FAI Sporting Code

Fédération Aéronautique Internationale

Section 7F – XC Scoring

CIVL GAP Centralised Cross-Country Competition Scoring for Hang Gliders and Paragliders Classes 1 to 5

2025 Edition, V1.0

Effective May 1st, 2025

Maison du Sport International
Av. de Rhodanie 54
CH-1007 Lausanne
(Switzerland)
Tél. +41 (0)21 345 10 70
Fax +41 (0)21 345 10 77
E-mail: sec@fai.org
Web: www.fai.org

FEDERATION AERONAUTIQUE INTERNATIONALE

MSI - Avenue de Rhodanie 54 – CH-1007 Lausanne – Switzerland

Copyright 2025

All rights reserved. Copyright in this document is owned by the Fédération Aéronautique Internationale (FAI). Any person acting on behalf of the FAI or one of its Members is hereby authorised to copy, print, and distribute this document, subject to the following conditions:

1. The document may be used for information only and may not be exploited for commercial purposes.
2. Any copy of this document or portion thereof must include this copyright notice.
3. Regulations applicable to air law, air traffic and control in the respective countries are reserved in any event. They must be observed and, where applicable, take precedence over any sport regulations

Note that any product, process or technology described in the document may be the subject of other Intellectual Property rights reserved by the Fédération Aéronautique Internationale or other entities and is not licensed hereunder.

RIGHTS TO FAI INTERNATIONAL SPORTING EVENTS

All international sporting events organised wholly or partly under the rules of the Fédération Aéronautique Internationale (FAI) Sporting Code[1] are termed *FAI International Sporting Events*[2]. Under the FAI Statutes[3], FAI owns and controls all rights relating to FAI International Sporting Events. FAI Members[4] shall, within their national territories[5], enforce FAI ownership of FAI International Sporting Events and require them to be registered in the FAI Sporting Calendar[6].

An event organiser who wishes to exploit rights to any commercial activity at such events shall seek prior agreement with FAI. The rights owned by FAI which may, by agreement, be transferred to event organisers include, but are not limited to advertising at or for FAI events, use of the event name or logo for merchandising purposes and use of any sound, image, program and/or data, whether recorded electronically or otherwise or transmitted in real time. This includes specifically all rights to the use of any material, electronic or other, including software, that forms part of any method or system for judging, scoring, performance evaluation or information utilised in any FAI International Sporting Event[7].

Each FAI Air Sport Commission[8] may negotiate agreements, with FAI Members or other entities authorised by the appropriate FAI Member, for the transfer of all or parts of the rights to any FAI International Sporting Event (except World Air Games events[9]) in the discipline[10], for which it is responsible[11] or waive the rights. Any such agreement or waiver, after approval by the appropriate Air Sport Commission President, shall be signed by FAI Officers[12].

Any person or legal entity that accepts responsibility for organising an FAI Sporting Event, whether or not by written agreement, in doing so also accepts the proprietary rights of FAI as stated above. Where no transfer of rights has been agreed in writing, FAI shall retain all rights to the event. Regardless of any agreement or transfer of rights, FAI shall have, free of charge for its own archival and/or promotional use, full access to any sound and/or visual images of any FAI Sporting Event. The FAI also reserves the right to arrange at its own expense for any and all parts of any event to be recorded.

1	FAI Statutes,	Chapter 1,	para. 1.6	
2	FAI Sporting Code,	Gen. Section,	Chapter 4,	para 4.1.2
3	FAI Statutes,	Chapter 1,	para 1.8.1	
4	FAI Statutes,	Chapter 2,	para 2.1.1; 2.4.2; 2.5.2 and 2.7.2	
5	FAI By-Laws,	Chapter 1,	para 1.2.1	
6	FAI Statutes,	Chapter 2,	para 2.4.2.2.5	
7	FAI By-Laws,	Chapter 1,	paras 1.2.2 to 1.2.5	
8	FAI Statutes,	Chapter 5,	paras 5.1.1, 5.2, 5.2.3 and 5.2.3.3	
9	FAI Sporting Code,	Gen. Section,	Chapter 4,	para 4.1.5
10	FAI Sporting Code,	Gen. Section,	Chapter 2,	para 2.2.
11	FAI Statutes,	Chapter 5,	para 5.2.3.3.7	
12	FAI Statutes,	Chapter 6,	para 6.1.2.1.3	

Editor's note

Hang-gliding and paragliding are sports in which both men and women participate. Throughout this document the words "he", "him" or "his" are intended to apply equally to either sex unless it is specifically stated otherwise.

Document history

Date	Version	Authors	Changes
2025-03-23	Edition 2025 V1.0	Jörg Ewald	• Apply all changes from the 2025 CIVL Plenary, see "2025" in 2.1 History. • Rename all internal references from "section" to "chapter" . • Clarify that distance, time, arrival, and leading points each rounded to one decimal place. • Clarify that after application of penalty, scores are again rounded to one decimal place. • Replace all instances of "spheroid" with "ellipsoid", since ellipsoid is the more general term. • Clarify that in Time Trials and in Races with multiple gates, pilots are scored for the flight segment that results in the biggest covered distance.
2024-12-26	Edition 2024 V1.1	Jörg Ewald	• 9.1.1: introduce definition of "tolerance zone" • 9.2.1: edited for clarification
2024-09-21	Edition 2024 V1.0	Jörg Ewald	• Correct layout errors and textual omissions introduced in previous editions • 6.2.2: edit for clarification and consistency with the rest of the document • 9.1.1: clarify tolerance values to be used in FAI Category 2 competitions • 9.1.2: edit for clarification and consistency with the rest of the document • 9.2.2: edit for clarification and consistency with the rest of the document • 9.2.2: clarify application of tolerance in goal lines • 9.4.1: clarify "best time" for HG and PG • 12.3.1: improve formula definitions for better readability

Contents

1 Introduction

This document contains all definitions required to score centralised cross-country competitions for both hang-gliding and paragliding. Its main purpose is to serve as an addendum to sections 7A and 7B of the FAI sporting code. Additionally, it should serve as an educational tool for all parties involved in such competitions, as a reference for the implementation of scoring systems, as well as a basis for future improvements and modifications.

1.1 Scope

The document's scope is restricted to scoring of FAI Category 1 cross-country competitions for hang-gliding and paragliding: World and Continental championships in both sports. CIVL's rule setting targets these competitions exclusively, whereas organisers of FAI Category 2 competitions as well as non-sanctioned competitions are free to score their competitions however they like. Most of them do follow CIVL's lead, though, so this document should also cover most Category 2 events.

1.2 Sources

The remainder of this document is based on:

"The GAP Guide" (2011 edition)

FAI Sporting Code Section 7A for Hang-Gliding (2013 edition)

FAI Sporting Code Section 7B for Paragliding (2013 edition)

The scoring implementation within CIVL's scoring software, FS (aka "FScomp")

Appendix C of the Paragliding World Cup Association's 2013 Competition Rules

1.3 Changes from previous edition

Changes from the previous edition of this document are marked in red colour. If the chapter title is marked in red it means the whole chapter text was added or changed.

1.4 Differences between Hang-Gliding and Paragliding

Initially, both hang-gliding and paragliding competitions used the same system for scoring, generally known as "GAP". But over time, through their two separate sporting codes, the two disciplines introduced more and more changes that would only apply to one, but not the other. This mainly in non-standard situations such as stopped tasks, pilots landing just short of goal, or pilots crossing the start line too early. Where not explicitly stated otherwise, the contents of this document always apply to both disciplines. Definitions applying only to one, but not the other, are clearly marked as such.

Text marked in blue applies exclusively to hang-gliding.

Text marked in orange applies exclusively to paragliding.

2 The GAP Philosophy

CIVL's scoring system is generally known as "GAP", named after the first-name initials of its three inventors Gerolf Heinrichs (G), Angelo Crapanzano (A) and Paul Mollison (P). Their intention was to "create a fair scoring system easily adaptable to any competition anywhere in the world, both for hang gliding and paragliding, with a philosophy that is easy for the pilot to understand, regardless of the mathematical complexity of the underlying formulas".

2.1 History

Work on GAP started in 1998, and it was officially introduced in 2000, to allow scoring of competitions based on GPS track logs, instead of photographic evidence as it had been used until then.

2002

An updated version, named "GAP 2002" was published. This introduced the concept of leading points, which are calculated by comparing the complete track logs of all pilots in a task. Leading points replaced the departure points used in GAP 2000

2005

A variation of GAP 2002 was introduced in Australia, named "OzGAP" or "OzGAP 2005". The difference to GAP lies mainly in the way arrival points are calculated, but this was never adopted by CIVL.

2008

"GAP 2008" was officially released. The main scoring mechanisms remained unchanged from the 2002 edition, but the implementation of GAP 2008 included several rules introduced in the sporting codes for either hang-gliding or paragliding. These cover stopped tasks, starting too early, and landing between the end of the speed section and goal.

2011

"GAP 2011" marked another software release where the main scoring remained unchanged from the 2002 definition and implementation. The main changes were all for paragliding: altitude bonus in stopped tasks, as well as a reduced number of available points in stopped tasks and in tasks with no pilots in goal.

2012

The "Jump the Gun" rule for early starts in hang-gliding competitions changed in S7A. This was implemented in FS, but this was also released, unfortunately, as "GAP 2011".

2014

The edition introduced several significant changes for paragliding, a few of which also applied to hang gliding. Most of those changes originated from the Paragliding World Cup Association (PWCA), and 2014 was the first time that both CIVL and PWCA scored their paragliding competitions using the same formula. The changes were:

1. Nominal launch competition parameter (hang gliding and paragliding)
2. Final glide decelerator(paragliding)
3. Goal shape, see 6.2.2 (paragliding)
4. Purely linear distance points, see 12.1 (paragliding)
5. Adjusted formula for leading points, see 12.3 (paragliding)
6. No more arrival points, see 12.4 (paragliding)
7. Scoring of stopped tasks, see 13.3 (hang gliding and paragliding)
8. Use of FTV for competition scoring, see 16(paragliding)

2015

This 2015 edition applies the 2014 changes in leading points calculations for paragliding now also to hang gliding. In paragliding, the use of final glide deceleration methods is no longer mandatory. QNH is now used as the primary altitude measurement. Distance calculations continue to be based on the FAI sphere,

the introduction of a new distance measurement regime, based on the WGS84 ellipsoid, has been postponed.

2016

In Paragliding, the leading points weight (maximum available leading points) is doubled compared to the previous version. This reduces the available time points. Also, if no pilot is in goal, the weight is now calculated as the ratio between task distance and actual distance covered by the pilot who flew the furthest. The maximum in this case is 0.1 (equivalent to 100 points for a task with task quality 1).

In Hang-gliding, the penalty for jumping the gun was increased for one point every three seconds to one point every 2 seconds.

2018

Distance measurement for paragliding is based on the WGS84 ellipsoid. For FAI Category 1 events, a lower turnpoint radius tolerance is chosen. In hang-gliding, the goal line is now also replaced by a semi-circle facing away from the previous turnpoint.

2020

The changes are:

1. Remove final glide decelerators
2. No more prescribed turnpoint direction (including start)
3. Clarify task distance calculation
4. Clarify rule for restarts for races with multiple start gates and for elapsed time tasks
5. Use a constant leading weight for paragliding
6. Task results are given with one decimal point, only round once when calculating competition results
7. Adopt the PWCA's leading points calculation for paragliding
8. Adopt the PWCA's time points calculation for paragliding and hang-gliding
9. Minimum time for stopped tasks depends on Nominal Time
10. Stopped tasks: Redistribute removed time points as distance points
11. FTV: Use best score for FTV validity

2021

The changes are:

1. Start the Leading Coefficient graph used for leading points calculation at task start time (11.3.1)
2. Fix error in time points reduction formula in stopped tasks with multiple start gates (12.3.5)
3. Clarification of order in which penalties are applied (12.3)

2022

The changes are related to the method of Task distance calculation (chapter 7.2)

2023

The changes are:

1. Use GNSS altitude as default (chapter 4.3)
2. Paragliding: New points distribution, using Leading Time Ratio (LTR) (chapters 6.1 and 11)
3. Hang-gliding Class 2: No more Arrival Points (chapter 11)
4. Simplification and correction of some mistakes in formulas for Leading Coefficient (chapter 12.3.1)

2025

The changes are:

1. Nominal Launch is no longer a Competition Parameter, but a fixed value of 96% (5, 10.1)
2. Nominal Goal is no longer a Competition Parameter, but a fixed value of 30% (5, 10.2)
3. Paragliding: In FAI Category 1 competitions, the FTV factor is always 25% (5.4.1)
4. The only start procedure is what was so far known as "Air Start", "Ground Start" is no longer covered by GAP (6)

5. The only task type is what was so far known as “Race task” (either as “Race to Goal” or as “Elapsed Time task”), “Open Distance” tasks are no longer covered by GAP (6.1)
6. The range for the Leading Time Ratio is now 0..26% (was 0..50%) (6.1, 11)
7. Orientation of Goal Line: Follow optimized route, instead of turnpoint centres. (6.2.3.1)
8. Rename “Race to Goal” to “Race” and “Elapsed Time” to “Time Trial” (6.3)
9. Define projection algorithm for planar calculations (7.1.2)
10. New definition of algorithm for route optimization (7.1.3)
11. Define algorithm for geodesic calculations (7.1.4, 7.1.5)
12. The relative turnpoint tolerance is now 0.1% for all competitions, and will be reduced to 0% in 2026 (9.1)
13. Tolerance is also applied to the straight portion of a goal semicircle (9.1.3)
14. Adjusted coefficient in Launch Validity formula, to correct a typo from about 2014 (10.1)
15. Paragliding: new leading coefficient calculation (12.3.1)
16. Hang-gliding: Score-back time is now a fixed value of 15 minutes (13.3.1)
17. Paragliding: Score-back time is now a fixed value of 5 minutes (13.3.1)
18. Paragliding: new minimum requirements for stopped tasks (13.3.2, 15)
19. Paragliding: Bonus Glide Ratio is now 2.5 instead of 4.0, and will be reduced to 0 in 2026 (13.3.6)
20. Bonus Distance is calculated only for the pilot’s position at Task Stop Time (13.3.6)
21. Competition results are given with one decimal place (15)

2.2 Scoring Process

Scoring follows a nine-step process, as depicted in Figure 1:

1. Setting the competition parameters, or “GAP parameters”, according to the competition site, the expected pilot level and the expected tasks. This happens once for each competition, at the outset, and must not be changed throughout the competition. See chapter 5.
2. Setting a task – this happens typically once per day on flyable days. See chapter 6.
3. Letting the pilots fly the task. See chapter 8.
4. Evaluating the task, by collecting all pilots’ track logs for this task, and determining for each pilot the distance flown and, if the end of speed section was reached, in what time this happened. See chapter 9.
5. Calculating the task validity based on the task’s statistical values such as fastest time to ESS, number of pilots in goal, average distance flown and several others. See chapter 10.
6. Points allocation: Calculating the maximum number of points awarded for distance, speed, leading and arrival, based on the task validity and the statistical values found in the task evaluation. See chapter 11.
7. Scoring each pilot’s flight, by calculating the awarded points for distance, speed, leading and arrival. The outcome, the pilots’ total score, is the sum of these four values. See chapters 12 and 13.
8. Ranking all pilots according to their total score for the task results. See chapter 14.
9. Aggregation of task results for competition scoring and ranking. See chapter 15.

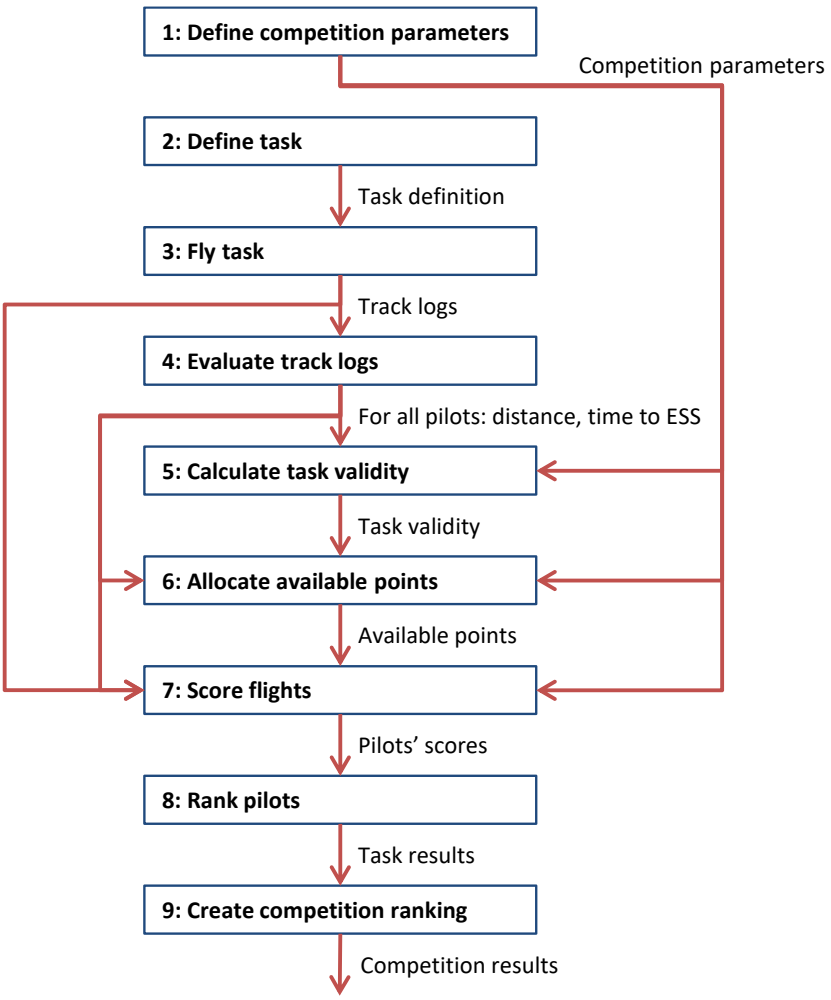


Figure 1: Scoring process

3 Definitions

The definitions of flights, locations, distances and times of CIVL GAP are described in Section 7A 5.2.1.

Flight
Free flight
Competition task
Competition flight
Take-off
Speed section
Start of speed section (SSS)
Turnpoint (TP)
Control zone
End of speed section (ESS).
Goal
Landing place
Task distance
Flown distance
Finish point
Race start
Start time
Start gate
Window open time
Task deadline
Finish time
Task time
Landing time

4 Measurements

4.1 Position

Coordinates of positions, such as turn points or pilot positions, are always given as WGS84 coordinates, based on the WGS84 ellipsoid. The coordinate format is UTM by default, but other formats can be chosen by organisers as appropriate.

4.2 Distance between two points

In general, task evaluation occurs in the x/y plain, therefore distance measurements are always exclusively horizontal measurements. Distances between two geographic points are calculated on the WGS84 ellipsoid, using the algorithms given in chapters 7.1.4 and 7.1.5.

For altitude bonus in stopped tasks (12.3.6), altitude is also considered, but this does not affect distance calculations between two geographic points.

For the more complex task distance calculations, as well as the calculation of pilots' distance along the course, see chapter 7.

4.3 Altitude

All altitude evaluation is primarily based on GNSS altitude, as recorded in the flight instrument tracklog. Recorded barometric altitude (the International Standard Atmosphere pressure altitude, QNE), from the primary tracklog or a backup log, and if necessary corrected by the scoring software for the pressure conditions of the task (QNH), may be taken into consideration only in case of problems with GNSS altitude logging.

Category 2 event organisers may choose to use barometric altitude instead of GNSS altitude.

4.4 Time

Time evaluation is based on UTC time, as given in GPS tracklogs. For better readability, times of the day may be expressed in local time for the competition location.

5 Competition Parameters

Before the first task, the following parameters must be defined by the meet director, or another person or group as defined by the competition's local regulations:

1. Nominal Distance
2. Minimum Distance
3. Nominal Time

The values set for these parameters define how each task's validity is calculated. They should therefore be chosen very carefully, considering the realistic potential of the flying site. Setting the values too low will prevent the formula from distinguishing between demanding, high-quality tasks and quick, easy low-quality tasks which are sometimes the only option due to weather conditions.

In addition, to allow for discards in competition results, the FTV factor must be set.

In previous versions of this document, two additional parameters had to be set:

1. Nominal Launch
2. Nominal Goal

From now on, a fixed value will be used for each of those, and they cannot be changed.

5.1 Nominal Distance

Nominal distance should be set to the expected average task distance for the competition. Depending on the other competition parameters and the distances flown by pilots, tasks shorter than Nominal Distance will be devalued in most cases. Tasks longer than nominal distance will usually not be devalued, if the pilots fly most of the distance.

For GAP to be able to distinguish between good and not-so-good tasks, and devalue the latter, it is important to set nominal distance high enough[13].

5.2 Minimum Distance

The minimum distance awarded to every pilot who takes off. It is the distance below which it is pointless to measure a pilot's performance. The minimum distance parameter is set so that pilots who are about to "bomb out" will not be tempted to fly into the next field to get past a group of pilots – they all receive the same number of points anyway.

5.3 Nominal Time

Nominal time indicates the expected task duration, the amount of time required to fly the speed section. If the fastest pilot's time is below nominal time, the task will be devalued. There is no devaluation if the fastest pilot's time is above nominal time.

Nominal time should be set to the expected "normal" task duration for the competition site, and nominal distance / nominal time should be a bit higher than typical average speeds for the area.

[13] See also this excellent series of articles on the subject: Part 1:http://ozreport.com/1360767307; Part 2: http://ozreport.com/1360858575; Part 3: http://ozreport.com/1360944246

5.4 FTV factor

5.4.1 FAI Category 1 competitions

5.4.2 Other competitions

The FTV factor is set by the organizers as they deem appropriate.

5.5 Nominal Launch

No longer a competition parameter, but a fixed value of 96%, which is used to calculate Launch Validity (10.1).

5.6 Nominal Goal

No longer a competition parameter, but a fixed value of 30%, which is used to calculate Distance Validity (10.2).

6 Task Setting

6.1 Definition of a task

A task definition consists of:

- A launch point, given as WGS84 coordinates
- Several control zones (6.2)
- A goal (6.2.3)
- An indication which of the control zones is the Start of Speed Section (often referred to as “Start” or “Airstart”)
- If goal does not serve as End of Speed Section: An indication which of the control zones is the End of Speed Section
- A launch time window
- A start procedure, including timing (6.3)
- Optionally, a task deadline
- Leading-Time-Ratio (LTR): The portion of points out of the pool reserved for time, arrival (hang-gliding only) and leading points that will be allocated to leading points. Can be set between 0% (no leading points) and 26% (of all the points not allocated to distance points, 26% go to leading, and 74% go to time points). The default LTR is 26% in paragliding, and 17.5% in hang-gliding. See also chapter 11.

6.2 Control zones

Control zones are geographical areas which must be reached by the pilots during a task. The types of control zones are:

1. Turnpoint cylinder, see 6.2.1
2. Line, see 6.2.2
3. Goal line, see 6.2.3.1

6.2.1 Turnpoint cylinder

A turnpoint cylinder is defined as:

- A centre point *c*, given as WGS84 coordinates
- A radius *r*, given in meter

A turnpoint cylinder is then given as the cylinder with radius *r* around the axis which cuts the x/y plain orthogonally at the cylinder’s centre point *c*. For task evaluation purposes, only the cylinder’s projection in the x/y plain is considered: a circle of radius *r* around *c*.

Note that the designation of “enter” or “exit” cylinder has been removed, to reduce a potential source of confusion and task setting errors. Whether a turnpoint is considered reached is determined either by the presence of a single tracklog point inside the turnpoint cylinder’s tolerance band, or by the presence of two consecutive tracklog points which lie on opposite sides of the turnpoint cylinder boundary (a “crossing”). The direction in which such a crossing occurs is irrelevant.

Task setters may still choose to indicate whether the start or subsequent turnpoint cylinders are “enter” or “exit”, to explain their intended task route. But pilots are not bound to those indications.

6.2.2 Line

The line control zone is defined as follows:

- Centre point c, and two end points e1 and e2, given by their WGS84 coordinates. These three points are calculated from the following parameters, following the procedures described below

- Waypoint w, given as WGS84 coordinates, one of the official competition waypoints
- Distance d between w and the line's centre point c, given in kilometer. The distance must lie between -100 km and 100 km, default is 0.0 km.
- Orientation o, given in degrees. This specifies in what world direction (relative to true north) we must travel the distance d from w to reach c. Orientation can be expressed in one of two ways:
    a. as decimal degrees in multiples of 2.5°
    b. as cardinal direction in English, which will then be converted into degrees according to Table 1.

  There is no default orientation, it must be specified.
- Length l, given in kilometer. This defines the distance between c and e1 and e2, respectively. The length must lie between 0.1 and 50.0 km, default is 1.0 km.

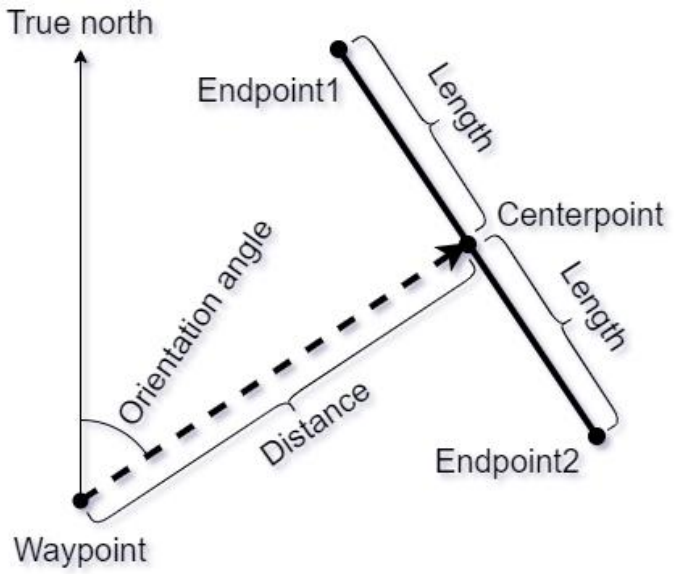


Figure 2: general linear control zone

Procedure for calculating the 3 points c, e1 and e2 (see Figure 2):

1. Calculate the geodesic line starting from the point w with the initial azimuth equal to the angle o.
2. On this geodesic line, find the point at travel distance d. This is the centre point c.
3. Find the arriving azimuth.
4. From c, calculate a geodesic line at azimuth equal to the arriving azimuth (from step 3) plus 90.0°.
5. On this geodesic line, find the point at travel distance l. This is the endpoint e1.
6. From c calculate another geodesic line at azimuth equal to the arriving azimuth (from step 3) minus 90.0°.
7. On this geodesic line find the point at travel distance l. This is the endpoint e2.

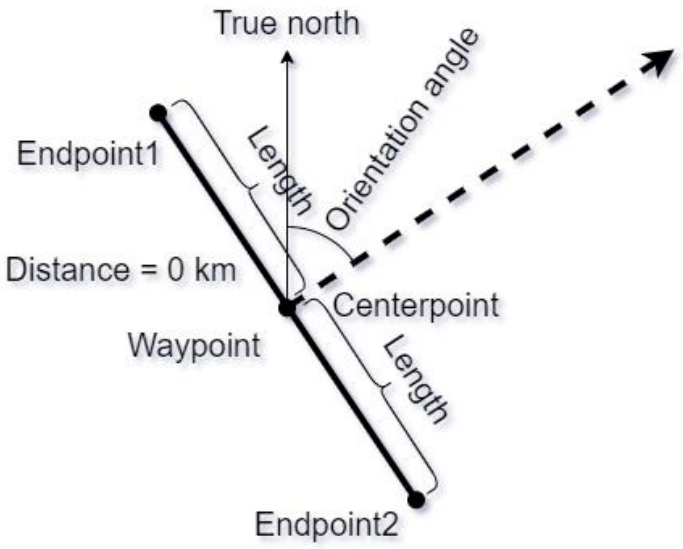


Figure 3: linear control zone with d=0

Procedure for calculating the 3 points c, e1 and e2 in case of d = 0 (see Figure 3):

1. The centre point c in this case is the same as the defining waypoint w.
2. To calculate e1 and e2, continue with steps 4 through 7 in the generic procedure.

6.2.2.1 Orientation

Orientation may be expressed as a decimal number (the angle in degrees between true North and the distance vector in clockwise direction), but it also can be expressed cardinal direction. The possible options cardinal directions, and their corresponding angles are given in Table 1.

Cardinal direction	Angle
N	0.0°
NNE	22.5°
NE	45.0°
ENE	67.5°
E	90.0°
ESE	112.5°
SE	135.0°
SSE	157.5°
S	180.0°
SSW	202.5°
SW	225.0°
WSW	247.5°
W	270.0°
WNW	292.5°
NW	315.0°
NNW	337.5°

Table 1: Cardinal directions

The total line length will always be equal to two times length l. For example, the default total line length will be 2.0 kilometers, because the default length is 1.0 km. The minimum total line length will be 200 m, because the minimum length is 0.1 km = 100 meters.

A negative value for the distance d means the same absolute distance, but in the opposite direction. For example, a line oriented at NE at distance -5.0 km will be absolutely the same as a line at distance 5.0 km, but oriented SW.

6.2.3 Goal

A goal can be either of:

- A turnpoint cylinder
- A goal line (6.2.3.1)

6.2.3.1 Goal line

The Goal Line is defined by:

a. A centre point *c*, given as WGS84 coordinates
b. A length l, given in meters
c. A previous point *p*, given as WGS84 coordinates

The previous point *p* is defined as the optimized route point on the last control zone before goal. This point is obtained as part of the task distance calculation, see chapter 7.2.

The goal line is defined as the line that crosses through *c*, lies perpendicular to the line between *p* and *c*, and extends by *l*/2 meters from c in each direction.

The goal line control zone consists of the semi-circle with radius *r* behind the goal line, when coming from *p*. See Figure 4. This allows to reach Goal from any direction.

Physical lines can be used in addition to the official, virtual goal line as defined by WGS84 coordinates, to increase attractiveness for spectators and media, and to increase visibility for pilots. Physical lines must be at least 50m long and 1m wide, made of white material and securely attached to the ground. The physical line must match as closely as possible the corresponding virtual line as defined by the goal GPS coordinates and the direction of the last task leg. It must not be laid out further from the previous turn point than the goal GPS coordinates.

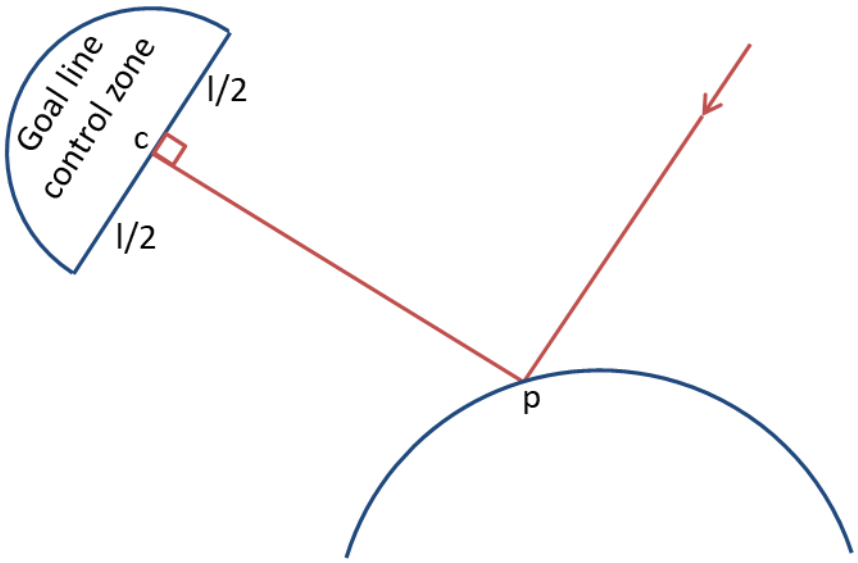


Figure 4: Goal line

6.3 Task types

The task type defines how an individual pilot's start time is determined. The two types are:

1. Race (6.3.1)
2. Time Trial (6.3.2)

Both task types use air starts: the competitors are free to launch any time during the launch window. If control zones are set before the start, the pilots are free to fly that part of the route and cross these control zones as they see fit. Race start is defined as the last crossing of the start control zone before continuing to fly through the remainder of the task.

6.3.1 Race

In a Race task, start is defined by one or more so-called "start gates". The first – or only – start gate is given as a daytime. Subsequent start gates are given as a time interval, along with the number of start gates.

Example 1: "We have a Race; the start gate opens at 13:00"

Example 2: "We have a Race with 5 start gates from 13:30 at a 20-minute interval." – the start gate times in this case are 13:30, 13:50, 14:10, 14:30, and 14.50.

Pilots are free to start any time after the first (or single) start gate. A pilot's start time is then defined as the time of the last start gate after which he started flying the speed section of the task.

Example 3: Given the start gates from Example 2 above, pilot A, crossing the start cylinder at 13:49:01, will be given a start time of 13:30. Pilots who start after 14:50 will be given a start time of 14:50.

Starting before the first (or only) start gate is considered a failed start. The two disciplines handle failed starts differently, see chapter 13.2.

6.3.2 Time Trial

In a Time Trial task, start is defined by a single "start gate", given as a daytime. Pilots are free to start any time after this start gate. A pilot's start time is then defined as the time at which he started flying the speed section of the task. Each pilot has therefore an individual start time.

Example 1: "We have a Time Trial, the start gate opens at 12:30" – pilot A starting at 12:31:03 has a start time of 12:31:03, pilot B starting at 15:48:28 has a start time of 15:48:28.

7 Task and flight distance calculations

Determining a task's distance is based on finding the shortest ("optimized") path for a route from a start point and an end point, touching any number of intermittent cylinders or lines. The same calculation is also used to determine the distance a pilot must still fly to complete a task from any position in their tracklog, and therefore their flight distance up to that point (9.3).

The calculation of such shortest path requires several algorithms. This chapter defines these algorithms and shows how they are used to calculate task and flight distances.

7.1 Algorithms

While track recordings and waypoints are given on the WGS84 ellipsoid, the calculations required to find the shortest path work in planar (Cartesian) geometry. Therefore, all WGS84 coordinates, of starting point, of the control zones (cylinder centres and line end points), and of the goal, must be transformed to Cartesian coordinates. We do not use the planar distance of the calculated shortest path. Instead, we transform the resulting path points for each control back to WGS84 coordinates, correct them for any error resulting from the projection, and then calculate the distance on the ellipsoid.

The calculations make use of the following algorithms:

1. **GeodesicToCartesian**, to calculate the Cartesian coordinates of a point on the WGS84 ellipsoid (7.1.1).
2. **PathFinder**, to calculate the path points that define the shortest path from point A to point B via a set of control zones, in Cartesian geometry (7.1.3).
3. **CartesianToGeodesic**, to calculate the WGS84 coordinates from the Cartesian coordinates of a point (7.1.1).
4. **DirectGeodesic**, to find a point on the WGS84 ellipsoid that is a given distance and direction from a given point (0).
5. **InverseGeodesic**, to find distance and direction between two points on the WGS84 ellipsoid (0).
6. **EllipsoidDistance**, an optimized version of InverseGeodesic, which only delivers distance, but with significantly less computational effort (7.1.5).
7. **FindTaskAreaCentre**, to find the centre of a task's area, as required by WGS84ToCartesian and CartesianToWGS84 (7.1.6).
8. **ProjectionCorrection**, to ensure that the path points in WGS84 lie on their corresponding control zone boundaries, regardless of any projection distortion that may occur (7.1.7).
9. **RouteOptimizer,** to calculate the path points that define the shortest path from point A to point B via a set of control zones, on the WGS84 ellipsoid, as well as the distance from A to B along this path (7.1.8).

7.1.1 Common definitions

$$geodesicRouteElement = \text{cylinder}(lat, lon, radius) \,|\, \text{line}(\text{point}_1(lat, lon), \text{point}_2(lat, lon)) \,|\, \text{goal}(lat, lon, radius, type)$$

$$cartesianRouteElement = \text{cylinder}(x, y, radius) \,|\, \text{line}(\text{point}_1(x_1, y_1), \text{point}_2(x_2, y_2)) \,|\, \text{goal}(x, y, radius, type)$$

$$geodesicRouteDefinition = \left\{ \begin{array}{c} startPoint(lat, lon), \\ geodesicRouteElement_1, \dots, geodesicRouteElement_n, \\ endPoint(lat, lon) \end{array} \right\}$$

$$cartesianRouteDefinition = \left\{ \begin{array}{c} startPoint(x, y), \\ cartesianRouteElement_1, \dots, cartesianRouteElement_n, \\ endPoint(x, y) \end{array} \right\}$$

$$geodesicPath = \{point_1, \dots, point_n\} \text{where } point_i = \text{point}(lat_i, lon_i)$$

$$cartesianPath = \{point_1, \dots, point_n\} \text{where } point_i = \text{point}(x_i, y_i)$$

7.1.2 GeodesicToCartesian & CartesianToGeodesic

To convert coordinates on the WGS84 ellipsoid to Cartesian coordinates, and back, a localized Transverse Mercator projection (LTM) is used, which is based on the centre point of the area of interest. This ensures a high accuracy within the task area, within 100 km of the centre point, with minimum distortions resulting from the conversions.

LTM differs from the well-known Universal Transverse Mercator projection (UTM) in the following ways:

1. Scaling depends on the centre point's latitude:
$$scaling = \begin{cases} centre.lat \leq 55°: 0.99994 \\ centre.lat > 55°: 0.99994 + \dfrac{centre.lat - 55}{60} * 1.3 * 10^{-4} \end{cases}$$
2. The centre meridian is defined by the centre point's longitude.
3. The cartesian point (0, 0) is defined as the centre point's Cartesian projection.

This can be achieved with the library PROJ[14]. Annex A gives a sample how PROJ can be used to create the two converters. For systems where PROJ is not available, an alternative implementation is also given in Annex A.

$\mathbf{GeodesicToCartesian}\big(point(lat, lon)\big): \boldsymbol{point(x, y)}$

$\mathbf{CartesianToGeodesic}\big(point(x, y)\big): \boldsymbol{point(lat, lon)}$

7.1.3 PathFinder

PathFinder is the name we gave the algorithm presented in Ding et al. (2018)[15], to find the shortest path from point A to B via a set of control zones in Cartesian space. Ding et al.'s paper handles circular control zones only. For linear control zones, their algorithm must be extended, to find the path point on the line. This extension is described in Annex B.

$\mathbf{PathFinder}(cartesianRouteDefinition): cartesianPath$

PathFinder needs to know when to stop finding a more accurate solution. For our purpose, a threshold of 10cm is sufficient to ensure that the solution is accurate within 1m.

$\varepsilon = 0.1\text{m}$

7.1.4 DirectGeodesic & InverseGeodesic

Geodesic algorithms solve two distinct problems on the WGS84 ellipsoid:

1. Find the distance d between two points A and B, and their relative directions a and a' ("azimuth"): This is the inverse geodesic problem.
2. Find the point B that lies at a distance d in direction a from a given starting point A: This is the direct geodesic problem.

To resolve these problems, we use the algorithms presented in Karney (2013)[16], and implemented in GeographicLib [17] . Alternatively, where GeographicLib is not available, implementations of the corresponding algorithms presented in Vincenty (1975)[18] serve the same purpose. Both sets of algorithms deliver results with similar accuracy (+/- 1 millimeter or less), several orders of magnitude above what is relevant for the scoring of a sport where distances are always measured in whole meters.

---

[14] https://proj.org

[15] Ding, Xie, Jiang, An Efficient Algorithm for Touring n Circles, EITCE 2018. Download here: https://www.matec-conferences.org/articles/matecconf/pdf/2018/91/matecconf_eitce2018_03027.pdf

[16] Karney, C. F. F. (2013). Algorithms for geodesics. *Journal of Geodesy*, 87(1), 43–55.

[17] https://geographiclib.sourceforge.io/

[18] Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 22(176), 88–93.

$\mathbf{DirectGeodesic}(point_1(lat, lon), distance, azimuth)$: $\boldsymbol{point_2(lat, lon)}$

$\mathbf{InverseGeodesic}\big(point_1(lat, lon), point_2(lat, lon)\big)$: $\boldsymbol{distance, azimuth}$

7.1.5 EllipsoidDistance

In many cases, we only need to calculate the distance between two points given as WGS84 coordinates and are not interested in their relative direction to each other. In these cases, rather than applying InverseGeodesic (0), we use EllipsoidDistance, which gives also very precise results, but with less computational effort and therefore higher speed. This is mainly relevant not for scoring software, but for navigation instruments which need to show pilots their distance from a cylinder, and which are limited in their computational speed.

$\mathbf{SpheroidDistance}\big(point_1(lat, lon), point_2(lat, lon)\big)$: $\boldsymbol{distance}$

A sample implementation of EllipsoidDistance is given in Annex C.

7.1.6 FindTaskAreaCentre

The algorithms GeodesicToCartesian and CartesianToGeodesic (7.1.1) require the definition of a centre point of the area of interest.

The FindTaskAreaCentre algorithm given below relies on two additional algorithms:

1. FindBoundingBox: To find the northern, eastern, southern and western boundaries of the smallest area that contains a set of points given as WGS84 coordinates (7.1.6.1).
2. FindCentrePointOfBox: To find the centre point of a bounding box, given by its south-western and north-eastern corners (7.1.6.2).

$\mathbf{FindTaskAreaCentre}(geodesicRouteDefinition)$:
$boundingBox_{temp} = \text{FindBoundingBox}(geodesicRouteDefinition)$
$centrePoint_{temp} = \text{FindCentrePointOfBox}\big(boundingBox_{temp}\big)$
$cartesianRouteDefinition_{temp} = \text{GeodesicToCartesian}\big(geodesicRouteDefinition, centrePoint_{temp}\big)$
$cartesianPath_{temp} = \text{PathFinder}\big(cartesianRouteDefinition_{temp}\big)$
$geodesicPath_{temp} = \text{CartesianToGeodesic}\big(cartesianPath_{temp}, centrePoint_{temp}\big)$
$correctedPath_{temp} = \text{ProjectionCorrection}\big(geodesicPath_{temp}, geodesicRouteDefinition\big)$
$boundingBox_{final} = \text{FindBoundingBox}\big(corrected\ Path_{temp}\big)$
$\boldsymbol{taskAreaCentre} = \text{CentrePoint}(boundingBox_{final})$

This must only be done once per task. The coordinates of *taskAreaCentre* should then be stored for this task and used for all further uses of GeodesicToCartesian and CartesianToGeodesic.

7.1.6.1 FindBoundingBox

Given a set of points P on the WGS84 ellipsoid, each given as latitude and longitude, the bounding box is defined as the smallest area defined by a maximum and minimum latitude and longitude that contains all points within P.

$\mathbf{FindBoundingBox}(P = \{\text{point}_1(\text{lat}, \text{lon}), \text{point}_2(\text{lat}, \text{lon}), \dots, \text{point}_n(\text{lat}, \text{lon})\})$
$minimumLatitude = \min(\forall \text{point}_i \in P: \text{point}_i.\text{lat})$
$maximumLatitude = \max(\forall \text{point}_i \in P: \text{point}_i.\text{lat})$
$normalizedLongitudes = \{\forall \text{point}_i \in P: ((\text{point}_i.lon + 180)\ mod\ 360) - 180\}$
$sortedLongitudes = SortAscending(normalizedLongitudes)$
$longitudeGaps = \{\forall \text{lon}_\text{i} \in sortedLongitudes, 1 < i \leq n: \text{lon}_\text{i} - \text{lon}_{\text{i}-1}\}$
$maxGap = \max(\forall \text{gap}_i \in longitudeGap)$

$maxGap > 180°$:
$minimumLongitude = lon_k\ (lon_j, lon_k \in sortedLongitudes\ \&\ k = j + 1\ \&\ lon_k - lon_j = maxGap)$
$maximumLongitude = lon_j\ (lon_j, lon_k \in sortedLongitudes\ \&\ k = j + 1\ \&\ lon_k - lon_j = maxGap)$

$maxGap \leq 180°$:
$minimumLongitude = \min(\forall lon \in normalizedLongitudes)$
$maximumLongitude = \max(\forall lon \in normalizedLongitudes)$

$\boldsymbol{boundingBox} = rectangle(point_{SW}(mimimumLatitude, minimumLongitude),$
$point_{NE}(mimimumLatitude, minimumLongitude))$

7.1.6.2 FindCentrePointOfBox

$\textbf{FindCentrePointOfBox}(box(point_{SW}, point_{NE})$:
$$lat = \frac{box.point_{SW}.latitude + box.point_{NE}.latitude}{2}$$

$box.point_{SW}.longitude > box.point_{NE}.longitude$:
$$lon = \frac{Box.point_{SW}.longitude + Box.point_{NE}.longitude + 360°}{2} \; mod \; 360$$
$lon > 180°: lon = lon - 360°$

$Box.point_{SW}.longitude \leq Box.point_{NE}.longitude$:
$$lon = \frac{Box.point_{SW}.longitude + Box.point_{NE}.longitude}{2}$$

$\boldsymbol{centrePoint} = point(lat, lon)$

7.1.7 ProjectionCorrection

Despite the use of the high-accuracy projection algorithms used to convert WGS84 coordinates to Cartesian coordinates, and back (7.1.2), the WGS84 projections of the path points found by PathFinder (7.1.3) still often do not lie exactly on the control zone's boundary. We therefore apply a simple correction where we find the point on the control zone boundary that is closest to the calculated path point and use this corrected point for all distance calculations.

$\textbf{ProjectionCorrection}(geodesicPath, geodesicRouteDefinition)$:
$n = geodesicRouteDefintion.routeElements.count$
$\forall point_i \in geodesicPath: 1 \leq i \leq n \;\&\; geodesicRouteDefinition.element_i.type = cylinder$:
$distance, azimuth = \text{InverseGeodesic}(point_i, geodesicRouteDefinition.element_i.point)$
$correctedPoint_i$
$= \text{DirectGeodesic}(geodesicRouteDefinition.element_i.point, azimuth, geodesicRouteDefinition.element_i.radius)$

$\forall point_i \in geodesicPath: 1 \leq i \leq n \;\&\; geodesicRouteDefinition.element_i.type = line$:
$lineLength, lineAzimuth$
$= \text{InverseGeodesic}(geodesicRouteDefinition.element_i.point_1, geodesicRouteDefinition.element_i.point_2)$
$pathPointDistance, pathPointAzimuth$
$= InverseGeodesic(geodesicRouteDefinition.element_i.point_1, point_i)$
$dAzimuth = |lineAzimuth - pathPointAzimuth|$
$corectedPathPointDistance = pathPointDistance * \cos(dAzimuth)$
$correctedPoint_i$
$= \text{DirectGeodesic}(geodesicRouteDefinition.element_i.point_1, lineAzimuth, correctePathPointDistance)$

$$\boldsymbol{correctedPath} = \begin{Bmatrix} geodesicRouteDefinition.startPoint, \\ \forall i: 1 \leq i \leq n: correctedPoint_i, \\ geodesicRouteDefinition.endPoint \end{Bmatrix}$$

7.1.8 RouteOptimizer

At the core of all task and flight distance calculations lies the RouteOptimizer algorithm that calculates the shortest path for a route from a start point through a series of control zones to a destination point. The result of this algorithm is a sequence of Geodesic points that represent the optimal crossing points for shortest overall distance from start point to destination, along with the Geodesic distance over all those points.

$\mathbf{RouteOptimizer}(geodesicRouteDefinition, taskAreaCentre)$:
$cartesianRouteDefinition = \text{GeodesicToCartesian}(geodesicRouteDefinition, taskAreaCentre)$
$cartesianPath = \text{PathFinder}(cartesianRouteDefinition)$
$geodesicPath = \text{CartesianToGeodesic}(cartesianPath, taskAreaCentre)$
$\boldsymbol{optimizedPath} = ProjectionCorrection(geodesicPath, geodesicRouteDefinition)$
$$\boldsymbol{optimizedDistance} = \sum_{0}^{n-1} \text{EllipsoidDistance}(\, correctedPath.point_i, correctedPath.point_{i+1})$$

7.2 Task distances

Task distance is defined as the distance of the optimized path from launch to goal. Speed section distance is defined as the distances of the optimized path from launch to ESS, minus the distance of the pre-start portion.

$launch = \text{point}(lat_{launch}, lon_{launch})$
$$task = \left\{ \begin{matrix} launch, \\ preElement_1, \dots, preElement_m, \\ startOfSpeedSection, \\ element_1, \dots, element_n, \\ endOfSpeedSection \\ postElement_1, \dots, postElement_n, \\ goal.point \end{matrix} \right\}$$
$$\boldsymbol{taskDistance} = \begin{cases} goal.type = line: \text{RouteOptimizer}(task).optimizedDistance \\ goal.type = cylinder: \text{RouteOptimizer}(task).optimizedDistance - goal.radius \end{cases}$$
$$launchToESS = \left\{ \begin{matrix} launch, \\ preTurnpoint_1, \dots, preTurnpoint_m, \\ startOfSpeedSection, \\ turnpoint_1, \dots, turnpoint_n, \\ endOfSpeedSection \\ endOfSpeedSection.point \end{matrix} \right\}$$
$launchToESSPath = \text{RouteOptimizer}(launchToESS).optimizedPath$
$launchToESSDistance = \text{RouteOptimizer}(launchToESS).optimizedDistance - endOfSpeedSection.radius$
$$preSpeedSectionDistance = \sum_{0}^{startOfSpeedSection.index-1} \text{EllipsoidDistance}(\, launchToESSPath.point_i, launchToESSPath.point_{i+1})$$
$\boldsymbol{speedSectionDistance} = launchToESSDistance - preSpeedSectionDistance$

8 Flying a task

8.1 Re-starting

In Races with multiple start gates and in Time Trials, pilots may return to the start and take a later start time after already having flown a portion of the Speed Section. They will be scored for that start which resulted in the biggest covered distance. If multiple starts resulted in them reaching goal, they will be scored for the last start after which they reached goal.

9 Task evaluation

From each pilot's track, task evaluation determines the distance this pilot flew along the task, and the time this pilot took to fly the speed section.

9.1 Tolerances

9.1.1 Cylinder tolerance

To compensate for the very slight distance measurement differences resulting from the use of different distance measurement algorithms in flight recorders and evaluation programs, a tolerance, consisting of a percentage and an absolute minimum, is applied when determining whether a pilot reached a cylinder (e.g. SSS, turnpoint of Goal). This had to be introduced so that a pilot reading the distance to the next cylinder centre from his flight instrument can rely on having reached the turnpoint when the distance displayed by the instrument is smaller than the defined turnpoint cylinder radius.

$relativeTolerance = 0.1\%\ (from\ GAP\ 2026: 0.0\%)$

$absoluteTolerance = 5m$
$turnpoint_i: innerRadius_i = min(radius_i * (1 - relativeTolerance), radius_i - absoluteTolerance)$
$turnpoint_i: outerRadius_i = max(radius_i * (1 + relativeTolerance), radius_i + absoluteTolerance)$

The tolerance zone is defined as the zone between the two cylinders defined by the turnpoint's center point and its innerRadius and its outerRadius, respectively.

9.1.2 Line tolerance

For the same reasons that a tolerance is applied to cylinder control zones, a tolerance is applied to the line control zones. The tolerance % is the same as for the cylinder control zones. The absolute tolerance for each line control zone (in meters) is calculated from the Distance and the Length (whichever is greater). The minimum tolerance is also 5 meters.

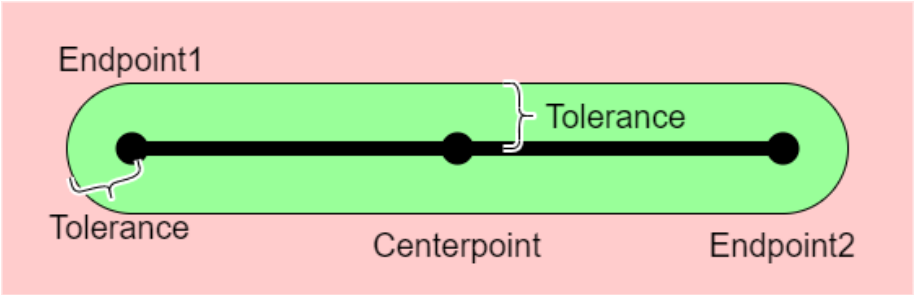


Figure 5: Line tolerance

9.1.3 Goal line tolerance

For the semi-circle portion of the goal line, the same tolerance is applied as for a regular turnpoint cylinder (9.1.1).

$toleranceDistance = outerRadius_{Goal} - radius_{Goal}$

The same tolerance is also applied to the straight portion of the goal line control zone: The tolerance zone's boundaries are here defined by the line l' that is parallel to the goal line but lies closer to the goal line's previous point p by toleranceDistance. Additionally, the boundary is defined by the two circles of radius toleranceDistance around the end points of line l.

See Figure 6.

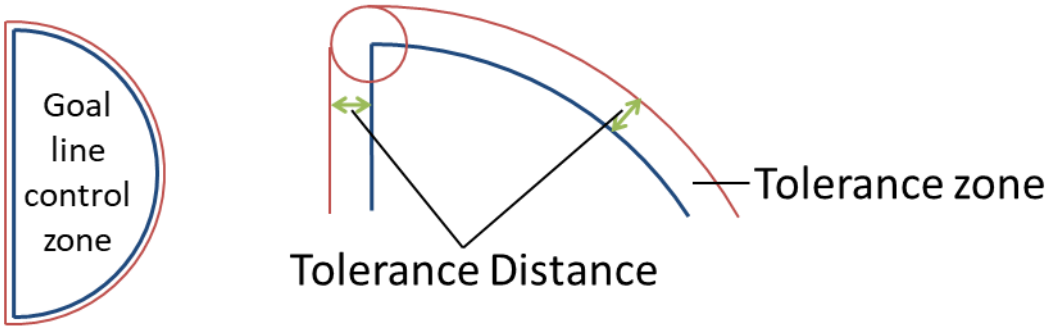


Figure 6: Goal line tolerance

9.2 Reaching a control zone

9.2.1 Reaching a cylinder

To determine whether a pilot reached a specific cylinder in the task, the pilot's track log must show evidence that:

1. The pilot reached the previous control zone in the task definition
2. It contains at least one valid crossing for the cylinder, which is defined as a crossing into or out of the turnpoint's tolerance zone, in any direction, recorded
    a. after the valid crossing of the previous cylinder in the task definition
    b. not earlier than the start time
    c. no later than the task deadline

$\forall i: \exists turnpoint_i: crossings_i = \forall j:$
$(((distance(turnpoint_i.center, trackpoint_{j-1} < innerRadius_i \wedge$
$distance(turnpoint_i.center, trackpoint_j) \geq innerRadius_i) \vee$
$(distance(turnpoint_i.center, trackpoint_{j-1} \geq innerRadius_i \wedge$
$distance(turnpoint_i.center, trackpoint_j) < innerRadius_i))$
$\vee$
$((distance(turnpoint_i.center, trackpoint_{j-1} \leq outerRadius_i \wedge$
$distance(turnpoint_i.center, trackpoint_j) > outerRadius_i) \vee$
$(distance(turnpoint_i.center, trackpoint_{j-1} > outerRadius_i \wedge$
$distance(turnpoint_i.center, trackpoint_j) \leq outerRadius_i)))$

Crossing time for each crossing is the time at which the corresponding tracklog point was recorded.

$$crossing_{i,j}.time = trackpoint_j.time$$

Given all *n* crossings for a turnpoint cylinder, sorted in ascending order by their crossing time, the time when the cylinder was reached is determined.

In Races with single start gate, the reaching time for a turnpoint is always the first crossing time after the start gate time, and after the reaching time of the previous turnpoint.

$reachingTime_i = crossing_{i,n}.time:$
$(n = min(m): crossing_m.time > reachingTime_{i-1})$

In Races with multiple start gates and in Time Trials, we must first determine which flight segment resulted in the biggest covered distance. Then the reaching time for SSS is the last SSS crossing time that is before all the subsequent turnpoint's reaching times within that segment. The reaching time for any turnpoint after SSS is that turnpoint's first crossing time after the previous turnpoint's reaching time.

$within\ the\ flight\ segment\ that\ gives\ the\ longest\ distance:$

$turnpoint_i = SSS: reachingTime_i = crossing_{i,n}.time$
$(n = max(m): crossing_{i,m}.time < reachingTime_{i+1})$
$turnpoint_i \neq SSS: reachingTime_i = crossing_{i,n}.time:$
$(n = min(m): crossing_{i,m}.time > reachingTime_{i-1})$

9.2.1.1 Start time

In Races tasks the start time for pilots who reached SSS after the first start gate time is equal to the last start gate time before their SSS reaching time.

$$\forall p : p \in PilotsFlown \wedge \exists reachingTime_{p,SSS} : startTime_p = SSS.gateTime_n :$$
$$n = \max(m) : SSS.gateTime_m < reachingTime_{p,SSS}$$

In Time Trials, the pilot's start time is equal to their reaching time of SSS.

$$\forall p : p \in PilotsFlown \wedge \exists reachingTime_{p,SSS} : startTime_p = reachingTime_{p,SSS}$$

9.2.2 Reaching a line

If there is a tracklog point closer to the line than the tolerance - the line control zone is considered as reached and the pilot can continue to fly to the next control zone in the task.

A line control zone may also be validated if the tracklog contains two consecutive points - one on one side of the line and the other on the other side of the line. Timestamps of the two tracklog points must also be evaluated - the speed that would be required to fly around the line and still set those two points must be greater than 120 km/h.

9.2.2.1 Example

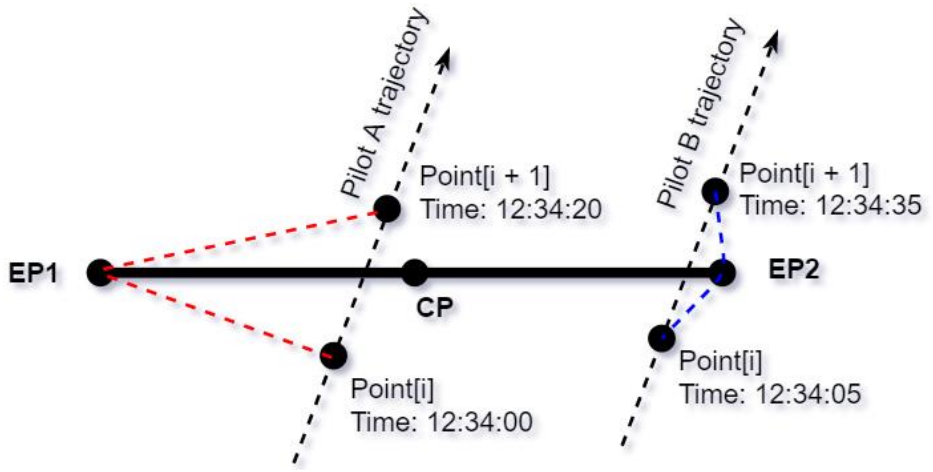


Figure 7: Sample line trajectories

Figure 7 shows the trajectories of two pilots crossing a line. They both do not have a point in the tolerance zone - the validation must be done by evaluating times and distances.

Pilot A has two points (Point[i] and Point[i + 1]) on the two opposite sides of the line. The time between the two points is 20 seconds. The length of the red dashed line is 2.0 km. Therefore, the speed required to go around the line is 2000 m / 20 s = 100 m/s = 360 km/h.

Pilot B has also two points on the two sides of the line, the time between the two points is 30 seconds and the length of the blue line is 800 m. Therefore, the speed required to go around the line is 800 m / 30 s = 26.67 m/s = 96 km/h.

For pilot A, the line is validated (speed > 120 km/h), but for pilot B, the line is **not** validated (speed < 120 km/h).

9.2.3 Reaching a goal line

If goal is defined as a line, with its semi-circular control zone at the line's backside when following the optimized route, it is reached if:

1. The line segment between two subsequent tracklog points intercepts the outer boundaries of the tolerance zone as defined in 9.1.3. Therefore, Goal can be reached from any direction.
2. The first of the two tracklog points shows the pilot as in flight.

9.2.3.1 Physical goal line

If a physical line is used, crossing either the virtual or the physical goal line counts as having reached goal. An official observation (through a goal marshal or similar) of a pilot crossing the line in flight overrules a negative goal crossing decision based on the pilot's tracklog. Not crossing a physical goal line for obvious safety reasons must be considered in the pilots' favour.

	The physical goal line is crossed when the hang glider's nose cuts the line, in the correct direction, before a landing is made.
	The physical goal line is crossed when the paraglider pilot's leading foot cuts the line, in the correct direction, before a landing is made.

9.3 Flown distance

To determine the distance of a pilot's flight, given the pilot's tracklog, we determine for each point where the pilot is still flying the remaining distance to goal from that point, considering any previously reached control zones (9.2). For this the same method is used as for calculating the task distance (7).Then we calculate the flight distance as the task distance minus the smallest of those remaining distances.

If a pilot flies less than minimum distance, they will be scored for their minimum distance. This also applies to pilots who are not able to produce a valid tracklog, but for whom launch officials verify launch within the launch window.

If a pilot reaches goal, they will be scored for the task distance.

$$task = \begin{Bmatrix} preElement_1, \dots, preElement_m, \\ startOfSpeedSection, \\ element_1, \dots, element_n, \\ endOfSpeedSection \\ postElement_1, \dots, postElement_n, \\ goal.point \end{Bmatrix}$$

$$\forall point_i \in track: isFlying(point_i):$$
$$remainingTask(point_i) = point_i + \left(task - reachedElements(point_i)\right)$$
$$remainingDistance(point_i) = \begin{cases} goal.type = line: \text{RouteOptimizer}\left(remainingTask(point)\right).optimizedDistance \\ goal.type = cylinder: \text{RouteOptimizer}\left(remainingTask(point)\right).optimizedDistance - goal.radius \end{cases}$$

$$\forall p: p \in PilotsLandingBeforeGoal: bestDistance_p = max(minimumDistance, taskDistance - min(\forall track_p.point_i: isFlying\left(track_p.point_i\right): remainingDistance(track_p.point_i)))$$

$$\forall p: p \in PilotsReachingGoal: bestDistance_p = taskDistance$$

9.4 Time for speed section

The time a pilot took to fly the speed section is determined by his start time (which is influenced by the task's start procedure and the time he crossed the start of speed section cylinder) and the time when he

crossed the end of speed section after reaching all previous turn points. The smallest unit for time measurement is one second.

Pilots who do not reach the end of speed section cylinder do not get a time.

$$\forall p : p \in PilotsReachingESS : time_p = timeAtESS_p - startTime_p$$

9.4.1 Best time

The best time achieved in a task, which is used to calculate time validity (see 10.3) and pilots' time points (see 12.2) is defined in hang-gliding as the shortest time of all pilots who reached ESS:

$$BestTime = \forall p \in PilotsReachingESS: Min(time_p)$$

In paragliding, a pilot's time is only considered for best time if they reached goal, since this is required for them to receive the corresponding time points:

$$BestTime = \forall p \in PilotsReachingGoal: Min(time_p)$$

10 Task Validity

The task validity is a value between 0 and 1 and measures how suitable a competition task is to evaluate pilots' skills. It is calculated for each task after the task has been flown, by multiplying the three validity coefficients: Launch validity, distance validity, and time validity.

$$TaskValidity = LaunchValidity * DistanceValidity * TimeValidity$$

10.1 Launch Validity

Launch Validity in paragliding competitions is determined by the Nominal Launch value (set at 96%), and the percentage of pilots who launch. If 96% or more of the pilots present at take-off launch, Launch Validity is 1. It decreases as the percentage of launching pilots drops below this threshold.

This mechanism serves as a safety feature. If a significant number of pilots choose not to launch due to unfavourable or dangerous conditions, the points awarded to those who do launch are reduced.

For scoring purposes, 'Pilots present' includes all pilots not marked as 'Absent' (ABS), comprising those who took off and those present but did not fly (DNF). DNF status should be assigned carefully, distinguishing between pilots who choose not to fly due to conditions and those absent for other reasons like illness.

$$NominalLaunch = 96\%$$

$$LVR = min(1, \frac{NumberOfPilotsFlying}{NumberOfPilotsPresent * NominalLaunch})$$

$$LaunchValidity = 0.028 * LVR + 2.917 * LVR^2 - 1.944 * LVR^3$$

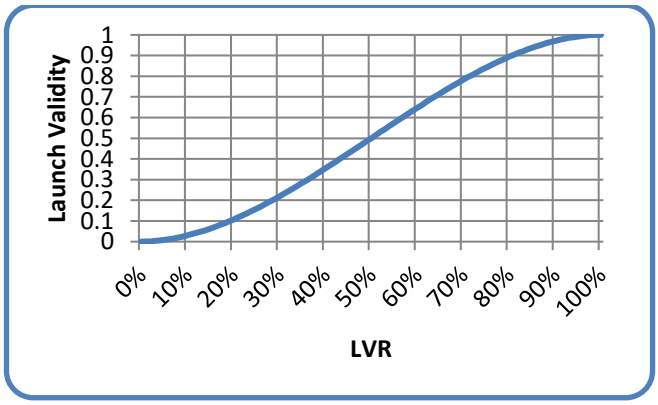


Figure 8: Launch validity curve

10.2 Distance Validity

Distance validity depends on the competition parameter Nominal Distance, the Nominal Goal value (set at 30%), the longest distance flown, and the sum of all distances flown beyond minimum distance. If the task distance is quite short in relation to nominal distance, the day is probably not a good measure of pilot skill because there would not be many decisions to make.

If a task is longer than nominal distance, the day will not be devalued because of distance validity, even if the nominal goal parameter value is not achieved, as long as a fair percentage of pilots fly a good distance. This sounds like a vague statement, but the task setter should try to set tasks that are reasonable for the day and achievable. If everyone lands in goal, you must ask if this was a valid test of skill - it probably was if the fastest time and the distance flown were reasonably long. If everyone lands short of goal, was it an unsuitable task but still a good test of pilot skill? You also can have the case where a task that is shorter than nominal distance, has a distance validity of almost 1. This will happen when a large percentage of

the pilots fly a large percentage of the course but, in this case, you still have a practical devaluation because there will be little spreading between pilots' scores.

In the formula below, 'p' denotes an individual pilot.

$NomGoal = 30\%$

$$SumOfFlownDistancesOverMinDist = \sum_p max(0, FlownDist_p - MinDist)$$

$$NomDistArea = \frac{((NomGoal + 1) * (NomDist - MinDist)) + max(0, (NomGoal * (BestDist - NomDist)))}{2}$$

$$DVR = \frac{SumOfFlownDistancesOverMinDist}{NumPilotsFlying * NomDistArea}$$

$$DistanceValidity = min(1, DVR)$$

10.3 Time Validity

Time validity depends on the fastest time to complete the speed section, in relation to nominal time. If the fastest time to complete the speed section is longer than nominal time, then time validity is always equal to 1.

If the fastest time is quite short, the day is probably not a good measure of pilot skill because there would not be many decisions to make and, because of this, luck can distort scores as there will be little possibility to recover any accidental loss of time.

If no pilot finishes the speed section, then time validity is not based on time but on distance: The distance of the pilot who flies the furthest in relation to nominal distance is then used to calculate the time validity the same way as if it was the time.

If one pilot reached ESS: $TVR = min(1, \frac{BestTime}{NominalTime})$

If no pilot reached ESS: $TVR = min(1, \frac{BestDistance}{NominalDistance})$

$$TimeValidity = max(0, min(1, -0.271 + 2.912 * TVR - 2.098 * TVR^2 + 0.457 * TVR^3))$$

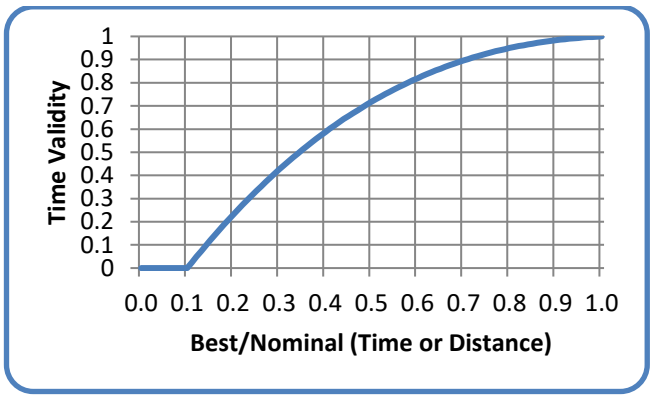


Figure 9: Time validity curve

11 Points Allocation

The available points for each task are 1000*Task Validity. These points are distributed between distance points, time points, leading points, and arrival points. The distribution depends on the percentage of pilots who reached goal before the task deadline, compared to pilots who launched, as well as the chosen goal form. It is expressed in terms of weight factors for each of the four categories: Distance weight, time weight, leading weight, and arrival weight. Weight factors are always between 0 and 1. A weight factor of 0.5 for distance, for example, means that 50% of the day's available overall points are available for distance points.

The parameter *LeadingTimeRatio* is set for each task, with a value between 0 and 26%.

The default LeadingTimeRatio in hang-gliding is 17.5%

The default LeadingTimeRatio in paragliding is 26%.

$$GoalRatio = \frac{NumberOfPilotsInGoal}{NumberOfPilotsFlying}$$

$$DistanceWeight = 0.9 - 1.665 * GoalRatio + 1.713 * GoalRatio^2 - 0.587 * GoalRatio^3$$

$$LeadingWeight = (1 - DistanceWeight) * LeadingTimeRatio$$

$$HG\ Class \neq 2: ArrivalWeight = (1 - DistanceWeight) * 12.5\%$$

$$HG\ Class = 2: ArrivalWeight = 0$$

$$GoalRatio = 0: LeadingWeight = (1 - DistanceWeight)$$

$$GoalRatio > 0: LeadingWeight = (1 - DistanceWeight) * LeadingTimeRatio$$

$$ArrivalWeight = 0$$

$$TimeWeight = 1 - DistanceWeight - LeadingWeight - ArrivalWeight$$

$$AvailableDistancePoints = round(1000 * TaskValidity * DistanceWeight, 0)$$
$$AvailableTimePoints = round(1000 * TaskValidity * TimeWeight, 0)$$
$$AvailableLeadingPoints = round(1000 * TaskValidity * LeadingWeight, 0)$$
$$AvailableArrivalPoints = round(1000 * TaskValidity * ArrivalWeight, 0)$$

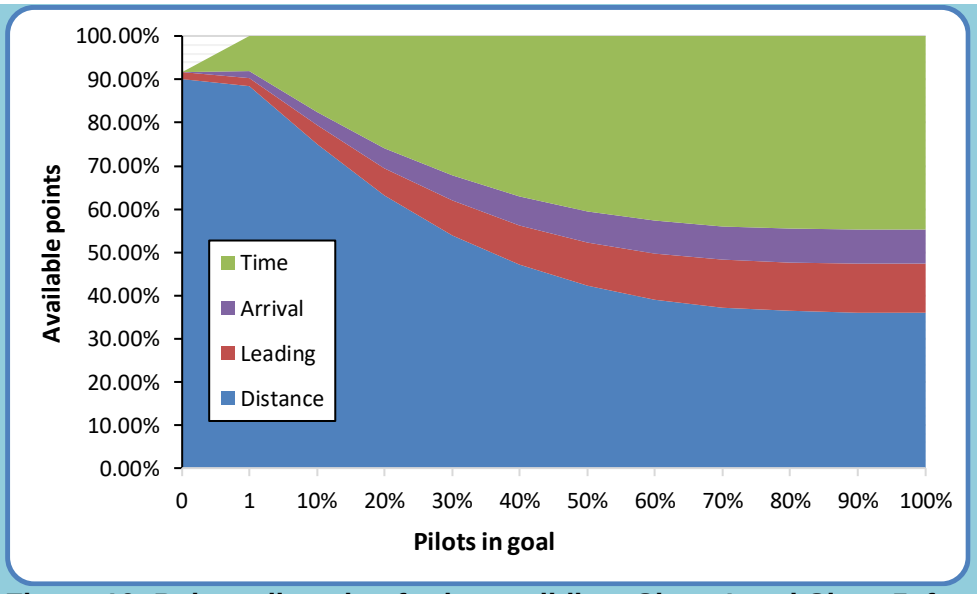


Figure 10: Points allocation for hang gliding, Class 1 and Class 5, for default *LeadingTimeRatio* of 17.5%

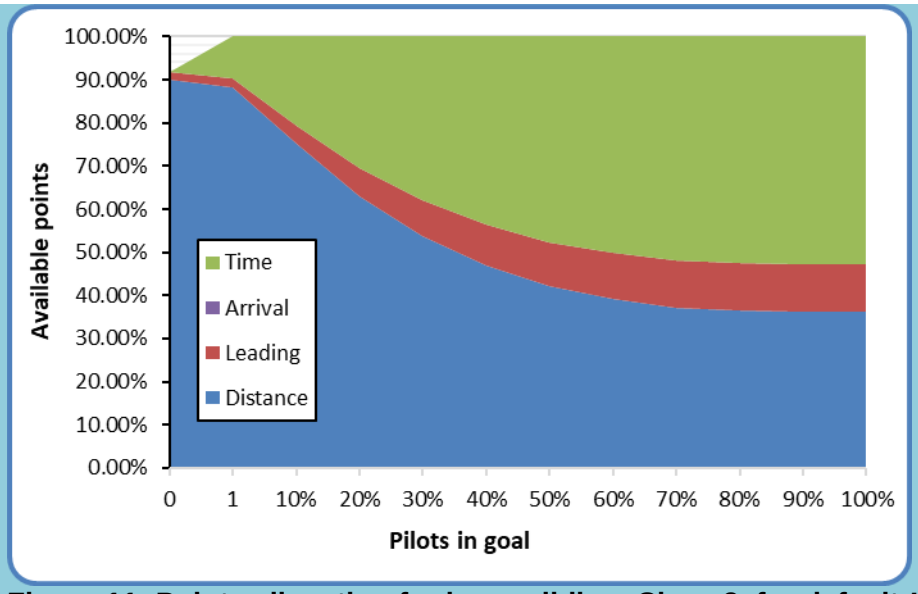


Figure 11: Points allocation for hang gliding, Class 2, for default *LeadingTimeRatio* of 17.5%

From the above it follows that in hang-gliding, if nobody reaches ESS, then a maximum of 900 points are available for distance and 18 points for leading but, of course, no points for time nor arrival.

$numberOfPilotsAtESS = 0$:

$AvailableDistancePoints = round(1000 * TaskValidity * DistanceWeight, 0)$

$AvailableTimePoints = 0$

$AvailableLeadingPoints = round(1000 * TaskValidity * LeadingWeight, 0)$

$AvailableArrivalPoints = 0$

$Max(AvailableDistancePoints) = 900$

$Max(availableLeadingPoints) = 18$

$Max(availableTotalPoints) = 918$

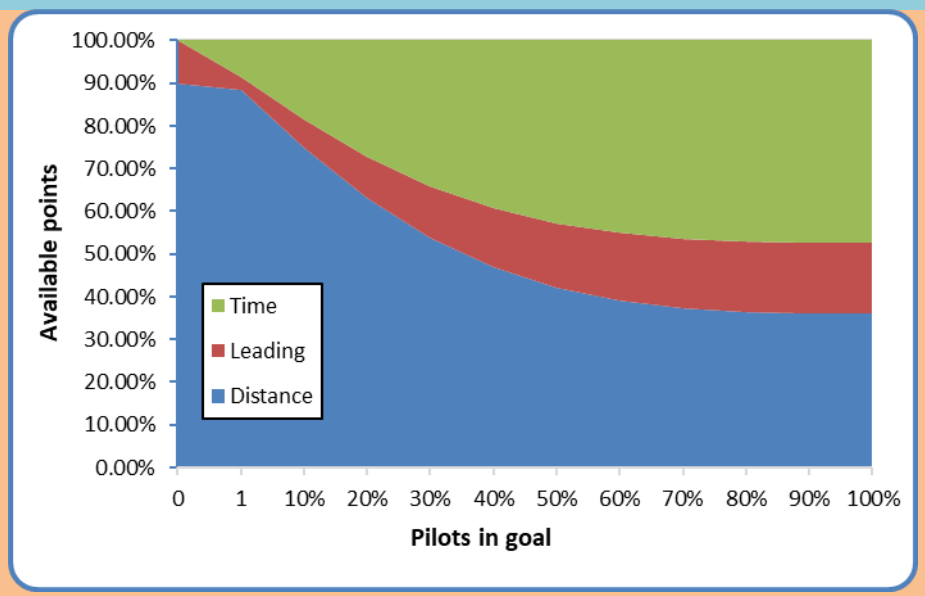


Figure 12: Points allocation for paragliding, for default *LeadingTimeRatio* of 26%

12 Pilot score

Each pilot's score is the sum of that pilot's distance, time, leading and arrival points, rounded to one decimal place:

$$\forall p: p \in PilotsLaunched: TotalScore_p = round(DistancePoints_p + TimePoints_p + LeadingPoints_p + ArrivalPoints_p, 1)$$

12.1 Distance points

The distance considered for each pilot to calculate distance points is that pilot's best distance along the course line, up until the pilot landed or the task deadline was reached, whichever comes first.

$$Distance_p = \max(\text{MinimumDistance}, \text{taskDistance} - \min(\forall track_p.point_i: shortestDistanceToGoal(track_p.point_i)))$$

Distance points are rounded to one decimal place.

- One half of the available distance points are assigned to each pilot linearly, based on the pilot's distance flown in relation to the best distance flown in the task. The other half is assigned taking into consideration the difficulty of the kilometers flown.
- $LinearFraction_p = \frac{Distance_p}{2 * BestDistance}$
- $iDist10_p = int(Distance_p * 10)$
  $DifficultyFraction_p = DiffScore_{iDist10_p} + ((DiffScore_{iDist10_p+1} - DiffScore_{iDist10_p}) * (Distance_p * 10 - iDist10_p))$
  $DistancePoints_p = (LinearFraction_p + DifficultyFraction_p) * AvailableDistancePoints$

- The available distance points are assigned to each pilot linearly, based on the pilot's distance flown in relation to the best distance flown in the task.
- $DistancePoints_p = \frac{Distance_p}{BestDistance} * AvailableDistancePoints$

In the case of a stopped task, a pilot's distance may be increased by an altitude bonus (see 12.3.2).

12.1.1 Difficulty calculation

- To measure the relative difficulty of each 100 meters of the task, we consider the number of pilots who landed in the successive few kilometers, and the distance flown.
- In a first step, for each 100-meter section of the task, the number of pilots who landed in that section is counted. Pilots who landed before minimum distance are counted as having landed at minimum distance. Only pilots who landed out are considered for this calculation, pilots who reached goal are not counted.
  $\forall i: i < \text{int}(MinDist*10): PilotsLanded_i = 0$
  $PilotsLanded_{\text{int}(MinDist*10)} = \sum_{\forall Pilot: \text{int}(Pilot.Distance*10) \le \text{int}(MinDist*10)} 1$
  $\forall i: i > \text{int}(MinDist*10) \,\&\, i \le \text{int}(MaxDist*10): PilotsLanded_i = \sum_{\forall q: q \in PilotsLandedOut: \text{int}(Distance_q*10) = i} 1$
- Then the difficulty for each 100-meter section of the task is calculated by counting the number of pilots who landed further along the task. If 100 pilots land out on a flight of 100 km, the next 3 km are considered. If 10 pilots land out in 100 km, the next 30 km are considered. The variable LookAheadDist contains the number of 100-meter slots to look ahead for this.

$$LookAheadDist = \max(30, round(\frac{30 * BestDistanceFlown}{NumberOfPilotsLandedOut}, 0))$$

$$\forall i :\leq \text{int}(MaxDist*10) : Difficulty_i = \sum_{j=i}^{j=\min(i+LookAheadDist, \text{int}(BestDistanceFlown*10))} PilotsLanded_j$$

$$SumOfDifficulty = \sum_i Difficulty_i$$

Relative difficulty is then calculated by dividing each 100-meter slot's difficulty by twice the sum of all difficulty values.

$$\forall i : i \leq \text{int}(MaxDist*10) : RelativeDifficulty_i = \frac{Difficulty_i}{2 * SumOfDifficulty}$$

Finally, we can calculate the difficulty score percentage for each 100-meter slot.

$$\forall i : i \leq \text{int}(MinDist*10) : DiffScore_i = \sum_{j=0}^{j=\text{int}(MinDist*10)} RelativeDifficulty_j$$

$$\forall i : i > \text{int}(MinDist*10) \,\&\, i < \text{int}(BestDistanceFlown*10) : DiffScore_i = \sum_{j=0}^{j=i} RelativeDifficulty_j$$

$$\forall i : i \geq \text{int}(BestDistanceFlown*10) : DifffScore_i = 0.5$$

The difficulty calculation does not apply to paragliding.

12.1.1.1 Example for difficulty calculation

For an example of how the difficulty calculation works, see Figure 13: Note how the slope of the green curve (the total Distance points) becomes steeper before an area where many pilots landed and flatter just after. The red circles show these areas before the big group at the 41 km mark, and after the 46 km mark. There are two reasons for this:

For safety and retrieval reasons, we do not want to encourage pilots to fly only a short distance past a group of landed pilots.

If a pilot lands somewhere, he or she probably got into trouble just before, and then glided a while before landing.

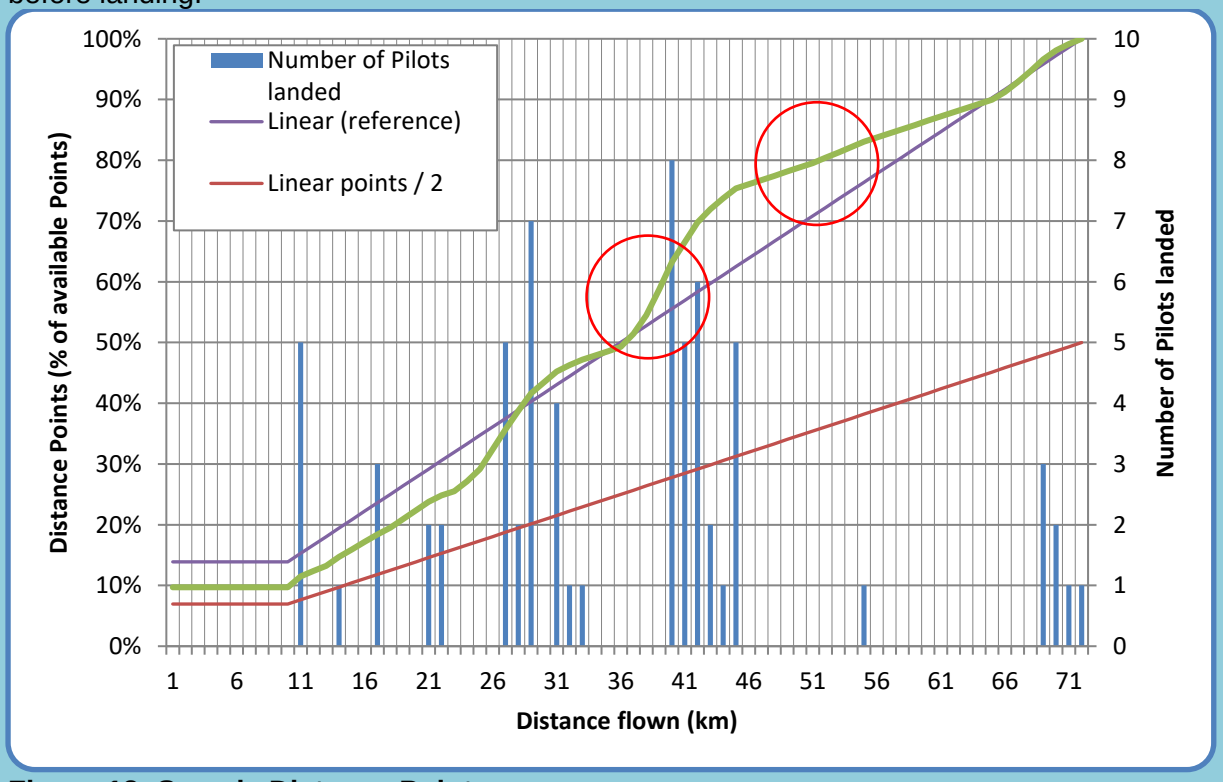


Figure 13: Sample Distance Points

12.2 Time points

Time points are assigned to the pilot as a function of best time and pilot time – the time the pilot took to complete the speed section. Slow pilots will get zero points for speed if their time to complete the speed section is equal to or longer than the fastest time plus the square root of the fastest time. All times are measured in hours.

$$SpeedFraction_p = max(\,0{,}1 - \sqrt[6]{\frac{(Time_p - BestTime)^5}{\sqrt{BestTime}}}\,)$$

$$TimePoints_p = SpeedFraction_p * AvailableTimePoints$$

Time points are rounded to one decimal place.

12.2.1 Examples

For three examples of Time Point distributions for tasks with different best times, see Figure 14 and Table 2.The best time is defined as the time of the fastest pilot over the speed section who also reached the goal.

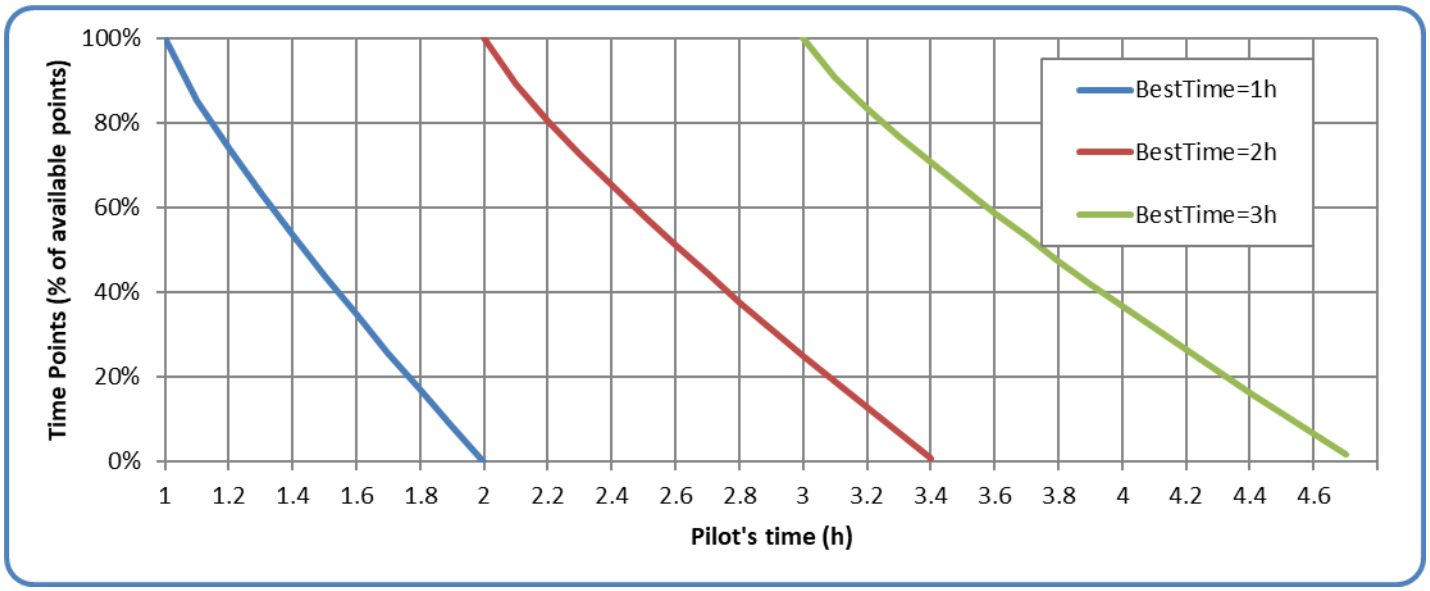


Figure 14: Sample time point distributions

Fastest Time	80% Time Points time	50% Time Points time	0 Time Points time
1:00	1:08:42	1:26:07	2:00:00
2:00	2:12:18	2:36:56	3:24:51 (3.41 hours)
3:00	3:15:04	3:45:14	4.43:55 (4.73 hours)

Table 2: Sample time point distribution (times in hours:minutes:seconds)

12.3 Leading points

Leading points are awarded to encourage pilots to start early and to reward the risk involved in flying in the leading group. Pilots will get leading points even if they landed before goal or the end of speed section.

$$LC_{\min} = \min(\forall p : p \in PilotsFlown : LC_p)$$

$$LeadingFactor_p = \max(0, 1 - \sqrt[3]{\frac{(LC_p - LC_{\min})^2}{\sqrt{LC_{\min}}}})$$

$$LeadingPoints_p = LeadingFactor_p * AvailableLeadingPoints$$

Leading points are rounded to one decimal place.

To get an impression of the way leading points are awarded depending on a task's minimal leading coefficient, see Figure 15.

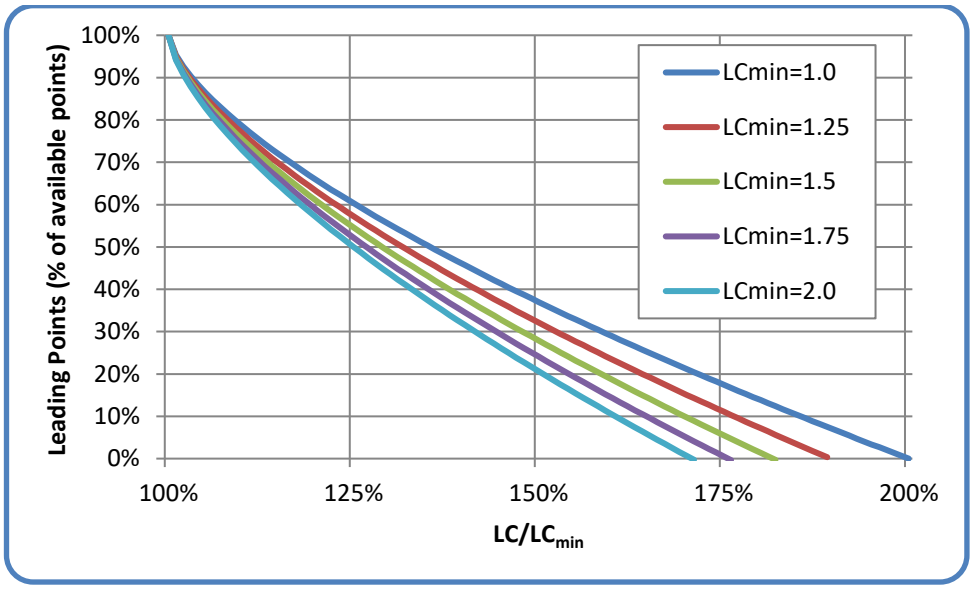


Figure 15: Leading points for various LC_{min}

12.3.1 Leading coefficient

Each started pilot's track log is used to calculate the leading coefficient (LC), by calculating the area underneath a graph defined by each track point's time, and the distance to ESS at that time. The times used for this calculation are given in seconds from the first start gate time (as defined for the task), to the time when the last pilot reached ESS. For pilots who land out after the last pilot reached ESS, the calculation keeps going until they land. The distances used for the LC calculation are given in kilometres and are the distance from each point's position to ESS, starting from SSS, but never more than any previously reached distance. This means that the graph never "goes back": even if the pilot flies away from goal for a while, the corresponding points in the graph will use the previously reached best distance towards ESS.

Calculation of the leading coefficient (LC) for each pilot follows this formula:

$$bestTrackPoint_p = \text{trackPointWithShortestDistanceToESS}(trackPointsInSS_p)$$

$$\text{minToESS}(tp_0) = speedSectionDistance$$

$$\forall i : i > 0, tp_i \in trackPointsInSS_p : \text{minToESS}(tp_i) = \min\big(\text{minToESS}(tp_{i-1}), \text{distToESS}(tp_i)\big)$$

$$\text{taskTime}(trackPoint) = \min(trackpoint.time, taskDeadline) - firstTaskStartTime$$

$$maxTime = \min(\max(lastOutlandingTime, lastESStime), taskDeadline)$$

- $leadingArea_p = \sum_{i:tp_i \in trackPointsInSS_p} (\text{minToESS}(tp_{i-1})^2 - \text{minToESS}(tp_i)^2) * \text{taskTime}(tp_i)$
- $missingArea_p = maxTime * \text{minToESS}(bestTrackPoint_p)^2$
- $LC_p = \frac{leadingArea_p + missingArea_p}{1800 * speedSectionDistance^2}$

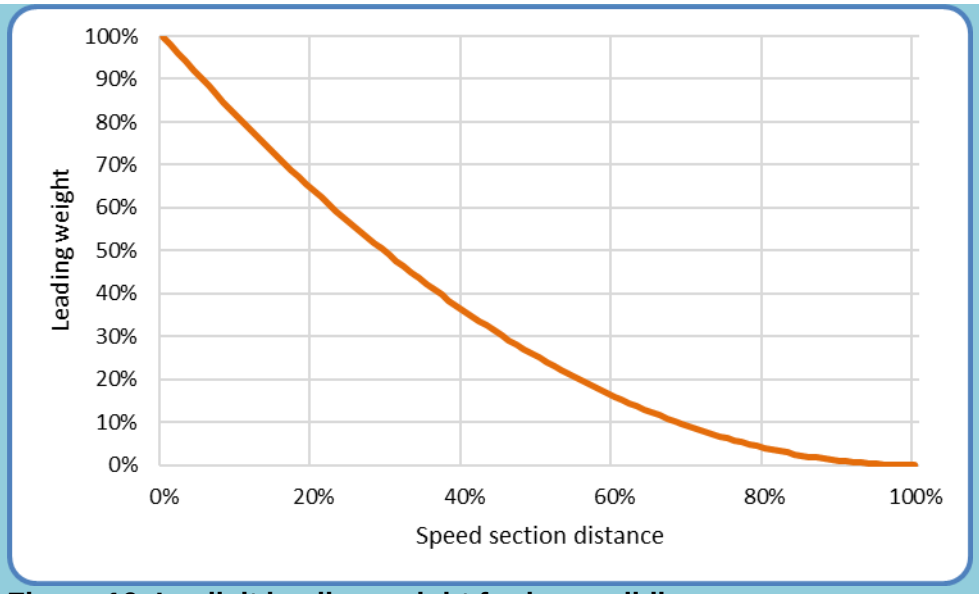


Figure 16: Implicit leading weight for hang-gliding

$leadingArea_p = \sum_{\forall i: tp_i \in trackPointsInSS_p} \text{minToESS}(tp_i) * \text{taskTime}(tp_i) * \int_{done(tp_{i-1})}^{done(tp_i)} \text{weight(x)}\ dx$

$missingArea_p = \text{minToESS}(bestTrackPoint_p) * \text{maxTime} * \int_{done(bestTrackPoint_p)}^{1} \text{weight(x)}\ dx$

$done(p) = 1 - (\text{minToEss}(p))/speedSectionDistance$

$LC_p = \frac{leadingArea_p + missingArea_p}{1800 * speedSectionDistance}$

$\text{weight}(v) = \text{weightRising}(1-v) * \text{weightFalling}(1-v)$

$\text{weightRising}(v) = (1 - 10^{9*v-9})^5$

$\text{weightFalling}(v) = (1 - 10^{-3*v})^2$

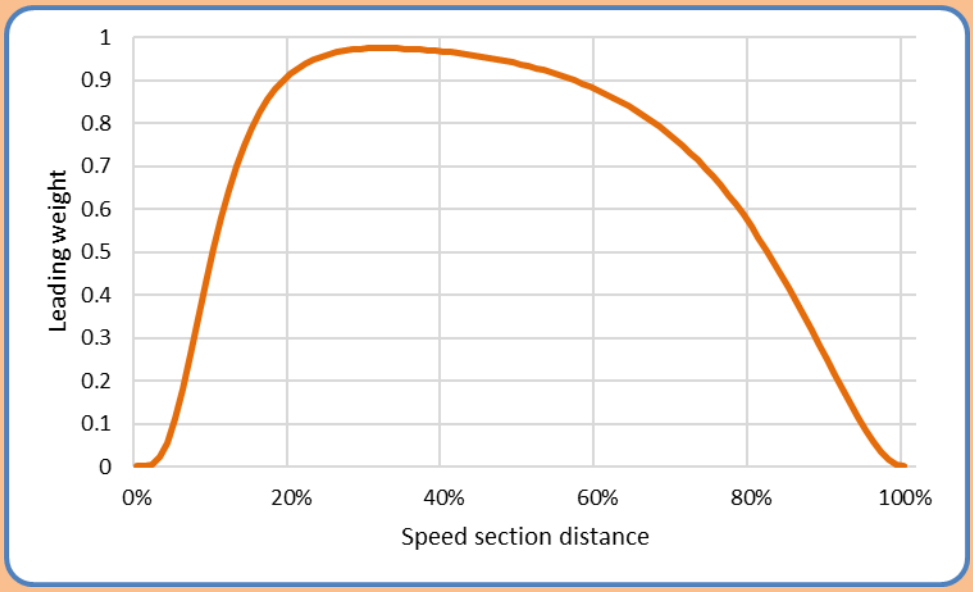


Figure 17: Leading weight for paragliding

12.3.2 Example

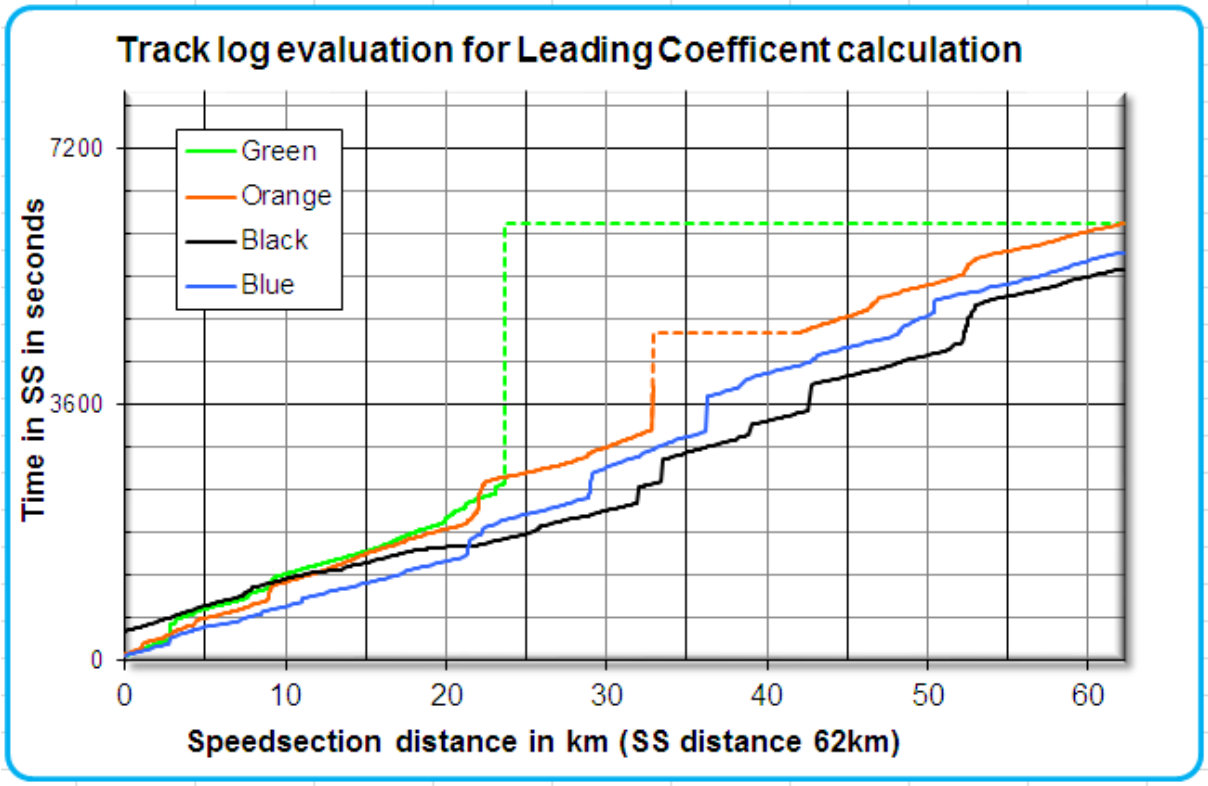


Figure 18: Sample track log graphs for LC calculation

Blue was the first to enter the speed section, but Black was the first pilot to cross the end of speed section. Green started at the same time as Blue, but landed short, after about 23km and just over 40 minutes of flight inside the speed section.

Black was fastest, therefore will get the most time points, but he started late, probably had pilots out front to show the way during the first 22km but was leading after that.

If a pilot lands along the course (Green), or if his track log is interrupted (Orange), his track log is completed as shown by the dotted lines: Missing parts are calculated as if the dotted line was the actual track log, so LC becomes bigger, lowering the leading points for that pilot, compared to a track where that part is not missing. A pilot landing just short of goal will be less penalised and could even get full leading points if he led for a long while.

The pilot who used best the earliest part of the day (i.e., Black, who has the smallest area below the track log graph) gets all the available leading points, while the others get their points according to the same formula used for the time points for the same reasons. If the task in the example is fully valid, and 30% of pilots reached goal, then Black will get all of the available 81 leading points and full time points, as he was fastest; Blue gets 45 leading points because he started early but was slower; Orange receives only 18 leading points as he was slow and had a gap in his track log; Green gets 0 points even though he started early, because he was the slowest and landed fairly short.

12.4 Arrival points

Arrival points depend on the position at which a pilot crosses ESS: The first pilot completing the speed section receives the maximum available arrival points, while the others are awarded arrival points according to the number of pilots who reached ESS before them. The last pilot to reach ESS will always receive at least 20% of the available arrival points.

$$AC_p = 1 - \frac{PositionAtESS_p - 1}{NumberOfPilotsReachingESS}$$

$$ArrivalFraction_p = 0.2 + 0.037 * AC_p + 0.13 * AC_p^{\,2} + 0.633 * AC_p^{\,3}$$

$$ArrivalPoints_p = ArrivalFraction_p * AvailableArrivalPoints$$

Arrival points are rounded to one decimal place.

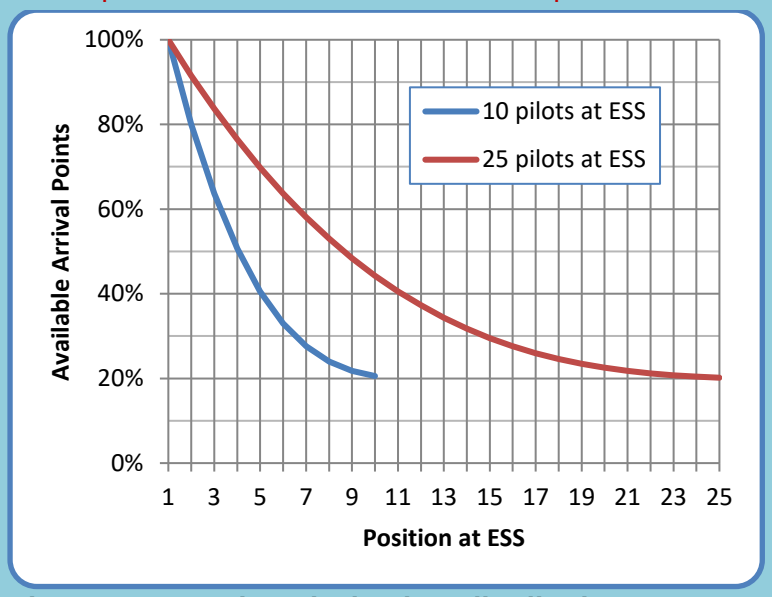


Figure 19: Sample arrival points distributions

No arrival points are awarded in Paragliding.

13 Special cases

13.1 ESS but not goal

In a task where ESS and goal are not identical, a pilot may reach ESS, but not goal.

Reaching goal is seen as 'validating' one's speed section performance. A pilot who does not reach goal after reaching ESS will lose a portion of his time points, as defined by the scoring system penalty parameter for this situation. He will also score full distance points for the distance covered and his full leading points. The timepoint penalty for not reaching goal is seen as a safety measure, since it encourages pilots to plan their final glide to ESS with enough altitude to safely reach goal.

For paragliders the scoring system parameter is to be set at 0% (i.e. no time points awarded) as this discourages high-speed final glides low to the ground.

For hang gliders the default scoring system parameter of 80% is recommended but can be changed by the local regulations to suit particular sites.

- $\forall p: p \in PilotsLandedBetweenESSandGoal: TotalScore_p = DistancePoints_p + LeadingPoints_p + 0.8 * (TimePoints_p + ArrivalPoints_p)$
- $\forall p: p \in PilotsLandedBetweenESSandGoal: TotalScore_p = DistancePoints_p + LeadingPoints_p + 0 * (TimePoints_p)$

13.2 Early start

An early start occurs if a pilot's last SSS control zone crossing occurred before the first (or only) start gate time.

In paragliding, pilots who perform an early start are only scored for the distance between the launch point and the SSS control zone, as calculated when determining the complete task distance (see 7.2).

In hang-gliding, the so-called "Jump the Gun"-rule applies: If the early start occurred within a time that is close to the first (or only) start gate time, the pilot is scored for his complete flight, but a penalty is then applied to his total score.

The penalty calculation is based on two values X and Y, which are set in S7A, but can be changed at the task briefing (presumably by the meet director and/or the task committee). For each X seconds a pilot starts early, he incurs a 1-point penalty, up to a maximum of Y seconds. If a pilot starts more than Y seconds early, he will only be scored for minimum distance.

$X_{default} = 2$

$Y_{default} = 300$

$timeDiff_p = firstStartGateTime - lastStartTime_p$

$timeDiff_p <= 0 : jumpTheGunPenalty_p = 0$

$timeDiff_p > Y : jumpTheGunPenalty_p = 0, totalScore_p = scoreForMinDistance$

$0 < timeDiff_p <= Y : jumpTheGunPenalty_p = \frac{timeDiff_p}{X}$

$totalScore_p = \max(totalScore_p - jumpTheGunPenalty_p, scoreForMinDistance)$

13.3 Stopped tasks

13.3.1 Stop task time

A task can be stopped at any time by the meet director. The time when a stop was announced for the first time is the “task stop announcement time”. This time must be recorded to score the task appropriately. For scoring purposes, a “task stop” time is calculated, by “scoring back”, or deducting a number of minutes from the announcement time. Pilots’ flights will only be scored up to this task stop time.

$scoreBackTime = 15\ min.$

$scoreBackTime = 5\ min.$

$$taskStopTime = taskStopAnnouncementTime - scoreBackTime$$

13.3.2 Minimum duration of stopped tasks

In hang-gliding, stopped tasks will be scored only if they ran for a sufficiently long time.

$$minimumTime = \min\left(1h, \frac{NominalTime}{2}\right)$$

In paragliding, no such minimum time requirement exists. Instead, low-validity stopped tasks will be excluded from the competition results (see chapter 15).

$minimumTime = 0$

13.3.3 Stopped task validity

For stopped tasks, an additional validity value, the Stopped Task Validity, is calculated and applied to the Task Validity.

$$DayQuality_{stopped} = LaunchValidity * DistanceValidity * TimeValidity * StoppedTaskValidity$$

Stopped Task Validity is calculated considering the task duration, the minimum time, task distance, the flown distances of all pilots, the number of launched pilots and the number of pilots still flying at the time when the task was stopped.

$$taskDuration = taskStopTime - \max(\forall p \in StartedPilots: startTime_p)$$
$$stoppedDurationValidity = \ taskDuration \geq mimumumTime ?\ 1 : 0$$

$$stoppedDistanceValidity = \sqrt{\frac{BestDistFlown - avg(\forall i: DistFlown_i)}{DistLaunchToESS - BestDistFlown + 1}} * \sqrt{\frac{stdev(\forall i: DistFlown_i)}{5}}$$

$$stoppedFlyingValidity = \left(\frac{NumPilotsLandedBeforeStopTime}{NumPilotsLaunched}\right)^3$$

$$StoppedTaskValidity = NumberOfPilotsReachedESS > 0\ ?\ 1 : stoppedDurationValidity * min(1, stoppedDistanceValidity + stoppedFlyingValidity)$$

13.3.4 Scored time window

For stopped Races with a single start gate, scoring considers the same time window for all pilots: The time between the race start and the task stop time.

$$typeOfTask = Race \wedge numberOfStartGates = 1:$$
$$\forall p: \rightleftarrows\ p \in StartedPilots: \rightleftarrows\ scoreTimeWindow_p = (startTime, taskStopTime)$$

Stopped Races tasks with multiple start gates, as well as stopped Time Trials, must be treated slightly differently: Only the time window available to the last pilot started is considered for scoring. This time

window is defined as the amount of time *t* between the last pilot's start and the task stop time. For all pilots, only this time t after their respective start is considered for scoring.

$typeOfTask \neq Race \vee numberOfStartGates > 1$:
$scoreTime = taskStopTime - max(\forall p: \rightleftarrows\ p \in StartedPilots: \rightleftarrows\ startTime_p)$
$\forall p: \rightleftarrows\ p \in StartedPilots: \rightleftarrows\ scoreTimeWindow_p = (startTime_p, startTime_p + scoreTime)$

This means that if the last pilot started and then flew for, for example, 75 minutes until the task was stopped, all tracks are only scored for the first 75 minutes each pilot flew after taking their respective start.

13.3.5 Time points for pilots at or after ESS

Pilots who were at a position between ESS and goal at the task stop time will be scored for their complete flight, including the portion flown after the task stop time. This is to remove any discontinuity between pilots just before goal and pilots who had just reached goal at task stop time.

A fixed number of points is subtracted from the time points of each pilot that makes goal in a stopped task. This amount is the amount of time points a pilot would receive if he had reached ESS exactly at the task stop time. This is to remove any discontinuity between pilots just before ESS and pilots who had just reached ESS at task stop time.

$typeOfTask = Race \wedge numberOfStartGates = 1$:
$timePointsReduction = timePoints(\ taskStopTime - startTime)$

$typeOfTask \neq Race \vee numberOfStartGates > 1$:
$timePointsReduction = timePoints(\ taskStopTime - max(\forall p: p \in PilotsReachedESS: startTime_p))$

$\forall p: p \in PilotsInGoal: finalTimePoints_p = timePoints_p - timePointsReduction$
$finalDistancePoints = distancePoints_p + timePointsReduction$

13.3.6 Distance points with altitude bonus

To compensate for altitude differences at the time when a task is stopped, a Bonus Distance is calculated for those pilots who still flew at the Task Stop Time:

1. Altitude above goal at Task Stop Time is determined
2. This altitude is multiplied by a Bonus Glide Ratio
3. The resulting Altitude Bonus is added to the task distance covered at Task Stop Time, disregarding any better distances achieved previously.
4. If the Bonus Distance (distance at stop + Altitude Bonus) exceeds the pilot's best distance up to Task Stop Time, it is used for Distance Points calculations, including the difficulty calculations applied in hang-gliding (see 12.1.1). Time Point and Leading Point calculations remain unaffected by the Bonus Distance.


$BonusGlideRatio = 5.0$


$BonusGlideRatio = 2.5\ (from\ GAP\ 2026: 0.0)$

$\forall p: p \in PilotsFlyingAtStopTime: lastPoint_p = track_p.point_{TaskStopTime}$

$\forall p: p \in PilotsFlyingAtStopTime: altitudeBonus_p$
$= \max(0, lastPoint_p.altitude - GoalAltitude) * BonusGlideRatio$

$\forall p: p \in PilotsFlyingAtStopTime: bonusDistance_p$
$= \min(taskDistance, taskDistance - shortestDistanceToGoal(lastPoint_p) + altitudeBonus_p$

$\forall p: p \in PilotsFlyingAtStopTime: ScoredDistance_p = \max(Distance_p, bonusDistance_p)$

13.4 Penalties

Penalties for various actions are defined in the rules. These penalties are either expressed as an absolute number (e.g., “100 points”) or as a percentage (e.g. “10% of the pilot’s score in the task where he performed the punishable action”). The corresponding number of points is then deducted from either the pilot’s task or competition results, depending on the punishable action:

1. For unsporting behaviour, the points are deducted from the pilot’s competition score to calculate their final competition score.
2. For all punishable actions, the points is deducted from the punished pilot’s task score to calculate their final task score.

$$finalScore_p = score_p - absolutePenalty_p$$

$$finalScore_p = score_p * (1 - percentagePenalty_p)$$

Penalties are applied in the following order:

1. “Jump the Gun”-Penalty
2. Percentage penalty or bonus
3. Absolute points penalty or bonus

1. Percentage penalty or bonus
2. Absolute points penalty or bonus

The penalty mechanism can also be used to award bonus points to a pilot for some actions like helping a pilot in distress. In that case the penalty must be given as a negative number.

After the application of penalties, scores are again rounded to one decimal place. The lowest score a pilot can attain in a task, regardless of any incurred penalties, is zero points.

14 Task ranking

All task scores are given with one decimal place.

Definitions of types of task ranking are in Section 7A-5.2.4.

- Overall task ranking
- Female task ranking
- Nation task ranking

15 Competition ranking

All competition scores are given with one decimal place.

In paragliding competitions, the results of stopped tasks are included in the competition ranking only if the Task Validity is 0.05 or higher.

Definitions of types of Competition ranking are in Section 7A-5.2.5.

- Overall competition ranking
- Female competition ranking
- Nation competition ranking
- Ties

16 FTV – Fixed Total Validity

Fixed Total Validity (FTV) is a procedure to score pilots on their best task performances, rather than all their tasks. Fixed Total Validity means the sum (total) of available points (validity) is set (fixed) to the same value for each competitor. It takes into account the competition parameter FTV_factor.

$$CalculatedFTV = (1 - FTV\_factor) * \sum_{t:Task} \frac{WinnerScore_t}{1000}$$

To calculate a pilot's FTV score, for all his flights:

1. Calculate a performance percentage for each day by dividing the pilot's day score by the day winner's points
2. Arrange all flights in descending order of performance percentage
3. Total up the flights' raw day scores (not performance percentages) in order of performance percentage until the sum of validities for those scores reaches the pre-decided Fixed Total Validity value.

If the last score added takes that pilot's total validity above the Fixed Total Validity, then only a fraction of that score is used so that the pilot's total validity is equal to the Fixed Total Validity.

$$\forall t: t \in ScoredTasks: Performance_{p,t} = \frac{Score_{p,t}}{WinnerScore_t}$$

$$SortedPerformance_p = sortDescending(\forall t: t \in ScoredTasks: Performance_{p,t})$$

$$OrderedValidities_p = orderByPerformance(SortedPerformance_p, \forall t: t \in ScoredTasks: \frac{WinnerScore_t}{1000})$$

$$OrderedScores_p = orderByPerformance(SortedPerformance_p, \forall t: t \in ScoredTasks: Score_{p,t})$$

$$FTV\_Score_p = \sum_{u=0}^{numberOfTasks} round\left(\begin{matrix} OrderedScores_{p,u} * \\ max(1, \frac{min(0, CalculatedFTV - (u > 0: \sum_{v=0}^{min(0,u-1)} OrderedValidities_{p,v}))}{OrderedValidities_{p,u}}) \end{matrix}, 1\right)$$

Annex A Implementation of GeodesicToCartesian & CartesianToGeodesic

This annex contains:

1. A sample of how to use the library PROJ[19] to create the converters to convert between WGS84 (geodesic) and Cartesian coordinates, as specified in the main document above.
2. A sample implementation for the converters, to be used in systems where PROJ is not available.

A.1 Use of PROJ

The following is in C#, but PROJ is available in several other programming languages as well.

```
using System;
using ProjNet.CoordinateSystems;
using ProjNet.CoordinateSystems.Transformations;

class Program
{
    static void Main()
    {
        // Define your reference point
        double refLon = 10.0; // Reference longitude
        double refLat = 50.0; // Reference latitude

        // Calculate the scale factor based on the reference latitude
        double la = abs(centerLatitude);
        double k0 = (la <= 55.0) ? 0.99994 : 0.99994 + ((la - 55.0) / 60.0) * 1.3E-04;

        // Create custom projection string with calculated scale factor
        string customProj = $"+proj=tmerc +lat_0={refLat} +lon_0={refLon} +k={k0:F10} +x_0=0 +y_0=0 +ellps=WGS84
+units=m +no_defs";

        // Initialize coordinate reference systems
        var geodesic = GeographicCoordinateSystem.WGS84;
        var cartesian = new CoordinateSystemFactory().CreateFromWkt(customProj);

        // Create transformations
        var ctFactory = new CoordinateTransformationFactory();
        var geodesicToCartesian= ctFactory.CreateFromCoordinateSystems(geodesic, cartesian);
        var cartesianToGeodesic= ctFactory.CreateFromCoordinateSystems(cartesian, geodesic);

        // Example usage
        double lon = 11.0;
        double lat = 51.0;

        // Convert WGS84 to Cartesian
        double[] cartesian = geodesicToCartesian.MathTransform.Transform(new double[] { lon, lat });
        Console.WriteLine($"Cartesian: X={cartesian[0]:F3}, Y={cartesian[1]:F3}");

        // Convert Cartesian back to WGS84
        double[] geographic = cartesianToGeodesic.MathTransform.Transform(cartesian);
        Console.WriteLine($"Geographic: Lon={geographic[0]:F6}, Lat={geographic[1]:F6}");
    }
}
```

A.2 Alternative implementation

The following Java implementation, courtesy of Daniel Dimov, accomplishes the same as the above. In systems where PROJ is not available, this code can be used either directly or translated into other programming languages.

```
/*
MIT License
```

[19] https://proj.org

Copyright (c) 2023 Daniel Dimov <danieldimov@gmail.com>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

```
*/
package org.fai.civl;

import static java.lang.Math.*;
import java.util.concurrent.atomic.AtomicLong;

public class CoordinateConverter {

    public static final double A_M = 6378137.0; // WGS84 equatorial radius in meters
    public static final double FLAT = 1.0 / 298.257223563; // WGS84 flattening

    private static final double D2R = PI / 180.0;
    private static final double R2D = 180.0 / PI;
    private static final double LIMIT_DELTA_ANGLE = 2.2; // beyond this angle from the central meridian is a "wrong
usage"
    private static final double FLAT_SQ_64 = FLAT * FLAT / 64.0;
    private static final double ONE_MINUS_F = 1.0 - FLAT;
    private static final double K1 = 0.0820944379 * 0.0820944379;
    private static final double K1_2 = K1 / 2.0;
    private static final double K2 = 0.006739496742 / 2.0;
    private static final double GEOMETRY_PRECISION = 1E-8; // precision of intersection, reflection and other geometry
calculation methods
    private static final Geodesic geodesic = new Geodesic(A_M, FLAT);

    // central coordinates are in degrees
    private final double centerMeridian; // in degrees
    private final double centerMeridianR; // in radians
    private final double centerLatitude; // in degrees
    private final double centerNorthing; // in meters
    private final double scaling; // scaling factor

    // how many times the converter is used beyond the normal angle difference from the central point
    private final AtomicLong wrongUsageCounter = new AtomicLong();

    public CoordinateConverter(double centerLatitude, double centerLongitude) {
        if (centerLatitude < -80.0 || centerLatitude > 80.0 || centerLongitude <= -180.0 || centerLongitude > 180.0)
{
            throw new IllegalArgumentException("Invalid center coordinates - latitude [-80,80], longitude (-
180,180]!");
        }

        centerMeridian = centerLongitude;
        centerMeridianR = centerLongitude * D2R;
        this.centerLatitude = centerLatitude;

        double la = abs(centerLatitude);
        scaling = (la < 55.0) ? 0.99994 : 0.99994 + ((la - 55.0) / 60.0) * 1.3E-04;
        PointCartesian p = toCartesian(centerLatitude * D2R, centerLongitude * D2R, centerMeridianR);
        this.centerNorthing = p.north; // centerNorthing is in meters
    }

    public long getWrongUsageCount() {
        return wrongUsageCounter.get();
    }

    // for internal use, angles are in radians, results are in meters
    private PointCartesian toCartesian(double latr, double lonr, double meridianr) {
        double cla = cos(latr);
        double cla2 = cla * cla;
        double s2la = sin(2.0 * latr);
        double sdlo = sin(lonr - meridianr);
        double t = 0.5 * log((1.0 + cla * sdlo) / (1.0 - cla * sdlo));
        double easting = t * scaling * 6399593.62 / sqrt((1.0 + K1 * cla2)) * (1.0 + K1_2 * t * t * cla2 / 3.0);
        double u = (3.0 * (latr + s2la / 2.0) + s2la * cla2) / 4.0;
```

```
        double northing =
                (atan(tan(latr) / cos((lonr - meridianr))) - latr)
                        * scaling * 6399593.625
                        / sqrt(1.0 + 0.006739496742 * cla2)
                        * (1.0 + K2 * t * t * cla2)
                        + scaling * 6399593.625 * (
                        latr
                                - 0.005054622556 * (latr + s2la / 2.0)
                                + 4.258201531E-05 * u
                                - 1.674057895E-07 * (5.0 * u + s2la * cla2 * cla2) / 3.0
                );
        return new PointCartesian(easting, northing);
    }

    public PointGeodetic toGeodetic(double easting, double northing) {

        double easting_ = easting;
        double northing_ = centerNorthing + northing;
        double north = abs(northing_);

        double k1 = north / 6366197.724 / scaling;
        double ck1 = cos(k1);
        double ck1_2 = ck1 * ck1;
        double k1a = 0.006739496742 * ck1_2;
        double k1b = 0.006739496742 * 3.0 / 4.0;
        double s2k1 = sin(2.0 * k1);
        double k2 = sqrt((1 + k1a));
        double k3 = scaling * 6399593.625 / k2;
        double k3a = easting_ / k3;
        double k3b = k1a * k3a * k3a / 2.0;
        double k4 = 1.0 - k3b;
        double k5 = 1.0 - k3b / 3.0;
        double k6 = 3.0 * (k1 + s2k1 / 2.0) + s2k1 * ck1_2;
        double k7 = (north - scaling * 6399593.625 * (
                k1 - k1b * (k1 + s2k1 / 2.0) + k1b * k1b * k6 * 5.0 / 3.0 / 4.0
                        - k1b * k1b * k1b * 35.0 / 27.0 * (k6 * 5.0 / 4.0 + s2k1 * ck1_2 * ck1_2) / 3.0
        )) / k3 * k4 + k1;
        double lonr = atan((exp((easting_) / k3 * k5) - exp(-(easting_) / k3 * k5)) / 2.0 / cos(k7));
        double k10 = atan(cos(lonr) * tan(k7)) - k1;
        double latr = (k1 + (1.0 + k1a - 0.006739496742 * sin(k1) * ck1 * k10 * 3.0 / 2.0) * k10);

        if (northing_ < 0.0) return new PointGeodetic(-latr * R2D, (lonr + centerMeridianR) * R2D);
        else return new PointGeodetic(latr * R2D, (lonr + centerMeridianR) * R2D);
    }

    public PointCartesian toCartesian(double latitude, double longitude) {
        // validate the input
        if (latitude < -90.0 || latitude > 90.0 || longitude < -184.0 || longitude > 184.0)
            throw new IllegalArgumentException("Valid ranges: latitude [-90,90], longitude [-184,184].");

        double lon = longitude;

        if (centerMeridian > 176.0 && longitude < 0.0) lon = longitude + 360.0;
        if (centerMeridian < -176.0 && longitude > 0.0) lon = longitude - 360.0;

        if (abs(latitude - centerLatitude) > LIMIT_DELTA_ANGLE || abs(lon - centerMeridian) > LIMIT_DELTA_ANGLE)
            wrongUsageCounter.incrementAndGet();

        PointCartesian p = toCartesian(latitude * D2R, lon * D2R, centerMeridianR);
        p.north = p.north - centerNorthing;
        return p;
    }
}
```

Annex B PathFinder extension for linear control zones

This annex describes the extension of the PathFinder algorithm, as presented in Ding et al. (2018)[20], to also find the shortest path through a task where one or several of the route elements are lines.

B.1 Line control zone optimization

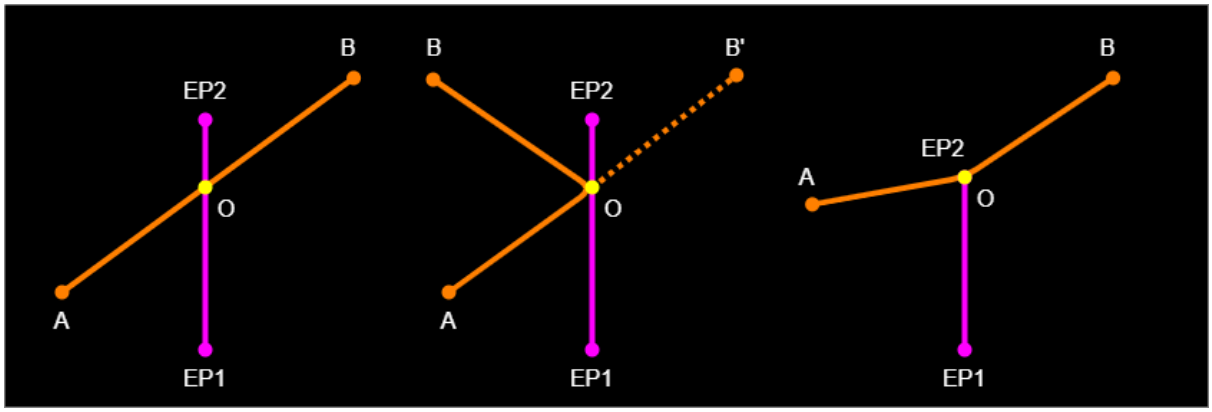
The optimization solution must find a point O on the line segment L such that the length of the route from previous point A to point O and then to next point B is minimum. There are two scenarios:

1. The line is in the middle of the task
2. The line is at the beginning or at the end of the task

B.1.1 Line in the middle of the task

For lines in the middle of the task, there are three cases for the positioning of the points A (previous point), B (next point) and the line L:

a) Line segment A-B and line segment L intersect: Point O is the intersection point of line segments A-B and L
b) B' is a reflection point of point B over line L and line segment A-B' and line segment L intersect: Point O is the intersection point of line segments A-B' and L
c) Line segments A-B and A-B' does not intersect line segment L: Point O is either Endpoint1 or Endpoint2. We calculate lengths of routes A-EP1-B and A-EP2-B and depending on which one is smaller - we choose the corresponding EP.



B.1.2 Line at the beginning or at the end of the task

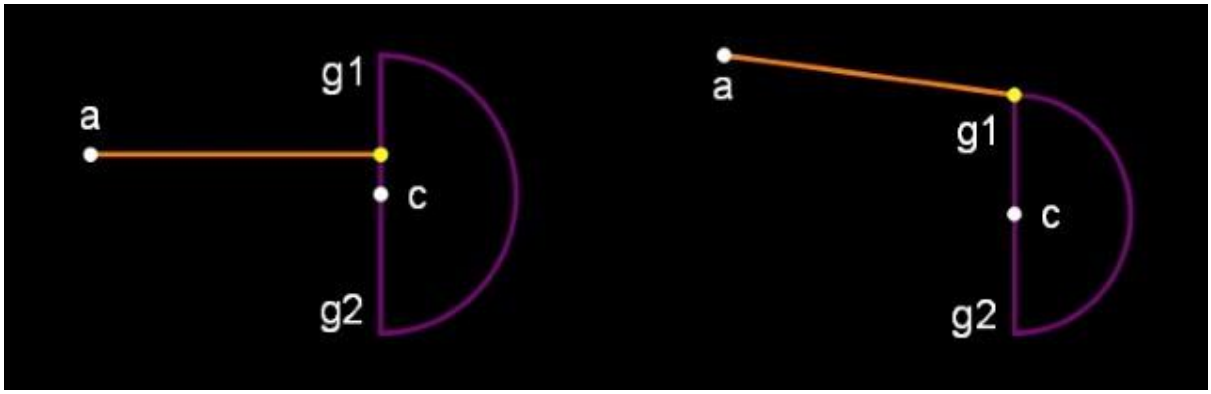
In the second scenario there are only 2 cases for the position of line L and point A/B (there is no other point because the line is first/last control zone in the task):

a) Point A projection A' on line L lies within the line segment L
b) Point A' lies outside of the line segment L

Finding the closest point on the line from point A follows the same procedure as the following code, which was originally created to find the closest point on a goal line that was not aligned with the optimized route:

[20] Ding, Xie, Jiang, An Efficient Algorithm for Touring n Circles, EITCE 2018. Download here: https://www.matec-conferences.org/articles/matecconf/pdf/2018/91/matecconf_eitce2018_03027.pdf

Finds the closest point on the goal line (g1, g2) from point A, storing the result in the C fix position. This will either be on the line itself, or at one of its endpoints.




```
//Inputs:
//line-arravofgoallineendpoints
//c,a-target(goal),previouspoint

functionprocessLine(line,c,a)
{
g1=line[0],g2=line[1];
len2=(g1.x-g2.x)**2+(g1.y-g2.y)**2;

if(len2==0.0){
//Errortrapping:g1andg2arethesamepoint
c.fx=g1.x;
c.fy=g1.y;
}else{
t=((a.x-g1.x)*(g2.x-g1.x)+(a.y-g1.y)*(g2.y-g1.y))/len2;

if(t<0.0){
//Beyondtheg1endofthelinesegment
c.fx=g1.x;
c.fy=g1.y;
}elseif(t>1.0){
//Beyondtheg2endofthelinesegment
c.fx=g2.x;
c.fy=g2.y;
}else{
//Projectionfallsonthelinesegment
c.fx=t*(g2.x-g1.x)+g1.x;
c.fy=t*(g2.y-g1.y)+g1.y;
}
}
}
```

Annex C Ellipsoid distance between two points

This annex gives a sample implementation of a partial solution to the inverse geodesic problem: It calculates the distance between two points given on the WGS84 sphere. The advantage of this implementation over solutions such as Vincenty or Kearny is that it is much faster to calculate, which can be crucial in navigation devices with limited computing performance. In cases where only the distance is required, and not the azimuth between the given points, this algorithm is preferrable.

The following Java implementation was supplied by Daniel Dimov. He found the underlying algorithm on the Web a few years ago. Despite considerable effort, we were unable to find that source again, nor the original author of the algorithm. If any of our readers can supply us with information regarding the algorithm's origins, we will highly appreciate it.

```
public static final double A_KM = 6378.137; // equatorial radius in km
public static final double FLAT = 1.0 / 298.257223563; // flattening
protected static final double FLAT_SQ_64 = FLAT * FLAT / 64.0;
protected static final double ONE_MINUS_F = 1.0 - FLAT;

public static double distanceRad(double lat1r, double lon1r, double lat2r, double lon2r) {

    if (lon1r == lon2r && lat1r == lat2r) return 0.0;

    double theta1 = atan(ONE_MINUS_F * tan(lat1r));
    double theta2 = atan(ONE_MINUS_F * tan(lat2r));

    double theta_m = (theta1 + theta2) / 2.0;
    double d_theta_m = (theta2 - theta1) / 2.0;
    double d_lambda = lon2r - lon1r;
    double d_lambda_m = d_lambda / 2.0;

    double sin_theta_m = sin(theta_m);
    double cos_theta_m = cos(theta_m);
    double sin_d_theta_m = sin(d_theta_m);
    double cos_d_theta_m = cos(d_theta_m);
    double sin2_theta_m = sin_theta_m * sin_theta_m;
    double cos2_theta_m = cos_theta_m * cos_theta_m;
    double sin2_d_theta_m = sin_d_theta_m * sin_d_theta_m;
    double cos2_d_theta_m = cos_d_theta_m * cos_d_theta_m;
    double sin_d_lambda_m = sin(d_lambda_m);
    double sin2_d_lambda_m = sin_d_lambda_m * sin_d_lambda_m;

    double H = cos2_theta_m - sin2_d_theta_m;
    double L = sin2_d_theta_m + H * sin2_d_lambda_m;
    double cos_d = 1.0 - 2.0 * L;
    double d = acos(cos_d);
    double sin_d = sin(d);

    double one_minus_L = 1.0 - L;

    if (sin_d == 0.0 || L == 0.0 || one_minus_L == 0.0) return 0.0;

    double U = 2.0 * sin2_theta_m * cos2_d_theta_m / one_minus_L;
    double V = 2.0 * sin2_d_theta_m * cos2_theta_m / L;
    double X = U + V;
    double Y = U - V;
    double T = d / sin_d;
    double D = 4.0 * T * T;
    double E = 2.0 * cos_d;
    double A = D * E;
    double B = 2.0 * D;
    double C = T - (A - E) / 2.0;

    double n1 = X * (A + C * X);
    double n2 = Y * (B + E * Y);
    double n3 = D * X * Y;

    double delta1d = FLAT * (T * X - Y) / 4.0;
    double delta2d = FLAT_SQ_64 * (n1 - n2 + n3);

    return A_KM * sin_d * (T - delta1d + delta2d);
}
```