Via Technology

EUROCONTROL

Trajectories Production Final Report

Via Technology Ltd.

Trajectories Production Final Report

Phone: +44 1202 708476

Table of Contents

Table of Figures

Table of Tables

Revision History

Issue	Date	Description
Draft A	30 May 2018	Initial version.
Draft B	1 June 2018	Incorporate internal review comments.
Draft C	8 June 2018	Add Executive Summary and Potential Improvements
1.0.0	21 June 2018	Accepted for release.

References

Number	Document
1	Price Enquiry No. 17-110452-A Trajectories Production via Cloud-Based Analytics for Performance Monitoring and Review, Reference: DPS/PRU/SQS/PR-Data2017
2	Trajectories Production Tender, Issue 1.1.0, Via Technology Ltd.
3	Trajectories Production Conceptual Design Report
4	Trajectories Production Infrastructure Definition Report
5	Trajectories Production Potential Algorithms Report
6	Trajectories Production Data Merging Report
7	Trajectories Production Airspace Intersections Report
8	Trajectories Production Data Sharing Design Report
9	Python Software Foundation: https://www.python.org/
10	PostGIS: https://postgis.net/
11	Docker: https://www.docker.com/
12	Kubernetes: https://kubernetes.io/
13	Google Cloud Platform: https://cloud.google.com/
14	ISO 8601 Date and Time format https://www.iso.org/iso-8601-date-and-time-format.html and https://en.wikipedia.org/wiki/ISO_8601
15	OOOI Data http://aspmhelp.faa.gov/index.php/OOOI_Data
16	ICAO Doc 8643 – Aircraft Type Designators, Special Designators https://www.icao.int/publications/DOC8643/Pages/SpecialDesignators.aspx
17	Mode S flights – Assignment of 24-bit aircraft addresses to State aircraft http://www.eurocontrol.int/articles/mode-s
18	Feasibility Study on Cloud-Based Analytics for Performance Monitoring (EUROCONTROL 16-110637-E Innaxis) CBA_Final_Report.pdf
19	Eurocontrol Specification for Surveillance Data Exchange ASTERIX Part 12 : Category 21 ADS-B Target Reports, EUROCONTROL-SPEC-0149-12
20	ICAO SSR Code Assignment System https://www.icao.int/NACC/Documents/eDOCS/ATM/ORCAM%20CAR%20SAM%20-%20SSR%20Codes%20en.pdf
21	Universally Unique Identifier

Number	Document
	https://en.wikipedia.org/wiki/Universally_unique_identifier
22	Airport Movement Data format: APDS_Data.pdf
23	ICAO Doc 9905 AN/471 Required Navigation Performance Authorization Required (RNP AR) Procedure Design Manual First Edition – 2009
24	Problem of Apollonius
25	Common Errors on Flightradar 24
26	scipy.interpolate.CubicSpline class
27	scipy.interpolate.interp1d class
28	https://en.wikipedia.org/wiki/Median_filter
29	https://en.wikipedia.org/wiki/Moving_average

Glossary

Item	Description
ADS-B	Automatic dependant Surveillance – Broadcast
ANSP	Air Navigation Service Provider
APDS	Airport Data Flow
CPR	Correlated Position Reports
ECEF	Earth Centred Earth Fixed
FR24	Flight Radar 24
IATA	International Air Transport Association
ICAO	International Civil Aviation Organisation
PRC	Performance Review Commission
PRU	Performance Review Unit
SAC	System Area Code
SIC	System Indetification Code
SSR	Secondary Surveillance Radar
TAS	True Air Speed
UUID	Universally Unique Identifier
WGS-84	World Geodetic System – 1984

Section 1

1 Executive Summary

1. The Trajectories Production project demonstrates that aircraft trajectory data from a variety of different sources can be merged to create reference trajectories.
2. The project also shows how a system can be implemented on a cloud platform using the Kubenetes orchestration platform. In particular, it demonstrates how a JupyterHub server and Kubenetes can be used to serve multiple simultaneous users.

1.1 Trajectory Formatting

3. Raw trajectory data is converted into common formats, i.e.: flight format, position format and event format.
4. This data is stored in .csv text files so that can easily be read by a wide variety of tools, including: text editors, spreadsheets, databases and data analysis software libraries.
5. Trajectory data files usually contain a specific type of data (flight, position or event data) for a single day. Each file contains data for approximately sixty thousand flights. Uncompressed, the biggest (positions) files are often larger than a Gigabyte. All files are compressed before being stored on the cloud, which reduces a positions file to approximately 200 Megabytes.
6. The common trajectory data format enables all trajectory data to be processed (e.g. cleaned, merged, analysed and interpolated) by the same set of tools, regardless of the original source of the raw data.
7. The common trajectory data format also enables converted trajectory data to be processed at any stage of the processing pipeline. For example, converted CPR data or ADS-B data can be analysed immediately after conversion, before it has been merged or even cleaned.

1.2 Trajectory Module

8. A key task of the project is data validation, i.e. data cleaning and quality assurance. Trajectory positions are cleaned at every stage of the process and error metrics calculated at each cleaning stage are stored on the cloud.
9. Data is also validated prior to being merged. Positions from different data sources for the same flight are only considered valid if they are within given horizontal and vertical thresholds of each other at the same time. Valid positions are then merged by grouping them together and sorting them in time order.

10. However, of the four position dimensions: Latitude, Longitude, altitude and time, time is the least accurate. An early or late time for an otherwise accurate position can cause the associated trajectory to “double-back” on itself.

11. Trajectory analysis takes the time ordered positions and derives a horizontal path from their Latitude and Longitude coordinates. The Latitude and Longitude coordinates are converted into Earth Centred, Earth Fixed (ECEF) coordinates, enabling a horizontal path composed of Great Circle legs and the turns between them, to be constructed anywhere on Earth without any of the distortions that are introduced by projecting Latitude and Longitude coordinates onto a planar surface.

12. Distances of positions along the horizontal path are derived and then used to re-order the positions. Next, the relative distances and times of the positions are used to calculate speeds between positions. The calculated speeds are then smoothed and used to back calculate the times when the aircraft was at the positions.

13. Altitude profiles and time profiles are created from the altitudes and times at distances along the horizontal path. Production trajectories are created by interpolating the horizontal path, altitude and time profiles at given distances along the path.

14. The ECEF horizontal path is the key to both trajectory analysis and interpolation. It enables trajectory analysis to handle large time errors in the input data; and it also enables Latitude and Longitude coordinates to be interpolated directly along Great Circle legs and around the turns between them.

1.3 Trajectory Assessment

15. Trajectory assessment determines sections of trajectories belonging to airspaces of interest.

16. Airspaces are stored in a GIS database. Standard GIS database functions are called to find horizontal intersections between the airspaces and trajectories in Latitude and Longitude coordinates.

17. The horizontal intersections determine the horizontal sections of trajectories belonging to airspaces; the 3D (horizontal and vertical) trajectory sections are determined by the altitude profile sections corresponding to the horizontal sections.

18. Again, the ECEF horizontal path is the key to finding the altitude profile sections that correspond to the horizontal trajectory sections.

1.4 Cloud Platform

19. The software is written in python and runs on a JupyterHub server in a Docker container on a Kubenetes cluster.

20. Kubenetes is the de-facto standard container orchestration platform and Docker is the de-facto standard container. Kubenetes was originally developed by Google but it is now an open source project supported by all the major cloud providers: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

21. JupyterHub enables multiple users to access a Jupyter server simply by entering the URL of the server in a Web browser. Jupyter notebooks support the Julia, python and R programming languages.

22. The software is written in python, a simple high-level language that has particularly good library support for developing data science applications. When used together with Jupyter notebooks, python enables users to view (and interact with) graphs and charts of data in their Web browser.

23. The disadvantages of python are it's memory consumption and processing speed. For example, merging consecutive day trajectories requires a compute node with 26GB of RAM and analysing a days positions file takes around a day.

Section 2

2 Introduction

24. This document is the Trajectories Production Final Report of Via Technology Ltd. for Eurocontrol Contract No. 17-110452-C Trajectories Production via Cloud-Based Analytics for Performance Monitoring and Review [1].

25. This report:
- Provides the rationale and principal description of the chosen solution;
- Provides technical documentation for the implemented algorithms;
- Contains a cost breakdown for the provision of the services on a cloud platform including additional costs for upscaling.

2.1 Objective

26. The Performance Review Commission (PRC) is a Commission established by the Permanent Commission of Eurocontrol to provide advice in order to ensure the effective management of the European air traffic management system through a strong, transparent and independent performance review system.

27. The Performance Review Unit (PRU) supports the PRC. Organisationally, it is part of the Eurocontrol Agency's Pan-European Sky Directorate. The PRU collect data from different sources, process, and store it. Integration and validation of data are fundamental tasks to create a value added set of analytic data. This report sets out how these tasks can be achieved within a cloud based environment.

28. The goal of trajectories production is to process and store analytic data for performance review purposes, consisting of:

- gate-to-gate trajectories,
- airspace intersections
- and ground and fleet data.

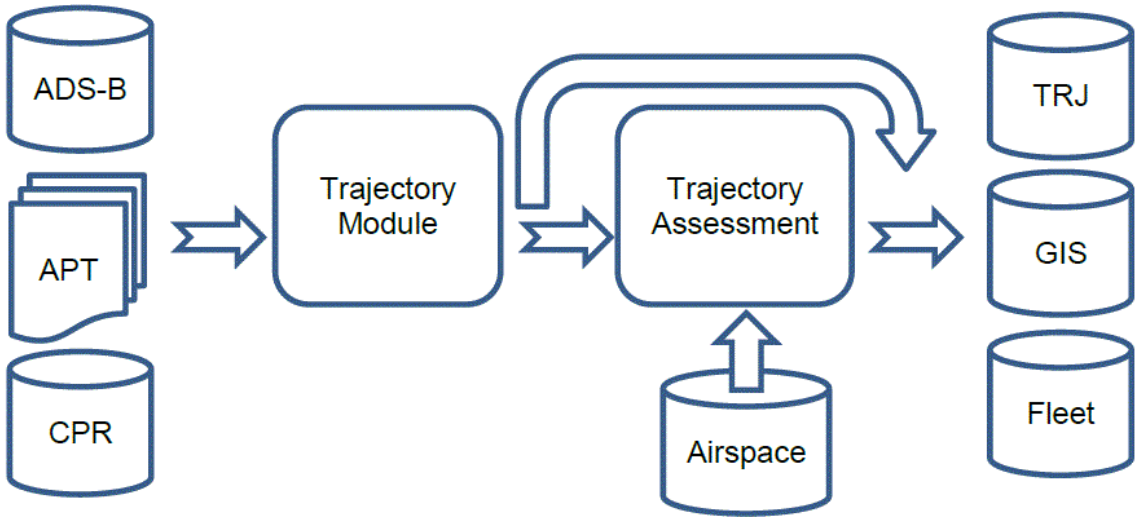

Figure 1: Overall Processing Pipeline

29. Figure 1 depicts a logical conceptualisation for the processing with the Trajectory Module validating and merging data from different sources and Trajectory Assessment determining airspace intersections and maintaining the ground and fleet data.

30. The chosen design solution is a combination of bespoke software (written in python [9]) and PostGIS [10] geospatial databases running inside Docker [11] containers in a Kubernetes cluster [12] on the Google Cloud Platform [13].

2.1.1 Overview

31. Figure 1 shows the overall processing pipeline: the Trajectory Module processes data from different sources to create reference trajectories for Trajectory Assessment.

2.1.1.1 Trajectory Module

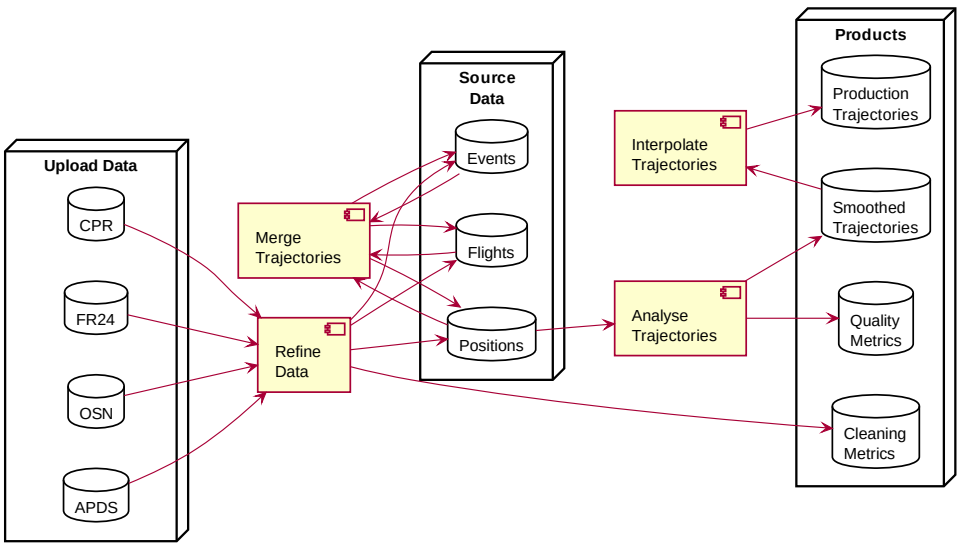

Figure 2: Trajectory Module Overview

32. The Trajectory Module refines and merges data from a variety of different sources, converting the data into common formats, cleaning and merging it with the aim of producing complete gate-to-gate trajectories for every flight, see Figure 2.

33. Trajectory position data is analysed by the Trajectory Module to produce smoothed trajectories which the Trajectory Module then interpolates to produce production trajectories with positions at 5 second intervals.

2.1.1.2 Trajectory Assessment

34. Trajectory Assessment takes smoothed trajectories from the Trajectory Module to find intersections with:
    - elementary sectors,
    - airport cylinders and
    - user defined airspaces.

2.1.2 Rationale

2.1.2.1 Programming Language and Database

35. Python was chosen as the software development language, because it is a simple high-level language that is particularly well suited for developing data science applications.

36. Jupyter Notebooks are a part of the Python development ecosystem. They enable developers to write and interact with python programs and their outputs in a web-browser. The name Jupyter is a concatenation of Julia, Python and R, since Jupyter notebooks also support the Julia and R programming languages.

37. Jupyter Notebook support was a key reason for choosing python as the software development language. The ability to view (and interact with) graphs and charts of data in Jupyter Notebooks was considered to be a great advantage.

38. The PostGres database with GIS extensions (PostGIS) database was chosen as it is an Open Source database with GIS support.

2.1.2.2 Orchestration System

39. The Kubernetes container-orchestration system chosen as it is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications that is supported by all the major cloud platforms: Amazon Web Services, Microsoft Azure and Google Cloud Platform.

40. The JupyterHub server is used to create a multi-user hub to run multiple single-user Jupyter notebook servers. The JupyterHub server is available as a Docker image, therefore Docker containers were chosen to use JupyterHub with Kubernetes.

2.1.2.3 Cloud Provider

41. There are multiple cloud providers each offering different trade-offs of: cost, scale, ease of use and compliance with information protection standards.

42. We have chosen to use the Google Cloud Platform, as it has:
    - the most applicable resources,
    - the best integration with Kubernetes, the chosen orchestration framework
    - and the best region selection for Europe.

Section 3

3 Trajectory Module

43. The Trajectory Module validates, merges, analyses and interpolates data from different sources: currently CPR, Flight Radar 24 ADS-B and APDS, see Figure 3.

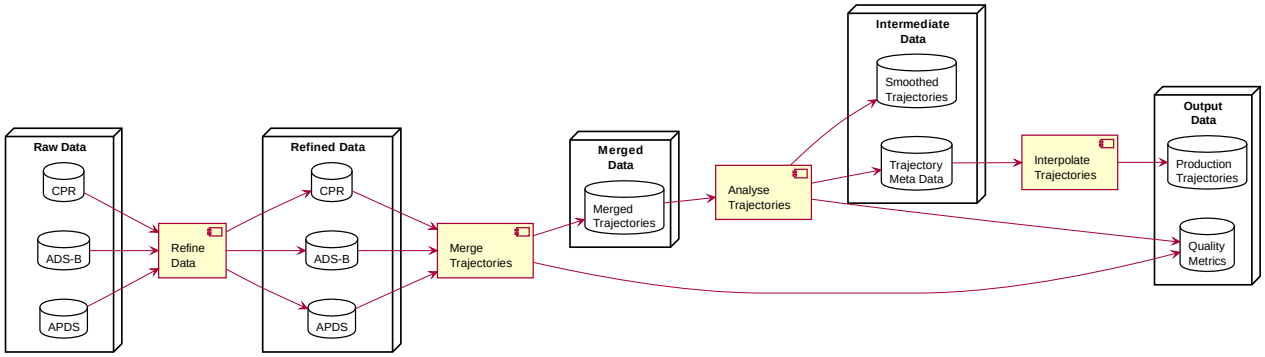

Figure 3: Trajectory Module

44. In the following sections, data refining and merging is considered separately from data analysis and interpolation.

3.1 Data Refining and Merging

45. Figure 4 is an overview of the data refining and merging processes.

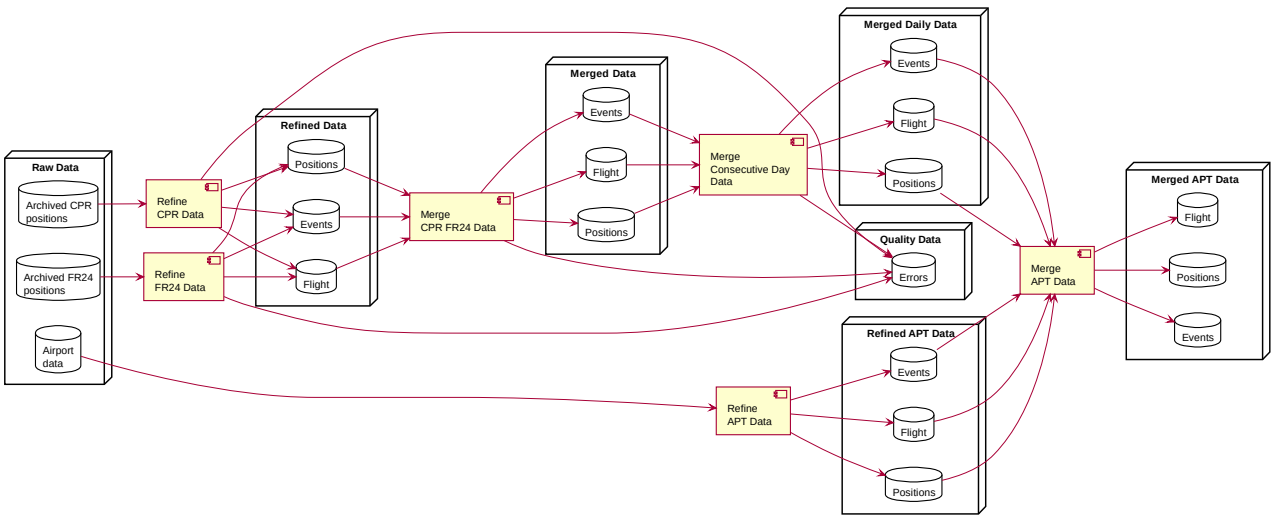

Figure 4: Data Refining and Merging

46. Data refining and merging combines raw data positional reports from CPR, ADS-B (currently Flight Radar 24, FR24) and APDS sources. Each data set must be refined, i.e. converted into a common format and “cleaned” prior to being merged with data from the other sources.

47. The PRU receive CPR and FR24 data daily while APDS data is received monthly, often in arrears. Therefore, data is refined and merged in two different processes:

1. Refine and merge CPR data and FR24 data
2. Merge the merged CPR and FR24 trajectories with APDS data.

48. The first process can be run whenever CPR and FR24 data are available, i.e. daily. While the second process can be run when the corresponding APDS data is available, i.e. monthly, see Figure 5.

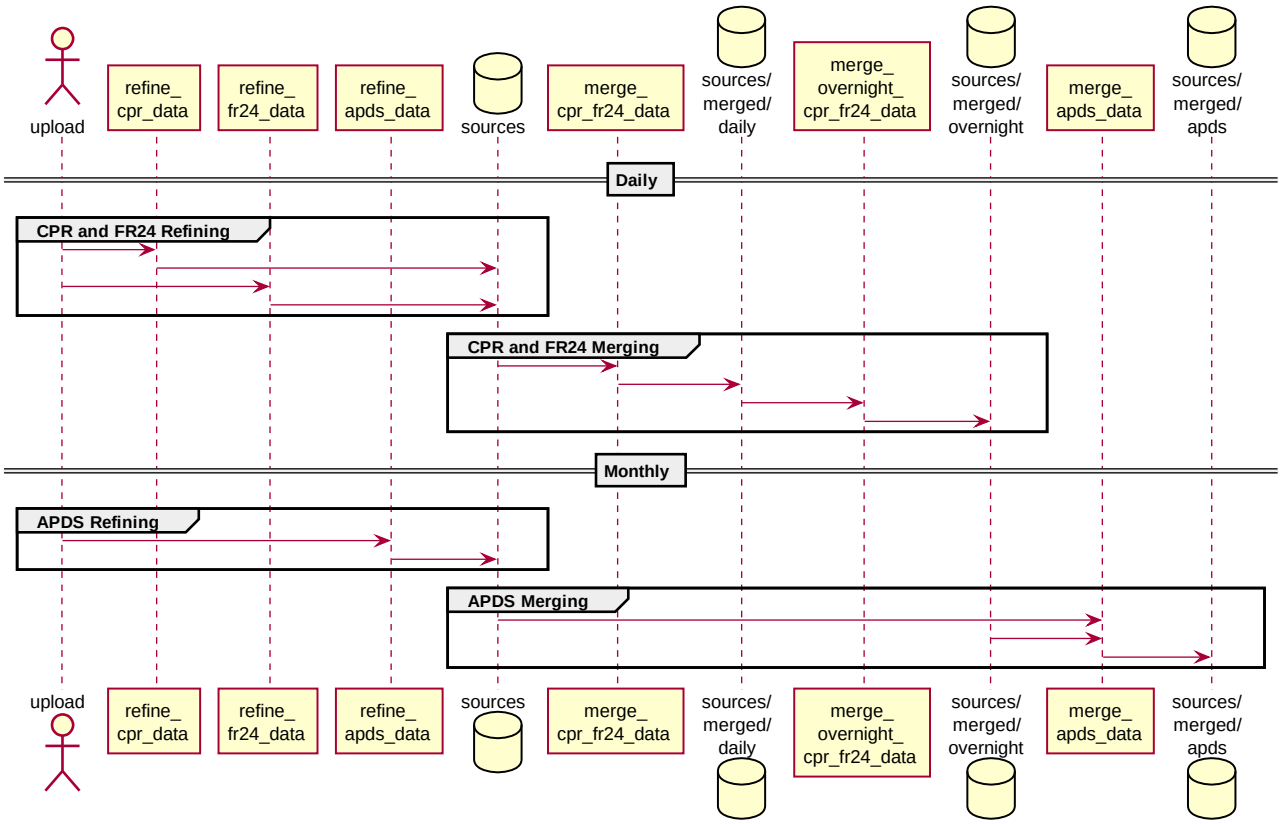

Figure 5: Data Refining and Merging Sequence

49. Note: there are two CPR-FR24 merging processes:

- merge CPR and FR24 flights for the same day;
- merge daily “merged” CPR and FR24 flights overnight, to produce complete gate-to-gate trajectories for flights that are airborne over midnight.

50. Figure 6 shows the CPR and FR24 data refining and merging sequence in more detail.

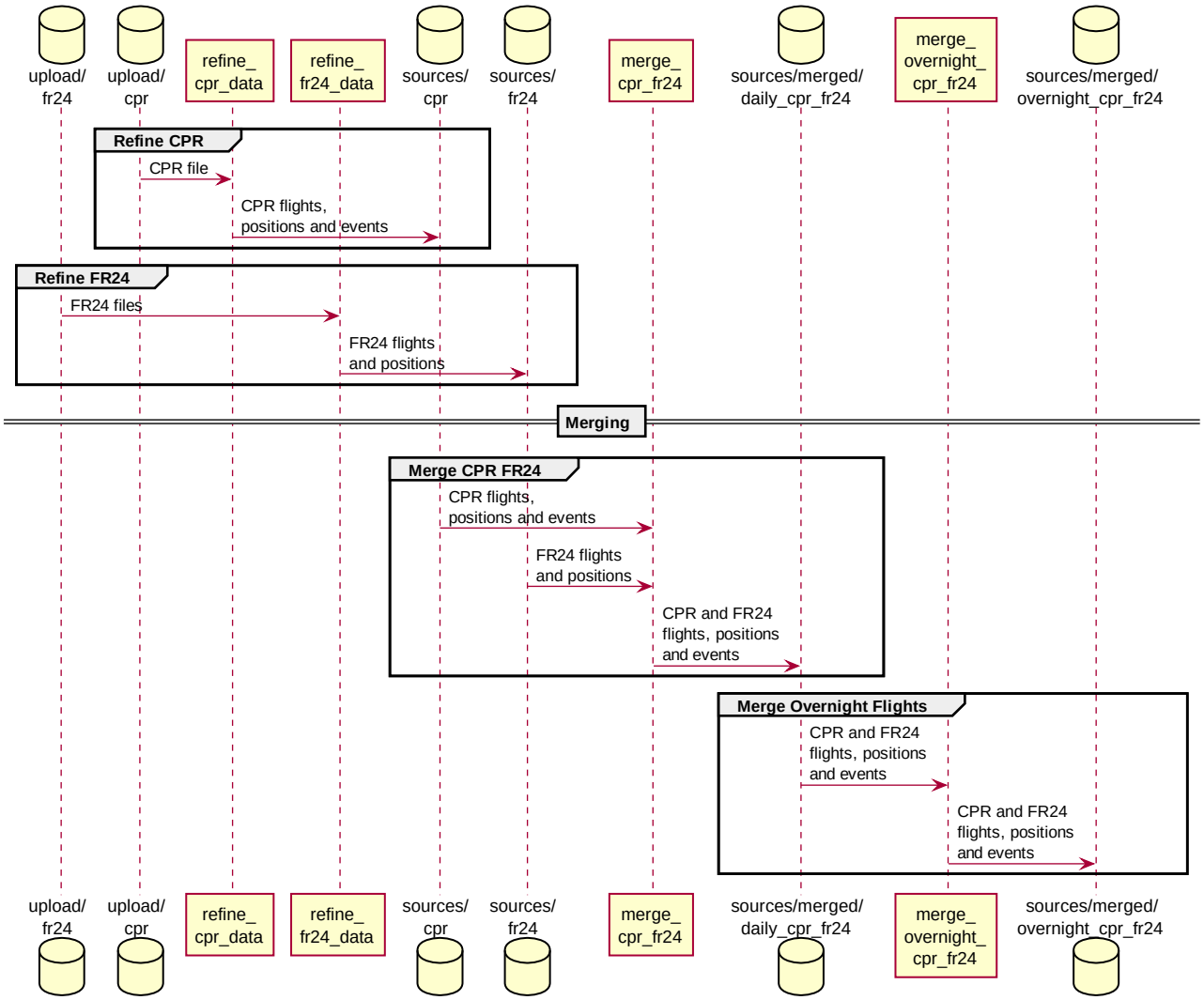

Figure 6: CPR and FR24 Refining and Merging Sequence

51. The CPR and FR24 data are refined separately before being merged together. The merged CPR and FR24 data can then be merged with the previous days merged CPR and FR24 data to enable complete “gate-to-gate” trajectories for overnight flights to be created.

52. When APDS data is available, it can also be refined and merged with the merged CPR and FR24 data.

53. Figure 7 shows the APDS data refining and merging sequence in more detail.

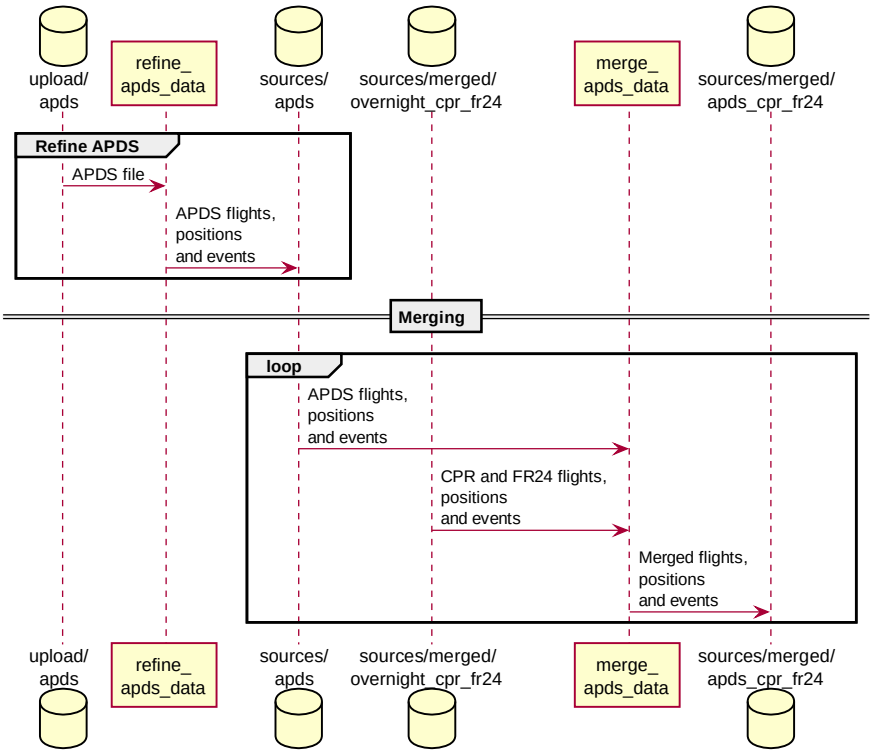

Figure 7: APDS Refining and Merging Sequence

54. The APDS data is refined to convert it into the same format as the merged CPR and FR24 data. Note: the locations of airport stands must be available to enable the refine APDS data process to create positions from the APDS data.

3.1.1 Refined Data Format

3.1.1.1 Flight Format

55. Refined trajectory flights contain the following fields:

Flight Field	Description
FLIGHT_ID	The id of the flight/trajectory.
CALLSIGN	The callsign of the flight.
AIRCRAFT_REG	The registration of the aircraft, empty if not available.
AIRCRAFT_TYPE	The ICAO aircraft type code, empty if not available.
AIRCRAFT_ ADDRESS	The 24 bit ICAO aircraft address, empty if not available.
ADEP	The ICAO code of the departure airport.
ADES	The ICAO code of the destination airport.
SSR_CODES	The SSR codes emitted by the aircraft during the flight.
PERIOD_START	The UTC start time of the flight in ISO 8601 Date and Time format [14] to a seconds precision.
PERIOD_FINISH	The UTC finish time of the flight in ISO 8601 Date and Time format [14] to a seconds precision.
SOURCE_IDS	The flight ids from the surveillance sources used to create the trajectory – initially empty.

Table 1: Flight Fields

56. The flight fields in Table 1 are populated by the fields available from the input sources. For example, AIRCRAFT_REG is not available from the CPR data, while SSR_CODES are not available from the APDS data.

57. Note: the FLIGHT_ID field shall contain the data source id for refined input data, and the SOURCE_IDS field will normally be empty. The SOURCE_IDS field is populated with the FLIGHT_IDs of the input data sources for merged flights.

3.1.1.2 Positions Format

58. Refined trajectory positions contain the following fields:

Position Field	Description
FLIGHT_ID	The id of the flight/trajectory.
DISTANCE	The distance from the first point along the derived path in Nautical Miles – initially empty.
TIME	The UTC time in ISO 8601 Date and Time format [14]. Input times are to a seconds precision, output times are to millisecond precision or greater.
LAT	WGS-84 Latitude in signed decimal degrees, North positive.
LON	WGS-84 Longitude in signed decimal degrees, East positive.
ALT	The barometric altitude (at 1013.25hPa) in feet.
SPEED_GND	The ground speed in Knots.
TRACK_GND	The ground track in degrees relative to True North.
VERT_SPEED	The climb or descent rate in feet per minute, descent is negative.
ON_GROUND	True if aircraft was on the ground, false if airborne.
SURVEILLANCE_ SOURCE	The source of the position, e.g.: CPR SAC SIC, FR24 radar_id or APDS.
AIRCRAFT_ADDRESS	The 24 bit ICAO aircraft address, or empty if not available.
SSR_CODE	The 4 character octal SSR code.

Table 2: Position Fields

59. The position fields in Table 2 are available from the CPR and FR24 ADS-B data. Some of the fields (e.g. speed, ground track and vertical speed) are not available in the airport movement data.

60. Note: the SURVEILLANCE_SOURCE field contains the source of the position: CPR, FR24 or APDS and a reference from the source data, e.g. the Asterix SAC SIC for CPR data, the radar_id for FR24 data and the relevant airport and stand for APDS data.

3.1.1.3 Event Format

61. Refined trajectory events shall contain the following fields:

Event Field	Description
FLIGHT_ID	The id of the flight/trajectory.
EVENT_TYPE	The event of interest, an enumerated type.
TIME	The UTC time from the input data source in ISO 8601 Date and Time format [14] to a seconds precision.

Table 3: Event Fields

62. Where the event type may be one of:

Event Type	Number	Description
SCHEDULED_ OFF_BLOCK	**0**	The scheduled off-block time of the flight.
GATE_OUT	**1**	Aircraft leaves gate or parking position. The actual off block time of the flight.
WHEELS_OFF	**2**	Aircraft takes off. The actual take off time of the flight.
WHEELS_ON	**3**	Aircraft touches down. The actual landing time of the flight.
GATE_IN	**4**	Aircraft arrives at gate or parking position. The actual on block time of the flight.
SCHEDULED_ IN_BLOCK	**5**	The scheduled in-block time of the flight.

Table 4: Event Types

63. Note: the Table 4 event names: GATE_OUT, WHEELS_OFF, WHEELS_ON and GATE_IN are taken from FAA Operations & Performance OOOI Data [15].

3.1.2 Refine CPR Data

64. Correlated Position Report messages (CPR plots) are received daily in a CSV file format, using a semi-colon(;) instead of a comma.

65. Uncompressed, the CPR files may reach a maximum size of 2 GB. However, the files are normally compressed (into gzip format), where the typical file size is approximately 200 MB.

66. CPR Flights are grouped together by TACT id. The TACT id is the equivalent of a FLIGHT_ID for the CPR data: i.e. all plots with the same TACT id should belong to the same flight. However, not all plots are assigned a TACT id. In the first weeks CPR

sample files up to 2% of CPR plots per day were not assigned a TACT id, see Table 5 in [6] Trajectories Production Data Merging Report.

67. The first weeks CPR sample files also contained up to 2% duplicate plots per day, see Table 6 in [6] Trajectories Production Data Merging Report.

68. Duplicate CPR plots and CPR plots that have not been assigned a TACT id are not required. Therefore, they are not processed and stored.

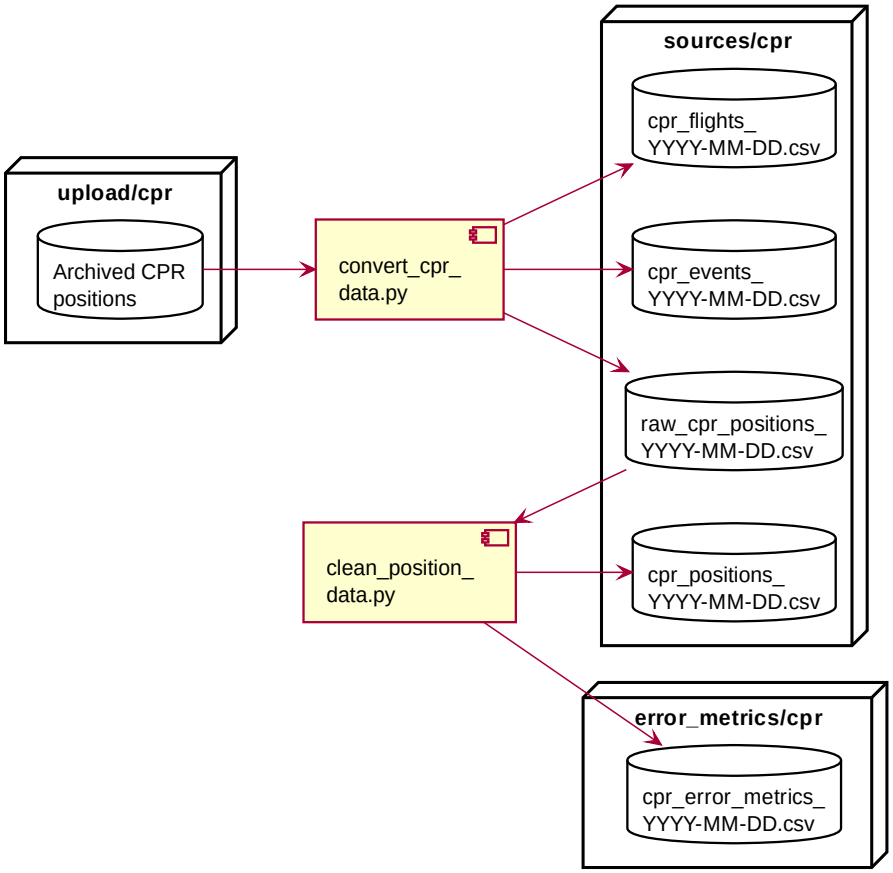

Figure 8: Refine CPR Data

69. A CPR file is converted into a flights, an events and a positions file by the python script, convert_cpr_data.py (see Figure 8), used as follows:

```
convert_cpr_data.py <filename>

where <filename> is a CPR file (gzip format or uncompressed)
named as described in: ArchievedCPRformat.pdf
```

70. It outputs a flights file, an events file and a positions file named: cpr_flights_YYYY-MM-DD.csv, cpr_events_YYYY-MM-DD.csv and raw_cpr_positions_YYYY-MM-DD.csv respectively. Where YYYY-MM-DD is the date extracted from the CPR filename in ISO 8601 format [14].

3.1.3 Clean Position Data

71. All trajectory data contains measurement errors and most trajectory data sources have erroneous positions. For example:
    - duplicate positions,
    - significant positional offsets,
    - spurious flight segments
    - and non-synchronised timestamps.

72. Trajectory cleaning removes gross trajectory position errors like those described above. The trajectory cleaning algorithms are described in Appendix A.

73. Positions files are cleaned by the python script, clean_position_data.py (see Figure 8 and Figure 9), used as follows:

```
clean_position_data.py <raw_positions_filename> [max_speed] [distance_accuracy]
```

74. where <raw_positions_filename> is a positions file (bzip2 format or uncompressed) named as described in section 3.1.1.2, max_speed is the maximum valid speed in Knots and distance_accuracy is the distance accuracy in Nautical Miles, default 750 Knots and 0.25 Nautical Miles respectively.

75. It outputs a positions file named: <positions_filename> (i.e. without “raw_”) and an error metrics file named: errors_<filename>. Where <raw_positions_filename> is the filename parameter.

76. The error metrics file contains the numbers of duplicate and erroneous positions for each trajectory. Erroneous positions may be classified as:

    - DUPLICATES – for duplicate positions (merged data only)
    - ADDRESSES – for invalid aircraft addresses (CPR data only)
    - DISTANCE – for invalid horizontal positions
    - ALTITUDE – for invalid vertical position

77. Table 5 is an example of the data in an error metrics file.

FLIGHT_ID	TOTAL	DUPLICATES	ADDRESSES	DISTANCE	ALTITUDE
205920	1	0	0	1	0
205931	1	0	0	0	1
206919	1	0	0	0	1
232696	2	0	0	2	0
238496	2	2	0	0	0
240753	4	1	0	3	0

Table 5: Data cleaning example csv output

3.1.4 Refine FR24 Data

78. ADS-B data is received from Flight Radar 24 and stored in a database by Eurocontrol. The FR24 data is then supplied in two CSV format files for each day: one containing positions and the other containing flight data.

79. The Flight Radar 24 ADS-B data includes trajectories for any vehicle fitted with an ADS-B transmitter, not just aircraft. It includes data for: ground vehicles, gliders and other vehicles, see [16] Aircraft Type Designators, Special Designators.

80. Table 6 shows the numbers of flights by vehicles with special designators in the FR24 data on each day of the first week of August 2017.

Aircraft Type	2017-08-01	2017-08-02	2017-08-03	2017-08-04	2017-08-05	2017-08-06	2017-08-07	Average
BALL	8	9	3	12	15	6	11	9.14
GLID	1879	2407	1816	2377	3151	4572	3032	2747.71
GRND	5	7	5	5	5	6	6	5.57
GYRO	4	5	1	3	5	2	1	3
PARA	0	0	0	0	0	0	0	0
UHEL	0	0	0	0	0	0	0	0
ULAC	42	50	30	26	79	76	64	52.43
SHIP	0	0	0	0	0	1	3	0.57
Totals	1938	2478	1855	2423	3255	4663	3117	2818.43

Table 6: ICAO Special Designators

81. All ADS-B transmissions include a 24-bit aircraft address. According to rules laid down by ICAO, “all aircraft must use a correctly assigned and unique ICAO 24-bit aircraft address at all times during a flight“, see [17] Assignment of 24-bit aircraft addresses to State aircraft. However, the first weeks FR24 ADS-B sample files contain approximately 50 flights per day with “incorrectly assigned” aircraft addresses, see Table 9 in [6] Trajectories Production Data Merging Report.

82. Flight Radar 24 normally populate the ADS-B flight data aircraft registration and aircraft type fields with the aircraft data corresponding to the 24-bit aircraft address. However, this data is not available for “incorrectly assigned” aircraft addresses. The ADS-B flight data aircraft registration and aircraft type fields for “incorrectly assigned” aircraft addresses are set to ‘00000000’ and ‘0000’ respectively.

83. Trajectories are not required for vehicles with special designators or incorrectly assigned aircraft addresses. Therefore, FR24 flights with aircraft types corresponding to the special designators (or ‘0000’) are not output.

84. The PRU are also aware of other suspicious aircraft addresses in FR24 ADS-B data. The software can also filter out trajectories with aircraft addresses in blocks of

suspicious addresses.

85. The Flight Radar 24 ADS-B data may also include synthetic data points. These are not “true” ADS-B reports and are also filtered out of the refined data

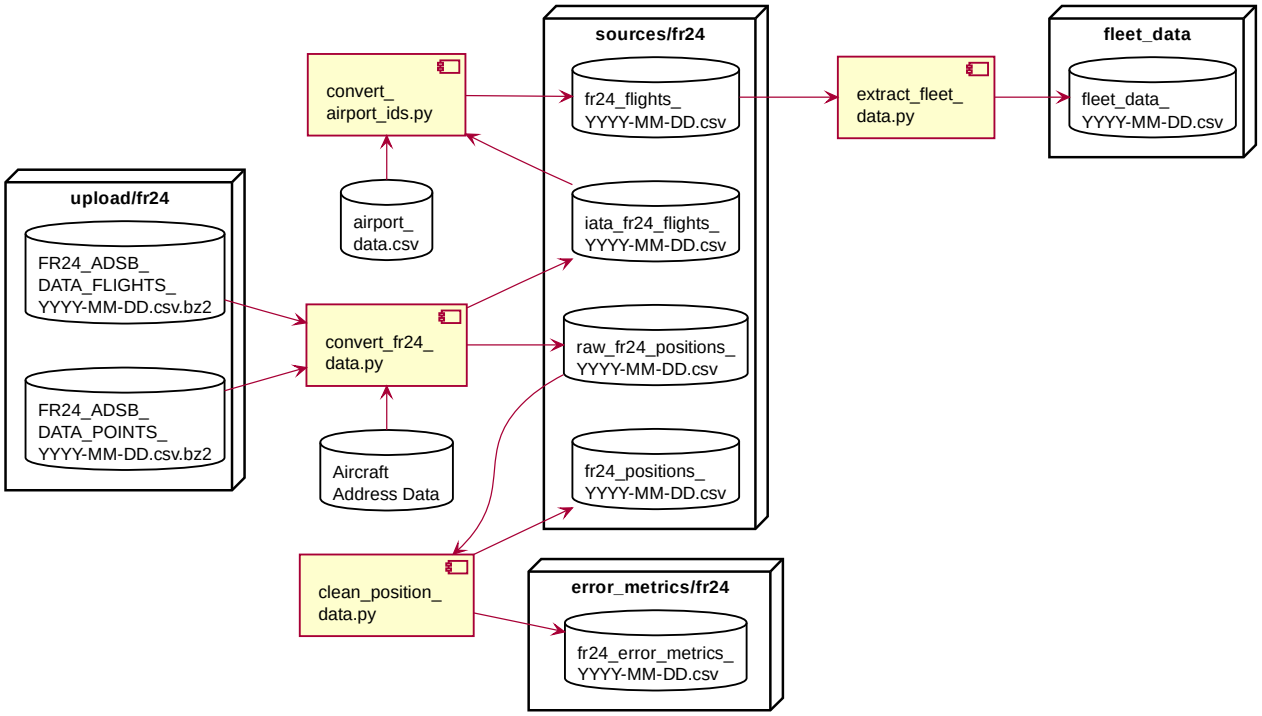

Figure 9: Refine FR24 Data

86. A pair of FR24 ADS-B (flight and positions) files are converted into a flight and positions file by the python script, convert_fr24_data.py (see Figure 9), used as follows:

```
convert_fr24_data.py <flights_filename> <points_filename>

where <flights_filename> and <points_filename> are the FR24
flights and positions files respectively.
The files may be in bz2 format or uncompressed.
```

87. It outputs a flights file and a positions file named: iata_fr24_flights_YYYY-MM-DD.csv and raw_fr24_positions_YYYY-MM-DD.csv respectively. Where YYYY-MM-DD is the date (in ISO 8601 format [14]) extracted from the FR24 filenames.

88. Note: the files must have the same date, otherwise the script will output:

```
Files are not for the same date!
flights date: YYYY-MM-DD points date:YYYY-MM-DD.
```

89. The converted FR24 position data is cleaned using the same clean_position_data.py script (see, Figure 8 and section 3.1.3) as used to clean the CPR positions (see Figure 8 and section 3.1.2).

3.1.4.1 Convert Airport Ids

90. The Flight Radar 24 ADEP and ADES fields contain 3 character IATA airport codes not the 4 character ICAO codes contained in the CPR and airport data. The departure and destination IATA airport codes must be converted to ICAO airport codes to match the CPR and airport data.

91. The IATA airport codes in the iata_fr24_flights_YYYY-MM-DD.csv file are converted into ICAO airport codes by running the python script, convert_airport_ids.py (see Figure 9), used as follows:

```
    convert_airport_ids.py <iata_fr24_flights_filename>
[airports_filename]

    where <iata_fr24_flights_filename> is the name of the
    iata_fr24_flights file
    Note: if airports_filename is not given then the file
airports.csv should be in the directory where the script is run.
```

92. It outputs a flights file with IATA airport codes transformed to ICAO airport codes and with “iata_” stripped from the start of the flights filename.

3.1.4.2 Extract Fleet Data

93. The Flight Radar 24 ADS-B flight data contains aircraft fleet data including:
    - aircraft registration number,
    - aircraft type and
    - aircraft address.

94. Fleet data is extracted from Flight Radar 24 ADS-B flight data files by running the python script, extract_fleet_data.py (see Figure 9), used as follows:

```
    extract_fleet_data.py <fr24_flights_filename>

    where <fr24_flights_filename> is the name of the fr24_flights
file.
```

95. It outputs a fleet data file file named: fleet_data_YYYY-MM-DD.csv

3.1.5 Merge CPR and FR24 Data

96. After the CPR and Flight Radar 24 ADS-B data for a day has been converted into a common format and cleaned, it can be merged, see Figure 10.

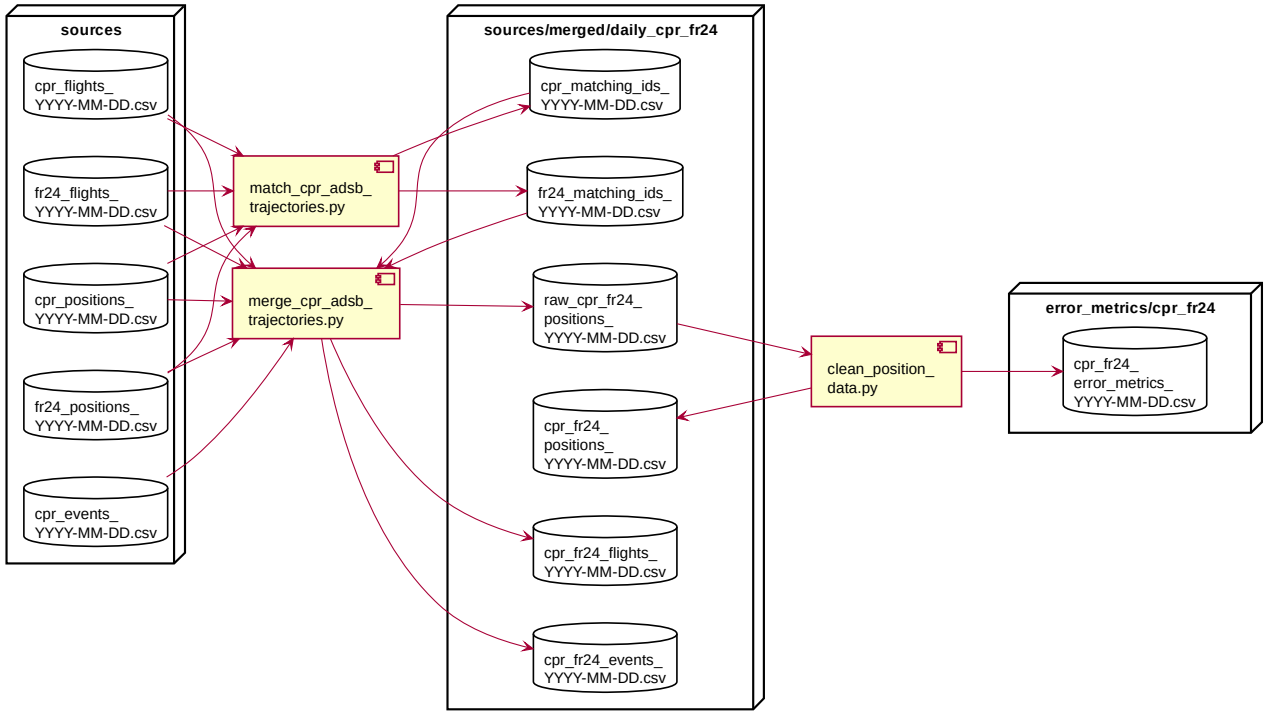

Figure 10: Merge CPR and FR24 ADS-B Data

97. As Figure 10 shows, the merging process is split into two parts: matching and merging.

98. Since trajectory data is in a common format the merging process simply involves:
- assigning a new id to the merged trajectory positions and flights,
- concatenating (and sorting) the relevant trajectory positions,
- filling in the fields of the merged flight with the fields from the appropriate source flights and merged positions.

99. The fundamental issue is matching the CPR and ADS-B trajectories to merge. The goal is to match as many trajectories as possible without matching trajectories from different flights.

100. The most accurate way of matching trajectory data is by comparing trajectory positions. However, comparing the positions of all combinations of CPR and ADS-B trajectories would be a very computationally intensive and time consuming process. It is more computationally efficient to match trajectories another way and compare trajectory positions to validate each match.

101. The CPR and ADS-B flight data have many fields in common that can be used to match trajectories, e.g.:
- callsign,

- trajectory period,
- aircraft address,
- departure and destination airports,
- and ssr code.

3.1.5.1 Callsign

102. The previous study (see [18] Feasibility Study on Cloud-Based Analytics for Performance Monitoring) found that the callsign field was a poor match, with only 20% of CPR and ADS-B flights sharing the same callsign.

Date	CPR Callsign Match %	CPR Callsign Time Matches %
2017-08-01	90.42%	87.87%
2017-08-02	90.64%	88.14%
2017-08-03	90.62%	88.24%
2017-08-04	90.67%	88.26%
2017-08-05	91.06%	88.67%
2017-08-06	90.64%	88.26%
2017-08-07	90.64%	88.00%
Totals	**90.67%**	**88.20%**

Table 7: Flight Callsign Percentages

103. Table 7 above shows the percentages of matching CPR and FR24 ADS-B callsigns relative to the number of CPR flights for each day of the first week of August 2017. On average, over 90% of the CPR flights shared their callsign with an ADS-B flight, and over 88% of the CPR flights shared their callsign with an ADS-B flight at the same time.

104. The results in Table 7 above demonstrate a considerably higher level of matching than that observed by the previous study. However, the previous study was performed using ADS-B data from FlightAware (not Flight Radar 24). The study concluded that FlightAware were using internal identification codes as callsigns, not the flight identification field transmitted by aircraft (see [19] Asterix ADS-B Target Reports Data Item I021/170).

3.1.5.2 Aircraft Address

105. Aircraft Mode-S/ADS-B transponders emit a 24-bit aircraft address. The aircraft addresses of civil flights are unique to each aircraft. Although the allocation of aircraft addresses to state aircraft is more complicated (see [17] Assignment of 24-bit aircraft addresses to State aircraft), no aircraft address should be assigned to more than one aircraft at the same time.

106. Therefore, if a CPR and an ADS-B trajectory both contain the same aircraft address then it is highly likely that they are for the same aircraft. If their periods also overlap then it is also highly likely that they are for the same flight.

107. The aircraft address is sent with every Mode-S and ADS-B squawk, so all ADS-B positions have an aircraft address. However, CPR positions only have an aircraft address if the signal was received by a Mode-S radar.

108. Table 8 shows the percentages of matching aircraft addresses relative to CPR flights with and without aircraft addresses for each day of the first week of August 2017.

Date	CPR Flights with Addresses %	Address Matches %	Address Time Matches %	Address Time Matches % of CPR Addresses	CPR Callsign Matches %
2017-08-01	68.52%	64.70%	62.34%	90.98%	98.78%
2017-08-02	68.73%	64.91%	62.62%	91.10%	98.83%
2017-08-03	67.90%	64.01%	61.90%	91.17%	99.08%
2017-08-04	68.50%	64.66%	62.49%	91.23%	98.92%
2017-08-05	69.31%	65.47%	63.14%	91.10%	98.79%
2017-08-06	68.51%	64.85%	62.09%	90.62%	98.89%
2017-08-07	68.16%	64.38%	62.03%	91.01%	98.88%
Totals	**68.51%**	**64.70%**	**62.37%**	**91.03%**	**98.88%**

Table 8: Flight Aircraft Address Percentages

109. The percentage of flights matching on aircraft address was around 65%, while 62% matched on aircraft address and time. This is lower than the percentage of flights that matched on callsign because only 68.5% of CPR flights had aircraft addresses. However, the percentage of flights matching on aircraft address and time is over 91% of the CPR flights with aircraft addresses, i.e. slightly higher than the percentage of flights matching on callsign alone (90.67%). Indicating that aircraft address and time provides a better match than callsign and time.

110. A small number of the CPR flights that matched on aircraft address and time, had different callsigns. However, nearly 99% of flights with matching aircraft addresses also had matching callsigns, see the last column in Table 8.

3.1.5.3 Departure and Destination Airports

111. Aircraft normally fly between departure and destination airports (a.k.a. city pairs); the departure and destination airport codes are recorded in the flight plan. Therefore, the departure and destination airports can also be used match CPR and ADS-B trajectories. However, ADS-B flights can only be matched after their departure and destination airport codes have been converted from three digit IATA airport codes into four digit ICAO airport codes, see section 3.1.4.1.

3.1.5.4 SSR Code

112. Another method that could be used to match CPR flights without aircraft addresses is SSR code. All aircraft transponders emit SSR codes that are received by ADS-B receivers and all ATC radars. Therefore, SSR codes could be used to match CPR flights without aircraft addresses. However, there are several issues concerned with using SSR codes to match flights over an area as large as Europe:

- more than one flight may squawk the same SSR code at the same time,
- some SSR codes are special, e.g. 7500, 7600 and 7700
- and nowadays pilots are trained NOT to switch the transponder to “standby” when changing SSR code, to ensure that TCAS is active at all times.

113. ATC attempt to use unique SSR codes within each operational airspace and often across airspace boundaries, see [20] ICAO SSR Code Assignment System. Unfortunately, there are only 4096 SSR codes available (including special SSR codes) so the SSR code squawked by an aircraft in one European airspace is quite likely to be squawked by other aircraft in different European airspaces. Therefore, multiple aircraft may squawk the same SSR code at the same time, so an accurate match can only be found by testing the positions of each pair of matching flights.

114. Also, aircraft may be assigned different SSR codes at different times during a flight. So each flight can have multiple SSR codes, each of which can be matched with other aircraft (which may also have multiple SSR codes) further increasing the numbers of potential flight positions to be tested.

115. Therefore, SSR codes are NOT used to match flight trajectories.

3.1.5.5 Matching Verification

116. All methods of matching CPR and ADS-B trajectories are capable of producing incorrect matches. Therefore, every CPR and ADS-B trajectory match is verified by calculating the proximity of their trajectory positions over a common time period.

117. The proximity test requires the trajectories to be within given horizontal and vertical distance thresholds at both the earliest and latest common times.

118. Table 9 below shows the percentages of trajectories with matching Aircraft Addresses that passed the proximity test on the first two days of August 2017 at different horizontal and vertical thresholds.

Date	2NM 500ft	2NM 5000ft	3NM 5000ft	3NM 50000ft	4NM 50000ft	5NM 50000ft
2017-08-01	69.24%	84.79%	89.94%	90.16%	93.25%	95.23%
2017-08-02	57.67%	73.70%	80.71%	81.01%	85.97%	90.15%

Table 9: Verified Aircraft Address Matching Percentages

119. The default horizontal and vertical distances are set 3NM and 50,000 feet respectively. However, they are optional parameters to match_cpr_adsb_trajectories.py and therefore they can be adjusted to optimise flight matching performance.

3.1.5.6 Merged Data Id

120. The CPR and ADS-B trajectories (and APDS data) have their own ids for each flight, which are recycled after a given period. Therefore, it is necessary to use a new id to uniquely identify each merged trajectory.

121. We use a Version 4 UUID (see, [21] UUID) to identify merged trajectories.

3.1.5.7 Multiple Matches

122. It is possible for more than one ADS-B flight to match a CPR flight and more than one CPR flight to match an ADS-B flight. For example, “match 1” in Figure 11 shows a simple match between a CPR flight and an ADS-B flight while “match 2” is a more complex match between two CPR flights and two ADS-B flights.

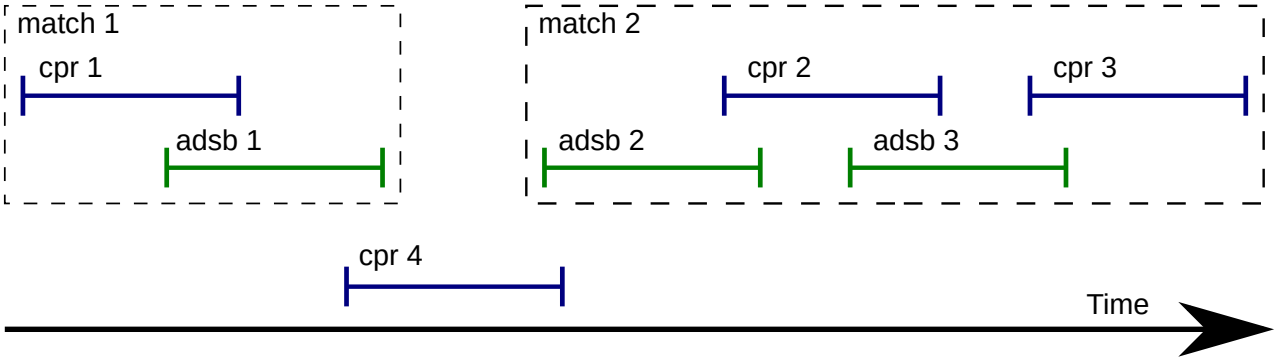

Figure 11: Matching Trajectories

123. Note: it is also possible for further matches to be found between groups of matching flights. For example flight “cpr4” in Figure 11 matches both “match 1” and “match 2” groups.

124. The many-to-many relationship between CPR and ADS-B flight ids by using the merged flight id as a foreign key, see Figure 12.

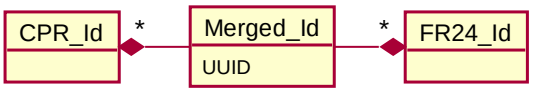

Figure 12: Flight Id Relationships

3.1.5.8 Matching Process

125. The stages of merging CPR and ADS-B trajectories are shown in Figure 13 and Figure 14.

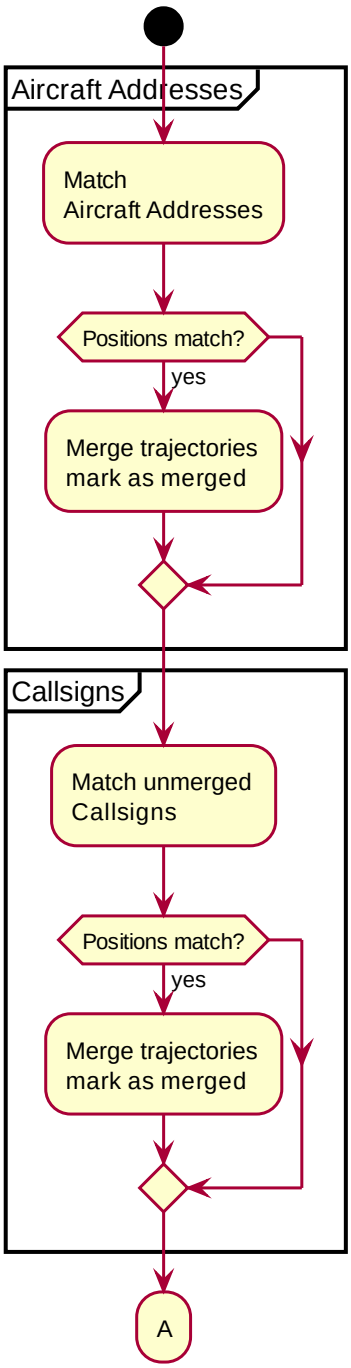

Figure 13: CPR ADS-B Matching Flowchart pt 1

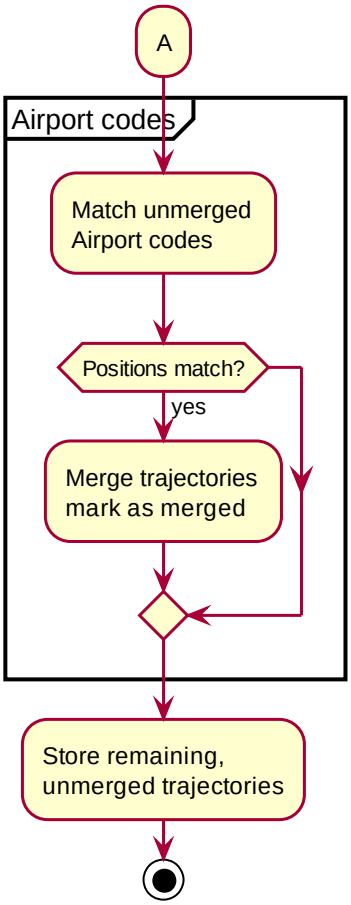

Figure 14: CPR ADS-B Matching Flowchart pt 2

126. The matching process stages are:
    1. Find CPR and ADS-B flights with the same aircraft address at the same time,
    2. Validate trajectory positions of matching flight pairs, record the valid matches,
    3. Find CPR and ADS-B flights with the same callsign at the same time,
    4. Validate trajectory positions of new matching flight pairs, record the new valid matches and merged matches.
    5. Update matches with merged matches and clear the merged matches.
    6. Find CPR and ADS-B flights with the same departure and destination at the same time.
    7. Validate trajectory positions of new matching flight pairs, record the new valid matches and merged matches.
    8. Update matches with merged matches and clear the merged matches.
    9. Find the remaining unmatched CPR and ADS-B flights and assign unused merged flight ids to them.

3.1.5.9 CPR and ADS-B Match Processing

127. Pairs of CPR and FR24 flights and positions files are compared and a pair of cpr and fr24 matching ids files are created by the python script, match_cpr_adsb_trajectories.py, used as follows:

```
    match_cpr_adsb_trajectories.py <cpr_flights_filename>
<adsb_flights_filename> <cpr_positions_filename>
<adsb_positions_filename> [distance_threshold]
[altitude_threshold]

    where <cpr_flights_filename> and <cpr_positions_filename> are
the CPR flights and positions files respectively and
<adsb_flights_filename> and <adsb_positions_filename> are the FR24
flights and positions files respectively. The files may be in bz2
format or uncompressed.
    distance_threshold is the horizontal matching distance in
Nautical Miles and altitude_threshold is the vertical matching
distance in feet.
```

128. It outputs two matching ids files named: cpr_matching_ids_YYYY-MM-DD.csv and fr24_matching_ids_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date (in ISO 8601 format [14]) extracted from the filenames.

129. Note: all input files must have the same date, otherwise the script will output:

```
    Files are not for the same date!
CPR Flights date: YYYY-MM-DD, ADSB Flights date: YYYY-MM-DD
CPR Positions date: YYYY-MM-DD, ADSB Positions date: YYYY-MM-DD.
```


3.1.5.10 CPR and ADS-B Merge Processing

130. Pairs of CPR and FR24 flights, positions and events files are merged into merged flights, positions and events files, using the matching ids files files above, by the python script, merge_cpr_adsb_trajectories.py, used as follows:

```
    merge_cpr_adsb_trajectories.py <cpr_ids_filename>
<adsb_ids_filename> <cpr_flights_filename> <adsb_flights_filename>
<cpr_positions_filename> <adsb_positions_filename>
<cpr_events_filename>

     where <cpr_ids_filename> and <adsb_ids_filename> are the
matching ids from match_cpr_adsb_trajectories.py,
<cpr_flights_filename>, <cpr_positions_filename> and
<cpr_events_filename> are the CPR flights, positions and events
files respectively and <adsb_flights_filename> and
<adsb_positions_filename> are the FR24 flights and positions files
respectively.
    The files may be in bz2 format or uncompressed.
```

131. It outputs files named: cpr_fr24_flights_YYYY-MM-DD.csv.bz2, raw_cpr_fr24_positions_YYYY-MM-DD.csv.bz2 and cpr_fr24_events_YYYY-MM-DD.csv.bz2 containing the merged flights, positions and events respectively. Where YYYY-MM-DD is the date (in ISO 8601 format [14]) extracted from the filenames.

132. Note: the files must have the same date, otherwise the script will output:

```
    Files are not for the same dates:
```
YYYY-MM-DD, YYYY-MM-DD, YYYY-MM-DD, YYYY-MM-DD, YYYY-MM-DD, YYYY-MM-DD, YYYY-MM-DD.

3.1.6 Merge Consecutive Days Data

133. To output gate-to-gate trajectories for flights that fly over midnight, it is necessary to merge trajectory data from consecutive days.

134. As with CPR and ADS-B merging, consecutive day merging is divided into matching and merging, see Figure 15.

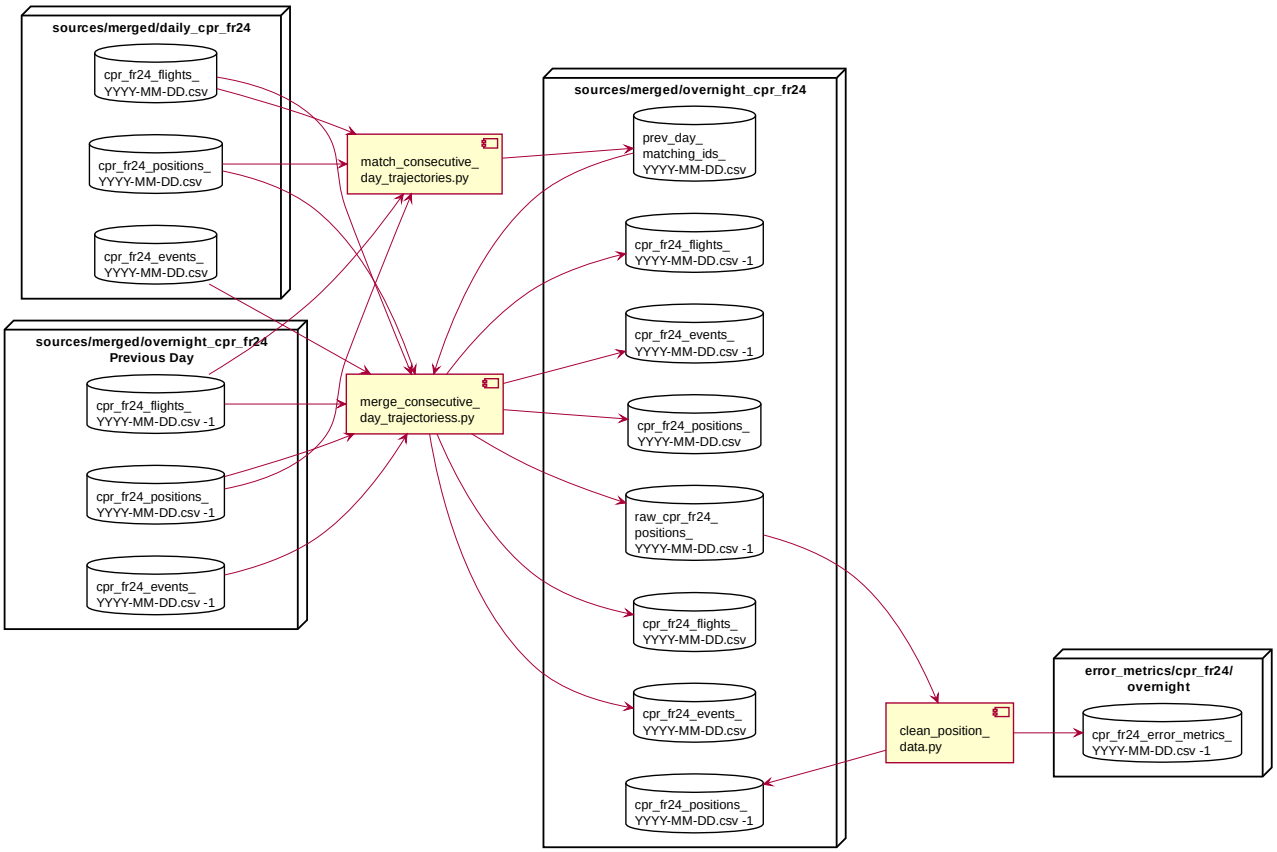

Figure 15: Merge Overnight Flight Data

3.1.6.1 Matching Overnight Flights

135. Consecutive day flights are matched that started and finished within a given time of each other and had the same aircraft address or callsign.

136. For example, Table 10 shows the number of CPR and ADS-B merged flights on each day of the first week of August 2017 with the number of merged flights that had the same aircraft address and/or callsign within 3 minutes of each other.

137. Note: the overall overnight matches in Table 10 is the callsign time matches / number of flights.

138. The matches are verified by calculating the time and horizontal distance between ends the trajectories and calculating the speed required to fly between them. The maximum speed threshold is an optional parameter to match_cpr_adsb_trajectories.py with a default value of 750 Knots.

Date	Flights	Aircraft Address Time Matches	Callsign Time Matches	Overall Overnight Matches %
2017-08-01/02	53270	699	1077	2.02%
2017-08-02/03	50151	679	1076	2.15%
2017-08-03/04	53913	695	1087	2.02%
2017-08-04/05	52944	694	1046	1.98%
2017-08-05/06	57765	863	1284	2.22%
2017-08-06/07	54075	777	1182	2.19%
Totals	**322118**	**4407**	**6752**	**2.10%**

Table 10: Merged Consecutive Day Flights: maximum 3 minutes apart

3.1.6.2 Merging Consecutive Day Flights

139. The trajectory positions of the next days flight are extracted from the positions data. Their ids are changed to the ids of the previous days flights and then merged with the previous days positions.

140. The flights, events and positions are removed from the next days flights, events and positions files to avoid duplication.

3.1.6.3 Consecutive Day Match Processing

141. Pairs of consecutive days flights and positions files are compared and a matching ids file is created by the python script, match_consecutive_day_trajectories.py, used as follows:

```
    match_consecutive_day_trajectories.py <prev_flights_filename>
<next_flights_filename> <prev_positions_filename>
<next_positions_filename> <prev_events_filename>
<next_events_filename> [maximum time difference] [max speed]

    where <prev_flights_filename> and <prev_positions_filename>
are the previous days flights and positions files respectively and
<next_flights_filename> and <next_positions_filename> are the next
days flights and positions files respectively.
    The files may be in bz2 format or uncompressed.
    maximum time difference is the maximum time between
trajectories in seconds, default 150 seconds.
     max speed is the maximum speed between trajectories in
Knots, default 750 Knots.
```

142. It outputs a matching ids file named: prev_matching_ids_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date of the next day (in ISO 8601 format [14]) extracted from the next days filenames.

143. Note: the files and positions files must have the same dates and the next date must be after the previous date, otherwise the script will output:

`Files are not for the correct dates prev flights date:` YYYY-MM-DD, next `Flights date:` YYYY-MM-DD, prev `Positions date:` YYYY-MM-DD, next `Positions date:` YYYY-MM-DD.

3.1.6.4 Consecutive Day Merge Processing

144. Pairs of merged flights and positions files are merged using the matching ids file above with the python script, merge_consecutive_day_trajectories.py, used as follows:

```
merge_consecutive_day_trajectories.py <prev_ids_filename>
<prev_flights_filename> <next_flights_filename>
<prev_positions_filename> <next_positions_filename>
<prev_events_filename> <next_events_filename>

 where <prev_ids_filename> is the matching ids from
match_consecutive_day_trajectories.py,
<prev_flights_filename>, <prev_positions_filename> and
<prev_events_filename> are the previous days flights positions and
events files respectively and
<next_flights_filename>, <next_positions_filename> and
<next_events_filename> are the next days flights, positions and
events files respectively.
The files may be in bz2 format or uncompressed.
```

145. `It` outputs new flights, positions and events files with “new_” prepended to the input flights, events and positions filenames.

3.1.7 Refine APDS Data

146. Airport (APDS) data is received from a number of European airports documenting aircraft departures, arrivals and positions where aircraft intersected 40NM and 100NM cylinders centred on the airports, see [8]. APDS entries are supplied in CSV format for a month.

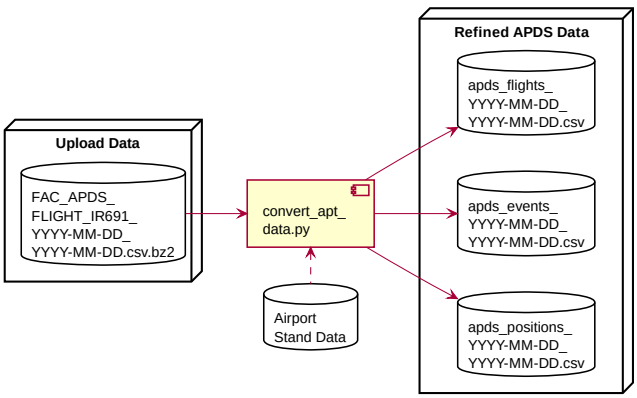

Figure 16: Refine APDS Data

147. APDS data is assigned APDS_IDs. However, the data is recorded separately for destinations and departures. Therefore, the destination and departure of the same flight will have different APDS_IDs.

148. APDS data contains: SCHEDULED_OFF_BLOCK, GATE_OUT and WHEELS_OFF times for departure flights and: SCHEDULED_IN_BLOCK, GATE_IN and WHEELS_ON times for arrival flights. These times are used to create APDS event records.

149. Where airport stand position data is available, the locations of the airport stands are combined with the GATE_OUT and GATE_IN event times, to create position records.

150. Note: the APDS data includes cylinder intersection positions at radii of 40NM and 100NM from each airport. However, the PRU does NOT require these intersection positions.

3.1.8 Merge APDS Data

151. As with the other merging processes, APDS merging is divided into matching and merging, see Figure 17.

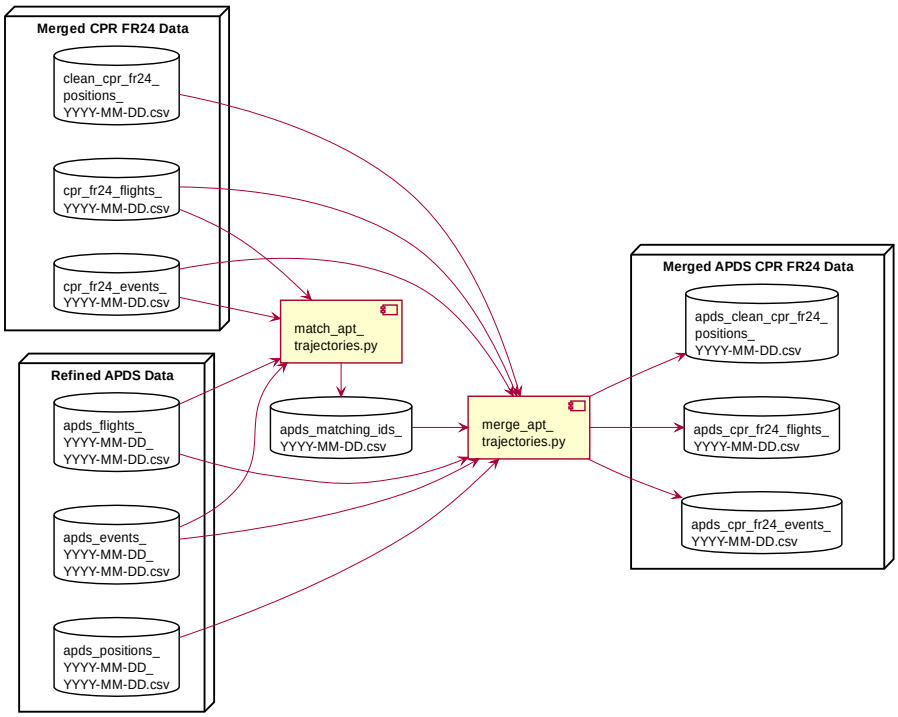

Figure 17: Merge APDS Data

3.1.8.1 Matching APDS Flights

152. APDS data does not contain aircraft addresses or SSR codes so APDS “flights” are matched using flight callsigns and departure/destination city pairs.

153. Also, APDS flight data does not contain time information, APDS times are recorded

in APDS event data. Therefore, APDS data is matched chronologically by event times, e.g. the time of the scheduled off block event from CPR data.

154. After matching, APDS data (events and positions) is added to merged CPR and ADS-B data. So the APDS matching process determines which APDS ids match merged CPR/ADS-B data ids for each day of merged CPR/ADS-B data.

3.1.8.2 Merging APDS Trajectories

155. The ids of matching APDS events and positions are replaced with merged data ids. The APDS positions and events are then merged with the CPT and ADS-B merged data positions and events for the given day.

3.1.8.3 APDS Flight Match Processing

156. A days flights and event files are compared to the APDS flights and event files for the month covering the day and a matching ids file is created by the python script, match_apt_trajectories.py, used as follows:

```
    match_apt_trajectories.py <daily_flights_filename>
<apds_flights_filename> <daily_events_filename>
<apds_events_filename>

    where <daily_flights_filename> and <daily_events_filename>
are the days flights and events files respectively and
<apds_flights_filename> and <apds_events_filename> are the APDS
flights and events files respectively.
    The files may be in bz2 format or uncompressed.
```

157. It outputs matching ids files named: apds_matching_ids_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date of the daily data (in ISO 8601 format [14]) extracted from the daily filenames.

158. Note: the daily files and positions files must have the same dates:

```
    Files are not for the same dates! Flights date:
YYYY-MM-DD, Events date: YYYY-MM-DD
```

3.1.8.4 APDS Trajectory Merge Processing

159. Daily merged events and positions files are merged using the matching ids file above with the python script, merge_apt_trajectories.py, used as follows:

```
    merge_apt_trajectories.py <apds_ids_filename>
<daily_positions_filename> <apds_positions_filename>
<daily_events_filename> <apds_events_filename>

     where <apds_ids_filename> is the matching ids from
match_apt_trajectories.py, <daily_positions_filename> and
<daily_events_filename> are the days flights, positions and events
files respectively, <apds_positions_filename> and
<apds_events_filename> are the APDS flights, positions and events
```



```
files respectively.
     The files may be in bz2 format or uncompressed.
```

160. It outputs new flights, positions and events files with “apds_” prepended to the input flights and positions filenames.

3.2 Trajectory Analysis and Interpolation

161. Trajectory analysis creates trajectory meta data and quality metrics by smoothing trajectory position data. Trajectory interpolation interpolates smoothed trajectory meta data to create production trajectories, see Figure 18.

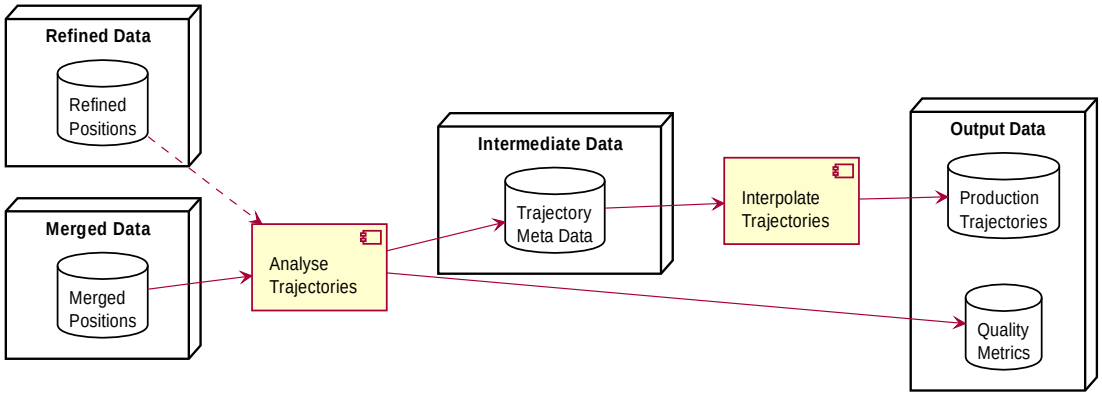

Figure 18: Trajectory Analysis and Interpolation

3.2.1 Trajectory Analysis

162. Trajectory analysis smooths trajectories by considering trajectory positions in: horizontal, temporal and vertical dimensions.

163. The horizontal path flown by an aircraft is derived from its time ordered trajectory positions and defined by a pair of sequences:

- path waypoints
- and turn initiation distances at the path waypoints.

164. It assumes that aircraft either fly straight (i.e. along route legs or on headings), or that they perform turns (either between route legs or onto headings). Manoeuvres such as holds, circuits, etc. are combinations of straight legs and turns.

165. After the horizontal path has been derived, trajectory positions are ordered by their distance (and then time) along the derived path.

166. Time and altitude profiles are derived from plots of times vs derived path distance and altitude vs derived path distance respectively.

167. The altitude profile is analysed to detect sections where the aircraft was in level flight. Climbing and descending sections (between level sections) are then smoothed using standard curve fitting algorithms.

168. The time profile is smoothed using a moving average filter which calculates the ground speed from different combinations of trajectory positions to account for time and/ or position errors from different surveillance sources.

169. The resulting smoothed trajectory consists of three elements (see Figure 19):
- a derived horizontal path,
- a smoothed time profile
- and a smoothed vertical profile.

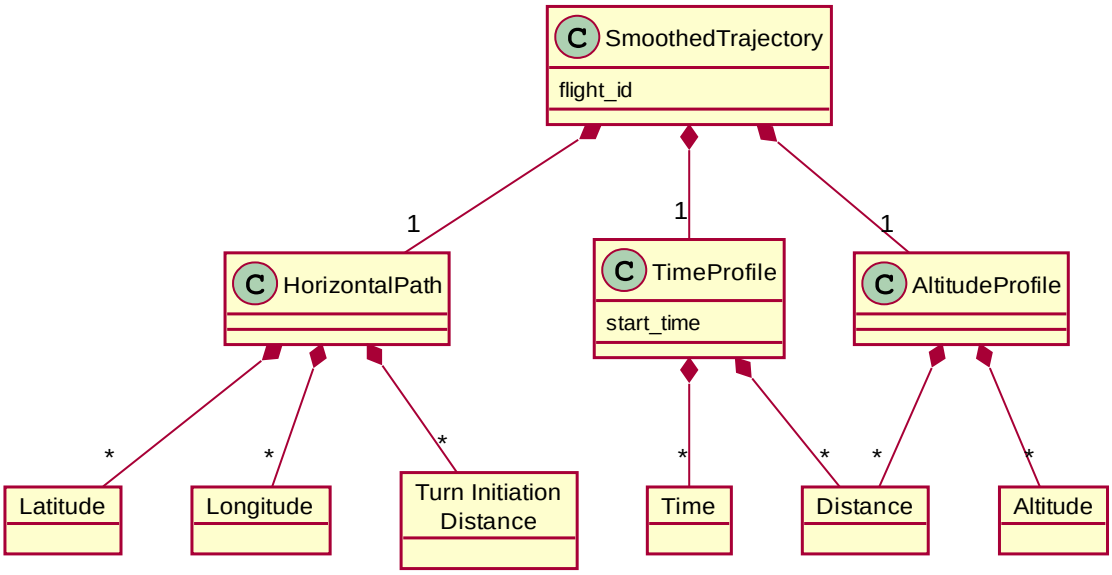

Figure 19: Smoothed Trajectory Class Diagram

170. The Trajectory Analysis smoothing algorithm is described in detail in Appendix B.

3.2.1.1 Trajectory Analysis Processing

171. Cleaned trajectory positions are analysed by the python script, analyse_position_data.py (see Figure 20) used as follows:

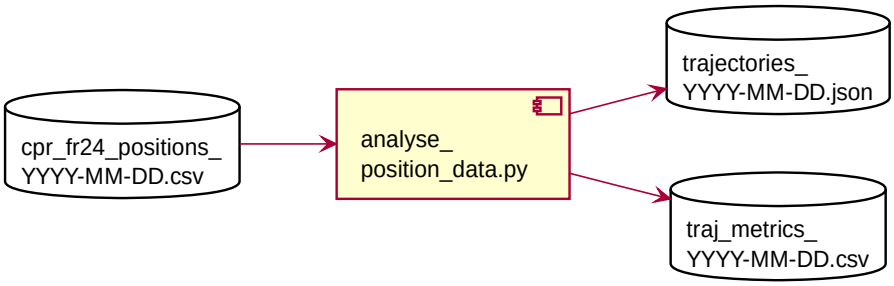

Figure 20: Position Analysis

```
    analyse_position_data.py <positions_filename>
[across_track_tolerance][time analysis method]
```



```
[median_filter_samples] [average_filter_samples]
[speed_max_duration] [logging_msg_count]

     where <positions_filename> is a positions file (bzip2
format or uncompressed) named as described in section
3.1.1.2,

across_track_tolerance is the across_track_tolerance in
Nautical Miles, default 0.5 Nautical Miles

time analysis method is the time analysis method, default
moving average speed ('mas')

median_filter_samples is the number of samples to use in the
speed moving median filter, default 5

average_filter_samples is the number of samples to use in the
speed moving average filter, default 3

speed_max_duration is the maximum time between to use the
custom average filter, default 120 seconds

and logging_msg_count is the number of trajectories between
each logging message, default 5000.
```

172. It outputs a trajectories file named: trajectories<date>.json and a trajectory metrics file named: traj_metrics<date>.csv. Where <date> is the input filename date in ISO8601 format, see [14].

3.2.2 Trajectory Interpolation

173. Production trajectories are calculated from the smoothed trajectory:
    - time profile,
    - horizontal path
    - and altitude profile.

174. Trajectory path distances corresponding to trajectory times at the required intervals (default 5 seconds) are calculated from the time profile using the scipy.interpolate.CubicSpline class, see [26].

175. Corresponding horizontal positions and are then calculated from the horizontal path at the path distances. Corresponding altitudes are calculated from the altitude profile at the path distances using the scipy.interpolate.interp1d class, see [27].

176. The ground speeds and vertical speeds of the output positions are calculated from the time profile and altitude profile respectively. Ground tracks are calculated from the derived path waypoints and calculated positions.

177. Note: the ground track is calculated at every position because even when aircraft fly in a straight line (i.e. along a Great Circle) their ground track varies with position, unless the aircraft is flying along a Great Circle that passes through the North and South poles.

3.2.2.1 Trajectory Interpolation Processing

178. Smoothed trajectories are interpolated by the python script, interpolate_trajectories.py (see Figure 20) used as follows:

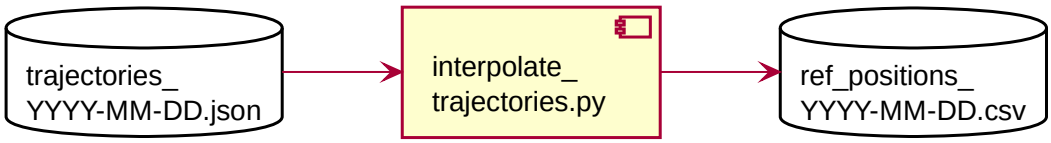

Figure 21: Trajectory Interpolation

```
interpolate_trajectories.py <trajectories_filename>
[straight_interval] [turn_interval]

where <trajectories_filename> is a trajectories file (bzip2
format or uncompressed) named as described in section 3.2.1.1, and
straight_interval and turn_interval are the times between output
positions, default 5 seconds and 5 seconds.
```

179. It outputs a positions file named: ref_positions<date>.csv. Where <date> is the input filename date in ISO8601 format, see [14].

Section 4

4 Trajectory Assessment

180. The Trajectory Assessment module is required to find spatial and temporal intersections with airspace volumes.

181. Airspace volumes can be:

- standard segments, i.e. elementary sectors and radial distances from airports,
- user-defined dimensions.

182. Elementary airspace sectors are defined by a range of flight levels and the vertices of closed polygons, Radial distances from airports are modelled by cylindrical volumes, centred on the airports.

183. The Trajectory Assessment module uses a GIS database to find spatial intersections with both sets of polygons and with cylindrical volumes.

4.1 Airspace Volumes

4.1.1 Elementary Airspaces

184. Elementary airspace sectors are 3D volumes, defined vertically by a range of flight levels (altitudes) from a bottom to a top level and horizontally by WGS84 latitude and longitude coordinates of the vertices of closed polygons. Elementary sectors cover the whole of an airspace without any gaps. For example, Figure 22 shows the first elementary airspace sector in the supplied data.

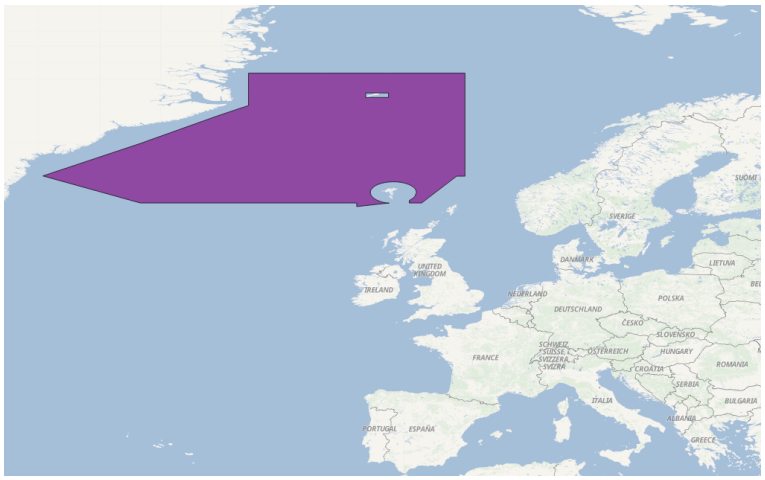

Figure 22: An Elementary Sector

4.1.2 Airport Cylinders

185. A cylindrical volume around an airport may be defined by a circle of a given radius centred at the Airport Reference Point (ARP) latitude and longitude coordinates.

186. A cylindrical volume is assumed to extend from ground level to infinity Therefore, the horizontal coordinates of a 3D airport cylinder intersection are the same as the horizontal coordinates of a 2D airport cylinder intersection.

187. Airport cylinder intersections are only required with trajectories that depart or arrive at the airports; they are not required for trajectories that fly over the airports.

4.1.3 User Defined Airspaces

188. Users may define their own airspaces as closed polygons spanning a vertical range or as cylinders around given positions.

189. User defined airspaces are defined in the same format as elementary airspaces and airport cylinders.

4.2 Sector Intersections

4.2.1 Trajectories

190. The output production trajectories are time ordered sequences of 4D (time, latitude, longitude and altitude) positions at regular (minimum 5 second) intervals.

191. Production trajectories are interpolated from smoothed trajectories produced by trajectory analysis (see [7] section 3.1). Smoothed trajectories are comprised of:
- Horizontal Paths,
- Vertical Profiles
- and Time Profiles.

192. Interactions are found with smoothed trajectories.

193. For the purpose of elementary sector intersections, there are two types of trajectories:
- 2D trajectories – which only occupy a single altitude
- and 3D trajectories – which occupy a range of altitudes.

194. It is only necessary to find horizontal intersections with elementary sectors that span the altitude of a 2D trajectory.

195. However, not only is it necessary to find horizontal (lateral) intersections with elementary sectors that span the altitude range of a 3D trajectory, it is also necessary to find vertical intersections, where a trajectory climbs (or descends) through the bottom (or top) of an elementary sector.

4.2.2 Horizontal Intersections

196. A 2D or 3D trajectory may only intersect elementary sectors that vertically span the altitude range of the trajectory, i.e.

```
sector bottom altitude <= trajectory maximum altitude and
trajectory minimum altitude < sector top altitude
```

197. Otherwise, the trajectory flies over or under the elementary sectors.

198. For example, Figure 23 shows the vertical profiles of six trajectories (labelled 1 to 6) around a sector (A, in blue). The vertical profiles of trajectories 1, 4 and 5 intersect sector A, while trajectory 2 flies under it and trajectories 3 and 6 fly over it.

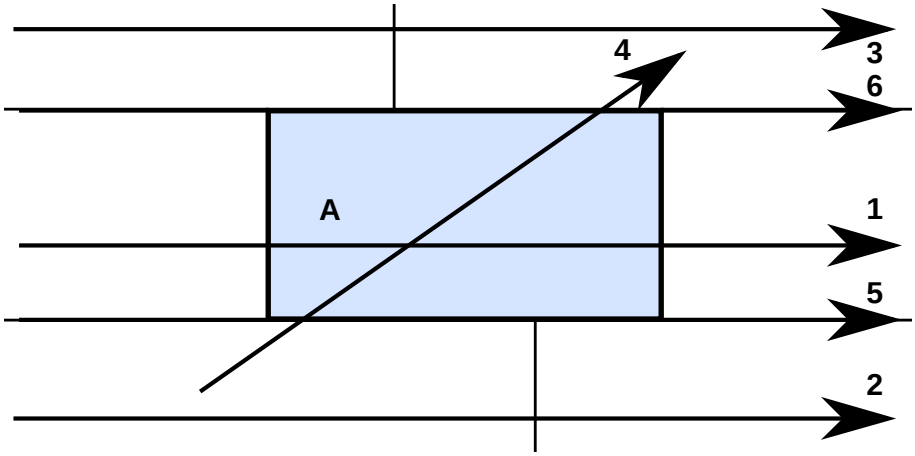

Figure 23: Trajectory and Sector Vertical Profiles

199. Note: because trajectories 5 and 6 fly along the opposite vertical boundaries of the sector, only one of the trajectories intersects the sector. In this case it is trajectory 5 where it has the same sector intersection points as trajectory 1. Trajectory 6 is at the top altitude of sector A and therefore does not intersect it.

200. The precise location of horizontal intersections can be found using the ST_Intersection standard spatial database function.

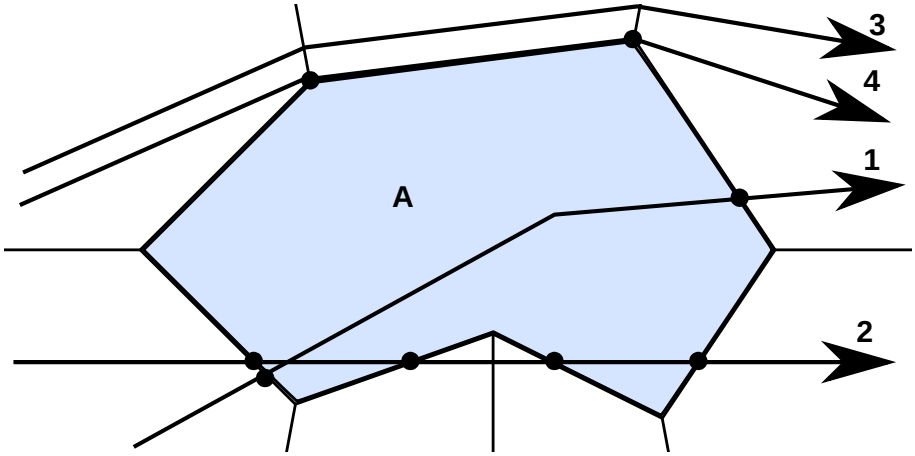

Figure 24: Horizontal Path and Sector Intersections

201. For example, Figure 24 shows the horizontal paths of four trajectories (labelled 1 to 4) around a sector (A, in blue). The paths of trajectories 1 and 2 both intersect sector A (trajectory 2, twice), while trajectory 3 flies past it.

202. There is a fourth case (trajectory 4), where a trajectory flies along a border. In this case the border is considered to be in one sector or the other (not both!). Precisely which sector the border is considered to be in depends upon their relative orientation. For example, it may be in the sector to the North or East and not the sector to the South or West, or vice versa. Trajectory 4 assumes that the border lies to the North and therefore it has two intersections at either end of the adjacent border.

4.2.3 Vertical Intersections

203. A 3D trajectory may only vertically intersect elementary sectors that it also intersects horizontally. Furthermore, a 3D trajectory may only vertically intersect an elementary sector where its altitude range spans the bottom or top of the elementary sector, i.e.:

```
trajectory minimum altitude < sector bottom altitude and
sector bottom altitude < trajectory maximum altitude and

or

trajectory minimum altitude < sector top altitude and
sector top altitude < trajectory maximum altitude and
```

204. The trajectory altitude range above is the altitude range of the 3D trajectory segment inside the elementary sector, i.e. from its horizontal elementary sector entry point to its horizontal elementary sector exit point.

205. The trajectory segment altitude range determines if (and how) a 3D trajectory segment intersects the elementary sector;

- trajectory segment altitude range outside sector altitude range – no intersection;
- trajectory segment altitude range inside sector altitude range – horizontal intersection(s) only;
- minimum trajectory segment altitude below sector bottom altitude – sector bottom intersection;
- maximum trajectory segment altitude above (or equal to) sector top altitude – sector top intersection.

206. The precise locations of sector bottom and sector top intersections are derived from the path distances along the vertical profile where the altitude equals the bottom/top elementary sector altitude(s).

207. The latitude and longitude coordinates are then derived at the path distances along the horizontal path.

4.2.4 2D and 3D Intersections

208. The GIS database only finds horizontal intersections not vertical intersections, which is fine for finding intersections for cruising flights and airport cylinder intersections. However, where a trajectory spans a large range of altitudes (e.g. climbing and/or descending) it may find intersections with sectors that are vertically separated, see Figure 25.

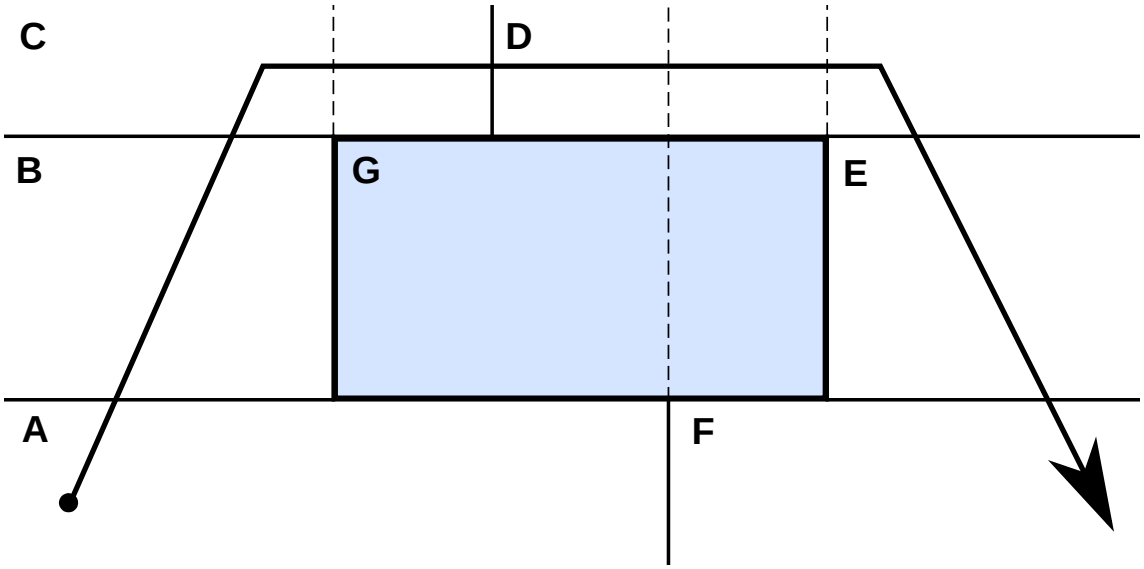

Figure 25: Trajectory and Sector Vertical Profiles

209. Horizontal sector intersections can be used to calculate the section of a trajectory within a sector so that the appropriate section of the trajectory altitude profile can be compared with the vertical range of the sector. For example in Figure 25 the altitude range of the trajectory spans sector G, but the section of trajectory between the boundaries with sector B and sector E is above the altitude range of sector G.

210. In order to determine which sections of each trajectory were within each sector, it is also necessary to know whether a trajectory originated in a sector or not. For example in Figure 25 the trajectory originates in sectors A, B and C so the first horizontal intersections between those sectors and their adjacent sectors are exit intersections not entry intersections.

211. Horizontal sector intersections are only valid if the altitude of the trajectory at the horizontal sector intersection point is within the altitude range of the sector. For example in Figure 25 only the horizontal intersection between sectors C and D is valid. Other horizontal intersections such as those between sectors A and F or sectors B and G are just used to calculate where vertical intersections occur.

212. Vertical intersections can occur wherever the altitude of a trajectory crosses the sector top or bottom altitude. Climbing and descending trajectories can have multiple vertical sector intersections instead of horizontal sector intersections. For example in Figure 25 sectors B and E only have vertical intersections, while sectors C and D have both a vertical and a horizontal intersection.

4.2.5 Sector Intersection Fields

213. Sector intersections are output in a .csv file containing the following fields:

Field	Description
FLIGHT_ID	The id of the flight/trajectory.
SECTOR_ID	The id of the sector.
IS_EXIT	True for sector exit positions, false for sector entry positions.
LAT	WGS-84 Latitude in signed decimal degrees, North positive.
LON	WGS-84 Longitude in signed decimal degrees, East positive.
ALT	The barometric altitude (at 1013.25hPa) in feet.
TIME	The UTC time to milliseconds precision (or greater) in ISO 8601 Date and Time format see [14].
DISTANCE	The distance along the horizontal path in Nautical Miles.

Table 11: Sector Intersection Fields

214. Trajectory sector intersections are found by the python script, find_sector_intersections.py, used as follows:

```
find_sector_intersections.py <filename>

where <filename> is a trajectories JSON file
```

215. It outputs a sector intersections file named: sector_intersections_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date extracted from the trajectories filename in ISO 8601 Date and Time format see [14].

4.3 Airport Intersections

216. Airport cylinder intersections are required for aircraft that depart or arrive at given airports.

217. Both airport departure and arrival intersections require trajectory positions that span the cylinder, i.e. the trajectory must have positions both closer to and further than the radius from the given Airport Reference Point (ARP).

218. For example, Figure 26 shows the horizontal paths of six trajectories (labelled 1 to 6) around an airport reference point (A, in blue). The paths of trajectories 1, 2 and 3 intersect the cylinder around A: 1 is a departure, 2 is an arrival and trajectory 3 flies past the airport, through the cylinder. Trajectory 4 departs airport A, but does not reach as far as the cylinder to intersect it.

219. There are also boundary cases where a trajectory is on the border of a cylinder, i.e. the distance of the closest point of a trajectory is at radius from the centre of the cylinder, see trajectory 5 and 6 in Figure 26. These cases are not considered for airport cylinders, but they are considered for user defined cylinders, see section 4.4.2.

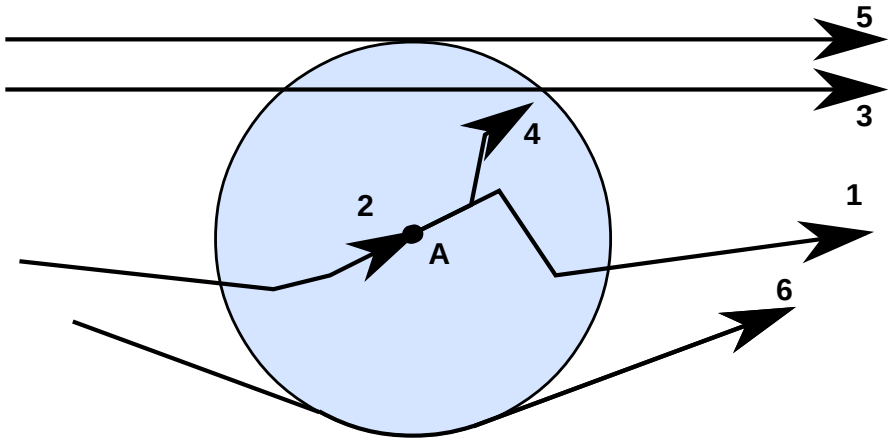

Figure 26: Trajectory Cylinder Intersections

220. In these boundary cases, an intersection occurs if the trajectory crosses the cylinder border, i.e. if the closest distance is less than the radius. If the closest distance of a trajectory to a cylinder centre is greater than or equal to the radius then there is no intersection. E.g. see trajectory 5 and 6 in Figure 26.

4.3.1 Airport Intersection Fields

221. Airport intersections are output in a .csv file containing the following fields:

Field	Description
FLIGHT_ID	The id of the flight/trajectory.
AIRPORT_ID	The ICAO code of the airport.
IS_ARRIVAL	True for airport arrivals, false for airport departures.
LAT	WGS-84 Latitude in signed decimal degrees, North positive.
LON	WGS-84 Longitude in signed decimal degrees, East positive.
ALT	The barometric altitude (at 1013.25hPa) in feet.
TIME	The UTC time to milliseconds precision (or greater) in ISO8601 Date and time format.
DISTANCE	The distance along the horizontal path in Nautical Miles.

Table 12: Airport Intersection Fields

4.3.2 Airport Intersection Processing

222. Trajectory airport intersections are found by the python script, find_airport_intersections.py, used as follows:

```
find_airport_intersections.py <flights_filename>
```



```
<trajectory_filename>

     where <flights_filename> and <trajectory_filename> are the
csv flights file and the JSON trajectories file respectively.
```

223. It outputs an intersection positions file named: airport_intersections_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date extracted from the flights and trajectories filenames in ISO 8601 format.

4.4 User Defined Airspaces

224. Users may define their own airspaces as closed polygons or as cylinders around given positions, between ranges of flight levels (altitudes).

4.4.1 Polygonal Airspaces

225. Polygonal airspaces are 3D volumes, defined vertically by a range of flight levels (altitudes) from a bottom to a top level and horizontally by WGS84 latitude and longitude coordinates of the vertices of closed polygons.

226. Intersections are found with polygonal airspaces in the same way that intersections are found with elementary airspaces, see section 4.2.

4.4.2 Cylindrical Airspaces

227. Cylindrical airspaces are defined vertically by ranges of flight levels (altitudes) from a bottom to a top level and horizontally by circles of a given radii centred at WGS84 latitude and longitude coordinates.

228. An example of a cylindrical airspace is a “no fly zone” around a crash site. Such a cylindrical airspace may have vertical limits.

229. Intersections are found with cylindrical airspaces in a similar way to the way that intersections are found with airport cylinders, see section 4.3.

230. The differences between user defined cylinders and airport cylinders are:

- Any trajectory can intersect a user defined cylinder, not just trajectories that arrive or depart within a cylinder;
- Intersections are found with trajectories that pass through a user defined cylinder as well as those that start or finish in a user defined cylinder. For example, trajectory 3 in Figure 26 has both entry and exit intersections with a user defined cylinder that would not be found with an airport cylinder.

4.4.3 User Defined Intersection Fields

231. User defined intersections are output in a .csv file containing the following fields:

Field	Description
FLIGHT_ID	The id of the flight/trajectory.
USER_ID	The id of the user defined airspace.
IS_EXIT	True for airspace exit positions, false for airspace entry positions.
LAT	WGS-84 Latitude in signed decimal degrees, North positive.
LON	WGS-84 Longitude in signed decimal degrees, East positive.
ALT	The barometric altitude (at 1013.25hPa) in feet.
TIME	The UTC time to milliseconds precision (or greater) in ISO8601 Date and time format.
DISTANCE	The distance along the horizontal path in Nautical Miles.

Table 13: User Defined Intersection Fields

4.4.4 User Defined Intersection Processing

232. Trajectory user defined intersections are found by the python script, find_user_intersections.py, used as follows:

```
find_user_intersections.py <filename>

where <filename> is a trajectories JSON file
```

233. It outputs a user intersections file named: user_intersections_YYYY-MM-DD.csv. Where YYYY-MM-DD is the date extracted from the trajectories filename in ISO 8601 Date and Time format see [14].

Section 5

5 System Infrastructure

5.1 Storage

234. Google Buckets

5.2 Subdirectory Structure

235. The bucket has the following directory structure:

```
pru-ta
|
|--airports
|  |--stands
|--airspaces
|  |--elementary
|  |--user_defined
|--backups
|  |--user1
|  |--user2
|  |--user3
|--products
|  |--airport_intersections
|  |  |--cpr
|  |  |--cpr_fr24
|  |  |--fr24
|  |--error_metrics
|  |  |--cpr
|  |  |--cpr_fr24
|  |  |--|--overnight
|  |  |--fr24
|  |--fleet_data
|  |--ref_positions
|  |  |--cpr
|  |  |--cpr_fr24
|  |  |--fr24
|  |--sector_intersections
|  |  |--cpr
|  |  |--cpr_fr24
|  |  |--fr24
|  |--traj_metrics
|  |  |--cpr
|  |  |--cpr_fr24
|  |  |--fr24
|  |--trajectories
```



```
|   |   |--cpr
|   |   |--cpr_fr24
|   |   |--fr24
|   |--user_defined_intersections
|   |   |--cpr
|   |   |--cpr_fr24
|   |   |--fr24
|--sources
|   |--apds
|   |--cpr
|   |--fr24
|   |--merged
|   |   |--apds_cpr_fr24
|   |   |--daily_cpr_fr24
|   |   |--overnight_cpr_fr24
|--upload
|   |--apds
|   |--cpr
|   |--fr24
```

5.2.1 Airports

236. This directory contains airport data, including ICAO/IATA airport codes airport positions data, etc. The stands sub-directory contains airport stand data for creating positions from airport movement data.

5.2.2 Airspaces

237. This directory contains airspace data, including elementary and user defined airspaces for airspace intersections.

5.2.3 Backups

238. This directory contains backups of notebooks, etc.

5.2.4 Products

239. The products directory contains directories for processing outputs, i.e.:

- airport, sector and user intersections;
- position cleaning error metrics;
- extracted fleet data;
- reference trajectory positions;
- trajectory meta data;
- and trajectory analysis metrics.

240. Note, each directory has a subdirectory for each of the different processed data sources:

- cpr,

- fr24
- and merged cpr_fr24.

5.2.5 Sources

241. The sources directory contains directories for each of the different processed data sources:
    - apds,
    - cpr
    - and fr24.
242. There is also a merged directory which contains the products of the merged data sources:
    - apds_cpr_fr24
    - and cpr_fr24.

5.2.6 Upload

243. The upload directory also contains directories for each of the different processed data sources:
    - apds,
    - cpr
    - and fr24.
244. The upload directory will be the drop location for incoming files. New source data files will be dropped in here and read by the refining operations.
245. The files in this directory should be deleted after they have been refined.

5.3 File Naming

246. All data files follow the same naming convention:
    - data state;
    - data source;
    - data type;
    - date;
    - file type suffix.
247. e.g. raw_cpr_fr24_positions_2017_08_25.csv.

5.3.1 Data State

248. The data state is optional and only set on files that require further processing, i.e.:

- raw – prepended to position files that have yet to be cleaned
- new – a prepended to merged cpr_fr24 files after overnight merging.

5.3.2 Data Source

249. The source of the original data, currently:

- apds – APDS airport movement data,
- cpr – Correlated Postion Report (CPR) data,
- fr24 – Flight Radar 24 ADS-B data,
- cpr_fr24 – merged cpr and fr24 data,
- apds_cpr_fr24 – merged apds, cpr and fr24 data.

5.3.3 Data Type

250. The type of the data, one of:

- events – event data,
- flights – flight data,
- positions – position data,
- error_metrics – position cleaning error metrics,
- fleet_data – fleet data extracted from Flight Radar 24 flight data,
- trajectories – trajectory position data,
- traj_metrics – trajectory position analysis metrics,
- ref_positions – reference trajectory positions,
- airport_intersections – airport cylinder intersections,
- sector_intersections – elementary sector intersections,
- user_intersections – user defined airspace intersections.

5.3.4 Date

251. The date in ISO8601 format, i.e. YYYY-MM-DD.

5.3.5 File Type Suffix

252. All the processed files are comma separated variables (CSV) and have a .csv suffix with the exception of the trajectories files which are in JavaScript Object Notation(JSON) and have a .json suffix.

253. The files are compressed using bzip2 compression before being put to bucket storage. Therefore all processed files have a .bz2 suffix after the .csv or .json suffix.

Section 6

6 Cloud Platform Costs

254. The Google Cloud Platform was chosen to run the Trajectory Module and Trajectory Assessment in a Kubernetes cluster. The following is a breakdown of the current costs of running the system on the Google Cloud Platform.

6.1 Cost Breakdown

255. Like other cloud platform providers, Google has different prices depending upon the location of compute engines and cloud storage, see [13].

256. We have assumed that the compute engines and cloud storage shall be located in Europe, specifically in Belgium wherever possible.

6.1.1 Compute Nodes

257. The compute nodes are the largest expense of the Google Cloud Platform.

258. The JupyterHub Notebook server requires compute nodes with 26GB of memory to refine and merge the source data. Specifically, over 20GB of memory is required to run the overnight data matching and merging processes, all other processes can run on nodes with 13GB of memory which are significantly cheaper.

259. The compute node prices shown in Table 14 below are per month. Note: that Google has sustained usage discounts which make the monthly costs shown in the table up to 30% cheaper than per minute billing.

Compute Engine	CPUs	Memory GB	Monthly	1 -Year	3-Years	Pre-emptable
n1-standard-2	2	7.5	$53.45	$48.06	$34.33	$16.06
n1-highmem-2	2	13	$66.53	$59.85	$42.75	$20.08
n1-highmem-4	4	26	$133.06	$199.70	$85.49	$40.16
Custom	2	26	$139.44	N/A	N/A	N/A

Table 14: Google Compute Engine Pricing Belgium May 2018

260. Note: Google suggest that creating a custom machine type can reduce costs. However, memory over 6.5 GB per CPU is charged as extended memory, which costs over twice as much as standard memory, making a custom machine with only 2 CPUs more expensive than a n1-highmem-4 machine, see the last row in Table 14 above.

261. A Kubernetes cluster normally runs with a minimum of 3 compute nodes. Therefore, the minimum compute node cost on the current platform is currently $399.18 per month. This could be reduced by making a 1 year or 3 year commitment, see columns 1-Year and 3-Years in Table 14.

262. The compute node cost scales linearly with the number of nodes, e.g. 10 n1-highmem-4 compute nodes will cost $1330.60 per month.

263. Note, the compute node costs could be nearly halved by running the overnight data matching and merging processes as Kubernetes jobs on n1-highmem-4 compute nodes enabling the Kubernetes pods to run on the cheaper n1-highmem-2 compute nodes.

6.1.2 Compute Storage

264. Persistent disk storage is required by:
- Databases,
- Compute Nodes
- and Users

265. The databases are currently configured with 50GB disks, the compute nodes are configured by default with 100GB disks and each user is currently configured to use 50GB disks.

266. Standard storage costs $0.04 per GB/month and SSD storage costs $0.17 per GB/month. We have chosen standard storage since there is little benefit to be gained using SSD storage. Therefore the storage costs are:

$$\$4 + \$4 * nodes + \$2 * users \qquad \text{Eq 1}$$

267. For example, a small system with 3 nodes and 4 users storage would cost $24 per month, while a larger system with 10 nodes and 20 users storage would cost $84 per month.

6.1.3 Cloud Storage

268. The trajectories and their associated data are stored in Google Buckets in the europe-west1 region.

269. Regional storage costs $0.02 per GB/month and we currently create approximately 2GB/day of data, so each days data costs approximately $0.04 per month. A years data in regional storage would cost approximately $14.60 per month.

6.1.4 Other Costs

270. The Kubernetes cluster requires a load balancer. Load balancers are charged by forwarding rules: $0.025 per hour per forwarding rule for up to 5 rules, i.e. $18 per month for a single rule.

271. Network ingress is free as is network egress to the same zone or to a GCP service in the same region. Otherwise egress between zones in the same region is $0.01 per GB and egress between regions is charged at $0.12 per GB.

272. Network egress costs are unlikely to be significant unless the project has many users outside of Europe. For example, our total network egress costs last month were under £0.10.

273. There are also costs for using the Container Registry, which is charged as multi-regional cloud storage ($0.026 per GB/month, £0.10 last month) and a Compute Engine Static IP charge (also £0.10 last month).

Section 7

7 Potential Improvements

274. While developing the Trajectories Production project our understanding of:
- trajectory data,
- the processing system
- and how the system will be used

275. has evolved.

276. The following sections are our main ideas for how the Trajectories Production system could be improved to:
- better match and merge CPR and ADS-B input data,
- calculate more accurate trajectories,
- reduce the costs of running the system,
- and process trajectories faster.

7.1 Algorithms

7.1.1 CPR – FR24 Flight Matching

277. The CPR-FR24 flight matching verification function (compare_trajectory_positions in trajectory_functions.py) rejects a large proportion of CPR-FR24 matches with identical aircraft addresses: currently it rejects between 10% and 20% of trajectories with the same aircraft address.

278. The current verification function tests the horizontal and vertical proximity of positions at the same time. Given our improved understanding of trajectory position error (i.e. along-track/time errors are considerably larger than across track errors), it would be worthwhile to investigate why CPR and FR24 positions (with the same aircraft address!) are so far from each other at the same time.

279. This knowledge could then be used to tune the horizontal and vertical proximity thresholds or to develop a more sophisticated matching verification function.

7.1.2 Derive Horizontal Path

280. The derive horizontal path algorithm seems to work well when aircraft are airborne, but issues have been observed with sharp turns when aircraft are manoeuvring on the ground. A new function (find_extreme_point_along_track_index) has been added to horizontal_path_functions.py to address this issue. However, it is currently disabled because using it to fix the ground issues may break the airborne paths...

281. This warrants further investigation.

7.1.3 Time/Speed Smoothing

282. The time smoothing algorithm has recently been changed from a curve fitting algorithm to one that calculates ground speeds between positions and then smooths the calculated speeds (with 3 different filters!) before using the smoothed speeds to back-calculate the position times. Currently either smoothing method may be selected with speed filtering as the default method.

283. The time/speed smoothing algorithm warrants further investigation. If the speed smoothing algorithm is preferred then the moving median and moving average filter parameters should be tuned and the time filtering method should be deleted.

7.2 Memory Use

284. The python software does not use memory efficiently, usually requiring around four times more memory to process a data file than the size of the data file itself.

285. The software is particularly memory intensive during matching and merging processes. The overnight matching and merging processes require the most memory and are responsible for the system requiring compute nodes with 26GB of RAM.

286. The overnight matching and merging processes could be redesigned to reduce memory consumption, enabling the system to run on (cheaper) compute nodes with less memory.

7.3 Processing Times

287. In addition to using memory inefficiently, some of the python applications take a long time to process each days data. The slowest applications are:
- clean_position_data.py
- analyse_position_data.py
- interpolate_trajectories.py
- find_airport_intersections.py
- find_sector_intersections.py
- and find_user_airspace_intersections.py.

288. Note that clean_position_data.py and analyse_position_data.py are bottlenecks during the early stages of processing, while: interpolate_trajectories.py and the find intersection applications all run after analyse_position_data.py and so they could all be run in parallel.

7.3.1 clean_position_data.py

289. clean_position_data.py is the fastest of the slow applications, normally taking under an hour to process a days positions. However, it is called four times for each days CPR and ADS-B input data, taking over 2.5 hours of the approximately 4 hours required to process each days input data.

290. Therefore just speeding up clean_position_data.py could halve the time required to input each days CPR and ADS-B input data from 4 hours per day to 2 hours per day.

291. There are two obvious ways that clean_position_data.py could be speeded up:

- remove the conversion from Lat-Long to ECEFPoints to calculate distances between positions and use a (numpy friendly) haversine function instead
- and/or rewrite it C++.

7.3.2 analyse_position_data.py

292. analyse_position_data.py takes approximately a day to process each days positions. It performs many calculations in ECEF coordinates and has many loops which are particularly slow in python.

293. The simplest and most efficient way to speed up analyse_position_data.py would be to rewrite it in C++.

7.4 System Architecture

294. It would be useful if the time required to process large batches of data could be reduced.

295. If the python code was built into a Docker container with just a python interpreter and the required libraries, (i.e. without JupyterHub) then many of the python applications could easily be run in parallel on temporary compute nodes using Kubernetes jobs or a similar mechanism.

296. Building the python code into an appropriate Docker container would be a relatively simple task. Redesigning the system to run in parallel on temporary compute nodes would be more complicated. However, it could reduce the “wall clock” time required to process many days worth of data considerably.

Appendix

A

A Trajectory Cleaning

297. Trajectory cleaning removes gross trajectory position errors such as:
- duplicate positions,
- significant positional offsets,
- spurious flight segments
- and non-synchronised timestamps.

A.1 Error Types

A.1.1 Duplicate Positions

298. Duplicate positions are redundant and are likely to distort trajectory analysis by giving duplicate positions excess “weight” in the Trajectory Smoothing algorithms.

299. Duplicate input positions shall be removed by the data conversion functions, see [6] Data Merging Report. However, merging may introduce new duplicate positions.

300. Where position data originates from an aircraft, different data sources may contain duplicate copies of the data. For example, ADS-B positions from different providers such as Flight Radar 24 and the Open Sky Network can originate from the same ADS-B transmissions. Many aircraft use a single piece of equipment to transmit both Mode-S squawks and ADS-B messages. Therefore, some Mode-S radar positions and ADS-B positions may contain identical data. These duplicate positions are removed.

301. Note: positions from conventional SSR radars are calculated by the radar, they not transmitted by the aircraft. Therefore, identical positions from different conventional radars (i.e. radar positions without an aircraft address) are valid.

A.1.2 Erroneous Positions

302. Position errors may take the form of individual erroneous positions or groups of erroneous positions which can occur where trajectories have been created by (incorrectly) merging trajectories from different flights.

303. For example, Figure 27 shows a CPR trajectory with a single horizontal position error over England.

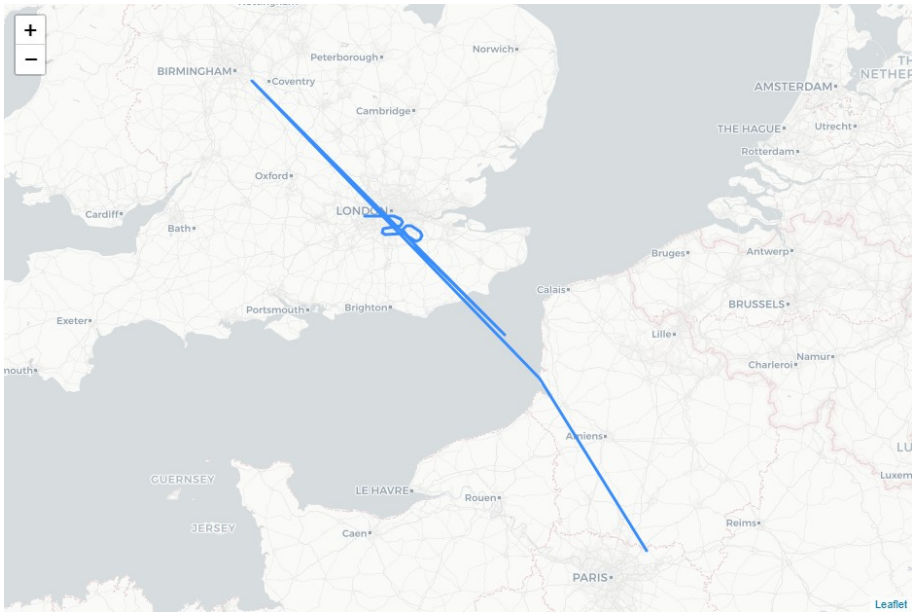

Figure 27: Trajectory with a Horizontal Position Error

304. While Figure 28 shows a CPR trajectory with multiple horizontal position errors over Germany.

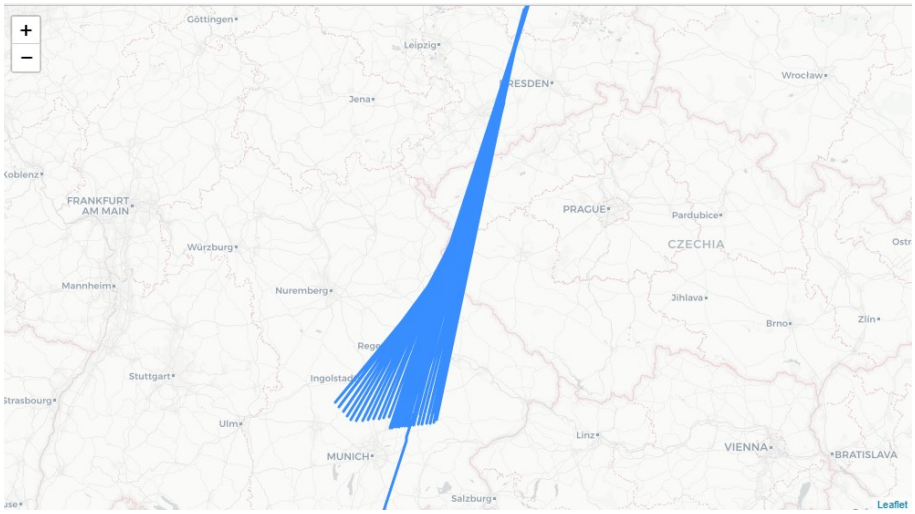

Figure 28: Trajectory with multiple Horizontal Position Errors

305. However, it is not only horizontal positions that can be erroneous.

306. For example, Figure 29 shows trajectory with an erroneous position past the runway at Copenhagen airport.

Figure 29: Trajectory with Horizontal and Vertical Errors

307. The erroneous position is more easily identified in the altitude -time profile of Figure 30: it is the position at 1000 feet after the aircraft has landed.

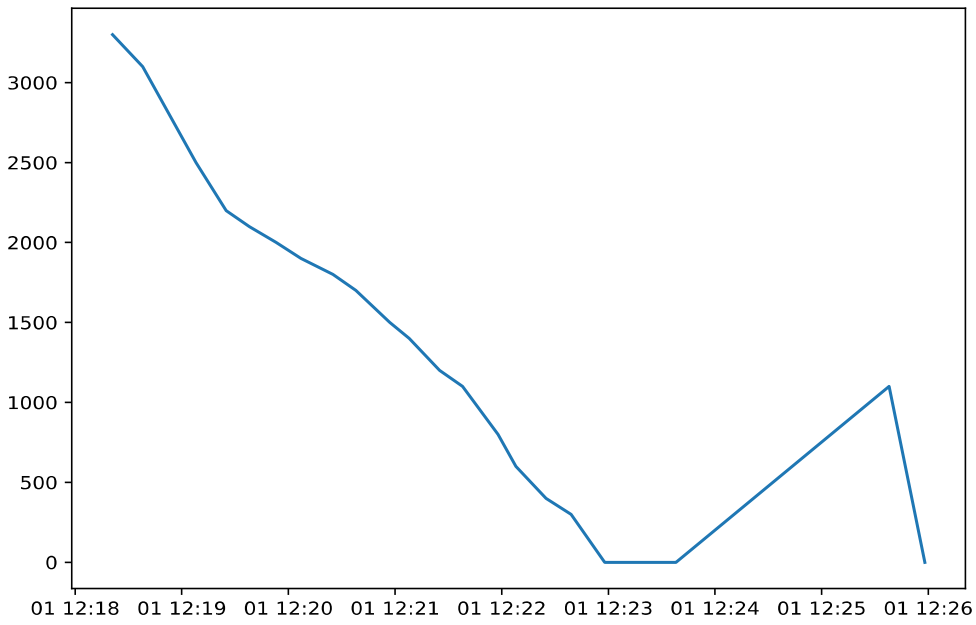

Figure 30: Altitude-Time Profile with a Vertical Error

A.2 Error Identification

A.2.1 Duplicate Positions

308. Duplicate positions are found by the pandas DataFrame duplicated function. However, only positions *with* aircraft addresses are marked as duplicates so that duplicate conventional radar positions are *not* removed.

A.2.2 Multiple Aircraft Addresses

309. Trajectory positions with different aircraft addresses are clearly erroneous. The most common aircraft address is stored in the flight data, therefore, positions with a different aircraft address are marked as erroneous.

310. Note: this should be a CPR only issue, since ADS-B trajectories are grouped together by aircraft address.

A.2.3 Horizontal Positions

311. Erroneous horizontal positions are identified by unusually high ground speeds.

312. The calculation of ground speed is very sensitive to the accuracy and precision of position timestamps. In particular, speed cannot be calculated between positions at the same timestamp (since it involves dividing by zero) and speeds calculated between positions at close timestamps are very inaccurate.

313. For example, Figure 31 shows three equidistant positions along a trajectory: **a**, **b** and **c** with their timestamps (to the nearest millisecond) rounded to the nearest second.

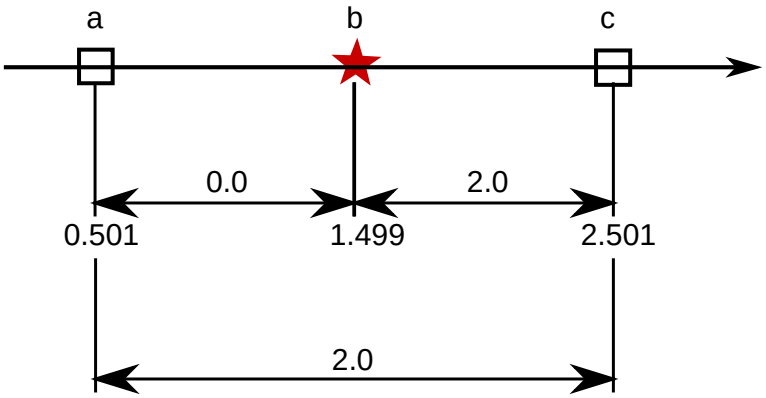

Figure 31: Time Rounding Error

314. In Figure 31, the rounded time difference between **b** and **c** is the same as the actual time difference between **a** and **c,** while the rounded time difference between **a** and **b** is zero when it should be just under a second.

315. Also, the low precision of raw trajectory times (one second for both CPR and ADS-B data) causes very inaccurate speeds between close positions. At its worst, the low time precision can double the calculated speeds of positions at adjacent timestamps.

316. One solution to the low time precision issue is to consider time precision and distance accuracy in the speed calculation, i.e.:

$$s = \frac{d - d_{accuracy}}{t_{delta} + t_{precision}} \quad \text{Eq 2}$$

317. Eq 2 underestimates speed, so high speeds are more likely to indicate erroneous positions and not close positions. Eq 2 also enables speed to be calculated between positions at the same time.

A.2.4 Vertical Positions

318. Erroneous vertical positions are identified by change in attitude (climbing, level or descending) accompanied by an SSR code change.

319. Like ground speeds, vertical speeds are also affected by the low precision of raw trajectory times. However, they are also affected by the low precision of raw trajectory altitudes: CPR altitudes have a precision of 100 feet, ADS-B altitudes have a precision of 25 feet. The low precision of trajectory altitudes reduces the effectiveness of using Eq 2 to calculate vertical speeds to detect invalid vertical positions.

320. Therefore, the attitude of a position relative to the previous position is determined and compared to the previous attitude. A change in attitude from climbing to descending (or vice versa) is suspicious, while a change in attitude to (or from) level is usually normal. However, an attitude change accompanied by a simultaneous SSR code change is likely to be an erroneous position.

321. Note: a change in attitude from climbing to descending (or vice versa) is often due to position time errors.

A.2.5 Quality Metrics

322. The numbers of duplicate and erroneous positions are recorded in the error metrics for each trajectory.

323. Erroneous positions are classified as:
- DUPLICATES – for duplicate positions (merged data only)
- ADDRESSES – for invalid aircraft addresses (CPR data only)
- DISTANCE – for invalid horizontal positions
- ALTITUDE – for invalid vertical position

FLIGHT_ID	TOTAL	DUPLICATES	ADDRESSES	DISTANCE	ALTITUDE
205920	1	0	0	1	0
205931	1	0	0	0	1
206919	1	0	0	0	1
232696	2	0	0	2	0
238496	2	2	0	0	0
240753	4	1	0	3	0

Table 15: Data cleaning example csv output

324. The error counts for erroneous positions are output in a csv file, see Table 15.

A.3 Cleaning Examples

325. Figure 32 shows the trajectory of Figure 27 after cleaning with the erroneous position removed.

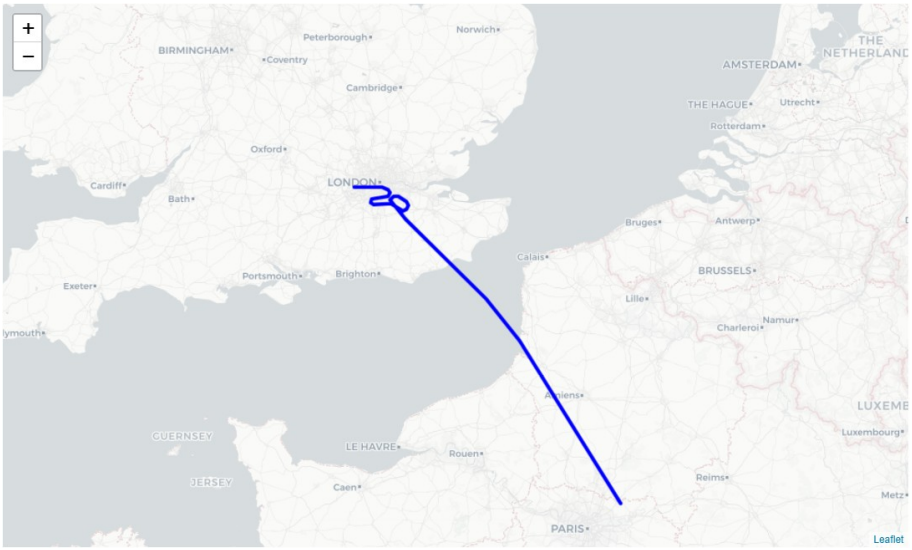

Figure 32: Trajectory with Horizontal Position Error cleaned

326. Figure 33 shows the trajectory of Figure 28 after cleaning with the erroneous positions removed.

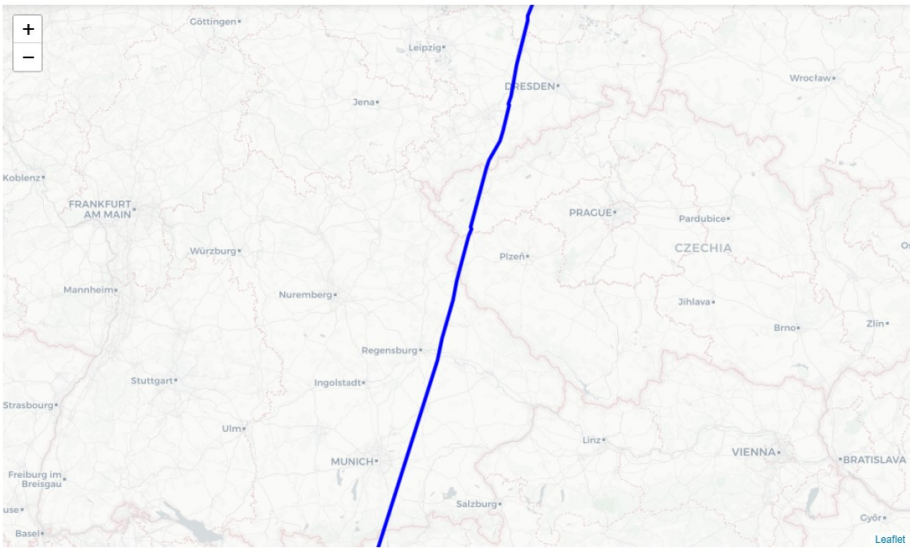

Figure 33: Trajectory with multiple Horizontal Position Errors cleaned

Figure 34: Trajectory with Horizontal and Vertical Errors cleaned

327. Figure 34 shows the trajectory of Figure 29 after cleaning with the erroneous position past the end of the runway removed. While Figure 35 shows the altitude-time profile of Figure 30 after cleaning, with the position at 1000 feet removed.

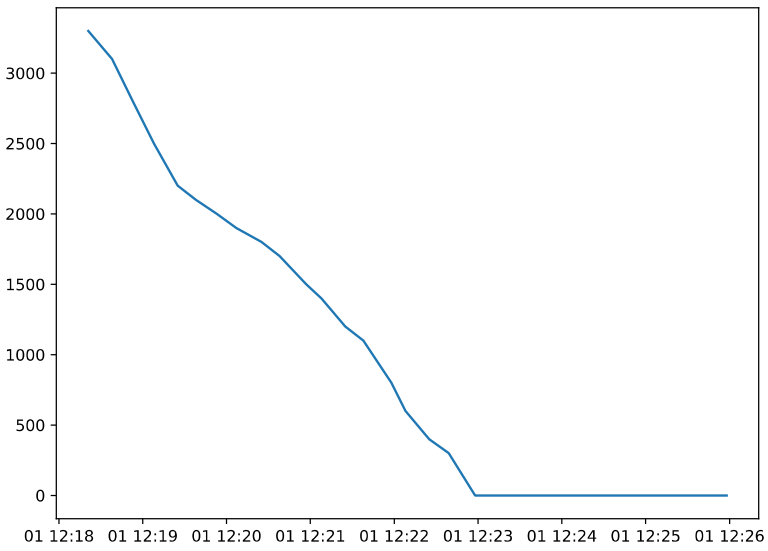

Figure 35: Altitude Time Profile with a Vertical Error cleaned

A.4 Trajectory Error Metrics

A.4.1 CPR Position Errors

328. Table 16 shows the percentages of CPR trajectories with different types of position errors for each day of the first week of August 2017.

Date	In Error	Duplicates	Addresses	Distances	Altitudes
2017-08-01	22.43%	3.32%	0.11%	14.46%	7.43%
2017-08-02	22.63%	3.50%	0.12%	14.22%	7.57%
2017-08-03	22.49%	3.37%	0.10%	14.89%	6.93%
2017-08-04	22.19%	3.32%	0.09%	14.85%	6.74%
2017-08-05	24.02%	3.74%	0.16%	15.91%	7.60%
2017-08-06	23.53%	3.35%	0.12%	15.42%	7.75%
2017-08-07	23.68%	4.28%	0.16%	14.70%	7.59%
Average	22.98%	3.55%	0.12%	14.91%	7.36%

Table 16: CPR Error Percentages

329. Almost a quarter (23%) of CPR trajectories have at least one erroneous position, with distance (i.e. horizontal speed) being the most common error, closely followed by altitudes and duplicates.

330. A histogram of the number of invalid CPR positions for 2017-08-01 shows that the majority of erroneous trajectories only had a few position errors, see Figure 36.

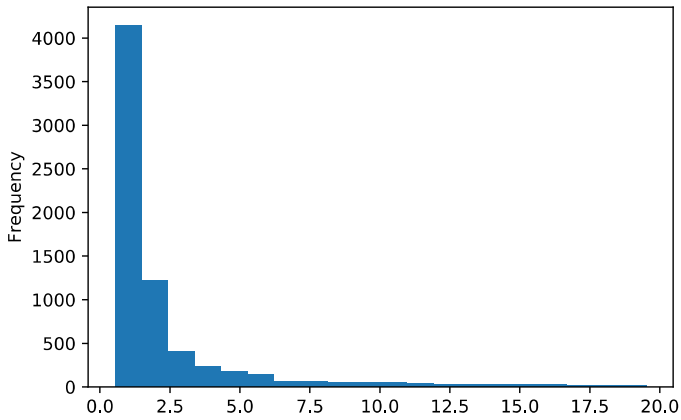

Figure 36: Histogram of Total CPR Position Errors

331. The Potential Algorithms report [5], section 2.5.2 contains more details about CPR position errors.

A.4.2 FR24 Position Errors

332. Table 17 shows the percentages of FR24 trajectories with types of position errors for the first week of August 2017.

Date	In Error	Distances	Altitudes
2017-08-01	46.56%	23.41%	31.41%
2017-08-02	45.48%	22.06%	31.08%
2017-08-03	44.26%	20.35%	30.92%
2017-08-04	43.27%	19.24%	31.13%
2017-08-05	46.05%	21.94%	32.01%
2017-08-06	44.69%	19.91%	31.62%
2017-08-07	43.74%	20.49%	30.64%
Average	44.85%	21.04%	31.26%

Table 17: FR24 Error Percentages

333. Nearly half (45%) of FR24 trajectories have at least one erroneous position, with altitude being the most common error at 31% followed by distance (i.e. horizontal speed) at 21%. There were no duplicate position or aircraft address errors.

334. There are nearly double the proportion of erroneous FR24 trajectories as erroneous CPR trajectories. Most of the increase is due the higher proportion of FR24 trajectories with erroneous altitudes, over four times the proportion of CPR trajectories with erroneous altitudes.

335. Histograms of the numbers of invalid positions for 2017-08-01 show that the majority of erroneous trajectories only had a few position errors, see Figure 37.

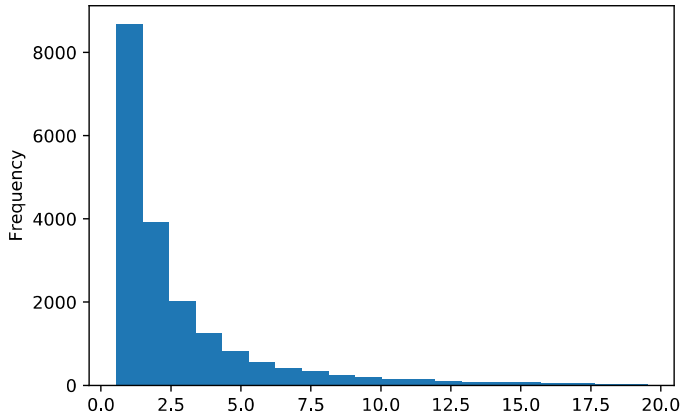

Figure 37: Histogram of FR24 Total Position Errors

336. The Potential Algorithms report [5], section 2.5.3 contains more details about FR24 position errors.

Appendix B

B Trajectory Smoothing

337. The trajectory smoothing algorithm smooths trajectories by considering trajectory positions in: horizontal, temporal and vertical dimensions.

338. The smoothing algorithm first determines the horizontal path flown by an aircraft before calculating smoothed time and vertical profiles.

B.1 Horizontal Path

B.1.1 Find Turning Points

339. Turning points are found as follows:

1. Create a baseline from the first point to the furthest point.
2. Calculate the widest point from the baseline, if it is further than a given tolerance then divide the baseline in two at the widest point.
3. Repeat 2 above until all the widest points are within a given tolerance.

340. For example, Figure 38 shows positions for a trajectory plotted as along track and across track distances relative to a baseline.

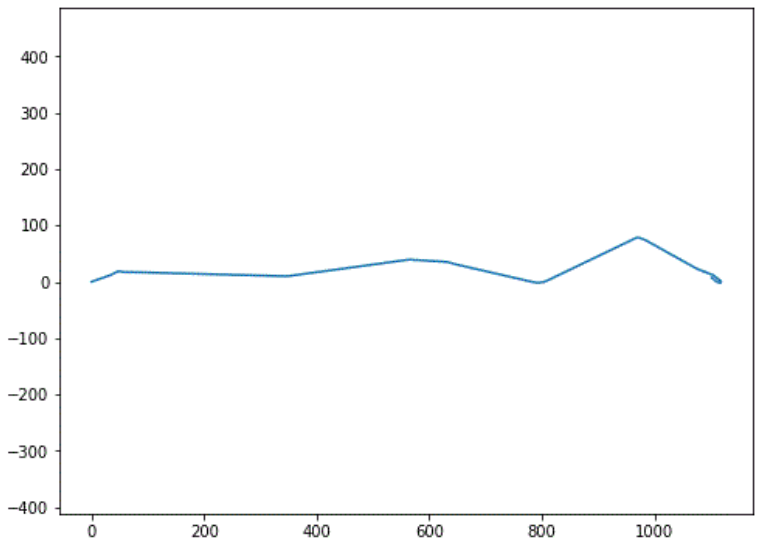

Figure 38: Trajectory Positions Relative to Baseline

341. Note: the baseline is the Great Circle Arc between the first trajectory point and the furthest point from it. Along track and across track distances are calculated using spherical Earth Centred, Earth Fixed (ECEF) coordinates of trajectory positions.

342. Figure 39 (a) shows the same trajectory with across track distances exaggerated.

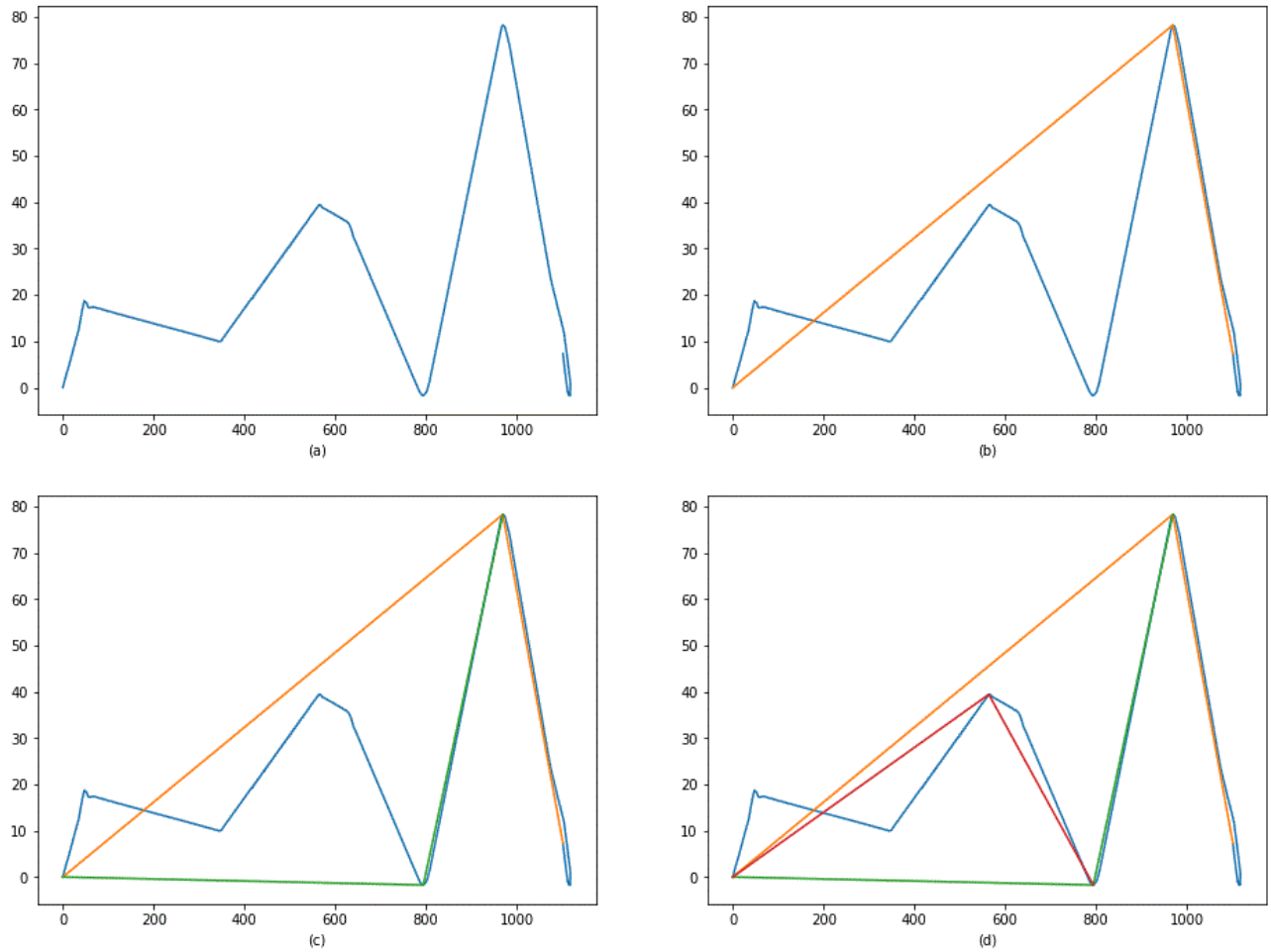

Figure 39: Recursively Calculating Widest Positions

343. Figure 39 (b) shows lines (in orange) drawn from the first and last points of the baseline to the widest point from the baseline. Figure 39 (c) shows lines (in green) drawn from the first orange line to the widest point from the first orange line and Figure 39 (d) shows lines (in red) drawn from the first green line to the widest point from the first green line.

344. The widest points indicate turns, either at the ends of straight legs or within holds or circuits. Figure 40 shows lines drawn between the turning points found for the trajectory of Figure 38.

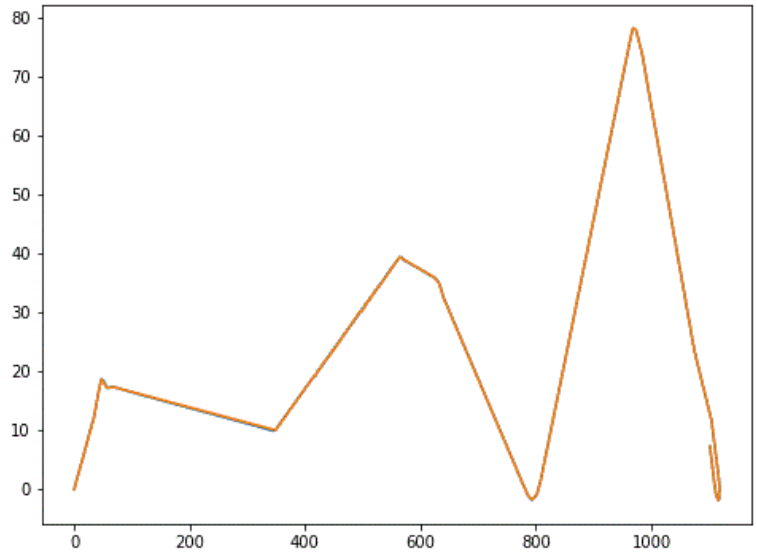

Figure 40: Lines Drawn Between Widest Points

B.1.2 Derive Path Legs

345. Where widest points are far apart, they are likely to be within turns between straight legs, not along the straight legs themselves, see points a, b and c in Figure 41.

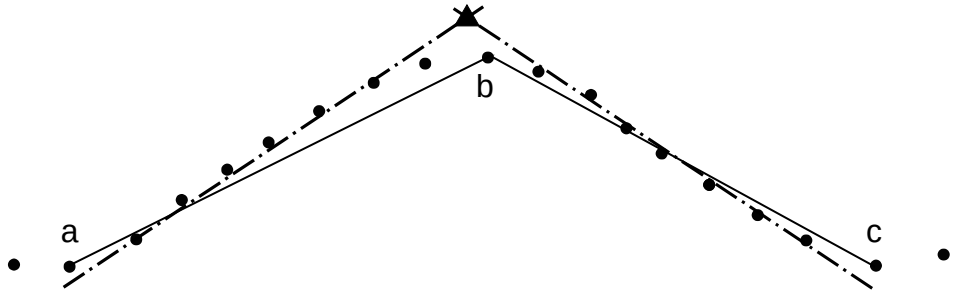

Figure 41: Widest Point Positions

346. Also, where there are sufficient points between widest points, the straight legs are the lines of best fit of the points between the widest points.

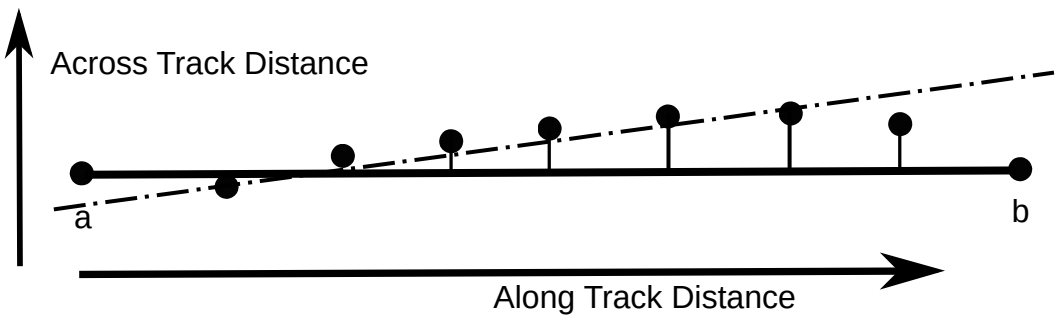

Figure 42: Path Leg Line Fitting

347. A line of best fit is calculated from the along track and across track distances of the points between the widest points, see Figure 42.

348. Path waypoints are calculated at the intersections of pairs of lines of best fit. The path legs lie between the path waypoints.

B.1.3 Calculate Turn Initiation Distances

349. A turn between straight legs can be defined by a turn radius (r) or a turn initiation distance (d), see [23] ICAO Doc 9905 AN/471 Required Navigation Performance Manual 3.2.10 and Figure 43, where:

$$d = r \tan \alpha / 2 \qquad \text{Eq 3}$$

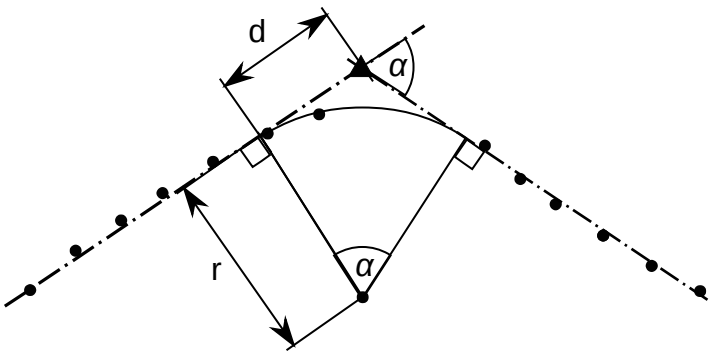

Figure 43: Turn Initiation Distance

350. Finding a circle that is tangent to both the inbound and outbound legs while passing through a turn point is a Problem of Apollonius [24], specifically: one point and two lines (PLL).

351. The turn radius is calculated from the distance (*d*) of the closest point to the intersection and its angle (θ) from the bisector of the turn legs using the cosine rule and solving the quadratic, see Figure 44 and Eq 4, Eq 5, Eq 6 and Eq 7.

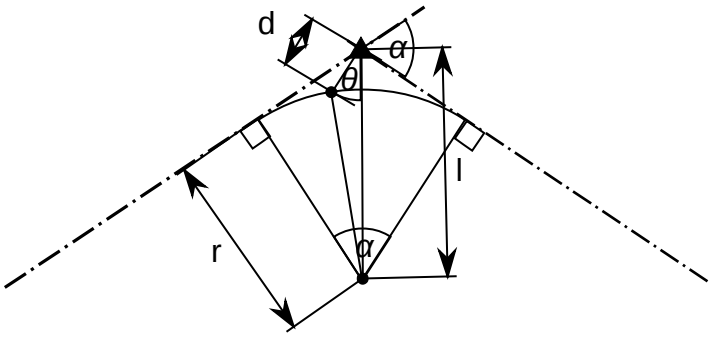

Figure 44: Turn Radius

352. Where:

$$l = \frac{r}{\cos \alpha/2} \quad \text{and} \quad \sin^2 \alpha/2 = 1 - \cos^2 \alpha/2 \qquad \text{Eq 4}$$

$$r^2 = d^2 + l^2 - 2\,dl\cos\theta = d^2 + \frac{r^2}{\cos^2 \alpha/2} - \frac{2\,dr\cos\theta}{\cos\alpha/2} \qquad \text{Eq 5}$$

353. Eq 5 is a quadratic in r^2:

$$\left(1 - \cos^2 \alpha/2\right) r^2 - 2\,d\cos\theta\cos\alpha/2\,r + d^2 = 0 \qquad \text{Eq 6}$$

$$r = d\cos\alpha/2\,\frac{\cos\theta \pm \sqrt{\cos^2\theta - \sin^2\alpha/2}}{\sin^2\alpha/2} \qquad \text{Eq 7}$$

354. Eq 7 may give two positive solutions for r, the larger value is the turn radius.

355. The turn initiation distance is calculated from the turn radius using Eq 3.

356. Note: Turns are only modelled within a range of turn angles. The minimum turn angle is one degree and the maximum turn angle is 150 degrees. Outside of this range, the turn is not modelled, i.e. the turn initiation distance is zero and the inbound and outbound legs go straight to the intersection point.

B.1.4 Calculate Path Distance

357. The distance between derived waypoints is accumulated to calculate the derived path distance.

358. The path distance is the distance flown by an aircraft *around* derived waypoints, i.e. it is the distance between waypoints minus the turn initiation distances plus the turn arc lengths at each turn, see Figure 43 and Figure 45. Where the turn arc length is:

$$d_{turn} = r\,\alpha \qquad \text{Eq 8}$$

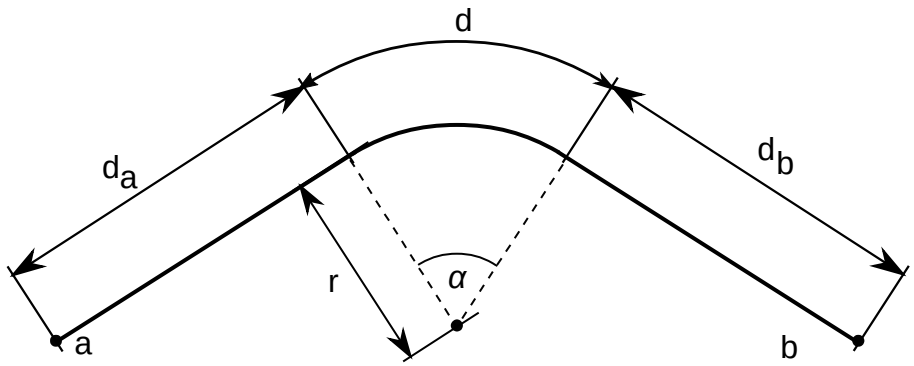

Figure 45: Path Distance

359. The path distance of an input position is the distance along the closest adjacent derived path straight leg or turn.

360. The closest adjacent derived path straight leg or turn is found for an individual point by calculating the closest distances of the point to each of the derived path legs and then choosing closest derived path leg. This is a “brute force” approach which is both computationally inefficient and may be incorrect for complex paths such as holds, which have duplicate derived path legs. Therefore, the closest derived path legs for sequences of points are found by calculating the closest distances of each point to the closest leg of the previous point (the current leg) and the preceding and subsequent derived path legs, see Figure 46.

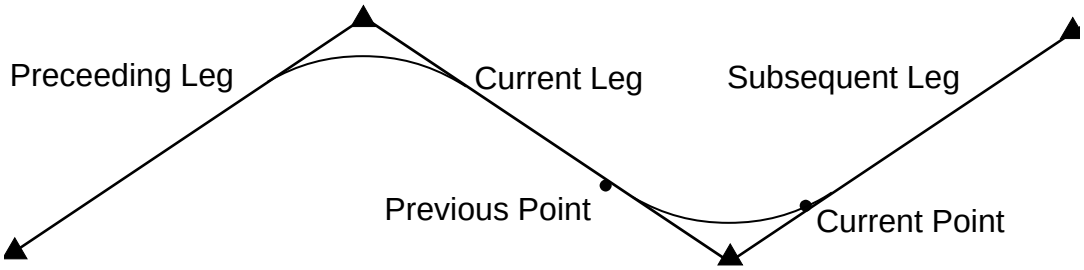

Figure 46: Find the Closest Leg from the Current Leg

361. The “brute force” approach is only used if the current, preceding and subsequent derived path legs are not within a given distance tolerance of the current point.

B.2 Calculate Time Profile

362. A plot of time versus trajectory distance should form a smooth curve. However, position times have a relatively low precision of one second and significant errors have been observed in ADS-B times, see [25] Common Errors on Flightradar 24 and when ordered by time, positions are often “out of order”.

363. Therefore, input positions are sorted in path distance (followed by time) order to derive the time and altitude profiles.

364. The inaccuracy of position times can be observed in plots of ground speed calculated from position path distances and times, e.g. see Figure 47.

365. The ground speed instability observed in Figure 47 from point 73 onwards is due to calculating distances and times between interleaved radar reports from different ANSPs in the CPR data. Any differences between source times and/or positions show up as large changes of calculated ground speed. The ground speeds calculated prior to point 73 are mostly from the same ANSP and are therefore much more stable.

366. It is worth noting that the time profile was originally smoothed using a curve fitting algorithm to a 5 degree polynomial function. However, the time standard deviation and maximum errors were significantly smaller using smoothed ground speeds than using the curve fitting algorithm.

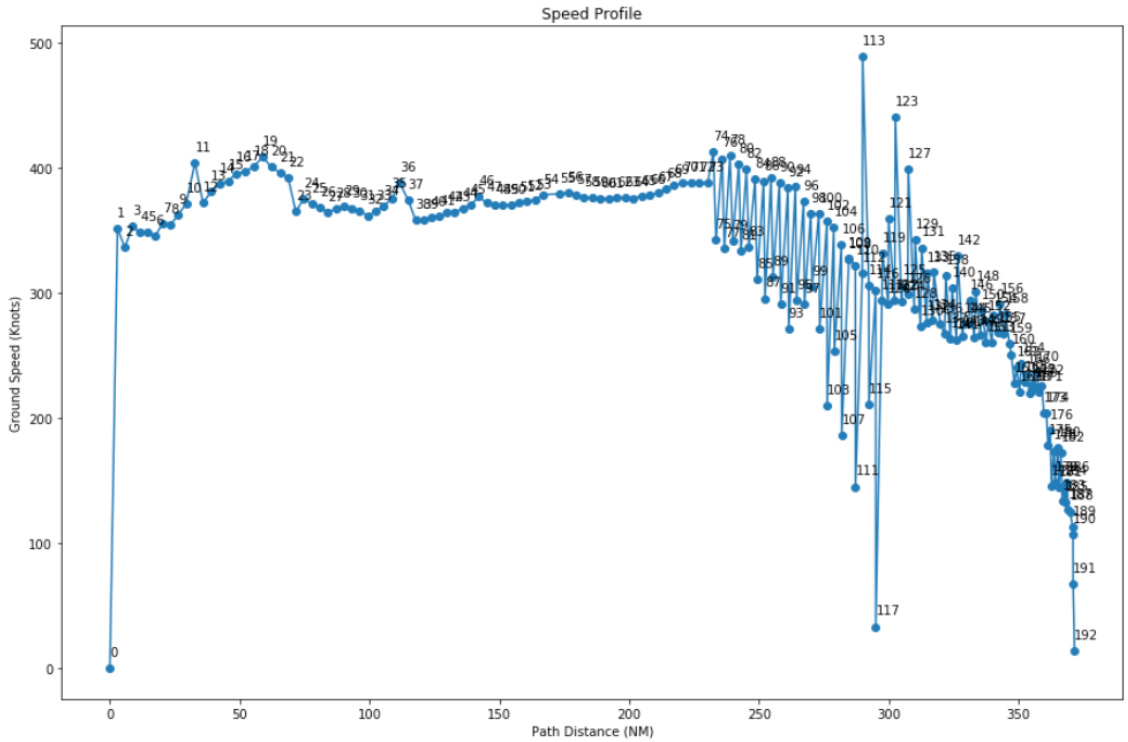

Figure 47: Example CPR Trajectory Ground Speeds

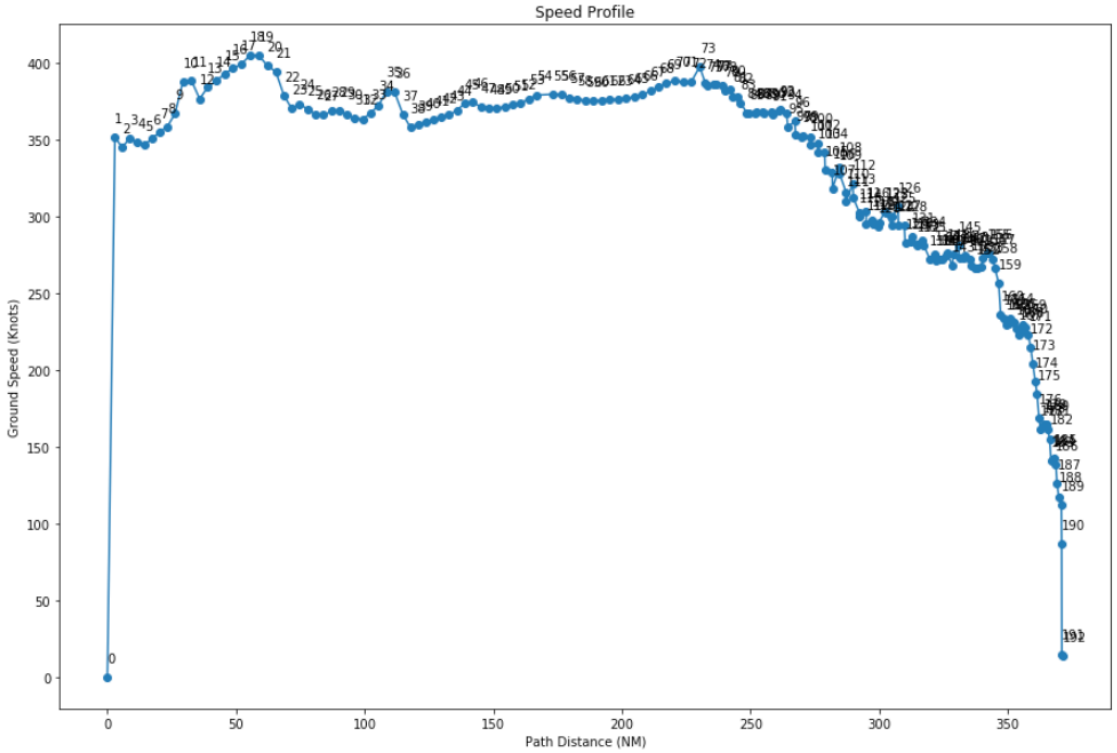

Figure 48: Smoothed Trajectory Ground Speeds

367. Trajectory ground speeds are calculated using a moving average filter that groups together positions from common surveillance sources to reduce the instability shown in Figure 47.

368. The ground speeds are further smoothed by passing through moving median[28] and moving average [29] filters. The filters are independently configurable via their samples sizes, with a sample size of one or less disabling the filter.

369. Smoothed trajectory times are then calculated from the smoothed ground speeds and distances along the horizontal path.

B.3 Calculate Altitude Profile

370. A plot of altitude versus trajectory distance (a vertical profile) should also form a smooth curve during climbs and descents. However, between climbs and descents, aircraft fly at single (cleared) altitude, which can be just be represented by the altitude and start and finish points.

371. Note: the altitude reported by an aircraft is always relative to 1013.25 hPa and therefore (strictly speaking) is a flight level.

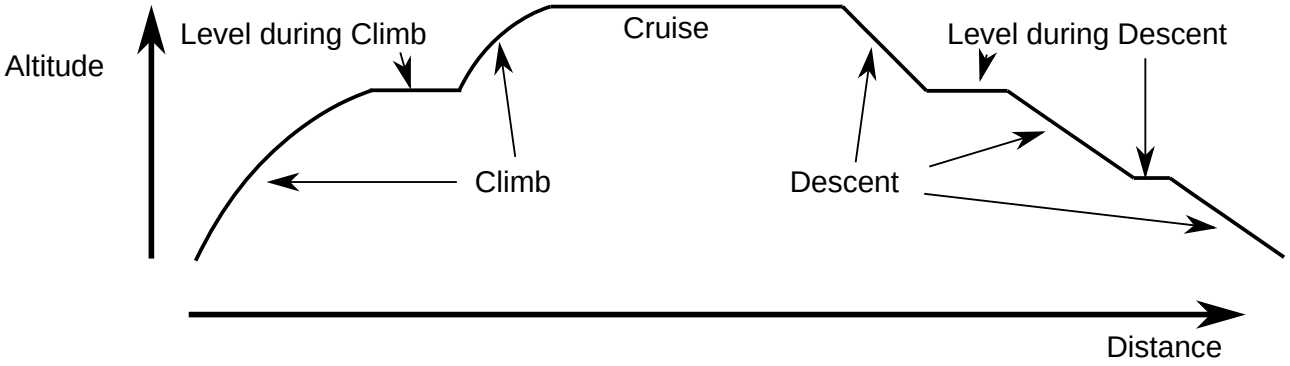

Figure 49: A Vertical Profile

372. Ideally all aircraft would fly a continuous climb to their cruising flight level and a continuous descent from their cruising flight level to their destination. However, ATC often require aircraft to perform “stepped” climbs and descents, where aircraft are required to fly at one or more cleared levels before being cleared to fly at their cruising level or descending to their destination, see Figure 49.

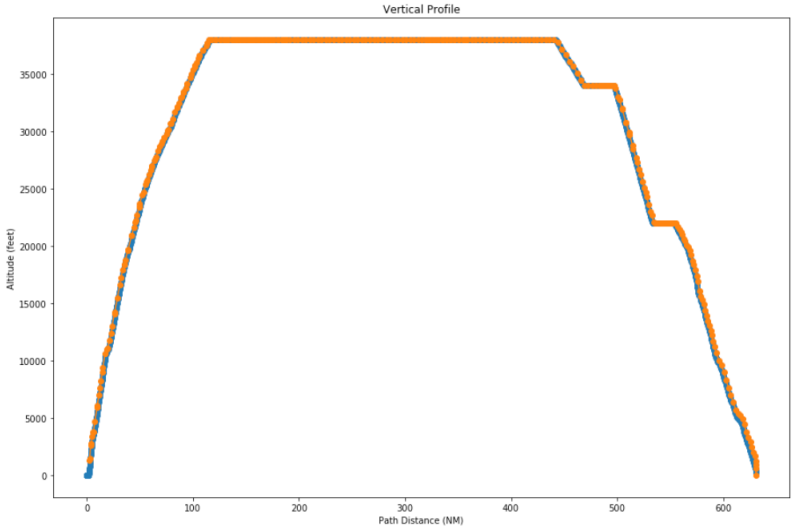

Figure 50: A Real Vertical Profile

373. Figure 50 shows a real vertical profile composed of data logger (blue) and CPR (orange) points.

374. Since the altitude of an aircraft should not change while it is cruising at a given flight level, it’s smoothed altitude is just the altitude corresponding to the cruising flight level.

375. The altitude smoothing algorithm finds sections of each altitude profile where an aircraft was cruising within the range (+/- 200 feet) of a flight level. The cruising sections are represented by the first and last points at the cruising altitude and the altitude standard deviation and maximum difference are calculated from altitude differences within the cruising sections.

