The problem is that these tools lack automation and function as separate executables. Currently there is no program that executes all these models in a pipelined manner. In this section we will only discuss the tools that are currently used by the ATO department since our program will incorporate algorithms equal or similar to the ones used in these tools.

### 2.2.1 INMTM v3.0

The Noise Model is implemented in this noise calculation tool created by respectively the TU Delft and the NLR. It implements the FAAs standard methodology for noise assessments. Since 1978 this has been the standard Integrated Noise Model (INM) in over 65 countries. The tool computes noise levels expressed in six time-based metrics at a user-specified regular grid based upon finite flight-segment data. It requires two user-provided text files: one describing the full trajectory expressed in a number of nodes and one defining a 2-dimensional (no elevation data) or 3-dimensional grid for noise calculation.

The calculations in this tool are conform world standards and are performed below one second. The client was content with this tool and therefore we decided to re-use and (if needed) further improve the tool in our final program.

### 2.2.2 NoiseLAss v2.0

The noise contouring algorithm and optimization model are implemented in this noise level assessment tool. The tool is basically a collection of assessment criteria which can be retrieved according to user specification whilst using a common input and output format.

The NoiseLAss tool calculates the noise levels in the enforcement points by interpolating the user-supplied noise input data on the specific point. This allows the user to define other points at which the noise level is of interest.

Additionally, the tool contains a number of dose-response relationships to assess the direct impact on population for six different noise metrics. These relationships can be used to quantify the impact of commercial aviation on near-airport communities for daytime and night operations, e.g. the number of expected awakenings due to a single night-time flyover and due to cumulative noise metrics.

Although the model is directly compatible with the output files of the INMTM v3 noise calculation tool, it still requires user supplied input files and does not perform real-time. Therefore the client wants us to build the contouring and

optimization models from scratch for our program following the algorithms used in this tool.

### **2.2.3 Visualization with Google Earth KML Objects**

KML (Keyhole Markup Language) is a file format used to display geographic data in an Earth browser such as Google Earth. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard. KML also enables the user to store tours and animations in Google Earth. Since KML meets all the requirements of our flight animation and since the ATO department is already familiar with this file format, KML seemed a perfect fit for our project. Next to KML we also considered alternatives like CityGML and Geography Markup Languages. Although these alternatives are fine, KML is better integrated with Google Earth. Therefore we decided to use KML objects with the Google Earth API for visualization in our program.

### 3 Requirements

For the process of requirements gathering we have mainly focused on interviewing our client Dr. Ir. Sander Hartjes, who is also a representative of the ATO department because of his academic function. He is equipped to inform us how the research department applies the mathematical models and what features in our program they would benefit from most. The resulting requirements and the main results of the interviews will be described in this section as follows: we summarize the results of these interviews in Section 3.1 and give a list of the requirements in Section 3.2.

#### 3.1 Interviews

To get a clear view of the customer's needs we interviewed our client Dr. Ir. Sander Hartjes during two extensive meetings of two hours each. Besides this, other meetings were also held with him to discuss and clarify the aerospace engineering part of the project. This was necessary since we did not have any knowledge of the theories of noise contouring and noise minimization before the project.

Dr. Ir. Sander Hartjes is employed as an Assistant Professor in the field of airline operations at the chair of Air Transport and Operations. His research mainly focuses on optimal aircraft performance regarding the reduction of noise and pollutant emissions. He has already been working at ATO since 2008 and therefore he is equipped to inform us on the way the department is going to use our program and what features the department will benefit from most.

Through our first interview with Sander we learned that the ATO department lacks a visualization tool for its research findings at conferences. The researchers, including Sander, still manually transform Matlab files containing trajectory coordinates into KML files and enter this into Google Earth for visualization. Sander told us that they increasingly need this process to be automated when dealing with large trajectories. This would spare a lot of their time that goes into manually importing flight data from Matlab into KML. Sander also informed us that their preference goes out to visualization in Google Earth since this is commonly used in their research field.

Additionally, the current visualization only consists of a simple animation of an airplane following a particular path in Google Earth. Sander told us that they would like to be able to visualize noise contours in a real-time 3D animation. The ATO researchers create noise contour diagrams with 3D model simulation software such as AutoCAD, but currently there is no way for the department to animate noise contours that are produced along a particular trajectory. Sander let us know that the implementation and visualization of noise contours in Google Earth is a crucial requirement for the program.

During our second interview Sander informed us, after consultation with other researchers of the department, that they would like our program to visualize noise contours not only in relation with produced noise levels but also regarding to iterative optimization of flight trajectories for minimum noise. Therefore, an optimization algorithm needs to be implemented that should not only take the actual measured noise volumes into account but also the area of the noise contours. The trajectories would then be optimized based on the volume and spreading of the noise. Recent studies show that the timing and spreading of

noise affect the population annoyance greatly and thus they are potentially important factors in the minimization of noise. This way of minimizing aircraft noise and visualizing noise contours is not only new to the ATO department but also to the research field itself. Noise contours are normally used to regulate sound and not as an indication for the optimization of trajectories.

Finally, when Sander showed us the tools that are currently used by the ATO department to calculate noise values and optimize trajectories, we noticed that these tools also lack automation. At the moment ATO researchers manually transform the output of the noise model in the right data form and then enter it into the optimization model. The research department would benefit greatly by an automated and pipelined execution of these processes in a standalone application.

### 3.2 Requirements

Below we present an initial list of formal requirements for the program. Additionally, we provide an extended list of user stories to specify the required behavior of the program in different scenarios.

- The program can calculate noise values in real time.
- The program can calculate and output the actual noise contours that are produced along a particular trajectory in real time.
- The program has the option to optimize a trajectory in an iterated manner to minimize the produced noise over populated areas that are affected. The optimization algorithm should be based on the area of the produced noise contours instead of the actual noise values.
- The program can visualize the (optimal) trajectory together with the noise contours in real-time 3D animation mapped on Google Earth.
- The program can calculate and visualize the effects of produced noise on population annoyance using the awakenings algorithm.
- The program should save the results of the visualization and, if requested, the intermediate calculated values in a particular format and directory specified by the user.
- The program should execute all the processes above (corresponding to the computation of noise values, noise contours, noise minimization and the visualization of noise contours) in an automated and pipelined manner.
- The program should offer all these tasks in a graphical user interface.
- 

### 3.3 User stories and scenarios

The main requirements that a user of the system will actually notice are most clearly presented in so-called user stories. These stories describe what the results of an action will be for a certain user. Unfortunately, some of the other requirements or design decisions can not be represented in the same format that

easily, since they do not really involve a user in the classic sense of the word. Instead some of the functions that the program should be able to use have been described in scenario format.

The user stories that describe the behavior as presented to the user can be summarized as follows:

### 3.3.1 Must have features

These features are must haves since the program would not be functioning and meet the (minimal) requirements of the customer without these implemented.

---

- **Given** I am an ATO Researcher and I want to specify a grid and trajectory,
  - **When** I read in an arbitrary data file (.dat extension) or text file indicating the input flight trajectory and grid (based on the RD-coordinate system),
  - **Then** the program should be able to handle this input without any problems.
- 

- **Given** I am an ATO Researcher and I want to save the resulting visualization,
  - **When** I click the button to save the 3D animation,
  - **Then** the program should save the used data and settings in the specified directory.
- 

- **Given** I am an ATO Researcher and I want to easily access a saved visualization during a conference,
  - **When** I select the file representing the previously stored visualization,
  - **Then** the program should automatically start the visualization without requiring any further user interaction.
- 

- **Given** I am an ATO Researcher and I want to calculate noise levels produced along a particular trajectory,
- **When** I click the button to start the noise calculation,
- **Then** the program should calculate and output four types of noise levels; A-Weighted Sound Exposure Level (SEL), Effective Perceived Noise Level (EPNL), A-Weighted Maximum Sound Level (LAMAX) and Tone-Corrected Maximum Perceived Noise Level (PNLTM).

- 
- **Given** I am an ATO Researcher and I want to calculate noise contours produced along a particular trajectory,
  - **When** I have selected the decibel value(s) I am interested in and the program has finished calculating the noise levels (former step),
  - **Then** the program should automatically start calculating the noise contours along the given trajectory for the decibel values I have selected.
- 

- **Given** I am an ATO Researcher and I want to save the calculated noise contours,
  - **When** I click the button to save the noise contour,
  - **Then** the program should save the noise contour as a csv file in the specified directory.
- 

- **Given** I am an ATO Researcher and I want to optimize a particular trajectory by minimizing the noise produced by the aircraft,
  - **When** I select the option to optimize the trajectory,
  - **Then** the program should start and keep updating the trajectory until no optimization is possible.
- 

- **Given** I am an ATO Researcher and I want to visualize a particular trajectory and the produced noise contours,
  - **When** I select the option to visualize the trajectory,
  - **Then** the program should show a real-time 3D animation mapped on Google Earth
- 

### 3.3.2 Should have features

These features are should haves since the basic components of the program would work without these features but it would be better (meet the requirements of the customer better) if these would be implemented.

---

- **Given** I am an ATO Researcher and I want to easily access a saved visualization during a conference,
  - **When** In windows explorer I double click the file representing the previously stored visualization,
  - **Then** Windows should automatically launch our application and start the visualization without requiring any further user interaction.
- 

- **Given** I am an ATO Researcher and I chose the wrong input file,
  - **When** I notice my mistake during the program run and I want to disrupt or cancel the current operation,
  - **Then** the program should abort the current operation after I clicked the cancel button.
- 

- **Given** I am an ATO Researcher and I want to analyze a real flight route from the FlightRadar24,
  - **When** I select to use FlightRadar24 as data source and specify a flight,
  - **Then** the program should fetch all data of the selected flight and start the noise calculation and visualization.
- 

### 3.3.3 Could have features

These features are could haves since they are not necessary to the customer and to the functioning of the program but it would be nice to include them if there is enough time left in our timeframe.

---

- **Given** I am an ATO Researcher and I want to analyze an active flight from the FlightRadar24,
  - **When** I select to use FlightRadar24 as data source and specify a flight,
  - **Then** the program should start fetching the data of the selected flight and start the noise calculation and visualization in real time.
- 

- **Given** I am an ATO Researcher and I want to specify a grid and trajectory,

- **When** I read in a Matlab file (.mat extension) indicating the input flight trajectory and grid (based on the RD-coordinate system),
  - **Then** the program should be able to handle this input without any problems.
- 

- **Given** I am an ATO Researcher and I want to estimate how long I have to wait for the results of the program,
  - **When** I clicked the button to start the noise calculation,
  - **Then** the program should continuously show me the current progress with a progress bar.
- 

- **Given** I am an ATO Researcher and I want to share the program with others,
  - **When** I start the program.
  - **Then** the program should be recognizable in a blink by showing its name and logo.
- 

### 3.3.4 Won't have features

These features are won't haves since the customer does not actually need these features to be implemented. There probably won't be enough time left for the following features but it could be interesting for a follow-up phase of the project.

---

- **Given** I am an ATO Researcher and I want to log all my activities in the program,
- **When** I click the logger button,
- **Then** the program should create a text file with an overview of the input files I entered and visualized during the current program run.

### 3.3.5 Scenarios

Whereas some scenarios will be developed as new features are picked to be implemented, a few scenarios that result in the automated execution of the program are given here:

---

- **Given** an input file indicating the trajectory and grid,
  - **When** the user selects to optimize and start the noise calculation,
  - **Then** the program should perform all operations including visualization in an automated manner without the user needing to perform any additional actions.
- 

- **Given** a particular trajectory and its corresponding noise contours (calculated by the program),
  - **When** the user has selected the option to optimize the trajectory,
  - **Then** the program should optimize the trajectory based on the area of the noise contours instead of the actual noise values.
-

## 4 Approach

In this section we will describe our approach to the project, starting with the development methodology as described in Section 4.1. In Section 4.2 we present a broad overview of our system design. The programming languages and tools we will use for the project are specified in Sections 4.3 and 4.4 respectively. Finally, the API of FlightRadar24 will be introduced in Section 4.5.

### 4.1 Development Methodology

During the software development phase, the team will be applying agile development methods. In particular, the Scrum development framework will be used. Alternatives we have considered include the waterfall methodology and XP, extreme programming.

We have opted not to use the waterfall method due to its rigid and inflexible nature. For this project we foresee having to cope with design changes, which is difficult and time-consuming once implementation has started in the waterfall method. Throughout the project we will be implementing multiple mathematical models and designing a GUI that should be responsive and user-friendly in all situations. In practice, the design of a GUI often has to go through multiple revisions to accommodate new requirements, incorporate new insights in the problem, etc.

On the other hand, agile methods feature an iterative approach to software development, and are flexible enough to alter the design of a product throughout a project. As a result, using an agile method enables us to focus on the core of the product during initial stages, and add more functionality as the project progresses. It also allows us to keep a list of features that are not required for a functional product, but would increase its usefulness if implemented. These features can be reprioritized based on the clients demands to ensure the final product contains the most requested and most valuable features that could be implemented in the limited timeframe.

Our choice for Scrum over XP as the preferred agile method is mainly based on experience of the team working with Scrum. On top of this, extreme programming is less suitable for small teams due to its strict requirements on code reviewing and testing, i.e., using pair programming. Due to the limited time available for this project, in some situations we prefer being able to work in parallel to ensure we cover more features. Additionally, in Scrum the work to be performed is planned in the sprint planning. We will work with one-week sprints. At the beginning of every week we will describe our goals for that week in a sprint plan. At the end of the week, all the items on the sprint plan need to be finished, fully tested and committed to the repository. A sprint reflection should also be made in which we discuss our progress during that sprint and possible improvements for the next sprint. Section 5 contains more detail on how we will ensure sufficient quality using our chosen methodology.

### 4.2 System Overview

As depicted in the diagram above, the models present the subsystems of our program. The Noise Model is responsible for the calculation of the noise values produced along a particular trajectory. Based on these noise values the Contour Model computes the noise contours that the user is interested in. Then the

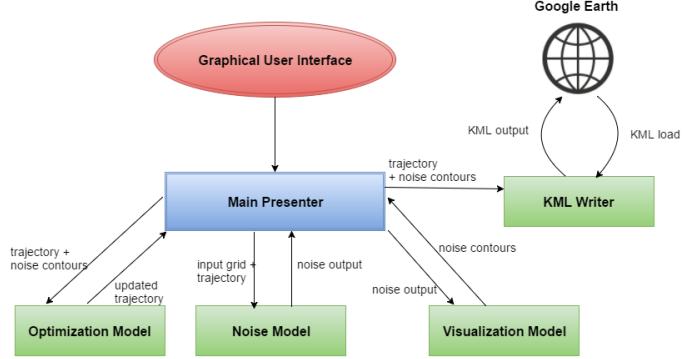


Figure 1: A broad overview of the system architecture

Optimization Model updates a trajectory in an iterated manner based on the calculated noise contours. The KML Writer creates the KML files based on the (optimal) trajectory and its noise contours. These KML files will automatically be entered into Google Earth for visualization. Changes to the KML files will automatically be transferred to Google Earth by a 'KML load' script, which periodically loads the files into Google Earth.

The subsystems exchange their outputs with the main presenter which uses the output of one subsystem as input for the next one. This happens according to the following pipeline:

1. The main presenter enters the initial input file of the user into the Noise Model.
2. The main presenter passes on the output of the Noise Model to the Contour Model.
3. If the user selected the option to optimize, the main presenter relegates the output of the Contour Model to the Optimization Model. Otherwise, the output of the Contour Model is directly passed on to the KML Writer. When the output of the Contour Model is first handed over to the Optimization Model a loop is initiated: the current trajectory is updated by the Optimization Model and the workflow of the program starts all over again by entering the updated trajectory as input into the Noise Model. This optimization process is repeated until no further optimization is possible.
4. The main presenter relegates the output of the Contour Model and the coordinates of the (optimal) trajectory to the KML Writer.

Following this method the models are executed in a pipelined manner. But the subsystems will be designed in such a way that, given the right input files, the subsystems can also be run separately from each other and function as standalone applications.

### 4.3 Programming Languages & Libraries

Because our program needs to solve optimization problems that require running computationally expensive calculations on a large amount of data points, code performance is an important factor. To this end, we hesitated between C++ and C# and decided to compare them.

C++ is known to be able to support high performance applications. However, we stumbled upon a few articles and blogs in which the performance of C++ was compared to C#. All these articles noted that C++ itself does not have speed advantages. It is the fact that highly experienced programmers can express the strengths of the language correctly which makes C++ code in some cases favorable over C#. From these statements one can conclude that a performance-conscious C# programmer can write programs with a performance similar to a well-written C++ program, however with the ease of being programmed in C#. Since we both are not that experienced with C++, being able to write high quality C++ code and to actually achieve a higher performance would take practice and time to learn and thus would be very time consuming for us. This could even negatively affect the functionality of our program. Therefore, we believe that the use of C# over C++ will increase our productivity. Even if a programmer succeeds to get a better performing implementation, the performance increase of C++ would probably not weigh up to the additional features that could have been added as a result of the time saved by programming in a more straightforward language.

Besides that, C++ has cross platform capabilities whereas C# depends upon the .NET framework which restricts its use to only the Windows platform. However, the benefit of having cross platform capabilities is not that important in our case since the client will use the software in combination with Matlab and Google Earth in a Windows environment. Therefore, we considered a few of the required components of our program with C++ and C# in our minds. Our program needs to be able to generate XML files, PNG files and will make use of a library with a collection of common data structures. After a quick search on Internet we noticed that such libraries were more easy to find for C#. Also, C# is specifically designed to work with the .Net framework and is geared to the modern environment of Windows and user interface. Therefore C# offers a lot more functionality and flexibility for GUI programming. With C++ you would have to hand code a lot of GUI functionality.

Mainly the first consideration regarding code performance and the GUI functionality that is built in C# seemed to us as valid reasons for choosing C++ over C#. To this end, we choose C# as the main language for our code.

**.NET Framework for GUI Development** After we decided to choose C# as the main language for our code, it was an obvious choice to develop our GUI with the .NET framework. The .NET Framework has become the mainstream environment for new Windows applications. It offers a lot of fancy options in case we decide to use innovative UI objects. Besides, the .NET framework contains a lot of predefined libraries for graph rendering and the creation of PNG's and color maps (which we are actually going to need for the visualization model). This helps to eliminate the amount of unnecessary codes and involves less coding for the developers. The framework also supports System.XML, which lets applications work with XML-defined data, including using XSLT and XPath. This comes in handy for us since we are going to use KML objects that are

based on XML-defined data for the visualization model of our program.

But we still had to decide between using Windows Forms or WPF. The single most important difference between WinForms and WPF is the fact that while WinForms is simply a layer on top of the standard Windows controls (e.g. a TextBox), whereas WPF is built from scratch and doesn't rely on just the standard Windows controls. This might seem like a subtle difference, but it really isn't, which you will definitely notice if you are either creating a very complex or a relatively easy GUI. Since WinForms restricts users to a collection standard windows controls, a GUI that requires control elements that deviate from this is almost impossible to construct. By default WinForms for example does not support adding a background image to a ListView component. A programmer who really wants to seek the limits of what is graphically feasible will stumble upon a vast number of restrictions in WinForms. With WPF on the other hand control elements can be created by the designer in a container like fashion. In WPF for example a button with background image is created by adding a panel containing the image to the button. This is a great strength of WPF, however the current WPF form designer makes designing forms in WPF more tedious since you have to do significantly more work yourself. To give an example, the alignment of control elements requires more effort and is less obvious in WPF compared to WinForms. Since our GUI will mainly focus on requesting parameters that are required for performing the calculations, no complex control elements nor heavy graphics are necessary. Therefore, for our Windows application in which the standard Windows look and feel is expected WinForms will suffice and with the currently superior design environment will save us precious time.

## 4.4 Tools usage

In order to develop a product of this size, both development and process tools can help to keep an overview of the tasks at hand. In order to easily keep track of the increasing number of lines of code, a good IDE and some version control software can be very insightful. Similarly planning tools such as the agile-orientated JIRA can help to provide a clear overview of the current ToDo-list. In Section 4.5.1 we describe the development tools we will use in this project, whereas Section 4.5.2 will focus on the process tools.

### 4.4.1 Functionality Tools

**INMTM v3.0** This noise calculation tool was provided by our client. It implements the FAAs standard methodology for noise assessments and therefore the calculations in this tool are conform world standards. The client is really content with this tool since its execution time is less than one second. Therefore we decided to re-use the tool in our final program. In our minds we hold the option for further performance improvements in later stages of our project. The tool is written in Fortran so this means that, if we would like to improve the performance further, we would need to learn Fortran in a short period of time or we would need to rewrite the entire noise model from scratch. This is something we should take in consideration depending on the overall performance of our end product and the customer's needs.

**Google Earth** Our client informed us that their preference goes out to visualization in Google Earth since this is commonly used in their research field. Besides, Google Earth provides a lot of options for satellite imagery of the entire earth in an interactive format. Compared to all its alternatives like Marble and Here Maps, Google Earth provides the most complete satellite imagery of the entire earth in 3D. One disadvantage of Google Earth is that the resolution of the image varies depending upon the location. This may make it more difficult for the user to view certain areas that are isolated or uninhabited. But since our visualization will mainly be located in the Netherlands and in populated areas surrounding airports, this should not be a problem for our program.

#### 4.4.2 Development tools

As has been previously described in Section 4.4, we will develop the largest part of our code in C#. For this part of the code, we will use Visual Studio. Visual Studio is both commonly used for C languages and is an IDE we have experience with. The alternative in the form of SharpDevelop has been considered, but since we have experience with Visual Studio, we believe this will allow for a smoother workflow and switching to the similar but slightly different SharpDevelop would hold no advantages. In contrast to an extensive IDE such as Eclipse, the alternative of Vim has also been considered. Vim is a command-line text editor that offers many development features for those that are used to its somewhat peculiar set of shortcuts. The downside is that for large projects it is quite difficult to keep a clear overview of all code.

In addition to an IDE, we will also require version control to easily keep track of code changes. Version control systems manage changes in code and/or documents, and provide a central location to store the code and documents. We choose Git as our version control tool. Our project team has experience with both Git and SVN, and both version control systems are commonly used in academia and industry. As we are using an agile development method, a working version of the product, preferably with new features, should be presentable at the end of every sprint. To maintain a stable version of the product, branched development can be a huge benefit. This allows for a main branch that contains the proven to be stable version of the software, with new features being developed in separate branches. Because Git has built-in support for branch-based development, it is preferred over SVN which offers a very primitive branching system which involves manually creating the folders for branches and merging them afterwards.

Given that the core of our application is written in C#, multiple testing frameworks are available. Out of the different options we decided to go with NUnit, since it is the most similar one to JUnit which the team has the most experience with. Additionally, NUnit integrates seamlessly with Visual Studio and it fully supports test class inheritance. MsTest has slightly more features for test logging, but we still choose for NUnit due to its native support in Visual Studio and our previous positive experience with JUnit.

#### 4.4.3 Process tools

In addition to the tools required for the development, we are also using a planning tool to keep track of our planning. JIRA is an online tool that allows for an overview of the current work, as well as some scrum-oriented features. One

of these features is the scrum board which mimics the structure of ToDo, Work in Progress, Done that is often done with sticky notes on a white board.

## 4.5 API

**FlightRadar24 for real-time flight data** For visualization of real flight routes we are going to use the API of FlightRadar24 to gather real-time air flight data. Its current version provides the broadest coverage of aggregated flight tracks and abstract information. The data is retrieved by a network of ADS-B receivers. FlightRadar24 does not provide a public API but they have a streaming endpoint for their data in JSON. We will be using a PHP API which is developed by a third party and provides all these data in JSON.

We considered the alternative API of FlightAware. They seem to have better routes/ flight plan data than FlightRadar24 but their feed is 5 minutes delayed. Because we want to visualize real flight routes (near) real-time, the delayed feed is no option for us. Another potential alternative is the OpenSky Network. This is a free open-source API that provides live visualization of air traffic and a real-time flight data feed. In contrast to FlightRadar24 it also offers access to historical raw data which would be very useful for testing our program. However, the location data feed is not that accurate and it contains no information on the flight status, such as the type of plane.

## 5 Quality Guarantees

In this section we talk about the quality of both the process and the code. In Section 5.1 we talk about the type of documentation we use in the code and why. In Section 5.2 we talk about the way we write and run tests. In Section 5.3 we will discuss the evaluation of the quality of our code, and how we guarantee the quality of our reports and documents.

### 5.1 Documentation

Two different types of documents will be required for this project, the rst being documentation of the process, including this document, the second being code documentation. All documents related to the process will be produced in LaTeX, supplemented with BibTeX when references are a part of the report. For these documents the version control system Git will also be used.

For our project, we choose to use the built in documentation tags in Visual Studio and Doxygen for documentation. The reason for using Doxygen is that it is one of the most popular documentation generation tools available for C# and numerous other programming languages like C and Python and therefore requires a flexible documentation generation tool. We choose either the standard deployed by Visual Studio or Doxygen if there is no standard, to generate documentation for the extensions.

### 5.2 Testing

As new functionalities are added to a product, it is always important to verify that existing functionality is not broken during the latest update. To this end a test suite containing both unit tests and integration tests can be a great help. To evaluate our code, we use test-driven development. With test-driven development,

Unit-tests are written before the code that should be tested. This allows us to specify the working of our code through the tests that is initially based on our design choices. These tests often take the form of user stories: I am X and if I do Y, then Z will happen. The use of a test-driven development approach leads to spending less time on debugging code, and to more modular and extensible code in general. Additionally, we want our code to be well-thought-out and ensure that both team members have a large understanding of the code. To this end, the unit-tests for a certain part of the product code are written by a different team member than the actual product code.

Simply writing the tests is of course not sufficient for them to be useful, since tests also need to be run to get any data out of it. A continuous integration server with hooks into Git allows you to have the tests run automatically on every push to the server. We will use this hook to run our unit tests using mocks after every push. Jenkins is a commonly used example of such a continuous integration platform and is also the one we will be using for this project. Though we have some experience with CruiseControl as well, Jenkins is more commonly used and provides all the required features. Testing in a live environment will be done at the end of every sprint, outside of the continuous integration tools.

### 5.3 Evaluation

Because we want to deliver a high-quality product to our customer, we evaluate both our code and documents carefully before delivery. To this end, we use multiple methods.

All documents we deliver are written using an iterative method. This means all sections are both read and edited by the other team member. This method has the following benefits:

- Both team members are aware of document contents.
- Section contents are refined and reflect the accurate opinion of the team.

At the end of the development process, we use benchmarking techniques to evaluate the performance of our software. Because the ATO department is going to use the program for actual flight trajectories, obtaining real-world representative workloads is a non-trivial task. The performance evaluation will consist of running real flight routes as input.





## SIG Evaluation

### B.1. Initial Evaluation

[Analyse] De code van het systeem scoort bijna 3.7 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de langere methoden ook naar voren als de meest complexe methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'KMLAnimationStep'-methode in ContourKMLAnimator.cs, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Binnen de extreem lange methodes in dit systeem, zoals bijvoorbeeld de 'FindNext'-methode in ContourPoint.cs, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

In jullie project zit er methodes zoals 'FindNext'-methode in ContourPoint.cs die lang zijn en aparte functionaliteit daarin zichtbaar is (bijvoorbeeld het vinden van het volgende punt op basis van de richting). Aparte functionaliteit moet in aparte methode met zelf-sprekend naam ontwikkeld worden.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.



C

## Product Planning

### Product Planning

---

April 29, 2016

## C.1. Introduction

The goal of our project is to create a standalone software application which can be used by researchers from the Air Transport & Operations department at TU Delft for model-based optimization and visualization of aircraft noise. To achieve a good development process and a successful software product careful planning is required.

In the second chapter of this document an analysis of our program is provided by describing the context, problem and stakeholder inputs. Based on this the features needed for the product are described in a high level backlog and the steps needed to complete the project are defined in a roadmap. Here you can find the major release schedule and goals.

Chapter 3 contains a detailed release plan specifying our milestones per release. In chapter 4 a definition of 'done' is given on backlog items, sprints and releases. This will enable us to know when a feature is done and when the corresponding sprint item can be closed.

## C.2. Product

This chapter gives a high level overview of the product. Paragraph 2.1 gives a brief analysis of the program and high level overview of the product backlog. In paragraph 2.2 the steps needed to create this program will be described in a roadmap of major releases. Here the major releases of the program are given in a clear overview.

### C.2a. Program Overview

#### Analysis context

Aircraft and airport noise are complex subject matters which have been studied for decades and are still the focus of many research efforts nowadays. Also at the department of Air Transport & Operations (ATO) at TU Delft's Faculty of Aerospace Engineering. ATO'S primary mission is aimed at increasing the understanding of noise impacts, identifying solutions to reduce those impacts and educating people on the issues. To understand flight navigation concepts and support their research findings at conferences, the researchers at ATO need a standalone application for model-based trajectory optimization and visualization of aircraft noise.

**Analysis problem** The ATO research department needs a program that represents a creative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. This requires the implementation of a mathematical model for aircraft noise and aircraft trajectory design. The model will be deployed by the research team to predict aircraft noise along a particular trajectory (flight route) in order to be able to minimize the produced noise over populated areas around airports. In order to implement this model the parameters that are required for the sound calculation need to be derived mathematically. Besides that, the program should also be able to visualize the contours produced along the simulated trajectory pictured on a real map. This requires an implementation of contours, which are 'noise footprints' that usually indicate areas of constant noise. However, they can also be used to depict other environmental effects of aircraft, such as gas emissions. Since ATO's research department focuses on the general environmental impact of aircraft, the program should be generic enough to help engineers in assessing the effect of noise or pollution in populated areas.

**Analysis stakeholder inputs** The application needs to be able to read in two arbitrary data (.dat extension) or text files indicating the input trajectory (including selected airplane model) and the grid that needs to be used. In case the user does not provide his own noise values, the files will be entered to the noise model. The noise model will then calculate the noise levels produced along the trajectory and the output will be transferred to the optimization model or will directly be sent to the visualization component. Additionally, the stakeholder will deliver us academic papers and extra explanation on the mathematical concepts behind the optimization model and contouring algorithm.

**Product requirements** Our product should be able to do the following tasks:

- The program can read in an arbitrary data file or text file specifying the input trajectory and grid
- The program can calculate (noise) contours over time
- The program can output and save the calculated contour values as an arbitrary data file
- The program is capable of iterative trajectory optimization based on noise contour areas and population annoyance of areas that are affected
- The program is able to evaluate 2500 trajectories for optimization in under 30 minutes
- The program can visualize trajectories in a 3D animation in Google Earth
- The program can read in KML files and directly visualize it in Google Earth
- The program can calculate and visualize the effects of produced noise on population annoyance using the awakenings algorithm
- The program should be compatible with input from other (noise) calculation tools
- The program should show the estimated execution time and current progress during trajectory optimization
- The program should execute all processes in an automated and pipelined manner
- The program should offer all these tasks in a graphical user interface

### C.2b. Roadmap

In the roadmap below major releases of the program are spread across different phases of software development and they are given in a clear overview.

Sprint	Week(s)	Main Activities	Focus
1	1 - 2	Orientation in subject matter, client interviews, requirements gathering	Orientation, Research
	2	- Set-up AppVeyor + CoverAlls - Analysis of the context delivered in Research Report + Product Planning - Analysis of the design delivered in Emergent Architecture	Orientation, Design
2	3	Contouring calculation + GE overlay, basic aircraft animation	Development
3	4	Basic implementation trajectory optimization (point-mass calculation)	Development
4	5	3D visualization of contours	Development
5	6	2D visualization of multiple trajectories, speed-up optimization	Development
6	7	First version GUI, calculation population annoyance	Development
7	8	Full implementation trajectory optimization (operational constraints), Final version GUI, Visualization population annoyance	Development
8	9	Write final report and prepare end presentation	Evaluation

## C.3. Release Plan

This chapter shows an initial release plan with milestones. For a detailed overview of the user stories describing the features of our program we refer you to section 3c of our research report.

### C.3a. Initial release plan (milestones, MRFs per release)

The milestones are spread across different sprints and described below with the corresponding goals. All the releases are continually tested with system testing throughout the entire project. Please note that a sprint corresponds with one week.

#### Sprint 1: 18/04/2016 - 24/04/2016

This release will contain at least the following features:

- A Product Planning document containing a roadmap, overall planning and user stories that are prioritized in consultation with the client.
- A set-up of the Emergent Architecture document containing a pipeline diagram representing the workflow of our program and the way in which the noise, optimization and visualization models are connected.

#### Sprint 2: 25/04/2016 - 01/05/2016

This release will contain at least the following features:

- A Research Report in which the problem, context and possible solutions are analysed. This will be discussed with the client.
- A trajectory and grid can be read in as an arbitrary data file
- An implementation of the algorithm that converts Rijksdriehoekscoördinaten to WGL coordinates (long/lat)
- Basic visualization of noise contours with an overlay in Google Earth

#### Sprint 3: 02/05/2016 - 08/05/2016

This release will contain at least the following features:

- An implementation of the contouring algorithm (refining the grid, finding switch points and clustering points with a similar noise level)
- Visualization of the noise contours plotted/ mapped in Google Earth
- An implementation of the spline interpolation algorithm to smoothen out the contour lines
- The option to output actual noise data
- The option to turn on or off particular noise contours in the visualization

#### Sprint 4: 09/05/2016 - 15/05/2016

This release will contain at least the following features:

- Basic implementation of the trajectory optimization model (point-mass calculation)

#### Sprint 5: 16/05/2016 - 22/05/2016

This release will contain at least the following features:

- Visualization of the input trajectory and noise contours in a 3D real-time animation in Google Earth (link all visualization components together)
- Generated visualizations can be saved as a KML file

#### Sprint 6: 23/05/2016 - 29/05/2016

This release will contain at least the following features:

- 2D contour visualization of an entire trajectory in Google Earth
- The estimated calculation time and current progress are shown during trajectory optimization

**Sprint 7: 30/05/2016 - 05/06/2016**

This release will contain at least the following features:

- A shell script in which all operations are pipelined and performed in an automated manner
- First version of the GUI containing a menu bar and tabs for the import of files, noise model (with raw noise level data as output) and optimization model.
- An implementation of the Awakenings algorithm to calculate population annoyance
- Emergent Architecture document presenting the final state of the architecture design.

**Sprint 8: 06/06/2016 - 12/06/2016**

This release will contain at least the following features:

- Full implementation of the trajectory optimization model (added: operational constraints) using GeneticSharp
- Make the trajectory optimization model multi-core and multi-threaded
- The option to insert and visualize multiple trajectories with 2D animation and static visualization in Google Earth
- Final version of the GUI representing the optimization and visualization pipelines
- A visualization of population annoyance in Google Earth (linked with the animation)
- Population density is shown with a heat map in Google Earth

**Sprint 9: 13/06/2016 - 19/06/2016**

This release will contain at least the following features:

- Solved bugs and other problems from the previous sprint(s).
- Final product containing at least all must-have and should-have requirements
- Final report about the developed, implemented, and validated software product

### C.3b. Definition of Done

This chapter describes the definition of done so that we both will have the same end goals. This will enable us to know when a feature is done and when the corresponding sprint item can be closed.

Our definition of done focuses on three levels: backlog items (features), sprints and releases.

**4.1 Level 1: Backlog items** We consider a backlog item as done when it has an test coverage of at least 75% for non-GUI elements. These tests can be divided into unit tests and other automated tests. For a feature to be merged (using a pull request) into the release version of the product, it has to be approved by the other team member. Their approval will be based on the test coverage, code readability, documentation and the overall code quality. Documentation should be added on every class and method.

#### 4.2 Level 2: Sprints

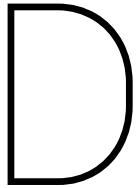
Every sprint corresponds to one week. At the end of a sprint, all the items on the sprint plan of the current sprint have to be finished, according to the definition of done for backlog items. A new release of our product should be submitted to the version control server (Github) master. All the unit tests should pass and the system should be improved based on added or extended features. There shouldn't be any bugs/errors left in the system and the program should behave and look like the client wanted. We should also have written a sprint reflection for that sprint and a new sprint plan for the coming sprint.

#### 4.3 Level 3: Releases

A release version of our product is done at the end of a sprint. The release version should contain the features that should have been implemented for the corresponding sprint. This also includes the minimal test coverage and other conditions described in the definition of done for a sprint. It is also important to note that the potential feedback of the client for that week should be processed before a release is completed. In the final release we should have implemented all must haves since the program can't function without these mandatory features. Most should haves (50 to 60%) and some could haves (30 to 40%), which are defined in 3.1, have to be implemented. As explained in chapter 3, these features are not necessary for the user and the system to work but our goal is to implement them partially since it would be nice to include them. It would make our program more user friendly. Lastly, the final release and other major releases should be approved by the client, the coach and obviously by ourselves too. This means that it should be simple and efficient to use for the client and well documented and designed by us. The SIG test is also an important part of this. They will determine if the code meets the standards and if it is clearly structured and documented. After completion of the project an executable will be delivered to run the program. Next to this, we will also offer an installer that creates a simple MSI package containing files such as example input files and a user guide to help the user with a quick project set-up.

As developers we also strive for maintainability and extendability, so that the system can be easily maintained, improved and updated by us or other people after our final release. This will also be taken into account throughout the entire project.





## Original Project Description

Aircraft and airport noise are complex subject matters which have been studied for decades and are still the focus of many research efforts nowadays. Also at the department of Air Transport & Operations at TU Delft's Faculty of Aerospace Engineering.

The project team will be assigned to implement a program in C++ for the prediction and visualization of aircraft noise. More specifically, this will involve:

- a) The implementation of a mathematical model for aircraft noise and aircraft trajectory design. The model will be deployed by the project team (and later on by our research team) to predict aircraft noise along a particular trajectory (flight route) in order to be able to optimize the trajectory and to reduce the produced noise. In order to implement this model the parameters that are required for the sound calculation need to be derived mathematically.
- b) The code should be optimized with multi-core processing to achieve real-time performance.
- c) Besides that, a tool should be developed to visualize the noise produced along the simulated trajectory pictured on a real map. This requires an implementation of noise contours, which are 'noise footprints' whose shape indicate areas of constant noise. Noise contours are a new subject to our research group and haven't been implemented before so this will be a challenging and novel topic.

The resulting program will present a creative and efficient way to compute and visualize aircraft noise along simulated and real flight routes.

# AeroVision

**Title of the project:** Model-based Optimization and Visualization of Aircraft Trajectories

**Name of the client organization:** Air Transport & Operations (ATO) at TU Delft

**Presentation Date:** 1 July 2016

## Project Description

Nowadays, major airports are dealing with more than a thousand flights a day leading to considerable impact on the surrounding populated areas in terms of noise. Recent studies show that the most effective way of minimizing noise is the calculation of noise minimal approach routes. The department of Air Transport & Operations (ATO) at TU Delft's Faculty of Aerospace Engineering is a research group that focuses on the understanding of noise impacts. To understand flight navigation concepts, ATO needs a standalone application for model-based trajectory optimization and visualization of aircraft noise.

To remedy this issue, we designed and implemented *AeroVision*: an integrated solution to the computation, optimization and visualization of noise contours. This includes (a) a generalised model for contour calculation requiring lower computational effort as compared with the current tool used by ATO; (b) a new multi-objective approach to trajectory optimization based on contour areas and population annoyance; (c) an extensible visualization component for 2D and 3D animation in Google Earth.

## Challenge

The challenge set out by ATO is to build a program that represents an innovative and efficient way to minimize and visualize aircraft noise along simulated and real flight routes. There are no existing programs that offer both optimization and visualization of aircraft noise. Therefore, the research department would benefit greatly by an automated and pipelined execution of these processes.

## Research

The research phase focused on a noise model for contour calculation and a new multi-objective approach to trajectory optimization based on contour areas and population annoyance.

## Process

We developed AeroVision using modern software engineering practices. The agile development method Scrum was used. To guarantee code quality, AeroVision's source code is thoroughly tested with the help of unit testing and manual testing. Automated regression testing through the use of continuous integration was also applied.

## Product & Outlook

The final version of AeroVision meets the functionality and quality required by ATO. We implemented all requirements except one could-have requirement. For trajectory optimization we were able to achieve a performance improvement of 84%. This led to a major improvement on the workflow management and automation of ATO.

---

## Development Team

**Name:** Hans Schouten

**Interests:** Computer science, web design, aviation  
**Role & Contribution:** Trajectory optimization and GUI design

**Name:** Elvan Kula

**Interests:** Data science, algorithm design, mathematics and business

**Role & Contribution:** Trajectory visualization and input compatibility

## Project Coach

**Name:** N. Dintzner

**Affiliation:** n.j.r.dintzner@tudelft.nl

## Client

**Name:** S. Hartjes

**Affiliation:** s.hartjes@tudelft.nl

## Contacts

Contact person 1: Elvan Kula, e.y.kula@gmail.com

Contact person 2: Hans Schouten, hansschouten95@outlook.com

# Bibliography

- [1] Jim Aarons. Sourcemonitor version 3.5, 2014. URL <http://www.campwoodsw.com/sourcemonitor.html>.
- [2] Uri Agassi. Creating isolines (contours) from grid data, 2011. URL <http://uri.agassi.co/2011/10/creating-isolines-contours-from-grid.html>.
- [3] David Allerton. *Principles of flight simulation*. John Wiley & Sons, 2009.
- [4] AppVeyor. Continuous delivery service for windows, 2016. URL <https://www.appveyor.com/docs>.
- [5] Secretary ATO. Ato research, 2015. URL <http://www.lr.tudelft.nl/en/organisation/departments/control-and-operations/air-transport-and-operations/research-fields/>.
- [6] Secretary ATO. S. (sander) hartjes, 2015. URL <http://www.lr.tudelft.nl/organisatie/afdelingen/control-and-operations/section-air-transport-and-operations/people/staff/s-sander-hartjes/>.
- [7] M. Bittner. Why do we need trajectory optimization?, 2015. URL <http://www.fsd.mw.tum.de/research/trajectory-optimization/>.
- [8] Eric R Boeker, Eric Dinges, Bill He, Gregg Fleming, Christopher J Roof, Paul J Gerbi, Amanda S Rapoza, and Justin Hemann. Integrated noise model (inm) version 7.0 technical manual. Technical report, 2008.
- [9] Noppadol Chadil, Apirak Russameesawang, and Phongsak Keeratiwintakorn. Real-time tracking management system using gps, gprs and google earth. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on*, volume 1, pages 393–396. IEEE, 2008.
- [10] ITD Clean Sky. Systems for green operations.(nd). *NoiseLAss v2. 0 Noise Level Assessment Tool Specification and Validation*.
- [11] The Boeing Company. *Boeing 747-400 Performance Engineers Manual*. John Wiley & Sons, 1993.
- [12] Coveralls. Coveralls feature tour, 2016. URL <https://coveralls.io/features>.
- [13] Technische Universitat Darmstadt. Dircol, 2015. URL <http://www.sim.informatik.tu-darmstadt.de/en/res/sw/dircol/>.
- [14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.
- [15] Google Developers. Kml documentation introduction, 2016. URL <https://developers.google.com/kml/documentation>.
- [16] SpaceWorks Enterprises. Why flightsight?, 2013. URL <http://www.spaceworkssoftware.com/products/desktop/flightsight.shtml>.
- [17] Federal Aviation Administration (FAA). Aircraft noise issues, 2014. URL [https://www.faa.gov/about/office\\_org/headquarters\\_offices/apl/noise\\_emissions/airport\\_aircraft\\_noise\\_issues](https://www.faa.gov/about/office_org/headquarters_offices/apl/noise_emissions/airport_aircraft_noise_issues).
- [18] Scott Hanselman. Appveyor - a good continuous integration system is a joy to behold, 2014. URL <http://www.hanselman.com/blog/AppVeyorAGoodContinuousIntegrationSystemIsAJoyToBehold.aspx>.

- [19] Tom Henderson. Echo vs. reverberation, 2016. URL <http://www.physicsclassroom.com/mmedia/waves/er.cfm>.
- [20] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.
- [21] Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.
- [22] Geography Markup Language. Citygml, 2016. URL <http://www.opengeospatial.org/standards/gml>.
- [23] John Leslie. Weather data access, 2016. URL <http://www.ncdc.noaa.gov/data-access>.
- [24] Integrated Noise Model. Version 7.0 user's guide. Technical report, Report No. FAA-AEE-07-04, Dinges, et al., Washington, DC: Federal Aviation Administration, 2007.
- [25] N. Naderi. Comparing the mstest and nunit frameworks, 2007. URL <https://blogs.msdn.microsoft.com/nnaderi/2007/02/01/comparing-the-mstest-and-nunit-frameworks/>.
- [26] Microsoft Developer Network. Code metrics values, 2015. URL <https://msdn.microsoft.com/en-us/library/bb385914.aspx>.
- [27] Ariadi Nugroho, Joost Visser, and Tobias Kuipers. An empirical model of technical debt and interest. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, pages 1–8. ACM, 2011.
- [28] Software Improvement Group (SIG). Het verhaal van sig, 2014. URL <https://www.sig.eu/nl/>.
- [29] Robin R Sobotta, Heather E Campbell, and Beverly J Owens. Aviation noise and environmental justice: The barrio barrier. *Journal of Regional Science*, 47(1):125–154, 2007.
- [30] Open Geo Spatial. Citygml, 2016. URL <http://www.opengeospatial.org/standards/citygml>.
- [31] NUnit Team. What is nunit?, 2015. URL <http://www.nunit.org/>.
- [32] Matthijs Teengens. Model of the boeing 747-400. *CF6-80C2B1F Engines*.
- [33] Universitat Politchnica De Catalunya (UPC). Sustainable air transportation, 2014. URL <https://www.icarus.upc.edu/en/research/air-transportation>.
- [34] VirtualAir. Documentation virtualair, 2010. URL <http://virtualair.sourceforge.net/>.
- [35] Hendrikus G Visser and RA A. Wijnen. Optimization of noise abatement departure trajectories. *Journal of Aircraft*, 38(4):620–627, 2001.
- [36] Dipl Math Oskar Von Stryk. Numerical solution of optimal control problems by direct collocation. In *Optimal Control*, pages 129–143. Springer, 1993.
- [37] Navinda Kithmal Wickramasinghe, Akinori Harada, and Yoshikazu Miyazawa. Flight trajectory optimization for an efficient air transportation system. *ICAS2012, Brisbane*, 2012.
- [38] Deniz Yuret and Michael De La Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 208–212. Citeseer, 1993.