Bachelor's Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Behavior Model of Flight Based on Visual Flight Rules

Jan Tóth

Supervisor: Mgr. Přemysl Volf, PhD.
Study program: Open Informatics
Branch of study: Computer and Information Science
May 2018

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tóth Jan** Personal ID number: **457116**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Behavior Model of Flight Based on Visual Flight Rules

Bachelor's thesis title in Czech:

Model chování pro let řízený vizuálními pravidly

Guidelines:

1. Study the problem of flying under visual flight rules.
2. Study the problem of map representation and its use for visual flight.
3. Study the problem of pilot behavior related to detection of other aircraft.
4. Design environment data representation for visual flight.
5. Design pilot behavior for visual flight.
6. Implement designed methods into the AgentFly system.
7. Validate experimentally designed model under various conditions.

Bibliography / sources:

[1] Řízení letového provozu ČR, Pravidla pro lety za viditelnosti - https://lis.rlp.cz/ais_data/aip/data/valid/e1-2.pdf
[2] Colvin, Kurt and Dodhia, Rahul and Dismukes, R Key: Is Pilots' Visual Scanning Adequate to Avoid Mid-Air Collisions?, Citeseer, 2005
[3] Orrell, Gregory L. : Effects of Primary TIS-B on the General Aviation Pilot: A Human-in-the-Loop Simulation, ICRAT, Berkley, 2012
[4] Olbricht, Roland M: OpenStreetMap in GIScience - Data Retrieval for Small Spatial Regions in OpenStreetMap, pages 101-122, Springer 2015
[5] https://wiki.openstreetmap.org/

Name and workplace of bachelor's thesis supervisor:

Mgr. Přemysl Volf, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Mgr. Přemysl Volf, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to express my gratitude to my supervisor, Mgr. Přemysl Volf, PhD., and also to his associate Ing. Lukáš Koranda for their valuable advice and guidance. This work would not be possible without their collaboration.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date .....

.....
signature

Abstract

The aim of this thesis is to design a behavior model of a flight based on Visual Flight Rules. The proposed model is to be implemented in multi-agent simulation system AgentFly to augment its current capabilities of simulating air traffic. This work primarily focuses on problems of a suitable map representation for flights operating under Visual Flight Rules, accurate visual detection of possible conflicts and their resolution in accordance with Visual Flight Rules. The map representation proposed by this work is created by the combination of data from height maps and OpenStreetMap database. Both detection and resolution of possible conflicts are based on observing the GPS coordinates of the other aircraft. The implementation created as a part of this thesis allows to thoroughly test the safety and reliability of flying under Visual Flight Rules in a multitude of situations.

Keywords: visual flight rules, multi-agent simulation, map representation, conflict detection, conflict resolution, safety testing

Supervisor: Mgr. Přemysl Volf, PhD.

Abstrakt

Cílem této práce je navrhnout model chování letu řízeného vizuálními pravidly. Navržený model má být implementován v multi-agentním simulačním systému AgentFly za účelem rozšíření jeho současných schopností simulovat leteckou dopravu. Tato práce se převážně zabývá problémy reprezentace mapy vhodné pro lety řízené vizuálními pravidly, přesnou vizuální detekcí možných kolizních hazardů a jejich řešením v souladu v vizuálními pravidly. Reprezentace mapy navržena touto prací je vytvořena spojením dat z výškových map a databáze OpenStreetMap. Detekce i řešení možných kolizních hazardů jsou založeny na pozorování GPS souřadnic druhého letadla. Implemetance vytvořena v rámci této práce umožnuje důkladně testovat bezpečnost a spolehlivost létání podle vizuálních pravidel v mnoha situacích.

Klíčová slova: pravidla pro let za viditelnosti, multi-agentní simulace, reprezentace mapy, detekce konfliktu, řešení konfliktu, testování bezpečnosti

Překlad názvu: Model chování pro let řízený vizuálními pravidly

Contents

Figures

Tables

Chapter 1

Introduction

Air traffic is the youngest means of transport. It is only over a century old [1]. At first, the number of planes was small, and there was no need for any specialized air traffic coordination. The first significant development of the air traffic began after the *World War I*. The development was soon followed by the first rules for air traffic (General Rules for Air Traffic) that were presented by International Commission for Air Navigation founded in 1919. Back then, simple and straightforward signals such as waving flags were used to coordinate air traffic from the ground. Pilots did not have any specialized instruments to navigate in the air, and they had to rely primarily on their vision. Nowadays, such flying is called a flight based on Visual Flight Rules (abbr. VFR) or a VFR flight or simply a VFR.

Over the course of modern history, air traffic has continuously grown. As the number of aircraft was increasing and newer airplanes with more advanced capabilities were being constructed (e.g., jet planes), new rules and technologies had to be introduced to ensure safety and efficiency of air traffic. At the turn of the 1950's and the 1960's, United States Federal Aviation Administration (abbr. FAA) and European Organization for the Safety of Air Navigation (abbr. EUROCONTROL) were established. Since then, they have become leading authorities in the creation of regulations regarding air traffic management (abbr. ATM) and safety [1].

Gradually, technologies such as radio, radar, and computer systems were adopted by the Air Traffic Services (abbr. ATS) that aim to ensure air traffic safety. ATS provides various individual services such as Air Traffic Control (abbr. ATC) or Flight Information Service (abbr. FIS) [2]. With the help of all these services, a new type of flying was introduced where pilots only rarely rely on their vision (which has proven to be somewhat unreliable [3]). In this type of flying, they turn to their technical equipment for an overview of their current situation, and they perform actions based on instructions from the local ATC. Unlike the pilots, air traffic controllers at the ATC can "see" the bigger picture, i.e., other aircraft or incoming storms, on computers in the control center and can appropriately advise the pilots to avoid dangerous situations. Such flying is termed a flight based on Instrument Flight Rules (abbr. IFR) or an IFR flight or simply an IFR. Nowadays, practically all large aircraft and many smaller ones as well fly under IFR as it allows them to

operate at much higher altitudes and in significantly worse weather conditions. These capabilities are a significant advantage for many reasons. Pilots flying under VFR rely on their vision, on what they can see, such as landmarks or terrain features - distinctive visual cues. Therefore, flying in or above the clouds where there are no distinctive terrain features to see, and operating in storms or even at night with reduced visibility is neither practical nor safe for VFR flights. Also, higher altitudes equal lower fuel consumption. All of these are the reasons that make IFR flights much more reliable and convenient to use for long-distance transportation [4]. Therefore, IFR flights make up the majority of all modern air traffic.

Flying under VFR may not be as popular as it used to be, but it remains the most popular operation mode for small aircraft. Moreover, it has been regaining some of its former attention with attempts to incorporate Unmanned Aerial Vehicles (abbr. UAV) into the air traffic on a larger scale, e.g., Amazon's project Prime Air [5, 6] or Google X's project Wing [7]. In these and other similar cases, fully autonomous UAVs should be integrated into day-to-day air traffic. These machines are small, and they are only able to fly at very low altitudes. Since in such altitudes only VFR flights are permitted, the UAVs need to fly under Visual Flight Rules.

This thesis aims to create a behavior model of a flight based on Visual Flight Rules (further referenced only as *model*). Such a model can then be used in an air traffic simulation to study modern ATC, identify its weaknesses and propose extensively tested solutions to them. Visual Flight Rules are still an essential part of modern air traffic, and thus they have a rightful place in any such simulation.

The task is further explored and described in Chapter 2. Theoretical solutions to problems specified there are proposed in Chapter 3. Practical implementation of the suggested solutions is then described in Chapter 4. After that, in Chapter 5, follow results of several experiments conducted on the resulting behavior model to assess its correctness and efficiency.

Chapter 2

Problem Specification

Several challenges need to be solved to create a proper behavior model of a flight based on Visual Flight Rules.

First of all, a generator of VFR flights is required to quickly and efficiently produce various scenarios that can be used to demonstrate possible situations pilot flying under VFR may encounter. Such scenarios are also essential to test all proposed strategies and correctness of their final implementation. The generator should produce the scenarios in an appropriate, portable and easy to handle format. Generated flight plans should also be somewhat reasonable. The VFR flights have their purposes, they usually avoid no-fly zones such as airspace above military bases, and like every flight, even they require a certain amount of fuel to fly their planned route.

After the creation of a flight plan, the pilot's behavior itself follows. First, the pilot needs to execute all pre-flight procedures such as file the flight plan and report the VFR flight to the nearest ATS. They also have to allow all other airplanes with higher priority, based on their status, to either land or to take off before they can depart from the (departure) airport. Following the takeoff, provided the flight is not required to execute orders from ATC (i.e., it is, in fact, a Visual Flight Rules flight), the pilot must maintain minimum flight altitudes defined in the current airspace, watch out for possible conflicts and solve them if necessary in compliance with Visual Flight Rules. Once the flight reaches its terminal (arrival) airport similar procedures to those performed before takeoff must be repeated before the pilot may finish the flight by landing and taxiing to its designated parking space. The rules are properly stated and further examined in Section 2.1.

For purposes of testing and future usage of this work, proposed strategies will be implemented in AgentFly simulation framework. This system simulates air traffic using agent-based modeling approach and is further described in Section 2.2.

The implementation is aimed to be as much general as possible. Some rules regarding aviation were unified by International Civil Aviation Organization (abbr. ICAO). These will be taken into consideration. However, Visual Flight Rules are one of many things that remains different throughout the world and each airspace may have its own, slightly different, VFR regulations. So for purposes of the simulation, rules issued by Air Navigation Services of the

Czech Republic (abbr. RLP) will be used.

2.1 Visual Flight Rules

In this section, a brief overview of the Visual Flight Rules issued by RLP is stated and discussed in regards to this thesis and its goals. The complete Visual Flight Rules can be inspected in [8, 9].

2.1.1 Airspace Overview

Airspace is a three-dimensional portion of the Earth's atmosphere. There are many defined airspaces all over the world, usually national (country airspace).

Each airspace is further divided into horizontal classes. ICAO defines seven such classes designated A through G. Classes A-E are considered to be controlled airspace, and every aircraft operating in any of these classes must follow the instructions from ATC [10]. Classes F and G are uncontrolled airspace, and aircraft operating in these classes usually do not have to maintain permanent contact with air traffic services.

RLP adopts the ICAO airspace division, but it effectively works only with classes C, D, E and G [11]. With regard to ICAO definitions, classes C, D, E are defined by RLP as controlled airspace and class G as uncontrolled airspace. Each class has different requirements on a VFR flight in areas such as visibility, speed, radio communication, etc. and there are also different services offered to VFR in each class. The list of all services provided to VFR flights and requirements on them in accordance with Visual Flight Rules issued by RLP can be viewed in [11].

The airspace is also divided into areas that may impose other restrictions on individual flights such as restricted or prohibited areas. Some of these areas are long-term, others are only short-term. Special clearances need to be obtained for the VFR pilots to be able to operate their aircraft in such zones. For that reason, VFR flights should, in general, actively avoid these areas.

2.1.2 Standard Airport Procedures

On controlled airports, pilots must perform several procedures [12].

On departure, pilots must first verify all their information and get clearance for starting the engines and for the departure. After that, they taxi to a holding point at a specific runway. There, they must wait until they receive *clear to take-off*. Only then, they may take off and leave the airport space.

On arrival, pilots must circle above the airport until they are given clearance to land. After landing, they must vacate the runway at first available exit and taxi to their designated position (gate or hangar - based on the type of the given flight).

On uncontrolled airports, the similar procedures should take place, only without the assistance of the ground control.

2.1.3 Minimum Flight Altitudes

The quote of definition of minimum VFR flight altitudes from [8] follows:

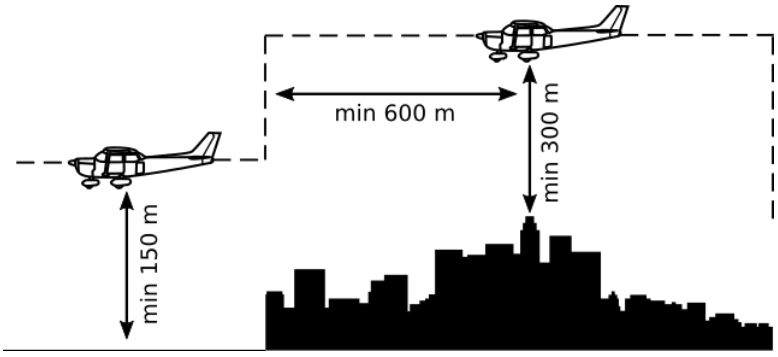
> Except when necessary for take-off or landing, or except by permission from the Civil Aviation Authority, a VFR flight shall not be flown:
>
> a) over the congested areas of cities, towns or settlements or over an open-air assembly of persons at a height less than 300 m (1000 ft) above the highest obstacle within a radius of 600 m from the aircraft
>
> b) elsewhere than as specified in a), at a height less than 150 m (500 ft) above the ground or water, or 150 m (500 ft) above the highest obstacle within a radius of 150 m (500 ft) from the aircraft.
>
> ([8])

Sometimes, other situations are included in the formulation, or different altitudes are stated, but this is the generally used formulation of minimum flight altitudes in Visual Flight Rules. Note, that the values specified in meters and the values specified in feet are not exactly the same which is caused by the conversion constant between meters and feet, but they are similar enough. It holds:

$$1\ ft = 0.3048\ m. [13]$$

Hence:

$$500\ ft = 152.4\ m \approx 150\ m,$$
$$1000\ ft = 304.8\ m \approx 300\ m,$$
$$2000\ ft = 609.6\ m \approx 600\ m.$$



Minimum flight altitudes in meters.

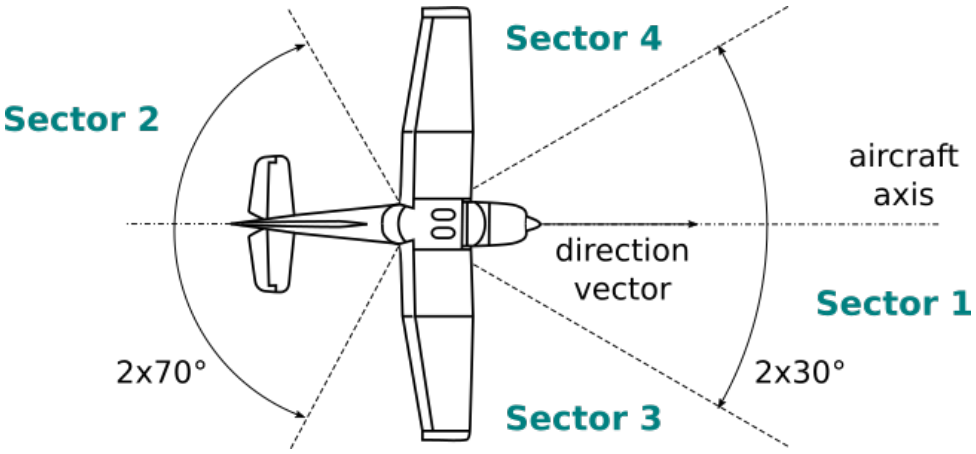
Figure 2.1: Minimum flight altitudes according to RLP

2.1.4 Collision Avoidance

During the flight, VFR pilots must constantly watch out for possible conflicts (collision hazards) with other traffic. If they find themselves in any such conflict, they have to perform an evasive action in accordance with the Visual Flight Rules. The quote of rules for avoiding collisions described in [8] follows:

> The pilot shall constantly monitor airspace in the vicinity of aircraft, regardless of class of airspace in which the aircraft is operating. An aircraft shall not be operated in such proximity to other aircraft as to create a collision hazard.
>
> **Right-of-way.** The aircraft that has the right-of-way shall maintain its heading and speed. An aircraft that is obliged by the following rules to keep out of the way of another shall avoid passing over, under or in front of the other, unless it passes well clear and takes into account the effect of aircraft wake turbulence.
>
> **Approaching head-on.** When two aircraft are approaching head-on or approximately so and there is danger of collision, each shall alter its heading to the right.
>
> **Converging.** When two aircraft are converging at approximately the same level, the aircraft that has the other on its right shall give way, except as follows:
>
> a) power-driven heavier-than-air aircraft shall give way to airships, gliders and balloons
>
> b) airships shall give way to gliders and balloons
>
> c) gliders shall give way to balloons
>
> d) power-driven aircraft shall give way to aircraft which are seen to be towing other aircraft or objects
>
> **Overtaking.** An overtaking aircraft is an aircraft that approaches another from the rear on a line forming an angle of less than 70 degrees with the plane of symmetry of the latter. An aircraft that is being overtaken has the right-of-way and the overtaking aircraft, whether climbing, descending or in horizontal flight, shall keep out of the way of the other aircraft by altering its heading to the right, until it is entirely past and clear. A sailplane overtaking another sailplane may alter its course to the right or to the left.
>
> ([8])

The task of deciding the type of conflict will further be simplified by dividing the surrounding area around an aircraft into four sectors as can be seen in Figure 2.2.



- **Sector 1** – head-on conflict
- **Sector 2** – overtaking conflict
- **Sector 3** – converge conflict with the other aircraft having right-of-way
- **Sector 4** – converge conflict with the given aircraft having right-of-way

Figure 2.2: Collision avoidance sectors

2.2 Simulation Framework

The behavior model of a VFR flight developed in this thesis will be implemented in the environment of AgentFly software. AgentFly is a multi-agent system enabling large-scale simulation of civilian and unmanned air traffic. The system is built on a multi-agent platform A-Globe [14].

2.2.1 A-Globe

A-Globe is an open multi-agent environment. It supports the integration of an arbitrary number of distributed computational processes, i.e., agents. Among other things, the A-Globe platform offers high scalability, modeling, and simulation of various environments in which the agents are to operate and visualization support. All of the above is used by AgentFly framework that is built on top of the A-Globe platform, and all of the above will be to some degree used to implement and test theoretical propositions made in this thesis.

A-Globe platform encapsulates all necessary services for large-scale simulations of real-world problems. It offers Geographical Information System-like services to the user which makes it suitable for simulating global air traffic as it can easily maintain exact GPS coordinates of all active airplanes (agents) and it can serve as a communication infrastructure between them.

2.2.2 AgentFly

The AgentFly is highly scalable multi-agent air traffic simulator. It is capable of worldwide simulation with hundreds of thousands of flights.

The system has two major components.

Firstly, it models aircraft and its pilot. Aircraft's characteristics are based on performance models from EUROCONTROL's Base of Aircraft Data (abbr. BADA). The pilot communicates with ATC using radio and is responsible for confirmation and implementation of provided control clearances.

Secondly, ATC agent that represents the human controller is simulated. The agent is responsible for a given sector and it provides ATM services in it. For each flight, ATC agents build an internal flight information model. The internal flight model integrates controller predictions and uncertainness.

Chapter 3

Approach

Several problems connected with VFR flights are already implemented in the AgentFly framework. The behavior of a controlled flight (behavior in controlled airspace), standard airport procedures and no-fly zones have already been solved in AgentFly. Since those problems have already been solved, this work will not discuss them any further. This thesis will focus solely on the parts of VFR flights' behavior that are not implemented in AgentFly yet and thus are the missing pieces for putting together a working behavior model of a flight based on Visual Flight Rules. As a result, this work will further on, without other specifications, always assume that the VFR flight is taking place in uncontrolled airspace (i.e., class G according to Visual Flight Rules issued by RLP) without any special restricting conditions.

In the following sections of this chapter, solutions to individual problems relating to Visual Flight Rules flights that have been specified in Chapter 2 and have not been solved yet in AgentFly are proposed.

3.1 Generation of VFR Flights

The first problem determined in Chapter 2 was a generation of reasonable VFR flights in an appropriate format. Such format will be a flight plan for each aircraft that is already used by IFR flights whose behavior is already implemented in AgentFly system. This flight plan is similar to the one that is used in real-life air traffic which makes its use even more adequate. Without considering administrative data such as flight identification etc., the flight plan consists of an ordered list of waypoints that the pilot is supposed to pass during the flight. Waypoints are coordinates on Earth, and they are expressed in latitude and longitude.

Flights based on Visual Flight Rules are often sightseeing flights. Therefore, it would make sense for their destinations to be famous landmarks such as castles, mountain ranges, etc. However, for this work, it is not necessary for the flights to have such realistic destinations. It is only necessary to generate series of scenarios that can be used to test the correctness of the proposed model. Generated flight plans should contain behavior such as flying from the airport to a particular destination, cruising above it for a while and then flying to another one or straight back to the airport. A smaller percentage of

flights could also be transport flights that would only fly from one airport to another. Flight plans meeting these criteria can easily be constructed without the need to obtain additional information regarding real-life landmarks in the surrounding area of the flight's departure airport. Thus, no real-life landmarks will be used as waypoints when generating flights plans for testing of the proposed model.

Generation of a flight plan itself is rather easy and straightforward. The core element of a flight plan is an ordered list of waypoints that are to be visited. With this in mind, the task of generating a flight plan can be reduced to a simple operation of rotation of a point in 3D space (not translation because airplanes fly along circular segments - arcs which are given by the shape of the Earth). The initial waypoint will be the position of the airport where the flight takes off from (expressed in AgentFly's Cartesian coordinate system). This position can then be rotated in an arbitrary direction by an arbitrary angle to generate the next waypoint of the flight plan. Repetition of this action may generate any number of waypoints. Position of the arrival (terminal) airport will be the final waypoint in the flight plan. This way, a flight plan, sufficient for the usage intended by this thesis, can be generated with ease and in a little amount of time.

3.2 Minimum Flight Altitudes

As was already mentioned above, VFR flights are often sightseeing flights, and their pilots fly as close to the surface as they can without violating the minimum flight altitude so that the passengers or even the pilots themselves see the ground in as much detail as possible. It is necessary to express elevation of surface the VFR is currently flying over to simulate such behavior. Furthermore, the surface must distinguish whether there is a congested area or not in order to comply with Visual Flight Rules stated in Subsection 2.1.3.

For that, it is appropriate to use real-world data describing terrain elevation and cities' areas along with their heights. According to the Visual Flight Rules, information about open-air assemblies should also be obtained as these are deemed congested areas too. But as open-air assemblies usually have a short lifespan and both their locations and durations often differ, this information would be difficult, sometimes even impossible, to obtain and work with and thus it will be further ignored.

Areas of cities can be obtained from OpenStreetMap which is an open, community-developed map of the world providing all its data as *open data* under Open Data Commons Open Database License [15].

However, when talking about areas of cities, it is essential to distinguish the area that belongs to the city based on the administrative partitioning of the country and the actual built-up area that classifies as a congested area of the city. The entire administrative area may contain dense forests, vast fields, water surfaces, etc. but the pilot must increase the altitude of his or her airplane only when flying over built-up areas. OpenStreetMap offers quite an extensive description of all administrative areas of cities. But unfortunately,

it lacks the information describing only built-up areas which are necessary to obtain for accurate simulation.

Two possible solutions come to mind as to how to solve this.

One of them being download all buildings of given city and join them together to one polygon covering all built-up areas of the given city. However, there are obvious problems with this approach. Imagine a city that consists of more than one built-up area. It could be that the city includes some neighboring villages or there is, in fact, a secluded area somewhere on the administrative area of that city (e.g., industrial areas or cottages used for summer vacationing). It would be possible to apply some clustering to the data to tackle this particular problem. But even though, other problems persist. Assume, there is a set of polygons of buildings that represent a single built-up area. At this point, the goal is to find the smallest polygon that covers all of the buildings. Such polygon is likely concave, and it cannot be constructed by merely joining one building to another as a different ordering of the buildings may produce completely different shapes. There are possible solutions to this problem as well. For instance, the buildings could be declared single points (e.g., their geographical centers) and then it would be a task of finding polygon that would characterize the shape of the set of those points (buildings). This topic has been explored before, and there are efficient algorithms that could solve it. Some of them have been studied and compared in [16]. This technique seems somewhat problematic but doable. The resulting shapes might be reasonable approximations of given cities' built-up areas.

The second approach is to obtain the entire administrative area and then subtract areas such as woods, water surfaces, etc. This way, obtained areas may be disjoint polygons describing separated city center, villages and secluded areas which is desirable. Intersection, union, and the difference between two polygons is required in many fields of computer graphics or geographic information systems (abbr. GIS), ergo even here efficient algorithms exist that could be used to solve the problem. For instance, an efficient algorithm for polygon operations is proposed in [17].

The second approach seems to be less problematic and more feasible when compared to the first one. Among other things, it doesn't require to keep all buildings of the city (of which there are potentially millions) in memory at the same time and perform operations on them. Hence, the second approach will be used to obtain data for this thesis. However, there is no right answer as to which of the proposed approaches is better. Advantages and disadvantages recognized only by brief examination of both methods have been discussed. The topic could be examined more rigorously and explored in much more detail, but that is not the aim of this work.

Even with polygons of built-up areas, the height of cities and elevation of terrain still need to be obtained in order to model the reality accurately. This information may easily be acquired from height maps created using satellite or aerial imagery.

In the following subsections, proposed technologies and techniques are further examined, and their use is specified. All of that, in order to obtain

data necessary for the resulting model to be able to maintain minimum flight altitudes according to the Visual Flight Rules.

3.2.1 OpenStreetMap

OpenStreetMap (abbr. OSM) is a free, editable map of the world that provides all of its data under Open Data Commons Open Database License. OSM is built by volunteers from all over the world [18]. As open, community-developed map, it may not be fully reliable, but data obtained from OSM will be sufficient for this work.

OpenStreetMap offers global geographic data. However, VFR flights are usually sightseeing flights, and they fly only locally. It is common for VFR flights to be performed over a single country or two if the pilot is cruising in the vicinity of a country border. For that reason, it is not necessary to download areas of cities all over the world and take them all into account during the testing of the implementation. It is only necessary to download cities in a single area and perform simulation using them. In this case, as this work takes into consideration the Visual Flight Rules issued by RLP, only cities of the Czech Republic will be downloaded and further processed. Interface allowing just that (download only small portions of the OSM data from appropriately tagged layers) is Overpass API [19].

It is possible to display data returned by Overpass API with ease, using Overpass turbo. Overpass turbo is web-based data mining tool for OpenStreetMap [20]. It accepts a query to Overpass API, sends it to an interpreter, transforms received data to geoJSON and displays the results on an interactive map. It is a useful tool for debugging complicated queries to OSM and visualizing received data.

Data in OpenStreetMap. OpenStreetMap models the physical world using three core elements. Those are nodes, ways, and relations [21]. Nodes define positions on the globe in latitude and longitude coordinates. Nodes are the only elements that possess information about their location. Ways are sequences of nodes that allow defining linear features and boundaries. Relations are structures that can hold specific sets of nodes, ways, and even other relations together and explain their collective significance. All of these elements may have various tags that specify, what they represent. A tag is a pair of two items - key and value [22] - and it is used to describe the meaning of each OSM element. OSM has an extensive list of tags that may be used for element description along with detailed instructions for tagging newly created elements.

The set of tags used by OSM is large. Unfortunately, no single tag describes built-up areas of a city. However, there are tags that distinguish areas such as forests, fields, meadows, lakes, ponds, etc. This fact further supports convenience of the proposed method to obtain built-up areas (i.e., download entire city administrative area and subtract all areas without buildings from that).

Overpass API. Overpass API is read-only application programming interface that acts as a database over OpenStreetMap and provides parts of the OSM based on tags of the entities or various GIS-like operations [19].

Overpass API accepts queries written in two querying languages - Overpass XML and Overpass QL (Overpass Querying Language). Overpass XML expects the query to be formatted as an XML (eXtensible Markup Language) document whereas Overpass QL is a language quite similar to the C programming language.

Apart from access to OSM data, one can query Overpass API for an additional data element next to the basic nodes, ways, and relations. This element is called area (filled polygon). All relations and enclosed ways with tags suggesting that the elements encapsulate actual area are processed into area elements. It is important to mention that the original relations or ways are not deleted. They are still in OSM, and one can query directly for them. But Overpass API offers additional filtering possibility by keeping track of these areas.

3.2.2 City Polygons

Once downloaded from OpenStreetMap, the data needs to be processed into final built-up areas.

When obtained from OSM, nodes of the polygons will have geographic coordinates (latitude and longitude). Stereographic (abbr. SG) projection will be used to convert these spherical polygons into planar ones. Once in the plane, the workflow is rather straightforward - take city's administrative area polygon and subtract all *polygons of nature* from it.

However, this operation has a couple of drawbacks on its own. Real-world polygons are quite detailed, and the repeated subtraction will gradually make the initial polygon even more complex (e.g., disjoint polygons and holes will likely be created) which will cause the subtraction to run significantly slower. So the operation would take an enormous amount of time to finish. Moreover, the resulting polygon (even if not too complex) would be made up of a high number of nodes. Such polygon is not suitable to be used by the model as it would take too much time to determine if current flight plan intersects with airspace above the polygon. This calculation needs to be very fast as it will be potentially performed in every position of the flight (i.e., in every step of the simulation).

A few algorithms will be proposed to obtain simpler polygons that are still good approximations of given city's built-up areas.

Firstly, subtracting from large polygon soon causes it to have many holes and thus to be extremely difficult to work with. It is a good idea to split the polygon into smaller pieces and then subtract from them. Afterward, if the resulting polygons share an edge (are touching), they will be joined back together. If they do not share any edge, then they will be declared two separate built-up areas.

Secondly, it is not necessary (even not desirable) for the polygons to consist of hundreds of points. Some ways to reduce the number of polygon vertices

are required. Two simple techniques and one advanced will be used in the thesis.

The simple techniques will be **grid rounding** of the vertices and **small angles removal**. Both are rather straightforward. In grid rounding, a regular grid will be laid on top of the polygon, and each of its vertices will be snapped to the closest point on the grid. The process is depicted in the Figure 3.1. In small angles removal, joined segments with an angle between them that is smaller than given threshold, will be replaced by a single line segment. The process is depicted in the figure Figure 3.2.

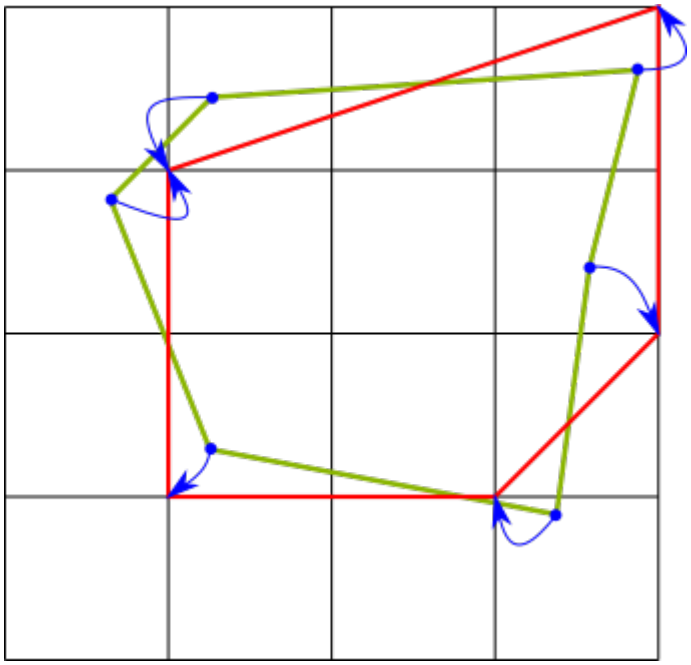


Figure 3.1: Grid rounding

The advanced technique used to reduce the number of polygon vertices will be **Ramer-Douglas-Peucker algorithm** (abbr. RDP) for line simplification that is further explained in paragraph Ramer-Douglas-Peucker Algorithm.

Last, but not least, polygon subtraction combined with rounding and computer's limited floating number precision could cause for the resulting polygon to degenerate. It is difficult to handle all possible degenerations, but one particular case will be addressed here. It is possible, in some cases, that self-intersecting polygon may be a result of the above-proposed processing. This degeneration can be resolved by finding the intersections and splitting the polygon into smaller simple polygons. If the situation of the self-intersecting polygon will arise, the following approach will be applied to resolve the situation. Points, where the polygon intersects itself, will be found using **Bentley-Ottmann algorithm** (abbr. BO). BO is further explained in paragraph Bentley-Ottmann Algorithm.

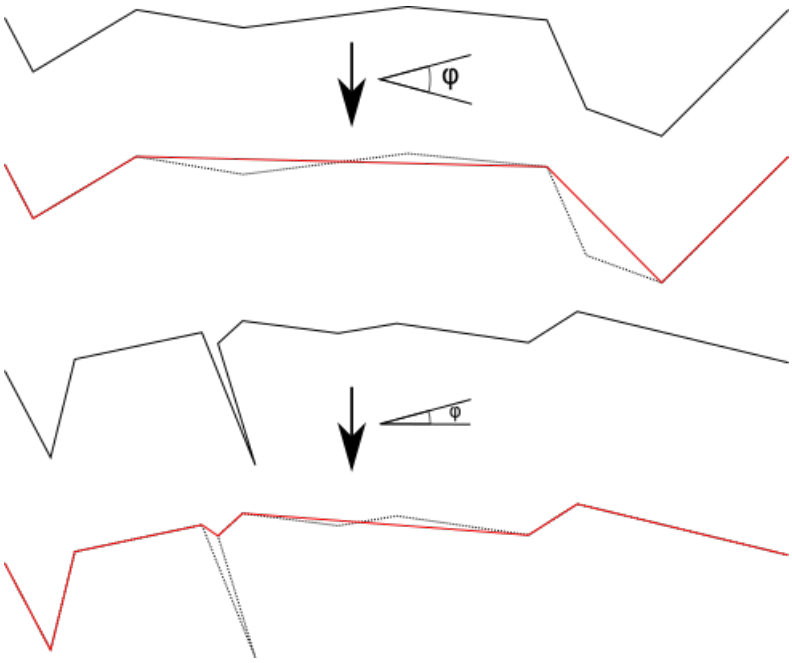


Figure 3.2: Small angles removal

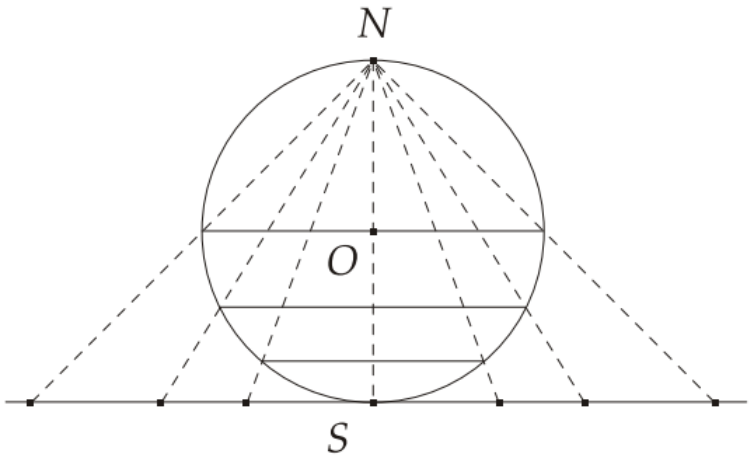
After all of this is done, a 2D polygon representing the built-up area of a city (further referred to just as *city polygon* or *city area*) will be obtained. However, VFR pilots need not only to know, if they are flying above a city but also the height of the city below them so that they can adjust their altitude according to Visual Flight Rules. A city is quite a complicated structure, and its height often differs at each position of its area. This problem will be solved by simplification of reality. For simulation purposes, the height of each city will be considered uniform across its entire area. This value will be the average height of the city.

The average height of a city will be obtained from height maps by random sampling of the city polygon. The area will first be triangulated. Then each triangle will be randomly sampled using the formula for random uniform triangle sampling presented in [23]. Average of sampled elevations will be computed, thus obtaining average height for each triangle making up the city polygon. Afterward, the weighted sum of the determined heights will be computed in order to gain the average height of the entire city area. The weights will be portions of the total area of the city polygon that each triangle occupies.

With the city height, the 2D polygon of the city's built-up area can be raised into a 3D polygon that encapsulates the city area as a whole. Once such *bounding box* is created for every city, the task of detecting if an aircraft is flying above a city reduces to a simple check if the flight route intersects with given 3D polygon.

The following paragraphs discuss the procedures and techniques suggested above to be used for obtaining the desired data in greater detail.

Stereographic Projection. Stereographic projection is a technique of mapping a spherical surface onto a plane [24]. It is a type of azimuthal projection that is based on casting rays from the projection center onto the target plane. In SG projection, the sphere is projected onto a tangent plane from a point that is located on the sphere, and that is antipodal to the contact point with the tangent plane. This projection has several convenient mathematical properties which are the reason for the SG projection being used.

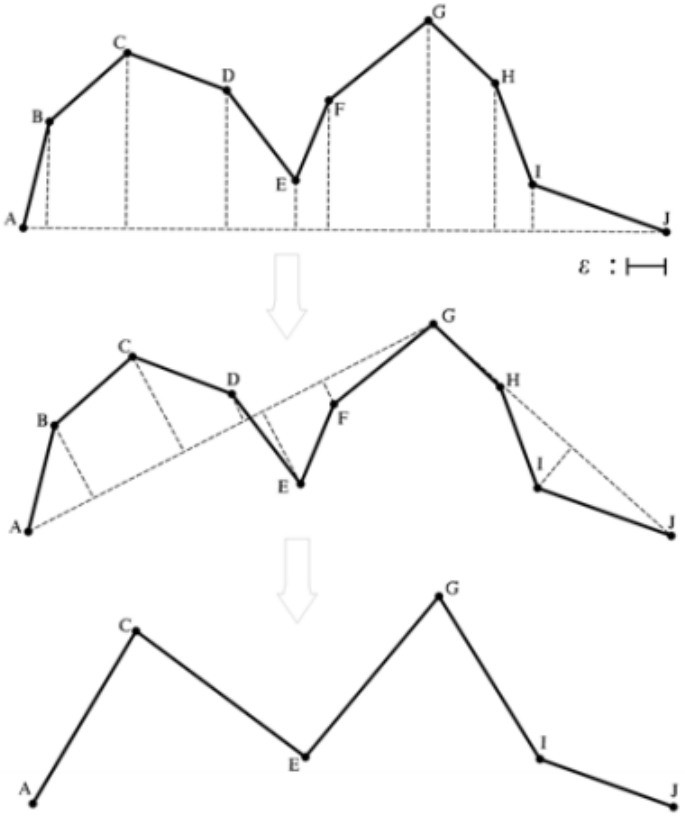


This figure has been taken from [24].

Figure 3.3: Stereographic projection principle

Ramer-Douglas-Peucker Algorithm. Ramer-Douglas-Peucker algorithm is an algorithm for line simplification [25]. It takes a line (polyline) to simplify and one parameter ϵ. The algorithm finds a point P. P is the farthest point from the connector of the endpoints of given line. If P's distance from the connector is less or equal to the parameter ϵ, the line is replaced by the connector itself. Otherwise, the polyline is split into two - the first going from the starting point to P and the second going from P to the endpoint. The procedure is recursively repeated on both new segments. Once execution stops, the simplified line is constructed. This algorithm can easily be modified to work on polygons and thus will be used during processing of the downloaded data.

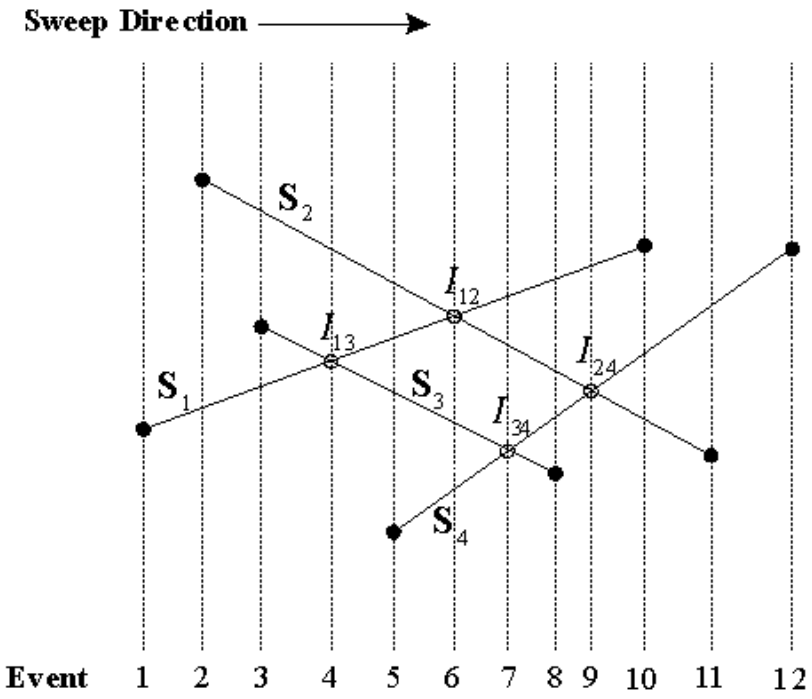
Bentley-Ottmann Algorithm. Bentley-Ottmann algorithm is an algorithm for finding intersections of line segments in 2D [26]. It uses a concept known as a *sweep line* and is sometimes referred to as a **sweep line algorithm**. First, endpoints of all line segments are linearly ordered by their coordinates. These points are known as events and are kept in an ordered structure known as event queue. Afterward, an imaginary line passes through the event queue and processes (*sweeps*) events one at a time. The sweeping line keeps track of all line segments that it currently intersects - by adding a line segment



This figure has been taken from [25].

Figure 3.4: Ramer-Douglas-Peucker algorithm

to a dynamic set when it encounters the first endpoint and by removing the line segment when it encounters the second endpoint. Only those segments that are in the sweep line's set at the same time can intersect each other. Moreover, the sweep line keeps the list ordered in *above-below* relation. When a segment intersects with another segment in sweep line's list, their positions are swapped, and the intersection is added to the event queue. When the intersection is encountered in the event queue, it is checked that there is not another intersection with above or below segment. If there is, another intersection is again added to the event queue, and appropriate exchanges are made. This outputs ordered list of all intersections. Bentley-Ottmann algorithm can easily be modified to work on self-intersecting polygons (by merely ignoring "intersections" of endpoints). It can also output the pairs of segments that form each intersection which is suitable for further polygon refactoring (i.e., splitting the polygon in found intersections).



This figure has been taken from [26].

Figure 3.5: Bentley-Ottmann algorithm

3.2.3 Terrain Elevation

When not over any congested area, every airplane must still be operated at least, according to Visual Flight Rules issued by RLP, 150 meters above the terrain. Such information may also be obtained from height maps of the given terrain. The elevation should be measured with regard to the highest obstacle in a 600-meter radius of the aircraft. As it would be necessary to compute average height of a 600-meter circle around the airplane in every moment of the simulation, this fact will be simplified from here on. The elevation will only be computed for discrete positions of the currently planned flight route, but it will be calculated much further than only 600 meters ahead. In every moment of the flight, the pilot is looking out of the cockpit a sees terrain features ahead. He or she can recognize hills, mountains or valleys and from that estimate the relative elevation to their current position. However, they can better recognize such features over a shorter distance rather than longer.

To account for all of the above, following procedure will estimate appropriate minimum flight altitude over the terrain. The height map will be sampled along the current flight route. Sampling will be divided into two parts - short-range sampling and long-range sampling. Each sampling will produce the highest altitude of the sampled section of the flight route. If the difference between plane's altitude and height produced by short-range sampling will violate Visual Flight Rules, the aircraft will start climbing. If the difference

between plane's altitude and height produced by short-range sampling will not violate Visual Flight Rules but the difference between aircraft's altitude and the elevation generated by long-range sampling will, then the airplane will maintain its current altitude as the pilot can fly lower for now without violating VFR or placing the aircraft in danger. If and only if both differences will not violate Visual Flight Rules, the pilot will start descending closer to the surface.

The last question is, how often should the terrain be sampled. In real-life, pilots scan their surroundings continuously with their vision, but in discrete simulation, this is modeled by sending and handling events. Hence, terrain sampling will also be triggered by a received event. The sampling will be done once. The initial sampling will give a position P where the aircraft is in violation with Visual Flight Rules (if no such location is found, then this position will be the position right after the sampling range). Then, an event will be planned to arrive a certain amount of time before the airplane reaches P. When the event is received, the sampling will be done again, new P will be found, and a new event will be scheduled.

The route sampling is depicted in Figure 3.6. S and L denote the short and long sampling range, respectively. ΔS signifies the sampling step at the short-range sampling and ΔL indicates the sampling step at the long-range sampling. h_S are h_L are the highest elevations obtained from short-range and long-range sampling, respectively.

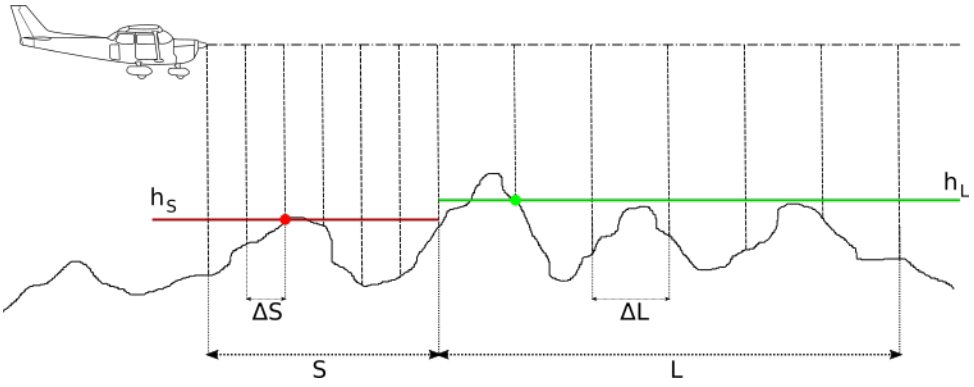


Figure 3.6: Flight route sampling

3.3 Collision Avoidance

Another problem that needs to be solved to create a model of flight based on VFR is the detection of a conflicts with another aircraft and execution of an appropriate evasive maneuver in accordance with Visual Flight Rules.

The majority of conflicts in real-world situations is solved using so-called see-and-avoid principle. The name is self-explanatory - pilots are to scan their surroundings for incoming traffic and when they spot a conflict with another aircraft, they are to perform evasive maneuver according to VFR.

However, this principle has its weaknesses [3]. Human pilots are generally not paying as much attention to their surroundings as they should. Furthermore, even with pilot's full attention, many other factors have negative effects on human vision, such as higher altitude or the direction of sunlight, that make see-and-avoid principle that much more unreliable. For this work and the simulation, a simplification of reality will be adopted. The pilot will be given a predefined, static field of view (abbr. FOV) a he or she will be able to see everything in it at all times. The continually changing range of pilot's vision based on the shape of the terrain ahead will be ignored, as well as pilots turning their heads and not being able to see all their surroundings at the same time.

If another airplane enters the FOV, its GPS position will be passed to the pilot to process. The pilot will not have access to any additional information about the conflict flight such as its planned flight route. The conflict will have to be detected only from the changes of the GPS position over time. Each pilot will have to detect the conflict on their own. There will be no central evaluator just like there is not one in real life.

3.3.1 Conflict Types

Visual Flight Rules specify three types of conflicts. The type of a single conflict will be determined based on sectors defined in Figure 2.2. The rules also state several exceptions for aerial vehicles such as airships, gliders, etc. These exceptions will be ignored. The model will assume all involved entities are a power-driven, heavier-than-air aircraft.

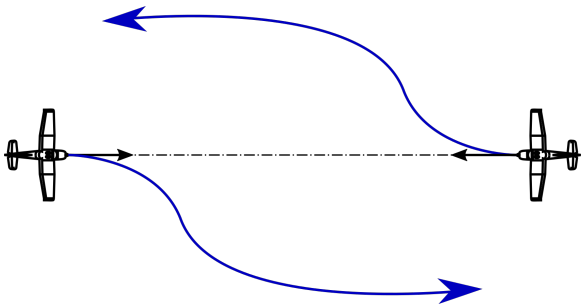
Head-on and converging conflicts are both straightforward, and there are not any significant special cases to consider. However, three situations classify as overtaking. The first one is that a faster aircraft is overtaking a slower one. That is a traditional overtaking conflict. The second one is that a slower aircraft is *overtaking* a faster one. If this situation occurs, there is no actual conflict as the overtaking itself never takes place. The third one is that both airplanes have approximately the same speed. In this case, the model will treat as the faster aircraft the one, that would reach the collision position (the "intersection" of airplanes' trajectories) first.

3.3.2 Conflict Resolution

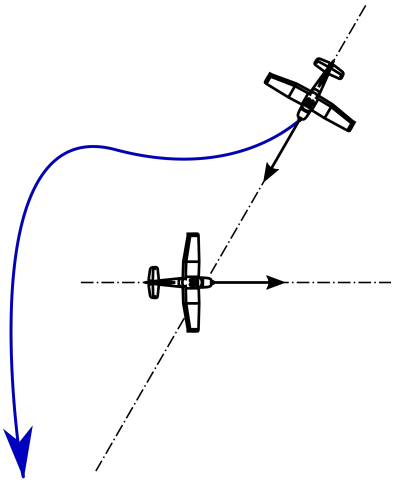
Each conflict type will be resolved differently, but all in accordance with rules stated in Subsection 2.1.4. Graphical demonstration of resolutions for each conflict type can be seen in Figure 3.7.

The goal of each resolution is to maintain a safe distance from the conflict by altering the flight route. The flight route must be altered in accordance with Visual Flight Rules. It is also a natural requirement that the flight route is changed as little as possible.

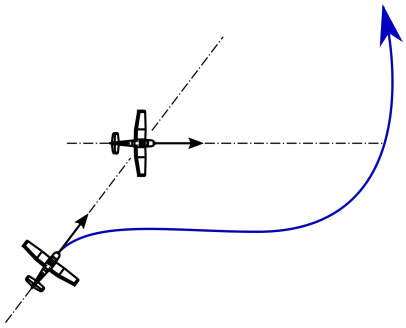
Additional waypoints will be generated to change current flight route in the simulation, They will be created with regard to the current positions of both the airplane and the detected conflict.



(a) : Head-on approaching resolution



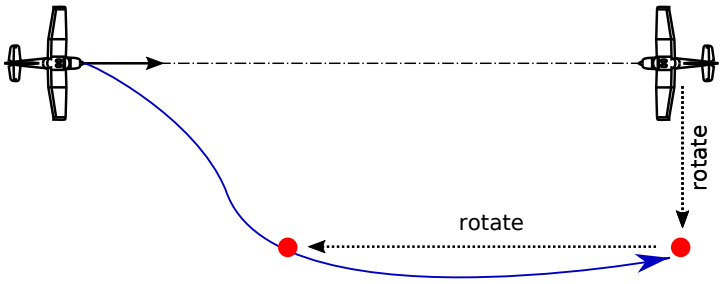
(b) : Converging resolution



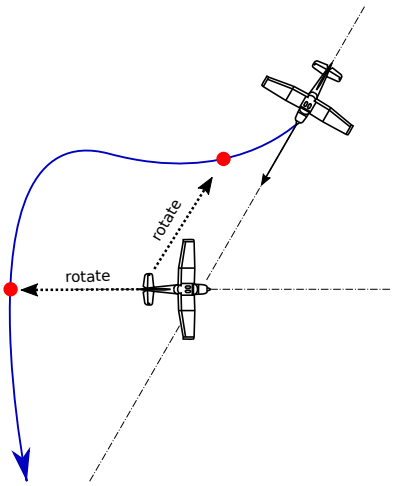
(c) : Overtaking resolution

Figure 3.7: Resolution of conflicts

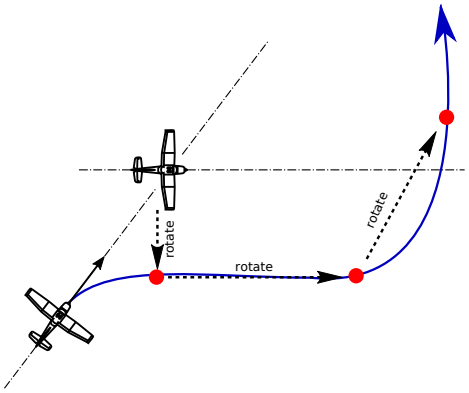
Positions will be rotated by a certain (necessary for safe evasion) angle in appropriate directions. The directions will be derived from the directions of the airplane and the conflict. Figure 3.8 approximately depicts intended transformations for each conflict type.



(a) : Head-on approaching resolution



(b) : Converging resolution



(c) : Overtaking resolution

Figure 3.8: Resolution of conflicts II

Chapter 4

Implementation

The model proposed by this thesis is implemented within AgentFly framework that is written in Java programming language. Hence, all source codes related to this work are also written in Java programming language.

The implementation only augments AgentFly framework, and thus, it honors the module structure and class hierarchy introduced by AgentFly. Written modules extend classes or implement interfaces that are used across the entire AgentFly to ensure readability, consistency and proper communication between individual modules. All source codes were implemented within *agentfly.atm.vfr.scenario* and *atc.utility* projects.

The main class that implements the proposed model is *EomPilotVfr*. It works with data supplied by *VfrFlightsGenerationScenarioPlayer* and *CitiesScenarioPlayer*.

EomPilotVfr is an AgentFly pilot module. In AgentFly, each entity (airplane) has its predefined behavior (based on filed flight plan) that can be altered at simulation runtime by supplying an arbitrary number of pilot modules. All pilot modules are part of a class hierarchy with base abstract class *EventOrganizerModule*.

As *EomPilotVfr* is also a pilot module that is intended to alter the predetermined behavior of VFR flights during the flight, it is part of the *EventOrganizerModule* hierarchy. The class consists of two major parts. The first one is responsible for maintaining minimum flight altitude over the entire course of the flight. The second one is aimed at detecting and resolving conflicts. If the two behaviors find themselves conflicting with each other, the collision avoidance takes precedence.

VfrFlightsGenerationScenarioPlayer and *CitiesScenarioPlayer* are both implementations of *ScenarioPlayer* interface that is a part of the AgentFly framework. Each scenario that is to be simulated by AgentFly is initialized by one or more implementations of the *ScenarioPlayer* interface. Every provided *ScenarioPlayer* is executed before the beginning of the simulation, and together, they prepare the simulation to run (e.g., they create all necessary agents or load data that will be required at simulation runtime).

VfrFlightsGenerationScenarioPlayer is responsible for generating all VFR flights that are to make an appearance during the simulation. *CitiesScenarioPlayer* creates 3D polygons encapsulating city built-up areas. It also prepares

them for optimized querying whether a given flight route intersects with any of them.

All of these modules along with several others that solve partial problems are described in more detail further on in this chapter.

4.1 Generation of VFR Flights

The generation of VFR flights is implemented in *VfrFlightsGenerationScenarioPlayer* module. By default, this class generates (pseudo-)random VFR flights. The randomization is partially parametrized and can be changed by the user.

The generation process is started by the selection of a departure airport. The position of the airport is expressed as a point on Earth (that is approximated as a regular sphere in AgentFly system) and rotated in a randomly selected direction. The newly obtained position (so-called waypoint) is considered to be the first destination of the flight. The new waypoint is then rotated again to generate another one. That is repeated a random number of times (with a predefined upper bound of the total number of destinations per flight). Finally, another airport is selected as the terminal point of the flight route (i.e., the arrival airport).

VfrFlightsGenerationScenarioPlayer distinguishes two types of VFR flights. The types are a sightseeing flight and a transport flight. For sightseeing flights, in addition to generating waypoints, the module also generates a cruising trajectory above the destination (e.g., a circle). Type of a flight is again selected randomly.

Apart from random flight generation, the class is also capable of generating VFR flights based on information passed in a data file. That allows users to supply their scenarios to the simulation in order to test the model using them.

Generated data are exported in instances of classes already implemented within AgentFly that are designed to hold information about the desired simulation scenario.

4.2 Minimum Flight Altitudes

The implementation of the behavior of maintaining minimum flight altitude consists of three separate tasks. The first one is to obtain polygons delimiting congested areas (only built-up areas for the purposes of this work). Next, terrain height maps need to be processed in order to be able to query for terrain elevation at any position. Finally, all of that data needs to be taken into consideration at simulation runtime.

4.2.1 City Data Creation

As was proposed in Section 3.2, the map data is downloaded from OpenStreetMap using Overpass API. A utility for the communication with Overpass

API online interpreter was created within AgentFly project. [27] was used as a partial inspiration for the implementation, but it wasn't used directly nor edited for the use in this thesis and the AgentFly framework as a whole. As the utility itself has nothing to do with air traffic, it was implemented in isolation from the core modules of the AgentFly, in *atc.utility* project, in *openstreetmap* package. The tool located in *openstreetmap.overpasser* package encapsulates several functionalities. It allows a fast and convenient building of queries in Overpass QL language, sending the created queries to Overpass API and parsing received responses into new Java objects defined in *openstreetmap.entities* package.

The main class that communicates with Overpass API is class *Overpasser*. *Overpasser* is a singleton object implemented as one-instance Java enumerate type. It holds the URL of the online Overpass API interpreter. It expects a query as a simple *String* that is sent to the online interpreter as a Hypertext Transfer Protocol (abbr. HTTP) request. Response from the Overpass API is returned as a stream with incoming data.

The obtained data need to be further processed. This task has also been implemented in *atc.utility* project, in *openstreetmap.processing* package. The main processing class is *Processor*.

Processor is again a singleton object implemented using one-instance Java enumerate type. The class has many fields (parameters) that can be adjusted to match the current need of thr user. The parameters include options such as what rounding (if any) should be applied to the polygons during processing or what (if any) SG projection should be used. The module also offers a parameter that determines if any debugging outputs will be produced during processing. Except for console output, images of the resulting areas can also be created using specialized *Visualizer* module. *Processor* awaits queries for cities that are to be downloaded and processed into polygons of their built-up areas. The class downloads the entire administrative area of each city. Afterward, it checks, if the city is made up of smaller administrative areas (such as villages without a town hall or separate city districts). If such subareas exist, they are downloaded one at a time. Along with them, areas of forests, water surfaces, etc. that intersect them are also fetched, and they are subtracted (polygon operation of difference) from the obtained administrative subareas. Once all the subtraction is done, *Processor* checks if some of the resulting polygons share a line segment. If that is the case, the polygons are joined together as one polygon. If the downloaded city has no such subareas, the subtraction is applied to the original city polygon. All resulting polygons are then checked by BO for intersecting themselves before they are returned. The implementation from [28] of BO was edited and incorporated into AgentFly to use in this situation. If it is detected that the polygon is self-intersecting, it is split into simpler polygons. However, the current implementation of the splitting is not ideal, and it would not handle some more complicated situations properly. Nevertheless, its results are more than sufficient, and if some inappropriate polygons are created, they are discarded. Once the computation is finished, all nodes in the original

administrative area tagged as a place (every settlement should have such a node that on a map is used to display the settlement's name) are downloaded and laid onto the resulting polygons. Each polygon that contains one of the place nodes is declared a built-up area with the name of the appropriate node.

Each built-up area is returned as an instance of the class *City*. *City* instances hold information such as city's name, polygon delimiting its built-up area and its average height.

4.2.2 City Data Integration

Exported instances of the class *City* are passed to *CitiesScenarioPlayer* when the simulation is started. This module converts *City* instances into instances of *GpsFpaSector* class that is already a part of AgentFly. It is used for airspace partitioning, and hence, it is optimized for querying whether or not a flight route intersects with its instance. All created sectors (cities) are encapsulated in an instance of *GpsFpaSectors* class.

GpsFpaSectors organizes all its sectors into binary search tree (abbr. BST). The created BST is a hierarchy of bounding objects (spheres), and it is constructed from its leaves to the root. First, all sectors are placed to the leaves of the BST. They are all their own smallest bounding object. Then repeatedly, two bounding objects are joined together (the two nodes of the BST are connected to a common parent node) and encapsulated by a new bounding sphere that bounds them both if and only if the created bounding sphere is the smallest one that can be created. This hierarchy can then be used for fast and easy detection if a flight route of any aircraft intersects with any city. And if so, that route is in violation of the Visual Flight Rules.

For the simulation purposes, it is suitable to visualize the obtained cities. To this end, module *CitiesLayerProvider* was created. *LayerProvider* is an abstract class that allows adding a new layer to the graphical user interface of the AgentFly system. As a part of *LayerProvider* hierarchy, the *CitiesLayerProvider* module displays all 3D polygons encapsulating the built-up areas of each city.

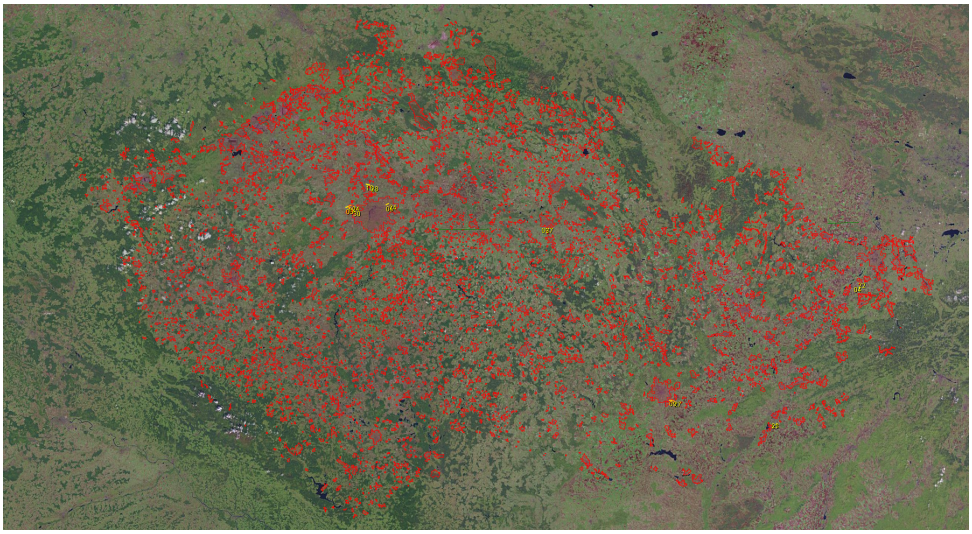
Visualization of the city built-up areas using AgentFly visualization tool with incorporated *CitiesLayerProvider* is shown in Figure 4.1.

4.2.3 Terrain Sampling

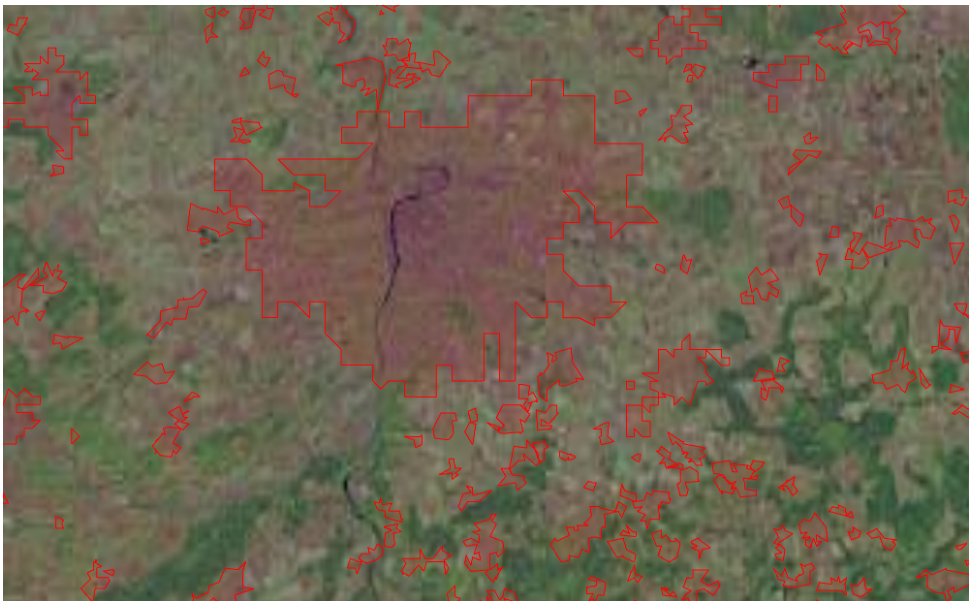
The terrain sampling is implemented using AgentFly module *ElevationMap*. This class loads height maps of the Earth and can be queried for a terrain elevation at any Earth coordinates specified by latitude and longitude.

4.2.4 Altitude Changes

The behavior of appropriately altering altitude based on the observation of the terrain elevation and congested areas is a part of *EomPilotVfr* module.



(a) : Cities of the entire Czech Republic



(b) : Detail of Prague and surrounding settlements

Figure 4.1: Cities of the Czech Republic

Firstly, the terrain is sampled in accordance with the strategy proposed in Subsection 3.2.3. All parameters of the sampling introduced in Figure 3.6 may be specified by the user in the configuration file. However, if not specified,

values default to the following:

$$S = 4\ km,$$
$$L = 6\ km,$$
$$\Delta S = 20\ m,$$
$$\Delta L = 50\ m.$$

Secondly, the hierarchy of the bounding objects constructed in *CitiesScenarioPlayer* is traversed to see if the current flight route intersects with a city either within the short sampling distance or the long sampling distance. Finally, the results are combined to obtain the minimum flight altitude for the short-range sampling (h_S) and the long-range sampling (h_L) that are in accordance with Visual Flight Rules.

At this point, there are several possible outcomes. Let A denote the current altitude of the airplane. Let E denote the event proposed in Subsection 3.2.3 that is called *CHECK_ALTITUDE_CONFLICT*.

a) h_S is higher than A. Then the airplane immediately increases its altitude to h_S, and the *EomPilotVfr* module plans E to arrive moments before reaching the position where h_S occurred.

b) h_S is smaller than A, but h_L is higher than A. Then the airplane maintains its current altitude A and the *EomPilotVfr* module plans E to arrive moments before reaching the position where h_L occurred.

c) Both h_S and h_L are smaller than A. Then the airplane decreases its altitude to the maximum of h_S and h_L and the *EomPilotVfr* module plans E to arrive moments before reaching the last position of long-range sampling.

4.3 Collision Avoidance

The detection and resolution of conflicts is the essential part of the *EomPilotVfr* module.

The pilot has a static FOV defined by range, horizontal angle, and vertical angle. FOV is static but has wider horizontal angle than normal human vision to account for real-life pilots turning their heads. FOV is visualized in AgentFly visualization tool using *SensorGpsRangeLayerProvider* class.

The parameters of FOV may be passed by the user. If not, they default to:

$$range = 10\ km,$$
$$vert\_angle = 90^\circ,$$
$$horiz\_angle = 180^\circ.$$

The goal of a possible evasion maneuver is to maintain given safe separation from the aircraft and to be in accordance with Visual Flight Rules. The safe separation is in current implementation set to 2 kilometers.

The visualization of both the FOV (orange circular sector) and the minimum safe separation (green circle) in AgentFly can be seen in Figure 4.2. The white corridor displays the safe separation along the whole planned flight route.

Figure 4.2: FOV visualization

4.3.1 Conflict Detection

Whenever an entity enters pilot's FOV, the pilot module obtains the entity's GPS coordinates. The coordinates are available to the pilot at each simulation step that the entity spends inside pilot's FOV.

For each entity that enters pilot's FOV an instance of a class derived from *ConflictEvaluator* module is instantiated. *ConflictEvaluator* is an abstract class that encapsulates evaluation of the possible conflict based on the changes in its GPS position. The class expects to be passed the new GPS coordinates at each simulation step via method *update*. The pilot module only works with a reference to *ConflictEvaluator*. Specific instances are created based on supplied parameters. That allows for a better generality of the model as it is not limited to a single conflict evaluation behavior. If a user wishes to test different evaluation behavior, they merely have to supply their class derived from *ConflictEvaluator* in parameters to the pilot module. All created instances are kept in a `HashMap`. *ConflictEvaluator* instances are referenced by their unique id for quick access when updating them.

For this thesis, *ExactGpsBasedConflictEvaluator* was implemented to serve as the evaluator. This module tracks the changes in the GPS position and from them approximates conflict's trajectory and speed. Based on the trajectories and speeds of both the conflict and the owner, the approximate minimum separation that will be achieved if both airplanes maintain current course is computed. If the separation is smaller than predefined minimum safe distance, the detected aircraft is treated as a conflict. Based on conflict's position, the type of conflict is determined (see Figure 2.2 and Subsection 3.3.1) so that the evasion maneuver is not in violation of Visual Flight Rules.

4.3.2 Conflict Resolution

If some of the detected entities are evaluated as a conflict, they are moved to a queue of conflicts waiting to be solved. Based on the type of the conflict, a different evasive maneuver is performed. However, generation of all maneuvers is implemented similarly. Additional waypoints are generated and added to the current flight plan to resolve the conflict. New waypoints are derived from the positions of the aircraft and the conflict. The positions are rotated in an appropriate direction derived from the orientations of both the aircraft and the conflict. The rotations are performed in a way so that the new flight plan is in accordance with defined evasive maneuvers in Visual Flight Rules (see Figure 3.8).

When the new flight plan is generated and registered, an event is scheduled to arrive a few moments later. The event signals that the pilot module should check if there were no complications and the evasion was successful. If not, then new maneuver is generated.

The conflicts are kept in the `HashMap` even after their resolution and even after they have left the pilot's FOV. They are only deleted when the *update* method has not been called on them for a while. That, to some degree, models human pilots who have some awareness of airplanes they detected during the flight. And only after a certain amount of time after the airplanes left their FOV, they "delete" those airplanes from their mind.

Chapter 5

Experiments

In this chapter, results of various experiments and tests conducted on implemented functionalities are presented. The experiments aimed to test the correctness of the implemented model.

It is an important note that scenarios presented further on in this chapter are purely fictional. They were only used to test the correctness of implemented model's behavior. The scenarios may violate specific air traffic regulations for a better demonstration of the model's properties.

For the test cases where hardware capabilities apply, the tests were performed on a computer with *GIGABYTE Z370 HD3P-CF* motherboard, *Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz* (6-core CPU), *64GB DDR4-2400* RAM and *M.2 NVMe* SSD.

5.1 City Polygons

The implemented model should be able to recognize that it is flying above a city (built-up area). This section discusses if the built-up areas obtained using proposed techniques have met the expectations set by this work and if they are sufficient for use by the implemented model.

In Subsection 3.2.2, it was suggested that precisely (i.e., without any rounding or approximations applied) computed built-up areas would be too difficult to calculate with during the flight. The section also proposed several techniques to approximate the built-up areas to obtain city representations that are more suitable for further computations.

In Table 5.1, approximated and accurate representations of several cities are compared. Each row in the table shows the number of vertices making up the original city polygon and resulting city representations for both accurate and approximate computations. The cities are described by the number of individual built-up areas, the total number of vertices making up those built-up areas and the time required to finish the computation of the given city representation.

The results signify that approximated representations are much reduced in comparison to the accurate results. The representations obtained without any approximations are in many cases even more complex than the original city polygons, and they also took relatively long to compute. The data suggest

that the accurate city representations are not suitable for the model to work with.

City	*Original*	*Accurate result*			*Approximate result*		
	Vertices	Areas	Vertices	Time [s]	Areas	Vertices	Time [s]
Praha	3590	1	12707	27047	1	103	707
Brno	3957	1	9183	1962	3	70	226
Ostrava	2604	1	8369	3791	2	74	154
Plzeň	2012	1	5078	1246	5	60	252
Liberec	2362	1	6382	528	2	40	84
Olomouc	2502	1	5377	247	1	67	78
Ústí nad Labem	3504	1	5480	447	4	119	66
Hradec Králové	1507	1	4300	297	2	94	59
Zlín	2441	3	8184	1990	6	111	56
Most	1621	1	3342	72	1	83	30
Karlovy Vary	1476	1	3454	196	1	52	36
Jihlava	2373	1	6425	1136	6	82	52
Tábor	1673	1	4574	497	1	42	71
Osek	1643	2	1419	14	1	13	10

Table 5.1: City representations

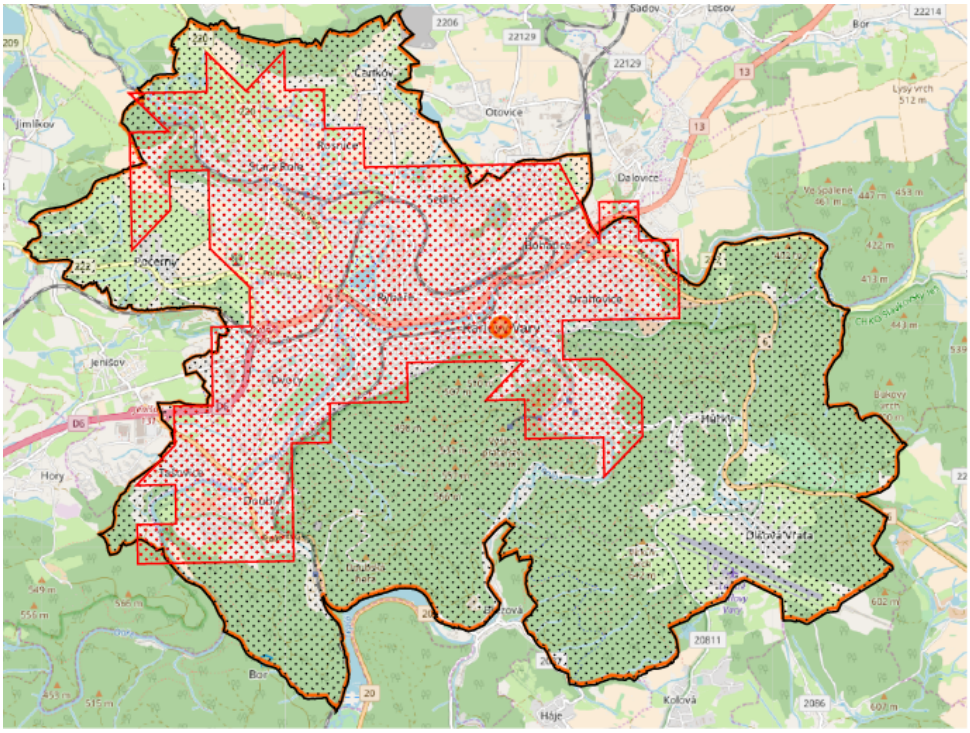
Further examination of the differences between accurately computed and approximated built-up areas is depicted in Figure 5.1. It shows the outputted built-up areas of the city of Karlovy Vary (both accurate and approximate). The built-up areas are displayed with red color. The original outline of the city is also shown (black contour) to better demonstrate performed calculations. It can be seen that applied rounding and simplifications crop the edges of the built-up area (somewhere even quite drastically) but the general area remains the same and center of the city (the densest area) is not omitted from the result.

(a) : Simplified built-up areas **(b) :** Exact built-up areas

Figure 5.1: Built-up areas of Karlovy Vary

To better judge the correctness of the resulting simplified built-up area, in Figure 5.2, the approximation of the built-up area of Karlovy Vary is laid over a screenshot from OpenStreetMap of the entire administrative area of the city. It can be seen that it is a relatively good result. The majority of built-up areas of the city are covered. Compared to the accurate result, the

approximated one excludes the area of the airport. That is not an issue, as airports' airspaces are special airspaces with additional restrictions and these airspaces are already (as was said in Chapter 4) implemented in AgentFly system.



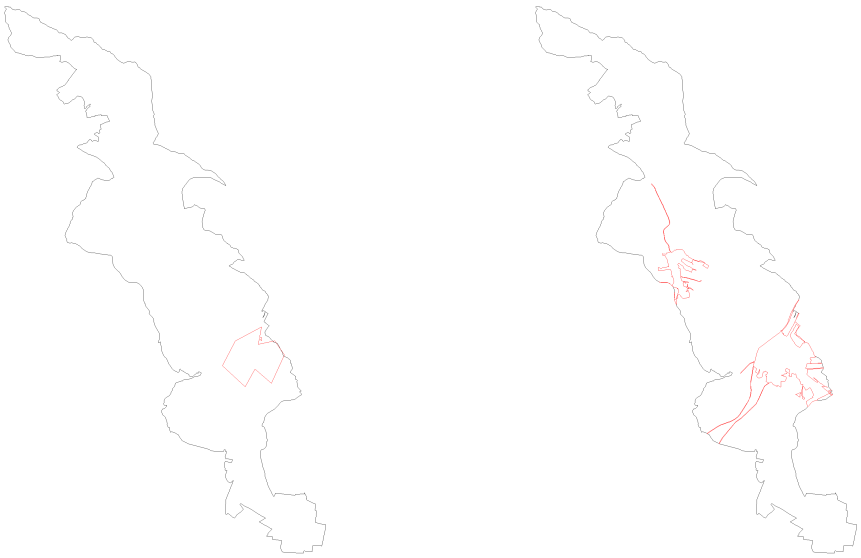
The map capture was taken from [29].

Figure 5.2: Built-up area of Karlovy Vary

Another comparison is provided for the city of Osek in Figure 5.3. Osek is a small town with approximately 5000 inhabitants in Ústecký Region of the Czech Republic. Here, in 5.3b, can be observed that accurate calculation preserved two separate built-up areas. It can be seen from a map that the two built-up areas are in fact the city of Osek and a village Dlouhá Louka (see Figure 5.4) that belongs to the administrative area of Osek. The accurate results contain very long and narrow "lines" - roads that were preserved in polygon subtraction. This undesired property is removed in the rounded results. The approximation also deletes the village Dlouhá Louka altogether. Though this will cause the model to violate Visual Flight Rules when flying over Dlouhá Louka, this also, to a certain degree, simulates the behavior of real pilots. A pilot may easily overlook small villages hidden in forests or mountains. Moreover, for the use of this thesis, it is not necessary for the data regarding built-up city areas to perfectly model the real world.

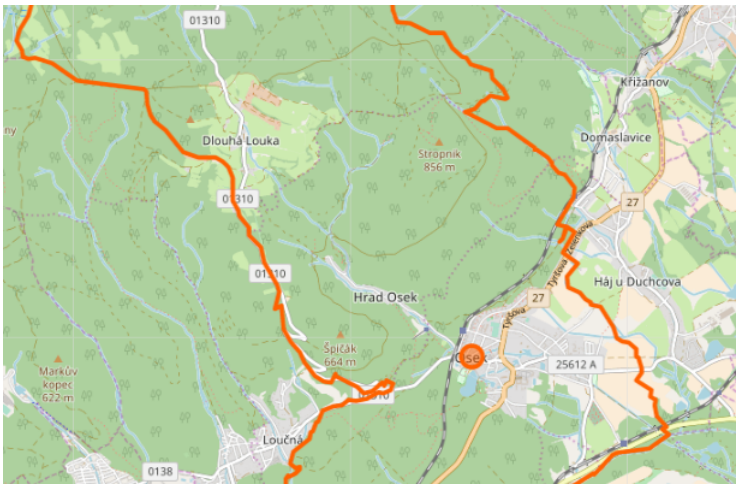
Again, the approximated built-up area is laid over the real map to see how good the result is. This is shown in Figure 5.5. Again, it is a fairly good approximation.

So far, the results seem to be good enough, but sometimes, they crop a



(a) : Simplified built-up areas **(b) :** Exact built-up areas

Figure 5.3: Built-up areas of Osek

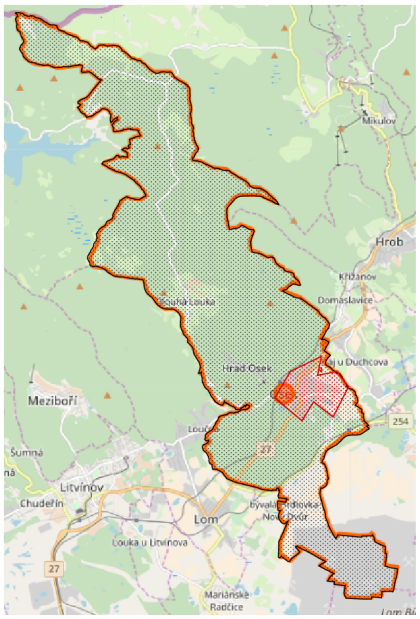


This capture was taken from [29].

Figure 5.4: Map of Osek and Dlouhá Louka

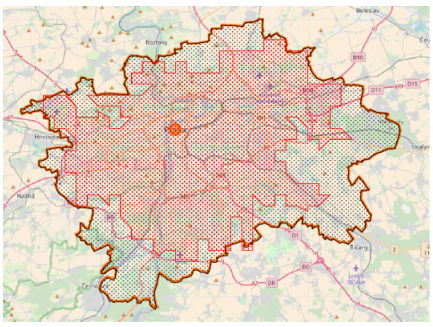
larger portion of built-up area's edges than it would seem to be necessary. The results for other three cities are shown in Figure 5.6 for further comparison. These show that the majority of cropped areas are locations with little building density and thus, the applied rounding and simplifications are well-aimed.

It is true that the polygon rounding and reducing omits some parts of built-up areas. The edges of built-up areas are sometimes cropped, and some smaller built-up areas are removed altogether. However, the resulting polygons seem to be, in general, a good approximation of real-world built-up

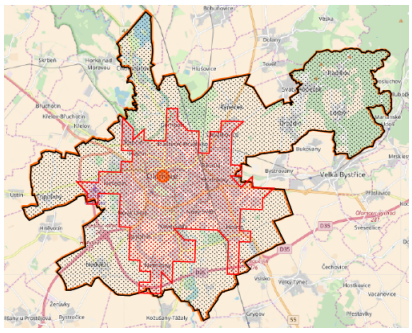


The map capture was taken from [29].

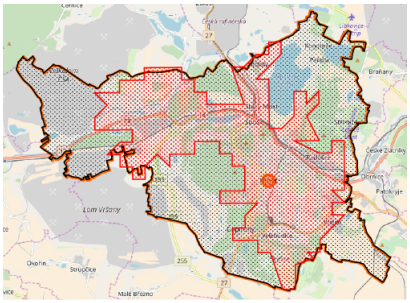
Figure 5.5: Built-up area of Osek



(a) : Built-up area of Prague



(b) : Built-up area of Olomouc



(c) : Built-up area of Most

Captures were taken from [29].

Figure 5.6: Built-up areas of other cities

areas of the cities of the Czech Republic. Moreover, produced polygons are also a good trade-off between geographic accuracy and computationally-

friendly shapes. Thus, the polygons are more than sufficient to be used for further calculations by the implemented model.

5.2 Altitude Changes

Here, the goal of the implemented model is to fly as low as possible without violating the Visual Flight Rules (i.e., fly as low as possible but above the minimum flight altitude). In the scenarios designed to test this property, the altitude of the aircraft was observed at its every position for the entire duration of the simulation. Simultaneously, minimum flight altitudes were computed for all aircrafts' positions as well. The subject of the tests in this section is the comparison of these values.

In the following tests, default sampling settings mentioned in Subsection 4.2.4 were used.

5.2.1 Altitude Test 1

In this scenario, scheduled flight originates in Prague and heads to water dam Lipno. From Lipno, it continues along the Šumava mountain range and Šumava National Park until it turns back to Prague. The scheduled route is displayed in the Figure 5.7.

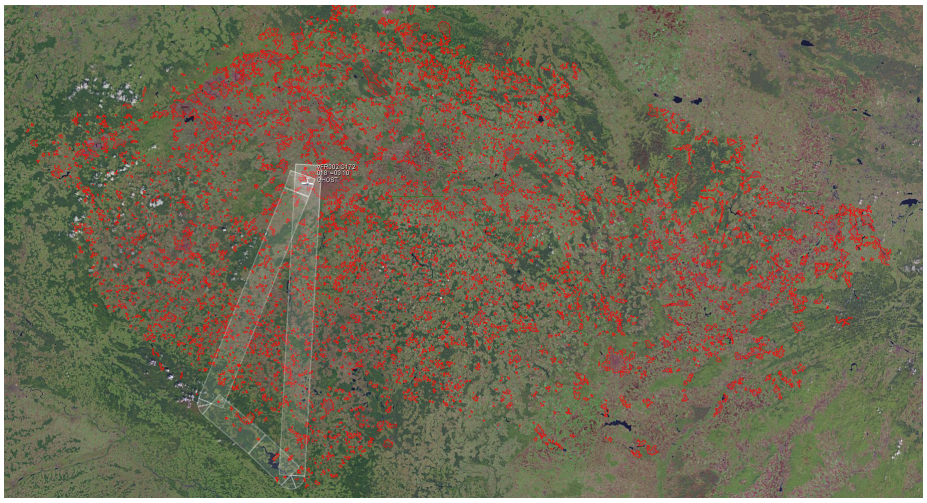


Figure 5.7: Flight route of Altitude test 1

The flight originates in lowlands and gradually moves towards mountains. It flies for a while in parallel with the mountain range before returning to the lowland. The graph comparing aircraft's altitude and minimum flight altitudes for the entire duration of the testing scenario is shown in Figure 5.8.

It can be seen that flight violates the minimum safe altitude only rarely and never for a long time. The occasional violation is caused by the discrete sampling of the height maps. Discrete sampling with a reasonable sampling step will never detect every terrain feature. However, this is not a problem as

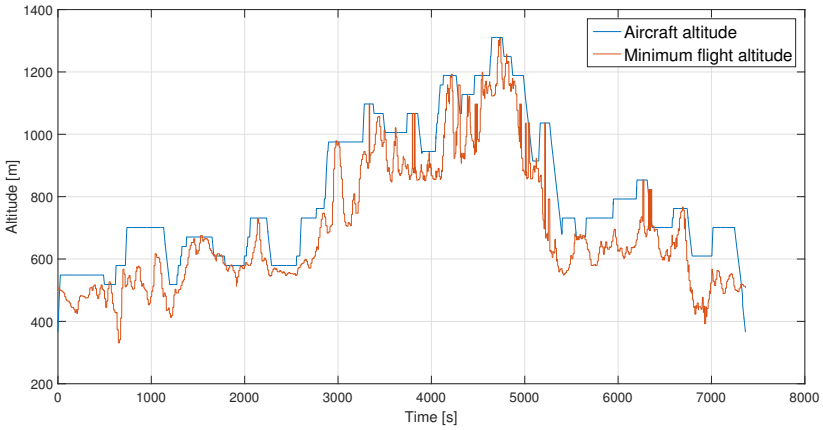


Figure 5.8: Results of Altitude test 1

this conveniently models the behavior of a real-life VFR pilot. The pilot often only guesses the appropriate altitude, at which he or she should currently be. And since they only estimate the actual minimum flight altitude, a few short moments below it are not considered a violation of the Visual Flight Rules. Only gross violations of the minimum flight altitude can be observed at the beginning and the end of the curves. However, that is not a violation of Visual Flight Rules as those describe the take-off and landing when a different set of rules applies.

The graph also shows that sometimes, the situation is misinterpreted and the flight increases its altitude even higher than necessary. That is again caused by the discrete sampling of the terrain elevation, but it is not an issue for the model as it does not violate Visual Flight Rules.

Another property that can be seen in the graph is that the flight does not merely copy the shape of the surface. Sometimes, it remains at a certain altitude for a longer period even though it could descend a little. That is, in fact, a desirable feature.

5.2.2 Altitude Test 2

In this scenario, scheduled flight originates in Karlovy Vary and heads to Prague. From there, it continues towards Krkonoše and its highest peak Sněžka. From Sněžka, the flight heads back to Karlovy Vary. The scheduled route is visualized in Figure 5.9.

The flight first descends to a lowland before climbing to the highest peak of the Czech Republic, from where it returns across the lowland back to where it started. The graph comparing aircraft's altitude and minimum flight altitudes for the entire duration of this scenario is shown in Figure 5.10.

Similar results as in Subsection 5.2.1 can be observed in this case. Not even the steep climb along the Krkonoše mountain range caused the implemented model to violate the minimum flight altitude delimited by the Visual Flight Rules.

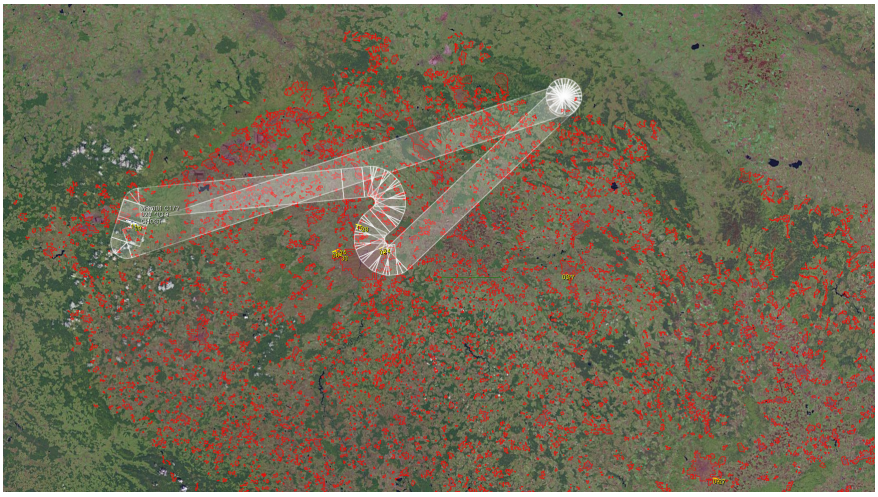


Figure 5.9: Flight route of Altitude test 2

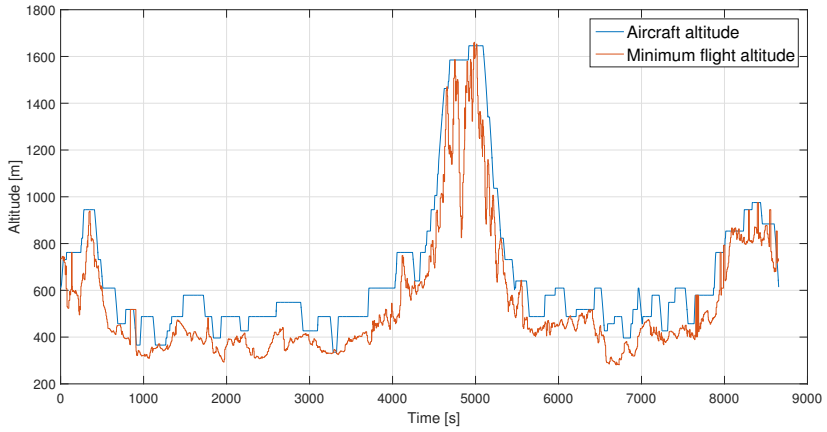


Figure 5.10: Results of Altitude test 2

5.3 Collision Avoidance

The essential part of the model is the ability to detect a conflict with an approaching aircraft and avoid the collision by an evasive maneuver in accordance with Visual Flight Rules.

There are different types of conflicts with other traffic. The expected classification of a conflict is depicted in Figure 2.2.

In the following tests, default sampling settings mentioned in Section 4.3 were used.

5.3.1 Single Conflict

In order to test as many conflict situations as possible, the conflicts were sent towards the aircraft from all positions on a circle around the aircraft with 15 degrees step.

There was always only one conflict at a time (i.e., two airplanes). Each airplane was only given the conflict's GPS position at each simulation step if the conflict was in the aircraft's field of view. The flights did not have access to any additional information. There was also no communication between the two airplanes.

Table 5.2 shows the result of the experiment. Data is taken from the point of view of the airplane in the center of the circle. For each angle on the circle around the aircraft, real type of the conflict is stated, the type as which the conflict was evaluated and the minimum distance (separation) that the airplanes maintained from each other. Some evaluations for overtaking conflicts are missing. In those situations, the airplane did not register the conflict before the conflicting aircraft performed an evasive maneuver on its own (i.e., in those situations pilot did not know at all about the possible danger).

Angle [°]	*Type*	*Evaluation*	*Separation [m]*
0	head-on	head-on	8787
15	head-on	head-on	9049
30	head-on/left	left	4457
45	left	left	4720
60	left	left	4989
75	left	left	6412
90	left	left	6485
105	left	left	5442
120	overtaking	overtaking	4278
135	overtaking	overtaking	4242
150	overtaking	-	3294
165	overtaking	-	2541
180	overtaking	-	6637
195	overtaking	-	3453
210	overtaking	-	3656
225	overtaking	-	5257
240	overtaking	-	4152
255	right	right	5646
270	right	right	5630
285	right	right	5527
300	right	right	4745
315	right	right	3701
330	head-on/right	right	2783
345	head-on	head-on	9227

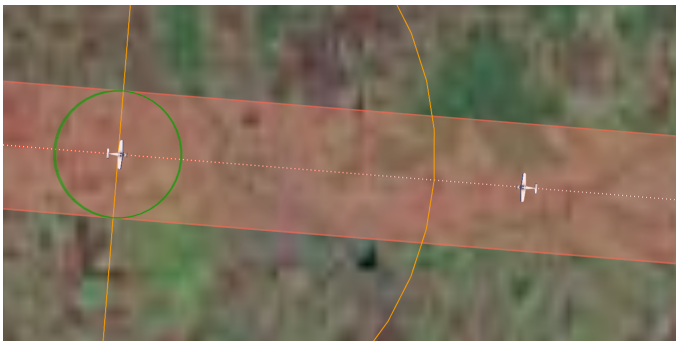
- **left** – Converging conflict. The conflict is incoming from the left side.
- **right** – Converging conflict. The conflict is incoming from the right side.

Table 5.2: Conflict resolution

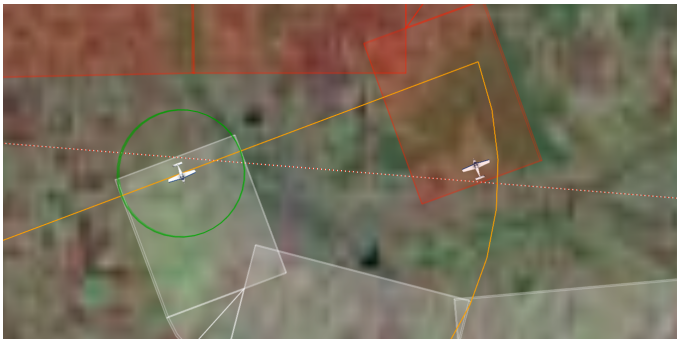
From the table can be seen that all situations were classified correctly and the minimum safe separation set to 2 kilometers was never violated. For the majority of overtaking conflicts, the pilot did not know at all that there was a conflict. That is a natural situation - usually, the airplane that is being overtaken does not know about the overtaking, and it is the responsibility of the other pilot to resolve such conflict.

The table also shows that detection of overtaking conflicts was asymmetrical. That is connected with the different kinds of evasive maneuvers that are performed in each situation.

Resolutions of some of the conflicts are depicted in Figure 5.11 and Figure 5.12. For better demonstration, see enclosed videos described in Section B.1.



(a) : Before conflict resolution



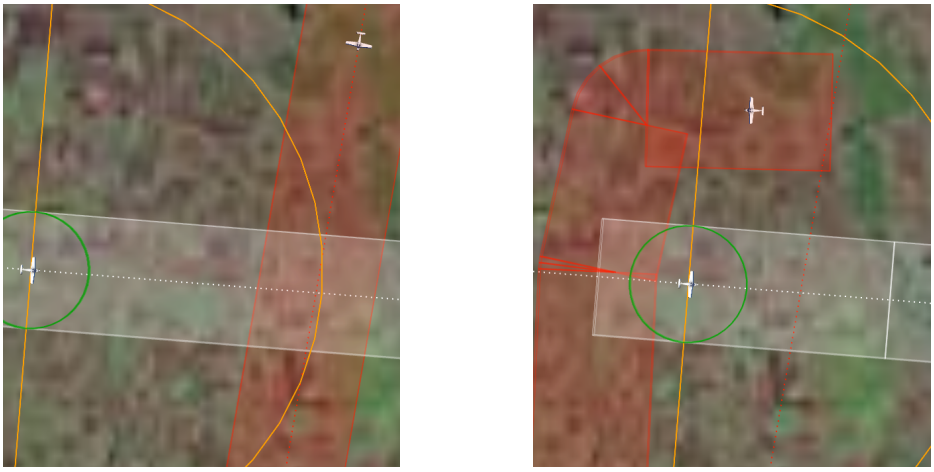
(b) : After conflict resolution

Figure 5.11: Head-on conflict resolution

5.3.2 Multiple Conflicts

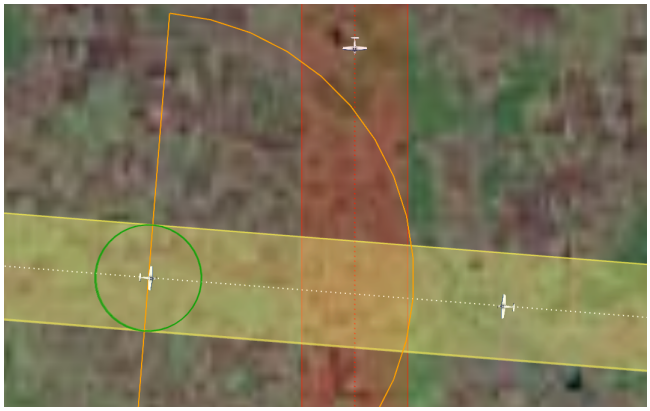
The collision avoidance of the implemented model was proposed and implemented for a resolution of one conflict at a time. However, a simple scenario with two simultaneous conflicts was also tested. The repeated application of the implemented techniques resolved all conflicts and none of the airplanes violated the minimum safe separation.

See Figure 5.13 and enclosed video described in Section B.2 for more detail.

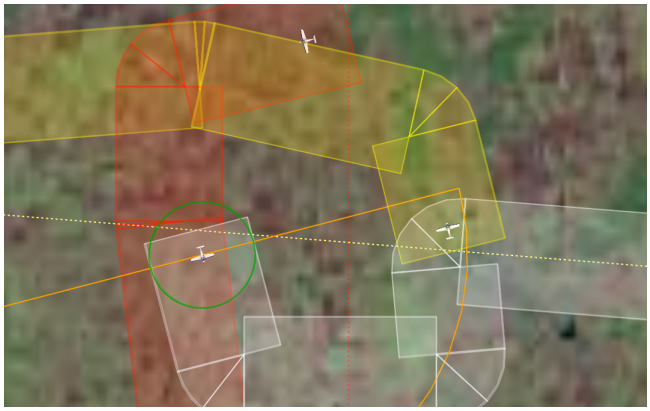


(a) : Before conflict resolution **(b) :** After conflict resolution

Figure 5.12: Converge conflict resolution



(a) : Before conflict resolution



(b) : After conflict resolution

Figure 5.13: Multiple conflicts resolution

5.3.3 Larger Scenario

30 VFR flights were generated for the final test as a small demonstration of fictional VFR traffic over the Czech Republic. All of flights originated and terminated at airports of the Czech Republic. The same altitude was filed for the flights so that the collision avoidance behavior would be engaged more often.

Several conflict situations occurred over the course of the scenario. All of the conflicts were detected and resolved in accordance with Visual Flight Rules. No two aircraft violated the minimum safe separation.

For an overview of the scenario, see the enclosed video described in Section B.3.

Chapter 6
Conclusion

The aim of this thesis is to propose and implement a behavior model of a flight based on Visual Flight Rules. The problem of flying under VFR is studied and inspected.

Problems connected with VFR identified during the studies are presented in the first part of this thesis. Further on, each problem is separately discussed and explored. The discussion also contains proposing a possible solution to the issues. Suggested solutions are implemented in AgentFly system and experimentally validated.

For the VFR flights to maintain minimum flight altitude, a suitable map representation is required. The map is constructed by the combination of height maps of the Earth that are discretely sampled and polygons delimiting city built-up areas that are obtained by processing of data from OpenStreetMap. The downloaded data must be adjusted for the purposes of VFR flights. All adjustments are made in preprocessing making the data ready and optimized for use by the VFR flights after take-off.

The VFR flight also has to correctly detect conflicts and resolve them in accordance with Visual Flight Rules. The detection of conflicts is based on knowing the exact GPS position of the conflict in pilot's field of view. From changes in that position, conflict's trajectory and minimum likely separation between the airplanes, should they stay on their current course, are calculated. If the computed minimum separation violates predefined safe separation, the incoming aircraft is treated as a conflict. The type of conflict as defined by Visual Flight Rules is derived from the approximated trajectory. Afterward, the conflict is resolved by an evasive maneuver in accordance with Visual Flight Rules.

The implementation is tested on several distinct scenarios. All results are satisfactory (see Chapter 5 and Appendix B). Therefore, the goals of this thesis are reached.

6.1 Future Work

Overall results of the experiments conducted on the implemented model are favorable. However, there are still several ways in which the results of this work may be improved.

VFR flights' destinations are currently generated entirely at random. The randomly chosen values are bounded to produce somewhat reasonable flights, but the relation between the generated positions and real-life landmarks is currently ignored. The generation of VFR flights could be improved by using real-life landmarks such as castles, mountains, etc. The real-world data could be possibly obtained from OpenStreetMap using the API implemented for the purposes of this work.

The built-up areas of real-world cities are another issue. The data obtained from OSM must be edited and re-purposed on quite a large scale before they can be used by the model. In Section 3.2 alternative approach to processing the OSM data is proposed, but it is not utilized. The approach could be further explored to compare its complexity and results with the currently used data. Other ways of obtaining the city built-up areas besides OSM could also be inspected in the future.

A considerable simplification of reality was introduced in detecting incoming traffic (conflicts). A better detection system could be proposed. The pilots do not statically stare forward but rather turn their heads which causes them not to see all their surroundings at once. They can be blinded by incoming sunlight, or they may not see everything ahead due to terrain features. The detection system accounting for all of the above could be created to improve the accuracy of the results of this thesis.

Bibliography

[1] Přemysl Volf. *Multiagent Simulation of Air Space and Air Traffic Management*. PhD thesis, Czech Technical University in Prague, January 2013.

[2] Air Navigation Services of the Czech Republic. Air Traffic Services. Online: `http://www.rlp.cz/en/services/our/Pages/default.aspx`. Last accessed: 21.5.2018.

[3] Kurt Colvin, Rahul Dodhia, and R Key Dismukes. Is Pilots' Visual Scanning Adequate to Avoid Mid-Air Collisions? In *Proceedings of the 13th international symposium on aviation psychology*, pages 104–109. Citeseer, 2005.

[4] Federal Aviation Administration. *Instrument Flying Handbook*, 2012. Online: `https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/FAA-H-8083-15B.pdf`.

[5] Amazon. Amazon Prime Air. Online: `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`. Last accessed: 15.4.2018.

[6] Amazon Unveils Futuristic Plan: Delivery by Drone. *CBS News*, December 2013. Online: `https://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/`. Last accessed: 15.4.2018.

[7] Sarah Deener. DRONING ON. *AOPA*, April 2015. Online: `https://www.aopa.org/news-and-media/all-news/2015/april/pilot/f_drones`. Last accessed: 15.4.2018.

[8] Air Navigation Services of the Czech Republic. Visual flight rules. Online: `https://lis.rlp.cz/ais_data/aip/data/valid/e1-2.pdf`, February 2016. Last accessed: 30.4.2018.

[9] Air Navigation Services of the Czech Republic. VFR Manual, Visual Flight Rules. Online: `https://lis.rlp.cz/vfrmanual/actual/enr_2_en.html`, April 2018. Last accessed: 1.5.2018.

[10] International Civil Aviation Organization. *Air Traffic Services, Annex 11, Chapter 2*, July 2001. Online: `http://mid.gov.kz/images/stories/contents/an11_en.pdf`.

[11] Air Navigation Services of the Czech Republic. VFR Manual, Airspace of the Czech republic. Online: `http://lis.rlp.cz/vfrmanual/actual/enr_1_en.html`, April 2018. Last accessed: 1.5.2018.

[12] Tomáš Hnídek. Simulation and Control of Airplanes at Airport Area. Bachelor's thesis, Czech Technical University in Prague, 2014.

[13] National Institute of Standards and Technology. Appendix E. General Tables of Units of Measurement. In *Handbook 133, Checking the Net Contents of Packaged Goods*. U.S. Government Printing Office, January 2011.

[14] David Šišlák, Michal Pěchouček, Přemysl Volf, Dušan Pavlíček, Jiří Samek, Vladimír Mařík, and Paul Losiewicz. Agentfly: Towards multi-agent technology in free flight air traffic control. In *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, pages 73–96. Springer, 2007.

[15] OpenStreetMap. Copyright and License. Online: `https://www.openstreetmap.org/copyright/en`. Last accessed: 30.4.2018.

[16] Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41(10):3224–3236, 2008.

[17] Zhiyuan Lin and Yan Li. An Efficient Algorithm for Intersection, Union and Difference between Two Polygons. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–4. IEEE, 2009.

[18] OpenStreetMap Wiki. About OpenStreetMap. Online: `https://wiki.openstreetmap.org/wiki/About_OpenStreetMap`. Last accessed: 30.4.2018.

[19] OpenStreetMap Wiki. Overpass API. Online: `https://wiki.openstreetmap.org/wiki/Overpass_API`. Last accessed: 15.4.2018.

[20] OpenStreetMap Wiki. Overpass turbo. Online: `https://wiki.openstreetmap.org/wiki/Overpass_turbo`. Last accessed: 15.4.2018.

[21] OpenStreetMap Wiki. Elements. Online: `https://wiki.openstreetmap.org/wiki/Elements`. Last accessed: 15.4.2018.

[22] OpenStreetMap Wiki. Tags. Online: `https://wiki.openstreetmap.org/wiki/Tags`. Last accessed: 15.4.2018.

[23] Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002.

[24] Martin Hložek. Sférická geometrie. Master's thesis, Západočeská univerzita v Plzni, August 2014.

[25] Yan-hua Liu and Wei-qing Chen. Line simplification algorithm implementation and error analysis. In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, volume 2, pages 64–68. IEEE, 2011.

[26] Dan Sunday. Intersections of Set of Segments. Online: `http://geomalgorithms.com/a09-_intersect-3.html`. Last accessed: 16.4.2018.

[27] Zsolt Kocsi. Overpasser. GitHub: `https://github.com/zsoltk/overpasser`. Last accessed: 21.5.2018.

[28] Petr Valenta. Bentley-Ottmann. GitHub: `https://github.com/valenpe7/bentley-ottmann`. Last accessed: 21.5.2018.

[29] OpenStreetMap. Online: `https://www.openstreetmap.org/`. Last accessed: 30.4.2018.

Appendix A

List of Abbreviations

API	.....	Application programming interface
ATC	.....	Air Traffic Control
ATM	.....	Air Traffic Management
ATS	.....	Air Traffic Service
BADA	.....	Base of Aircraft Data
BO	.....	Bentley-Ottmann Algorithm
BST	.....	Binary Search Tree
EUROCONTROL		European Organization for the Safety of Air Navigation
FAA	.....	Federal Aviation Administration
FIS	.....	Flight Information Service
FOV	.....	Field of View
GIS	.....	Geographical Information System
HTTP	.....	Hypertext Transfer Protocol
ICAO	.....	International Civil Aviation Organization
IFR	.....	Instrument Flight Rules
OSM	.....	OpenStreetMap
RDP	.....	Ramer Douglas Peucker Algorithm
RLP	.....	Air Navigation Services of the Czech Republic
SG	.....	Stereographic [Projection]
UAV	.....	Unmanned Aerial Vehicle
VFR	.....	Visual Flight Rules
XML	.....	eXtensible Markup Language

Appendix B
Videos

This appendix describes videos enclosed to the thesis. The videos were made to better demonstrate the results of this work. They were all recorded using the AgentFly visualization system.

B.1 Single Conflict

There are five videos in total dedicated to the standard conflict situations. They are listed in Table B.1 and they are all located in `videos/2_planes` directory on the enclosed CD.

File	Description
`0_deg.avi`	The airplane and the conflict are approaching head-on. They both alter their course to the right.
`75_deg.avi`	The conflict is coming up on the left side of the airplane. The airplane has right-of-way and the conflict avoids the airplane by altering its course to the right.
`180_deg.avi`	The conflict is coming on the airplane from behind in parallel direction. The conflict overtakes the airplane by altering its course to the right.
`225_deg.avi`	The conflict is coming on the airplane from behind in non parallel direction. The conflict alters its course and passes the airplane in parallel direction. Once it is well ahead of the airplane, it turns back to its original destination.
`285_deg.avi`	The conflict is coming up on the right side of the airplane. The conflict has right-of-way and the airplane avoids the conflict by altering its course to the right.

Table B.1: Videos of standard conflicts

B.2 Multiple Conflicts

There is one video depicting three airplanes on a collision course with each other. Two airplanes are approaching head-on. The third is incoming from a

side. Gradually, all airplanes alter their course to the right.

The video file is `3_planes.avi` and it can be found in the `videos` directory on the enclosed CD.

B.3 Larger Scenario

The last video shows a fictional VFR traffic over the Czech Republic within five hours interval with 30 VFR flights in total. Multiple conflict situations (along with their resolutions) can be observed over the course of the scenario.

The video file is `traffic_demo.avi` and it can be found in `videos` directory on the enclosed CD.

Appendix C

Contents of Enclosed CD