High Performance Conflict Detection and Resolution for Multi-Dimensional Objects

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des Grades
Doktor der Naturwissenschaften
Dr. rer. nat.
genehmigte Dissertation von

Dipl.-Inform. Alexander Kuenz
geboren am 08.09.1974 in Bad Harzburg

2015

1. Referent	Prof. Dr. Franz-Erich Wolter
2. Referent	Prof. Dr. Dirk Kügler
3. Referent	Prof. Dr. Gabriel Zachmann
Tag der Promotion	20.07.2015

Preliminary Publications

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

Patents

Kuenz, A. and N. Peinecke (2011, February). Effiziente 4D-Konflikt-Erkennung für großräumige Szenarien, Verfahren zur Ermittlung einer potenziellen Konfliktsituation. Patent (EP 2 457 224 A2).

Kuenz, A. and N. Peinecke (2012, June). Method for determining a potential conflict situation. Patent (US 20120158278).

Publications

Kuenz, A. (2014). Increasing the margins - more freedom in trajectory-based operations. In *Proc. IEEE/AIAA 33rd Digital Avionics Systems Conf. DASC 2014.* (Best Paper of Session)

Kuenz, A., G. Schwoch, and F.-E. Wolter (2013, October). Individualism in global airspace - user preferred trajectories in future ATM. In *Proc. IEEE/AIAA 32nd Digital Avionics Systems Conf. DASC 2013.* (Best Paper of Track)

Kuenz, A. and G. Schwoch (2012, October). Global time-based conflict solution: towards the overall optimum. In *Proc. IEEE/AIAA 31st Digital Avionics Systems Conf. DASC 2012.*

Kuenz, A. (2012, June). Optimizing tomorrows ATM using 4D-trajectory-based operations. In *ODAS 2012.*

Kuenz, A. (2011, October). A global airspace model for 4D-trajectory-based operations. In *Proc. IEEE/AIAA 30th Digital Avionics Systems Conf. DASC 2011.* (Best Paper of Session)

Kuenz, A. and N. Peinecke (2009, October). Tiling the world - efficient 4D conflict detection for large scale scenarios. In *Proc. IEEE/AIAA 28th Digital Avionics Systems Conf. DASC 2009.*

Abstract

4D-trajectory based operations is one of the big enabler for future high-capacity, efficient and environmentally friendly air traffic management. Every aircraft is scheduled to fly along a predicted 4D path that can be calculated from gate to gate pre-flight. 4D-trajectories are optimized individually for aircraft taking into account performance models, routes, weather conditions and airline preferences. However, individual calculation of trajectories does not ensure conflict-freeness with surrounding traffic. This work describes an efficient algorithm detecting conflicts for large traffic scenarios. Conflict detection is performed between aircraft trajectories, also taking into account environmental constraints like severe weather zones and restricted areas. Basic idea is an N-dimensional bisection of airspace allowing a significant reduction of complexity. Thus, potential conflicts are identified very fast. A slower high precision conflict check is performed on potential conflicts only. On average, conflicts of one 4D-trajectory can be detected in a European traffic sample holding more than 33 000 flights in less than 2.5 ms on standard PC hardware.
Fast detection times are predestined for trial-and-error conflict resolution. Different conflict resolution methods are illustrated, taking into account the major key performance areas in air traffic management, e. g., safety, efficiency, and predictability. As an example, deconfliction is performed on an optimized version of aforementioned European traffic sample holding 33 000 flights.

Keywords: Conflict Detection and Resolution, 4D-Trajectory-Based Operations, N-dimensional Bisection

Zusammenfassung

Ein wichtiger Bestandteil kapazitätserhöhender, effizienter und umweltfreundlicher Zukunftskonzepte im Luftverkehrsmanagement sind 4D-Trajektorien. Jedes Luftfahrzeug fliegt in einem derartigen Szenario entlang einer 4D-Flugbahn, die bereits vor dem Start für den gesamten Flugweg berechnet werden kann. Diese Flugbahnen werden von Fluggesellschaften individuell für Flugzeugmuster, Routen und Wetterbedingungen optimiert. In einem gemeinsamen Luftraum sind individuell berechnete Trajektorien jedoch in der Regel nicht konfliktfrei. Diese Arbeit beschreibt einen effizienten Algorithmus zur Erkennung von Konflikten in sehr großen Szenarien. Die Konfliktidentifizierung erkennt Annäherungen zwischen Verkehrsteilnehmern sowie Verletzungen von Beschränkungsgebieten wie Schlechtwetter- und Flugverbotszonen. Die Grundidee ist eine N-dimensionale Bisektion des Luftraums zur signifikanten Reduktion der Gesamtkomplexität. Durch dieses Verfahren werden potenzielle Konflikte sehr schnell identifiziert. Ein präziser und langsamerer Algorithmus zur finalen Entscheidung wird lediglich auf den zuvor identifizierten potenziellen Konflikten durchgeführt. In einem Europäischen Verkehrsszenario mit mehr als 33 000 Flugzeugen können mit Hilfe des Algorithmus alle Konflikte einer 4D-Trajektorie im Mittel in weniger als 2.5 ms auf Standard-PC-Hardware identifiziert werden.
Die geringen Antwortzeiten erlauben, dass über intelligente Versuch-und-Irrtum-Verfahren effizient Konfliktlösungsstrategien umgesetzt werden können. Für den 4D-Flugverkehr werden Lösungsstrategien aufgezeigt, die neben der Konfliktfreiheit noch andere Faktoren wie Effizienz, Emissionsvermeidung und Planbarkeit berücksichtigen. Beispielhaft werden Lösungsalgorithmen auf eine optimierte Variante des zuvorgenannten Europäischen Verkehrsszenarios mit 33 000 Flügen angewendet.

Schlagworte: Konflikterkennung und -lösung, 4D-trajektorienbasierte Konzepte, N-dimensionale Bisektion

Acknowledgements

This work would not have been possible without the support of several people, to whom I am greatly indebted.

First of all, I like to thank Prof. Dr. Franz-Erich Wolter for accepting to supervise my thesis. He always gave me good advice and support.

I am also grateful to my head of institute Prof. Dr. Dirk Kügler for taking over co-advisorship. Furthermore, he and my head of department, Dr. Bernd Korn, provided me with a great working environment at DLR's Institute of Flight Guidance and tolerated my research on conflict detection and avoidance even without directly feeding into a project from the beginning.

I also would like to thank Prof. Dr. Gabriel Zachmann for his kind willingness to be third accessor.

At DLR, I had (and still have) very nice fellows supporting the completion of my thesis. Without claim to be complete, I thank Dr. Niklas Peinecke for co-authoring my patent on conflict detection and initiating the link to my supervisor, Christiane Edinger for developing the highly sophisticated Advanced Flight Management System I used as trajectory prediction engine, Gunnar Schwoch for preparing optimized traffic scenarios, and Ralf Kohrs and Uwe Teegen for proof-reading. More generally, I thank all participants of the coffee breaks of department FL23 for endless, all-embracing, and very fruitful discussions on many details of this work. I am grateful to everyone using the *NDMap*-implementation, extending the field of application, pointing me to problems, and requesting interesting new features.

Last but not least, my thanks go to my wife Anja and my children Leonard and Annelie for accepting countless working hours in the evenings and weekends for writing this thesis.

Contents

Acronyms

4DCo-GC	4 Dimensional-Contracts - Guidance and Control
AABB	Axes Aligned Bounding Box
ACAS	Airborne Collision Avoidance System
ADS-B	Automatic Dependent Surveillance-Broadcast
AFMS	Advanced Flight Management System
ANS	Air Navigation Service
AOC	Airline Operation Center
Arr	Arrival
ATC	Air Traffic Control
ATCo	Air Traffic Controller
ATM	Air Traffic Management
ATRA	Advanced Technology Research Aircraft
ATTAS	Advanced Technologies Testing Aircraft System
BADA	Base of Aircraft DAta
BSP	Binary space partitioning
BVH	Bounding Volume Hierarchy
CAS	Calibrated Air Speed
CDA	Continuous Descent Approach
CFMU	Central Flow Management Unit

CGAL	Computational Geometry Algorithms Library
Clb	Climb
CPA	Closest Point of Approach
CPU	Central Processing Unit
Crs	Cruise
CTAS	Center-TRACON Automation System
DDR	Demand Data Repository
Dep	Departure
DFS	DFS Deutsche Flugsicherung GmbH
DLR	Deutsches Zentrum fuer Luft- und Raumfahrt (German Aerospace Center)
dops	Discrete Orientation Polytopes
Dsc	Descent
DWD	Deutscher Wetterdienst
E-TMA	Extended TMA
EFR	Extended Flight Rules
ENU	East-North-Up
ERAT	Environmentally Responsible Air Transport
EUROCONTROL	European Organisation for the Safety of Air Navigation
FACTS	Future Aeronautical Communications Traffic Simulator
FAGI	Future Air Ground Integration
FL	Flight Level
FMS	Flight Management System
FP7	Seventh Framework Programme for Research
ft	feet
Glonass	Globalnaja Nawigazionnaja Sputnikowaja Sistema
GPS	Global Positioning System
ICAO	International Civil Aviation Organization
IFR	Instrument Flight Rules
k-d tree	*k*-dimensional tree

KDS	Kinetic Data Structure
KML	Keyhole Markup Language
KPA	Key Performance Areas
kts	knots
LAnAb	Leise An- und Abflüge
LDLP	Low Drag Low Power
LRM2020	Luftraummanagment 2020
MTCD	Medium Term Conflict Detection
NDMap	N-Dimensional Map-Implementation
NextGen	Next Generation
NM	Nautical Miles
PHARE	Programme for Harmonised Air Traffic Management Research in EUROCONTROL
SCDA	Segmented Continuous Descent Approach
SESAR	Single European Sky ATM Research
SLD	Supercooled Large Droplet
SSR	Secondary Surveillance Radar
STCA	Short Term Conflict Alert
SuLaDI	Supercooled Large Droplets Icing
TBO	Trajectory Based Operations
TMA	Terminal Maneuvering Area
TOD	Top Of Descent
TRACON	Terminal Radar Approach Control
VFW	Vereinigte Flugtechnische Werke
VLSI	Very-large-scale integration
VolcATS	Volcanic ash impact on the Air Transport System
VR	Virtual Reality
VTS	Vertical Tile Size
WGS84	World Geodetic System 1984

ZFB	Zentrum für Flugsimulation Berlin

Symbols

A	N-dimensional tile
$\vec{a}_{\max}$	Maximum tile values for each dimension N
$\vec{a}_{\min}$	Minimum tile values for each dimension N
a	Equatorial radius of Earth ellipsoid
B_O	Axis aligned Bounding Box of Object O
b	Polar distance from center of Earth ellipsoid
$\vec{D}$	Number of necessary subdivisions in $\mathbb{R}^N$
f	Flattening of Earth ellipsoid
$\mathfrak{G}$	Minimum gap time between two conflicts
λ	Longitude
$L(\vec{p}_1, \vec{p}_2)$	Line segment from $\vec{p}_1$ to $\vec{p}_2$
N	Dimension in $\mathbb{N}$
O	Object. Either trajectory or volume
$\mathfrak{P}$	2D-Polygon
φ	Latitude
$\vec{p}$	Point in $\mathbb{R}^N$
$\vec{S}$	Mandatory separation in $\mathbb{R}^N$
$\mathfrak{S}$	Simplex in $\mathbb{R}^N$
T	Trajectory object
τ	Common time
t_i	Trajectory instances
$\mathfrak{V}$	Polygon Volume, implementation of volume object

V	Volume object

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The two major Air Traffic Management (ATM) initiatives *Single European Sky ATM Research (SESAR)* in Europe and *Next Generation (NextGen)* in the United States foresee drastic changes in ATM already for the year 2020 (SESAR Joint Undertaking, 2013; Federal Aviation Administration, 2013). According to the SESAR Consortium (2008), performance targets for the year 2020 compared to a year 2005 reference are (amongst others):

- A 73 % increase of Instrument Flight Rules (IFR) flights in Europe to a total of 16 million annual flights.
- A 50 % decrease of en-route and terminal Air Navigation Service (ANS)-cost in Europe per flight.
- A minimum of 98 % scheduled flights departing on time with an average departure delay of less than 10 min for the remaining flights.
- At least 95 % flights arriving on time with an average arrival delay of less than 10 min for the remaining flights.
- A 3 times increased safety level per flight.

- 10 % fuel savings per flight on average due to ATM improvements.

One key element of SESAR and NextGen to reach these ambitious goals is 4D-Trajectory Based Operations (TBO) (SESAR Consortium, 2007, 2010; Federal Aviation Administration, 2012).

In a 4D-TBO environment every aircraft flies along a predicted *4D-trajectory*, describing the flight using 3D-positions (latitude, longitude and altitude) referenced by time. The expected benefits are:

- Predictability of trajectories in advance allowing early planning of operations, e. g., conflict detection and avoidance, high precision flow control and arrival sequence planning.
- Safety benefiting from well-known future positions of aircraft for each moment in time.
- Improved cost efficiency and less environmental impact by optimizing routing, vertical profiles, and fuel burn for single aircraft and the global traffic situation.

4D-TBO enables a paradigm shift from tactical to strategical Air Traffic Control (ATC). Instead of adjusting flights on severe weather, crossing traffic, and other upcoming events just in time, these issues are supposed to be respected pre-emptively. Assuming proper forecasts for aforementioned events, overall flight performance can be improved significantly by performing more efficient avoidance maneuvers.

However, allowing each aircraft to fly its personal optimum profile does not work in a global traffic scenario. Conflicts would occur with surrounding traffic. Globalization of ATM necessitates conflict freeness for large airspaces, e. g., for the whole of Europe as requested by SESAR. Detection and resolution of aforementioned conflicts is the central topic of this work.

Even though Airline Operation Centers (AOCs) usually respect issues like weather phenomena and restricted flight areas already when providing optimized profiles it is beneficial to integrate conflict detection also for these environmental constraints in order to model the whole global picture. Thus, a conflict resolution algorithm does not violate environmental constraints when solving traffic-based conflicts.

Conflicts in aviation are usually defined as a violation of lateral and vertical separation criteria. Typical stipulated lateral separations are 5 Nautical Miles (NM) for en-route airspace, and 3 NM in the Terminal Radar Approach Control (TRACON) area (ICAO, 2007). Vertically, a separation of 1000 feet (ft) is usually required. Several more specific rules exist, e. g., for in-trail flights due to wake turbulence issues and independently operated parallel runways.

Compliance to separation metrics can either be validated by monitoring the horizontal and vertical distance between each pair of aircraft for every time, or assigning a separation cylinder to each aircraft representing the safety area that must not intersect with any other safety cylinder (fig. 1.1).

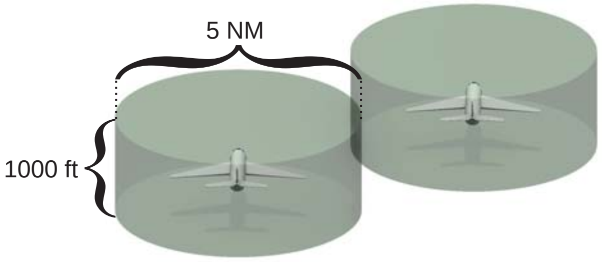


Figure 1.1 – En-Route Metrics for Airborne Conflicts

Applying 4D-TBO, a future loss of separation between aircraft can be predicted well in advance based on their 4D trajectories. When trajectories come too close for a common future time, the trajectories are in conflict. Especially for large traffic scenarios with many participants conflict detection between all trajectories is a computational complex task. Using basic algorithms conflict detection for one day of German air traffic can last several hours (section 4.3).

Based on a list of conflicts from the conflict detection algorithm, conflict resolution can be applied to solve conflicts strategically. Whatever conflict resolution strategies are used, the solutions should be validated to really solve the conflict and not create new conflicts further downstream using the conflict detection algorithm again.

This work focuses on an efficient conflict detection algorithm and its impact on conflict resolution. The document is structured as follows:

- Chapter 2 gives background information about search algorithms, 4D trajectories, conflict detection and conflict resolution algorithms.
- Chapter 3 describes the implemented algorithm facilitating conflict detection for an arbitrary number of dimensions.
- Chapter 4 illustrates how the generic algorithm from chapter 3 is adapted, configured and optimized for conflict detection in the aviation domain using 4 dimensions. Results from conflict detection runs are presented for a German and a European traffic sample.
- Chapter 5 focuses on conflict resolution using a trial-and-error method taking advantage of the high performance conflict detection algorithm. The traffic sample covers one modified day of air traffic in Europe containing most direct routes from departure to destination.
- Chapter 6 describes the validation of the product.
- A summary and outlook of the thesis is given in chapter 7.

Chapter 2

Related Work

The chapter provides an overview about basic topics addressed by the thesis. Section 2.1 focuses on different data structures that are related to the tree structure of the presented conflict detection algorithm. An introduction on general conflict detection techniques is given in section 2.2. Especially the conflict resolution depends on performance limitations of aircraft. Section 2.3 provides background details on the generation of aircraft trajectories.

Different algorithms performing conflict detection in aviation are summarized in section 2.4. Already existing conflict resolution techniques are described in section 2.5. Section 2.6 explains different geodetic Earth models and map projections in order to let aircraft fly shortest route on Earth.

2.1 Mathematics and Algorithms

This section summarizes fundamentals from mathematics and algorithms that are related to this work. The algorithm described is based on N-dimensional bisection in order to reduce the problem's complexity.

2.1.1 Bisection

Bisection is a method in mathematics and computer science that repeatedly cuts intervals into two parts. The method is used where a complex problem can be divided into two smaller problems (Lewis et al., 1981).

A common use case is, for example, searching a number in a sorted array. The number is compared to the central element of the array. If it is bigger, the same is done within the upper interval, otherwise in the lower interval until the number is found.

The complexity of the bisection method is $\mathcal{O}(\log N)$, with N being the number of elements. Bisection can be performed using a binary tree structure.

2.1.2 Binary Tree

A binary tree is a tree data structure representing the bisection method. Each node has at most two children. Depending on the use case, the subdivision is data dependent or predefined by the initial problem (region tree), where each node contains the data elements corresponding to the sub-region (section 2.1.3).

2.1.3 Binary Space Partitioning Tree

Binary space partitioning (BSP) is a class of trees subdividing a space into convex subsets recursively. The subdivision is usually done along hyperplanes. Famous representatives of BSP trees are quadtrees (2D) and octrees (3D). A quadtree represents a partition of space in two dimensions. The root tile contains the entire starting region. Each node is subdivided in four children until reaching the leaves. While quadtrees are used for segmentation of 2D-maps, octrees fulfill the identical task within 3D-space. Nodes of an octree have 8 children generated by 3 subdividing planes, with exception of the leaves. Dworkin and Zeltzer (1993) propose to use a hex-tree in order to represent dynamic motion of objects in a static way. The conflict detection algorithm described in chapter 3 uses an N-dimensional BSP tree.

Even if quadtree and octree usually have their cutting lines/-planes aligned with the coordinate axes, the hyperplanes which partition the scene may have an arbitrary orientation. Figure 2.1 shows an example containing both the distribution of 9 points p_i on a 2D-plane and the corresponding quad tree based on the generated quadrants Q. An octree is used in fig. 2.18, page 65.

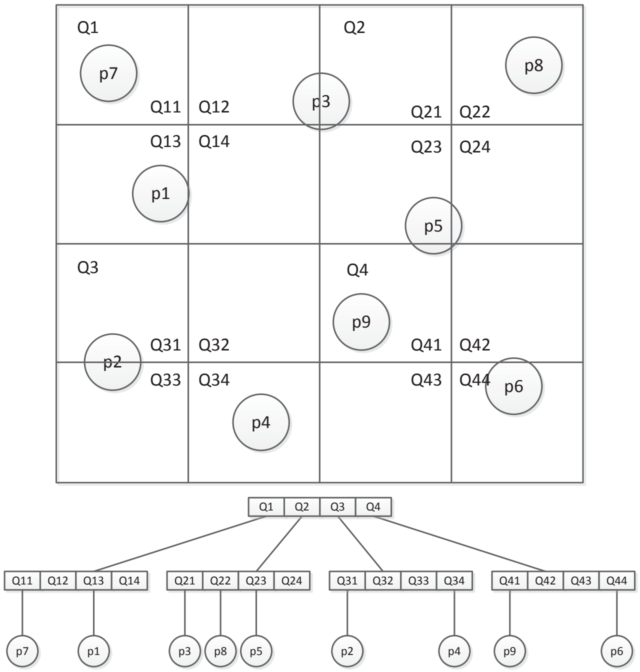


Figure 2.1 – Quad Tree Example

Initially, BSP was developed for simplifying the render process in 3D computer graphics (Schumacker et al., 1969; Fuchs et al., 1980; Naylor, 1993). Types of application in computer graphics are:

- Drawing objects in order of distance from the viewer from back to front. Even on today's z-buffer supporting hardware sorting is still necessary when drawing transparent objects (Kelly et al., 1994).

- Cutting complex objects into primitives easier to handle.

Other applications of BSP trees are collision detection, ray tracing and other calculations on complex spatial scenes. A special k-dimensional BSP tree is the k-d-tree.

2.1.4 k-dimensional Tree

The k-dimensional tree (k-d tree) is a BSP for organizing points in k-dimensional space described by Bentley (1975). Every non-leaf node has two children. Thus, a k-d tree splits only once per level along a hyperplane. Splitting a k-d tree once in every of the k dimensions creates a tree of depth k. k-d trees are not necessarily balanced. The splitting hyperplanes are not necessarily at the center or median point of the interval.

Figure 2.2 shows an example containing both the distribution of 9 points on a 2D-plane and the corresponding k-d tree.

Wald and Havran (2006) describe how to use k-d trees for ray tracing. They provide an algorithm to build the tree in $\mathcal{O}(N \log N)$. Instead of storing points, there are variations of the k-d tree working on volumetric objects (Rosenberg, 1985; Houthuys, 1987).

In contrast, the algorithm proposed in chapter 3 divides up to N dimensions in each level. A dimension is omitted for splitting only if the minimum tile size would be violated by the subdivision in that dimension. Splitting hyperplanes are always aligned to axis, and always split a dimension in the center of the interval. The simplicity of the structure allows very fast access to tree tiles utilizing small portions of memory, only.

2.1.5 R-Tree

Another tree structure providing spatial access methods is the R-tree proposed by Guttman (1984). Manolopoulos et al. (2006) describe various types of R-trees and their applications. Key idea of

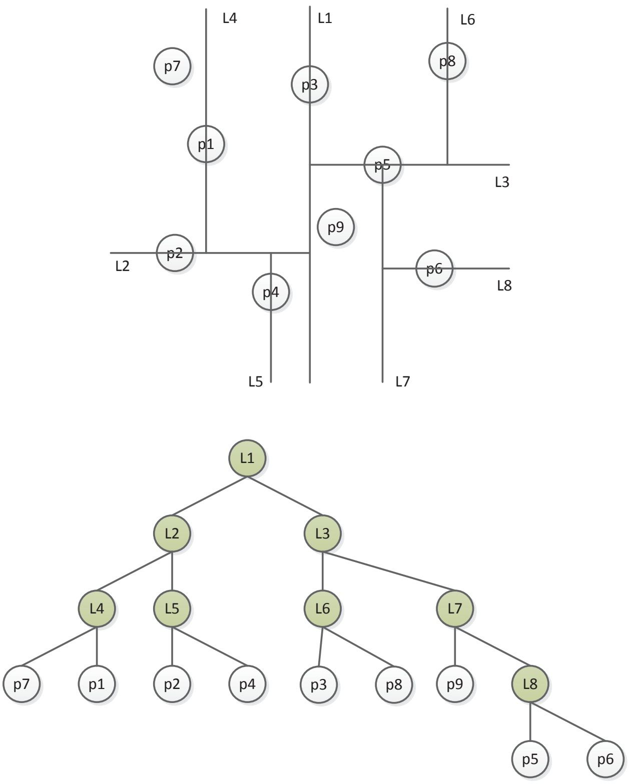


Figure 2.2 – k-d Tree Example

an R-tree is to group closely spaced objects and represent them as a minimum bounding rectangle object in the next higher level of the tree. Each group has a predefined maximum number of entries. A minimum fill is usually defined as a percentage of maximum number. Figure 2.3 shows an example containing both the distribution of 9 points on a 2D-plane and the corresponding R-tree with a maximum of three entries.

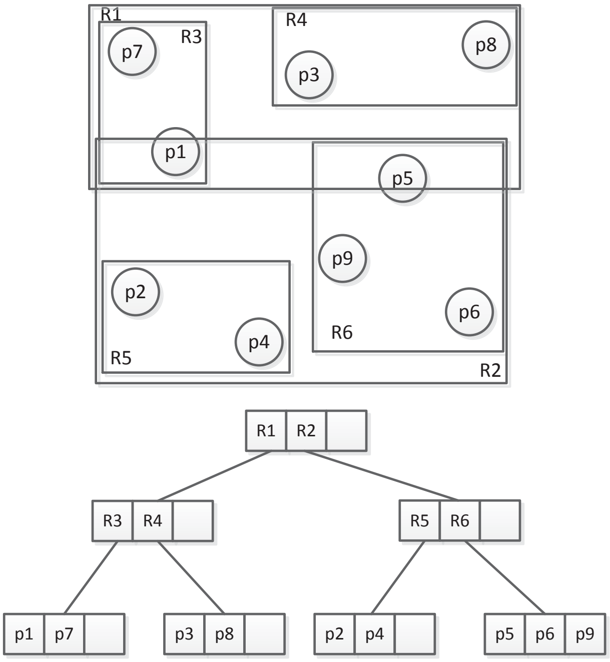


Figure 2.3 – R-Tree Example

This tree structure is especially efficient for finding the k nearest neighbor using a spatial join. A variation of the R-tree is the R*-tree storing points and volumetric objects. Beckmann et al. (1990) describe how to access points and rectangles efficiently using an R*-tree.

2.2 Conflict Detection Mechanisms

Conflict detection mechanisms are applied in many application fields as wire and component layout in Very-large-scale integration

(VLSI), motion planning in robotics, solid modeling, ray tracing, Virtual Reality (VR), and many more. Common goal is usually to detect conflicts in an efficient, accurate and robust way. A good overview on conflict detection is provided by Mount (1997).

Identifying conflicts between two geometric objects and the computation of the intersection region depends strongly on the type of objects. This section describes general techniques for collision detection.

2.2.1 Multiple Phases for Complex Scenarios

Scenarios for conflict detection may become very complex. Since a high precision conflict detection between objects often is computational expensive due to high level of detail and complex metrics, a division into two phases may be beneficial:

- The *broad phase* identifies potential conflicts only. This phase usually works on low detail data (e. g., bounding boxes instead of high detail objects) and omits as many object pairs as possible from the second phase. A good broad phase generates a low number of false-positives (i. e., a potential collision was identified, but it turns out to be a near miss) while ensuring that it does not produce any false-negatives (i. e., existing collisions are not identified as potential ones).
- The *narrow phase* performs the final collision detection with high accuracy. Often, the broad phase provides, in addition to the two objects, also information on where these objects may intersect.

If the range of detail is large, additional phases in-between may be beneficial.

2.2.2 Discrete and Continuous Motion

Collision detection can be performed for static scenarios and scenarios containing moving objects. The motion can be taken into account in two ways:

- In case of *discrete motion*, all objects are moved to their positions for a common specific time. This global time is increased with constant or adapted steps. Collision detection is performed on each predicted frame. The time step size needs to be chosen with care. Too small time steps increase computational effort, while too big time steps increase the probability of missing collisions. The optimum time step size depends on objects' shapes and speeds. A chosen step size Δ_t ensures that all conflicts with a minimum duration of Δ_t can be found. Detection of collisions with durations less than Δ_t cannot be guaranteed. The time step size for reactive real time simulations is trivial and depends on the achievable collision prediction rate. Even for real time simulations, a continuous motion simulation is reasonable, because it guarantees to detect all collisions with their accurate time.

- *Continuous motion* avoids the discretization of time. Collision detection is performed on positions depending on the additional dimension t. Since calculation of collisions is much more complex without discretization of time, many methods simulate motion discretely. However, especially for small, fine-grained objects that move fast, an adequate selection of the time step size is difficult.

2.2.3 Convex Polygon Intersection

The convex polygon intersection is often used in the narrow phase. Intersections between two convex polygons can be detected in logarithmic time $\mathcal{O}(\log N)$. An algorithm with this complexity was proposed by Dobkin and Kirkpatrick (1983): Assuming that both polygons are given as a list of vertices in counterclockwise order, first the lowest and highest y-coordinates are determined for polygons P and Q. Using a variant of binary search (compare section 2.1.1) this can be performed in $\mathcal{O}(\log N)$. Polygons P and Q are then split into two convex chains P_L, P_R and Q_L, Q_R at lowest and highest y coordinates. Semi- infinite rays are attached to beginning and end of each chain. For right oriented chains, rays run parallel to the x-axis towards $+\infty$, for left oriented towards $-\infty$. P and Q intersect if and only if P_L intersects Q_R and P_R intersects Q_L.

Figure 2.4 reveals how P is split into P_L and P_R. Furthermore it depicts how the median edge lines of P_L and Q_R intersect. If the intersection point is on both edges, the intersection is already identified. Otherwise, the algorithm distinguishes between an empty region that is untouched by both vertex chains, and the LR region, that can be reached according to the convexity assumption by both chains. Depending on the geometry of intersection, half of at least one vertex chain can be eliminated, marked in orange (fig. 2.5).

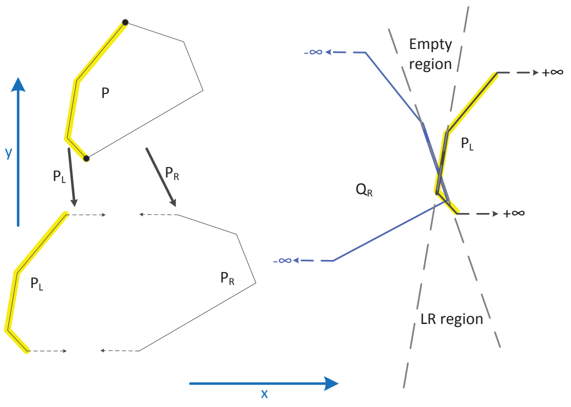


Figure 2.4 – Intersection Calculation between two Convex Polygons

2.2.4 Simple Polygon Intersection

Without convexity assumption, conflict detection becomes more complex. First of all an algorithm is required providing a decision on the simplicity of a given polygon. A polygon is simple if the vertex chain is not self-intersecting. A common approach to check for self-intersection is trying to triangulate the polygon. If the triangulation process fails, self-intersection is a possible reason. In particular, self-

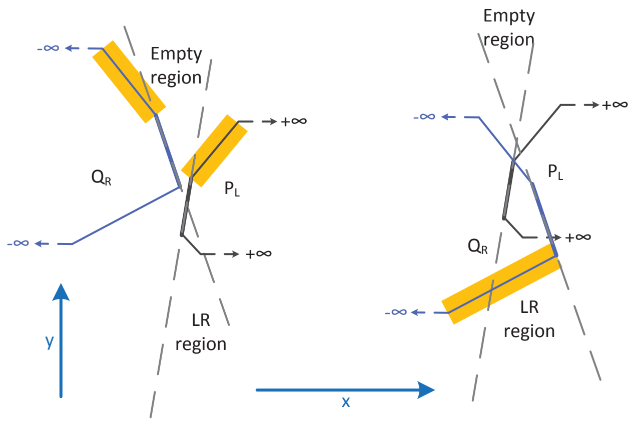


Figure 2.5 – Eliminating Half of at least one Vertex Chain

intersection can be determined in $\mathcal{O}(N)$ using a modified version of the linear-time triangulation algorithm (Chazelle, 1991).

The same complexity can then be achieved for the intersection test of two simple polygons by merging both polygons using a narrow channel and determining self-intersection subsequently.

2.2.5 Plane Sweep Algorithms

Plane sweep is a class of algorithms detecting intersections between multiple objects with usually simple geometry. Thus, a set of N line segments can be tested for intersection in $\mathcal{O}(N \log N)$ (Shamos and Hoey, 1976). The plane sweep method can be considered as broad phase algorithm selecting potentially conflicting object pairs.

Plane sweep is based on simulating a left-to-right sweep on a plane using a vertical sweepline. While the sweepline goes from left to right, a list of segments intersecting the sweepline is maintained, sorted from bottom to top by intersection point. The idea is to reduce intersection calculations to consecutive pairs in this list instead of testing all $\mathcal{O}(N^2)$ pairs.

The sweepline is moved from left to right jumping from one event to the next. Events represent either line segment end points or intersections between two line segments. Events are stored in a priority queue sorted by their x-value.

Objects are usually classified as:

- Sleeping: object is not reached yet by the sweepline.
- Active: object is intersected by the sweepline.
- Dead: object is completely passed by the sweepline.

Using plane sweep all k intersections of N line segments can be reported in $\mathcal{O}((N+k)\log N)$ time (Bentley and Ottmann, 1979). Intersections between any pair of k convex N-gons can be identified in $\mathcal{O}(k \log k \log N)$ time (Reichling, 1988).

Plane sweep also performs well for not too complex problems in higher dimensions. Thus, it is possible to detect intersections between any pair of N spheres in 3D-space in $\mathcal{O}(N \log^2 N)$ time (Hopcroft et al., 1983).

2.2.6 Intersection and Range Searching

Intersection and range searching focuses on intersections between axis aligned rectangles. Thus, it can be applied for rectangular objects in the narrow phase, but also for Axes Aligned Bounding Box (AABB) broad phase calculations. According to Edelsbrunner and Maurer (1981) two axis aligned rectangles A and B intersect if:

- A contains the left bottom point of B; or
- The left border line of A intersects with the bottom border line of B; or
- The bottom border line of A intersects with the left border line of B; or
- B contains the left bottom point of A.

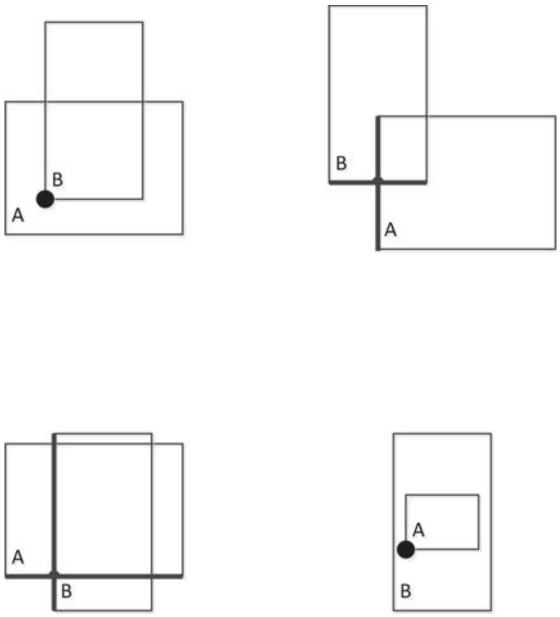


Figure 2.6 – Four Types of Intersection according to Edelsbrunner and Maurer (1981)

Figure 2.6 depicts the four types of intersection listed above.

Calculating all intersecting pairs from a set of N rectangles can be performed in $\mathcal{O}(N \log N)$ time for 2 dimensions (Chazelle, 1988). Generalization on higher dimensions using hyper rectangles is possible and adds an additional factor of $\log N$ in time for each additional dimension.

2.2.7 Bounding Volume Hierarchy

A bounding volume of a geometric object is a volume containing the whole object. Bounding volumes simplify collision detection especially for complex object geometries. If two bounding volumes do not intersect, the corresponding objects also do not intersect. Bounding volumes with simple geometric structure allow fast detection of intersections, but usually generate more false alarms because they cannot represent the contained object very tight. Different types of bounding volumes are commonly used:

- AABBs have a simple representation and can be calculated fast.
- Object oriented bounding volumes have a more complex representation, are more difficult to calculate, and containment check is slower.
- Spherical bounding volumes allow very fast check of intersection.
- $k-$Discrete Orientation Polytopes (dops) introduce a refinement of standard rectangular bounding volumes.

The idea of Bounding Volume Hierarchy (BVH) is building a tree holding the object's bounding volume in the root node. Nodes are subdivided into at least two parts by partitioning the geometric object into at least two subsets and calculating the corresponding bounding volumes. That way, the union of bounding volumes converges towards the shape of the geometric object with increasing depth of the tree. The partitioning is performed until each subset contains one primitive only or according to a predefined abort criterion, representing a leaf of the tree.

Conflict detection is performed on object's root nodes first. If the root bounding volumes do not intersect, the objects do not intersect. Otherwise, the children of the conflicting root nodes are examined for intersection. If the intersection test is positive between leaves, a conflict is very likely or even ensured, if the leaves have the exact shape of the corresponding geometric subset.

The BVH technique is also often used for intersection tests between Bézier curves. Bézier curves are frequently used in computer graphics to model smooth curves. Bézier curves allow indefinitely scaling. Since a Bézier curve is completely contained in the convex hull of its control points (Prautzsch et al., 2002), in particular the calculation of bounding volumes is very simple.

Zachmann (1998); Klosowski et al. (1998) demonstrate how to apply BVH in VR for haptic force-feedback simulation. They investigate on using different 3-dimensional bounding boxes, Klosowski et al. concentrating on:

- 14-dops using the 6 halfspaces that define the facets of an AABB and 8 diagonal halfspaces cutting off the corners.
- 18-dops using the 6 halfspaces that define the facets of an AABB and 12 halfspaces cutting off the edges.
- 26-dops using the 6 halfspaces that define the facets of an AABB and 20 halfspaces cutting off 8 corners and 12 edges.

Bode and Hecker (2013) use BVH for conflict detection between airborne trajectories. They distinguish between broad and narrow phase during the conflict detection process, where the BVH represents the narrow phase. The broad phase, represented by a combination of R-tree and interval tree, reduces the number of pairwise comparisons for the narrow phase. Bode and Hecker use an older and less performant publication of the algorithms presented in this thesis as a baseline (see Kuenz (2011)). Current results prove that their approach consumes about the triple time (5 ms versus 16 ms) to compare one trajectory against 30 000 others. In their outlook, they declare their intention to investigate on real 4D-data structures for the broad phase, proposing k-d tree and BSP as candidates.

Teschner et al. (2005) propose to use BVH for collision detection respecting deformable objects in an adapted way. Instead of binary partitioning, they prefer 4-ary or even 8-ary trees because of the better overall performance when updating or refitting the hierarchies in case of motion. Furthermore, the hierarchy is strictly oriented on the mesh topology of the object on generation, assuming that this topology does not change on deformation. The article also gives details on techniques like stochastic collision detection, distance fields, and self collision detection. Fields of application are robotics, surgery simulation, and cloth simulation.

The approach described in this thesis does not utilize BVH. BVH is a powerful method for objects that are likely to collide. As also

Bode and Hecker experienced, conflict detection based on BVH has limited efficiency for sparse air traffic, necessitating a partitioning in broad and narrow phase. Fast conflict detection avoids comparisons between objects that are really far away from each other. A domestic flight in Spain cannot conflict with a domestic German flight, and there is no need to compare any bounding boxes. Therefore, this work focuses on a really strong broad phase, eliminating as many comparisons as possible in the first place. The narrow phase is required very rarely, and the position of a possible conflict is already limited to a very small area in all dimensions. Furthermore, the cylindrical shape of objects (at least for aircraft objects) allows fast collision detection without using BVH. Even when using complex distance metrics like World Geodetic System 1984 (WGS84), the results from the algorithms in use are very promising.

2.2.8 Kinetic Data Structures

Kinetic Data Structures (KDSs) are often used to model continuously moving objects in a geometric system, e. g., for the purpose of collision detection. According to Guibas (2001), a KDS can be specified by

- a set of *certificates* defining elementary geometric relations,
- a *motion plan* describing the motion of objects in the near future,
- *events* representing the violation of KDS certificates, and
- an *event queue* holding all events sorted by the time of certificate violation.

The future time of failure for a certificate can be predicted if motion plans are available for all of its objects. An event is classified *external* when the combinatorial structure of the attribute changes, and *internal*, when the structure stays the same and the certificate needs to be changed.

The performance of a KDS depends on four factors (Guibas, 2001):

- *responsiveness:* A KDS is responsive if the cost for repairing a failed certificate and updating the attribute computation are small. Small quantities are considered to be $\mathcal{O}(n^{\epsilon})$ in the problem size.
- *efficiency:* A KDS is efficient if the number of certificate failures is comparable to the number of external events.
- *compactness:* A KDS is compact if the number of certificates is close to linear in the degrees of freedom of the moving system.
- *locality:* A KDS is local if each object participates in few certificates, only.

Basch et al. (1999) demonstrate how to apply KDS for collision detection. Instead of focusing on collision comparisons as illustrated in section 2.2.7 for BVH, Basch et al. concentrate on the free space between moving objects. This free space is subdivided into cells of a certain type. This space deforms while the objects move. The KDS proof of separation remains valid unless cells become self-intersecting. Desired characteristics for the cell types are

- self-collisions are easy to detect,
- tiling can adjust to the motion of the objects, and
- easy update of the tiling in case of self-collision.

Basch et al. use an external relative geodesic triangulation defining a set of flexible shells surrounding each of the objects. The space between these shells is subdivided into pseudo-triangles. Once a pseudo-triangle self-intersects as a result of moving objects, a certificate fails, either necessitating an update of the certificate or representing a potential conflict.

Abam et al. (2006) illustrate how to apply KDS for collision detection in 3 dimensions. While most 2-dimensional approaches decompose the free space between polygons into pseudo-triangles, a suitable decomposition for free space in 3D is less obvious. Abam et al. use guarding points around each object, assuming that the objects are fat. The positions of guarding points ensure that the larger object must contain at least one guard from the smaller object in case of collision.

The proposed solution handles events in $\mathcal{O}(\log^6 n)$ time and processes $\mathcal{O}(n^2)$ events in the worst case. The authors state that their approach

> "...should be seen as a proof that good bounds are possible in theory–whether a simple and practical solution exists that achieves similar worst-case bounds is still open." (Abam et al., 2006)

Zachmann and Weller (2006) combined KDS with BVH to a high performance conflict detection for deformable objects. While the pure conflict detection is performed using the BVH based on AABB and BoxTree, the KDS is used to keep the BVH up-to-date according to the internal deformable object with the minimum number of updates.

Implementations for specialized KDS are available from the Computational Geometry Algorithms Library (CGAL) (The CGAL Project, 2015). CGAL provides implementations for Delaunay triangulation (two and three dimensions) and regular triangulation (three dimensions). Furthermore, exact and inexact operations on primitives are supported along polynomial trajectories.

2.2.9 Sweep and Prune

Cohen et al. (1995) propose to project three dimensional AABB from moving 3D-objects onto the x, y, and z axes. They generate three sorted lists, one for each dimension holding the corresponding object intervals. By applying insertion sort, holding the lists sorted can be performed in $\mathcal{O}(n)$. Furthermore, they keep track of changes in overlap status with an effort of $\mathcal{O}(n + s)$, s being the number of pairwise overlaps. Whenever two objects overlap in all three lists, the corresponding polytope pair is declared active. Thus, the sweep and prune is a broad phase algorithm. The exact collision detection routine (i. e., the narrow phase) is called only for active polytope pairs. The motion is simulated by small time steps, assuming that objects are moving only slightly from frame to frame.

Coming and Staadt (2005) suggest a kinetic sweep and prune for collision detection. Instead of simulating small time steps, they apply a KDS to keep the interval lists sorted. An event is scheduled for each pair of adjacent list elements to catch the crossing of elements. Especially for linear motion, the time of crossing can easily be predicted. That way, discrete time steps can be avoided and time is handled continuously.

2.2.10 Broad Phase Based on Delaunay Triangulation

Tavares and Comba (2007) based their collision detection algorithm on a delaunay triangulation, supposed to be applied in the broad phase. While a triangle vertex represents the center of mass of an object, the edges represent object pairs to be checked in the narrow phase. For each frame of the animation, the triangulation is updated. The performance results do not show benefits compared to sweep-and-prune and brute-force bounding box methods.

2.2.11 Spatial Subdivision

When objects are small on average and they are distributed uniformly, subdivision of their containment volume is beneficial. Typical underlying structures for spatial subdivision are BSP tree (section 2.1.3), k-d Tree (section 2.1.4), and R-tree (section 2.1.5)(Samet, 1990). In contrast to all aforementioned conflict detection mechanisms spatial subdivision can easily be used with any number of dimensions.

2.2.12 Spatial Hashing

Spatial hashing is a special form of spatial subdivision using a hash-function mapping 3D positions to a 1D hash table index. The hashing is performed in the broad phase, leading to a hash table where each index contains a small set of objects to be compared in the narrow phase.

The idea of Teschner et al. (2003) is spatial hashing on objects consisting of tetrahedrons. Each primitive is mapped into a hash table, allowing collision detection between objects and self collision detection. The hash values are calculated from discretized vertex positions $(\lfloor x/l \rfloor, \lfloor y/l \rfloor, \lfloor z/l \rfloor)$ with l being the grid cell size. After mapping all vertices, the proposed algorithm computes all hash values that are affected by the AABB of a tetrahedron. If a tetrahedron is mapped to an index containing vertices from other tetrahedrons, a penetration test is performed. Collisions between tetrahedrons and edges are not considered. The hash function is defined as $\text{hash}(x, y, z) = (73\,856\,093 \cdot x \oplus 19\,349\,663 \cdot y \oplus 83\,492\,791 \cdot z)$ mod n, with n being the size of the hash table.

Spatial hashing does not build an explicit 3D data structure. Instead memory is used only for storing the hash table. The size typically depends on

- Size of the global bounding box.
- Size of objects.
- Acceptable collision risk for non-related objects being hashed on the same index.
- Inner data structure used within hash map.

Considering the scenario setup illustrated in table 4.1 on page 125, the scenario contains $2^{D_x} * 2^{D_y} * 2^{D_z} * 2^{D_t}$ cells, summing up to $2^{39.73}(\sim 10^{12})$ entries. Depending on the choice of the inner data structure and the accepted overlap of hash values, a direct advantage concerning memory usage is not expectable.

Another difference between explicit spatial data structures and spatial hashing is the level of detail that needs to be taken into account when inserting new objects. While the spatial hashing needs to mark all affected grid elements, tree-structured spatial subdivision only needs to adapt to the associated subdivision layer. Although aircraft and their occupation volume are small compared to minimum grid size, this difference especially arises when considering the assigned trajectories. Thus, on insertion in a spatial hashing structure, whole trajectories of aircraft need to be upsampled to the underlying grid structure and mapped into the hash table. Applying dynamic spatial subdivision, the mapping to final grid size is only relevant in densely occupied areas.

2.3 4D Trajectory Prediction

When trying to detect and solve conflicts it is essential to take the problem complexity of the application area into account. If, for example, the motion of objects can easily be described in a closed analytical form, an analytical solution may be beneficial. If objects follow complex trajectories depending on many factors as described here, other data representations should be preferred. Therefore, this section describes the ways to operate an aircraft, and the prediction of an aircraft trajectory, either pre-flight or airborne.

The primary goal is to move the aircraft from the departure to the arrival airport, satisfying the very high safety standards of air transport, in accordance with economic and environmental demands, and providing acceptable working conditions for the pilot crew. While this is true for decades now, the way how to do so changed significantly. Even though following list also represents the chronological order of employment, today all three types of control are typically integrated in modern transport aircraft:

- The most basic way to fly a transport aircraft is by giving instructions using the control column or side stick and the thrust lever. The pilot directly controls the aircraft.
- A more automated way of flying an aircraft is using the autopilot. The autopilot controls lateral and vertical speed, bearing, altitude and localizer/glideslope in approach mode. The pilot controls the aircraft on a higher, semi-automatic level.
- A Flight Management System (FMS) provides a fully automatic control system for flying an aircraft. The pilot enters

whole routes into the FMS. The system predicts a trajectory from departure to destination and automatically guides the aircraft along this route. The pilot manages the aircraft. The pilot still may use the control systems above, however this decreases predictability and typically efficiency of flight. The lower level systems are the safety net for the FMS.

Figure 2.7 shows the cockpit from the former research aircraft Advanced Technologies Testing Aircraft System (ATTAS) operated by Deutsches Zentrum fuer Luft- und Raumfahrt (German Aerospace Center) (DLR) providing all aforementioned ways of controlling an aircraft. The ATTAS is a Vereinigte Flugtechnische Werke (VFW) 614 twin engine jet transport aircraft modified with worldwide unique equipment for research purpose (fig. 2.8). ATTAS was recently retired after 26 years in service. Its direct successor is Advanced Technology Research Aircraft (ATRA), a modified Airbus A320-232.

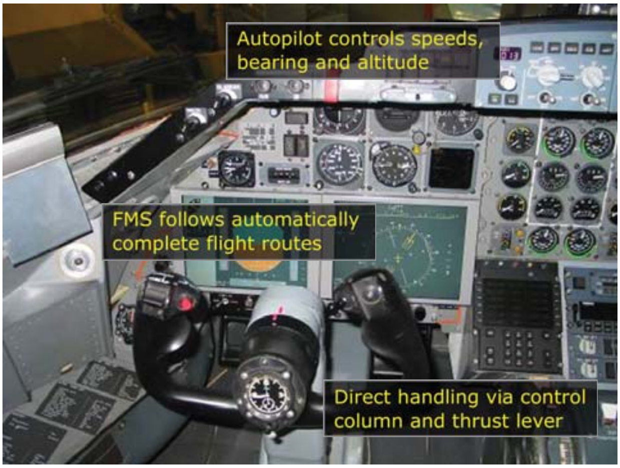

Autopilot controls speeds, bearing and altitude
FMS follows automatically complete flight routes
Direct handling via control column and thrust lever

Figure 2.7 – ATTAS Cockpit allowing 3 Ways of Flying

ATTAS is equipped with one of world's most advanced FMS. The Advanced Flight Management System (AFMS) was developed by the Institute of Flight Guidance at DLR Braunschweig. Various flight trials with ATTAS and also first trials with ATRA proved

Figure 2.8 – DLR's former Research Aircraft ATTAS

high accuracy and flexibility of the AFMS (Korn and Kuenz, 2006; Kuenz et al., 2007).

Using an FMS for automatic flight execution does not only reduce pilot's workload but is also the on-board technical enabler for TBO.

2.3.1 The Advanced Flight Management System

This section holds a description of DLR's AFMS being a good representative for a modern FMS. The initial version of the AFMS was started in early 1990s (Adam and Kohrs, 1992; Kohrs, 1992; Czerlitzki, 1994; Czerlitzki and Kohrs, 1994) and since then continuously improved.

By means of strategic trajectory planning and a corresponding guidance module the AFMS allows planning of highly accurate 4D-trajectories and following them with little deviations autonomously.

Figure 2.9 shows the in- and output data of the AFMS. Generation of 4D-trajectories is performed based on a list of waypoints describing the route from departure (or actual position when already airborne) to the destination, altitude, speed and time constraints, aircraft's performance data and an accurate weather forecast. The weather forecast for flight trials is provided by Germany's national

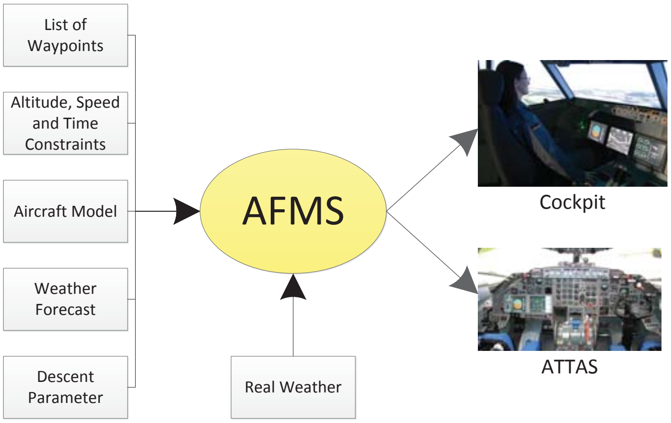


Figure 2.9 – In- and Outputs of AFMS

meteorological service Deutscher Wetterdienst (DWD). Using the descent parameters given by the pilot the AFMS allows predicting:

- Low Drag Low Power (LDLP) approaches with selected intercept altitude and level length. LDLP tries to use clean configuration as long as possible. Besides a significantly higher noise emission, extracting slats, flaps and gear increases drag and thus reduces flight's efficiency. However, the LDLP intercepts the final glideslope in level flight in order to allow late adjustments for the final descent.

- Continuous Descent Approach (CDA) with selected intercept altitude. The CDA also makes use of low drag configurations. Furthermore, CDA avoids the final level at glideslope intercept. Therefore, the aircraft flies higher and thus more efficient. Noise pressure level immissions on the ground are reciprocally proportional to the altitude, doubling the altitude reduces noise pressure level by 50 % (Deutsches Institut für Normierung, 1999).

- Segmented Continuous Descent Approach (SCDA) with selected start of steep descent and intercept altitude. The steep descent profile is produced by an early configuration in high altitudes. The steep descent part is flown with flaps and gear out. Drag is increased for this procedure, and thus also fuel burn. However, depending on aircraft type, the noise influence of the higher altitude of the profile can outweigh the increased noise emissions due to configuration.

Figure 2.10 shows example profiles for the supported approach types. All profiles have in common the final segment down to the runway threshold that is usually a descent with 3°.

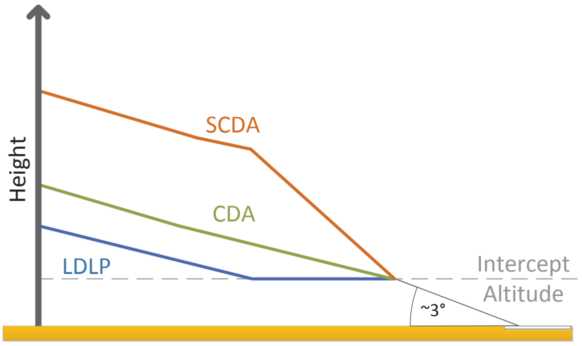


Figure 2.10 – Approach Types supported by AFMS

The term CDA is often used non-stringently in literature, it developed to a buzz word - and most airport support their own kind of CDA procedure. Loose definitions allowing level flights with maximum defined lengths and very shallow descent parts are often allowed. DLR's advanced CDA has the following advanced features compared to a standard CDA:

- Commencing the advanced CDA from an altitude where the aircraft is silent on the ground there is no level flight until touchdown.
- Descents are performed with engines idle. Thus, sink rate and flight path angle are not necessarily constant while descending.

Idle thrust does not only reduce noise emissions of the engines but also reduces noise immissions on the ground and fuel consumption due to higher and therefore more economical flight profiles.

- The vertical profile can be specified independently of the lateral path. This enables the implementation of special procedures like curved approaches.

One main task when calculating a 4D-trajectory performing a CDA is to predict an appropriate position for the Top Of Descent (TOD). First, the AFMS calculates the glideslope intercept point by means of glideslope angle, intercept altitude and runway threshold position and elevation. The AFMS calculates the TOD by stepping backward from the glideslope intercept point (see fig. 2.11), implying an idle descent to the glideslope intercept.

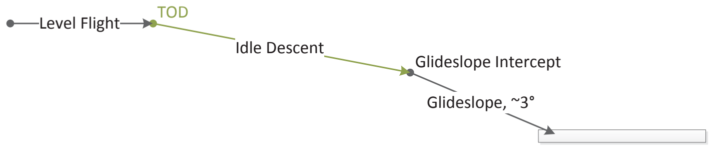


Figure 2.11 – Calculating the TOD

The foreseen airspeeds depend on phase of flight and type of aircraft. Optimum speeds for different flight phases and all other relevant information about aircraft are published for most transport aircraft by the European Organisation for the Safety of Air Navigation (Eurocontrol) in the Base of Aircraft DAta (BADA), current version in use is 3.9 Eurocontrol (2011).

Once having generated a 4D-trajectory the AFMS provides guidance commands to fly along the calculated trajectory. A 4D-trajectory consists of a lateral route with altitude and time information for every waypoint. If an appropriate connection to the autopilot is available these commands are directly forwarded to the aircraft that will automatically follow the trajectory. If such a connection is not available the guidance commands can be displayed as instructions to be carried out by the pilot. The AFMS guidance commands control the aircraft in all 4 dimensions (lateral, vertical and time).

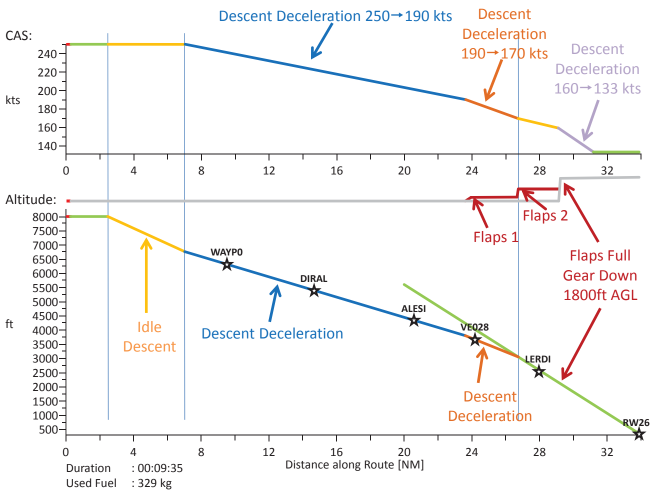


Figure 2.12 – CDA Trajectory for Airbus A330-300

For a precise prediction and guidance along 4D-trajectories the AFMS has also to consider the aircraft's configuration. The higher drag and lift coefficients of extended flaps otherwise leads to deviations which might not be accepted in a 4D TBO traffic management, as described by de Muynck et al. (2011).

Figure 2.12 depicts an example of an advanced CDA calculated by the AFMS for the Airbus A330-300. Usually, a flight is subdivided into five phases: Departure (Dep), Climb (Clb), Cruise (Crs), Descent (Dsc) and Arrival (Arr). The example starts in cruise flight.

The TOD is in Flight Level (FL) 80 where the aircraft is in clean configuration. The descent starts idle with a constant Calibrated Air Speed (CAS) of 250 knots (kts). This is followed by an energy sharing phase where the aircraft both descents and decelerates. The glideslope is intercepted at 3000 ft with 170 kts, flaps just coming out to position 2. At 1800 ft above ground level the aircraft is configured for landing (flaps full, gear down). Flying the standard glideslope approach the aircraft will need thrust to hold the landing speed on the very last part before landing. Deviations may occur during the execution of an advanced CDA due to:

- Insufficient or imprecise aircraft performance data.
- Jitter in the configuration points.
- Bad weather forecast.
- ...

When forced to deviate from the predicted trajectory because of unforeseen influence described above, the AFMS guidance functionality tries to hold the time deviation at minimum and in exchange accumulate the altitude error. The altitude error is compensated when intercepting the glideslope. This type of readjustment depends on whether the aircraft is too high or too low.

Being in time and having a positive altitude error (aircraft is too high) means that the aircraft has too much energy left. Since the engines are idle in descent there is no way out with the thrust. Therefore, the AFMS reacts by increasing the drag. If the AFMS detects a positive altitude error when intercepting the glideslope it brings forward dynamically the configuration times for flaps and gear. A negative altitude error (aircraft is too low) implies a lack of kinetic energy. An early reaction in form of setting higher thrust should be avoided because:

- Slow response times of jet engines make a closed loop control difficult.
- Even small changes of the engine speed are felt disturbing by the passengers.

A negative altitude error is corrected by insertion of a less steep segment. Only in extreme cases this segment will be a level segment. In order to get rid of the missing energy, the AFMS brings forward the point of leaving idle thrust. Thus, there is no new phase of closed loop low power control but a small extension of the thrust phase just before landing.

Figure 2.13 and fig. 2.14 are noise footprints for an Airbus A320 approach to Frankfurt via Gedern. The trajectories were calculated by the AFMS and fed into the DLR noise calculation tool SIMUL (Boguhn, 2007). The noise areas start with a dark blue for >55 dB(A) and increase in steps of 5 dB(A). The difference between fig. 2.13 and fig. 2.14 illustrates the noise benefit achievable by selecting the advanced CDA descent in favor of LDLP.

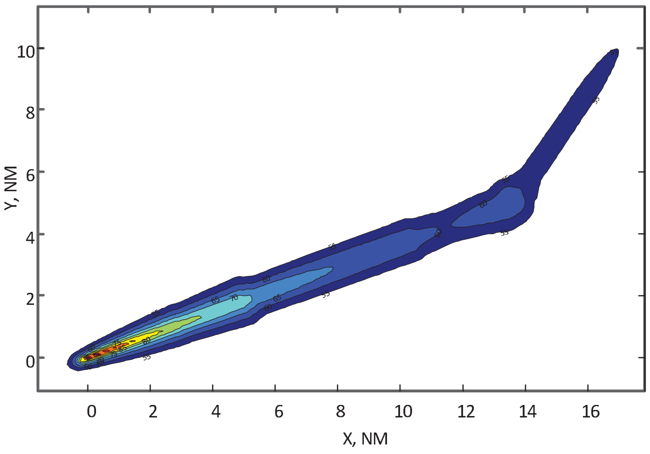


Figure 2.13 – Noise Footprint of LDLP

The AFMS prediction and guidance capabilities have been validated in several simulation runs using the A330-300 Full Flight Simulator formerly operated by the Zentrum für Flugsimulation Berlin (ZFB) and flight trials with DLR's test aircraft ATTAS and ATRA.

Both the simulations and real flight trials were performed starting in FL70-FL110 with enough way left to touchdown allowing

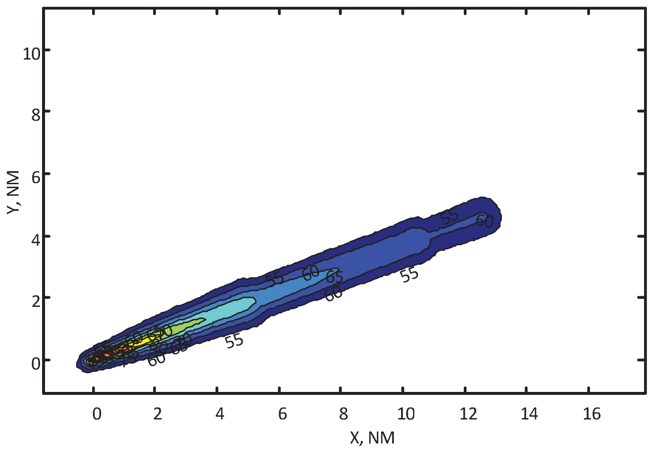


Figure 2.14 – Noise Footprint of CDA

prediction of advanced CDA and LDLP approaches. The ATTAS and ATRA flights were arranged at the base airport Braunschweig principally using the runway 26. The destination airport for the A330 simulations was Munich.

Figure 2.15 displays a typical result of an advanced CDA approach with ATTAS. The figure depicts with a red line the occurred altitude error with a maximum of 70 ft, the airspeeds, the times of flaps and gear transitions and the altitude profile. Typical precision for more than 100 approaches was a maximum of ±150 ft altitude error and ±5 s time deviation at the touchdown point.

The figure also shows that the altitude error peaks are at the transitions from one flight path angle to the next, e. g., at the TOD and the glideslope intercept point. This behavior is reasoned by the fact that the AFMS does not calculate flight path angle transitions and therefore they are not part of the trajectory. Hence, the altitude error calculation by building the difference between actual altitude and trajectory tends to be bigger than the real altitude error.

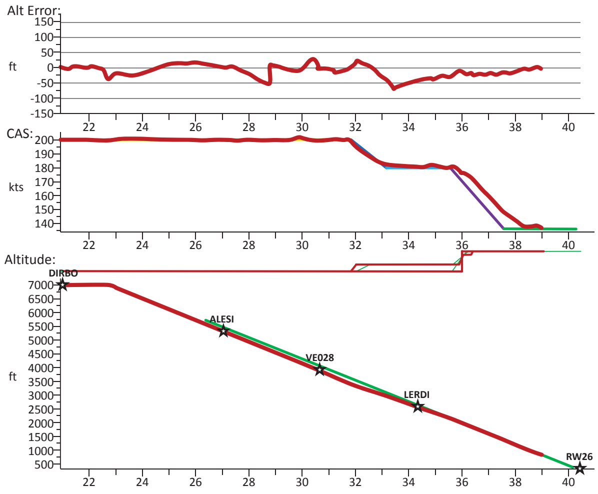


Figure 2.15 – CDA flown by ATTAS

Two flight trials have been untypical with a time precision of rather bad 10 seconds. Investigations revealed that an imprecise weather forecast reasoned the unusual time deviation. On one occasion, the Harz mountains created a constant downdraft in their lee, where Braunschweig is situated, but the weather forecast file only contained information about the horizontal wind and not about vertical wind components. On the other occasion, a mini jet stream was encountered between 5000 ft and 10 000 ft, but the wind was only forecasted at these two altitudes and around, not in-between. Thus the jet stream was not detectable in the wind data and not taken into account when predicting the trajectory. Finally, considering the imprecise weather forecast, a time deviation of 10 seconds is not bad at all for more than 30 NM of flight.

The main driving factor for the A330 simulations was to prove the usability of the AFMS for any aircraft listed in the BADA folder (currently 295 aircraft) and not ATTAS only. The results in fig. 2.16 prove high quality of the AFMS prediction as well as a suitable BADA modeling. Top down, the figure depicts the time

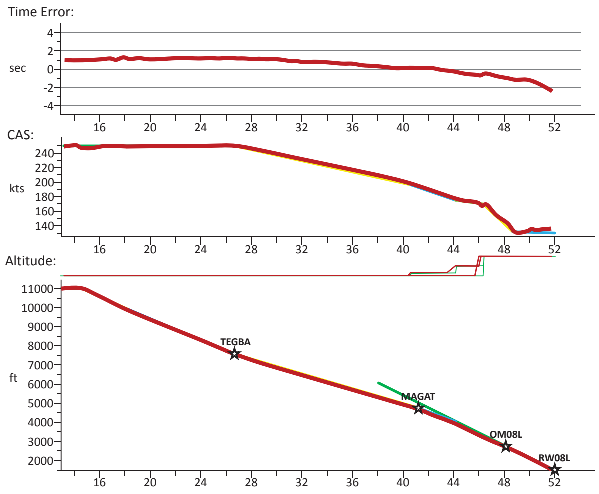


Figure 2.16 – CDA flown by Airbus A330-300

error, the airspeed profile, times for flaps, slats and gear transitions and the altitude profile for the whole descent to Munich.

The green line ending at the runway in the altitude profile is the glideslope angle, in Braunschweig 3.5 degree and for Munich 3.0 degree. Remarkable in the altitude profile is the much smoother descent because of the better glide angle of the A330.

Typical maximum altitude errors of 100 ft and time deviations of up to 3 seconds at touchdown have been evaluated with the A330 full flight simulator. The higher precision with the A330 compared to ATTAS can be attributed to the missing realistic weather. There is also a meteorological model available for winds and gusts for the A330 simulator, but it is not as unforeseeable as reality.

2.4 Conflict Detection in Aviation

Several algorithms already exist performing conflict detection in aviation. Nowadays, a conflict in aviation based on radar and/or Automatic Dependent Surveillance-Broadcast (ADS-B) position data usually exists if lateral distance in latitude/longitude plane is lower than 5 NM and the vertical distance is below 1000 ft. In the vicinity of airports in approach and departure phase, a reduced separation is often used depending on wake categories of both preceding and succeeding aircraft. Bigger aircraft produce stronger wake turbulence while smaller aircraft are more sensitive to wake turbulence areas. The International Civil Aviation Organization (ICAO) defines separations collected in table 2.1 (ICAO, 2007), based on following categories:

- *Heavy (H)*: All aircraft types with 136 000 kg or more maximum take-off weight.
- *Medium (M)*: All aircraft types with less than 136 000 kg, but more than 7000 kg maximum take-off weight.
- *Light (L)*: All aircraft types with 7000 kg or less maximum take-off weight.

Table 2.1 – Separation in [NM] Depending on Wake Categories

		Preceding aircraft		
		Heavy	Medium	Light
Succeeding aircraft	Heavy	4	3	3
	Medium	5	3	3
	Light	6	5	3

Although the Airbus A380 is classified as heavy it produces significantly more wake turbulence than all other heavy classified aircraft. Therefore, the ICAO advises 6 NM distance to succeeding heavy, 7 NM distance to succeeding medium, and 8 NM distance to succeeding light aircraft (ICAO, 2008).

In addition, special separation requirements exist for parallel and nearly parallel runways. According to ICAO (2004b), two major modes of operation are distinguished for arrivals:

- For independent parallel approaches, radar separation minima *are not* prescribed for aircraft on extended runway center lines.
- For dependent parallel runways, radar separation minima *are* prescribed for aircraft on extended runway center lines.

According to ICAO (2004a), the minimum separation for dependent approach operations between runway center lines is 915 m; independent operations are recommended with at least 1035 m separation. If the airport's Secondary Surveillance Radar (SSR) does not provide:

- a minimum azimuth accuracy of 0.06 degrees; or
- an update period of 2.5 seconds or below; or
- a high resolution display featuring position prediction and deviation alert;

a larger distance of 1310 m or even 1525 m is necessary for independent parallel approach operations, depending on whether the aircraft operation would be adversely affected. However, at least a minimum azimuth accuracy of 0.3 degrees or better and an update period of 5 seconds or less is required for independent approach operations.

Dependent runway approaches foresee a 3 NM radar separation for flights approaching on the same runway and 2 NM radar separation for aircraft landing on adjacent runways.

ICAO (2004b) also defines dependent and independent parallel runway operations for departures and mixed mode.

2.4.1 Conflict Situations in Aviation

Conflict detection is necessary in different situations with different time horizons. This section gives a summary on conflicts in aviation, starting with the longest planning horizon.

2.4.1.1 Sector Load and Flow Control Conflicts

Every aircraft flying according to IFR in Europe needs to file a flight plan and feed it into the Central Flow Management Unit (CFMU). The flight plan contains, amongst others:

- Estimated time of departure and arrival.
- Cruise speed.
- Requested flight route.
- Cruise flight level.
- Departure and arrival airport.
- Alternate airport.
- Type and callsign of aircraft, wake category.
- Number of persons on board.

Based on the flight plan, the CFMU (operated by Eurocontrol) assigns slots at departure and arrival airports. The CFMU validates that all sectors used by the requesting aircraft are still below their maximum load limit. The load factor of sectors is not calculated on single aircraft events, but estimated on traffic flows. Time horizon for CFMU planning goes from 6 days to one hour before departure.

2.4.1.2 Trajectory Based Conflicts

Modern ATM concepts are often based on TBO. Every aircraft flies according to a 4D trajectory that is conflict free from all other trajectories. Furthermore, additional information can be taken into account as weather phenomena, restricted areas and other constraints. TBO can be performed with several time horizons:

- **Strategical**: Several months before departure, airline operation centers need to have rough estimations of departure and arrival times, e. g., in order to sell tickets. Typically, accuracy of these predicted times is rather low because they are based on vague weather forecasts. However, even if the predicted trajectories differ from the really flown ones, a general feasibility proof based on individual flights can be performed.

- **Pre-tactical**: Coming closer to the departure time, trajectories can be updated with more accurate information. Changing trajectories necessitates new conflict detection and resolution.
- **Tactical**: When aircraft cannot comply with their assigned trajectories, new trajectories need to be predicted that are conflict-free from the surrounding traffic and other constraints.

2.4.1.3 Medium-Term Conflict Detection

Medium Term Conflict Detection (MTCD) recognizes conflicts of an aircraft usually with a time horizon of 20 minutes ahead at maximum. This conflict detection can be based on different data, e. g., on trajectories using TBO, aircraft intend data broadcasted via ADS-B, or extrapolation of current flight data.

2.4.1.4 Short-Term Conflict Alert

Short Term Conflict Alert (STCA) is usually used as an Air Traffic Controller (ATCo) support tool with a prediction horizon of 2 minutes. STCA is integrated as a safety net; its presence is ignored when calculating airspace capacity. The conflict detection is usually based on extrapolating current aircraft state data.

2.4.1.5 Airborne Collision Avoidance System

The Airborne Collision Avoidance System (ACAS) was introduced in order to reduce (near) midair collisions (Eurocontrol, 2012). ACAS is the last safety net on the air side. The conflict detection is based on SSR transponder signals; non transponding aircraft are not detected. ACAS works on small time scale. The maximum generation time of a traffic advisory is 48 seconds before reaching the Closest Point of Approach (CPA). With a maximum generation of 35 seconds, ACAS issues a resolution advisory. According to (ICAO, 2002), ACAS I supports *see and avoid* but does not provide resolution advisories. Resolution advisories are always vertical in today's dominant version ACAS II, and coordinated with other ACAS II equipped aircraft via a selective mode S link, so that two aircraft choose complementary maneuvers.

2.4.2 Trajectory-Based Conflict Detection

Conflict detection on a set of N aircraft trajectories can easily be performed by brute-force conflict detection between every possible pair of two trajectories. This results in $N(N-1)/2$ comparisons and thus a complexity of $\mathcal{O}(N^2)$. A conflict between two trajectories t_1, t_2 can be detected by segment and point comparisons of the two trajectories (Schwoch, 2008).

The segment based method compares every segment of t_1 with every segment of t_2. If both segments described by great circle subsets intersect, a collision is detected. However, this approach does not detect parallel conflicting trajectories.

Therefore, the point based method compares each trajectory point of t_1 with the time-corresponding point of t_2, and the other way round. If the distance is below the mandatory separation distance, a conflict is detected.

If trajectory points are too sparse, new interpolated trajectory points are added in order to catch all conflicts. This procedure is used as the baseline conflict detection algorithm for performance tests in chapter 4. As the algorithm's complexity indicates this method is not reasonable for large scenarios with many trajectories.

An advanced baseline can be defined by adding an optimization: trajectories not having a common time frame (i. e., t_1 arrives before t_2's departure, or t_1 departs after t_2's arrival) are early omitted since they do not conflict. This method is used as advanced reference in chapter 4. The performance benefit of the optimized baseline algorithm is significant.

Further optimization following the same basic idea was performed within the Programme for Harmonised Air Traffic Management Research in Eurocontrol (PHARE). Kremer et al. (1999) proposed a method of performance increase by means of coarse filters. Coarse filtering reduces number of trajectory pairs under investigation for a final conflict check. If aircraft do not have overlapping flight levels, the corresponding pairs can be omitted for conflict detection. Furthermore, Kremer proposed to cut flights into the parts departure, en-route, and descent. For each part, the bounding boxes can be checked for intersection with the potential conflict partner's bounding boxes to reduce number of final conflict probes.

2.4.3 Subdivision of Airspace

Koeners and de Vries (2008) described an approach on how to subdivide a given airspace in three dimensions (latitude, longitude and time) to reduce the effort of conflict detection. A static grid is filled by trajectories, grid cells are either marked as occupied or free (fig. 2.17). Cells occupied by only one trajectory are colored blue for trajectory 1 and green for trajectory 2, while cells occupied by both trajectories are colored red. Koeners and de Vries propose a cell size being half of the separation size. When storing routes in the grid, a cell buffering technique copies data also to all neighboring cells (Jardin, 2005). Since the subdivision is static this method is only reasonable for strongly limited geographical areas, e. g., the terminal maneuvering area of an airport.

An octree subdividing latitude, longitude and altitude is used by Hildum and Smith (2004); Smith (2008). Subdivision is applied each time an octant contains a conflict. The octree is stored using a linear octree structure as described by Gargantini (1982). The idea of a linear octree is a linear access on octants by means of a key derived from the latitude, longitude and altitude position. This allows access on every octant organized in a binary tree in $\mathcal{O}(\log N)$ time. Each octant then stores a list of intersecting objects with the corresponding intersection times. Figure 2.18 shows an example of a conflict situation. The proposed algorithm also checks neighbor-octants in order to detect every conflict.

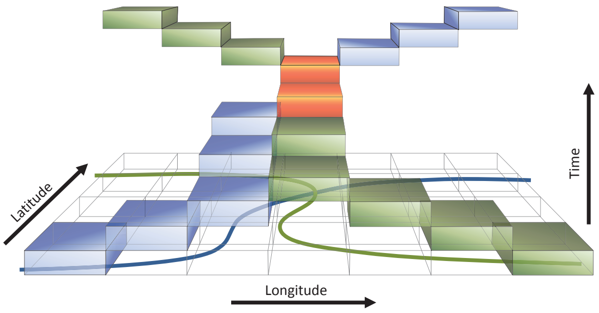


Figure 2.17 – Example Grid with Conflict according to Koeners and de Vries (2008)

This approach is especially beneficial if time extension of the scenario is small, and separation is guaranteed laterally or vertically. The described method is not efficient for areas majorly ensuring separation by time, e. g., an octant containing the threshold of an arrival runway.

Jardin (2003) uses a static 3D-grid subdividing latitude, longitude and time. He applies a grid spacing of 5 NM and 30 seconds in time, resulting in ∼16 MiB of memory necessary for Continental United States per flight level. Each grid cell holds a single binary value indicating if the cell is occupied by other aircraft or weather. An explicit conflict list is not generated by the algorithm since Jardin performs direct conflict solution when a new trajectory conflicts with already occupied cells on insertion. Of course, neighboring cells also need to be taken into account.

Jen Chiang et al. (1997) proposed a static 3D-geometric subdivision with minimum requested separation size. Conflicts are detected by searching multiple aircraft in one tile and the 26 surrounding neighbors. This approach is fast since most neighbor tiles are empty in practice. However, the tests need to be performed for several discrete times with time offset Δt. Section 3.3.1 describes a different method for handling neighborhoods.

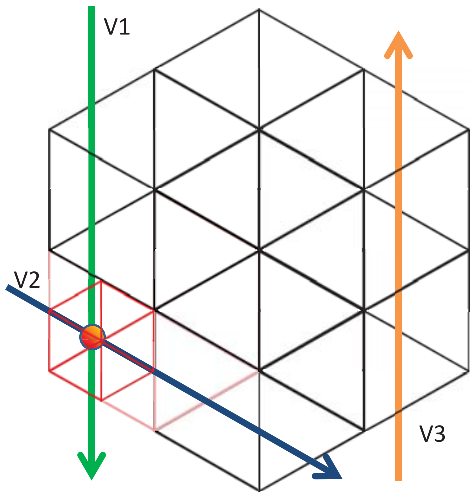


Figure 2.18 – Conflict in Octree between V1 and V2 according to Hildum and Smith (2004)

2.4.4 Conflict Detection Considering Uncertainty

Amongst others, Erzberger investigated the idea of respecting trajectory uncertainties in conflict detection (Erzberger et al., 1997). Claiming at least 10 minutes ahead time for conflict detection, the optimum time is a trade-off between efficiency and certainty. The earlier a resolution maneuver is initiated, the more efficient it is. However, the more look ahead time is applied on trajectories, the more uncertainty is integrated in the conflict decision.

The conflict detection functionality is based on trajectories and statistical model of prediction errors, represented as ellipsoids in three-dimensional space (fig. 2.19). In order to limit conflict detection trials, Erzberger et al. propose three methods:

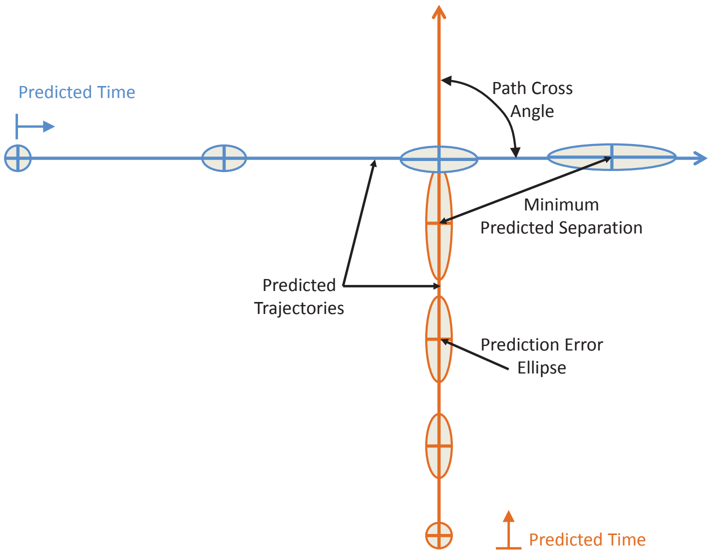


Figure 2.19 – Trajectory Prediction Error Ellipses according to Erzberger et al. (1997)

- Trajectory pair pruning omits conflict detection if trajectories are spatially exclusive in either altitude or horizontal position. Therefore, it is checked if the trajectories fly in separated flight levels and if their lateral bounding boxes intersect. That way, 60 % to 80 % of all possible trajectory pairs are pruned.

- Separation computations are minimized. For example, lateral calculations are avoided if aircraft are already separated vertically. Lateral cross-distances are calculated only if x- and y-distance are both below laterally required separation.

- Time skipping avoids conflict detection on trajectory segments that are already ensured to have no conflict. Thus, if two trajectories have a big distance at one test time τ, their distance is assumed to not converge faster towards zero than $10\,000\,\mathrm{ft\,min^{-1}}$ vertically and $0.33\,\mathrm{NM\,s^{-1}}$ (about 2 Mach at standard sea level conditions) laterally. Therefore, if two aircraft have a distance of 200 NM at a given time τ, it is not necessary to test them again before $\tau + 10$ minutes.

The combination of the above described methods leads to a conflict detection time of 10 seconds for up to 800 trajectories and a prediction horizon of 30 minutes running on the scalable multi-workstation architecture of Center-TRACON Automation System (CTAS).

In contrast to Erzberger's proposal the conflict detection algorithm described in this work has no direct support for uncertainty management for several reasons:

- Trajectory prediction and execution accuracy has increased in the last decades, compare section 2.3.1.
- A prediction horizon of 30 minutes is not enough in a TBO environment handling gate-to-gate traffic. An uncertainty forecast for flights lasting several hours is hard to predict, if possible at all, with an acceptable accuracy.
- This approach handles whole trajectories. Thus, it covers also the bottlenecks of today's airspace, which are usually the runways. Aircraft are staggered densely on runways both for arrivals and departures. Allowed deviations from their foreseen trajectories is only few seconds in arrival and departure phase. Including uncertainties that accumulate during the flight to the arrival phase decreases runway's throughput to an unacceptable level.

2.5 Conflict Resolution in Aviation

In general there are three types of conflict resolution in aviation as depicted in fig. 2.20:

- Lateral conflict resolution avoids a predicted conflict by a lateral detour.
- Vertical conflict resolution lets one aircraft sink below or climb above the conflict partner.

- Time-based conflict resolution changes the time of arrival at the conflict.

Direction, duration and most efficient type of conflict solution depends on the conflict properties. Aforementioned resolution

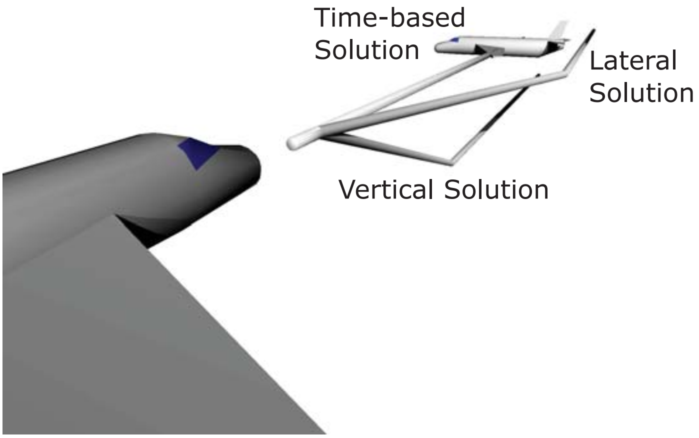


Figure 2.20 – Three Types of Conflict Resolution

types are not feasible in all situations. A combination of the given resolution types may result in more efficient maneuvers.

For instance, resolution advisories from the ACAS (section 2.4.1.5) should be directly executed by a pilot. Due to the short time horizon, time-based conflict solution is not envisaged. Since horizontal resolution of SSR is rather low, ACAS resolution advisories are only vertical nowadays.

Especially when considering TBO and strategic planning of trajectories, conflict solving can generally be performed in all three ways for en-route conflicts.

In contrast, vertical conflict resolution is not an option for conflicts on the runway because threshold elevation is a geographical constraint. The same accounts for lateral resolution with the exception of allowing to reroute the aircraft from/to another close-by runway.

Several publications already handle conflict resolution. The following sections give an extract.

2.5.1 Conflict Resolution using Trial-and-Error

Erzberger et al. (2010) propose to not only search for one solution solving the conflict under investigation but continue the search and afterwards select the best from several solutions. A major factor for a good solution is a small introduced time delay. Since conflict resolution is complex, Erzberger et al. distinguish several cases for lateral (fig. 2.21), vertical (fig. 2.22) and time-based (fig. 2.23) solutions.

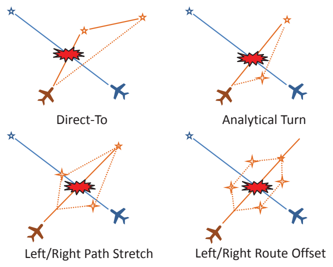


Figure 2.21 – Horizontal Resolution according to Erzberger et al. (2010)

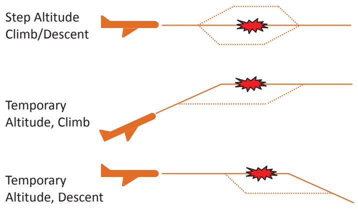

Step Altitude Climb/Descent
Temporary Altitude, Climb
Temporary Altitude, Descent

Figure 2.22 – Vertical Resolution according to Erzberger et al. (2010)

Some of the resolution types are not applicable in any circumstance. For instance, the lateral direct-to maneuver depends on an existing dogleg that can be bypassed directly to a waypoint further downstream. Doglegs are commonly used today but should not be standard in envisaged TBO. In exchange, if a direct-to is possible it is usually even more efficient than the unresolved situation before.

According to Erzberger et al., the path stretching has the best success rate among the horizontal resolution types. The additional waypoint is put on an ellipse with the aircraft's position as one focus and the return point as second focus. The radii depend on the specified delay.

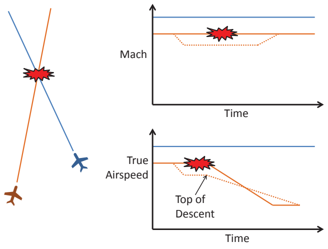


Figure 2.23 – Time-Based Resolution according to Erzberger et al. (2010)

Summarizing, Erzberger et al. create ∼128 potential solutions (2 direct-to, ∼16 vertical, 6 analytical turn, 4 route offset, ∼80 path stretch, and ∼20 speed). Based on these, he generates ∼128 trial trajectories with a trajectory predictor and probes them in the conflict detector. From up to 15 solutions, he chooses the one that fits best according to predefined metrics.

2.5.2 Conflict Resolution using Genetic Algorithm

Durand et al. (1996) propose to solve conflicts globally with genetic algorithms. Starting with a population of conflicting trajectories, three basic operators are used to influence conflicts: selection, mutation and crossover. A fitness function is defined, defining the desired properties of the population, e. g., having no conflicts and being cost efficient. The selection operator picks suitable individuals from the population to breed a new generation. The mutation operators ensure genetic diversity by changing one or more values from single individuals. The crossover operator generates children based on more than one parent individual. The process iterates as long as fitness improves. Problems of genetic algorithms are:

- Quality of the overall process strongly depends on the quality of the fitness function. Defining a good fitness function sometimes is a difficult task. If the fitness function is complex, frequent calls destroy the overall performance. Fitness functions should be continuous instead of false/true in order to give hints for convergence direction.

- Genetic algorithms scale badly with complexity. Large populations with many parameters span a large search space.

- Genetic algorithms tend to converge to local optima instead of finding a global optimum. This behavior can be avoided by changing the fitness function appropriately.

- It is difficult to estimate convergence behavior and time to solution. Certification of genetic algorithms is (at least) difficult in aviation.

2.5.3 Conflict Resolution based on Potential Fields

Another approach of solving conflicts is based on potential fields' theory (Kelly and Eby, 2000; Roussos et al., 2008). The basic idea is to use an electrostatic modeling of the problem where aircraft are handled as electrons with negative charge. Since same charges push away each other, conflict avoidance is directly integrated in the model. Destinations have a positive load which attracts aircraft. Aircraft are moving towards their destination while being influenced from the resulting electric field. Thus, aircraft/ electrons being instantiated at their departure fly a most likely conflict free route towards their destination.

Calculation of complex global situations is very demanding and time consuming using potential fields. Therefore, this kind of conflict resolution is mostly used on local conflict situations. A big problem is constraining the algorithm in order to get flyable trajectories respecting the performance and maneuverability limitations of aircraft and constraints from ATM.

2.5.4 Conflict Resolution based on Light Propagation

Dougui et al. (2010) propose an algorithm based on the different refractions of light. Refraction indices are assigned according to the congestion of airspace. Optimal trajectories are then found by calculating the shortest light path between two points respecting the local metrics built, e. g., from refraction indices and aircraft protection zones. Calculation times are reduced by limiting the search space using a branch-and-bound method.

The proposed algorithm runs in (2D+time) and usually comes up with a flyable conflict free trajectory.

2.5.5 Conflict Avoidance in Crowd Simulation

Crowd simulation is often used in computer graphics and movie productions when the movement and behavior of a large amount of objects and characters shall be simulated. The individual entities behave according to assigned rules and thus interact with their environment, e. g., by avoiding conflicts with other entities. Although crowd simulation is often used to simulate human behavior in a 3-dimensional system, the same mechanisms can easily be adapted to a 4-dimensional system simulating aircraft traffic.

Foudil et al. (2009) perform crowd simulation in a discretized coordinate system modeled as a grid of cells. The path of an object is generated using the A^* algorithm (Hart et al., 1968), each object occupying one cell at each time. The predicted path is collision free from static objects. Conflicts with other entities are classified in Toward (head-to-head), Away (overtake situation) and Glancing (side-on). Conflict avoidance is then performed on human life experience, e. g., by taking the smallest deviation for the Toward case and performing a slow down to avoid an Away collision.

Unfortunately, the proposed fallback procedure to stop walking when no solution can be found is not directly transferable on aircraft, being strongly limited by deceleration gradient and minimum air speed. Especially queuing at saturated arrival runways requires other techniques, preferably with a larger look ahead time.

Golas et al. (2013) propose a hybrid approach of local collision avoidance in the vicinity of crowd entities and approximate, long-range collision avoidance. The long term look ahead is based on an extrapolation of the known path. In order to avoid wrong decisions based on approximated data, the look ahead time is restricted.

Even though the approach of Golas et al. has a higher potential to draw near the overall optimum than a pure local collision avoidance, a centralized global system seems more suitable to find the most efficient solution than a distributed crowd simulation with each entity having a very restricted view.

2.5.6 Extended Flight Rules

The Extended Flight Rules (EFR) do not provide an algorithm solving conflicts but a method to identify for a conflict situation which aircraft has right of way (Duong et al., 1996). The priority assignment considers maneuverability, current flight phase, and speed of both aircraft.

If two aircraft in normal operation and same flight phase having an encounter the faster aircraft must give way to the slower.

In normal operation with different flight phases, priority is assigned as described in table 2.2. The letter R on the main diagonal corresponds to aforementioned distance/speed rule. A and B means priority is given to aircraft A or B respectively. The EFR also define further rules, amongst others, for encounters between more than two aircraft, different equipped aircraft, and emergencies.

2.6 Geodetic Earth Systems

Since the algorithms described in this paper assume a Euclidean coordinate system while the problem in focus is Earth related, this section describes some commonly used Earth models. Chapter 4 illustrates how the shape of Earth is taken into account by the algorithms.

Table 2.2 – Priority for Aircraft in Different Flight Phases (Duong et al., 1996)

		Aircraft B							
	Phase	Climb			Cruise		Descent		
Phase	Subphase	Initial	Intermed.	Final	Normal	Pre-Desc.	Initial	Intermed.	Final
Aircraft A Climb	Initial	R	A	A	A	A	A	A	B
	Intermed.	B	R	B	B	B	B	B	B
	Final	B	A	R	B	B	A	A	B
Cruise	Normal	B	A	A	R	A	A	A	B
	Pre-Desc.	B	A	A	B	R	A	A	B
Descent	Initial	B	A	B	B	B	R	B	B
	Intermed.	B	A	B	B	B	A	R	B
	Final	A	A	A	A	A	A	A	R

Geodetic Earth models are simplified geometrical systems describing the shape of Earth. Positions on Earth are often given in East-North-Up (ENU) notation referenced by a longitude $\lambda \in [-\pi, \pi)$, latitude $\varphi \in [-\pi/2, \pi/2]$, and a height. Positions on the northern hemisphere have a positive latitude, positions on the eastern hemisphere positive longitudes, and positions being further away from Earth center higher heights. The 3D-position of a surface point $P(\varphi, \lambda)$ depends on the geodetic reference system being used (fig. 2.24).

A simple model of Earth is the spherical system, assuming that the shape of Earth is close to a sphere. There are several different Earth radii used in literature depending on the optimization criteria usually lying between 6370 km and 6380 km.

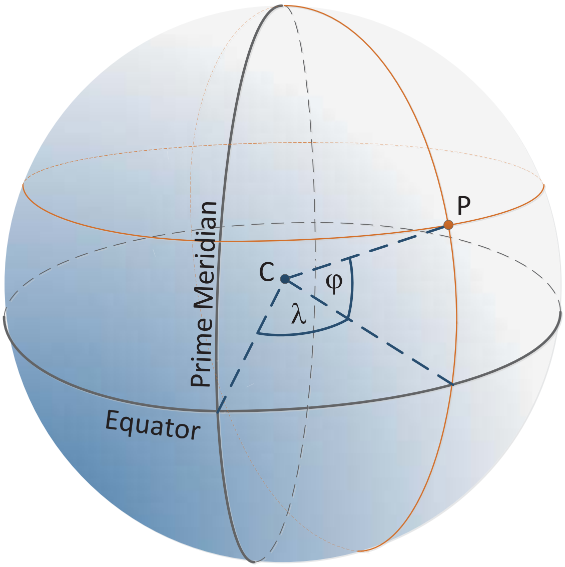


Figure 2.24 – Point P with Latitude φ and Longitude λ

Since the Earth is flattened at the poles by $\sim 21\,\text{km}$ more accurate geodetic systems are based on oblate ellipsoids of revolution. The flattening f is defined as

$$f = \frac{a-b}{a} = 1 - \frac{b}{a} \tag{2.1}$$

a is the equatorial radius

b is the polar distance from the center

Table 2.3 gives an overview on global reference ellipsoids for Earth in chronological order. Following a decision of the ICAO, WGS84 is the commonly used reference ellipsoid in aviation since 1989.

Table 2.3 – Reference Ellipsoids for Earth

Name	Equatorial axis a(m)	Polar axis b(m)	Inverse flattening $1/f$
Spherical	6 371 000	6 371 000	∞
Airy 1830	6 377 563	6 356 257	299.32
Bessel 1841	6 377 397	6 356 079	299.15
International 1924	6 378 388	6 356 912	297.00
WGS 72	6 378 135	6 356 751	298.26
WGS 84, GSR 80	6 378 137	6 356 752	298.257

2.6.1 Distances on Earth

The shortest distance between two points on a sphere is an orthodrome, a segment of a great circle. A great circle is the intersection of a sphere with a plane going through the center of the sphere. Non identical two points p_1, p_2 which are not exactly on opposite sides of the sphere define exactly one great circle. The minor arc is the shortest distance between the two points along the sphere's surface. The distance between p_1 and p_2 can be predicted based on the inner angle between the polar vectors at the sphere center by multiplication with the sphere's radius.

Calculation of shortest distance on an ellipsoid's surface is significantly more difficult. Thaddeus Vincenty's formula performing an iteration towards the ellipsoidal distance is often used in aviation (Vincenty, 1975). Vincenty's formula works for any oblate ellipsoids of revolution and thus can easily be adapted to WGS84. Even though the algorithm is an iterative approach, calculation time is fast because of quick convergence. Depending on the requested accuracy, only few iteration steps are necessary, typically well below 10.

2.6.2 Map Projection

This section describes some widely used map projection techniques to map a spherical/ellipsoidal Earth system on a 2D plane. Chapter 4 explains how the conflict detection algorithm of chapter 3 is adapted for usage with geodetic Earth systems based on these mappings. There is no optimum projection, every mapping creates distortions.

2.6.2.1 Sinusoidal Projection

The sinusoidal projection (also known as Mercator equal-area projection, fig. 2.25) is a pseudo-cylindrical projection preserving the area (Snyder, 1987). It shows relative sizes accurately, but distorts shapes and directions. The transformation from sphere (φ, λ) into plane (x, y) coordinates is simple:

$$x = (\lambda - \lambda_0) \cdot \cos(\varphi), y = \varphi \tag{2.2}$$

λ_0 is the central meridian

2.6.2.2 Mercator Projection

The Mercator projection (fig. 2.26) is a cylindrical map projection. Since it represents loxodromes (i. e., lines crossing all meridians of longitude at the same angle) with constant course, it is the standard map projection for nautical purposes. The Mercator projection preserves well angles and shapes of small objects, but distorts larger objects. The scale increases from the Equator to the poles, where it is infinite.

$$x = (\lambda - \lambda_0), y = \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right] \tag{2.3}$$

λ_0 is the central meridian

2.6.2.3 Transverse Mercator Projection

The transverse Mercator projection (sometimes also referred to as Gauss-Krüger projection, fig. 2.27) is a variant of the Mercator projection. It features a freely adjustable central meridian and thus constructs local high accuracy maps all around the globe.

$$x = \frac{1}{2} k_0 \ln \left[\frac{1 + \sin\lambda \cos\varphi}{1 - \sin\lambda \cos\varphi} \right], y = k_0 \arctan\left(\sec\lambda \tan\varphi\right) \qquad (2.4)$$

$k_0 = \sec\varphi$

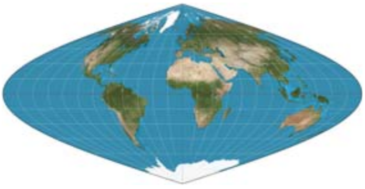


Figure 2.25 – Sinusoidal Projection

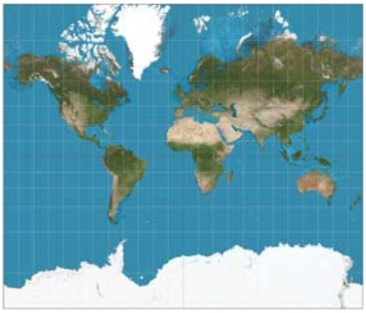


Figure 2.26 – Mercator Projection

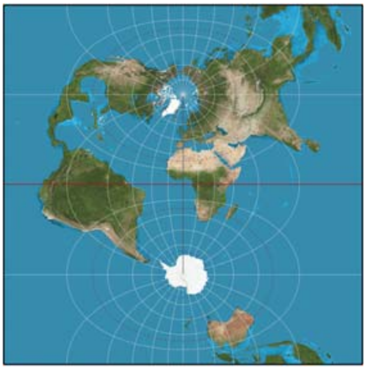


Figure 2.27 – Transverse Mercator Projection

Chapter 3

Conflict Detection

Efficient algorithms are required to perform conflict detection for complex problems in a reasonable time. Several algorithms already exist realizing high performance conflict detection, but usually come from different application areas with different environmental constraints.

The constraints for conflict detection in aviation are:

- Objects (e. g., aircraft) and their occupied space are small compared to the containment area (i. e. airspace). Typical shapes of occupied space are cylinders with diameters of 5 NM and heights of 1000 ft (fig. 1.1 on page 25). Although the underlying non-linear coordinate system increases effort, overall collision detection in narrow phase is simple.
- Aircraft are typically staggered vertically with valid 1000 ft separation. Approximation of the separation cylinder with a sphere would result in a high false-positive rate.
- Aircraft move fast in airspace, typically resulting in conflicts with short durations. Aircraft move on continuous trajectories. Due to their high speed compared to their size discretization of time should be avoided.

- 4D airspace is sparsely filled with objects only. However, designated areas of airspace are crowded, especially areas in the vicinity of airports.
- No-fly zones with complex 4D shapes are additional hazards in airspace that shall be respected.
- Different conflict metrics may apply for different parts of airspace.
- WGS84 distance metrics are complex to calculate (section 2.6).
- Even though the trajectories of aircraft are assumed to be well-known for whole flights, updates of trajectories should be possible with low costs.
- The expected number of conflicts in aviation scenarios is usually low. Even a dense scenario with unaligned trajectories usually yields less than one conflict per aircraft on average (compare table 4.2 on page 143).
- The algorithm should allow to dynamically add (e. g., new scheduled departures) and delete (e. g., canceled flights) objects.

Since already pairwise comparisons of trajectories are computationally expensive, an efficient broad phase avoiding pairwise conflict detections in the first place is desirable. BVH as described in section 2.2.7 allows a very fast first comparison between two trajectories based on their bounding boxes, but does not skip the comparison completely.

Applying KDS from section 2.2.8 requires the generation of certificates and prediction of events for aircraft. Furthermore, a KDS implementation for three dimensions seems to be ambitious (no known 3D-implementation yet), especially with cylindrical shaped objects.

Also the sweep and prune approaches described in section 2.2.9 are not reasonable for the given problem. Handling the dimensions one-by-one would result in a large number of altitude intersections each generating an event even if the objects are laterally well separated.

The 3D subdivision described in section 2.4.3 using octrees is promising, but a static subdivision of worldwide airspace with a reasonable memory equipped computer would allow a flat tree only. Furthermore, it is not intuitive to skip one dimension (Koeners and de Vries skipped altitude while Hildum and Smith skipped time), as long as no memory constraints exist.

The algorithm developed in this thesis is based on N-dimensional bisection. The fundamental idea is based on an early exclusion of conflict checking whenever a conflict is obviously impossible. In 4D-airspace, a conflict check may be excluded if two aircraft are separated laterally (fig. 3.1). Separation is ensured either in x-direction, y-direction, or a combination of both. If two aircraft fly in significantly different altitudes, they are separated vertically (fig. 3.2). Finally, aircraft are allowed to fly exactly the same route with exactly the same vertical profile if they only fly at different times (fig. 3.3).

3.1 Definitions

This section gives some definitions for basic objects referred to within the following chapters.

A *line segment* in N-dimensional space is defined as a point set by two points $\vec{p}_1$ and $\vec{p}_2$:

$$L(\vec{p}_1, \vec{p}_2) = \{\vec{x} : \vec{x} = \vec{p}_1 + \mu\,(\vec{p}_2 - \vec{p}_1)\} \tag{3.1}$$

$\vec{x}, \vec{p}_1, \vec{p}_2 \in \mathbb{R}^N$

$0 \leq \mu \leq 1, \mu \in \mathbb{R}$

A *trajectory* is defined as a set of connected line segments by k points $\vec{p}_1, \ldots, \vec{p}_k$:

$$T(\vec{p}_1, \ldots \vec{p}_k) = L(\vec{p}_1, \vec{p}_2) \cup L(\vec{p}_2, \vec{p}_3) \cup \ldots \cup L(\vec{p}_{k-1}, \vec{p}_k) \tag{3.2}$$

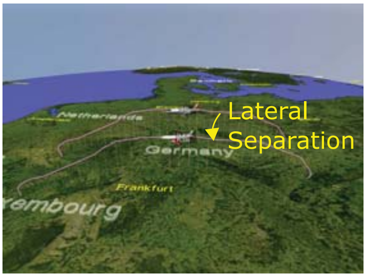


Figure 3.1 – Lateral Separation between 4D-Trajectories

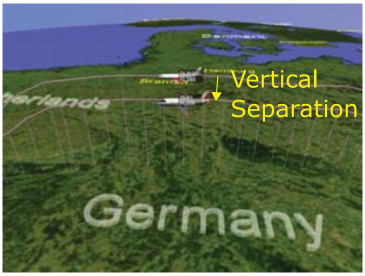


Figure 3.2 – Vertical Separation between 4D-Trajectories

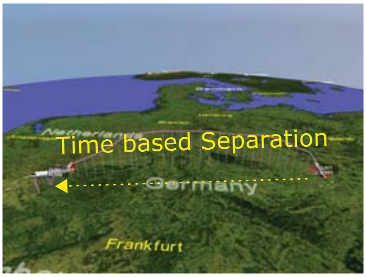


Figure 3.3 – Time-based Separation between 4D-Trajectories

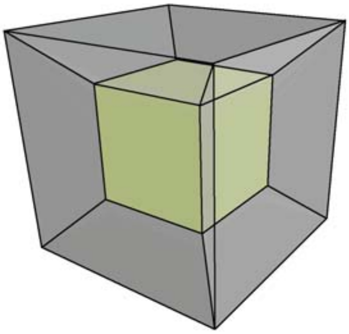


Figure 3.4 – 4D-Hypercube (Tesseract)

$$\vec{p}_1, \ldots, \vec{p}_k \in \mathbb{R}^N$$
$$k \geq 2, k \in \mathbb{N}$$

A trajectory T is one of two basic object types that are supported for conflict detection. The other *object* type is a volume V. We will refer to an object O as either a trajectory T or a volume V.

A volume V is based on simplices, the N-dimensional analogue of a triangle with $(N+1)$ vertices. Well known simplices are points (0-simplex), line segments (1-simplex), triangles (2-simplex), and tetrahedrons (3-simplex). A *simplex* $\mathfrak{S}$ can be defined as the convex combination of $(N+1)$ affine independent points as

$$\mathfrak{S}(\vec{p}_1, \ldots, \vec{p}_{N+1}) = \left\{ \vec{x} : \left(\vec{x} = \sum_{i=1}^{N+1} \mu_i \vec{p}_i \right) \wedge \left(\sum_{i=1}^{N+1} \mu_i = 1 \right) \right\} \tag{3.3}$$

$$\vec{x} \in \mathbb{R}^N$$
$$\vec{p}_1, \ldots, \vec{p}_{N+1} \in \mathbb{R}^N : det(\vec{p}_2 - \vec{p}_1, \vec{p}_3 - \vec{p}_1, \ldots, \vec{p}_{N+1} - \vec{p}_1) \neq 0$$
$$0 \leq \mu_i \leq 1, \mu_i \in \mathbb{R}$$

A simplex s_1 is an N-face of a simplex s_2 if it is equal to one N-face of s_2.

$$\text{NFace} : \mathfrak{S} \times \mathfrak{S} \to \{0, 1\} \tag{3.4}$$

$$\text{NFace}(s_1(\vec{q}_1, \ldots, \vec{q}_N), s_2(\vec{p}_1, \ldots, \vec{p}_{N+1})) = \begin{cases} 1, & \text{if } s_1(\vec{q}_1, \ldots, \vec{q}_N) = \\ & \left\{ \vec{x} : \left(\vec{x} = \sum_{i=1}^{N+1} \mu_i \vec{p}_i \right) \wedge \left(\sum_{i=1}^{N+1} \mu_i = 1 \right) \wedge (\exists j : \mu_j = 0) \right\} \\ 0, & \text{else} \end{cases} \tag{3.5}$$

$$\begin{aligned}\vec{x} &\in \mathbb{R}^N \\ j &\in \{1, \ldots, N+1\} \\ \vec{p}_1, \ldots, \vec{p}_{N+1} &\in \mathbb{R}^N \\ \vec{q}_1, \ldots, \vec{q}_N &\in \mathbb{R}^N \\ 0 \leq \mu_i &\leq 1, \mu_i \in \mathbb{R}\end{aligned}$$

Finally, a *volume* V is defined as a homogeneous simplicial N-complex recursively by a set of simplices connected by faces of dimension N-1:

$$V(s_1, \ldots, s_k) = \begin{cases} s_1, & \text{if } k = 1 \\ V(s_1, \ldots, s_{k-1}) \cup s_k, & \text{if } k > 1 \end{cases} \tag{3.6}$$

$$\begin{aligned}\exists j : (\text{NFace}(s_j \cap s_k, s_j) = 1) \wedge (\text{NFace}(s_j \cap s_k, s_k) = 1) \\ s_1, \ldots, s_k \in \mathfrak{S}(\vec{p}_1, \ldots, \vec{p}_{N+1}) \\ j \in \{1, \ldots, k-1\}\end{aligned}$$

The bisection algorithm described in this chapter operates on tiles. A tile A is a subset of a N-dimensional Euclidean space defined by two vectors holding minimum ($\vec{a}_{\min}$) and maximum ($\vec{a}_{\max}$) values for each dimension.

$$A(\vec{a}_{\min}, \vec{a}_{\max}) = \{\vec{x} : (a_{\min,i} \leq x_i \leq a_{\max,i}) \text{ for all } i \in \{1, \ldots, N\}\} \tag{3.7}$$

$\vec{x}, \vec{a}_{\min}, \vec{a}_{\max} \in \mathbb{R}^N$

A similar definition can be given by the Cartesian product of N intervals:

$$A(\vec{a}_{\min}, \vec{a}_{\max}) = [a_{\min,1}, a_{\max,1}] \times [a_{\min,2}, a_{\max,2}] \times \ldots \times [a_{\min,N}, a_{\max,N}] \tag{3.8}$$

One dimension of an object can be defined to be a time dimension. If a time dimension is specified, let it be assigned to the last dimension N of an object without loss of generality. Time τ then serves as a common reference for conflict detection. Therefore, a trajectory needs to be one-dimensional along time axis and capable of being parameterized by a time τ.

$$\exists_1(\vec{x} \in T) : x_N = \tau \tag{3.9}$$

$$\vec{x} \in \mathbb{R}^N$$
$$a_{\min,N} \leq \tau \leq a_{\max,N}$$

This restriction does not account for volumes though, volumes can be $(N-1)$-dimensional along time. The functions $V(\tau)$ and $T(\tau)$ correspond to the volume/trajectory for the given time τ

$$V(\tau) = \{V(s'_1, \ldots, s'_m) \subset V(s_1, \ldots, s_k) : s'_{i,N} = \tau \quad \text{for all } i \in \{1, \ldots, m\}\} \tag{3.10}$$

$$T(\tau) = \{\vec{x} \in T : x_N = \tau\} \tag{3.11}$$

$$\vec{x} \in \mathbb{R}^N$$

$O(\tau)$ resolves to $T(\tau)$ or $V(\tau)$ for trajectories or volumes, respectively. The result of $O(\tau)$ has always full dimension N with the Nth dimension being constantly equal to τ in order to avoid different dimensions in calculations.

3.2 N-Dimensional Conflict

This section defines if two objects are in conflict. It needs to be distinguished if a time reference shall be taken into account, or if all dimensions are treated equally.

3.2.1 Conflicts without Time Reference

Two trajectories $t_1, t_2 \in T$ are in conflict if the separation of trajectories is lower than a predefined mandatory separation $\vec{S}$ in all N dimensions.

$$\exists(\vec{p}_1 \in t_1)\exists(\vec{p}_2 \in t_2) : (|\vec{p}_1 - \vec{p}_2|_i < S_i) \text{ for all } i \in \{1, \dots, N\} \tag{3.12}$$

$\vec{p}_k \in \mathbb{R}^N$, point on trajectory t_k

$\vec{S} \in \mathbb{R}^N$, mandatory separation

A volume $v_1 \in V$ is in conflict with an object o_1 if v_1 contains partly o_1.

$$\exists \vec{p}_1 : (\vec{p}_1 \in o_1) \wedge (\vec{p}_1 \in v_1) \tag{3.13}$$

3.2.2 Conflicts with Time Reference

If a time axis is specified within the N-dimensional vector, it is used as a common reference of all objects. Two trajectories $t_1, t_2 \in T$ are in conflict, if their distance at a common time τ is lower than the necessary separation $\vec{S}$ in all N dimensions.

$$\exists \tau : |t_1(\tau) - t_2(\tau)|_i < S_i \text{ for all } i \in \{1, \dots, N\} \tag{3.14}$$

$\tau \in \mathbb{R}$, common time

$\vec{S} \in \mathbb{R}^N$, mandatory separation

A volume $v_1 \in V$ is in conflict with an object $o_1 \in (T \cup V)$ if v_1 contains partly o_1 at a common time τ.

$$\exists \tau, \vec{p}_1 : (\vec{p}_1 \in o_1(\tau)) \wedge (\vec{p}_1 \in v_1(\tau)) \tag{3.15}$$

$\tau \in \mathbb{R}$, common time

The separation in time dimension is zero by definition. Therefore, time separation does not need to be checked explicitly. Nevertheless, the time component of $\vec{S}$ shall be positive.

Obviously, a conflict with time reference is also a conflict without time reference for $S_N > 0$ because the difference in time dimension is zero. Therefore, data with time reference can be treated with both algorithms. Checking conflicts with time reference is a specialization of the standard check without time reference.

Setting $S_N > 0$ to an arbitrary small epsilon

$$S_N = \epsilon \tag{3.16}$$

$\epsilon > 0$

provides identical results with and without time reference. However, there are good reasons to consider time separately:

- The mandatory separation $\vec{S}$ is also used as minimal tile size for the bisection algorithm. Simply setting one dimension to ϵ produces unwanted results.
- Having a common reference between all objects can be used beneficially as a speed-up by reducing necessary comparisons.
- Based on monotony of time dimension binary search can be applied.
- The conflict tree size can be reduced significantly in time reference mode (section 3.3.3).

3.3 N-Dimensional Tiling Algorithm

Detecting N-dimensional conflicts in a large set of objects is a time consuming task. For example, Germany's airspace is an operation area of more than 10 000 flights on busy days, each flight represented by a high-precision 4D-trajectory with several hundred sampling points. Comparing each trajectory against each other without any optimization produces unacceptable response times.

In order to avoid extensive conflict detection, the provided algorithm separates objects by generating N-dimensional tiles holding just one object. The tiles are organized in a tree structure. Depending on the number of dimensions N, one node of the tree has up to 2^N children.

Starting condition for the tiling algorithm is an N-dimensional orthogonal tile being large enough to hold all objects in all dimensions in terms of a convex hull. Objects are treated by the algorithm consecutively. In the beginning, this root tile is empty.

Objects are inserted in the root tile. The conflict detection is performed directly on insertion. As soon as a tile (e. g., the root tile) is affected by more than one object, the tile is subdivided in all dimensions by adding up to 2^N children. Figure 3.5 illustrates the subdivision process for the one-, two-, and three-dimensional cases. Tile shape for the 4D-case is a 4D-hyperrectangle, its cubic version also known as tesseract(fig. 3.4). Table 3.1 gives an overview on tile shapes and their subdivision.

Table 3.1 – Overview on N-dimensional Bisection

Dimension	Tile shape	Number of tiles in depth			
		1	2	3	m
1	Line	2	4	8	2^m
2	Rectangle	4	16	64	4^m
3	Rectangular box	8	64	512	8^m
4	4D-Hyperrectangle	16	256	4096	16^m
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n-D Hyperrectangle	2^n	$2^{2\cdot n}$	$2^{3\cdot n}$	$2^{m\cdot n}$

Having in mind the aviation background, the 4-dimensional case is of particular interest. Therefore, fig. 3.6 depicts the sixteen first-level children of a tesseract.

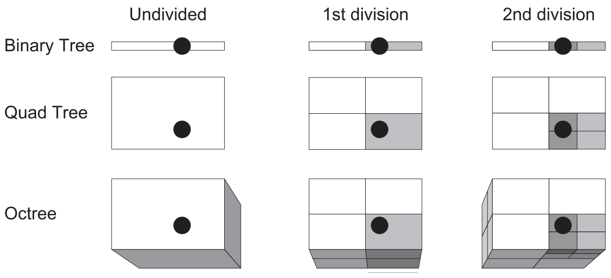


Figure 3.5 – Bisection for 1-3 Dimensions

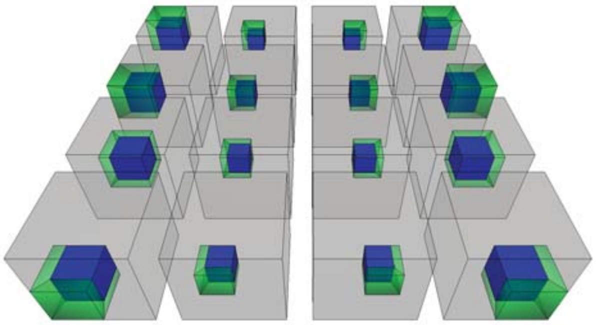


Figure 3.6 – First Level Bisection of Tesseract with 16 Children

The subdivision is performed until all leaf tiles:

- Are affected by one object only; or
- The minimum tile size is reached. Reaching the minimum tile size with more than one object is interpreted as a potential conflict. Therefore, the minimum tile size should be set to conflict size or above in order to detect all conflicts. Section 3.3.10 explains how minimum tile size should be set.

3.3.1 Affected Tiles

Due to the orthogonal layout, a tile can be described by two N-dimensional vectors $\vec{a}_{\min}$ and $\vec{a}_{\max}$ (eq. (3.7)). The tile size $\vec{\delta}$ is the difference:

$$\vec{\delta} = \vec{a}_{\max} - \vec{a}_{\min} \tag{3.17}$$

$\vec{\delta}, \vec{a}_{\min}, \vec{a}_{\max} \in \mathbb{R}^N$

A tile is subdivided only if it is affected by more than one object. Affection can be both:

- Penetration of a tile by an object (fly-through, section 3.3.2); or
- Vicinity of a tile to an object (fly-by, section 3.3.3).

If an object's distance to a tile is closer than mandatory separation distance, it affects the tile.

Figure 3.7 demonstrates a situation with two objects in different tiles having a conflict. Both tiles contain a penetrating fly-through and a fly-by object.

In this algorithm, fly-by objects are directly stored in the fly-by tiles on subdivision. Compared to other algorithms in literature (e. g., section 2.4.3) where all neighboring tiles need to be investigated, this procedure has three main advantages:

- It is not necessary to store neighborhood information. A hyperrectangle of dimension N has $3^N - 1$ direct/diagonal

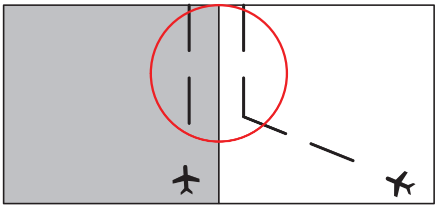


Figure 3.7 – Objects in Different Tiles having a Lateral Conflict

neighbors. For a 4D map, this would increase necessary memory by $3^4 - 1 = 80$ pointers per tile – 640 B on 64 bit computer.

Identifying neighbors dynamically, without storing them, is not an option since neighbor search is too slow.

- Keeping neighborhood information consistent is a complex task and computational expensive in a dynamic heterogeneous tree.

- In contrast to a static array, it is not feasible to check for conflicts against all candidates from neighboring tiles in a dynamic tree. After the first subdivision of root tile, every tile is a direct neighbor of every other tile. Thus, conflict check has to be limited on objects close to the border.

3.3.2 Check for Penetration

A tile A is penetrated by an object O if at least one point of O is within A.

$$\exists(\vec{p} \in O) : (\vec{p} \in A) \tag{3.18}$$

This equation is good for the sake of mathematical definition, but since objects are defined as infinite point sets, it cannot be checked directly by a computer program.

Due to the orthogonality of the underlying coordinate system, penetration of a tile by an object can be re-formulated allowing an implementation on a computer. An object penetrates an N-dimensional tile if:

- All points of the object are within the tile; or
- At least one boundary of the object intersects with one of the $2 \cdot N$ tile boundaries; or
- The object contains the complete tile (only relevant for volumes).

Bullet one is easy to verify. Only one (e. g., the first) point $\vec{p}$ of the object needs to be checked.

$$\forall i: \; a_{\min,i} \leq p_i \leq a_{\max,i} \tag{3.19}$$

$i \in \{1, \ldots, N\}$
$\vec{p} \in \mathbb{R}^N$

If the tile encloses the test point $\vec{p}$, the object obviously penetrates the tile A. If the result is negative, not all points of O lie in the tile.

Bullet three can be checked with comparable effort. Only one point $\vec{p}$ (e. g., $\vec{a}_{\min}$) of the tile needs to be checked to be within a volume V.

$$(\vec{p} \in V(s_1, \ldots, s_k)) \Leftrightarrow \exists i : \vec{p} \in s_i \tag{3.20}$$

$i \in \{1, \ldots, k\}$

Again, if the volume contains the point $\vec{p}$, the volume obviously penetrates the tile A. If the result is negative, not all points of A are located within the object.

If both checks are negative, the intersection of tile and object boundaries need to be checked. For each dimension, an N-dimensional tile has a lower and an upper boundary. Since tiles are axis-aligned, the shape of tile boundaries has one dimension less than the tile itself (table 3.2).

Table 3.2 – Boundaries of N-dimensional Tiles

Dim.	Tile shape	Boundary shape	Number of boundaries
1	Segment	Point	2
2	Rectangle	Segment	4
3	Rectangular box	Rectangle	6
4	4D-Hyperrect.	Rectangular box	8
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n-D Hyperrect.	$(n$-1)-D Hyperrect.	$2 \cdot n$

The final check whether object boundaries intersect with the tile depends on the object type. The corresponding algorithms are explained in section 3.4.1 for trajectories and within section 3.4.2 for implemented volumes.

3.3.3 Check for Vicinity

The vicinity test can be performed easiest by using a dilatation of the tile. Instead of checking distances to a given tile, the tile is extended by the separation margins. Figure 3.8 depicts a trajectory that penetrates both the tile itself and the extended fly-by tile. Based on the two N-dimensional vectors $\vec{a}_{\min}$ and $\vec{a}_{\max}$ describing the tile, the dilatation tile can be described by

$$\vec{a}'_{\min} = \left(\vec{a}_{\min} - \vec{S}\right) \text{ and } \vec{a}'_{\max} = \left(\vec{a}_{\max} + \vec{S}\right) \tag{3.21}$$

$$a_{\min}, a_{\max}, a'_{\min}, a'_{\max} \in \mathbb{R}^N$$
$$\vec{S} \in \mathbb{R}^N, \text{mandatory separation}$$

Thus, vicinity check can be performed as described in section 3.3.2 using $\vec{a}'_{\min}$ and $\vec{a}'_{\max}$.

In time reference mode, a dilatation of time is not necessary because proximity in time is not relevant for the conflict decision.

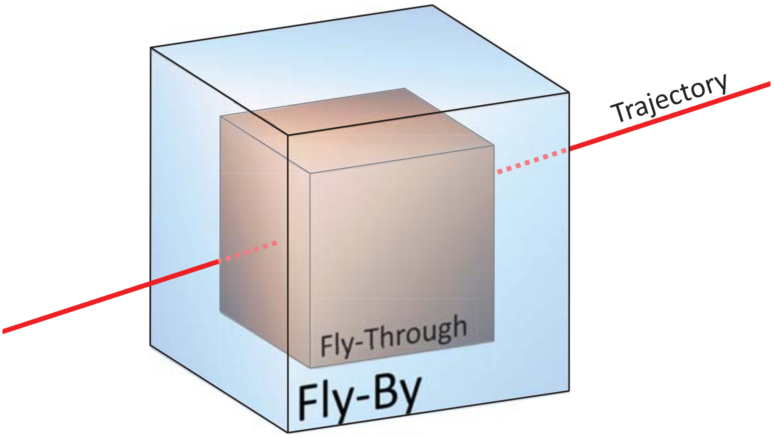


Figure 3.8 – Trajectory in Fly-Through and Fly-By Zone

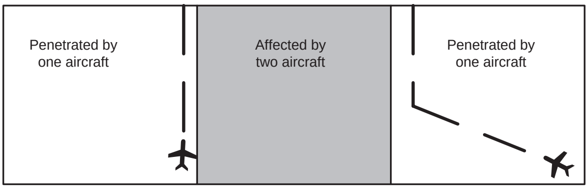


Figure 3.9 – No Conflict without Fly-Through Object in Center Tile

Objects are only compared at identical times τ. Avoiding the dilatation in time provides a significant decrease of used memory and significant increase of calculation speed. For the aviation example, this results in ~25 % better run-time and ~40 % less memory. Therefore, we set for the time reference mode

$$a'_{\min,N} = a_{\min,N} \text{ and } a'_{\max,N} = a_{\max,N} \tag{3.22}$$

A tile contains a potential conflict only if:

- It is affected by at least two objects; and
- Penetrated by at least one object.

A tile with fly-by objects, only, does not contain any conflicts (fig. 3.9).

3.3.4 Symmetric Simplification

Figure 3.7 on page 93 illustrates that conflicts are detected twice. Both tiles displayed contain a fly-by and a fly-through object, each. While this is technically not a problem since detected conflicts are merged, symmetric situations are both memory and time consuming. Thus, the idea of this section is avoiding symmetric situations by skipping the fly-by object in one of the two tiles.

A clear rule needs to be defined in order to ensure that fly-by situations are at least taken into account by one tile. The implemented method skips fly-by objects by their relative position to the tile. For each dimension i it is checked whether the tile-relevant part of the object is below the corresponding $a_{\min,i}$ or above $a_{\max,i}$. If no dimension is found where the object is constantly below or above the corresponding tile dimension, the fly-by object is kept, resulting in a potential extra effort.

Otherwise, the fly-by object is kept only if it is constantly above the maximum tile value for at least one dimension. Focusing on the left tile in fig. 3.7, the right trajectory is constantly above the maximum x-value of the left tile. Concerning the right tile, the left trajectory is constantly below the x-value of the right tile and thus is omitted. For the aviation example, this technique improves run-time by ∼15 % and decreases memory usage by ∼25 %.

3.3.5 Full Containment

If a volume V contains one point of a tile A and volume's boundaries do not intersect the tile's boundaries, the volume V includes the complete tile.

$$\forall \vec{x} \in A \Rightarrow \vec{x} \in V \tag{3.23}$$

Every other object also penetrating this tile has a real conflict with the volume V. No extra information concerning volume V is gained with further subdivision of the appropriate internal tile. Therefore, volumes containing whole tiles can be treated as a fly-by object for the tiling decision.

For example, if only one aircraft flies in German airspace, and the trajectory is checked for conflict against a volume representing German airspace, subdivision only needs to be done until one tile is completely inside German airspace and also contains the trajectory.

Figure 3.10 depicts two aircraft flying through the volume object "German airspace". The southern aircraft enters and leaves Germany at the east-boundaries. The northern aircraft starts within the German volume, and leaves the volume through the upper boundary. Once a tile contains a trajectory and the tile is completely inside the German volume, it is not necessary to perform further subdivision, although the tile technically is penetrated by two objects.

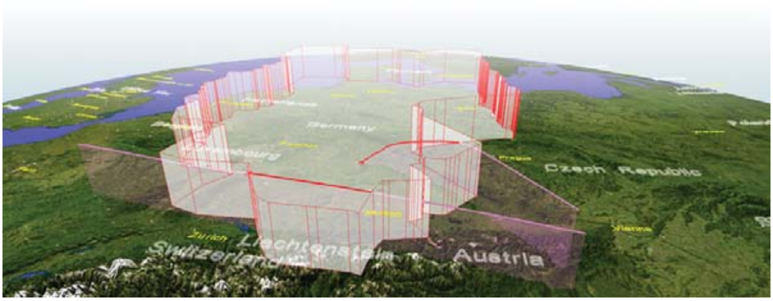


Figure 3.10 – Two Trajectories in Conflict with the Volume German Airspace

3.3.6 Bounding Boxes

Check for penetration can be further sped up using bounding boxes. A bounding box B_O of an object O is a closed volume that completely contains the object O.

$$\forall \vec{x} \in O \Rightarrow \vec{x} \in B_O \tag{3.24}$$

Two objects O_1 and O_2 do not intersect, if their bounding boxes do not intersect.

$$(B_{O_1} \cap B_{O_2} = \emptyset) \Rightarrow (O_1 \cap O_2 = \emptyset) \tag{3.25}$$

In order to exclude intersections as early as possible, it is beneficial to construct small bounding boxes.

Amongst others (e. g., spherical, eight-direction discrete orientation polytope, and the convex hull, compare (Ericson, 2005)), two different types of bounding boxes are commonly used:

- Axis aligned bounding boxes have a simple representation, can be calculated fast, and containment of a point $\vec{p}$ can be identified with $2 \cdot N$ comparisons very efficiently. Depending on the orientation of the object, axis aligned bounding boxes can be much larger than the object.

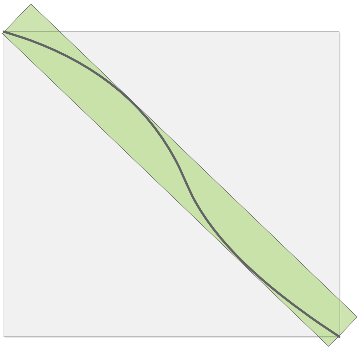


Figure 3.11 – Axis Aligned vs. Object Oriented Bounding Box

- Object oriented bounding boxes have a more complex representation, are more difficult to calculate, and containment check is slower. Object oriented bounding boxes approximate an object usually with a smaller volume than axis aligned bounding boxes.

Figure 3.11 shows an example for both axes aligned (light grey) and object oriented bounding box (light green) types in 2D. If objects' sizes are big compared to distances between objects, and they are usually not aligned to axis, object aligned bounding boxes might be beneficial. This approach focuses on the simpler and faster axis aligned bounding boxes:

$$\begin{aligned} B_O(b_{\min}, b_{\max}) = \\ \{\vec{x} \in O : (b_{\min,i} \leq x_i \leq b_{\max,i}) \text{ for all } i \in \{1, \ldots, N\}\} \end{aligned} \tag{3.26}$$

Conflicts can be excluded early by a non-intersection of bounding boxes. Since the geometry of a bounding box is similar to that of a tile, comparisons can be performed inter objects and between objects and tiles. Two bounding boxes $b1$ and $b2$ do not intersect if

$$\exists i : (b1_{\max,i} < b2_{\min,i}) \vee (b1_{\min,i} > b2_{\max,i}) \tag{3.27}$$

$i \in \{1, \ldots, N\}$

3.3.7 Building the Tree

Task of the conflict tree is to represent all objects in a consistent way supporting a fast detection of separation violations.

Concerning the implementation, the conflict detection module holds separately:

- A list of all objects.
- The tree structure with references to the objects.
- A list of conflicts between stored objects.

A major invariant of the software is that all entries are consistent when the module is accessible from the outside. Therefore, when a new object is added to the tree, the object is added to the internal object list, all necessary tiles are generated immediately, and finally the list of conflicts is updated.

When a tile needs to be subdivided because it is affected by at least two objects with at least one penetration, all 2^N children need to be checked for affection by the new object. Some of the children tiles might already exist from earlier inserted objects, all other affected tiles need to be generated. This recursive process is performed until further subdivision is not required or separation size is reached.

If a leaf is completely inside of a/some volume(s) (section 3.3.5), a conflict with the volume(s) is detected. If one of the leaves has minimum tile size $\vec{S}$ a potential conflict is detected with all other objects of the tile (section 3.5). Otherwise, the object is proven to be free of conflict.

3.3.8 Monotonic Dimensions

When adding a new object, all dimensions are checked for monotony. If a dimension increases or is partly constant with increasing index, the dimension is monotonically increasing. If a dimension decreases or is partly constant with increasing index, the dimension is monotonically decreasing. In time reference mode, the time dimension of an implemented trajectory object is always strictly increasing due to eq. (3.9). But also other dimensions can be monotonic, for instance a trajectory going strictly from West to East has a monotonic longitudinal axis.

When identified, monotony can be used beneficially for speeding up intersection calculation with an axis-oriented plane. Instead of checking each trajectory segment to have vertices on opposite sides of the plane, binary search can be applied to identify the correct segment directly.

3.3.9 Balancing the Tree

When configuring the tree, dimensions are usually independent. Value range and mandatory separation might differ significantly between dimensions. Dimensions can even have different physical units. The user has to ensure that units are used consistently for each dimension. Based on the starting interval $I = [\vec{I}_{\min}, \vec{I}_{\max}]$ and the separation $\vec{S}$, the number of subdivisions $\vec{D}$ can be defined as

$$D_i = \log_2 \left(\left(I_{\max,i} - I_{\min,i} \right) / S_i \right) \tag{3.28}$$

$$\begin{aligned} \vec{D}, \vec{I}_{\min}, \vec{I}_{\max}, \vec{S} &\in \mathbb{R}^N \\ i &\in \{1, \dots, N\} \end{aligned}$$

If the resulting vector has a big range $\max(D_i) - \min(D_i)$, the minimum tile size is reached in different tree depths. Although this is technically not a problem, such behavior produces an unbalanced tree that subdivides deeper tiles in a subset of dimensions only.

If such behavior is unwanted, the software provides an automatic balancing functionality. When the interval I and minimum tile size

$\vec{S}$ is set, starting intervals are adapted in order to reach minimum tile size in every dimension at the same subdivision step.

For example, when trying to model one day of the Earth in 4 dimensions latitude, longitude, altitude and time, the setup of table 3.3 can be used. For this example, special properties of Earth

Table 3.3 – Root Tile covering one Day on Earth

Dim.	Unit	Min	Max	S	(Max-Min)/S	D_i
x	Deg	-180	180	$^5/_{60}$	4320	12.08
y	Deg	-90	90	$^5/_{60}$	2160	11.08
z	FL	-10	500	10	51	5.67
t	s	0	86 400	90	960	9.91

are not taken into account, see chapter 4 for details. A balanced starting condition can be achieved by adapting the initial ranges in order to reach the separation minima for all dimensions within the same step. The intervals are extended on one side only to avoid additional sub-trees. Being extended, only the original part of the interval will contain flights. Table 3.4 shows the same setup after balancing.

Balancing the tree has negative effects on memory usage and conflict detection times in practice (section 4.5.4 on page 152). However, balancing makes the tree look more natural, and therefore is beneficial, e. g., for debugging and visualization purposes.

Table 3.4 – Balanced Root Tile holding one Day on Earth

Dim.	Unit	Min	Max	S	(Max-Min)/S	D_i
x	Deg	-180	180	$^5/_{60}$	4320	12.08
y	Deg	-90	270	$^5/_{60}$	4320	12.08
z	FL	-10	43 190	10	4320	12.08
t	s	0	388 800	90	4320	12.08

3.3.10 Broad Phase vs. Narrow Phase

The N-dimensional tiling algorithm performs the broad phase and does not identify real conflicts but potential conflicts, only. The final conflict check needs to be performed afterwards in the narrow phase. Thus, all real conflicts are handled twice:

- First by the tiling algorithm with low costs C_T.
- Second by the final conflict check that might be very expensive in terms of calculation costs C_F.

If the likelihood of conflicts P_C is low, exclusion of many non-conflicting situations is beneficial. Conflict likelihood increases when coming further down in the tiling tree. The total costs C calculate as

$$C = P_C \cdot (C_T + C_F) + (1 - P_C) \cdot C_T \tag{3.29}$$

$$P_C \in \mathbb{R}, 0 \leq P_C \leq 1 \text{ (conflict likelihood)}$$
$$C_T, C_F > 0, C_T \ll C_F \text{ (costs)}$$

Using the tiling algorithm is beneficial as long as C is lower than C_F. Obviously this strongly depends on $\frac{C_T}{C_F}$ and P_C. Therefore, the size of the smallest tile does not necessarily need to be the conflict detection size. If conflicts have a high likelihood and $\frac{C_T}{C_F}$ is close to one, smallest tile size should be bigger than the mandatory separation size. Therefore, the tiling algorithm is called with an increased requested separation $\vec{S}$, while the final conflict check (section 3.5) uses the original separation.

Especially in time reference mode (section 3.2.2) time separation can be freely adjusted in order to achieve best possible detection performance.

3.3.11 Memory Limitation

The bisection algorithm in its basic version performs a time-memory tradeoff. In order to reach fast computation times, it uses a tremendous amount of memory. Depending on the problem and computer hardware, the algorithm might exceed physical available memory. Therefore, two mechanisms are implemented helping to adjust the tradeoff:

- Static: By means of a max tree depth, the user can limit the refinement of the tree. The algorithm does not generate nodes that exceed the defined tree depth. This method is an indicator only to reduce memory size; it does not allow to explicitly name a memory limit.

- Dynamic: The algorithm provides a feature of monitoring memory usage against available system memory. If the ratio exceeds a given value (e. g., 70 %), it automatically decreases the depth of tree by one. Depending on the tree size and especially on the number of leaves, this operation might take some time. However, it prevents memory problems on reduced hardware. The tree depth reduction is performed every time the given ratio is exceeded.

3.3.12 Tile Knowledge

As already mentioned in section 3.3.7, each tile stores references to the objects affecting it including the type of affection (fly-through or fly-by, and if the object contains the complete tile). In order to increase algorithm's speed, more information is stored in tiles.

When verifying tile affection, the segments of trajectories are checked successively. Negative results can beneficially be used for further subdivisions. Trajectory segments not affecting a node will also not affect any of its children because every child is a real subset of its father.

Therefore, the interval affecting a tile is stored for each object. In the beginning, this interval is set to $[1, K]$ with K being the number of trajectory definition points. If possible this interval is reduced when going down the tree.

3.4 Supported Objects

The implemented software provides two objects in its current version: trajectories and a special implementation of volumes, the polygon volume. A variation of a polygon volume, the moving polygon volume, is described in section 3.4.3. Further objects can be defined easily if necessary.

3.4.1 N-Dimensional Trajectories

The trajectories supported by the module directly implement as a list of N-dimensional points connected by line segments, as defined in eq. (3.2), page 83. Concerning the penetration check defined in section 3.3.2, one boundary of the object intersects with one of the $2 \cdot N$ tile boundaries if an edge of the trajectory intersects with a tile boundary. Edges E can be identified for trajectories as the line segments from eq. (3.2). An object's edge E intersects with a tile boundary if corresponding vertices e_b and e_e are on opposite sides of the boundary

$$\exists i : (e_{b,i} - b_i) * (e_{e,i} - b_i) \leq 0 \tag{3.30}$$

$$\begin{aligned} i &\in \{1, \ldots, N\} \\ \vec{b} &\in \{\vec{a}_{\min}, \vec{a}_{\max}\} \\ \vec{e}_b, \vec{e}_e &\in \mathbb{R}^N \text{being begin and end of edge } E \end{aligned}$$

in one dimension and the intersection point $\vec{p}$ of the edge with the boundary $\vec{b}$ for the same dimension i

$$\vec{p} = \frac{b_i - e_{b,i}}{e_{e,i} - e_{b,i}} * \vec{e}_e + \left(1 - \frac{b_i - e_{b,i}}{e_{e,i} - e_{b,i}}\right) * \vec{e}_b \tag{3.31}$$

$$\begin{aligned} \vec{b} &\in \mathbb{R}^N \\ \vec{e}_b, \vec{e}_e &\in \mathbb{R}^N \text{being begin and end of edge } E \\ i &\in \{1, \ldots, N\} \end{aligned}$$

is within the tile in all dimensions.

If at least one edge intersects with a tile boundary, the object penetrates the tile. If:

- Not all points of the trajectory are in the tile; and
- No edge intersects with a tile boundary;

it does not penetrate the tile.

3.4.2 N-Dimensional Volumes

Volumes are not implemented as generic as in eq. (3.6). Reasons for defining volumes another way are:

- Simplicial complexes are not the most intuitive way to imagine a volume.
- Simplicial complexes have more degrees of freedom than a typical user needs.
- Segmentation of a volume into simplices is a non-trivial task for humans, especially for high dimensions.
- Intersection of simplicial complexes with tiles is complex for high dimensions.

Instead of generic simplicial complexes, the implementation is based on simple 2D-polygons $\mathfrak{P}$. We define 2D-polygons $\mathfrak{P}$ as a complex of 2-simplices (i. e., triangles, compare eqs. (3.3) to (3.6)):

$$\mathfrak{P}(s_1, \ldots, s_k) = \begin{cases} s_1, & \text{if } k = 1 \\ \mathfrak{P}(s_1, \ldots, s_{k-1}) \cup s_k, & \text{if } k > 1 \end{cases} \tag{3.32}$$

$$\exists j : (\text{NFace}(s_j \cap s_k, s_j) = 1) \wedge (\text{NFace}(s_j \cap s_k, s_k) = 1)$$
$$s_1, \dots, s_k \in \mathfrak{S}\,(\vec{p}_1, \vec{p}_2, \vec{p}_3)$$
$$j \in \{1, \dots, k-1\}$$

Every dimension higher than 2 is defined by an interval, being constant for all points of the 2D polygon. Thus, a polygon volume $\mathfrak{V}$ is defined as the Cartesian product

$$\mathfrak{V} = \mathfrak{P} \times \left[\vec{I}_{\min}, \vec{I}_{\max}\right] \tag{3.33}$$

$\vec{I}_{\min}, \vec{I}_{\max} \in \mathbb{R}^{\mathbf{N\text{-}2}}$

A volume as described above intersects with a tile if all $N-2$ interval dimensions intersect with the corresponding tile intervals, and the polygonal part intersects with the $2D$-part of the tile. Therefore, fastest way to exclude an intersection is to find a dimension j where

$$(I_{\min,j} > a_{\max,j+2}) \vee (I_{\max,j} < a_{\min,j+2}) \tag{3.34}$$

$$j \in \{1, \dots, N-2\}$$
$\vec{a}_{\min}, \vec{a}_{\max} \in \mathbb{R}^N$ being the tile limitations

If the tile includes all intervals, the polygon's outline needs to be checked. While the polygon's outline would need to be extracted explicitly from the point set eq. (3.32), the software implementation creates polygons from a given outline. Thus, the outline is explicitly available.

Since the polygon's outline has the same format as a (closed) 2D-trajectory (section 3.4.1), the same algorithms can be applied. Having in mind the check for one point of the volume being part of the tile, and for one point of the tile being within the volume (section 3.3.2) validation of intersection between tiles and polygon volumes are rather fast.

As written before (section 3.3.5), full containment is of interest for polygon volumes. Full containment can easily be identified. A tile is completely within a polygon volume if:

- One point of the tile is within the polygon volume; and
- The polygon volume does not intersect with the tile boundaries

Due to the application of airborne traffic, the software is configured to detect conflicts involving trajectories, only. Therefore, conflicts between volumes are omitted, only trajectory/trajectory and trajectory/volume situations are of relevance.

Based on polygon volumes, more complex constructs can be defined by combination. Staggering polygon volumes above each other allows to define volumes that change extension with altitude (e. g., for representation of cumulonimbus clouds that usually have a flat layer at the top). Defining different polygon volumes for short connected time periods represents a volume changing shape in time (e. g., for representation of ash-clouds).

3.4.3 N-Dimensional Moving Volumes

For the time referenced conflict detection, an extension of polygon volumes is provided by the ability of assigning a trajectory $T_{\mathfrak{V}}$ to a volume $\mathfrak{V}$. This trajectory defines an offset in all (N-1) dimensions as a function of time. Thus, the trajectory adds the capability of moving the polygon along an assigned path. The original time interval of $\mathfrak{V}$, if defined, is ignored. In accordance with eq. (3.33), a polygon volume $\mathfrak{V}(\tau)$ for time τ is defined as the Cartesian product

$$\mathfrak{V}(\tau) = \mathfrak{P} \times \left[\vec{I}_{\min}, \vec{I}_{\max}\right] \times [0,0] + T_{\mathfrak{V}}(\tau) \tag{3.35}$$

$\vec{I}_{\min}, \vec{I}_{\max} \in \mathbb{R}^{\textbf{N-3}}$

$T_{\mathfrak{V}} \subset \mathbb{R}^N$ is the assigned trajectory

The penetration detection works similar to that described for the non-moving polygon volume described in section 3.4.2 while respecting the movement induced by the assigned trajectory. A bounding box intersection trial between the tile and the bounding box of the polygon volume for the time interval $[a_{\min,N}, a_{\max,N}]$ allows a quick check if a situation represents a possible collision, at all. When positive, it needs to be verified if:

- The moving polygon contains the whole tile. One point of the tile (e. g., the center point $0.5 \cdot (\vec{a}_{\min} + \vec{a}_{\max})$) needs to be checked to be inside the polygon.
- The tile contains the whole polygon. One point of the polygon needs to be checked to be inside the tile.
- The tile intersects with the moving polygon. If both tests above provide negative results, polygon intersection needs to be tested. The intersection test also helps to calculate complete containment (section 3.3.5). If one point of the polygon is inside the tile (see above) and the tile does not intersect with the polygon volume, the tile is completely within the polygon volume.

A moving polygon volume $\mathfrak{V}(\tau)$ is in conflict with a tile A if

$$\exists \tau : \exists \vec{x} : (\vec{x} \in \mathfrak{V}(\tau)) \wedge (\vec{x} \in A) \tag{3.36}$$

$\tau \in \{a_{\min,N}, a_{\max,N}\} \wedge (\exists_1 \vec{y} \in T_{\mathfrak{V}} : y_N = \tau)$

However, checking the intersection between a volume and a tile for every time τ is not possible that way on a computer. Discretization of time is necessary, but also introduces several problems:

- Discretization reduces the accuracy of results. If a situation of interest is active for a very short time period, the resolution of discretization needs to be good enough to detect the issue.
- High resolution discretization results in many computational expensive calculations. Especially with large tiles (i. e., the tiles close to the tree root) and long trajectories performance would be low using straight forward discretization.

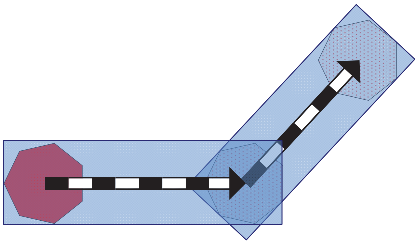


Figure 3.12 – Shape Generated by Moving Polygon

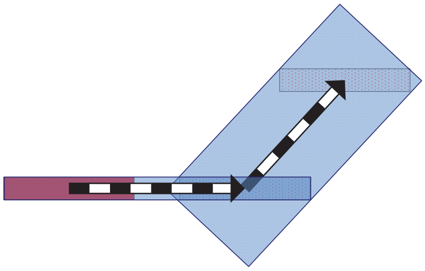


Figure 3.13 – Shape Generated by Less Symmetric Polygon

Therefore, another approach is chosen to calculate intersections. Figure 3.12 shows the blue shape generated by the red moving polygon in 2D along two trajectory segments in black/white. The blue corridor represents the polygon along the two trajectory segments. The only values of interest for the corridor are the points with biggest distance back and front (along track), and left and right (cross-track) from the central trajectory. These values define a rectangle in 2D and change from one segment to the next, depending on the trajectory's direction and polygon's shape.

The idea is translating x/y into along/cross-track components and thus overlaying along-track distance and time. Figure 3.13 shows a less symmetric polygon creating a different shape.

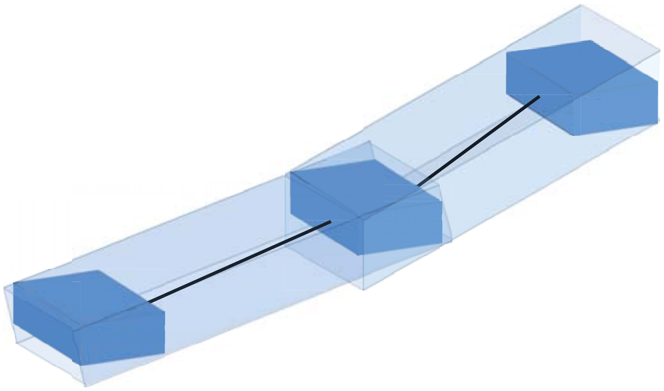


Figure 3.14 – 3D-Corridor for Moving Polygon

The corridor can also be calculated vertically and for all other higher dimensions, see fig. 3.14. The corridor is an object oriented (i. e., trajectory segment oriented) bounding volume (compare section 3.3.6) for the moving polygon. That way, a conservative representation of the moving polygon for a specified time interval is provided.

Figure 3.15 illustrates the calculation of higher dimension's corridor sizes using the example of altitude. Since the vertical size of a polygon is constant for the whole polygon by definition, it is enough to investigate the four corners of the red polygon volumes. Based on the gradient $z' = (\Delta z/\Delta\text{distance})$ of the trajectory segment, the standard altitude interval from start of trajectory segment needs to be extended by:

- $(\text{AlongTrackBefore} * z')$; and
- $(\text{AlongTrackBehind} * z')$.

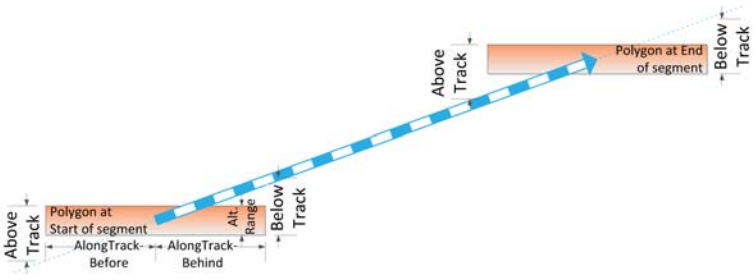


Figure 3.15 – Generation of Vertical Corridor

Based on the calculated corridor, intersections with the tile (if any) can be found fast and easily. Therefore, the trajectory defining the volume's movement is intersected with all tile boundary planes. If the intersection point is closer to a tile boundary than the corridor's size, the corridor intersects with the tile. Since the cross-track tile size depends on the trajectory segment direction, it is beneficial to center the trajectory in cross track direction in order to reach a symmetric (and thus no direction dependent) lateral corridor size. Figure 3.16 depicts an intersection of a moving polygon volume with a tile. Even though the centered trajectory does not penetrate the tile boundaries, the intersection point with the front plane of the tile marked by the red ball is close enough to indicate a possible conflict between tile and corridor.

Finally, the potential conflict needs to be verified to be a real conflict by searching an intersection between the volume and the tile. The potential conflict provides an appropriate time frame for that verification. Since the search area now has very limited size, time can be discretized to small values, e. g., to one time unit.

3.5 N-Dimensional Conflict Detection

As explained above, output of the broad phase algorithm described in section 3.3 represents a list of potential conflicts. Depending on the application area, potential conflicts might not fulfill all condi-

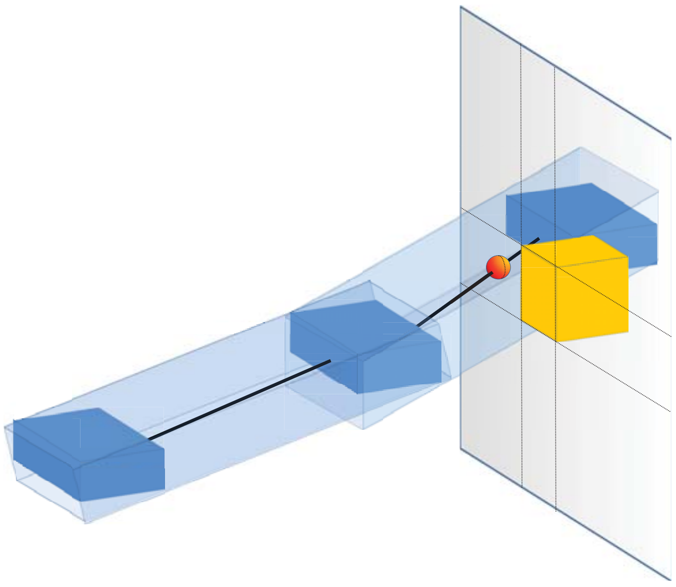


Figure 3.16 – Intersection between Moving Polygon and Tile Plane marked with Red Sphere

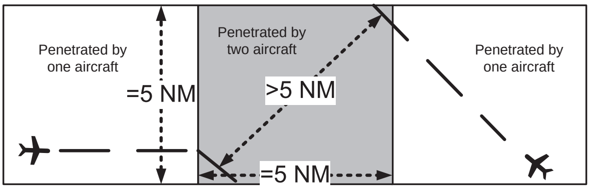

Penetrated by one aircraft
=5 NM
Penetrated by two aircraft
>5 NM
=5 NM
Penetrated by one aircraft

Figure 3.17 – Two Non-conflicting Objects in same Tile

tions of a real conflict. In aviation, separation minima are typically defined as a 2D circle in latitude/longitude plane. Figure 3.17 shows a situation where one tile is penetrated by two aircraft. However, even if latitude and longitude of both flights are closer than the allowed 5 NM, they are separated with the circular metric mentioned before.

The narrow phase conflict detection check is also essential if it was decided to reduce depth of the tree by stopping the broad phase process earlier (e. g., going down to a separation of 10 NM only, compare section 3.3.10) or memory limits were reached (section 3.3.11).

Validating a potential conflict as a real conflict might be a complex task. Therefore, this check is not done within the software core, but the detection module expects a pointer to an external function deciding if two given points are in conflict:

$$\textbf{bool}\ \text{isRealConflict}(\text{point}\ \vec{p}_1, \text{point}\ \vec{p}_2, \text{trend}\ \vec{t}_1, \text{trend}\ \vec{t}_2) \quad (3.37)$$

$\vec{p}_1, \vec{p}_2, \vec{t}_1, \vec{t}_2 \in \mathbb{R}^N$

In time reference mode, the trends $\vec{t}_1$ and $\vec{t}_2$ hold the differential coefficient for points $\vec{p}_1$ and $\vec{p}_2$ along the corresponding trajectories based on one time unit. Trends can be used if conflict situation depends on current state of objects. In aviation, aircraft might be defined to have a conflict depending on their vertical speed.

The conflict function defined in eq. (3.37) is called for trajectory objects only, because volumes are checked with regard to penetration, only.

Summarizing, once the tiling algorithms indicates potential conflicts, these need to be verified as real conflicts. The final conflict check depends on the involved object types and is described in the following sections.

3.5.1 Conflict between Trajectories

Section 3.2.1 and section 3.2.2 define a real conflict situation without and with time reference respectively. In the following, the process of conflict validation based on potential conflicts is described for trajectories.

3.5.1.1 Trajectory Conflict including Time Reference

Based on the identified potential conflict and the corresponding tile A, a τ fulfilling eq. (3.14) needs to be identified in the interval $\tau \in [A_{\text{min},N}, A_{\text{max},N}]$. Since A is a leaf tile, the interval is rather small and can be searched with a small discretization in time. As soon as the conflict function from eq. (3.37) turns true for one τ, a conflict is identified.

Once a conflict is identified, it is extended in time dimension in order to get start and end of conflict. A minimum gap time $\mathfrak{G}$ between two conflicts ensures that two parallel trajectories with small position jitter are continuously in conflict when moving close to mandatory separation $\vec{S}$.

The minimum gap time $\mathfrak{G}$ is also useful for the process of calculating minimum and maximum conflict time. Starting from the conflict-proved time τ, the conflict function from eq. (3.37) is checked at $\tau \pm k \cdot \mathfrak{G}$ with $k = 1, 2, 3, \ldots$. When the conflict check is negative, the step size is continuously reduced down to the foreseen result accuracy. Figure 3.18 shows how conflict start and end are iteratively calculated starting at the conflict-proved time τ. This procedure allows fast detection of the conflict interval. However, due to the discretization of time, this approach may leave gaps

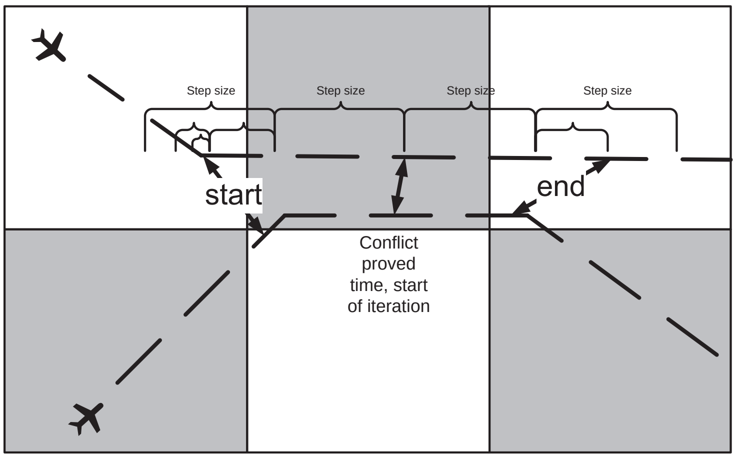


Figure 3.18 – Calculation of Start and End of Conflict

that are smaller than the minimum gap time $\mathfrak{G}$. Therefore, a final merging is necessary, see section 3.5.3.

3.5.1.2 Trajectory Conflict without Time Reference

Without time reference, a tile contains a conflict if two trajectories t_1, t_2 come closer than the given mandatory separation $\vec{S}$ (eq. (3.12)). Therefore, the minimum distance between two trajectories needs to be calculated. Before calculating the distance, trajectories are normalized with $\vec{S}$ in order to have a common metric for all dimensions.

Based on the normalized trajectories, all relevant segments from t_1 are checked against all relevant segments from t_2. Segments are relevant if they affect the tile. If the result is less or equal 1 in each dimension, the result is finally verified with the external conflict function eq. (3.37).

If conflicts are identified for multiple segments, they are merged. The trajectory point index for the first conflict partner is stored for later use, see section 3.5.4 for details.

3.5.2 Conflict between Trajectory and Volume

As mentioned before, a conflict between a volume and a trajectory exists when the trajectory penetrates the volume. As for trajectories, the conflict detection procedure differs with regard to time reference available or not.

3.5.2.1 Trajectory/Volume Conflict including Time Reference

The conflict with time reference is comparable with the procedure described in section 3.5.1.1. Instead of testing the conflict function it is checked if the volume for time τ contains the trajectory point.

3.5.2.2 Trajectory/Volume Conflict without Time Reference

In contrast to the inter-trajectory conflict detection described in section 3.5.1.2, conflict detection for volumes is not performed within tile boundaries only, but for the whole volume. Therefore, each segment of the trajectory is checked for intersection with the volume boundaries. In addition, start and/or end of trajectory within the volume is stored as a conflict.

Based on that conflict list, conflicts are sorted according to the trajectory point index and grouped to pairs. These pairs then hold entry and exit point of the polygon volume.

3.5.3 Conflict Merging

As described in section 3.5.1.1, the discretization of time may produce gaps that are smaller than the minimum gap time $\mathfrak{G}$. Figure 3.19 illustrates this behavior. Since the third time offset points to a no-conflict situation, the conflict is limited to the left side of the conflict gap. Another position of the conflict gap could have produced the correct result containing the whole conflict. However, taking the gap problem into account directly when calculating the conflict interval would be computational expensive.

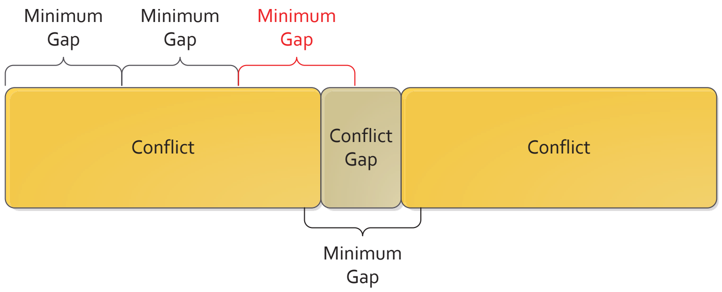


Figure 3.19 – Discretization of Time prevents Gap Jump

In order to ensure that no conflict gap is smaller than minimum gap size $\mathfrak{G}$ a final merge is performed when a new conflict is identified. If another conflict already exists with identical conflict partners combined with a smaller distance to the new conflict than $\mathfrak{G}$, both conflicts are merged.

Having knowledge about the extent of conflicts helps with early occlusion of already detected conflicts. The conflict in fig. 3.18 is detected in all three upper and the center lower tile. Since the first tile detecting the conflict calculates the whole conflict area, all subsequent conflicts are based on a time τ, already characterized by the first conflict, and these may be omitted.

3.5.4 Output Format

The result of conflict detection is a list of conflicts. Each conflict is described by:

- References to both conflict partners. For conflicts between heterogeneous object types, trajectories are always first in order.

- Start and end time of conflict in time reference mode. If no time reference is specified, start and end contain the linearly interpolated point index for the first conflict partner.

- Trends of both objects. In time reference mode, the trends hold the differential coefficient for the conflict duration along the corresponding trajectories based on one time unit. Trends are zero for non-trajectory objects and inactive time reference mode.

- The Closest Point of Approach (CPA) holds the point where the distance between both objects is smallest. In order to compare distances for heterogeneously scaled coordinate systems each dimension is normalized using the corresponding entry of mandatory separation $\vec{S}$. If objects intersect, the CPA is the first intersection point. Otherwise, the CPA is the midpoint between closest corresponding points of objects.

  The CPA is useful when displaying conflict situations.

- The phase of conflict classifies conflicts for each object and comes from the aviation domain. Conflicts appearing no later than two minutes after start of trajectory are considered to be departure conflicts. Conflicts ending no earlier than two minutes before end of trajectory are considered to be arrival conflicts. Conflicts with positive/negative climb-rate are considered to be climb/descent conflicts.

  Furthermore, object's conflict types are merged to a common conflict type. Table 3.5 shows the resulting global conflict type. The choice of conflict resolution algorithm strongly depends on the conflict type.

  Table 3.5 – Phase Merging of Conflict Types

	Dep	Arr	Clb/Crs/Dsc
Dep	Dep	ArrDep	Other
Arr	ArrDep	Arr	Other
Clb/Crs/Dsc	Other	Other	Other

- The 2D-length of conflict can be used for prioritization/classification issues.

- The minimum 2D-distance of conflict helps with adjustment of lateral conflict solution. Other algorithms are used if two trajectories intersect or if they just come close.
- Reference to one node of conflict. This is just a cross reference for internal use.

Conflicts between 3 or more partners are reported separately. That means, if three trajectories t_1, t_2, t_3 have a conflict at a common point/interval, $\binom{3}{2} = 3$ conflicts will be reported (t_1 with t_2, t_1 with t_3, and t_2 with t_3). The common conflict of four trajectories is reported as $\binom{4}{2} = 6$ conflicts.

3.6 Objects in Focus

In some cases, it is not relevant to get all conflicts of a scenario, but only the conflicts involving a pre-defined subset of objects. Of course, this can be easily achieved by filtering the conflicts of interest from the final conflict list but it can also be done more efficiently.

The software implementation allows to define a list of objects in focus. Since the knowledge about focus objects is already used when building the tree, defining objects in focus should be performed before adding object. Tiles are subdivided the same way as before, but only if it contains at least one object in focus. Depending on the number of focused objects compared to the scenario size, this may reduce both memory usage and calculation times drastically.

Figure 3.20 and fig. 3.21 show the detected conflicts for an unfocused and focused scenario.

3.7 Software Implementation

The software developed is written for standard PC hardware using C++, graphical output is done using OpenGL. While the primary operating system in use is Linux, it works as well on a Microsoft Windows platform. The number of dimensions of the problem and

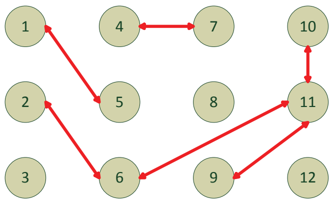


Figure 3.20 – 12 Objects and their Conflicts

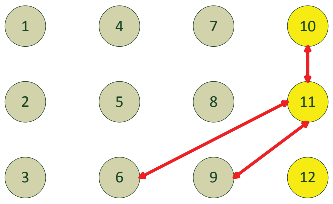


Figure 3.21 – Focus on Objects 10-12

the information if it contains a time reference are stored using template variables. This increases compilation times, but holds software complexity low. The core software module has ∼8500 lines of code and can be compiled for 32 bit and 64 bit hardware. The latter one is preferred because it allows addressing more than 4 GiB of memory and therefore allows handling larger scenarios. Since many pointers and references are used in the code, the 64 bit version needs more memory for the same scenario though.

The software module is called N-Dimensional Map-Implementation (*NDMap*), providing above described N-dimensional tiling algorithm and additional functionality for conflict resolution. Listing 3.1 shows a small example code demonstrating how to use the *NDMap*.

The output of this example is shown in listing 3.2. The result reports two conflicts between the trajectory and the concave polygon volume. Adding an object took around 1.7 ms, 116.5 KiB of memory are used. The tree has 1 element of level 0 (this is the root tile), 5 tiles of level 1, 4 of level 2, . . ., and 139 tiles of level 13. In total, the tree contains 662 nodes. 684 objects have been inserted in nodes as penetrating object and 254 as fly-by.

All results in this work are calculated on a Dell Precision T7500 computer powered by an Intel Xeon X5687 with 3.6 GHz running a 64 bit Linux operating system (Suse 12.2). Since the algorithm performs well with a really huge scenario, the computer is equipped with well above standard 96 GiB main memory. Although the Cen-

Listing 3.1 – NDMap Sample Program

```
#include "NDMap.hpp"
NDMap<4, true> myMap; // create a map with 4 dimensions
     including time reference
myMap.setInterval('x', -180, +180); // longitude in
    [-180,180] degree
myMap.setInterval('y', -90, +90); // latitude in
    [-90,90] degree
myMap.setInterval('z', 0., 100000.); // altitude in
    [0,100000] feet
myMap.setInterval('t', 0., 86400.); // time in
    [0,86400] seconds (=one day)
myMap.setSeparationMinima(NDVec<4>(5 / 60., 5 / 60.,
    1000.,90.));// 5 minutes x and y, 1000ft, and 90
    seconds
myMap.setDebug(false);
myMap.setTiming(); // Turn on timing of add/del methods
myMap.setIdentifier ("Test 4D");

NDPoly<4,true> temp("Poly"); // create a polygon
temp.addPoint(NDVec<2>(0,35)); // Two dimensions only
temp.addPoint(NDVec<2>(.4,37)); // Object exists in all
     altitudes...
temp.addPoint(NDVec<2>(0,40));  // ...and for every
    time
temp.addPoint(NDVec<2>(1.5,38));
temp.addPoint(NDVec<2>(.6,35));
myMap.addObject(&temp,true); // store polygon in map

NDTraj<4,true> traj("Traj"); // create a trajectory
traj.addPoint(NDVec<4>(.1,30,10000,0)); // South to
    North flight from y=30
traj.addPoint(NDVec<4>(.1,50,10000,50));// to y=50
    degree at x=.1
myMap.addObject(&traj,true); // store trajectory in map

cout << myMap; // print relevant information
```

Listing 3.2 – NDMap Output of Sample Program

```
Test 4D containing 1 trajectory 1 polygon 4D incl. time
    ,
Pen: 684 FlyBy: 254
Sep: (0.08333333333, 0.08333333333, 1000, 90)
 116.5 KByte, Add: 1.67050001 ms
 0:1, 1:5, 2:4, 3:8, 4:8, 5:9, 6:20, 7:26, 8:22, 9:38,
     10:124, 11:122, 12:136, 13:139, [=662];
 2 conflicts:
1 t[12.5, 13.75] between Traj/Dep and Poly/Poly at
    (0.1, 35, 10000, 12.5)(Polygon)
2 t[23.125, 24.66666667] between Traj/Dep and Poly/Poly
     at (0.1, 39.25, 10000, 23.125)(Polygon)
No label filtered
```

tral Processing Unit (CPU) had 4 cores (8 threads), the algorithm did not use parallel processing in its current version.

Chapter 4

Conflict Detection in 4D-Airspace

This chapter describes how the conflict detection algorithm explained in chapter 3 can be used for conflict probe of 4D-trajectories in aviation. First of all, the algorithm needs to be configured correctly. Therefore, the number of dimensions N is set to 4 and time reference mode is activated. Table 4.1 shows the setup for each dimension covering one day of world-wide air traffic. S defines the mandatory separation for each dimension. D is the resulting depth for each dimension.

Table 4.1 – Setup of Conflict Map for 4D-Airspace

Dim.	Unit	Min	Max	S	D*
x/Longitude	Deg	-180	180	$^5/_{60}$	12.08
y/Latitude	Deg	-90	90	$^5/_{60}$	11.08
z/Altitude	ft	-1000	100 000	1000	6.66
t/Time	s	0	86 400	90	9.91

* with $D = \log_2\left(\frac{\max - \min}{S}\right)$

4.1 Topological Isomorphism of Earth

4.1.1 The Earth-Mode

The last but one column of table 4.1 holds the mandatory separation $\vec{S}$. The value 5/60 is supposed to be the equivalent of 5 NM for longitudes and latitudes. Due to Earth's sphere-like shape, this is not true for longitudes. One nautical mile is about one minute of arc measured along latitude, or about one minute of arc of longitude at the equator (fig. 4.1). All longitudinal minutes aside the equator measure less than 1 NM. Therefore, 5/60° is about 5 NM on the equator, but less aside.

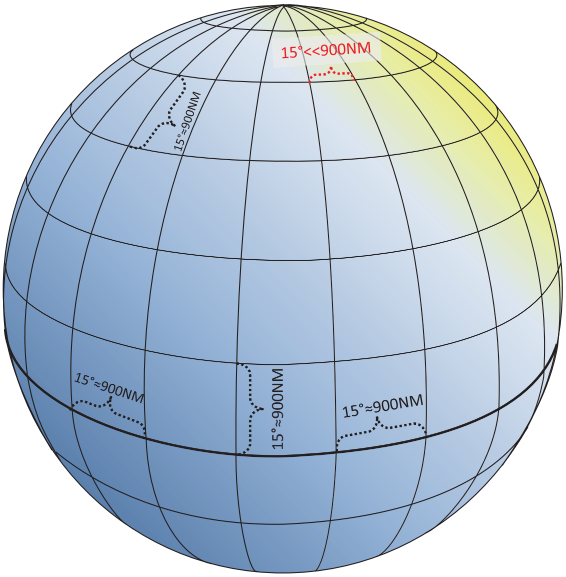


Figure 4.1 – Latitudes and Longitudes on Earth

If a tile is smaller than the mandatory separation $\vec{S}$ conflicts may be overlooked. The real distance between longitudinal arcs is illustrated best by the sinusoidal projection (fig. 2.25 on page 78).

Obviously, the distance between 5 longitudinal minutes at the poles is zero and therefore lower than any longitudinal separation S_λ.

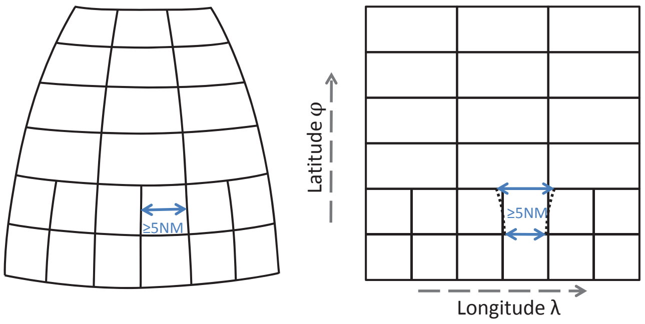


Figure 4.2 – Spherical and Corresponding Cartesian Model

When representing Earth in a Cartesian coordinate system, it is preferable to have an overlapping model (i. e., an aircraft flying close to a border between two tiles penetrates both tiles) instead of having an uncovered gap between two tiles.

In order to respect the non-parallelism of longitudes the tiling algorithm provides an Earth-mode. In Earth-mode the conflict detection algorithm uses longitudes λ directly as x coordinates, see fig. 4.2. However, the non-parallelism of longitudes is taken into account by adapting the longitudinal separation S_λ to the appropriate latitude φ:

$$S'_\lambda(\varphi) = \frac{S_\lambda}{\cos\varphi} \tag{4.1}$$

When precision needs to be very high and a spherical Earth model is not accurate enough, the elliptical Mercator projection leads to (Osborne, 2008):

$$S'_\lambda(\varphi) = \frac{S_\lambda \cdot \left(1 - e^2 \sin^2\varphi\right)}{\cos\varphi} \tag{4.2}$$

$e^2 = 2f - f^2 = 0.00667053982$ is the eccentricity of Earth
with $f = 0.0033408505$ being the flattening of Earth

Both eq. (4.1) and eq. (4.2) are not defined at the poles for $\varphi = \pm 90°$. Coming close to the Poles, objects occupy all longitudes.

Since $e^2 \sin^2 \varphi$ is close to zero, elliptical effects are very small and often negligible in favor of faster calculations. For the spherical formula, the longitudinal minutes matching 5 NM double at a latitude of 60°, since $S'_\lambda(60°) = S_\lambda/0.5$. For more complex objects, it is not necessary to recalculate $\cos\varphi$ each time. For not too big objects, a common φ can be used, based on a conservative estimation choosing the latitude of the object point being farthest from the equator. This can be calculated easily by analysis of the object's bounding box.

Even though the decision is not critical in terms of overall correctness, the subdivision process also needs to be adapted in Earth mode. Since tiles are rectangular, but the longitudinal range changes with latitude, we need to specify which longitudinal range is taken for the subdivision decision. Starting with the whole Earth as root tile, longitudinal range at both minimum (Antarctic) and maximum (Arctic) latitude is zero and thus not a good choice. The maximum (full) longitudinal range is reached in the center of the latitude interval. Concerning the longitudinal subdivision, a tile $A(a_{\min}, a_{\max})$ is generated as long as

$$(a_{\max,\lambda} - a_{\min,\lambda}) \geq S'_\lambda \left(\frac{a_{\max,\varphi} + a_{\min,\varphi}}{2} \right) \tag{4.3}$$

Due to the tree structure and above formulated tiling decision, the tiling is not perfect at the Poles. Regarding the distance criteria one tile is enough very close to the Poles, but the tree structure subdivides longitudes in every depth step as long as it is beneficial for the central latitude of a tile. Thus, several Pole tiles are usually generated with an overlapping area of influence.

4.1.2 Great Circle Connections

As described in section 2.6.1, shortest surface connections on a spherical Earth model are great circles. Disregarding wind influence, it is most efficient for aircraft to fly along great circles. However, the tiling algorithm is based on a Cartesian coordinate system where shortest connections between points are straight lines. Depending on distance and direction, great circle distance and route differ significantly from the Cartesian direct connection (fig. 4.3).

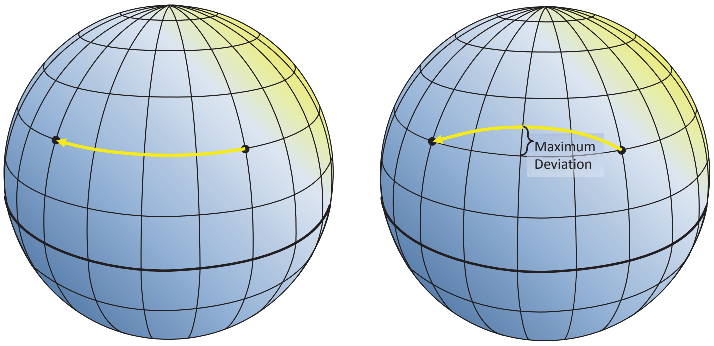


Figure 4.3 – Cartesian vs. Great Circle Connection

Realizing great circles in the tiling algorithm is no problem technically. However, it would increase computation times since great circle intersection is far more complex than axes-aligned Cartesian intersection calculation.

The difference between Cartesian and great circle connection gets smaller with more sampling points. Thus, conflict detection works precise enough with a sufficient number of sampling points. If the trajectory provider is unable to deliver denser points, the implemented object class provides functionality interpolating new points based on great circle and WGS84 calculation before performing conflict probe. Figure 4.4 shows how the distance between great circle and Cartesian connection depends on length of flight and its latitude, departure and arrival positions both having same latitudes. Since the algorithm's sensitivity concerning computation time against point density is low, trajectories can be interpolated with a high number of points (compare section 4.4.3 on page 147).

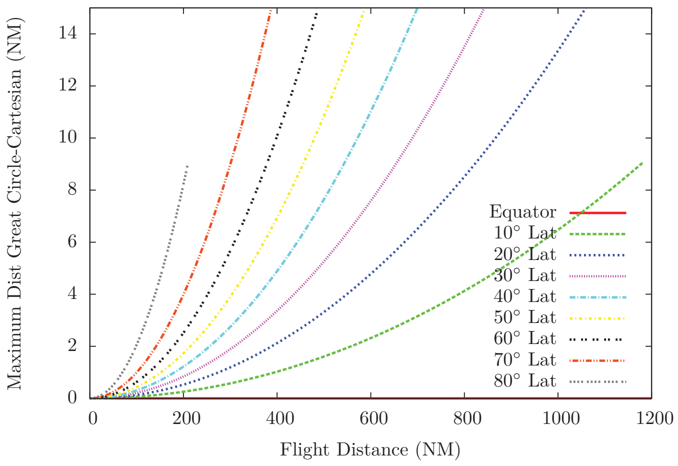


Figure 4.4 – Influence of Distance and Latitude on Great Circle-Cartesian Distance

The object class implementation also provides functionality for calculation of essential points from a trajectory with too many points. When a trajectory was interpolated for conflict detection it can later on be reduced for, e. g., drawing purpose. The reduction is performed using an N-dimensional version of the Douglas-Peucker-algorithm proposed by Ramer (1972); Douglas and Peucker (1973). Based on a maximum allowed deviation, the algorithm approximates a trajectory with a reduced number of points. The algorithm begins with connecting start and end of trajectory as shown in fig. 4.5. Following the divide and conquer principle, the algorithm recursively divides the line segment where the trajectory is furthest from the polygon in order to insert a new sampling point. The algorithm is implemented for N-dimensional fitting. The distance function is normalized by the given maximum allowed deviation vector. The overall distance metric is defined as the maximum of the components in order to align sensitive axes first.

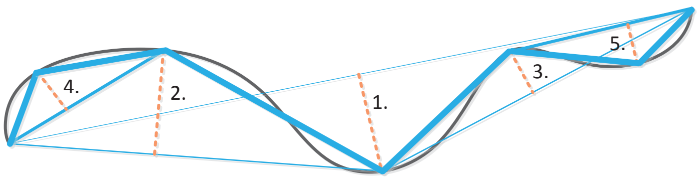


Figure 4.5 – Douglas-Peucker for Polygonal Approximation

4.1.3 Singularity and Discontinuity of Longitudes

Earth's longitude is singular at the Poles and has a discontinuity at the ±180° meridian. Positions at Poles, i. e., latitude $\varphi = \pm 90°$, have an undefined longitude. Furthermore, the latitude/longitude grid is highly non-linear close to the Poles. However, the Pole singularity is solved using the Cartesian ENU coordinate system shown in fig. 4.2. When checking conflicts with tiles the singularity of eqs. (4.1) to (4.2) needs to be taken into account. Close to Pole objects can be defined to be within all longitudinal Pole tiles, if there exist more than one. In worst case this results in a potentially conflicting object having no real conflict.

The discontinuity at the ±180° meridian (also known as international date line) is more difficult to respect in an adequate way. The most efficient way of handling the discontinuity is avoiding it. Thus, if the air traffic scenario under investigation is limited to Europe, flights typically do not cross the date line. Working on a scenario with traffic between the United States and Asia is possible by using a longitudinal interval of $[0, 360]$ instead of $[-180, 180]$ as long as the prime meridian through Greenwich is not crossed. That way, each scenario not touching one selected meridian can be aligned by a suitable longitudinal interval avoiding the problem of discontinuity. However, if no such untouched meridian can be identified, another solution is necessary.

On a spherical Earth model, longitudinal meridians -180° and 180° (generally $(k \cdot 360° - 180°), k \in \mathbb{Z}$) are identical. The Cartesian representation (fig. 4.2 on page 127) puts the endpoints of the interval to the left and right border respectively, producing the maximum possible distance for identical longitudes. While a great circle connection on the spherical model is defined to be the shortest connection between both points, the Cartesian representation only provides one direct connection within the interval. Aiming for the shortest connection shortcuts via the longitudinal discontinuity need to be taken into account. The green and red connections in fig. 4.6 show the short and long great circle routes from $P1$ to $P2$ in spherical and Cartesian representation.

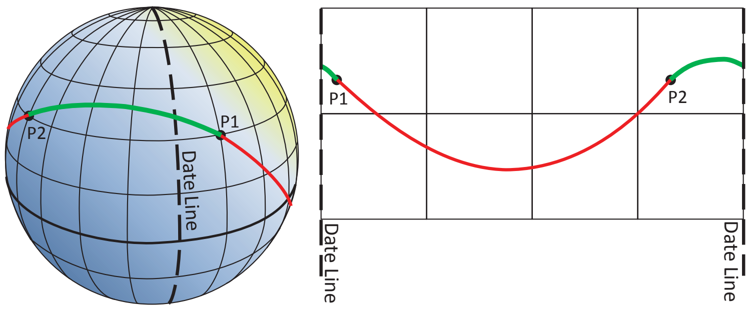


Figure 4.6 – Shortest Connection on Spherical and Cartesian Representation

The major issue with the connections crossing the date line is not respecting it in a proper way, but finding an efficient solution without downgrading the overall performance. The implementation respects the wrap-around for the longitudinal dimension in Earth-mode. Additional sampling points are automatically inserted for trajectories having the shortest connection via the longitudinal discontinuity λ_X. In total, four points are inserted around λ_X:

- The first point p_1 is inserted just before passing the discontinuity. The point is located at $a_{\max,\lambda} - \varepsilon$ for increasing and $a_{\min,\lambda} + \varepsilon$ for decreasing longitudes.
- The second point p_2 has same coordinates as p_1, but a marked altitude that is not within the scenario's altitude interval.
- The third point p_3 has same coordinates as p_4 with the same (marked) altitude as p_2.
- The fourth point p_4 is inserted just after passing the discontinuity. The point is located at $a_{\min,\lambda} + \varepsilon$ for increasing and $a_{\max,\lambda} - \varepsilon$ for decreasing longitudes.

The marked points p_2 and p_3 are inserted in order to guarantee that the large Cartesian connection $\overline{p_2 p_3}$ is not considered for conflict detection without retarding the code (dotted horizontal line in fig. 4.7). Invalid conflicts on the vertical segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ are very unlikely and can be omitted directly after conflict detection. An example is given in section 6.5. This procedure solves all issues concerning the detection of trajectories penetrating a tile. However, a conflict between two aircraft flying parallel on opposite sides of λ_X still stays undetected.

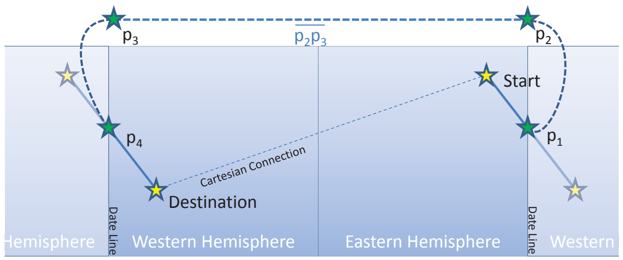


Figure 4.7 – Passing the Date Line in Cartesian Coordinates

Therefore, the fly-by detection also needs to be adapted. The dilatation procedure defined in section 3.3.3 does not respect a possible longitude discontinuity. Dilatation of a tile at the discontinuity boundary does not produce one single but two unconnected

N-dimensional spaces. As explained in section 3.3.4 on page 97 fly-by objects only need to be taken into account when passing on the positive (i. e., right) side. Since the likelihood of an aircraft being less than 5 NM right of the longitudinal discontinuity is low, objects are checked on insertion if they are relevant as discontinuity fly-by. Thus, the extra fly-by volume is checked only for few relevant objects.

4.1.4 Alternative Earth Mapping

Pole and discontinuity issues raise the question if there are other more suitable mappings from 4D-airspace into a 4-dimensional Cartesian coordinate system.

4.1.4.1 Spherical Coordinate System

A mapping avoiding pole and discontinuity issues is the spherical coordinate system depicted in fig. 4.8. Every point on (and in) Earth has unique 3D-coordinates. Airspace surrounds Earth and can be addressed the same way.

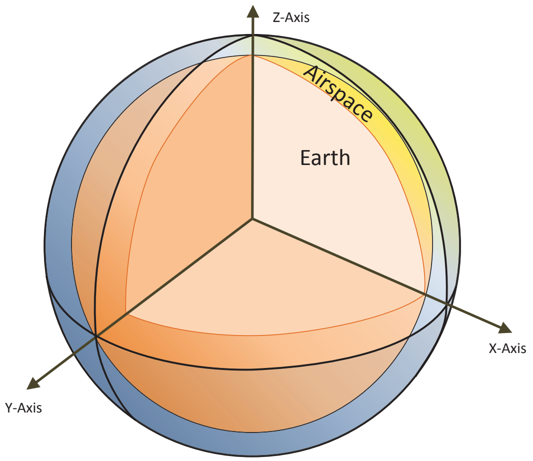


Figure 4.8 – Alternate Earth Mapping

Containing no air traffic at all, the Earth volume is free of conflicts and thus will only be subdivided at the Earth surface by the conflict detection algorithm. Due to a large Earth radius of ∼6 370 km and a small airspace height of typically less than 20 km, the conflict detection algorithm needs to focus on a small but widely spread part of the initial volume. Even though this results in a deep tree (degrading also the performance), the tiling algorithm is predestined for these conditions. The reason why the spherical coordinate system was not used in this work is the alignment of conflict metrics.

Using spherical coordinates, conflict dimensions are not aligned to the coordinate system anymore. Thus, altitude separation needs to be calculated by building the difference between Pythagorean distances of both trajectories to Earth center. Distances concerning latitude and longitude are even more complex to distinguish. While small tiles at the North Pole have z as altitude and x, y for longitude and latitude, z is the latitude at the Equator. The unaligned axes also complicate fly-by detection. In addition, the user interface would need to map between usual latitude/longitude/altitude representation and internal x, y, z coordinates introducing workload and accuracy degradation.

Another problem with spherical mapping is the support of sparse trajectories. Long distance trajectories with few sampling points rely on a proper interpolation. While linear interpolation is acceptable for the mapping described in section 4.1.1, it is problematic with spherical coordinates. The direct connection between two spherical points of same distance to center gets closer to the center. If the center angle between the two points is big enough, the direct connection will penetrate the Earth volume. In extreme, the connection between two opposite points goes through the Earth center. Thus, linear interpolation varies altitude between two points having same altitude. Since altitude is more sensitive than lateral separation (1000 ft versus 5 NM), very dense trajectory points are necessary to stay accurate enough, or the interpolation method needs to be changed to a more complex one.

4.1.4.2 Geodesic Dome

Another option to represent the Earth surface without poles and discontinuity is using triangles instead of intervals of latitude and longitude. Fuller (1943) proposed a world map on the surface of an icosahedron, a geometrical object approximating a sphere with 20 equilateral triangular faces. Today, the regular icosahedron is often used as the base for geodesic domes. Each triangle can be split into 4, 9, 16, 25, ..., k^2 ($k \in \mathbb{N}$) equilateral triangles to get finer granularity (Woo et al., 1999). Projecting the new vertices from the triangle face onto the sphere generates better approximations. Figure 4.9 shows an icosahedron with 4 projected sub-triangles.

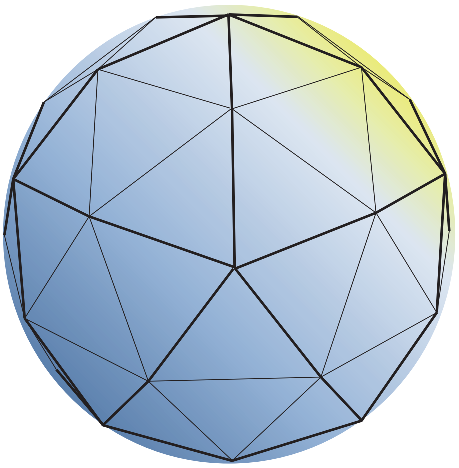


Figure 4.9 – Icosahedron with 4 Triangle Subdivision

In practice, the algorithm described in this work would need severe adaptations to handle triangular areas instead of intervals of latitude and longitude. While latitude and longitude are almost independent in the Cartesian tiling algorithm, triangles connect the two dimensions. The grid is not as regular as the latitude/longitude grid, and therefore more complex to describe. Furthermore, the number of twenty faces does not fit well into the tree structure because it is not a power of two.

Integration of the triangular layout in the generic *NDMap* breaks the general approach, and a separate approach would be preferable. Furthermore, since triangle borders are not axis-aligned, degradation in computation speed is anticipated. All in all, the higher structural complexity is not a good trade-off for the absence of poles and discontinuities.

4.2 Traffic Samples and Conditions

In the remaining work, conflict detection and resolution is basically performed on two scenarios described in this section, one scenario covering one day of German traffic, and one scenario holding European traffic.

4.2.1 German Traffic Sample

Germany has one of World's densest net of airfields and airports (∼550 (Central Intelligence Agency, 2007)). The German air navigation service provider DFS Deutsche Flugsicherung GmbH (DFS) controlled 3.06 million flights in 2011 (DFS Deutsche Flugsicherung, 2011). The German traffic sample is based on one of the busiest days 2008 holding 10 256 flights. Using flight plans and radar tracks, trajectories were generated with the AFMS to get a traffic sample with realistic assumptions. However, since aircraft comply with the performance models and weight assumptions from the AFMS, the resulting sample is by no means identical to the real traffic day, and thus contains ∼3 k conflicts based on a conflict definition of 5 NM/1000 ft in and above 5000 ft and 3 NM/1000 ft below.

Trajectories are not complete from departure to destination but cut close to the country borders in order to cover only the German part of flight (fig. 4.11). Figure 4.10 shows the distribution of flights along daytime. Peak traffic level is ∼500 concurrent flights. The sample contains flights departing from, arriving in, and flying through Germany. Some callsigns are used twice on the same day; the second flight replaces the first one in conflict detection on occurrence. Further information on the German traffic sample is

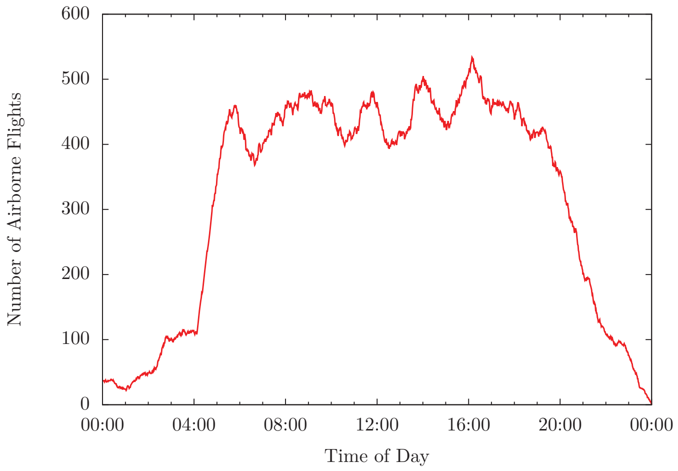


Figure 4.10 – Airborne Aircraft in German Traffic Sample

gathered in table 4.2. The untypical high maximum flight durations and distances are provoked by some experimental flights also being part of the traffic sample. These particular aircraft were obviously not accomplishing a transport task but performed special missions producing abnormal data.

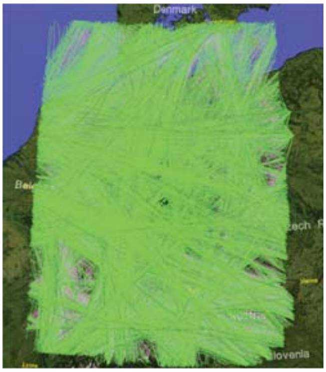


Figure 4.11 – German Air Traffic Sample

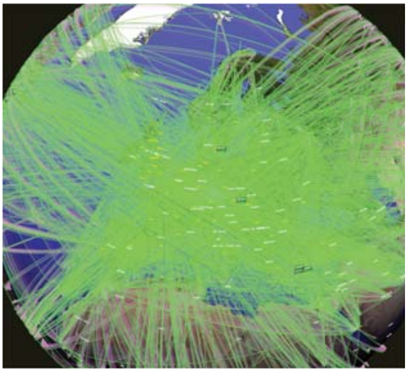


Figure 4.12 – Optimized European Air Traffic Sample

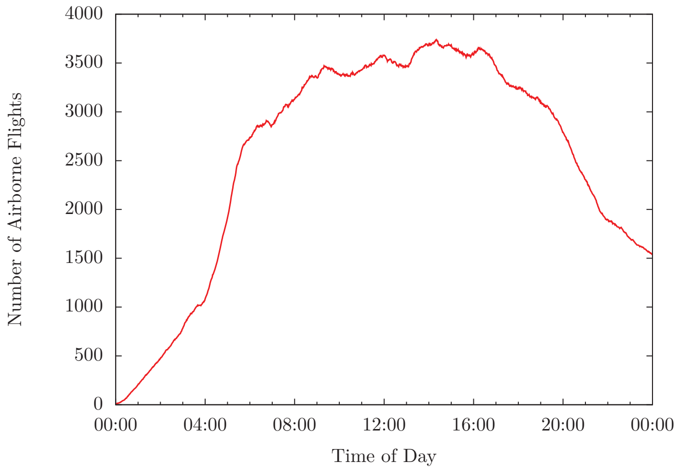


Figure 4.13 – Airborne Aircraft in European Traffic Sample

4.2.2 European Traffic Sample

The European traffic sample holds 33 069 airborne movements. In contrast to the German sample, flights are not cut at European borders, but are usually complete from departure to destination. The data is taken from the Demand Data Repository (DDR) provided by Eurocontrol (2010). The sample contains flights departing from, arriving in, and flying through Europe. As in the German sample, some callsigns were used multiple times a day. However, since the callsign is used as identifier in the *NDMap* the callsigns were unified by appending numbers in order to investigate the full traffic. As fig. 4.13 shows, the sample starts with departures at midnight, and holds all aircraft flying into the next day. The peak level of ∼3700 aircraft being airborne at the same time is significantly higher compared to Germany because:

- The scenario contains more than 3 times more movements; and

- Flights are complete from take-off to touchdown. An aircraft departing from the United States flying into Europe later on is counted for the whole flight duration.

The European traffic sample was modified to create an optimized scenario. In order to fly as efficient as possible, flights were put to most direct routes and optimum altitude and speed profiles. Since the aircraft do not respect today's airway structure the sample represents a future scenario.

The only data taken from the original sample is:

Callsign as an identifier for each flight.

Aircraft type describing the performance of specific aircraft

Departure and arrival runways defining the route to be flown

Departure time as the time reference for the flight

Cruise flight level defining the maximum altitude of flight.

All flights are optimized by applying the direct great circle connection from departure to arrival airport. Since the original runways are used, up to two waypoints are inserted at each airport in case angles get too pointed (Kuenz and Schwoch, 2012). Figure 4.14 shows an example where four additional points (D_1 and D_2 for the departure and A_1 and A_2 for the arrival) are necessary to get a feasible route layout. The new constraints for the trajectory predictor (compare section 2.3) were gathered by Gunnar Schwoch (DLR) in the context of the 4 Dimensional-Contracts - Guidance and Control (4DCo-GC) project under the European Union's Seventh Framework Programme for Research (FP7).

Based on aforementioned inputs, new optimized trajectories are generated using the AFMS described in section 2.3.1. Besides having much more efficient trajectories, the main difference to the real life traffic without conflicts is a significantly increased number of conflicts. The use of flexible route structures changes flight durations significantly by flying shorter great circle routes and aircraft-optimized speeds. This results in uncoordinated en-route and arrival segments and an overall conflict count of nearly 29 000.

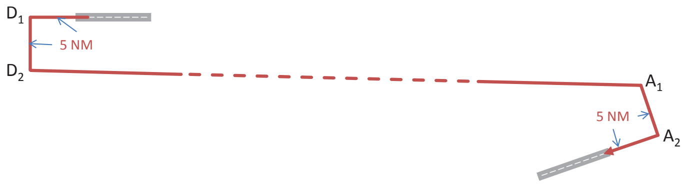


Figure 4.14 – Four Points in Worst Case for a Flyable Route Layout

Further information on the European traffic sample is collected in table 4.2.

4.3 Results from Conflict Detection

Detection of conflicts becomes complex with a large number of objects. Figure 4.15 shows conflict detection times for the German sample using the basic algorithm described in section 2.4.2. Comparing each trajectory against each other results in conflict detection times of more than 3.5 days for ∼10 k trajectories[1]. Due to the quadratic complexity $\mathcal{O}(N^2)$ using even bigger scenarios is unwise with this algorithm. Figure 4.16 depicts the results of the optimized version of the algorithm described in section 2.4.2. The overall detection time is reduced to less than 5.5 hours by checking conflicts only for aircraft being airborne simultaneously.

Even though the worst case complexity is still quadratic (e. g., if all aircraft are airborne at the same time), the conflict detection time has a nearly linear relation to the number of flights at least for the given scenario complexity. However, the overall poor performance of ∼1.8 s detection time per trajectory restricts the variety of possible application areas.

[1]Be aware of the y-axis scaling being non-consistent in this section due to very different computation times.

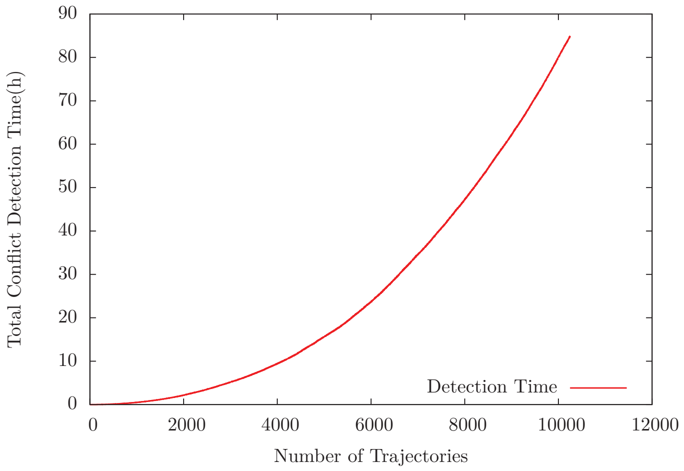


Figure 4.15 – Detection Times for German Sample Baseline

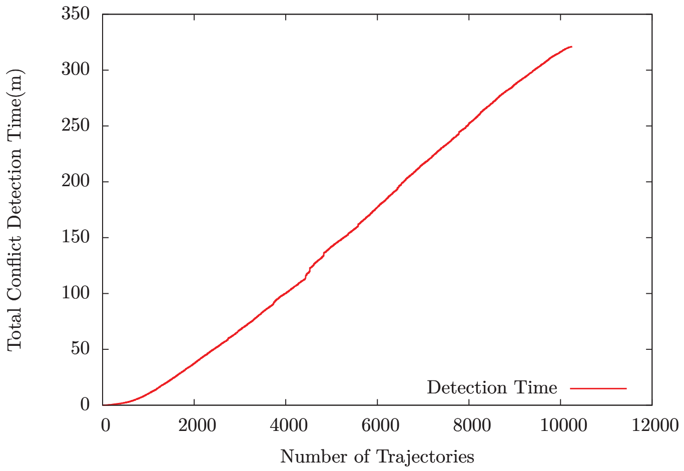


Figure 4.16 – Detection Times for German Sample Advanced Baseline

Table 4.2 – Properties of Traffic Samples

		German	European
Aircraft	[#]	10 256	33 069
Uniques	[#]	9783	33 069
Extension		Cut-to-border	Complete
Conflicts	[#]	3051	28 986
Heavy	[%]	12.5 %	9.6 %
Medium	[%]	83.8 %	83.8 %
Light	[%]	3.7 %	6.6 %
Duration	$(\mu \pm \sigma)$	0:46:04 ± 0:19:11	2:01:45 ± 2:10:11
[H:MM:SS]	Interval	[0:35, 7:43:38]	[7:45, 17:59:57]
Distance	$(\mu \pm \sigma)$	287.6 ± 118.4	820.9 ± 1021.8
[NM]	Interval	[1.9, 2615.3]	[19.5, 8310]
Points	$(\mu \pm \sigma)$	47.1 ± 18.9	89.1 ± 61.0
[#]	Interval	[2, 454]	[6, 1019]
Latitude	$(\mu \pm \sigma)$	50.3 ± 2.0	45.6 ± 10.3
[deg]	Interval	[46.5, 55.2]	[-35.0, 88.1]
Longitude	$(\mu \pm \sigma)$	10.1 ± 2.6	8.0 ± 28.2
[deg]	Interval	[5.5, 15.9]	[-123.2, 140.4]
Altitude	$(\mu \pm \sigma)$	24 914 ± 12 225	19 715 ± 14 042
[ft]	Interval	[0, 45 000]	[-72, 47 000]
Time	$(\mu \pm \sigma)$	12:37:24 ± 5:10:57	13:49:40 ± 3:54:02
[H:MM:SS]	Interval	[-2:21:01, 23:59:57]	[-51:57, 39:05:36]

The tiling algorithm described in this document reduces detection times significantly. Figure 4.17 depicts the performance of the 4-dimensional tiling algorithm. The detection of all conflicts is performed in ∼18.5 s for the German traffic sample. Thus, the average detection time per trajectory is as low as 1.8 ms. Figure 4.17 also shows the allocated memory of the hexadecimal tree. Using about 58 KiB per trajectory the algorithm can handle much more than the German area on standard off-the-shelf personal computer hardware. Both computation time and memory usage are mostly linear for the given scenario complexity. The computational performance can be tuned to 1.61 ms per trajectory by adapting the minimum tile duration (fig. 4.24 on page 153). However, this increases the average memory usage to 68 KiB per trajectory.

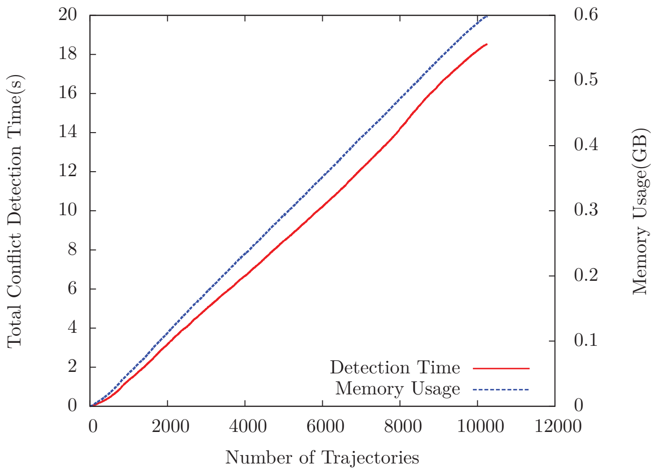


Figure 4.17 – Detection Times for German Sample with Tiling Algorithm

Figure 4.18 depicts the results from the tiling algorithm for the European sample. The prediction time is nearly linear with an average of 5.3 ms per trajectory. The steeper part in the center and the more shallow start and end are likely caused by the time based sorting of the scenario. Traffic density is lower around midnight and higher during daytime, compare fig. 4.13. The computational performance can be tuned to 4.97 ms per trajectory by optimizing the minimum tile duration (fig. 4.25 on page 153).

Trying to find a reason for the rather big difference in calculation times to the German sample (5.3 ms vs. 1.8 ms) is a complex job. Performance indicators are discussed in section 4.4.

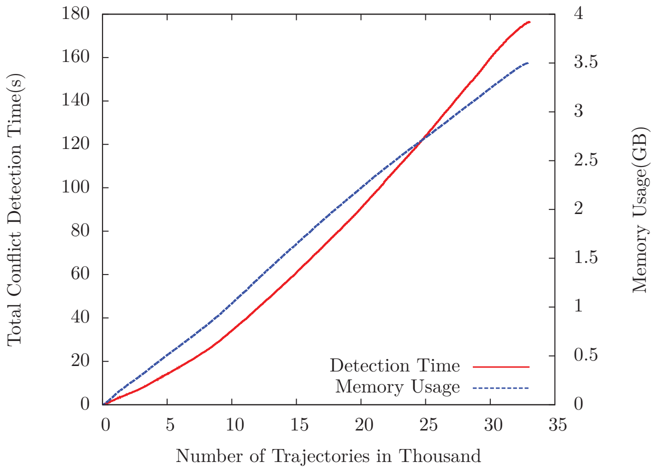


Figure 4.18 – Detection Times for European Sample with Tiling Algorithm

4.4 Performance Indicators

Worst case performance of the provided algorithm is $\mathcal{O}(N^2)$ in particular for the case that all N trajectories are identical. Building the tree would not occlude any trajectory pairs from conflict probe. Thus, every single trajectory needs to be checked against each other, as also done by the baseline algorithm. However, as can be read from the promising results, in practice the algorithm performs very well.

The main influencing factors on the algorithm's run-time are discussed in following sections.

4.4.1 Number of Trajectories

Since the conflict probing is done sequentially the number of trajectories has influence on the total run-time. Talking about calculation times per trajectory, the number of trajectories has only weak influence. If the trajectories are well distributed in the 4D-airspace, calculation times are low even for a high number of trajectories. If trajectories are identical, computation of conflicts is complex for few trajectories already.

The only weak dependence of trajectory count from calculation times per trajectory is proven by performing conflict detection on a quadrupled German traffic sample. Initially, the German traffic sample does not leave the Northern and Eastern hemisphere. The quadrupled German traffic sample is generated by first mirroring traffic at the zero meridian, and mirroring the resulting traffic at the equator. This leaves densities and length of trajectories constant, but quadruples the total trajectory count. As shown in fig. 4.19, the memory is four times higher than the memory used for the German traffic sample. In contrast, the computation time per trajectory nearly stays the same with ∼2.3 ms per trajectory. The additional ∼0.5 ms per trajectory may be a result from the larger arrays for trajectories and conflicts, e. g., when verifying if a conflict has already been identified before.

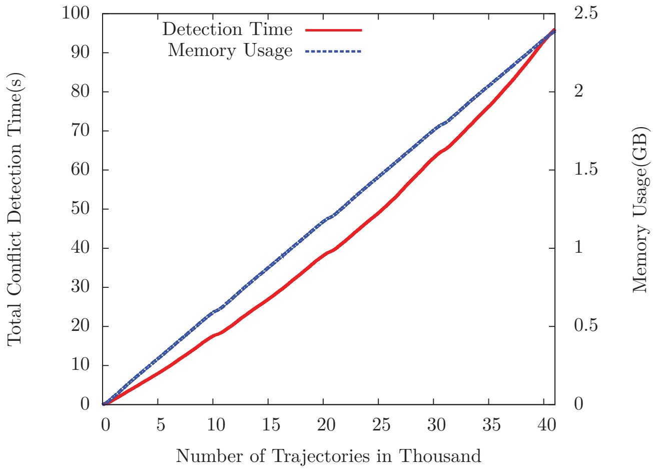


Figure 4.19 – Detection Times for Quadrupled German Scenario

4.4.2 Density of Trajectories

Dense areas with many trajectories have a higher potential for conflicts. On average, the generated tree has a higher depth for denser occupied airspace. Thus, higher density has a negative impact on computation times per trajectory.

4.4.3 Number of Trajectory Sampling Points

The number of trajectory sampling points increases computational work when calculating intersections, e. g., with tile boundaries. Effects can be illustrated by performing conflict detection on a trajectory set containing more trajectory points. Therefore, the German traffic sample was densified using WGS84-interpolation to a minimum of one trajectory point each 0.5 NM. Leaving all other parameter unchanged, average and standard deviation of trajectory points is increased to 599 ± 244 compared to 47 ± 19 originally.

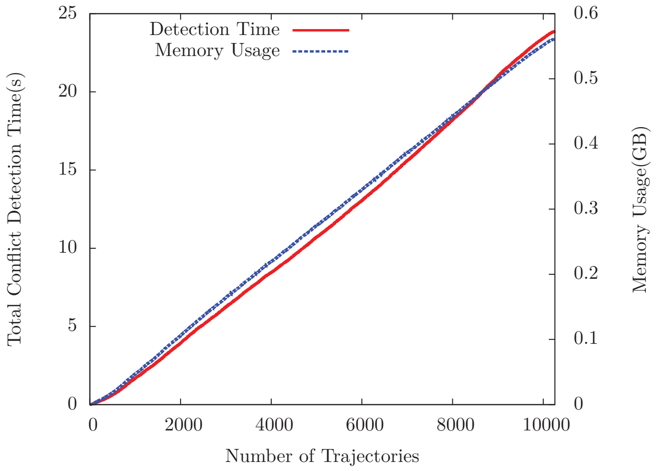


Figure 4.20 – Detection Times for Increased Number of Sampling Points

Figure 4.20 shows that memory usage stays nearly unchanged while conflict detection time increases by ∼29 % for 1270 % of the initial sampling points. Thus, the link between number of sampling points and calculation times is not very strong. The fact that memory usage even decreases slightly can be explained by the WGS84-interpolation while the *NDMap* performs linear interpolation internally.

4.4.4 Length of Trajectories

The trajectory length has significant influence on computation times because long trajectories affect many tiles. Thus, a doubled length of a trajectory is comparable to two single length trajectories in terms of computational burden.

4.4.5 Number of Potential and Real Conflicts

Obviously, a high number of conflicts in a scenario increases computational work. The average depth of the hexadecimal tree gets higher, and number of final conflict detection probes increases.

4.4.6 Summary of Performance Indicators

Finally, coming back to the difference of computation times between German and European sample, scenario density is probably not the reason because 10 000 trajectories in Germany is a higher average load than 33 000 trajectories over whole Europe. Main reasons for the worse performance of the European sample are likely the nearly 3 times longer trajectories and the higher number of sampling points. Also the allocated memory is high compared to the German sample. About 106 KiB of memory was used per trajectory.

4.5 Performance Optimization for Conflict Detection

This section discusses the choice of minimum tile sizes in order to guarantee a mandatory separation $\vec{S}$. As described in section 3.3.10

on page 104 the minimum tile size needs to be at least as big as the minimum separation $\vec{S}$ in order to detect all conflicts, but might also be bigger. Therefore, detection runs were performed with different configurations of the hexadecimal tree to find the optimum set of parameters. In this section, all runs were performed using the German traffic sample.

4.5.1 Lateral Tile Size Optimization

Starting with the lateral tile size, fig. 4.21 shows detection times and memory allocation for lateral separation distances from 5 NM to 20 NM. The detection time increases apart from some jitter monotonically with larger lateral tiles. The memory usage decreases slightly with increasing lateral size of tiles. The figure also shows the average number of penetrations and fly-bys per tile. Especially an increasing number of fly-bys per tile is noticeable. As depicted on fig. 3.8 on page 96, fly-by zones get larger with increasing lateral separation size. The number of penetrations stays constant. The reduction of tree depth counteracts the increasing number of fly-bys, resulting in an overall decreasing memory usage. Summarizing, it is beneficial for run-time to set the minimal lateral tile size to the lateral conflict size. Using larger tiles for the leaves reduces the necessary memory slightly, but also increases calculation times.

4.5.2 Vertical Tile Size Optimization

The results look different when changing vertical tile size instead of lateral size as plotted in fig. 4.22. While memory usage decreases slightly for increasing vertical size, detection times increase. More eye-catching are three steps where memory usage and calculation time decrease simultaneously. This behavior can be explained by having a closer look at table 4.1. The altitude dimension needs the lowest number of subdivisions (6.66) to get down to minimum tile size. By increasing the smallest vertical tile size, the number of necessary subdivisions is further decreased. Table 4.3 shows the necessary number of subdivisions for increased Vertical Tile Size

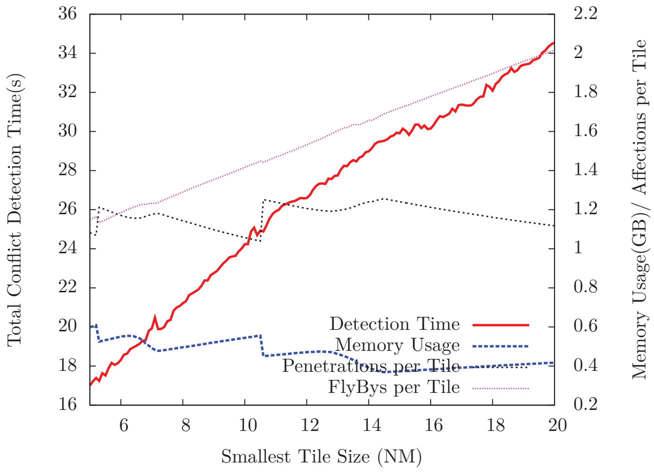


Figure 4.21 – Variation of Tile Size

(VTS). Obviously, the steps in fig. 4.22 are placed where the number of vertical subdivisions decreases by one.

Table 4.3 – Vertical Dimension with Increased Vertical Tile Size

Min	Max	VTS	D*
-1000	100 000	1000	6.66
-1000	100 000	1578	6.00
-1000	100 000	3156	5.00
-1000	100 000	6312	4.00

* with $D=\log_2\left(\frac{\max-\min}{\text{VTS}}\right)$

When the vertical subdivision downto minimum tile size is finished in the tree, the whole tree starts getting thinner. Each node has a maximum of 8 instead of 16 children. This is beneficial for memory and run-time. Summarizing, a starting value of 1000 ft is a good choice. Another efficient opportunity is using the next point where the subdivision count decreases, namely 1578 ft or slightly above to be on the safe side. Using even higher minimum

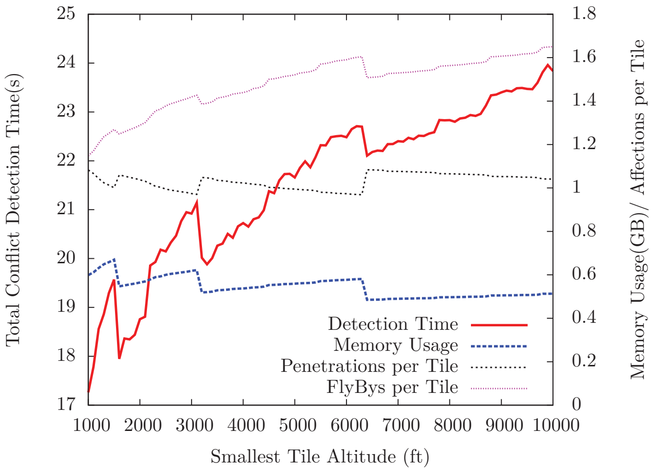


Figure 4.22 – Variation of Altitude

values is not beneficial for overall run-time. The approach of taking the subdivision positions into account is respected by the modified balancing algorithm in section 4.5.4.

4.5.3 Time-based Tile Size Optimization

The effects of varying the time-based tile size are shown in fig. 4.23. With a very fine granularity of one second, both memory usage and computation times are high. Too large tiles result in low memory usage, but increased computation times. However, the initially chosen 90 seconds representing a typical time-based separation between two succeeding aircraft with approach speeds are not the best choice in terms of computation times. Table 4.4 shows the step-creating increments of time depth. In order to increase the calculation speed a minimum tile duration below 84.375 s should be selected. Figures 4.24 and 4.25 show the results for a tile duration of 80 s for Germany and Europe respectively. Average detection times are 1.61 ms for Germany and 4.97 ms for Europe per trajectory.

Table 4.4 – Time Dimension with Increased Tile Duration

Min	Max	Duration	D*
0	86 400	90	9.91
0	86 400	84.375	10.00
0	86 400	168.75	9.00
0	86 400	337.5	8.00
0	86 400	675	7.00

* with $D=\log_2\left(\frac{\max - \min}{\text{Duration}}\right)$

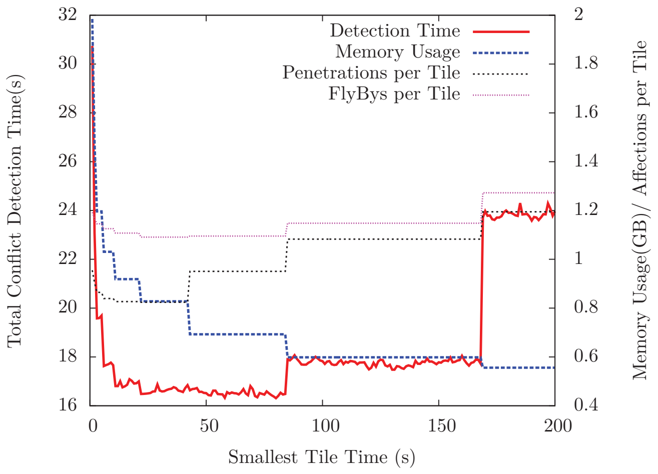


Figure 4.23 – Variation of Time

4.5.4 Balancing the Tree

The results in this section are based on a minimum time duration of 80 s, as experienced in the last section.

As described in section 3.3.9, balancing of a tree adapts starting intervals of each dimension in a way that ensures reaching minimum tile size for all dimensions at the same level of depth. The results from an automatically balanced tree for the German traffic sample are shown in fig. 4.26. While using about 25 percent more memory

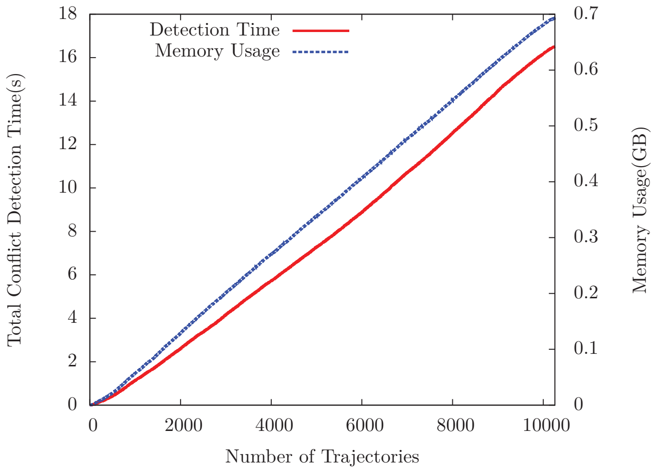


Figure 4.24 – Results for 80 Seconds Tile Duration (Germany)

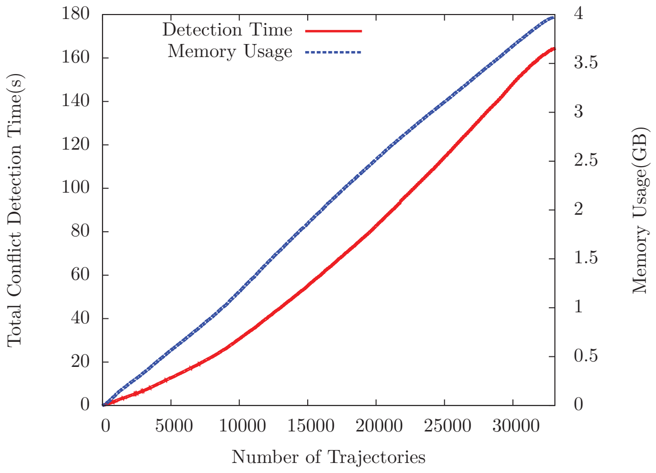


Figure 4.25 – Results for 80 Seconds Tile Duration (Europe)

than the unbalanced version (compare fig. 4.17), calculation time remains the same. Balancing on the European traffic sample also results in 25 percent higher memory usage and 7 percent worsened calculation speed.

All in all, the balancing does not seem to be reasonable in terms of possible performance gain.

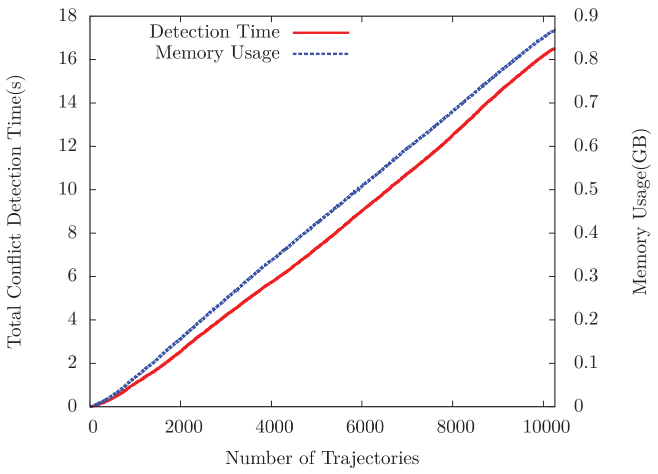


Figure 4.26 – Results from Automatically Balanced Tree

Trying to solve the trade-off between memory and computation times another balancing approach was developed. Instead of balancing all dimensions to the same tree depth, each dimension is handled independently in the new approach.

Table 4.5 – Modified Balancing Parameter

		Initial		*Modified*		
Dim.	Min	Max	D	Max	D	S
x	-180	180	12.08	503	13.001	$^5/_{60}$
y	-90	90	11.08	252	12.001	$^5/_{60}$
z	-1000	100 000	6.66	127 089	7.001	1000
t	0	86 400	10.08	163 954	11.001	80

As was experienced with the vertical and time-based optimization in sections 4.5.2 and 4.5.3, an integer number of subdivisions seems to be preferable. Therefore, the new balancing algorithm enlarges starting intervals in order to achieve an integer number of subdivisions. Since overshooting is critical, the next higher integer is approximated with an offset of +0.001. Thus, the smallest tile size stays just above the separation minimum. Table 4.5 shows the new parameters calculated by the modified balancing algorithm. As used before, D is the number of subdivisions, given by $\log_2((\max - \min)/S)$.

As depicted in fig. 4.27 the modified balancing algorithm does not provide the expected benefits. Compared to the unbalanced version, run-time is increased by 14.3 % from 1.61 ms to 1.84 ms per trajectory while increasing necessary memory by 23 % to 83 KiB per trajectory.

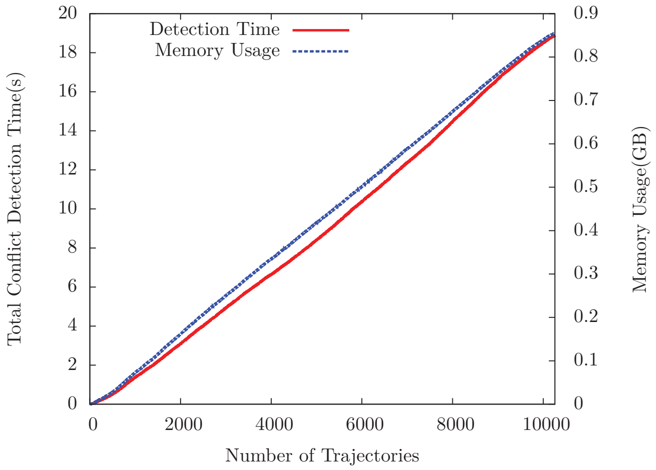


Figure 4.27 – Results from Modified Balancing Algorithm

4.5.5 Focus on Aircraft

As described in section 3.6, run-time and memory usage can be drastically improved when the collision detection focuses on a subset of trajectories. An exemplary application is the responsibility of an air traffic controller for a predefined subset of aircraft as described in section 4.7.2.

Figure 4.28 depicts results for trial runs with 1-200 selected aircraft based on the German traffic sample. Aircraft are selected by random from the traffic sample and registered in the *NDMap* before loading the scenario. The random selection of aircraft impedes a monotonic increase of total detection time. Selecting only one aircraft, generation of the whole tree takes 1.1 seconds while using 401 KiB memory. Focusing on six aircraft as done in the Luftraummanagment 2020 (LRM2020) project, the tree generation including conflict detection for the selected 6 flights takes 1.5 s.

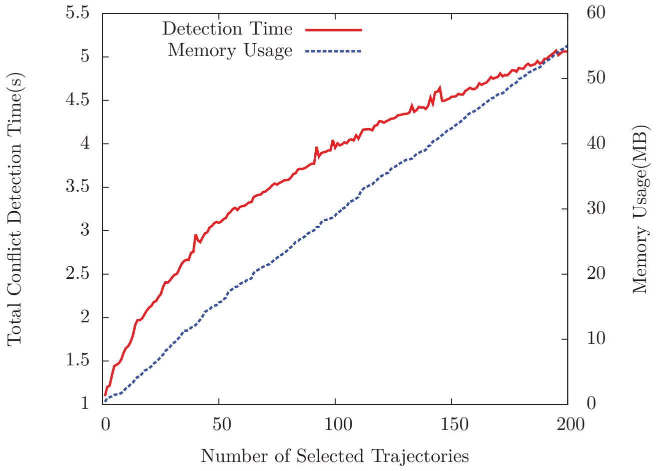


Figure 4.28 – Algorithm's Performance for Different Numbers of Selected Aircraft

Selecting aircraft increases performance without creating a big overhead. The break-even point in terms of performance compared to the standard setup is reached with ∼7600 selected aircraft, where the calculation takes ∼18 s and uses 583 MiB of memory. Thus, if a focus on a subset of aircraft can be set, this should be done unless the subset contains more than 7600 aircraft for the German traffic sample.

4.5.6 Shrinking the Root Tile

This section investigates if starting with large root tiles penalizes the algorithm's performance significantly. Selecting the whole world as root tile for the German traffic sample with latitudes in [46.5°, 55.2°] and longitudes in [5.5°, 15.9°] (table 4.2 on page 143) may result in an unnecessary long search path for conflicts.

Therefore, the conflict detection was executed with a shrunk root tile with latitudes in [46.5°, 55.2°], longitudes in [5.5°, 15.9°] and altitudes in [-500 ft, 50 000 ft]. The unexpected results are depicted in fig. 4.29. The shrunk root tile results claim 45 % more memory usage and a 37 % increase of calculation time. Turning on the balancing algorithm produced even worse results.

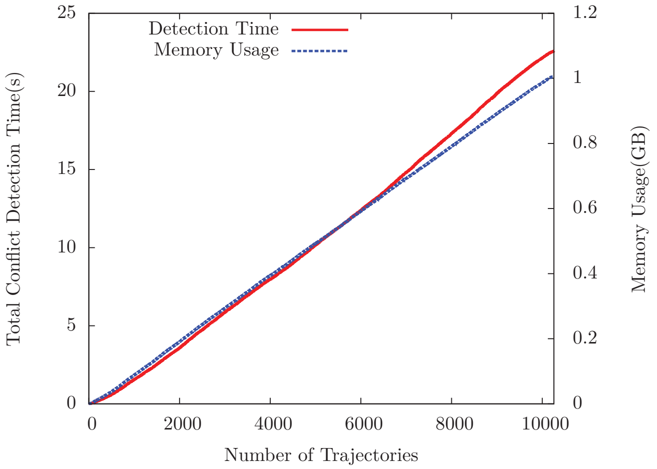


Figure 4.29 – Results from Shrunk Root Tile

Trying to explain the increase in memory usage and run-time, table 4.6 lists the number of generated tiles in each depth level for both defined root tiles. As expected, the shrunk root tile needs less depth than the Earth root tile (10 versus 11), and the shrunk root tile generates a thicker tree close to the root (16 versus 4 nodes in level 1). In total, the shrinking increases the number of generated tiles significantly by 45 %. Since the only values changed are the number of subdivisions for each single dimension, the reason needs to be the shift between the dimensions against each other. One possible explanation is that it might be beneficial to let the time dimension split before all other dimensions split occupied airspace. However, trials increasing all dimensions but time to magnify that effect were not successful.

Table 4.6 – Number of Nodes - Earth vs. Shrunk

	Earth-Root	Shrunk-Root
0	1	1
1	4	16
2	8	256
3	32	3388
4	116	39 838
5	1754	364 277
6	15 050	1 114 110
7	65 182	617 153
8	249 573	537 019
9	602 255	445 249
10	878 849	374 011
11	593 092	N/A
Total	2 405 916	3 495 318

The observed behavior indicates that the algorithm is sensitive against different starting conditions. It also illustrates that the proposed input balancing is not perfect – extending the shrunk tile to Earth dimensions obviously is a better option.

However, the conditions leading to the best calculation performance are nontrivial and for sure also not independent from the input data. Several trials with different starting intervals were performed with the result that the Earth intervals as initially chosen are close to optimum. Future work on that topic may result in an even better input balancing algorithm.

4.6 Comparison with Octrees

Since the implementation of the *NDMap* is generic concerning the number of dimensions, detection runs can easily be performed with less than 4 dimensions. This section shows results from two 3D-octree configurations. In order to get the same final conflicts as in the 4D version, the external conflict function still checks conflicts in all 4 dimensions.

Figure 4.30 presents the results using an octree holding latitude, longitude, and altitude as proposed by Hildum and Smith (compare section 2.4.3 on page 63). Comparing the outcome with the 4D-configuration in fig. 4.24, the procedure needs only 12 % of memory, but uses a 36 times higher calculation time.

Generation of the data was a bit difficult. The final conflict detection needs a 4-dimensional vector to validate the conflict. However, if time is not included in the tree, the corresponding 4D-point cannot be identified clearly without ambiguity. Therefore, the *NDMap* was used in 4D mode. In order to avoid benefits from the time dimension, a minimum tile duration of 86 400 s (i. e., one day) was applied.

Figure 4.31 shows results from another octree configuration of the *NDMap* holding latitude, longitude, and time. This time, the *NDMap* was configured in real 3D mode. Using about 364 MiB of memory, this configuration needs about 220 % of the time from the reference 4D-hexadecimal configuration in fig. 4.24. The relative good performance indicates that two aircraft rarely cross above each other at the same time. It also proves that the provided algorithms perform well in 3 dimensions.

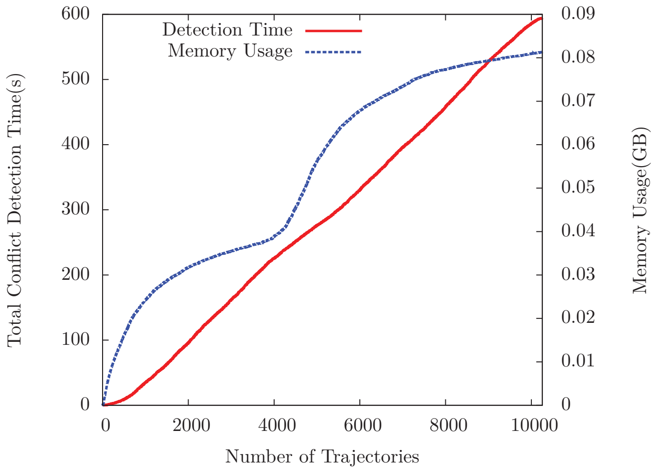


Figure 4.30 – 3D-Octree Latitude/Longitude/Altitude

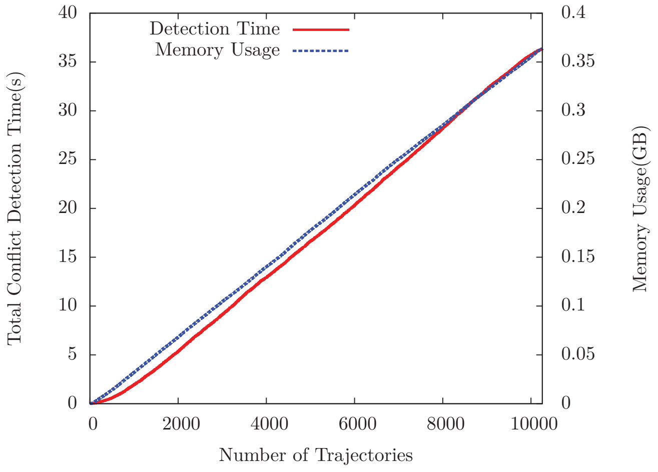


Figure 4.31 – 3D-Octree Latitude/Longitude/Time

4.7 Application in Projects

The software module *NDMap* has already been used by several projects at DLR. The applications differ in applied time horizons and traffic complexity significantly. This section gives short descriptions for some of the projects and especially focuses on the application of the *NDMap*.

4.7.1 Future Air Ground Integration

The Future Air Ground Integration (FAGI) project has taken place from 2007 to 2009 and used a first version of the *NDMap* module for STCA and TBO (Kuenz and Edinger, 2010a,b). The project focused on providing a transition from today's vector-based control of aircraft into a TBO environment supporting both FMS equipped and unequipped aircraft.

The FAGI concept distinguishes aircraft by their equipage:

- Aircraft equipped with a 4D-FMS are capable to predict and fulfill 4D trajectories board-autonomously. The expected accuracy is $\pm 6\,\mathrm{s}$ as one out of nine aircraft in European traffic was capable in 2007 (Smedt and Berz, 2007).
- Conventional equipped aircraft without an onboard 4D-FMS are incapable of high precision board-autonomous 4D-guidance.

The key element of the proposed 4D concept for an advanced Terminal Maneuvering Area (TMA) handling is the late merging of all arriving traffic. Before merging, arriving aircraft are separated procedurally by staggering them laterally in an Extended TMA (E-TMA) allowing each aircraft to fly its individual speed and altitude profile. When entering the E-TMA, a time constraint for the merging point is assigned to each aircraft. The late merging point for all aircraft is located just before the final approach. The early assignment of time constraints enables arriving aircraft to fulfill the requested time in an efficient way, i. e., speed adaptation. Therefore, the E-TMA is rather big (80 NM to 120 NM radius). If speed variation is not enough to reach the constraint, strategic path stretching can be used to delay aircraft further. Parallel routes from every entry point to the late merging point enables faster aircraft to overtake slower aircraft (fig. 4.32).

Aircraft not entering near a static E-TMA entry are guided by means of dynamic routing. Depending on the equipage, there are two different approaches possible:

- 4D-equipped aircraft are able to generate their own optimum 4D-trajectory board-autonomously. They fulfill a ground predicted time constraint at late merging with high precision, ideally without any interventions after the negotiation process with ground control. Equipped aircraft fly direct routes to the late merging point. Aircraft violating their contract with ground control are supposed to be degraded to unequipped aircraft and thus follow the trombone routing
- Unequipped aircraft are not able to fulfill the given constraints on their own. Therefore, they are supposed to be guided by a ground based 4D guidance system as described by Kuenz et al. (2008). The ground-based guidance system generates speed vectors and, if necessary, also shortcuts to reach the target time. To get them precise in time at the merging point, they are guided along a trombone approach allowing rather late correction of merging times.

Aircraft flying the trombone can be delayed to allow insertion of short term departures and simplify handling of emergency situations.

Since a trajectory was generated for each aircraft flying trajectory based conflict detection was applied for the E-TMA using the *NDMap*.

However, since not every aircraft was FMS-equipped, an accurate adherence of the trajectory could not be assumed for all aircraft. Therefore, short term conflict detection was implemented using current aircraft state vectors' extrapolation into near future. Based on current 3D-position and 3D-velocity, position was extrapolated by 2 minutes into future. The corresponding trajectory holding current and future position was fed into the *NDMap* in order to provide STCA functionality to the air traffic controllers.

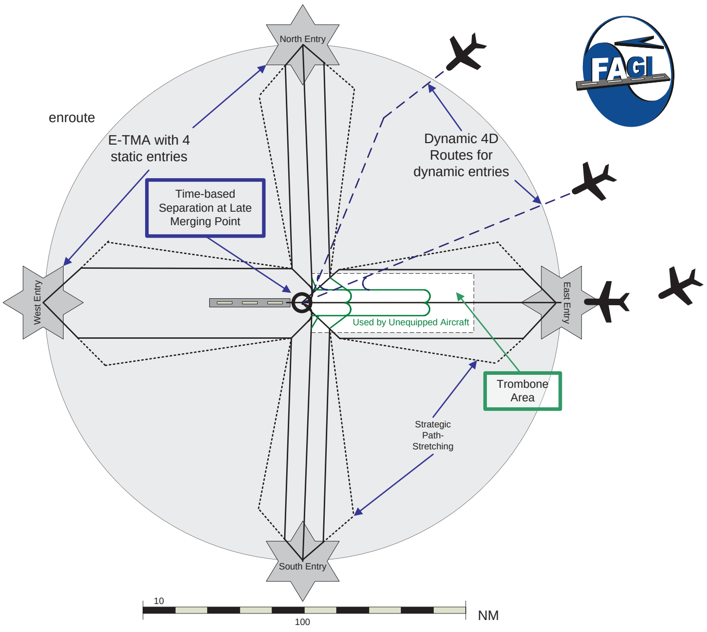

North Entry
South Entry
West Entry
East Entry
enroute
E-TMA with 4 static entries
Dynamic 4D Routes for dynamic entries
Time-based Separation at Late Merging Point
Used by Unequipped Aircraft
Trombone Area
Strategic Path-Stretching
FAGI
10
100
NM

Figure 4.32 – The FAGI Concept

Properties of the applied STCA are:

- Very short trajectories with usually only 2 points, duration of 2 minutes and a trajectory length below 20 NM.
- Very high update rate, depending on update rate of aircraft states.
- Low conflict likelihood.

4.7.2 Luftraummanagement 2020

Sectorless ATM is the main concept element of the project LRM2020 (Korn et al., 2009; Birkmeier et al., 2010). Instead of assigning controllers to geographical areas, controllers are assigned directly to several (e. g., 4-6) flights accompanying them during their whole stay in upper airspace. This should reduce the number of necessary controllers as well as communication overhead when switching sectors. Since assigned aircraft are in geographical independent areas, the controller needs powerful assistance tools allowing him to handle his traffic safe and efficiently.

Conflict detection is performed with a mid-term time horizon. While a short term horizon reduces efficiency and increases urgency to react, a long term horizon is not beneficial due to both uncertainties and for ATM controllers unusually early conflict warnings.

Since controllers have only to deal with aircraft that are within their responsibility, the *NDMap* can be optimized for every single controller by focusing on the assigned aircraft. That way, memory usage and detection times are drastically improved, compare section 4.5.5.

4.7.3 Volcanic Ash Impact on the Air Transport System

The Volcanic ash impact on the Air Transport System (VolcATS) project tries to increase safety and improve adaptation of ATM under volcanic eruption influence (Schlager et al., 2012). Project goals are:

- Satellite-based identification and short-term forecast of ash-free airspace.
- Installation and proving of miniature sensors for SO_2 and particles for scheduled flights allowing an early warning when flying into an ash cloud.
- Development of ATM procedures for fast adaption of airspace in case of volcanic ash issues.

In times of the Eyjafjallajökull crisis in 2010, ash contaminated airspace was clustered into three groups:

- Areas of low contamination with $0.2\,\mathrm{mg/m^3}$ to $2\,\mathrm{mg/m^3}$.

- Areas of medium contamination with 2 mg/m^3 to 4 mg/m^3.
- Areas of high contamination with more than 4 mg/m^3.

Figure 4.33 shows conflicts between air traffic and the ash cloud generated by Grímsvötn in May 2011. The different classification of contamination is drawn in different colors.

Since the forecast of ash contamination is not accurate enough today, the classification is not taken into account for the definition of no-fly zones. Instead, aircraft are not allowed flying into visible ash zones.

However, improvement of the forecast accuracy and onboard sensors will allow a more sophisticated adaption of ATM in times of volcanic eruption in the future. The gathered information shall be merged in a repository center (Vujasinovic, 2012a).

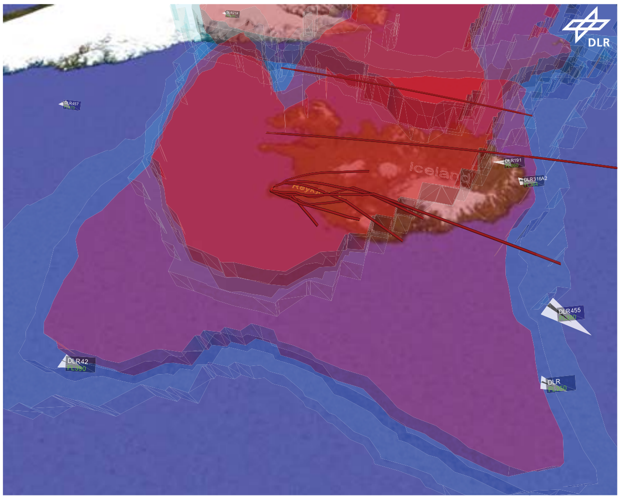


Figure 4.33 – Conflicts between Air Traffic and Volcanic Ash Cloud

A big difference to other aforementioned projects is the classification into designated severity classes. If an ash cloud cannot be avoided at least the trajectory with minimum overall contamination is desired. This does not necessarily mean that areas with high contamination need to be avoided hardest though. Leaving an ash cloud by shortly breaking through a high contamination area may be preferable to a long stay in medium conditions. Since the project is still in its beginning, progress concerning modeling the traffic and contamination is small so far. However, first ideas exist.

The most vulnerable parts of an aircraft to be damaged by volcanic ash are engines. Ash contains silicium that melts at hot temperatures in the combustion chamber. The hot ash then cools down on the turbine blades and may stall the engine (Vujasinovic, 2012b).

Thus, there is a link between contamination severity and engine temperature. For the sake of an interesting example, let us assume that designated combinations of engine temperature and ash concentration are acceptable. The idea is to use the severity parameter as an additional dimension to model both ash concentration and engine temperature. That way, new conflict rules can be established. Table 4.7 illustrates how the new dimension can be defined. Values from 0 to 2 are assigned to ash concentrations from high to low. Values from 0 to -2 are assigned to engine temperatures from high to low. Based on the absolute difference, the new dimension raises a conflict for an exemplary separation value of 2.5 if and only if:

- Engine temperature is high; *or*
- Ash concentration is high; *or*
- Ash concentration and engine temperature are both medium.

Table 4.7 – Example Setup for Severity Dimension

		Airspace Contamination		
		High (0)	Med (1)	Low (2)
	High (0)	0	1	2
Engine Temp.	Med (-1)	1	2	3
	Low (-2)	2	3	4

Following the example, medium engine temperature is acceptable in low contamination only, while low engine temperature is acceptable in low and medium contaminated airspace. However, trials in an engine test bench are necessary to show if above made assumptions are reasonable.

Of course, above described procedure is just a rough estimation for the real effects. The modeling also can be done in much more detail adding several new dimensions covering properties like number, size, shape, weight, melting point, and chemical composition of particles. In order to take advantage from the *NDMap* an appropriate representation of data is necessary allowing comparison with a separation minimum, as performed for the severity example. Logical combination of parameters shall then be done within the external conflict function.

4.7.4 Supercooled Large Droplets Icing

The Supercooled Large Droplets Icing (SuLaDI) project investigates icing problems in aviation (Voggenreiter and Etzenbach, 2012). Icing is especially dangerous for measurement equipment aboard of aircraft by distorting signals, engine inlets reducing thrust, and wings destroying the aerodynamics by increasing drag and reducing lift. While Jeck (2002) gives a definition of standard icing conditions, SuLaDI focuses on untreated Supercooled Large Droplet (SLD) conditions that represent a very specific and dangerous hazard to aircraft.

Assuming an accurate icing area forecast, the *NDMap* is used to avoid icing areas. Since the icing zones are additional restrictions to already existing constraints from ATM, conflict resolution from icing areas also needs to respect the baseline ATM constraints in a safe and efficient way. In particular, maneuvers avoiding icing areas shall not affect other traffic.

Icing zones are modeled as polygonal areas with minimum and maximum altitude and time as hazard for all aircraft trajectories. Furthermore, icing areas can be moved by assigning movement trajectories.

Figure 4.34 presents conflict solution from an exemplary SLD icing area.

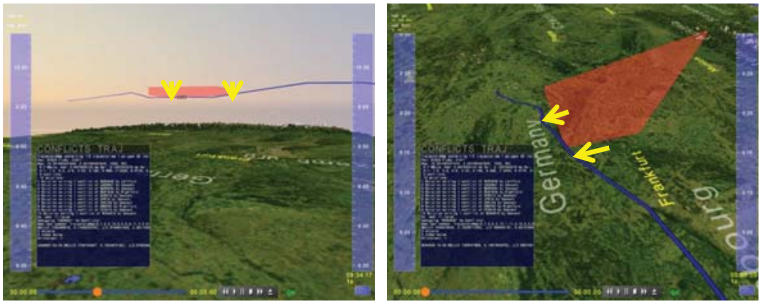


Figure 4.34 – Vertical and Lateral De-Confliction from SLD-Icing Area

As proposed already for the ash cloud in the previous section, adding one or more new dimensions holding icing conditions and aircraft state/equipage may be beneficial to model more complex behavior, e. g., allowing an aircraft equipped with special de-icing equipage to fly through a weak SLD area.

4.7.5 4 Dimensional-Contracts - Guidance and Control

The 4DCo-GC project tries to solve the conformance monitoring problem in ATM (The 4DCo-GC Consortium, 2013; Joulia and Le Tallec, 2012). Nowadays, air traffic control is responsible for monitoring the aircraft's conformance with given instructions based on the available information on ground. Depending on the source of information (e. g., primary radar, SSR, ADS-B) position data is less or more accurate.

Aircraft usually know better about their own position using satellite-based positioning systems like United States' Global Positioning System (GPS), Russian's Globalnaja Nawigazionnaja Sputnikowaja Sistema (Glonass) and European's Galileo. Furthermore, aircraft know their intent and whether current deviations from given instructions are on purpose (e. g., in order to fly a more efficient route) and therefore acceptable or if the guidance is physically unable to follow the instructions, e. g., given by a 4D-trajectory.

4DCo-GC puts the conformance monitoring responsibility into the aircraft. In order to distinguish between allowed and severe deviations from a trajectory, a contract defining safety and freedom margins is negotiated between ground control and aircraft. While ground control ensures that contracts of no two aircraft intersect, each aircraft has the task to stay in its assigned contract. For each point in time, the contract defines allowed cross-track deviations specified in nautical miles, along-track deviations as a time, and vertical deviations as an allowed altitude offset. As soon as an aircraft forecasts that it cannot comply anymore with the given contract, a new contract is calculated by the ground module.

Figure 4.35 shows how a contract is defined in the 4DCo-GC project. The whole blue area around each aircraft is called contract bubble. An aircraft has a time-continuous contract bubble at each time from start to end of flight. For each time, the contract bubble is safe from conflicts with all other contract bubbles. Contract bubble might touch each other, but not intersect. Contract bubbles are smaller around airports and bigger in less dense traffic situation like en-route areas.

Aircraft are prevented to fly to the border of their contracts by the green safety bubble (SB). Safety bubbles must stay within the contract bubble in order to avoid dangerous contract violations. Thus, the allowed position of each aircraft is limited to the yellow freedom bubble. The freedom bubble can be used by each aircraft to optimize efficiency of flight.

Assuming constant contract margins along time, conflict detection between contracts can be performed based on trajectories using a separation adapted to the size of contract margins. Variable contracts are much more difficult to handle, e. g., using morphing volumes moving along a trajectory.

The *NDMap* was used within 4DCo-GC for generation of a conflict free scenario based on the optimized Europe sample. Since the optimized sample contains nearly 29 000 conflicts, effective conflict resolution algorithms were necessary in order to create a

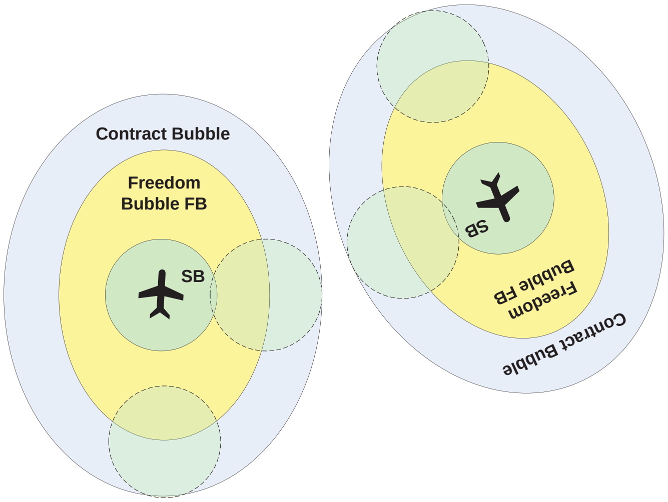


Figure 4.35 – Contract Definition in the 4DCo-GC Project

conflict free sample. Conflict resolution algorithms are described in chapter 5.

Chapter 5

4D Conflict Resolution

As described in section 2.5, conflict resolution can either be performed laterally, vertically or time-based. The *NDMap* implementation provides algorithms for all three types of conflict solution. All procedures use the trial-and-error method taking advantage of the high performance of the conflict detection algorithm.

Trial-and-error is not only facilitated by fast conflict detection times (i. e., times for adding a new trajectory) but also by fast deletion times of 0.4 milliseconds for the German and 1.6 milliseconds for the European sample. Thus, trajectories can be probed for conflicts in 2.0 ms (1.6 ms+0.4 ms) resulting in 500 trial-and-error runs a second for the German sample and 1000 ms/(5.0+1.6) ms = 151 probes a second for the European sample. Deletion times are that high for the European sample because, for instance, conflict deletion in the 29 k entries conflict vector is slow. Probing speed usually increases while reducing the number of remaining conflicts.

5.1 Global Trial-and-Error Conflict Resolution

Conflict resolution is a complex task. Several issues need to be taken into account when trying to solve a conflict:

- The right of way is either given by a set of predefined rules (e. g., the EFR from section 2.5.6 on page 73) or dynamically assigned based on optimization metrics.
- The maneuver avoiding a conflict may create new conflicts on the remaining route further downstream. Decision rules need to be defined if conflicts have different priorities (e. g., depending on the flight phase), and whether moving a conflict further downstream is an optimization and therefore a potential solution.
- Calculation of ambitious resolutions with minimum trajectory change is usually more efficient, but also increases computational effort.
- If multiple solutions are possible, the best solution needs to be identified. Metrics are necessary to determine the preferred solution.

Whether a conflict resolution is good or not needs to be well defined. As for the global optimization of whole ATM, broadly accepted metrics are necessary to define the global optimum. ICAO (2005) defines following 11 expectations also known as Key Performance Areas (KPA) for the global ATM in alphabetical order:

- Access and equity.
- Capacity.
- Cost-effectiveness.
- Efficiency.
- Environment.

- Flexibility.
- Global interoperability.
- Participation by the ATM community.
- Predictability.
- Safety.
- Security.

Since the expectations are not independent conflicts of interest need to be solved by a trade-off. An exception from the trade-off process is usually *safety*, since an acceptable safety is always highest priority in aviation. Besides safety, keeping efficiency high is one of the main factors when solving a conflict. However, also other KPAs are relevant. Since the bottlenecks in ATM are usually the airports, holding an earlier accepted required time of arrival at the destination airport is essential, ensuring the KPA *predictability*.

Regarding conflict resolution responsibility, there is also a trade-off between predictability and efficiency. Defining the right of way aircraft according to a rule set like EFR increases *predictability* but might not be the most efficient solution. The global optimum for this decision might even be reached if both aircraft are responsible to solve the conflict together, each solving a part of the conflict.

When performing trial-and-error conflict resolution, new probe trajectories need to be generated. Calculations can be performed with different accuracy. While the complexity of trajectories is not a problem for the *NDMap*, it might be an issue for the (external) trajectory prediction engine not being able to deliver aircraft specific efficient trajectories fast enough.

It is not beneficial to use high accuracy trajectories for high frequency conflict resolution testing for some reasons (Kuenz, 2011):

- Performance: The AFMS described in section 2.3.1 needs on average $\sim 100\,\mathrm{ms}$ to calculate realistic and flyable trajectories based on aircraft model, list of waypoints, altitude, speed and time constraints, and profile parameter. That is already pretty fast compared to commercial FMS. However, the prediction of vertical and speed profiles is an iterative process. Thus, if altitude or time (e. g., required time of arrival) constraints are tricky and difficult to meet, the prediction process might take up to 500 ms.

- Accuracy: Airborne trajectories are difficult to calculate by a ground tool, considering a global ground based conflict resolution tool. An aircraft knows best about its own performance. It provides up-to-date weather information by direct measurement in flight. Current aircraft weight is only known in aircraft, not on ground. Furthermore, airline specific procedures might not be published for ground tools. A study by Stell (2010) on estimating descent trajectories of real flights with a ground based tool illustrates difficulties localizing top of descent and predict meter fix times with high precision.
- Regeneration: When deviations occur in a flight, e. g., due to bad weather forecast, aircraft adapt their trajectories. These adaptations cannot be foreseen in the planning phase anyhow.
- Ambition: The best resolution maneuver is not the technical smart solution exactly fulfilling separation minima at any time. A more conservative approach with slightly worse efficiency including some safety margins should be preferred to abate remaining uncertainties.

Therefore, the *NDMap* provides built-in conflict resolution algorithms generating simplified trial trajectories. Obstacle avoidance maneuvers are generated

- Lateral, vertical and time-based.
- Based on performance indicators extracted from the initial trajectory: Track-change, climb, descent, accelerate and decelerate capabilities.
- If no resolution is found, deviation from original path is successively increased.
- If possible, avoidance maneuvers are generated for both conflicting objects. The better solution gets implemented.

- Providing simple metrics for solution rating. In the current version, route extension size and flight time are taken into account.

Multiple conflicts are solved in chronological order. Result of the resolution process is a list of avoidance maneuvers, each containing:

- Reference to the object implementing the solution.
- Reference to the first conflict being solved.
- Number of conflicts being solved. Especially lateral and time based solutions might solve (but also create) subsequent conflicts.
- Type of maneuver: climb x feet, descent x feet, avoid left by x NM, avoid right by x NM, advance or delay flight by x seconds.
- Trial trajectory.

Anyhow, once having selected the best solution out of all trial trajectories, a more precise AFMS trajectory should be generated in order to validate the solution.

No matter how a conflict is solved, trial trajectories are always generated for the complete remaining route. This way, downstream conflicts can be taken into account. In the probing process, the trial trajectory replaces the original conflicting trajectory. The updated conflict list is compared to the original list of conflicts. If the new situation is rated better, the probe trajectory is considered to be a solution. After probing with all foreseen trial trajectories is finished, the best rated solution, if any, is assigned as the solution of conflict.

5.2 Lateral Resolution

A lateral conflict resolution is shown in fig. 5.1. Calculation of the trial trajectories is mainly based on start and end of conflict. Trial trajectories are generated by moving the trajectory segment between start and end of conflict orthogonal to its direction. In

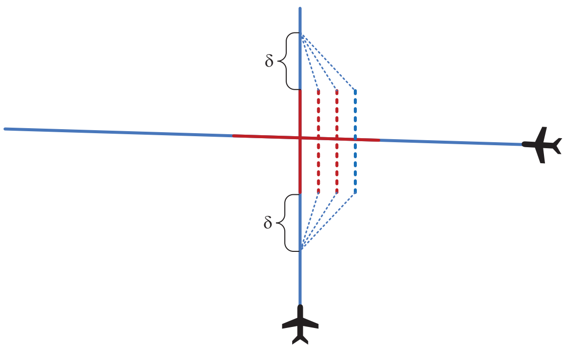


Figure 5.1 – Lateral Resolution of Conflict

order to keep the rest of trajectory stable, two additional points are inserted before start of conflict and behind end of conflict. The corresponding δ is set to 120 seconds in the current version.

Although the figure shows for simplicity only probe trajectories deviating to the right for one aircraft, probe trajectories are always calculated for both aircraft in both directions, aiming for the overall best solution. The parallel route offset starts with 0.5 NM and is increased by 0.5 NM until finding a solution or reaching a predefined maximum.

Lateral conflict resolution has side-effects on the time. The flight duration increases due to the lateral detour if speed stays unchanged. Both altitude and speed are assumed to be unchanged on the extra lateral segments. The resulting time variation produces a non-symmetric situation concerning left and right detour. For the conflict situation shown in fig. 5.1, the depicted solution is an efficient candidate. Compared to the time of original lateral intersection of trajectories, the aircraft from the south reaches the new lateral intersection later due to the detour, and the aircraft from the east earlier. A deviation to the left delays the lateral intersection for both flights.

If and how this algorithm solves a conflict mainly depends on intersection angle, phase of flights, and speeds of aircraft. Lateral

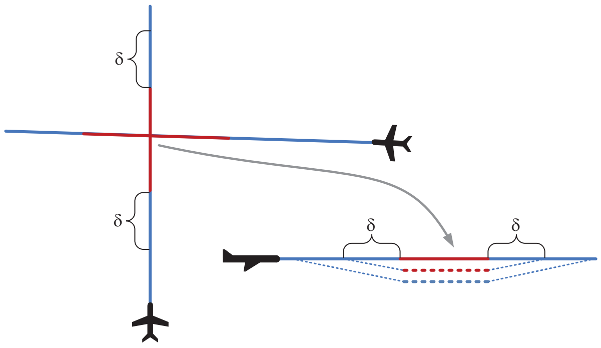


Figure 5.2 – Vertical Resolution of Conflict

conflict resolution is not applicable for conflicts in the vicinity of airports. Obviously a lateral route offset is not possible on the runway, but usually also not reasonable in the dense airspace around airports.

5.3 Vertical Resolution

The vertical conflict resolution is depicted in fig. 5.2. Basically, the same algorithm as for the lateral resolution is applied. The trajectory segment between start and end of conflict is moved in altitude. In order to leave the remaining trajectory unchanged, extra points are inserted before start and after end of conflict. The corresponding δ is set to 120 seconds per 1000 ft vertical deviation.

Again, probe trajectories are calculated for both aircraft in both directions. In contrast to the lateral resolution, the horizontal route length stays the same. Assuming not too large altitude steps, climb and descend speeds are of low importance, and times nearly stay unchanged. Altitude is increased and decreased by steps of 1000 ft as long as no solution was found or a maximum is reached.

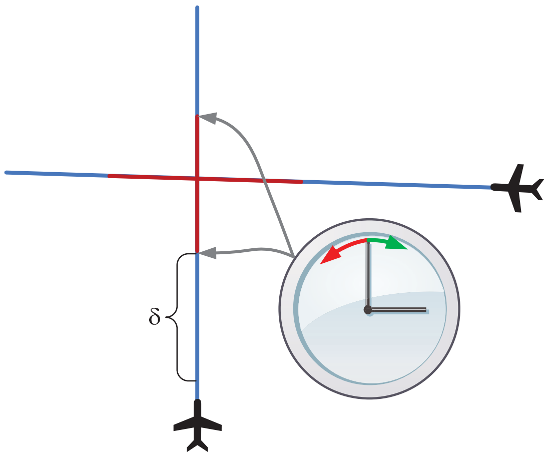


Figure 5.3 – Time-Based Resolution of Conflict

At the runway threshold, also the vertical resolution is not applicable due to the fixed threshold elevation. Depending on environmental constraints like noise sensitive areas, vertical resolutions can be reasonable in the vicinity of airports. In the case the aircraft is in climb or descent, the procedure needs small adaptations. Climbing aircraft insert a level flight interrupting the climb in order to dive below a conflict. Aircraft being close to their descent may bring forward part of it in order to create a lower conflict-resolving level flight.

5.4 Time-Based Resolution

The main idea about time-based conflict resolution is reaching the conflict point's original position at another conflict free time. The overflight time of a 3D-position (e. g., the start of conflict position in fig. 5.3) can be adapted by different means.

Assuming the aircraft is already airborne, overflight times can be revised by speed variations and path shortening or stretching. As discussed before, path shortening requires doglegs in the initial route to bypass, which are avoided in efficient route layouts. On the other hand, path stretching creates such doglegs and makes the initial efficient routing inefficient. Thus, speed adjustment is the proper mean to delay or advance an aircraft to avoid conflicts. However, assuming that original foreseen speeds are most efficient for the given aircraft, adjustments also degrade overall performance. Furthermore, the δ in fig. 5.3 needs to be rather large, e. g., at a typical speed of 300 kts, δ needs to be $\sim$30 min to adjust overflight time by 1 minute using a speed variation of 10 kts.

Assuming a strategic planning of air traffic where all aircraft are still on ground, a good way for time-based conflict avoidance is shifting the whole flight in time from departure to arrival.

5.4.1 Moving Whole Flights in Time

Shifting whole flights in time allows keeping flight duration constant. If a flight is shifted by few minutes only:

- Speed and altitude profiles can be assumed to stay unchanged compared to the original route.
- Wind may change slightly due to different flight time, however the influence is marginal and therefore negligible.
- Trajectories are as efficient as the initial trajectories.
- New trajectories can be calculated by decreasing/increasing times, no new profile needs to be calculated. Thus, prediction of trajectories is much faster.

Trying to adjust all trajectories of a scenario to get the optimum solution is too complex even with very fast conflict detection and avoidance algorithms. Therefore, intelligent pre-selection of aircraft is necessary.

Figure 5.4 demonstrates the high traffic density. It contains Frankfurt/Main traffic with 1365 flights only (i. e. 4.1 %), extracted from the European scenario. The coordinate system is time above latitude and longitude. The plot shows the whole day of traffic, with midnight on ground level and the end of day as upper delimiter. Trajectories are drawn in blue, with the conflicting segments in red. Figure 5.5 shows a close-up of the same data, scaling the time-axis by factor 20 and moving the ground layer to $t = 40\,000\,\mathrm{s}$. Since aircraft always climb when flying in XYT-notation, the flight direction is directly visible, and type of conflicts can directly be extracted from the illustration.

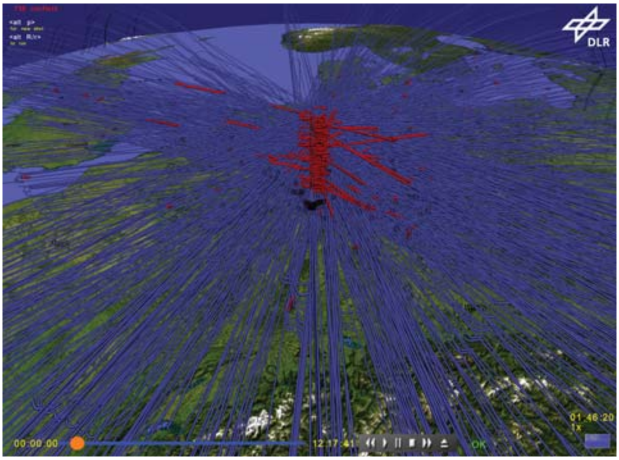


Figure 5.4 – Flights from/to Frankfurt-Main as XYT-Diagram 24h

5.4.1.1 Global Algorithm

The global algorithm moves trajectories in time in order to reduce scenario's conflict count. First of all, conflicts of the scenario are sorted chronologically. Cycling through all conflicting objects, time shifts are implemented on the corresponding trajectories using steps of 10 seconds up to a maximum shift time, e. g., 10 minutes. Afterwards, the trajectories are probed against the complete scenario. As soon as a conflict is solved by the shifting operation and the total number of conflicts for the corresponding flight is also reduced, the solution is considered to be the new trajectory for this flight.

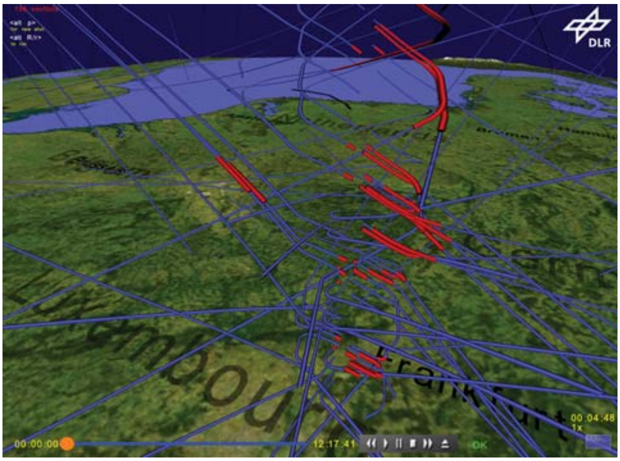


Figure 5.5 – Flights from/to Frankfurt-Main as XYT-Diagram around Noon

5.4.1.2 Recursive Algorithm

Also the recursive algorithm tries to solve conflicts chronologically. Conflicting aircraft are moved in time in steps of 5 s in order to resolve the conflict. Having already applied the global algorithm, this will typically not produce a proper solution because it is basically the same procedure as above described global conflict solution algorithm except from the finer granularity of 5 s. The main difference is the handling of generated conflicts. Since the conflict detection algorithm validates the whole trajectory for conflicts, follow-up conflicts are well known for each probe trajectory.

Probe trajectories generating more follow-up conflicts than having solved before are skipped by the global algorithm. In contrast, the recursive algorithm validates if generated follow-up conflicts can be avoided by time shift. As soon as conflict with a new aircraft arises this aircraft is also shifted by 5 s in the appropriate direction. Thus, that new conflict is immediately solved by re-establishing the original distance. Since this might produce another conflict with another aircraft, this algorithm is performed recursively and might move several aircraft simultaneously. The recursion is stopped at a

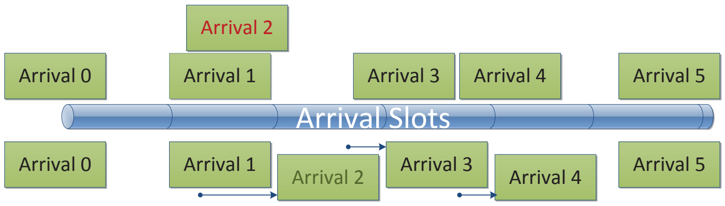


Figure 5.6 – Solving Conflicts Recursively

predefined depth (e. g., 10). Since moving an aircraft by 5 s might produce several conflicts all along the whole trajectory, recursive conflict resolution may become computational complex for high density scenario. As soon as a conflict cannot be resolved with the given recursion depth, disadvantageous operations are undone.

Figure 5.6 shows the general idea using the example of an arrival ladder. Above the arrival slot ladder, the conflict is displayed. Arrival 2 is in conflict with arrival 1. However, due to bad distribution of arrival 3 and arrival 4, there is no free slot for arrival 2. The described algorithm delays arrival 2 until detecting a new conflict with arrival 3. Since arrival 3 is the first aircraft producing a new conflict, it is also delayed. This produces another conflict between arrival 3 and arrival 4. By delaying all three arrivals furthermore, a solution can be found that is depicted below the arrival slot ladder. The solution implies that delaying the mentioned flights does not generate new conflicts somewhere else.

If delaying arrival 3 had generated a new conflict at its departure or in its en-route segment, the corresponding successor also would have been delayed.

Summarizing, this algorithm does not only focus on aircraft in conflict situations but also shifts aircraft non-involved in conflicts in order to make room for conflict solutions.

Furthermore, this algorithm implements a shared conflict resolution when possible. If no solution can be found for arrival 2 in fig. 5.6, the flight is nevertheless delayed as much as possible without creating new conflicts. Thus, arrival 2 can already solve one part of the conflict. The remaining part of the conflict can be potentially solved by bringing forward arrival 1.

5.4.2 Flight Duration Adaptation

Another mean of solving conflicts time-based is changing the duration of flight. Due to the assumption that the initial flight's efficiency is optimized, this degrades the performance. However, if traffic density is too high, constant flight duration constrains the number of solvable conflicts. In order to be still as efficient as possible, small adaptations should be preferred.

Flight duration adaptations should also take into account:

- Overall flight duration: long haul flights can handle higher adaptations than short haul flight.
- Aircraft model: some aircraft have a tight flight envelope regarding speed. Others allow changing speed in a bigger interval.
- Aircraft's efficiency: concerning global efficiency, it is more efficient to penalize efficient aircraft, and allow less efficient aircraft to fly their optimum route. However, this would privilege inefficient aircraft which does not sound like a fair idea. A well balanced trade-off between the KPAs needs to incorporate *access and equity* to ensure overall fairness.

5.5 Deconflicting European's Optimized Traffic

This section describes deconfliction trials based on the European traffic sample described in section 4.2.2 on page 139. Figure 5.7 shows the conflicts of the optimized scenario where each aircraft flies its preferred route. Conflicts are illustrated by red trajectory segments. Thus, short conflicts (e. g., rectangular intersections) are rather small, while two aircraft sharing the same route with a too small time offset are quite long.

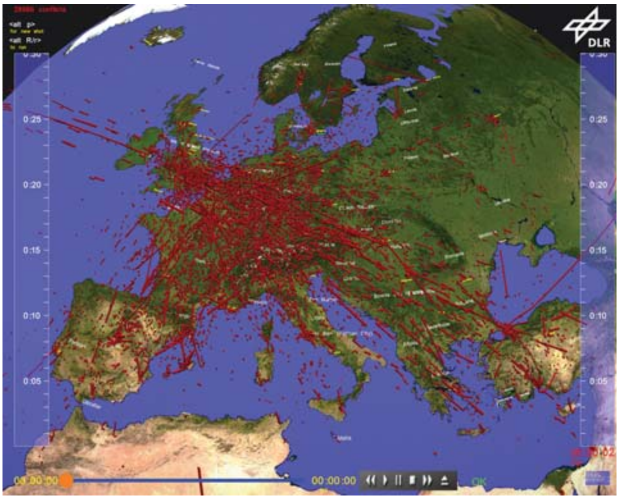


Figure 5.7 – 29k Conflicts in European Sample

The statistics in fig. 5.8 show that at least 9036 aircraft are free of conflict. 9181 aircraft have one single conflict on their gate-to-gate route. On average, a conflicting aircraft violates the separation to 2.4 other flights. This proves the importance of probing whole flights when solving conflicts.

The single flight generating 21 conflicts is non-experimental and non-military. The Airbus A321 leaves Lisbon around 5 o'clock in the morning with destination Munich. Only the cruise flight level is striking with FL344, provoking conflicts with aircraft in FL340 and FL350.

As an airport example, fig. 5.9 shows the situation in and above Munich. Also other big airports are visible in the background having dense concentrations of conflicts. As explained before, different conflict types can be solved in different ways. Therefore, conflicts are classified as shown in table 3.5 on page 120.

The distribution of conflicts to phases of flight is depicted in fig. 5.10. Even if nearly half of all conflicts are en-route, the conflicts in the vicinity of airports are more severe in terms of resolution. Since original departure times are taken into account

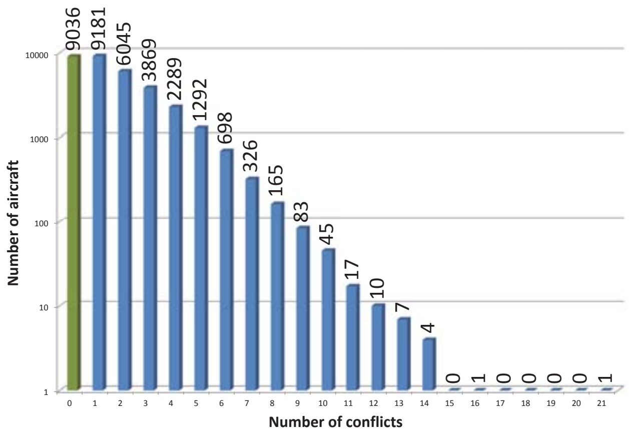


Figure 5.8 – Conflicts per Flight

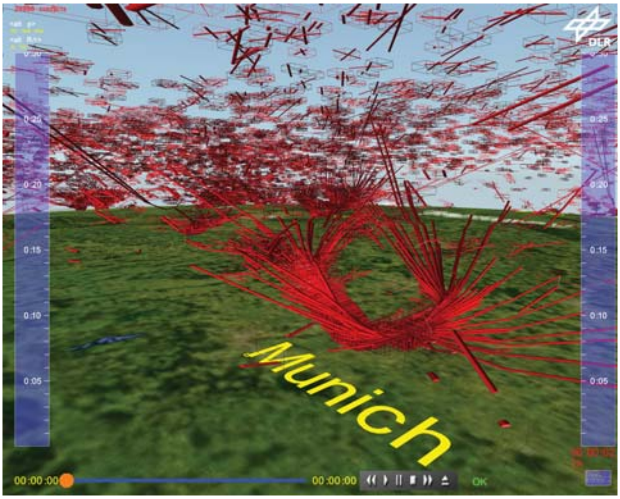


Figure 5.9 – Conflicts in Vicinity of Munich (Example)

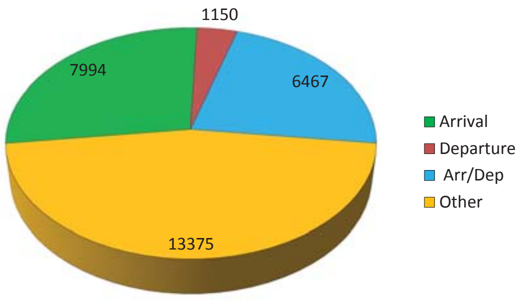


Figure 5.10 – Conflicts by Flight Phase

when optimizing the scenario, flights are separated at the departure runway threshold and thus the number of departure conflicts is low. Having departure conflicts at all can be reasoned by:

- Misinterpreted departure runways from the original data. Especially departures (and also arrivals) from parallel runways are difficult to distinguish.
- Inaccurate data with insufficient resolution.
- Too conservative conflict metrics. ICAO (2007) defines special procedures for, e. g., parallel or near-parallel runways (chapter 6.7: Operations on Parallel or Near-Parallel Runways) allowing a separation below 3 NM. This accounts for departure and arrival traffic.

Arrival and departure/arrival conflicts are as expected because arrival times changed with more direct routes and optimized speed profiles.

Depending on a conflict's classification different solution methods can be applied. As described before, there are obvious restrictions for conflict resolution close to runway threshold:

- Since the threshold height is fixed, it is not possible to provide a vertical solution.
- Lateral solution is feasible only for airports with multiple runways and when the arrival runway is not fixed.

- Time-based solution is the most promising way to solve conflicts on the thresholds.

The limited possibilities for conflict resolution at the runway thresholds advises to solve threshold conflicts first using time-based shifting of trajectories as described in section 5.4.1.1.

5.5.1 Airport-Focused Conflict Resolution

A significant reduction of conflicts already can be reached with a maximum time shift of [-30 s, +30 s]. The number of airport located conflicts reduces to ∼64 %. Figure 5.11 depicts the remaining conflicts for each category. Even though the algorithm focuses on conflicts in airport vicinity, also en-route conflicts are reduced as a spin-off from trajectory movement.

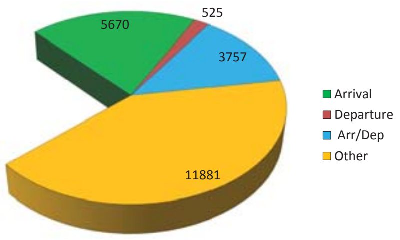


Figure 5.11 – Conflicts after Shift of ±30 seconds

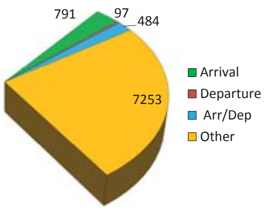


Figure 5.12 – Conflicts after Shift of ±10 minutes

Allowing a time shift of [-10 min, +10 min], airport conflicts are reduced to ∼9 % (fig. 5.12), still without touching the scenario's efficiency. Figure 5.13 illustrates how the number of airport related conflicts can be reduced with the corresponding maximum allowed time offset. The top of the blue bars represents the number of remaining airport-related conflicts.

Applying the recursive time shift described in section 5.4.1.2, conflicts at the airport can be further reduced to ∼6 % of initial airport conflicts (fig. 5.14) with a maximum recursion depth of 10.

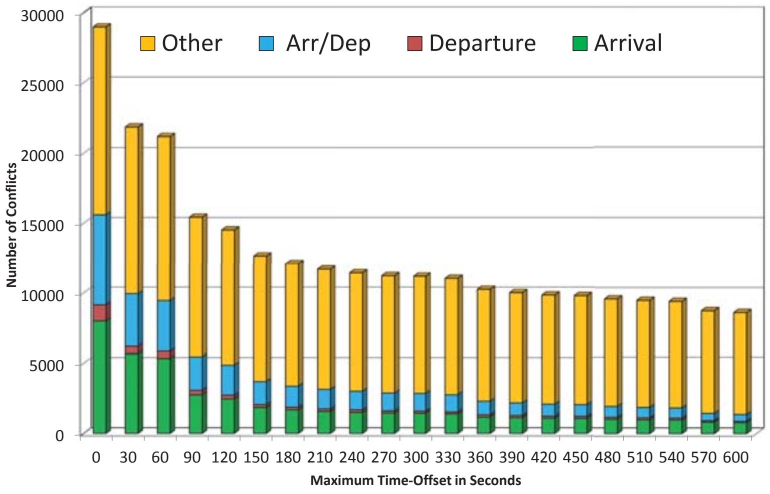
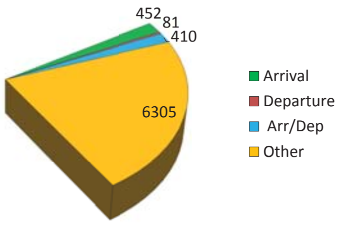


Figure 5.13 – Airport-Focused Reduction in Relation to Time-Shift



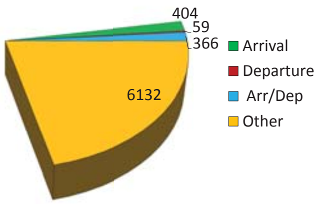


Figure 5.14 – Recursive Optimization Level 10

Figure 5.15 – Recursive Optimization Level 20

Figure 5.15 shows the results for an extended recursion depth of 20 leaving 829 remaining conflicts close to airports.

5.5.2 Global Conflict Resolution

For a global conflict resolution, all conflicting trajectories are probed by movement in time without focusing on airport conflicts. Starting scenario is the outcome from the airport-related conflict resolution described before. In order to avoid creation of new airport related conflicts as a possible side-effect from multiple en-route resolutions a weighting of conflicts is introduced. Therefore, airport-related conflicts are defined to be 10 times more severe than en-route conflicts. This ensures that no trade-off is done between en-route and airport conflicts in favor of en-route conflicts. In contrast, the weighting factor even supports solving further airport-related conflicts with the trade-off of generating new en-route conflicts.

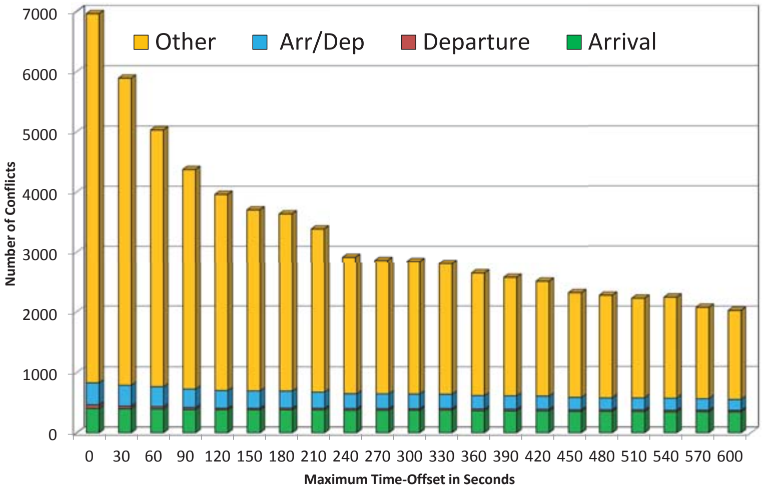


Figure 5.16 – Global Reduction in Relation to Time-Shift

The progress of the global time shifting is shown in fig. 5.16. The total number of conflicts can be reduced drastically. The weighting of conflicts does not only preserve resolutions for airport related conflicts but even allows further reduction. The distribution of conflicts with a time shift of ±10 minutes is depicted in fig. 5.17. After applying the recursive algorithm with a maximum recursion depth of 20, the final distribution shown in fig. 5.18 is achieved. Thus, initially ∼29 k conflicts can be reduced to 1647 without decreasing aircraft's efficiency.

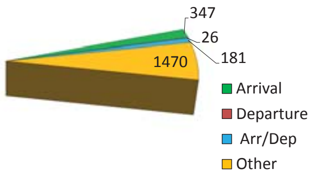


Figure 5.17 – Conflicts after Global Shift of ±10 minutes

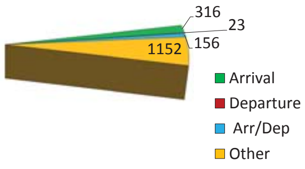


Figure 5.18 – Global Recursion Optimization Level 20

5.5.3 Resolution of En-Route Conflicts

En-route conflicts can be reduced further with lateral and vertical maneuvers. Due to the weighting of airport versus en-route conflicts this even further reduces the number arrival conflicts. The result after lateral and vertical resolution leaving less than 500 conflicts is shown in fig. 5.19.

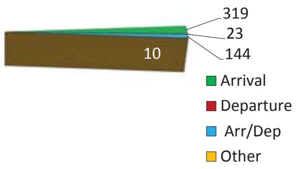


Figure 5.19 – Conflicts after Lateral and Vertical Resolution

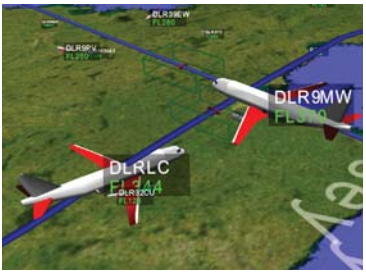


Figure 5.20 – Very Short En-route Conflict in Red

Obviously, not all conflicts are solvable with the algorithms applied. Figure 5.20 shows one example for a remaining en-route conflict. Flight DLRLC departed from Paris-Charles-de-Gaulle, is still in climb and heads for the United States. Flight DLR9MW departed from London-Stansted, already reached cruise FL370, and heads for Murcia-San Javier. Both flights have a conflict according to the selected conflict metrics for as short as 0.7 seconds. The closest point of approach is reached when DLR9MW passed the path intersection

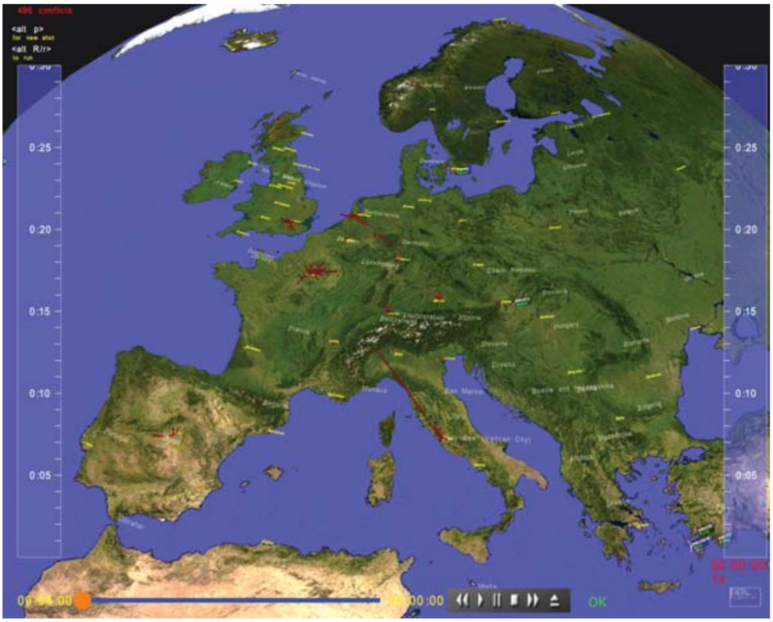


Figure 5.21 – Remaining Conflicts

All remaining conflicts are depicted in fig. 5.21. Conflicts are concentrating on major European airports like Paris-Charles-de-Gaulle, Amsterdam Schiphol, London-Heathrow, Rome-Fiumicino, Frankfurt Main, Zurich, and Munich Franz Josef Strauss. Most of the mentioned airports have a parallel or near-parallel runway system allowing reduced separation for parallel approaches and departures. Thus, at least some of the remaining conflicts are no conflicts in real life thanks to special operations allowing closer separation.

Since this work focuses on conflict detection and resolution while the exact specification of a conflict is done by the user by the external conflict function, this adaptation is not handled here.

Furthermore, not all effort was put yet into solving conflicts. One parameter not adapted yet is the flight duration. Thus, trying to solve an arrival conflict by delaying an arrival by 10 seconds relies on the freedom to simultaneously delay the departure by also 10 seconds so far. Allowing to change the flight duration offers a good potential to solve the remaining conflicts. However, the aforementioned improvement on conflict specification should be done first in order to guarantee that a conflict-free solution exists in terms of airport load.

In order to get a conflict free scenario for the 4DCo-GC project, another conflict resolution method was applied. The remaining 496 conflicts were solved by flight cancellation. Although deletion of one aircraft from each conflict reduces the total number of aircraft to 32 573, it ensures a still big and conflict-free scenario.

Chapter 6

Verification and Validation

This chapter describes a verification and validation of the software module developed in this work. According to Boehm (1979), validation and verification formulate different questions concerning the quality of a software project:

- Validation: "Are we building the right product?"
- Verification: "Are we building the product right?"

Since the product generated in this work was not created according to explicitly defined product requirements it is nevertheless difficult to distinguish between verification and validation. The development of the *NDMap* started with the rough idea of subdividing airspace in all 4 dimensions in order to allow efficient conflict detection. While the basic concept remained, promising results with big scenarios led to an extensive usage in projects all formulating their own requirements. Verification (or according to Popper (1974) falsification) of the software is usually done on programming via testing software invariants.

In order to validate that the software is reasonable for its operational purpose, several use cases are checked below. Each use case contains the scenario description in a Keyhole Markup Language

(KML) like style[1], a screenshot showing the result, and a printout from the corresponding *NDMap* object. All scenarios are completely fictional and do not represent reality. The trajectories are artificial and do not necessarily take aircraft performance parameter into account. Conflict detection is done with the aforementioned metrics of 5 NM lateral and 1000 ft vertical.

6.1 Nominal Case

The scenario described in listing 6.1 defines one trajectory from West to East at 50° latitude and three additional trajectories crossing the first in different angles. Figure 6.1 depicts the resulting three conflicts. As expected, the rectangular intersection results in a short conflict, while the most pointed intersection has a much bigger extension.

Listing 6.2 shows the corresponding *NDMap* output.

Thus, the *NDMap* object uses 119.5 KiB of memory with a total number of 415 generated tiles. Average time to add a trajectory was 0.48 ms. Three conflicts were detected with duration from 13.9 s to 57.5 s.

6.2 Pseudo-Parallel Case

The scenario described in listing 6.3 defines two trajectories going exactly from South to North with a constant longitudinal distance of 0.1°. Both trajectories start at the Equator where a tenth of a degree approximates conflict-free 6 NM. Going further to the North, the corresponding distance to a tenth of degree decreases, and therefore results in a conflict beginning at the latitude where $\cos\varphi = 5/6$. Thus, the conflict should start at $\arccos(5/6) = 33.56° = 33°33.6'$ on a sphere. However, the implementation uses Vincenty's formula based on WGS84 ellipsoid where the Equatorial radius is slightly bigger compared to common sphere models (compare

[1]Note that altitudes are defined in meters in KML but converted to feet in the *NDMap*, latitudes and longitudes are converted from degree to radian

Listing 6.1 – Nominal Case Scenario

```
<Placemark id="EASTWEST"> <name>EASTWEST</name> <
    coordinates>
60,50,10000,10000
0,50,10000,16000
</coordinates> </Placemark>

<Placemark id="SOUTHNORTH"> <name>SOUTHNORTH</name> <
    coordinates>
10,0,10000,10000
10,60,10000,16000
</coordinates> </Placemark>

<Placemark id="DEG45"> <name>DEG45</name> <coordinates>
61,0,10000,9900
1,60,10000,15900
</coordinates> </Placemark>

<Placemark id="DEG10"> <name>DEG10</name> <coordinates>
28,45,10000,12100
-10,55,10000,18100
</coordinates> </Placemark>
```

Listing 6.2 – Nominal Output

```
TrajectoryMap containing 4 trajectories 0 polygons 4D
    incl. time,
Pen: 399 FlyBy: 243
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 119.5 KByte, Add: 0.4849243164 ms
 0:1, 1:2, 2:2, 3:4, 4:10, 5:11, 6:18, 7:56, 8:29,
     9:31, 10:65, 11:67, 12:65, 13:54, [=415];
 3 conflicts:
1 t[14993.1, 15007] between EASTWEST/Crs and SOUTHNORTH
    /Crs at (0.1745241984, 0.872673367, 32808.39895,
    15000.1)(Other)
2 t[14891.7, 14908.3] between DEG45/Crs and EASTWEST/
    Crs at (0.1920385843, 0.8726384604, 32808.39895,
    14899.7)(Other)
3 t[15071.3, 15128.8] between DEG10/Crs and EASTWEST/
    Crs at (0.1570368765, 0.8726689937, 32808.39895,
    15100.3)(Other)
```

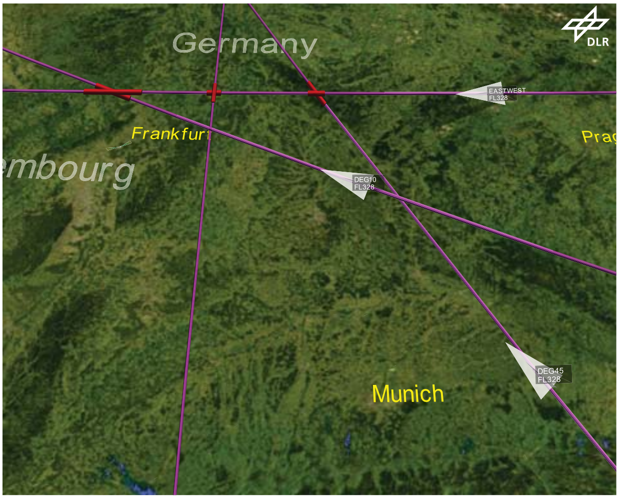


Figure 6.1 – Four Trajectories with Conflicts

Listing 6.3 – Pseudo-Parallel Scenario

```
<Placemark id="EAST"> <name>EAST</name> <coordinates>
10,0,10000,10000
10,60,10000,16000
</coordinates> </Placemark>

<Placemark id="WEST"> <name>WEST</name> <coordinates>
9.9,0,10000,10000
9.9,60,10000,16000
</coordinates> </Placemark>
```

Listing 6.4 – Pseudo-Parallel Output

```
TrajectoryMap containing 2 trajectories 0 polygons 4D
    incl. time,
Pen: 5014 FlyBy: 1661
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 1.1 MByte, Add: 7.225585938 ms
 0:1, 1:2, 2:1, 3:3, 4:5, 5:9, 6:18, 7:66, 8:64, 9:134,
     10:542, 11:459, 12:854, 13:1574, [=3732];
 1 conflicts:
1 t[13380.1, 16000] between EAST/Arr and WEST/Arr at
    (0.173660256, 1.047040501, 32808.39895, 15999.1)(
    Arrival)
```

section 2.6.1). Therefore, based on the more precise WGS84 model, the conflict starts at ∼33°50′ and remains active until reaching end of trajectories.

Figure 6.2 depicts the resulting conflict. Listing 6.4 holds the results from the corresponding *NDMap*.

As discussed earlier, nearly identical trajectories cause creation of many nodes, 3732 in total. Also the memory usage of 1.1 MiB and detection times of 7.2 ms are above typical average.

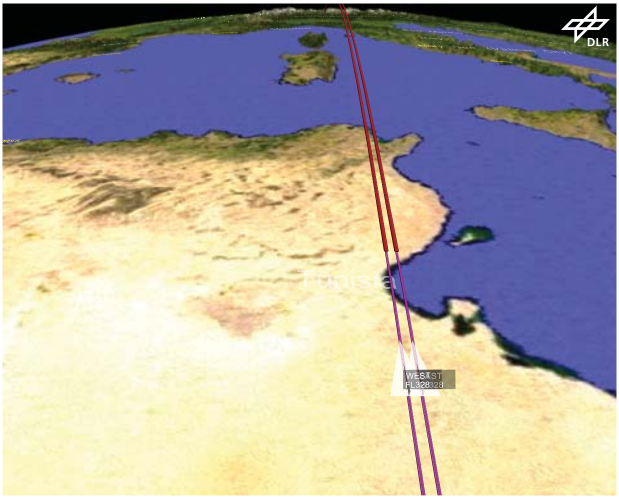


Figure 6.2 – Pseudo-Parallel Trajectories

6.3 Conflict Jitter Case

This use-case is supposed to validate the anti-jitter functionality described in section 3.5.3. Therefore, the scenario described in listing 6.5 defines two trajectories flying mostly parallel from West to East, one with a constant latitude of 50° while the other one varies between 49.9° and 49.95°. A tenth of degree latitude approximates conflict-free 6 NM, while a twentieth degree approximates to conflicting 3 NM. As defined by the input file, conflict distance is underrun three times. However, the first two conflicts are merged into one because the distance in-between is too short. The third conflict is too far away to get merged.

Figure 6.3 depicts the resulting conflicts. Listing 6.6 shows the results from the corresponding *NDMap*.

As experienced with the pseudo parallel case, conflict detection is rather slow and memory consuming due to nearly identical routes. The first conflict starting at $t = 10\,133$ s is the union of the first two separation violations.

Listing 6.5 – Jitter Scenario

```
<Placemark id="NORTH"> <name>NORTH</name> <coordinates>
0,50,10000,10000
60,50,10000,16000
</coordinates> </Placemark>

<Placemark id="SOUTH"> <name>SOUTH</name> <coordinates>
0,49.9,10000,10000
1,49.9,10000,10100
2,49.95,10000,10200
3,49.95,10000,10300
4,49.9,10000,10400
5,49.95,10000,10500
6,49.95,10000,10600
7,49.9,10000,10700
11,49.9,10000,11100
12,49.95,10000,11200
13,49.95,10000,11300
14,49.9,10000,11400
60,49.9,10000,16000
</coordinates> </Placemark>
```

Listing 6.6 – Jitter Output

```
TrajectoryMap containing 2 trajectories 0 polygons 4D
    incl. time,
Pen: 654 FlyBy: 434
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 184.0 KByte, Add: 1.068359375 ms
 0:1, 1:2, 2:1, 3:2, 4:3, 5:6, 6:13, 7:44, 8:84, 9:22,
     10:41, 11:66, 12:228, 13:126, [=639];
 2 conflicts:
1 t[10133.6, 10666.5] between NORTH/Crs and SOUTH/Crs
    at (0.05229006584, 0.8722282946, 32808.39895,
    10299.6)(Other)
2 t[11133.5, 11366.4] between NORTH/Crs and SOUTH/Crs
    at (0.2095267825, 0.8722282946, 32808.39895,
    11200.5)(Other)
```

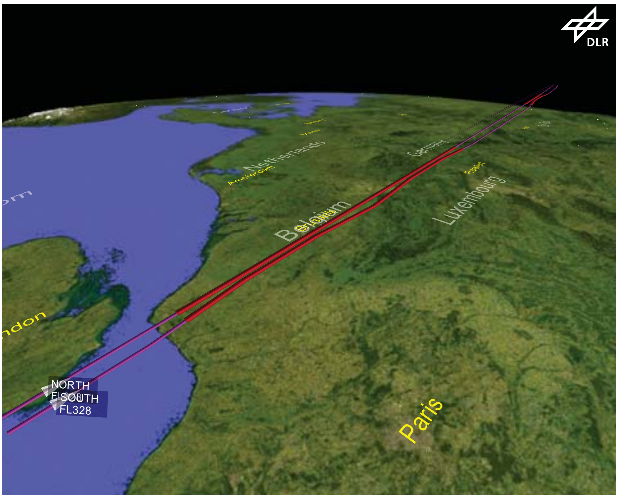


Figure 6.3 – Jitter in Conflict

Listing 6.7 – Pole Scenario

```
<Placemark id="POLE1"> <name>POLE1</name> <coordinates>
-90,50,10000,10000
-90,90,10000,16000
</coordinates> </Placemark>

<Placemark id="POLE2"> <name>POLE2</name> <coordinates>
90,50,10000,10000
90,90,10000,16000
</coordinates> </Placemark>
```

Listing 6.8 – Pole Output

```
TrajectoryMap containing 2 trajectories 0 polygons 4D
    incl. time,
Pen: 1380 FlyBy: 3317
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 803.2 KByte, Add: 4.845458984 ms
 0:1, 1:3, 2:4, 3:6, 4:8, 5:12, 6:52, 7:100, 8:99,
     9:204, 10:413, 11:609, 12:1006, 13:272, [=2789];
 1 conflicts:
1 t[15993.8, 16000] between POLE1/Arr and POLE2/Arr at
    (0, 1.570773099, 32808.39895, 15999.8)(Arrival)
```

6.4 Singularity Case

The scenario described in listing 6.7 defines two trajectories flying exactly from South to North, starting at a latitude of 50° and ending at the North Pole 90°. The trajectories are located on opposite sides of Earth at longitudes of −90° and 90°. Both trajectories reach North Pole in same altitudes at the same time.

Figure 6.4 depicts the resulting conflict above North Pole. The corresponding *NDMap* output is shown in listing 6.8.

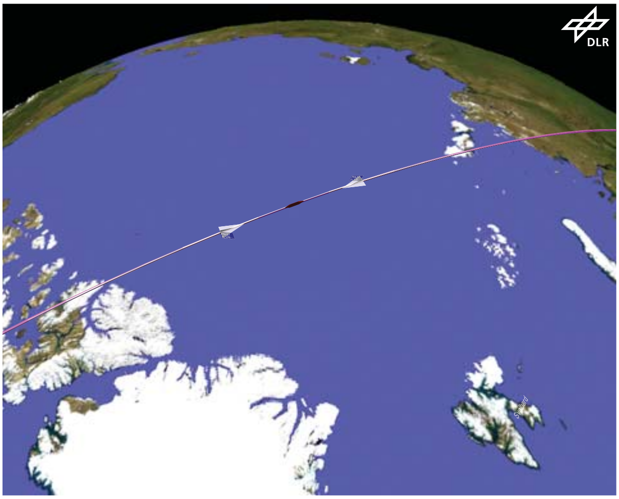


Figure 6.4 – Conflict at North Pole

As already explained in section 4.1.1 the optimal representation contains only one longitudinal tile at the Poles. Due to the tree structure each tile contains its children, and multiple tiles are generated close to the Poles. Thus, the *NDMap* generates a rather high total number of 2789 nodes. However, the conflict at the North Pole is detected reliably.

6.5 Discontinuity Case

The scenario described in listing 6.9 defines three trajectories. Two aircraft fly exactly from South to North at longitudes 179.98° East and West respectively. The third trajectory intersects the dateline at the Equator.

Figure 6.5 depicts the two resulting conflicts. The pseudo parallel routes are in conflict constantly. The East-West trajectory is in conflict with the flight at longitude 179.98° East. The *NDMap* output is shown in listing 6.10.

Listing 6.9 – Discontinuity Scenario

```
<Placemark id="WEST"> <name>WEST</name> <coordinates>
-179.995,-50,10000,10000
-179.995,50,10000,15950
</coordinates> </Placemark>

<Placemark id="EAST"> <name>EAST</name> <coordinates>
179.995,-50,10000,10000
179.995,50,10000,15950
</coordinates> </Placemark>

<Placemark id="EASTWEST"> <name>EASTWEST</name> <
    coordinates>
178,1,10000,12500
-178,-1,10000,13500
</coordinates> </Placemark>
```

Listing 6.10 – Discontinuity Output

```
TrajectoryMap containing 3 trajectories 0 polygons 4D
    incl. time,
Pen: 4041 FlyBy: 8300
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 2.3 MByte, Add: 5.306966146 ms
 0:1, 1:4, 2:6, 3:13, 4:17, 5:29, 6:55, 7:102, 8:102,
     9:392, 10:788, 11:1410, 12:2679, 13:2546, [=8144];
 2 conflicts:
1 t[10000, 15950] between EAST/Dep and WEST/Dep at (0,
    -0.8726646304, 32808.39895, 10000)(Departure)
2 t[12975.9, 12980.9] between EAST/Crs and EASTWEST/Crs
     at (3.140681647, 0.0009402631346, 32808.39895,
    12978.9)(Other)
```

Listing 6.11 – Internal Representation of EASTWEST Trajectory

```
Trajectory for EASTWEST(6 Points)
(3.106686115, 0.01745329238, 32808.39895, 12500)
(3.14158637, 3.141596834e-06, 32808.39895, 12999.91)
(3.14158637, 3.141596834e-06, 1e+30, 12999.911)
(-3.14158637, -3.141596834e-06, 1e+30, 13000.089)
(-3.14158637, -3.141596834e-06, 32808.39895, 13000.09)
(-3.106686115, -0.01745329238, 32808.39895, 13500)
```

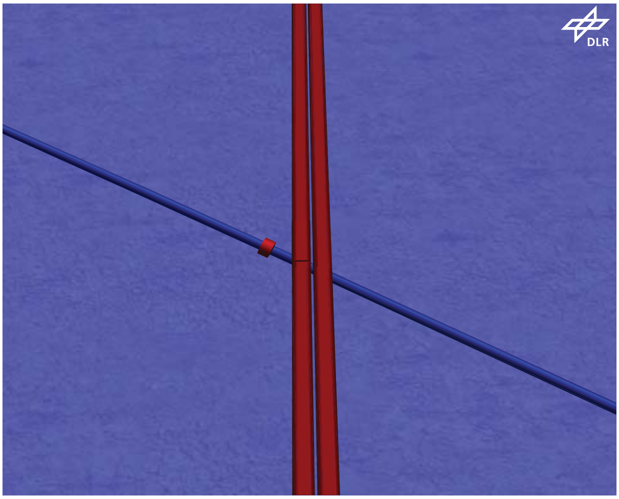


Figure 6.5 – Conflicts at Date Line

As experienced earlier, the *NDMap* generates a rather high amount of nodes due to parallelism of EAST and WEST. Listing 6.11 shows a dump of the internal EASTWEST trajectory. The first and last points are the original points of the trajectory, while points 2-5 are automatically generated. Point 2 and point 5 hold the interpolated positions just before and after passing the date-line. Point 3 and point 4 hold the connection between point 2 and point 5 not lying in the *NDMap*'s interval at an altitude of 10^{30} ft.

6.6 Polygon Volume Case

The scenario described in listing 6.12 defines three objects. Two flights are defined by trajectories. *DLR1* departs from Nuremberg with direction South-East. *DLR2* crosses the Southern part of Germany from West to East.

The third object called *Germany* is a polygon volume approximating the German borders. The scenario importer uses minimum and maximum altitudes for a volume object as lower and upper boundaries. Thus, the German polygon has a lower boundary of 8000 meters and goes up to 11 300 meters.

Figure 6.6 depicts the three resulting conflicts. The trajectories are not in conflict with each other. *DLR1* has a conflict between 8000 and 11 300 meters with the polygon volume. *DLR2* has two conflicts with the German polygon interrupted by the non-German part of Lake Constance. The *NDMap* output is shown in listing 6.13.

The *NDMap* generates a rather low amount of nodes (35), uses 10.1 KiB of memory only. The average detection time per object is 0.11 ms. This good performance is reached by means of the full containment technique described in section 3.3.5.

Adding the lines from listing 6.14 to the German polygon volume makes the volume move in time. The polygon volume appears at time zero (i. e., midnight). At time 36 000 s (i. e., ten o'clock) the polygon is supposed to be 5000 meters lower. At noon (43 200 s), the polygon is still 5000 meters below start condition. Thereafter, the polygon is no more valid and disappears. Between the given times, the polygon state is linearly interpolated. Thus, the whole polygon descends continuously between midnight and ten o'clock and stays there until noon.

This modification has two effects that are visible in the *NDMap* output (listing 6.15). The volume is already below *DLR2* when it enters the polygon's boundaries laterally. Therefore, there are no more conflicts with *DLR2*. The conflict with *DLR1* starts earlier because of the reduced altitude and has a shorter duration because the aircraft climbs while the volume descends.

Listing 6.12 – Polygon Volume Scenario

```
<Placemark id="DLR1">
  <name>DLR1</name>
  <coordinates>
  11.103056,49.493333,548.6,303.0
  12.603611,49.085278,9022.1,973.0
  13.360278,48.750833,11338.6,1257.0
  13.490833,48.659444,11704.3,1318.0
  13.606667,48.561944,11856.7,1379.0
  15.108611,47.255000,11887.2,2195.0
  </coordinates>
</Placemark>
<Placemark id="DLR2">
  <name>DLR2</name>
  <coordinates>
  5.544444,48.245278,11277.6,3957.0
  15.106944,46.899167,11277.6,6733.0
  </coordinates>
</Placemark>
<Placemark id="GERMANY">
  <name>GERMANY</name>
  <LinearRing>
  <coordinates>
6.499999, 54.999989,11300
7.999998, 54.999989,8000
8.333332, 55.066655
...
[many points skipped to increase readability]
...
6.499999, 53.666656
6.499999, 54.999989
  </coordinates>
  </LinearRing>
</Placemark>
```

Listing 6.13 – Polygon Volume Output

```
TrajectoryMap containing 2 trajectories 1 polygon 4D
    incl. time,
Pen: 52 FlyBy: 0
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 10.1 KByte, Add: 0.1146647135 ms
 0:1, 1:3, 2:4, 3:5, 4:6, 5:16, [=35];
 3 conflicts:
1 t[892.1825376, 1252.267736] between DLR1/Clb and
    GERMANY/Poly at (0.2168154297, 0.8575587731,
    26246.71916, 892.1825376)(Polygon)
2 t[4673.562491, 4806.546029] between DLR2/Crs and
    GERMANY/Poly at (0.1398495709, 0.8359744606,
    36999.99872, 4673.562491)(Polygon)
3 t[5140.595825, 5807.581754] between DLR2/Crs and
    GERMANY/Poly at (0.1679282864, 0.8320218268,
    36999.99872, 5140.595825)(Polygon)
```

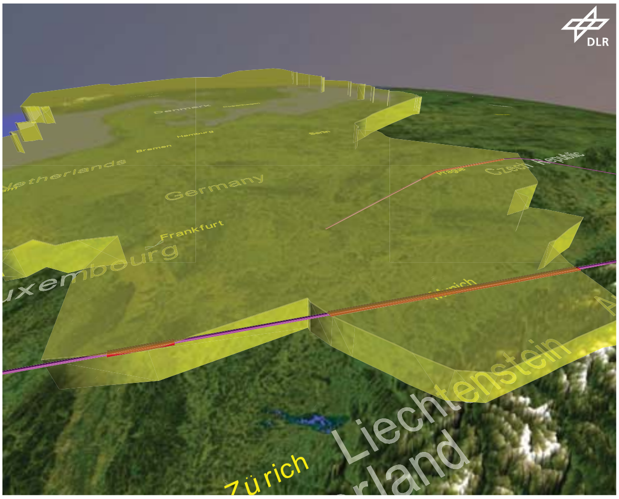


Figure 6.6 – Conflicts with Germany Volume

Listing 6.14 – Trajectory for German Polygon Volume

```
  <LineString>
  <coordinates>
0,0,0,0
0,0,-5000,36000
0,0,-5000,43200
  </coordinates>
  </LineString>
```

Listing 6.15 – Moving Polygon Volume Output

```
TrajectoryMap containing 2 trajectories 1 polygon 4D
    incl. time,
Pen: 46 FlyBy: 0
Sep: (0.001454441043, 0.001454441043, 1000, 90)
 8.6 KByte, Add: 2.865722656 ms
 0:1, 1:2, 2:2, 3:3, 4:6, 5:16, [=30];
 1 conflicts:
1 t[882.6, 1231.3] between DLR1/Dep and GERMANY/Poly at
     (0.2164564936, 0.857656381, 25865.7097, 883)(
    Polygon)
```

Chapter 7

Conclusions and Outlook

This document describes a new and high performance method for conflict detection between multi-dimensional objects. Allowing to model problems with an arbitrary number of dimensions, the proposed algorithm offers a wide range of applications. The basic idea of the algorithm is an N-dimensional bisection of (air)space until reaching a given separation size. In order to achieve a good performance regarding calculation times and memory usage, several techniques were applied:

- Special handling of a selectable common time/reference dimension reduces overall memory usage and increases performance.
- Instead of considering neighborhood of tiles, the algorithm distinguishes between fly-through and fly-by objects already when building the tree.
- The algorithm is optimized to detect symmetric situations once only. This especially applies for fly-by situations where one fly-by object is already omitted when building the tree.
- Large polygons are optimized by the full containment check in order to avoid unnecessary subdivisions.

- Conflict detection is sped up by checking bounding boxes first.
- Monotony of dimensions is used beneficially, for example, by applying binary search.
- Trials with different input value balancing techniques did not improve performance.
- If the user has a focus on a subset of objects, preselection functionality of objects is provided increasing conflict detection speed significantly.
- Running the algorithm on a system with low memory is facilitated by both static and dynamic memory saving mechanisms.

Conflict detection can be performed for N-dimensional trajectories and N-dimensional polygon volumes. Furthermore, a trajectory can be assigned to a polygon volume in order to model a volume movement. The result of the tiling algorithm is a list containing all detected conflicts.

Furthermore, a mapping is described from Earth-coordinates to the Cartesian coordinate system used in the tiling algorithm. A special Earth mode implemented in the *NDMap* respects further adaptations like non-uniform size of longitudes and avoidance of singularity issues at Poles and date line.

Based on a German and European traffic scenario, performance tests yield 1.6 ms detection time using 68 KiB per trajectory and 5 ms detection time using 120 KiB per trajectory respectively.

Based on these promising results, conflict resolution was performed in a trial-and-error manner with 151 probes a second on the European traffic sample. Since conflicts at the airports offer limited possibilities for resolution (i. e., usually only time-based), a focus was put onto solving airport related conflicts first.

Conflict solving was performed on an optimized European traffic sample flying most direct routes with aircraft-optimized flight profiles containing $\sim$29 000 conflicts initially. Allowing a maximum time shift of $\pm$10 minutes, conflicts were reduced to 1647 without degrading aircraft's efficiency. Applying lateral and vertical resolution algorithms on the remaining conflicts, overall conflict count can be reduced to 496.

Further freedom to solve conflicts lies in the adaptation of flight times that are still untouched. However, a better modeling of separation minima for arrival and departure traffic should be integrated first in order to respect, for example, relaxed separation requirements in independent parallel runway operations.

One main goal at implementation level was to keep the solution as generic as possible. This also applies for further enhancements. Thus, parallel runway and in-trail detection for wake-turbulence separation should be possible without an underlying static runway database. On the other hand, detection needs to be fast without a significant downgrade of current conflict detection times. In order to help the user with in-trail and independent parallel runway situations, corresponding generic information shall be predicted within the *NDMap* and provided to the external conflict function. This could be implemented, for example, by means of an internal 3-dimensional *NDMap* holding latitude, longitude and bearing of departure and arrival positions. As soon as a new departure or destination does not collide with an entry in the *NDMap* a new runway is generated.

Another more sophisticated option in terms of realism leaves the generic approach and uses real world airport/runway information. Integration of airport specific data should not be implemented in the *NDMap* though, but should be provided by the external conflict function. One method performing this in an automated way is described by Geister (2012).

Expectations from the more accurate modeling close to airports are an improved calculation of conflicts. Using real wake-turbulence separation instead of 3 NM below 5000 ft increases the number of conflicts. Since each pair of flights is forced already in the current version to have 5 NM separation until diving below 5000 ft and remaining route length to catch up is usually short, effects are limited though. Using wake-turbulence separation instead of 5 NM standard above 5000 ft for in-trail flights in the TRACON typically decreases number of conflicts.

Since many of the remaining conflicts are close to airports with independent parallel runways, the number of remaining conflicts is expected to decrease significantly with the implementation of independent parallel runway separations.

Concerning the algorithm's performance, improved data structures may be beneficial for storing relevant information. Especially for big scenarios, the vector data type holding the conflicts seems to be a bottleneck and decreases performance.

Another option to further increase performance is parallel processing. Especially in the last decade, single CPU performance increased moderately, only. In the same time period, the number of processing cores increased significantly. In order to get performance improvements from the increased number of cores, algorithms need to be adapted to allow distribution on multiple CPUs.

In order to allow parallel processing without blocking of semaphores, multiple instances should handle different, non-overlapping sub-spaces. One main instance could distribute trajectories or parts of trajectories on a number of sub-space instances. One major issue is the dynamic assignment of sub-spaces to process instances. Without analyzing the traffic scenario prior conflict detection, it is difficult to estimate a well balanced load distribution. Another challenge is holding all information consistent across multiple instances, especially when considering that the processing order may be different from the input data sequence.

An easier to implement approach for distributing the algorithm to multiple CPUs uses smaller sub-tasks, e. g., testing if an object penetrates a given tile. Thus, multiple tiles can be checked at the same time. When a tile necessitates subdivision, the 2^N children could be checked in parallel. However, tasks should have comparable calculation effort, the management overhead gets more significant with smaller tasks, and the high frequent synchronization may further reduce expected benefits.

Especially the results from shrinking the root tile in section 4.5.6 illustrate that the proposed balancing algorithms still do not reach the optimum. Although the issue is complex, further effort might result in a beneficial input balancing.

Further research can also be spent on conflict resolution algorithms. For example, a time-based conflict resolution adapting the speeds is not integrated yet, but is beneficial especially for a more tactical conflict resolution. However, current results illustrate that improvement of separation definitions in the vicinity of airports should be implemented first in order to concentrate on real conflicts only.

Several past and ongoing projects at DLR already have proven that this work is the technical enabler for conflict detection and resolution on huge traffic scenarios. The high performance and flexibility concerning support of different object types opens a new field of applications. Thus, the *NDMap* will help to make future air traffic more predictable, efficient, and environmental friendly.

Chapter 8

Update after Disputation

Since the disputation took place ∼18 months after submission of this thesis, several work proposed in the outlook already has been performed when passing to press. Because most of the new features confirm the high value of the presented algorithms, this chapter gives a brief summary on what has been achieved since submission.

8.1 Conflict Metric

The conflict metric has been extended to cover parallel runway configurations for approach mode as defined in section 2.4. Furthermore, the dynamic wake turbulence separation has been implemented. As a result, it was possible with the presented conflict resolution algorithms to get the European scenario free of conflict.

Since resolution of the European scenario worked very well, I furthermore extended the separation requirements from 5 NM to 6.5 NM, 8 NM, 10 NM, and even 12 NM. The idea was to use the well known 5 NM as safety separation, and leave the remaining space for allowed deviations from the trajectory. These allowed deviations may be used pro-actively by the aircraft to optimize the trajectory furthermore. They also give freedom when deviations occur, e. g., due to a bad weather forecast.

Even with the increased separation requirements, generation of a conflict free traffic scenario was always possible. Results are summarized in table 8.1. The last three rows hold necessary adaptations to get the optimized scenario resolved:

- The average *Time Shift* describes the average absolute time shift each aircraft was shifted from its initial departure time;
- The average *Add. Climb* specifies how many extra feet each aircraft has to climb compared to its initial route on average;
- The average *Add. Dist* states how many extra distance each aircraft has to fly compared to its initial route on average.

More detailed information is available from Kuenz (2014).

Table 8.1 – Results from Trials with Increased Separation

	Scenario				
	5 NM	6.5 NM	8 NM	10 NM	12 NM
Separation	5 NM	6.5 NM	8 NM	10 NM	12 NM
Freedom	0 NM	0.75 NM	1.5 NM	2.5 NM	3.5 NM
Conflicts	24 162	28 328	33 115	40 562	48 942
En-Route	11 137	15 303	20 088	27 534	35 912
Airport	13 025	13 025	13 027	13 028	13 030
Time Shift	93.9 s	120.8 s	143.4 s	192.2 s	200.2 s
Add. Climb	16.9 ft	40.8 ft	59.8 ft	159.2 ft	220.0 ft
Add. Dist.	3.7 m	68 m	137 m	1.3 NM	3.5 NM

8.2 Performance

Improved data structures have been integrated in the algorithm especially increasing the conflict detection speed for large scenarios. New links from objects to their conflicts increase coordination effort

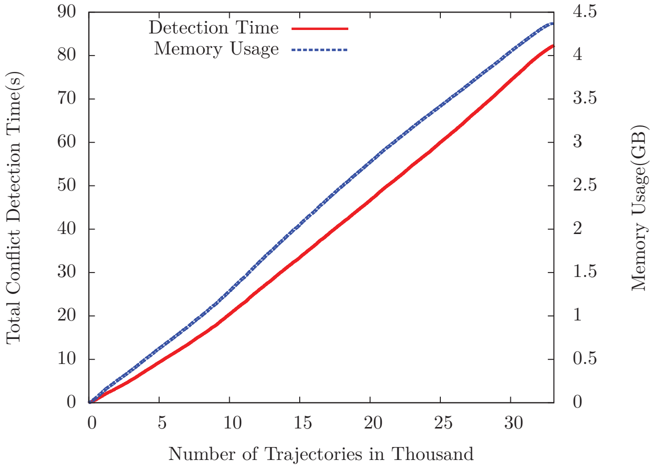


Figure 8.1 – Improved Detection Times with New Data Structures for Europe

slightly only, but increase significantly the speed to validate if the conflict already exists. Furthermore, hash maps were added to find objects much quicker than before. Figure 8.1 shows the increased speed of the *NDMap* with improved data structures. With about 82 s the algorithm halves the original detection time. Even for the smaller German traffic sample, the *NDMap* performs about 20 % better, compare fig. 8.2.

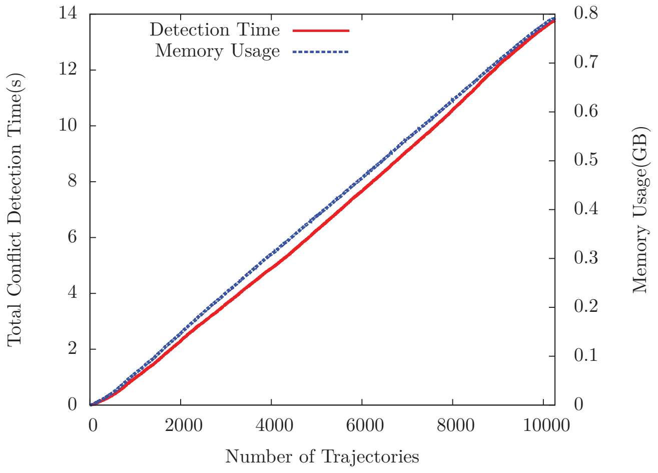


Figure 8.2 – Improved Detection Times with New Data Structures for Geramy

Bibliography

Abam, M., M. Berg, S.-H. Poon, and B. Speckmann (2006). Kinetic collision detection for convex fat objects. In Y. Azar and T. Erlebach (Eds.), *Algorithms - ESA 2006*, Volume 4168, pp. 4–15. Springer Berlin Heidelberg.

Adam, V. and R. Kohrs (1992). On board planning of 4D-trajectories. *AGARD CP 504*, 16.1–16.12.

Basch, J., J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang (1999). Kinetic collision detection between two simple polygons. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, Philadelphia, PA, USA, pp. 102–111. Society for Industrial and Applied Mathematics.

Beckmann, N., H. P. Kriegel, R. Schneider, and B. Seeger (1990). The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data - SIGMOD '90*, pp. 322.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM 18(9)*, 509–517.

Bentley, J. L. and T. A. Ottmann (1979). Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers C-28*(9), 643–647.

Birkmeier, B., B. Korn, and D. Kügler (2010, October). Sectorless ATM and advanced SESAR concepts: Complement not contradiction. In *29th Digital Avionics Systems Conference.*

Bode, S. and P. Hecker (2013, August). Efficient 4D trajectory conflict detection for large scale atm simulations using bounding-volume hierarchies and time-spatial indexing. In *Proceedings of the AIAA Modeling and Simulation Technologies (MST) Conference.*

Boehm, B. (1979). Guidelines for verifying and validating software requirements and design specifications. In *Euro IFIP 79.*

Boguhn, O. (2007). Internal Project Leiser Flugverkehr II. Technical report, Institute of Aerodynamics and Flow Technology, DLR Göttingen.

Central Intelligence Agency (2007). *The CIA World Factbook.* United States.

Chazelle, B. (1988). A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput. 17*, 427–462.

Chazelle, B. (1991). Triangulating a simple polygon in linear time. *Discrete & Computational Geometry 6*, 485–524.

Cohen, J. D., M. C. Lin, D. Manocha, and M. Ponamgi (1995). I-collide: An interactive and exact collision detection system for large-scale environments. In *In Proc. of ACM Interactive 3D Graphics Conference*, pp. 189–196.

Coming, D. S. and O. G. Staadt (2005). Kinetic sweep and prune for collision detection. In F. Ganovelli and C. Mendoza (Eds.), *Workshop On Virtual Reality Interaction and Physical Simulation.*

Czerlitzki, B. (1994). The experimental flight management system: Advanced functionality to comply with atc constraints. *Air Traffic Control Quarterly Vol. 2(3)*, 159–188.

Czerlitzki, B. and R. Kohrs (1994). 4D Flight Management - Planungsfunktionen zur Einhaltung von Zeitvorgaben. *ZFW - Zeitschrift für Flugwissenschaften und Weltraumforschung 18, Heft 1*, 40–47.

de Muynck, R., T. Bos, A. Kuenz, C. Edinger, L. Rappich, and S. Törner (2011). Real time ATC simulation of time based CDA operations. In *CEAS 2011*.

Deutsches Institut für Normierung (1999, Oct). DIN ISO 9613-2: Dämpfung des Schalls bei der Ausbreitung im Freien.

DFS Deutsche Flugsicherung (2011). Luftverkehr in Deutschland - Mobilitätsbericht 2011.

Dobkin, D. P. and D. G. Kirkpatrick (1983). Fast detection of polyhedral intersection. *Theroretical Computer Science 27*, 241–253.

Douglas, D. and T. Peucker (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer 10(2)*, 112–122.

Dougui, N., D. Delahaye, S. Puechmorel, and M. Mongeau (2010). A new method for generating optimal conflict free 4D trajectory. *Proc 4th International Conference on Research in Air Transportation 4*, 185–191.

Duong, V., E. Hoffman, L. Floc'hic, J.-P. Nicolaon, and A. Bossu (1996). Extended flight rules (EFR) to apply to resolution of encounters in autonomous airborne separation. `http://www.asas-tn.org/library/asassworksdonesbysrdsbod/freer/freer/rules-02.pdf` (March 2013), Boston.

Durand, N., J.-M. Alliot, and J. Noailles (1996). Automatic aircraft conflict resolution using genetic algorithms. In *Proceedings of the 1996 ACM symposium on Applied Computing*, SAC '96, New York, NY, USA, pp. 289–298. ACM.

Dworkin, P. and D. Zeltzer (1993). A new model for efficient dynamic simulation. In *Proceedings of the Eurographics Workshop on Animation and Simulation*, pp. 135–147.

Edelsbrunner, H. and H. A. Maurer (1981). On the intersection of orthongonal objects. *Information Processing Letters 13*, 177–181.

Ericson, C. (2005). *Real Time Conflict Detection.* Morgan Kaufmann.

Erzberger, H., T. A. Lauderdale, and Y.-C. Chu (2010). Automated conflict resolution, arrival management and weather avoidance for atm. In *27th Congress of the Internation Council of the Aeronautical Sciences*, Volume 27.

Erzberger, H., R. A. Paielli, D. R. Isaacson, and M. M. Eshow (1997). Conflict detection and resolution in the presence of prediction error. In *Proc. 1st USA/Eur. Air Traffic Manage. Res. Development Seminar*, pp. 50–56.

Eurocontrol (2010). DDR reference manual. `http://www.eurocontrol.int/services/demand-data-repository-ddr` (Jan. 2013).

Eurocontrol (2011). Base of aircraft data (BADA) version 3.9. `http://www.eurocontrol.int/eec/public/standard_page/proj_BADA.html` (Jan. 2013).

Eurocontrol (2012). Airborne collision avoidance system ii. `http://www.eurocontrol.int/dossiers/acas-ii` (Jan. 2013).

Federal Aviation Administration (2012). Implementation plan. `http://www.faa.gov/nextgen/media/ng2011_implementation_plan.pdf` (Jan. 2013).

Federal Aviation Administration (2013). Next generation air transportation system. `http://www.faa.gov/nextgen` (Jan. 2013).

Foudil, C., D. Noureddine, C. Sanza, and Y. Duthen (2009). Path finding and collision avoidance in crowd simulation. In *Journal of Computing and Information Technology.*

Fuchs, H., Z. M. Kedem, and B. F. Naylor (1980). On visible surface generation by a priori tree structures. In *SIGGRAPH '80 Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pp. 124–133.

Fuller, B. (1943, March). *Life Presents R. Buckminster Fuller's Dymaxion World*, pp. 41–55. Henry Luce.

Gargantini, I. (1982). Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing 20*, 365–374.

Geister, D. (2012). *Constraint Generierung für domänenspezifische Modellierungssprachen.* Ph. D. thesis, Leibniz Universität Hannover.

Golas, A., R. Narain, S. Curtis, and M. C. Lin (2013). Hybrid long-range collision avoidance for crowd simulation. *IEEE Transactions on Visualization and Computer Graphics 1*, 29–36. Early Access.

Guibas, L. J. (2001). Kinetic data structures. In Chapman and Hall/CRC (Eds.), *Handbook of Data Structures and Applications*, pp. 23–1–23–18. Mehta, Dinesh P.; Sahni, Sartaj.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, Volume 1984, pp. 47–57.

Hart, P. E., N. J. Nilsson, and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *4*(2), 100–107.

Hildum, D. W. and S. F. Smith (2004). Scheduling safe movement of air traffic in crowded air spaces. *The Knowledge Engineering Review 27*, 309–331.

Hopcroft, J. E., J. Schwartz, and M. Sharir (1983). Efficient detection of intersections among spheres. *Internat. J. Robot Res. 2*, 77–80.

Houthuys, P. (1987). Box sort, a multidimensional binary sorting method for rectangular boxes, used for quick range searching. *The Visual Computer 3*, 236–249.

ICAO (2002). Annex 10 to the convention on international civil aviation: Aeronautical telecommunications. Technical report, International Civil Aviation Organization.

ICAO (2004a). Annex 14 to the convention on international civil aviation: Aerodromes. Technical report, International Civil Aviation Organization.

ICAO (2004b). Doc 9643: Manual on simultaneous operations on parallel or near-parallel instrument runways (SOIR). Technical report, International Civil Aviation Organization.

ICAO (2005). Doc 9854: Global air traffic management operational concept. Technical report, International Civil Aviation Organization.

ICAO (2007). Doc 4444: Air traffic management. Technical report, International Civil Aviation Organization.

ICAO (2008). Guidance on A380-800 wake vortex aspects. Technical report, International Civil Aviation Organization.

Jardin, M. (2003). Real-time conflict-free trajectory optimization. In *5th USA/Europe ATM R&D Seminar*.

Jardin, M. (2005). Grid-based strategic air traffic conflict detection. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, California.

Jeck, R. K. (2002). Icing design envelopes (14 cfr parts 25 and 29, appendix c) converted to a distance-based format. Technical report, Federal Aviation Administration.

jen Chiang, Y., J. T. Klosowski, C. Lee, and J. S. B. Mitchell (1997). Geometric algorithms for conflict detection/resolution in air traffic management. In *Proceedings of the 36th IEEE Conference on Decision and Control*, Volume 2, pp. 1835–1840.

Joulia, A. and C. Le Tallec (2012). Aircraft 4D contract based operation: The 4DCo-GC project. In *28th Congress of the International Council of the Aeronautical Sciences*.

Kelly, M., K. Gould, B. Pease, S. Winner, and A. Yen (1994). Hardware accelerated rendering of CSG and transparency. In *SIGGRAPH'94 Proceedings of the 21st anual conference on Computer graphics and interactive techniques*, pp. 177–184.

Kelly, W. E. and M. S. Eby (2000). Advances in force field conflict resolution algorithms. In *AIAA Guidance, Navigation and Control Conference and Exhibit.*

Klosowski, J. T., M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *4*(1), 21–36.

Koeners, J. and M. de Vries (2008). Conflict resolution support for air traffic control based on solution spaces: Design and implementation. In *Proc. IEEE/AIAA 27th Digital Avionics Systems Conf. DASC 2008.*

Kohrs, R. (1992). Planung von 4D-Trajektorien an Bord eines Flugzeugs. *DGLR Jahrbuch 2*, 943–952.

Korn, B., C. Edinger, S. Tittel, D. Kügler, T. Putz, O. Hassa, and B. Mohrhard (2009). Sectorless ATM — a concept to increase en-route efficiency. In *Proc. IEEE/AIAA 28th Digital Avionics Systems Conf. DASC 2009.*

Korn, B. and A. Kuenz (2006, Oktober). 4D FMS for increasing efficiency of TMA operations. In *Proc. IEEE/AIAA 25th Digital Avionics Systems Conf. DASC 2006.*

Kremer, H., W. Vertegaal, and R. Jansen (1999). Phare advanced tools conflict probe - final report. Technical report, Programme for Harmonised ATM Research in EUROCONTROL.

Kuenz, A. (2011). A global airspace model for 4D-trajectory-based operations. In *Proc. IEEE/AIAA 30th Digital Avionics Systems Conf. DASC 2011.*

Kuenz, A. (2012, Juni). Optimizing tomorrows ATM using 4D-trajectory-based operations. In *ODAS 2012.*

Kuenz, A. (2014). Increasing the margins - more freedom in trajectory-based operations. In *Proc. IEEE/AIAA 33rd Digital Avionics Systems Conf. DASC 2014.*

Kuenz, A., H. Becker, C. Edinger, and B. Korn (2008, September). Performance-based TMA handling for mixed traffic using a ground based 4D-guidance for unequipped aircraft. In 26th ICAS Congress (Ed.), *26th Congress of the International Council of the Aeronautical Sciences*, Number 26 in ICAS.

Kuenz, A. and C. Edinger (2010a, September). Future air ground integration: A scalable concept to start with green approaches today. In *27th Congress of the Internation Council of the Aeronautical Sciences*.

Kuenz, A. and C. Edinger (2010b, Oktober). Green approaches without trade-off: Final results from the FAGI-project. In *Proc. IEEE/AIAA 29th Digital Avionics Systems Conf. DASC 2010*.

Kuenz, A., V. Mollwitz, and B. Korn (2007, Oktober). Green trajectories in high traffic TMAs. In *Proc. IEEE/AIAA 26th Digital Avionics Systems Conf. DASC 2007*.

Kuenz, A. and N. Peinecke (2009, Oktober). Tiling the world - efficient 4D conflict detection for large scale scenarios. In *Proc. IEEE/AIAA 28th Digital Avionics Systems Conf. DASC 2009*, Volume 28 of *DASC Conference*.

Kuenz, A. and N. Peinecke (2011, February). Effiziente 4D-Konflikt-Erkennung für großräumige Szenarien, Verfahren zur Ermittlung einer potenziellen Konfliktsituation. Patent (EP 2 457 224 A2).

Kuenz, A. and N. Peinecke (2012, June). Method for determining a potential conflict situation. Patent (US20120158278).

Kuenz, A. and G. Schwoch (2012). Global time-based conflict solution towards the overall optimium. In *Proc. IEEE/AIAA 31st Digital Avionics Systems Conf. DASC 2012*.

Kuenz, A., G. Schwoch, and F.-E. Wolter (2013). Individualism in global airspace - user-preferred trajectories in future ATM. In *Proc. IEEE/AIAA 32nd Digital Avionics Systems Conf. DASC 2013*.

Lewis, G. N., N. J. Boynton, and F. W. Burton (1981). Expected complexity of fast search with uniformly distributed data. *Inform. Proc. Let. 13*, 4–7.

Manolopoulos, Y., A. Nanopoulos, and Y. Theodoridis (2006). R-trees: Theory and applications. *Springer. ISBN 978-1-85233-977-7 1*, 15–34.

Mount, D. M. (1997). Geometric intersection. In J. E. Goodman and J. O'Rourke (Eds.), *Handbook of Discrete and Computational Geometry* (2 ed.)., Chapter 38, pp. 857–876. CRC Press LLC, Boca Raton, FL.

Naylor, B. (1993, May). Constructing good partitioning trees. In *Graphics Interface (annual Canadian CG conference).*

Osborne, P. (2008). The mercator projections. Technical report, University of Edingburgh, Edinburgh.

Popper, K. (1974). *Autobiography: The philosophy of Karl Popper*, Volume Abschnitt 8. P. A. Schilpp.

Prautzsch, H., W. Boehm, and M. Paluszny (2002). *Bezier- and B-spline techniques.* Springer.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing 1*, 244–256.

Reichling, M. (1988). On the detection of a common intersection of k convex objects in the plane. *Informa. Process. Lett. 29*, 25–29.

Rosenberg, J. (1985). Geographical data structures compared: A study of data structures supporting region queries. *IEEE Transaction on CAD Integrated Circuits Systems 4(1)*, 53–67.

Roussos, G. P., G. Chaloulos, K. J. Kyriakopoulos, and J. Lygeros (2008). Control of multiple non-holonomic air vehicles under wind uncertainty using model predictive control and decentralized navigation functions. In *Proc. 47th IEEE Conf. Decision and Control CDC 2008*, pp. 1225–1230.

Samet, H. (1990). *The Design and Analysis of Spatial Data Structures.* Addison-Wesley.

Schlager, H., H. Mannstein, B. Weinzierl, F. Linke, and T. Mühlhausen (2012). Projektplan volcanic ash impact on the air transport system. Project plan.

Schumacker, R. A., B. Brand, M. G. Gilliland, and W. H. Sharp (1969). Study for applying computer-generated images to visual simulation. Technical report, U.S. Air Force Human Resources Laboratory.

Schwoch, G. (2008, June). Konflikterkennung und -lösung in zukünftigen Luftfahrtszenarien basierend auf 4D-Flugbahnen. Technical report, Technische Universität Braunschweig.

SESAR Consortium (2007). The ATM target concept D3. Technical report, SESAR Consortium.

SESAR Consortium (2008). Sesar master plan D5. Technical report, SESAR Consortium.

SESAR Consortium (2010). Sesar brochure: Modernising the european sky. Technical report, SESAR Consortium.

SESAR Joint Undertaking (2013). Sesar. `http://www.sesarju.eu` (Jan. 2013).

Shamos, M. I. and D. Hoey (1976). Geometric intersection problems. In *Proc. th Annual Symp. Foundations of Computer Science*, pp. 208–215.

Smedt, D. D. and G. Berz (2007). Study of the required time of arrival function of current FMS in an ATM context. In *Proc. IEEE/AIAA 26th Digital Avionics Systems Conf. DASC 2007*.

Smith, S. F. (2008). Constraint-based techniques for managing movement in crowded airspaces. In *7th INO Workshop*.

Snyder, J. P. (1987). Map projections: A working manual. *USGS Professional Paper 1395*, 243–248.

Stell, L. (2010). Analysis of flight management system predictions of idle-thrust descents. In *Proc. IEEE/AIAA 29th Digital Avionics Systems Conf. DASC 2010*.

Tavares, D. L. M. and J. L. D. Comba (2007). Broad-phase collision detection using delaunay triangulation. Technical report, Universidade Federal do Rio Graqnde do Sul (UFRGS).

Teschner, M., B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross (2003). Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of the Vision, Modeling, and Visualization Conference*, pp. 47–54.

Teschner, M., S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino (2005). Collision detection for deformable objects. In *Computer Graphics Forum*, Volume 24, pp. 61–81.

The 4DCo-GC Consortium (2013). 4DCo-GC project. `http://www.4dcogc-project.org/` (Jul. 2013).

The CGAL Project (2015). *CGAL User and Reference Manual* (4.6.1 ed.). CGAL Editorial Board.

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. In *Survey Review*, Volume XXIII.

Voggenreiter, H. and W. Etzenbach (2012). DLR@UNI - Die institutionalisierte Zusammenarbeit zwischen DLR und Universitäten. 9-22.

Vujasinovic, R. (2012a). ATM and volcanic ash: An advanced approach to adverse event. In *International Conference on Research in Air Transportation - ICRAT*, Volume 5.

Vujasinovic, R. (2012b). Volcanic ash events: When the role of decision maker is assigned to a pilot. In *Proc. of Integrated Communications Navigation and Suerveillance (ICNS) Conference*.

Wald, I. and V. Havran (2006). On building fast kd-trees for ray tracing, and on doing that in O(NlogN). In *In Proceedings of the 2006 IEEE Symposium on interactive ray tracing*, pp. 61–70.

Woo, M., J. Neider, T. Davis, and D. Shreiner (1999). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2* (3rd ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Zachmann, G. (1998). Rapid collision detection by dynamically aligned dop-trees. In *Proceedings of Virtual Reality Annual International Symposium*, Atlanta, GA, USA, pp. 90–97.

Zachmann, G. and R. Weller (2006). Kinetic bounding volume hierarchies for deformable objects.

Index

Lebenslauf

Name	Dipl.-Inform. Alexander Kuenz
Anschrift	In den Grashöfen 50 38110 Braunschweig Telefon Beruf: (0531) 295 3008 Alexander.Kuenz@dlr.de

Geburtsdatum und -ort

08. 09. 1974 in Bad Harzburg

Familienstand

verheiratet, ein Sohn (⋆ 2008) und eine Tochter (⋆ 2014)

Ausbildung

8/1981-6/1994	Schulausbildung, Abitur am Werner-von-Siemens Gymnasium Bad Harzburg
9/1994-3/1997	Ausbildung zum Mathematisch-Technischen Assistenten (MaTA) beim DLR Braunschweig
3/1997-10/1997	Arbeit als MaTA beim DLR Braunschweig
10/1997-10/2001	Informatikstudium an der Technischen Universität Braunschweig mit der Gesamtnote sehr gut

Studienschwerpunkte: Theoretische Informatik, Rechnerstrukturen, Telematik, Datenbanken, Signalverarbeitung

Studienarbeit: Entwicklung einer Client/ Server-Applikation für eine Navigationsdatenbasis

Diplomarbeit: An Architectural Model for Heterogeneous Hardware/Software Platforms

Berufliche Erfahrung

10/1997-10/2001	Studienbegleitende Arbeit am Institut für Flugführung, DLR Braunschweig, Abteilung Anthropotechnik und Simulation
seit 11/2001	Wissenschaftlicher Mitarbeiter in der Abteilung für Pilotenassistenz
2001-2004	Mitarbeit an den Projekten ASSIST, ATTAS Upgrade und Wasla Hale
2004-2007	Arbeitspaketleitung im DLR-internen Projekt Leiser Flugverkehr II
2005-2008	Arbeitspaketleitung im BMBF-Projekt Leise An- und Abflüge (LAnAb) im Forschungsverbund Leiser Verkehr
2007-2009	Projektleitung des DLR-internen Projektes Future Air Ground Integration (FAGI)
2007-2009	Verantwortung für die Arbeitsanteile des Instituts für Flugführung am DLR-internen Projekt Future Aeronautical Communications Traffic Simulator (FACTS)
2007-2011	Projektleiter des DLR-Anteils im FP6-Projekt Environmentally Responsible Air Transport (ERAT)
2007-2009	Mitarbeit im europäischen Trajectory Prediction Gremium
seit 2008	Dozent bei CCG-Seminaren
2010-2013	Projektleiter des DLR-Anteils und Leiter des Arbeitspakets Integration im FP7-Projekt 4 Dimensional-Contracts - Guidance and Control (4DCo-GC)

seit 2011	Projektleiter für die Arbeitsanteile des Instituts für Flugführung im HGF-Projekt Supercooled Large Droplets Icing (SuLaDI)

Sprachkenntnisse

Englisch fließend in Wort und Schrift

Grundkenntnisse in Französisch, Spanisch und Italienisch

Fortbildungen

Scientific Skills z.B.	Noise Abatement Procedures Workshop GBAS-Workshop WakeNet Workshop Gate-to-Gate Open Days Courage Trajectory Prediction Workshop Genspace-Workshop (Lotsentraining) BZF und AZF Funksprechzeugnisse Teilnahme an und Beiträge zu diversen Konferenzen
Soft Skills z.B.	Moderation von Gesprächsrunden Verständliche und überzeugende Gesprächsführung mit Kollegen und Kunden Intensivkurs Projektmanagement

Braunschweig, 21. September 2015