ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

Collision Detection Algorithm for a Practical Airborne Collision Avoidance System

Bachelor's Thesis

Patrick Oberlin
poberlin@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Peter Belcák
Prof. Dr. Roger Wattenhofer

March 28, 2022

Acknowledgements

First of all, I thank my supervisor Peter Belcák for his support and guidance throughout the thesis. The topic combines two fields I am quite fond of, and I thank Prof. Dr. Wattenhofer for offering such an interesting project to work on. Furthermore, I thank my friends, colleagues and family for their support, for proofreading the thesis, and for their patience when I started talking about the project on every occasion.

Abstract

Good weather facilitates light and sporting aircraft (LSA) activity. LSA pilots have natural incentives to fly their planes inside areas of lift, aiming to remain airborne for as long as possible. The more aircraft engage in this activity, the more difficult it is for a pilot to keep track of the aircraft in their vicinity. Monitoring relative positions of aircraft close by is, however, crucial to collision prevention. The objective of this thesis is to provide an automated solution for tracking planes to reduce pilot workload and detect potential collisions early on.

This thesis proposes a trajectory prediction algorithm based on positional and sensoric data, and a collision detection algorithm foreseeing possible collision courses in order to warn pilots of imminent danger. The latter algorithm further performs on-line detection of properties and whereabouts of thermic streams to improve its predictions. The algorithms are complemented by an example Python implementation, and a graphical simulation environment to showcase their behaviour in example scenarios.

Contents

Acknowledgements **i**

Abstract **ii**

1 Introduction **1**

2 Trajectory Prediction **2**

2.1 Algorithm 2

2.1.1 What a prediction looks like 2

2.1.2 Structure 3

2.2 General decision process 4

2.2.1 Methods used 7

2.3 Parameters 11

2.3.1 User Inputs 11

2.3.2 Other Parameters 12

3 Thermal and Wind detection **14**

3.1 Thermal model and its limits 14

3.2 Detection and handling 14

3.3 Wind 16

3.3.1 Improvement ideas 17

4 Interaction with other aircraft **18**

4.1 Communication 18

4.2 Collision Detection 19

5 The Simulation **21**

6 Analysis **24**
6.1 Time Analysis 24
6.2 Limits of the Algorithm 26
6.3 Conclusion 26

7 Further Work **27**
7.1 Future Ideas 27

Bibliography **28**

Chapter 1

Introduction

When several aircraft are flying in a confined space, as for example a thermal poses, the risk of a collision increases with the number of planes. The pilot of each aircraft has to notice and keep track of every aircraft that poses a potential threat. As long as there are only a few aircraft around, it is easy enough to keep some distance between them to reduce the risk of a crash. But when the airspace becomes more crowded, the intermediary distances will decrease, making sudden changes of course more impactful. In addition, it becomes way harder for a pilot to detect possible collision courses.

The aim of this thesis is to address the abovementioned emergent difficulties. Though a pilot should never cease paying attention to their surroundings, the algorithm aims at detecting dangers in case one is still missed. This is achieved by tracking an aircraft's flight using GPS data and estimating possible future trajectories it might take, then exchanging and comparing these trajectories with the other aircraft to find potential crash courses. Since the problem particularly arises in proximity to thermals, thermal detection is a core element that sets this algorithm apart from established technologies, such as FLARM [1]. Thermals are recognized in-flight and transmitted on-line between aircraft to maximize the accuracy of the generated predictions. Capitalizing on this advantage, potential collisions can be anticipated with higher reliability.

The thesis is part of a larger project targeting the mentioned problem. The final goal of the latter is to develop a device pilots can take with them implementing the described behaviour. Since this is the first thesis within the project, the algorithm could not be tested in its envisaged environment. Counteracting this issue, a simulation was created that is able to schematically visualize the past and future flight paths as well as thermals. It provides a user interface (UI) allowing to add and remove aircraft and thermals at runtime, and an API that allows to create custom scenarios, e.g., to test new features. Since it is written in Python, new ideas can be implemented fairly quickly.

Chapter 2

Trajectory Prediction

2.1 Algorithm

General Hints

To forego the need to specify the meaning of the following terms on every occurrence in the thesis as well as in the code, here is how they will be used in the thesis:

- Heading: Denotes a GPS direction in degrees, thus $\in [0, 360)$.
- Bearing: Denotes a GPS direction in radian, thus $\in [0, 2\pi)$.
- Direction: Refers to a normalized, 3-dimensional vector indicating the x, y, and z component of the aircraft's movement. Multiplying direction with *speed* yields the velocity vector, where *speed* is the current speed of the aircraft.

2.1.1 What a prediction looks like

A prediction is represented in one of two ways. The first is how one's own prediction is stored on the device. It can be directly used in the collision detection algorithm. The prediction is stored as a chronologically sorted list of future positions, each having a longitude, a latitude, an altitude, and a timestamp. The interval between these positions is determined by dividing the size of the aircraft by its current speed, leading to a parameter we shall call Δt. The size of the aircraft is provided by the user entering it into his device. This calculation leads to spheres that overlap each other by exactly half their radius (see **Figure 2.1**). The resulting positions can be easily compared to other aircraft's. If the distance between them is too small, a collision will happen in case both aircraft follow the predicted trajectory.

However, the list stores a lot of data, which is bound to cause problems when communicating our prediction to the aircraft in the vicinity. Thus, a more

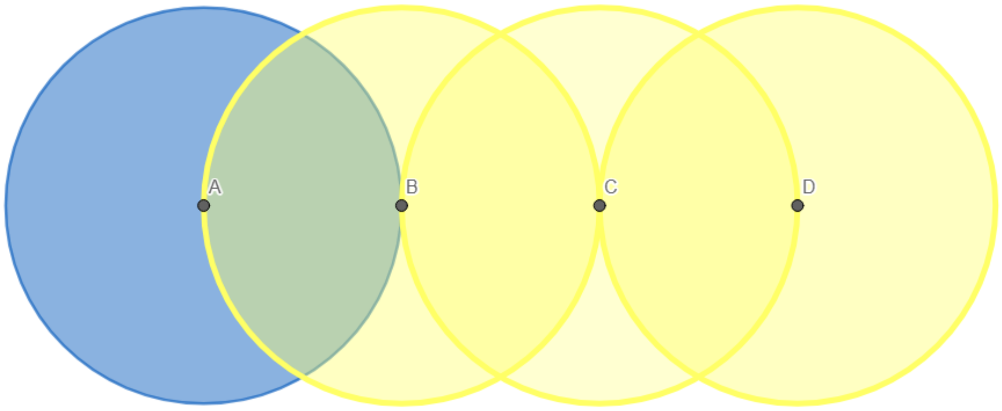

Figure 2.1: Schematic figure of the collision spheres. The blue sphere represents the aircraft, the yellow spheres are the spheres indicating the predicted future positions with a time step of Δt. Δt is always chosen such that the spheres overlap each other. This allows for comprehensive collision detection while not needing a large number of spheres.

compact form is necessary. The second way trajectories are saved is as a set of parameters fully describing the trajectory. These parameters are the following:

Position	The current position
Direction	The direction the aircraft is currently flying
Speed	The current speed of the aircraft
Size	The radius of the aircraft's collision sphere
Type	The type of the trajectory ("straight" or "left/right turn")
Rate of turn	The rate of turn if the trajectory type is a turn, else null

From these parameters, once exchanged, an aircraft is capable of calculating the list of future positions of the other aircraft in the area, which then can be used to detect looming collisions.

2.1.2 Structure

The core element of the algorithm is a class called `TrajectoryPredictor`. This class only needs to be fed the GPS data, and the prediction and collision detection function can be called when there is CPU time available. Optimally this is done once per second to keep the predictions accurate. The predictor manages a submodule called `ThermalTracker`, which recognizes and organizes thermals. It is used by the predictor to get the thermal strength at a given position. In principle, the `ThermalTracker` does not have to be separated from the `TrajectoryPredictor`, but doing so increases clarity. The tracker keeps a list of all known thermals (recognized by itself as well as learnt from other aircraft).

A class named GPS was created to provide functionality with regard to the positions. A GPS instance has a latitude and a longitude, both of which are stored in *rad*, an altitude in *meter*, and a timestamp. A GPS object can either be generated from a 4-tuple consisting of the four values, or from a line of an `.igc` file, which is a common format to store flight track data. The class is equipped with methods to add two positions, which is useful when generating the list of future positions, find the distance between two locations, and get the bearing when looking from one position to another. To find the distance between two GPS coordinates, an equirectangular approximation is used, following the formula from `Movable Type Scripts` [2]

2.2 General decision process

In this section, the decision process of the `TrajectoryPredictor` is explained. This process analyzes the GPS data and decides which trajectories are likely future flight paths. These paths are stored so that they can be broadcast and compared to other trajectories.

Variables used in the algorithm:

$\mathcal{L}$	Array-like object that stores all likely trajectories
$\mathcal{P}$	Array-like object that contains the last 15 positions
$\mathcal{P}'$	Array-like object containing the last 3 positions
$\mathfrak{p}$	The latest position received from the GPS module (i.e., the current position)

In **line 1** of **Algorithm 1**, the parameters *speed* and Δt are adjusted so that all following calculations are done with the latest data. In particular, the speed is recalculated as the average speed over the last 10 seconds, capped at 10 positions.

The function LINAPPROX in **line 3** will be explained in more detail below. It calculates a linear approximation of the points $\mathcal{P}$ and their mean squared error (MSE). The error is then compared in **line 4** against a threshold $\sigma = 1m$. If the error is smaller, the straight trajectory is computed and stored to be used in the collision detection. Using a fixed threshold value is only a first approach. To improve the decision making of the algorithm, an adaptive value can be used that is continuously improved during or after a flight. This would also allow to personalize the parameter to the style of the pilot.

After adding the line prediction, the algorithm checks if the aircraft has flown through a thermal during the positions in $\mathcal{P}$ (**line 7**). If this is the case, the prediction will not be accurate if the thermal ended within $\mathcal{P}$. To account for the case where this did happen, an alternative vertical speed is calculated by

Algorithm 1: Function PredictTrajectory

```
UpdateSpeed();
initialize $\mathcal{L}, \mathcal{P}, \mathcal{P}', \mathfrak{p}$;
$line, MSE \leftarrow$ LinApprox($\mathcal{P}$);
if $MSE \leq \sigma$ then
    $\mathfrak{L} \leftarrow$ GetPrediction($line$);
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{L}$;
    if flown through thermal then
        $\mathfrak{L}' \leftarrow$ adjust $\mathfrak{L}$;
        $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{L}'$;
    if distance($line, \mathfrak{p}$) $\geq \tau$ then
        if last positions allow a turn then
            $\mathfrak{T} \leftarrow$ PredictTurn();
            $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{T}$;
            if flown through thermal then
                $\mathfrak{T}' \leftarrow$ adjust $\mathfrak{T}$;
                $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{T}'$;
else
    $\mathfrak{T} \leftarrow$ PredictTurn();
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{T}$;
    if flown through thermal then
        $\mathfrak{T}' \leftarrow$ adjust $\mathfrak{T}$;
        $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{T}'$;
$\mathfrak{L}_{short} \leftarrow$ LinApprox($\mathcal{P}'$);
if $\frac{\partial^2 z}{\partial t^2} \neq 0$ or a turn prediction was made then
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{L}_{short}$;
return $\mathcal{L}$
```

subtracting the average height gain due to thermals from the total height gain in the considered time period. Using this new value, another prediction is computed and added to $\mathcal{L}$.

To conclude whether a turn is possible as well, two conditions are checked (**lines 10 to 11**). The first one is that the current position needs to be off in comparison with the line that was calculated. The line's fixpoint is the mean point of $\mathcal{P}$, thus the direction vector will, if the aircraft is turning, not point exactly towards the current position, but rather beside it. If this distance is larger than $\tau = 3m$, a turn is not considered unlikely and the second condition is checked. More specifically, the 3 latest rates of turn are calculated and compared. In order for the second condition to pass, these rates either have to be of the same sign, or the ones that are of opposite sign than the latest have to be smaller than $0.05rad/s \approx 2.8°/s$. Both the $3m$ and $0.05rad/s$ thresholds need to be tested and improved from real-life data when the first prototype is built. If now both conditions are met, a turn prediction is computed, and again an alternative turn is added if the aircraft recently flew through a thermal.

However, if the line approach is not likely, i.e., the condition in **line 4** evaluates to false, a turn prediction is added in any case. This ensures that always at least one prediction is made. The procedure then is the same as in **lines 12 to 16**.

Independently of the decisions above, a linear approximation for the last three positions is made in the end (**line 23**). If there is a change in the vertical speed due to a change in the aircraft's pitch angle within $\mathcal{P}$ but not $\mathcal{P}'$, the line prediction $\mathfrak{L}$ will not be accurate because it will include the positions before the pitch change until they are not contained in $\mathcal{P}$ anymore. For example, if the aircraft has just leveled off, $\mathfrak{L}$ will predict a decreasing climb rate over the next 15 seconds even though the aircraft is not climbing anymore. Since the shorter prediction $\mathfrak{L}'$ stops considering the thermal after three new positions, it will be the more accurate forecast during approximately 12 seconds. There is little gain in adding another adjusted prediction for the thermal case as before since the short line will adapt to that very quickly anyways.

During many manoeuvres, both $\mathfrak{L}$ and $\mathfrak{L}'$ will be computed and added to the list $\mathcal{L}$, which is intended. $\mathfrak{L}$ will be more accurate and stable, e.g., during turbulence, while $\mathfrak{L}'$ will be quick to adapt to a turn, ascent, or descent that is being initiated or ended.

2.2.1 Methods used

Find Linear Approximation

Variables used in the algorithm:

$\mathcal{P}$ Array-like object that contains the last 15 positions

$\overrightarrow{\mathcal{D}}$ The direction the aircraft is flying

Algorithm 2: Function LinApprox

```
  Input: $\mathcal{P}$
1 $\overline{\mathcal{P}} \leftarrow$ MEAN($\mathcal{P}$);
2 $\overrightarrow{\mathcal{D}} \leftarrow$ SVD($\mathcal{P} - \overline{\mathcal{P}}$);
3 if $\overrightarrow{\mathcal{D}}$ is badly off then
4     $\overrightarrow{\mathcal{D}} \leftarrow \overrightarrow{\mathcal{D}} * (-1)$;
5 $MSE \leftarrow$ ComputeActualmse;
6 $line \leftarrow (\overline{\mathcal{P}}, \overrightarrow{\mathcal{D}})$;
7 return line, MSE
```

The singular value decomposition (SVD) returns the direction of the linear/first order polynomial approximation calculated from the input positions $\mathcal{P}$, minimizing the mean squared error (MSE). Since the singular value decomposition only fits a line over the data and does not care about the timely order of the positions, the returned values might point exactly in the opposite direction of what one would expect. In **lines 3 to 4**, it is checked whether the difference between the approximation $\mathcal{D}$ and the direction calculated from the last two positions is larger than $135°$. If so, the direction vector is turned by $180°$. The threshold of $135°$ will only lead to mistakes when an aircraft has turned more than $270°$ since the last time a GPS position was saved (see **Theorem 2.2**). The interval between two positions is aimed to be 1s, which means it is close to impossible to fulfil this condition. If a single position is lost somehow (not just delayed by less than $1s$), errors might occur during extreme manoeuvres, but even then not for extended periods of time.

The mean squared error is then calculated so that it can be used for comparison in **Algorithm 1**. The line that is returned is represented by a point on the line, here the mean point $\overline{\mathcal{P}}$ from **line 2**, and its direction $\overrightarrow{\mathcal{D}}$.

Get Prediction

The Algorithm GetPrediction returns a list of future positions that are computed from the *line* that was computed by LinApprox. The computation is

Algorithm 3: Function GetPrediction

```
    Input: line
1   $\overrightarrow{\mathcal{V}} \leftarrow speed \cdot \Delta t \cdot \overrightarrow{\mathcal{D}}$;
2   $\mathfrak{s} \leftarrow$ GPS($\overrightarrow{\mathcal{V}}, \Delta t$) ;
3   $\mathcal{L}_1 \leftarrow$ compute future positions from $\mathfrak{s}$;
4   if $\mathcal{L}_1$ goes through a thermal then
5       $\mathcal{L}_2 \leftarrow$ amend $\mathcal{L}_1$;
6       return $\{\mathcal{L}_1, \mathcal{L}_2\}$;
7   else
8       return $\{\mathcal{L}_1\}$;
```

implemented using a *step* variable $\mathfrak{s}$ that is of type GPS (see **Section 2.1.2**). In **line 1**, a vector $\overrightarrow{\mathcal{V}}$ is calculated such that it contains the distance the aircraft will move during Δt in each dimension. From $\overrightarrow{\mathcal{V}}$, the step $\mathfrak{s}$ can be initialized as a GPS instance (**line 2**). Further, the timestamp of $\mathfrak{s}$ will be Δt in order for the following procedure to work. The list of positions can be generated from the current position $\mathfrak{p}$ by iteratively adding $\mathfrak{s}$ and storing the resulting position (**line 3**). This procedure makes use of the adding functionality of the GPS class mentioned in **Section 2.1.2**, which sums all four components of the given positions.

If any position in $\mathcal{L}_1$ is inside a thermal, a second trajectory $\mathcal{L}_2$ is computed in **line 5** by adding any thermic impact on the afflicted positions. Whenever the trajectory does lead through a known thermal, this computation yields a prediction accounting for it, and one not accounting for it. This is important because thermals are not constant in time, but oftentimes bubbles that detach themselves from the floor at intervals.

Predict Turn

The turn prediction builds upon the following two common geometric theorems (see **Figure 2.2**):

Theorem 2.1 (Central Angle Theorem)**.** *The inscribed angle β of a circle is twice the circumferential angle α.*

Proof. Consider the special case where $\overline{CD}$ is exactly the diameter. Then, the triangle ABD is isosceles, and thus $\angle ADB$ and $\angle ABD$ are equal. The sum of the interior angles in $\triangle ABD$ is then

$$\alpha + \alpha + (180^\circ - \beta) = 180^\circ$$
$$\Leftrightarrow 2 \cdot \alpha = \beta.$$

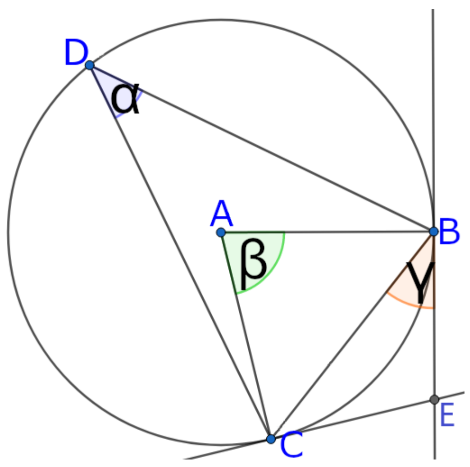

Figure 2.2: Outline of the angles for **Theorems 2.1 to 2.2**

The case depicted in **Figure 2.2** can then be interpreted as two times the special case by dividing the triangles with a diameter through D and A. Since both parts fulfill the condition, also the sum does. Thus,

$$2 \cdot \alpha = \beta$$

□

Theorem 2.2 (Chord Tangent Angle Theorem)**.** *The chord tangent angle γ is equal to the circumferential angle α.*

Proof. The triangle ABC is isosceles, thus

$$\angle ABC = \frac{180^\circ - \beta}{2} = \frac{180^\circ - 2\alpha}{2} = 90^\circ - \alpha.$$

Using $\angle ABC + \gamma = 90^\circ$, it follows:

$$\gamma = 90^\circ - \angle ABC = 90^\circ - (90^\circ - \alpha) = \alpha$$

□

Remark 2.3. Both theorems are valid independently of the locations of B, C, D on the circle.

Variables used in the algorithm:

$\mathfrak{p}'$ The second latest position (i.e., the one right before $\mathfrak{p}$)

$\mathfrak{t}$ The rate of turn

$\mathfrak{b}$ The current bearing

ρ: The radius of the turn

$\mathfrak{c}$ Chord length, i.e., the length of the straight line between two consecutive points if the aircraft flies on the arc of the circle.

Algorithm 4: Function PredictTurn

```
1  $\mathfrak{t} \leftarrow$ getRateOfTurn() ;
2  $\mathfrak{d} \leftarrow$ getBearing($\mathfrak{p}' \rightarrow \mathfrak{p}$) ;
3  $\mathfrak{b} \leftarrow \mathfrak{d} + \frac{\mathfrak{t}}{2}$ ;
4  $\mathfrak{c} \leftarrow \frac{\text{distance}(\mathfrak{p},\mathfrak{p}')}{\Delta t_{\mathfrak{p}\mathfrak{p}'}} \cdot \frac{\sin\left(\frac{\mathfrak{t}\cdot\Delta t}{2}\right)}{\sin\left(\frac{\mathfrak{t}}{2}\right)}$;
5  repeat
6      create $\mathfrak{s}$;
7      $\mathfrak{p}_{new} \leftarrow \mathfrak{p}_{old} + \mathfrak{s}$;
8      $\mathfrak{T}_1 \leftarrow \mathfrak{p}_{new}$;
9      $\mathfrak{b} \leftarrow \mathfrak{b} + \mathfrak{t} \cdot \Delta t$;
10     $\mathfrak{p}_{old} \leftarrow \mathfrak{p}_{new}$;
11 until enough points are calculated;
12 if $\mathfrak{T}_1$ goes through a thermal then
13     $\mathfrak{T}_2 \leftarrow$ amend $\mathfrak{T}_1$;
14     return $\{\mathfrak{T}_1, \mathfrak{T}_2\}$;
15 else
16     return $\{\mathfrak{T}_1\}$
```

In **Algorithm 4**, a list of positions is returned assuming the aircraft is turning at a constant rate (the non-constant turn has not been addressed yet). The general principle is the same as in `GetPrediction`: create a step $\mathfrak{s}$ and add it to the current position iteratively. This time, though, $\mathfrak{s}$ needs to be adjusted after every step to create a curve. In order to do that, the parameters in **lines 1 to 4** are introduced.

First of all, the rate of turn $\mathfrak{t}$ is calculated. It is averaged over the last ten positions to gain some stability but stay adaptive at the same time. The rate of turn can be computed from the difference in bearing between three positions using **Theorems 2.1 to 2.2** and dividing it by the time difference ($\angle CDB$ corresponds to exactly half the angle the aircraft has turned during moving from B to C).

The bearing $\mathfrak{d}$ is the bearing when looking from $\mathfrak{p}'$ towards $\mathfrak{p}$ (B to C). This is decidedly not the same as the current bearing, as can be seen in **Figure 2.2**. Specifically, $\mathfrak{d}$ is the bearing from B to C while the actual bearing $\mathfrak{b}$ of the aircraft at C is along $\overline{EC}$, the tangent of the circle. To actually find the current bearing of the aircraft, half the rate of turn has to be added, although multiplied by the time difference between $\mathfrak{p}'$ and $\mathfrak{p}$. The time difference is usually $1s$, but might be larger due to device failures or the CPU coping with a huge workload.

Finding the chord length between the current and the next position can be divided into three steps:

$$d = \frac{\textsc{distance}(\mathfrak{p}, \mathfrak{p}')}{\Delta t_{\mathfrak{p}\mathfrak{p}'}} \tag{2.1}$$

$$\rho = \frac{d}{2 \cdot \sin\left(\frac{\mathfrak{t}}{2}\right)} \tag{2.2}$$

$$\mathfrak{c} = 2 \cdot \rho \cdot \sin\left(\frac{\mathfrak{t} \cdot \Delta t}{2}\right). \tag{2.3}$$

In these steps, d denotes the chord length that is traveled between $\mathfrak{p}'$ and $\mathfrak{p}$, normed to $1s$. Since $\mathfrak{t} \cdot 1s = \beta$ (**Figure 2.2**) due to the two theorems, the formula for the radius ρ can be used by substituting β with $\mathfrak{t}$. The chord length then follows from the third equation, but since the future positions will usually be $\neq 1s$, the factor Δt is needed to scale the angle the plane actually turns. For small angles, i.e., low rates of turn or high speeds (then Δt will be small), the small angle approximation could be used, leaving only the Δt as a factor, but this does not work well in the general case. The equation in **line 4** is obtained by inserting (2.1) into (2.2), and then (2.2) into (2.3).

From these variables, the step $\mathfrak{s}$ can be generated, using

$$\begin{aligned} \Delta longitude &= \mathfrak{c} \cdot \sin(\mathfrak{b}) \\ \Delta latitude &= \mathfrak{c} \cdot \cos(\mathfrak{b}). \end{aligned}$$

If DISTANCE returns a value in m, $\Delta latitude$ and $\Delta longitude$ have to be divided by earth's radius to convert them to *rad*. In the python code, this is the case, but in an embedded system, distances should be kept in *rad* at all times to save computation time and to prevent numerical errors stemming from repeated conversion. As step size in altitude, the linear approximation value in z-direction is used. $\mathfrak{s}$ can then be used the same way as in **Algorithm 3**, but it has to be recreated with the adjusted bearing (**line 9**) after every iteration because of the change in direction.

If the prediction passes through a thermal, an adjusted prediction is computed in **Line 13** in the same manner as in **Algorithm 3**.

2.3 Parameters

2.3.1 User Inputs

In the algorithms above, many parameters were used. At the current state, the size of the aircraft is the most crucial one to be provided by the user. Whether or not to add a safety factor onto it and how large it should be is to be decided primarily from real-life testing data and experience, but the user might be able to advocate for a safer value or for one that keeps false alerts to a minimum. Another parameter that a pilot could decide, at least to some degree, is how far the `predictor` should look into the future. A choice between, e.g., 30, 45, or 60 seconds could be given. The pilot could then choose one according to his or her preferences, flight style (for a very acrobatic style, 60s will not be worth the computation time), and traffic density to keep the prediction running smoothly.

The type of the aircraft could also be put to good use, especially to get a best possible first estimate of the glide ratio of the plane, which is an essential parameter for thermal detection (**Section 3.2**). For example, sail planes usually have a ratio around 40 while paragliders' ratio is about eight to ten. Alternatively, the glide ratio could be provided by the user if an approximate value is known. Further use of the type could be to change to a different mode for skydivers (see **Section 7.1**).

2.3.2 Other Parameters

Several further parameters have to be set programmatically. One group consists of the ones that indicate how many positions or how much past time should be taken into account for the prediction or a decision.

- How many future points will be computed for the prediction. This depends on the size and speed of the aircraft. This parameter is recalculated at the beginning of every prediction as seen in **Algorithm 1, line 1**.
- How many past positions are considered to calculate the speed and Δt in updateSpeed. The initial value is 10.
- Time to calculate the speed. This parameter indicates how far back the method updateSpeed should consider positions. Its current value is $10s$. The value that is reached first between these two parameters is pivotal.
- The number of points that are taken into account when calculating the linear approximation. This refers to the first, longer approximation that is made, not the shorter one. The latter looks at 3 positions to gain maximal flexibility. The former, on the other hand, considers 15 past points.
- How many points have to be indicating a turn in the same direction to allow a turn. This is explained in more detail in the explanation to **Algorithm 1, Line 11**. To make the algorithm adaptive, it is set to 3, which is the minimal value so that the calculations can be performed at all.
- How many points are considered to calculate the average rate of turn. The use of this parameter is explained in **Algorithm 4**. At this state, 10 positions are considered. Taking into account more positions would increase

stability during calm flights while less would render the turn prediction more sensitive to quick changes (e.g., a slalom course).

The second group are the threshold values.

- Threshold for accepting the straight line approximation (σ). It is compared to the MSE and is set to $1m$, which might seem like quite a low bar. But if σ is not met, the more adaptive prediction $\mathfrak{L}_{short}$ is computed for sure, guaranteeing that at least one straight prediction is made.
- The threshold $\tau = 3m$ for the current position $\mathfrak{p}$, against which $\mathfrak{p}$ is tested whether it is too close to make a turn credible.
- A threshold for $|\frac{\partial^2 z}{\partial t^2}|$, deciding if a change in vertical speed might have occurred recently.

Apart from the very first of these parameters, the values are chosen only from simulation experience, which is likely to be biased. They need some thorough testing and adjusting before the system can actually be deployed. To find suitable values, a statistical approach could be used that keeps the information on which predictions were made. Later on, this data can be used to check if the actual manoeuvre that has just been flown was one of the predictions, and accordingly adjust the values.

The values could also be found using machine learning. This would cause a demand for vast amounts of data, which could be taken from websites like `xcontest` [3].

Some form of learning on the device itself would be beneficial as well. Then, the device could continue learning from real data and the thresholds can be personalized to the pilot's flight style. Different profiles can be made for different scenarios (in a competition, the flight style might differ from the casual Sunday afternoon flight), multiple pilots using the same plane, or a pilot using different planes, especially different types of aircraft.

Chapter 3

Thermal and Wind detection

3.1 Thermal model and its limits

Thermals are modeled as vertical cylinders, described by the center point of the bottom area, their height, and their radius. At the moment, a constant strength is assumed within the whole cylinder. This model can be improved in different ways.

As one possible improvement, the cylinder should be allowed to be askew. This is important whenever there is more wind than just a light breeze, or the thermal lies close to mountainsides because thermals can stick to them rather than climb vertically.

The assumption of constant strength throughout the thermal is less accurate the higher from the ground a thermal is observed. This is due to an effect that makes the center of a thermal become stronger and could be modeled by making the strength a function of the distance to the center. As a basis for the improvement of the thermal model, the master's thesis by Christopher E. Childress on *An empirical model of thermal updrafts using data obtained from a manned glider* could come in handy. [4]

3.2 Detection and handling

Variables used in the algorithm:

$\mathfrak{p}''$	The position right before $\mathfrak{p}'$, i.e. usually, the position $2s$ ago
z_p	The altitude of position p
Δz	The altitude gain since $\mathfrak{p}'$
Δv	The change in speed
Δz_v	The loss of altitude due to gain of speed, or, vice versa, the altitude that was bargained for by losing speed

Δz_{gr} The expected altitude that should be lost due to the gliding property of the aircraft

Δz_{th} The altitude gain that is unaccounted for by the other variables, i.e., that is caused by a thermal

Algorithm 5: Procedure ThermalDetection

```
1  $\Delta z \leftarrow z_{\mathfrak{p}} - z_{\mathfrak{p}'}$;
2  $v_{old} \leftarrow \frac{\text{DISTANCE}(\mathfrak{p}',\mathfrak{p}'')}{t_{\mathfrak{p}'}-t_{\mathfrak{p}''}}$;
3  $v_{new} \leftarrow \frac{\text{DISTANCE}(\mathfrak{p},\mathfrak{p}')}{t_{\mathfrak{p}}-t_{\mathfrak{p}'}}$;
4  $\Delta v \leftarrow v_{new} - v_{old}$;
5  $\Delta z_v \leftarrow \frac{(\Delta v)^2}{2g} \cdot sign(\Delta v)$;
6  $\Delta z_{gr} \leftarrow \frac{\text{DISTANCE}(\mathfrak{p},\mathfrak{p}')}{glideratio}$;
7  $\Delta z_{th} \leftarrow \Delta z + \Delta z_{gr} + \Delta z_v$;
8  $vs_{th} \leftarrow \frac{\Delta z_{th}}{t_{\mathfrak{p}}-t_{\mathfrak{p}'}}$;
9  if $vs_{th} \geq \vartheta$ then
10     add $\mathfrak{p}$ to a thermal;
```

Algorithm 5 checks if the current positions lies within a thermal using the energy conservation law and the principle of a glide ratio. The glide ratio indicates how much altitude is lost when a glider travels a certain distance forward. It is almost constant in the weight of an aircraft and varies only slightly at operational velocities, thus a constant value works well in a first model and is used in this thesis. From the glide ratio, the expected altitude loss since the last position can be calculated (**line 6**).

A second factor is the altitude gained or lost due to a change in speed. It is calculated in **lines 2 to 5**, using

$$\frac{1}{2} \cdot mv^2 = mgh \tag{3.1}$$

$$\Leftrightarrow h = \frac{v^2}{2g}, \tag{3.2}$$

where $g = 9.81\frac{m}{s^2}$ is the gravitational acceleration. Inserting Δv, which is the difference between the current speed and the speed at the last stored position, into (3.2) yields the absolute value of the height gain or loss due to deceleration or acceleration, respectively. To get a signed value indicating whether height was gained or lost, the value is multiplied by the sign of the difference in velocity in **line 5**.

The third component is the actual height loss based on GPS, which is computed in **line 1**. As long as the plane is gliding (i.e., not motorized) and there is no thermal, these three values should sum up to 0. Note that Δz is the height **gain** while Δz_v and Δz_{gr} denote the **loss** of such. As a consequence, if the sum does not add to 0, the difference has to be caused by a thermal. In **lines 7 to 8**, the vertical speed of the thermal is calculated and in **line 9**, this value is then compared to a minimum strength of $\vartheta = 0.5\frac{m}{s}$. The threshold filters GPS imprecision by not allowing small values. The value of ϑ might also be adjusted if several consecutive positions have a smaller value. In that case, continuous imprecision becomes highly unlikely because it would mean that the GPS shield provides increasingly faulty data. Since planes usually have the ability to break, sink areas can not be reliably noticed with that procedure. Especially in common approach paths, many sink areas would be detected where, in fact, there are none, hence not the absolute value is taken in **line 9**.

A possible improvement concerns the glide ratio. A more detailed model could be developed, allowing for a speed dependency. This relation would have to be learnt from experience during use to continuously improve the precision of detected thermals. It is very aircraft specific, and would require a pilot to reliably swap profiles when flying different aircraft to prevent wrong calculations from being made.

There is still one situation that cannot be solved with the model described in **Algorithm 5**. If a plane is flying against a strong headwind and that wind diminishes in between two polled positions, the plane will very quickly accelerate (with respect to the ground, as only ground speed (GS) can be measured) without losing altitude. In the model, this would lead to a wrongfully detected thermal. A solution to this issue is explained in **Section 3.3.1**.

3.3 Wind

With only GPS positions as a basis to determine wind, there are two options, both of which are hardly acceptable if a reliable prediction in windy conditions is to be made. The first is to use the minimal airspeed of the aircraft, which would have to be provided by the user. If the ground speed (which is the speed that can be measured through GPS) falls below the minimal indicated airspeed (IAS) but the aircraft is not losing altitude quickly (which would be the measurable component of a stall), there must be headwind keeping the aircraft in the air. With this information, only one direction of the wind can be determined, though. Thus, to determine the x and y component of the wind, two aircraft have to be in such a state, heading in different directions. This, however, still only leads to a lower bound of the wind speed. In a similar manner, an upper bound could be determined if the maximum indicated airspeed is known, assuming it is never exceeded. For this to be useful, though, an aircraft has to be flying quite close to that limit.

The second option is to analyze the flight path. If an aircraft is climbing within a thermal, its course has approximately the form of a helix. If the helix is askew, this is most likely due to the airmass moving sideways. The circle pattern can be recognized by looking at the heading of the aircraft. Then, after a full circle has been flown, the lateral shift in position must be due to wind, assuming once again a more or less constant turn. This assumption is what renders this approach mediocre at best since an aircraft will rarely describe a near-perfect circle, be it due to a terrain blocking the path, the sun glaring into the pilot's face, the wind blowing in the opposite direction of the pilot's intention, or another aircraft that has to be avoided.

3.3.1 Improvement ideas

To improve wind and thermal detection, a solution that obviates the need for these options would be helpful. One element that can achieve this are indicated airspeed sensors. They can be installed on the wings and provide real-time speed data to the predictor. The wind detection would tremendously profit from their information, but also the thermal detection and, ultimately, the turn prediction if the sensors are attached at the wingtips. As for the wind detection, the speed calculated from the GPS data can be compared to the IAS, the difference yielding the wind speed. There are still at least two aircraft needed, flying in different directions, but because the comparison can be made at all times, this is not really a constraint anymore. This poses a strong improvement to the first method of wind detection without sensors, namely because it can happen every few seconds and because it does not only result in an upper or lower bound. But it also presents an improvement on the second approach, which needs to look at a large part of the flight history, because it can be calculated using only a few data points and does not need a few minutes of flying before the first wind can be detected.

The IAS sensors can also solve the mentioned problem of wrongfully detected thermals. If the model uses IAS instead of GS, rapidly changing wind does not influence Δv and thus the thermal recognition.

In order to keep the data on a thermal up to date, the strengths at known positions could be weighted according to how far in the past they have been discovered. Thus, if a thermal is not observed for some time and its strength changes, new data points will quickly outweigh older ones. This allows the predictions to quickly adapt to the natural cycle of thermals.

Chapter 4

Interaction with other aircraft

4.1 Communication

Aircraft have to frequently exchange the data of their predictions so that every aircraft in the vicinity can check for collisions. There are multiple possible approaches to achieve that. The simplest would be for every airplane to broadcast its position every second, marked with a unique ID. Then, other aircraft could build a history for every aircraft in the area, and predict their movement from these data points. This would lead to an unnecessarily high number of calculations, though, since each trajectory will be calculated by every aircraft, leading to a computation time of $\mathcal{O}(n^2)$, where n is the number of aircraft around.

As a correction, solving the problem of each trajectory being computed multiple times, an aircraft could compute its own trajectories and then send them to the other planes. While this concept solves the computation time issue, it will overload the radio frequency very quickly since a lot of data has to be transmitted. In situations where an aircraft is, say, initiating a turn while inside a thermal, there will be 10 different predictions, each possibly containing 100 future positions. Sending these 10 predictions will take multiple seconds (this is still per aircraft!). Thus, this approach is not feasible, even though the computation time for each aircraft is $\mathcal{O}(n)$.

To reduce the data that is transmitted, the trajectory data is sent in the second form explained in **Section 2.1.1**. General information is hereby only sent once per transmission, i.e., the current position and direction, the size, and the speed of the plane. There are two possible directions because there is a small difference depending on whether a turn or a straight trajectory is assumed for a certain prediction (see the difference between $\mathfrak{d}$ and $\mathfrak{b}$ in **Algorithm 4**). There could be a bit indicating how many directions are sent to save some data in case only straight lines or only turns are predicted. Trajectory specific data is sent as 2-tuples containing the trajectory type, and rate of turn in case the type is a turn. As of now, the trajectory type is a binary variable (straight or turn), but this could change later, so, in the calculation below, 2 bits are reserved for it. In **Table 4.1**, an estimation of the amount of data that has to be transmitted is shown.

Parameter	Description	Bytes
Position	4 numbers (lon, lat, alt, time)	$4 \cdot 32bit = 16$B
Direction	$2 \cdot 3$ numbers (x, y, z)	$6 \cdot 32bit = 24$B
size	collision radius of the plane	$16bit = 2$B
speed	-	$16bit = 2$B
sum	sum of constant bytes	44B
type	type of the trajectory	$2bit$
rate of turn	-	$30bit$
sum per trajectory	-	4B

Table 4.1: Overview of transmitted data

As an example, let us extend these numbers to the transmission of ten trajectories, which is a rather high but well possible number. In this case, the total data sums to $44Byte + 10 \cdot 4Byte = 84Byte$. Assuming a transmission rate of $R = 5kB/s$, this would, in theory, allow for up to $\frac{S_{max} \cdot R}{84B} = 21.9$ aircraft, where $S_{max} = \frac{1}{e} = 0.368$ is the maximum throughput of a slotted ALOHA transmission protocol. This is only a rough estimation, though, as the retransmission rate is ignored. Since the length of a transmission will fluctuate considerably, a fixed window size as used in slotted ALOHA is not a good choice either. Using an approach with variable window size could hence improve the number of aircraft that can be supported. When the number of aircraft grows too large, some trajectories could be omitted. This would have to be handled by the predictor as it could decide which predictions are the least likely.

Periodically, knowledge about thermals or wind has to be exchanged. This may happen every few seconds, but can be adapted in case the radio frequency is already heavily in use.

4.2 Collision Detection

In this section, the algorithm to detect collisions from the generated data is propounded.

Variables used in the algorithm:

$\mathcal{T}$ A list of the aircraft's own positions

$\mathcal{U}$ another aircraft's trajectory

$\mathfrak{d}$ the distance below which it is considered a collision

$\mathcal{L}$ An array-like object storing all collisions that are found

Algorithm 6: Function Detect Collision

```
Input: $\mathcal{T}$, $\mathcal{U}$, $\mathfrak{d}$
Function Detect Collision:
    $\mathcal{L} \leftarrow \emptyset$;
    $t_0 \leftarrow$ earliest $t \in \mathcal{T}$;
    forall $u \in \mathcal{U}$ : $u$ earlier than $t_0$ do
        if DISTANCE$(t_0, u) \leq \mathfrak{d}$ then
            $\mathcal{L} \leftarrow (t_0, u)$;
    forall $(t_{-1}, t, t_1) \in \mathcal{T}^3$: $t_{-1}, t$ and $t_1$ consecutive do
        forall $u \in \mathcal{U}$ between $t_{-1}$ and $t_1$ do
            if DISTANCE$(t, u) \leq \mathfrak{d}$ then
                $\mathcal{L} \leftarrow (t, u)$;
    return $\mathcal{L}$;
```

Algorithm 6 compares every point of the trajectory of aircraft A with the closest points of the trajectory of aircraft B (closest refers to time). Thus, every point is compared to at least two points of the other trajectory. If the time gap Δt of aircraft A is larger than the one of another aircraft, a point $\mathfrak{p}$ of A's trajectory will be compared to all points that lie, on the time scale, between point $\mathfrak{p}$ itself and the next point in A's trajectory, which has a timestamp of Δt more than $\mathfrak{p}$.

Since sudden movements cannot be foreseen, collision spheres can be made bigger than the size of the aircraft itself. How large could be made dependent on a security factor entered by the user to take into account his or her flight style, and from experience during testing. It might also be adjusted programmatically based on how many aircraft are close by to give the device a chance to prevent itself from getting overloaded. If the collision distance is too big, warnings might occur frequently when there is no actual danger.

A bargain could be to have two different sphere sizes that are compared. If only the larger ones collide, a weaker warning can be issued so that the pilot can differentiate. In all cases, an avoidance suggestion can be made based on trajectories that have been calculated anyways. Since these trajectories are plausible future trajectories, they will usually be feasible in flight. As a short example, consider an aircraft that is ending a turn. Its predictor will likely be calculating both a straight trajectory in the current direction and a turn whose trajectory is close to the continuation of the turn. If a collision is detected on the straight flight path, the system could trigger an advice to keep turning if no collision was detected there. If, on the other hand, the turn will lead to a collision, the pilot could be advised to end the turn and fly straight. Since the straight trajectory has just been checked for collisions, this is, at least in that moment, a safe advice.

Chapter 5

The Simulation

Alongside the algorithm, a simulation was created so that the ideas could be visualized and tested. To accelerate implementation, python was chosen as the programming language for both the implementation of the algorithm and the simulation. As environment for the simulation, Panda3D was elected. Being a 3D engine made for game development, it promised to provide a lot of useful functionality in modelling aircraft and thermals.

To stay as close to reality as possible, total decoupling of the `Predictor` class from the simulation was targeted. To simplify this step, an `Aircraft` class was created that simulates the plane and invokes the different functions of the predictor. The decoupling ensures that only data that would be provided in reality as well is used by the `Predictor`.

An `Aircraft` provides a FLY method that moves the model in the environment. There is an option to add some noise to the flight to create a more authentic setting. It can be toggled and adjusted from the user interface (UI). When noise is turned on, the fly method also adjusts the speed according to altitude lost or gained by said noise. The noise is modeled by a two-step Gaussian distribution, meaning that every second, three drifts $\mathfrak{D}_i$ are determined by

$$\mathfrak{D}_i \sim \mathcal{N}(0, \sigma), i \in \{x, y, z\},$$

with σ the standard deviation that can be adjusted in the UI. The engine provides a parameter dt that indicates the time that has passed since the last frame. FLY is called every frame to move the aircraft smoothly. During each call, a random x, y, and z movement are determined by

$$i \sim \mathcal{N}(\mathfrak{D}_i, dt \cdot \sigma), i \in \{x, y, z\},$$

where the factor dt is taken into the distribution. It does not have to be squared since σ denotes the standard deviation and not the variance.

The class also visualizes the predictions that are made by the `Predictor` as well as any thermals known to the latter. Further, it provides methods to send and receive prediction data, replacing the radio transmission, as well as one to start the collision detection based on the received data.

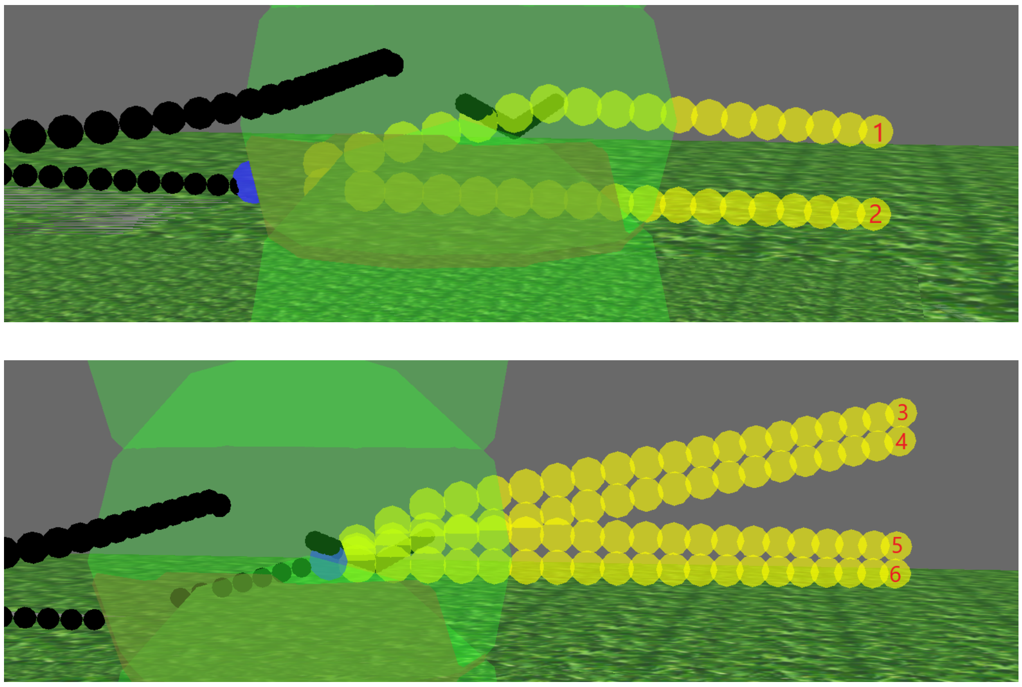

Figure 5.1: An aircraft right before and inside a thermal. Please note that only every other prediction sphere (yellow) is shown for simplicity.

The UI allows creating new aircraft and thermals at runtime using buttons, and trajectories of existing planes can be changed in an options dialogue, and aircraft can be deleted. Further, scenarios can be loaded at runtime and noise can be toggled and its value adjusted. While the simulation is paused for these dialogues, it is also possible to pause it by button or by pressing `P`, allowing to move the camera to a more informative angle or to analyze a situation in more detail.

In **Figure 5.1**, an example of the simulation is depicted. The black spheres are the positions obtained from the GPS shield. From these, the predictions, which are illustrated in yellow, are calculated. The blue sphere represents the collision sphere of the aircraft and the green spheres portray a thermal.

In the top image, the aircraft has been flying in a straight line and is about to enter the thermal. Thus, two predictions are made, that is, prediction 1 assuming the thermal is still present, and prediction 2 ignoring the thermal in case it has dissolved (see **Algorithm 3, lines 4 to 5**).

In the bottom image, the aircraft is amidst the thermal, which is why there are twice as many predictions now. Prediction 4 corresponds to $\mathcal{L}_1$ in **Algorithm 3 (Line 3)** while prediction 3 ($\mathcal{L}_2$) additionally adds the strength of the thermal, considering the event that the current climb is not due to the thermal, yet there

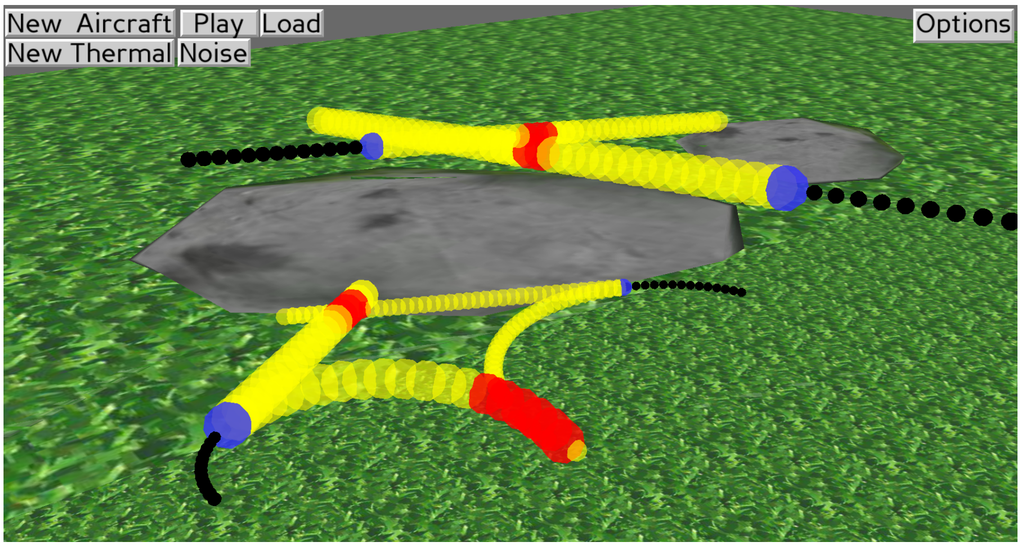

Figure 5.2: A screenshot of the simulation featuring the collision highlighting.

is a thermal in front of the plane (which is the current state of information). This event is quite unlikely, but the calculation is similar to prediction 5, which in turn is crucial that it be computed. Prediction 4 also factors in that we might not yet have detected the full extent of the thermal (because no plane has flown there yet) as its trajectory keeps going in the exact same direction as the recent points.

Prediction 6 covers the case where either the thermal ends at the current position of the aircraft because the thermal's size has changed, or because the climb was not caused by the thermal in the first place and ends now. Prediction 5 assumes that the climb is caused by the thermal and will continue until the thermal as it is currently known ends.

In **Figure 5.2**, a second example is shown. It displays a simple scenario to show the collision detection. The collision detection algorithm did at no point use any of the game engine's tools for collisions. Rather, the positions in the predictions are compared, colliding ones stored, and using the latter the scene is searched for the yellow spheres at the collision location so that they can be coloured in red.

Chapter 6

Analysis

6.1 Time Analysis

There are three crucial components that have to be examined for the time analysis.

1 Trajectory prediction

2 Computation of received trajectories

3 Trajectory comparison

The trajectory prediction is a swift process as it involves at most 10 trajectories that need to be computed. The constant part consists mainly of the linear approximation, but since only 15 positions are considered, this computation is not very time intensive. Further, the prediction time does not depend on the surroundings, namely thermals or the number of other aircraft. The average time of the prediction is about 0.015 seconds while the maximum is around 0.065 seconds. This times were achieved on an Intel i7 with 16GB of RAM.

As only parameters describing a trajectory are exchanged between aircraft, each of these trajectories has to be converted into a list of positions. This accounts for $50 - 60\%$ of the total collision detection time, the other $40 - 50\%$ originating from the trajectory comparison.

The sum over all comparisons between two trajectories has a runtime of $\mathcal{O}(n_t \cdot m)$, where n_t is the number of trajectories an aircraft has computed as its own predictions, and m is the total number of received trajectories. But because the number of trajectories an aircraft computes has a fixed upper bound of 10 trajectories, we have $\mathcal{O}(n_t) = \mathcal{O}(1)$ and $\mathcal{O}(m) = \mathcal{O}(n_t \cdot n) = \mathcal{O}(n)$, where n is the number of aircraft in the area.

When looking at the total number of comparisons $n_c = n_t \cdot m$ that are made, the linearity is conserved as $\mathcal{O}(n_t^2) = \mathcal{O}(1)$. This relation is shown in **Figure 6.1**.

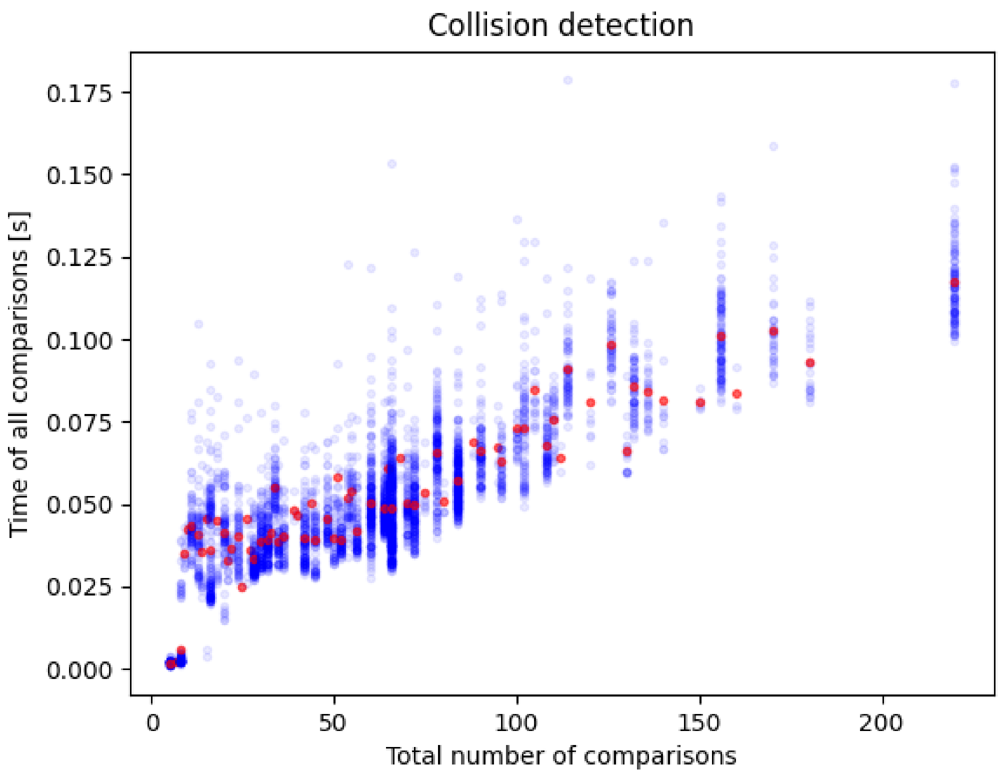

Figure 6.1: Computation time as a function of the total number of comparisons that are made

The blue dots are single data points, and the red dots represent the mean computation time for a given number of comparisons. The variance between neighbouring red dots comes from the fact that not every number of comparisons n_c can be achieved from every n_t or m. Computing an additional trajectory takes significantly longer than an additional comparison. This can be reasoned from the proportions mentioned above. While there are m trajectories that have to be computed, there are $n_c = n_t \cdot m$ comparisons, but both take roughly the same amount of time to execute.

In a scenario with 4 aircraft flying through thermals, the average time consumed by the computation of trajectories was $0.032s$ while the comparison took $0.025s$, giving an average collision detection time of $0.057s$. The peak values accumulated to $0.11s, 0.10s$, and $0.21s$. This is still well below the one-second limit that was aimed at for the frequency of these calculations.

6.2 Limits of the Algorithm

If one or two planes are added to the scenario described above, the linear growth is still visible. But in situations where several aircraft change their trajectory at the same time, the simulation cannot keep up with the amount of calculations that should be done every second, for the computation cannot be distributed on different machines but has to be made on the same device, hence gaining a quadratic runtime. With two more planes (that makes eight in total, or twice as many as in the original scenario), the simulation exceeds its limit and does not work properly anymore. Factoring in the quadratic runtime, this translates to a factor of 4 for the algorithm running on the aircraft in a distributed manner. Not wanting to run close to the operational limit, about 10 aircraft can be accommodated.

As the implementation on the device is planned to be written in C, the computations will likely run a lot faster, allowing the the algorithm to reliably work among more aircraft.

Turns are predicted under the assumption that the rate of turn is constant over the last few data points. This makes it hard to anticipate turns or their ending. If a turn is initiated slowly but steadily, the current model will not predict an accurate trajectory.

When presented with faulty data, the algorithm will consequently compute imprecise outputs. Some sort of input filter is necessary, deciding if an input is amiss.

6.3 Conclusion

The simulation shows that predictions are made in a way that seems logical when a human looks at them. Of course there are some that seem counter-intuitive in most situations, but cover a special case and are thus nevertheless important. It can further be concluded that collisions based on the predictions are recognized accurately since trajectories do not touch each other without being detected.

The simulation can be used to implement future ideas and test them without the need of many flight hours. Panda3d also provides a $C++$ interface, thus this could even be done closer to the language that will be used for the device.

Thermals, the core element of this thesis, are detected and used in the predictions. Wide-spread solutions like FLARM, which is designed for powered aircraft, do not consider thermals, rendering their collision warnings rather inaccurate in regions with lift areas. Since this is where gliders of any kind will usually fly, this is a crucial functionality to be provided to these planes.

Chapter 7

Further Work

7.1 Future Ideas

There are several parts of the trajectory prediction that can be examined in more detail, and other interesting usages of the device. Here, a few that have not been mentioned in the related chapter will be presented.

- **Terrain data.** The collision avoidance advisory does not yet account for obstacles. Yet, obstacles like mountains are omnipresent in the world of gliders. Taking terrain data into account for the course advisory would increase safety. In the current version, a pilot has to check for obstacles before he or she can safely engage in the advised flight path. In a situation where a collision may be imminent, this might be beyond any pilot's capacity. In such a situation, a highly reliable advisory is invaluable.
- **Compass.** Using a compass to determine the current heading of the aircraft could enhance the prediction in different ways. If the current bearing was known, the decision process could be improved, capitalizing on the difference between the straight-line and the turn assumption, reversing the argument in **Algorithm 4**. Further, from the difference between the heading according to the GPS data and the heading obtained from a compass, wind could be detected fairly easily. A crucial condition for this is that the device be fixated to the aircraft in a way that it cannot change its orientation compared to the aircraft.
- **Mode for parachutes.** Such a mode would allow a skydiver to be noticed by aircraft also equipped with the device. The latter could be warned if they are getting close to an area where a skydiver will be diving through soon. The data generated by a skydiver could prove useful in wind detection (requiring a detailed model, though, that takes the change of wind direction with increasing altitude into account).

Bibliography

[1] "Flarm," 2022. [Online]. Available: https://flarm.com/

[2] "Movable type scripts," 2022. [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html

[3] "World xcontest 2022 [home]," 2022. [Online]. Available: http://www.xcontest.org/

[4] C. E. Childress, "An empirical model of thermal updrafts using data obtained from a manned glider," Master's thesis, The University of Tennessee, May 2010. [Online]. Available: https://trace.tennessee.edu/cgi/viewcontent.cgi?article=1659&context=utk_gradthes