SATERA D6.2 Space-based MLAT algorithms prototypes

Deliverable ID:	**D6.2**
Project acronym:	**SATERA**
Grant:	**101164313**
Call:	**HORIZON-SESAR-2023-DES-ER-02**
Topic:	**HORIZON-SESAR-2023-DES-ER-02-WA2-1**
Consortium coordinator:	**UPV**
Edition date:	**10 November 2025**
Edition:	**01.00**
Status:	**Official**
Classification:	**PU**

Abstract

This document describes the prototype reference software design and implementations of the positioning, tracking, integrity algorithms developed in T6.1, T6.2, and T6.3 respectively, ready for integration into the end-to-end simulation tool in WP7. This deliverable will also describe the details of the verification conducted on the prototypes before they are integrated in the simulation tool, as well as any final adjustment to the algorithms described in D6.1 needed as a result of the verification process.

EUROPEAN PARTNERSHIP

Co-funded by the European Union

Authoring & approval

Author(s) of the document

Organisation name	Date
UNITOV	21/08/2025
UPV	15/10/2025
COLLINS	21/08/2025

Reviewed by

Organisation name	Date
UPV	13/11/2025
UNITOV	10/11/2025
COLLINS	13/11/2025
UNIV COIMBRA	
INPE	

Approved for submission to the SESAR 3 JU by

Organisation name	Date
UPV	13/11/2025
UNITOV	13/11/2025
COLLINS	19/11/2025
UNIV COIMBRA	19/11/2025
ENAIRE	19/11/2025
INPE	16/11/2025

Rejected by[1]

Organisation name	Date

Document history

Edition	Date	Status	Company Author	Justification
00.01	21/08/2025	Draft	UNITOV	First Draft

[1] Representatives of the beneficiaries involved in the project.

00.02	09/10/2025	Draft	UNITOV	Second Draft
00.10	29/10/2025	Draft	UNITOV	Third Draft
01.00	10/11/2025	Official	UNITOV	Final

SATERA

SPACE-BASED COMPOSITE ADS-B AND MULTILATERATION SYSTEM VALIDATION THROUGH SCALABLE SIMULATIONS

SATERA

This document is part of a project that has received funding from the SESAR 3 Joint Undertaking under grant agreement No 101164313 under European Union's Horizon Europe research and innovation programme.

Table of contents

List of figures

List of tables

1 Executive Summary

This document presents the localisation algorithms, the tracking filters and the input/output specifications used in SATERA to perform space-based multilateration (MLAT). Three kinds of measurements are used: Time of Arrival (TOA), Frequency of Arrival (FOA) and Angle of Arrival (AOA). This takes the name of Enhanced MLAT (E-MLAT).

The aim of this document is to describe in detail all the algorithms for the initial position estimation of the aircraft and the subsequent tracking over its trajectory, including the management of multiple targets at once. Since the E-MLAT is meant to be a backup for when GNSS in unavailable, an integrity indicator is also considered, so that the Automatic Dependent Surveillance – Broadcast (ADS-B) data can be validated against the E-MLAT based estimated position.

All the processing is carried out in the Central Processing Station (CPS), while the satellites only provide the measurements. The data that needs to be transmitted to the CPS are defined in this document, together with the output data, after processing. Finally, a Multi-Target-Tracking approach is proposed and described.

Co-funded by the European Union

2 Introduction

2.1 Purpose of the document

This document contains the algorithm high level description for estimating the location of an aircraft via the satellite-based multilateration system proposed in SATERA, to track multiple aircraft along their trajectory and to monitor the integrity of the GNSS data transmitted by the aircraft.

2.2 Intended readership

This document is shaped for those who are directly or indirectly involved in the advancement, regulation, and operational management of European airspace. The intended readership includes, but is not limited to:

- European Commission and SESAR JU: For their roles in initiating and sponsoring projects to advance Air Traffic Management (ATM) capabilities.
- Regulatory Bodies and Aviation Authorities: Including European Union Aviation Safety Agency (EASA), overseeing safety and regulation, crucial for integrating new airspace operations.
- Global Institutions: Such as International Civil Aviation Organization (ICAO) and United Nations Office for Outer Space Affairs (UNOOSA), highlighting the need for global operational harmonization.
- EUROCONTROL Member States and National or Multinational Space Agencies: Planning policies and strategies for space and advanced aviation operations integration.
- Air Navigation Service Providers: Managing daily air traffic and integrating diverse operations within the ATM system.
- State Organisations and Military: With specific space operations requiring integration into civil and commercial operations.
- Aerospace Industry Stakeholders: Manufacturers and operators needing insights into interoperability and operational frameworks.
- Commercial Space Operators: Engaged in launch and satellite operations, benefiting from ATM integration processes.
- Research Institutions and Academia: Contributing to the knowledge base on space-based Communication, Navigation and Surveillance (CNS) and its integration into the European ATM.
- Policymakers and Government Agencies: Setting policies on airspace and space operations.
- Airspace Users: Including commercial, general aviation, and unmanned aircraft operators adapting to new operations.
- Other Industry Partners: Highlighting the need for advancements in communication, surveillance, and navigation to support future operations.

2.3 Background

SATERA proposes the design, development, and validation of a space-based composite ADS-B + E-MLAT surveillance system. This concept targets a twofold goal: to provide a GNSS-independent space-based surveillance layer additional to the state-of-the-art space-based ADS-B and to enable an

independent verification and validations of GNSS data received on the ground from space-based ADS-B systems.

The system envisages a constellation of small satellites in Low Earth Orbit (LEO) acting as receiving stations for ADS-B signals. Each satellite will be equipped with specialized instruments to capture ADS-B transmissions from aircraft and measure key electromagnetic wave characteristics, including Time of Arrival, Angle of Arrival, and Frequency of Arrival.

The collected data will be relayed between satellites and transmitted via a downlink to a CPS on the ground. Given the constraints of space-based communication—such as limited bandwidth and susceptibility to noise—optimized data encoding, compression, and routing algorithms will be employed to ensure reliable transmission while minimizing network congestion.

At the CPS, the data will undergo multilateration processing to estimate aircraft state. These estimates will be cross-checked against the vector state reported in ADS-B messages, with an integrity indicator added to the ADS-B Target Report.

Finally, both ADS-B and E-MLAT reports will be integrated into the ATM surveillance system for Air Traffic Control (ATC) operations (Figure 1).

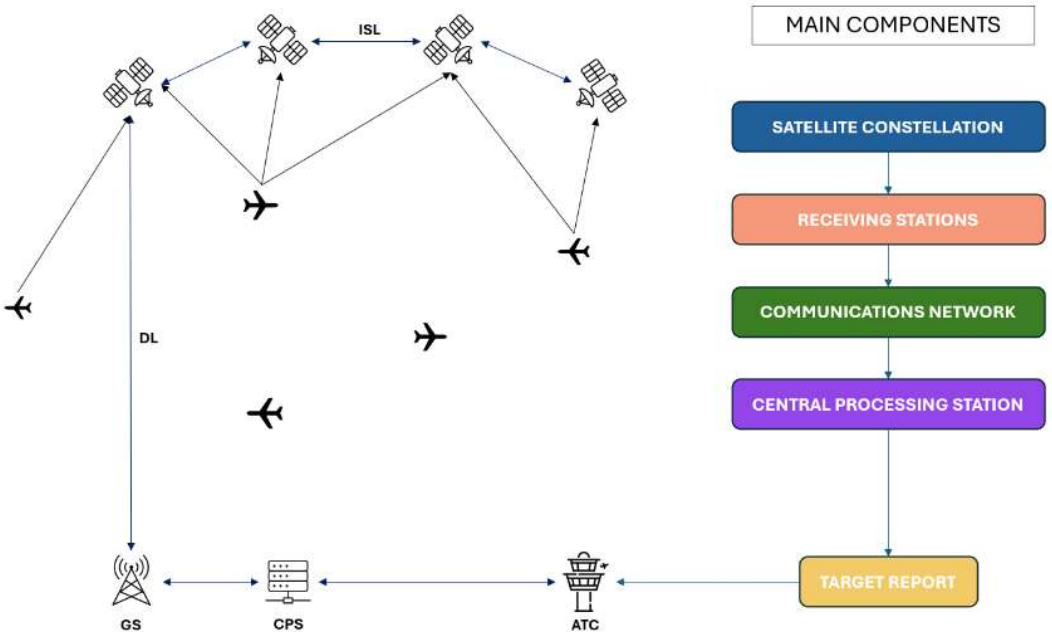

Figure 1: SATERA Block Diagram.

SATERA Expected Outcomes are:

- Functional requirements of a space-based composite ADS-B + E-MLAT.
- Evaluation tool of space-based composite ADS-B + E-MLAT surveillance systems.
- Simulator of MLAT receiving stations.
- Communication network simulator.

- Simulator of the central processing station.
- Space-based E-MLAT systems' performance prediction tool.

2.4 Scope

The space-based E-MLAT system is aimed at estimating the aircraft position independently from the use of GNSS, using TOA, FOA and AOA measurements taken on board LEO satellites. This document serves the following purposes:

- Provide the algorithm description for the initial position estimation of an aircraft that is being tracked for the first time.
- Provide the algorithm description for the subsequent tracking of the aircraft as it moves along a trajectory.
- Provide the algorithm description for the integrity checks performed in the CPS to validate the GNSS-based estimated position against the E-MLAT one.

2.5 Structure of the document

The rest of the document is organised as follows: Section 3 introduces an overall description of the CPS as presented in Deliverable D6.1 [1]; Section 4 presents the localisation algorithm for the initial position estimate for all measurement types; Section 5 is about the sequential filtering algorithms used for tracking the aircraft with subsequent MLAT measurements; Section 6 discusses the integrity indicator algorithm; the Appendix in Section 9 contains the source code of the MATLAB implementation for all algorithms discussed.

3 Introduction on the Central Processing Station

Recalling [1], the first important objective of the Central Processing Station is to produce a continuous estimation of the aircraft position in the coverage area of the system.

This type of problem, usually denoted as Multi Target Tracking (MTT) [2][3], consists of:

- **Detection and Measurement:** The first step in MTT involves detecting objects and obtaining measurements such as position, velocity, and acceleration. Sensors like radar, LiDAR, and cameras (or MLAT and ADS-B data in the SATERA case) are commonly used for this purpose. However, the collected data often contains noise and false detections, requiring preprocessing.

- **Data Association:** It determines which measurements belong to which target. Several techniques are used to handle this problem, such as: Nearest Neighbour, that assigns the closest measurement to each target; Joint Probabilistic Data Association, that considers multiple possible associations and assigns probabilities; Multiple Hypothesis Tracking, that maintains multiple hypotheses about associations and selects the most probable one over time.

- **State Estimation and Filtering:** Once data are associated, the system estimates and predicts the state of each target using filtering techniques. Among the most commonly used filtering methods are: the Kalman Filter (KF), which is effective for linear motion models with Gaussian noise; the Extended Kalman Filter (EKF), that handles non-linear motion models; the Unscented Kalman Filter (UKF), that provides better performance in highly non-linear systems; the Particle Filter, that is useful for complex and highly non-Gaussian tracking scenarios.

- **Track Management,** that involves:

    - *Track Initiation*: identifying new targets and creating unique track IDs for the target.

    - *Track Maintenance*: continuously updating and predicting target positions (using data association and state estimation).

    - *Track Termination*: removing tracks when targets disappear for an extended period.

A **track** in MTT refers to the representation of an individual target's movement over time. It typically includes the target's estimated position, velocity, and other relevant attributes based on sensor measurements. Tracks are continuously updated as new observations are received, allowing the system to maintain an accurate representation of each object's trajectory.

A typical way to represent the observables is the so-called **plot**, that refers to a single measurement or detection provided by a sensor (such as radar, LiDAR, or cameras, or in our case the MLAT and ADS-B data) at a given time. A plot typically includes information related to the position and velocity of the target. Since raw sensor data can contain noise and false detections, multiple plots are processed and associated with existing tracks to estimate target trajectories accurately.

In SATERA, the track represents the airplanes with their trajectory and history. Each airplane continuously transmits ADS-B messages that are captured from different satellites and transferred to the CPS via the satellite network.

The TOA, AOA and FOA measurement of each ADS-B message shall be grouped to form a plot for a given aircraft. The plot will contain all the measurements and the ADS-B message information. ***ADS-B messages also contain the ICAO address of the aircraft to clearly identify its identity and can be used for the plot-to-track association step.***

Having the plot associated with the respective track, it is possible to produce a new state estimation of the track using the track information (related to the past measurement) and the new measurements (the ones contained in the plot). Moreover, the track status can be also forecasted for future time instants. Finally, if the track does not receive plots for a given time, it should be terminated. Whenever a plot with an ICAO address not associated to any existing track arrives at the CPS, a new track must be initialised.

The described process is better detailed in the following, where details about the processing flows and interaction of the CPS with the other SATERA components are reported.

Co-funded by the European Union

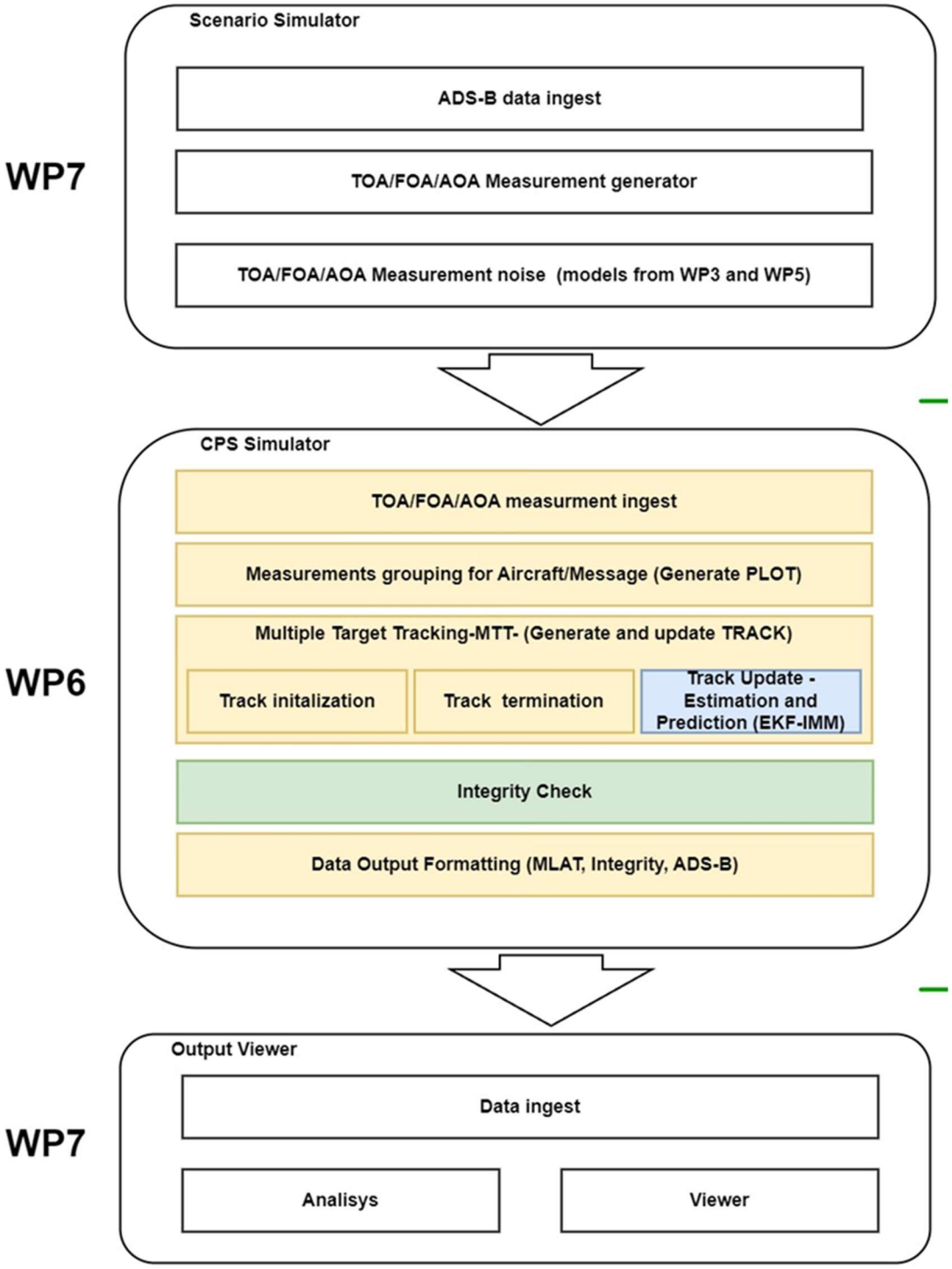

Figure 2: WP6 positioning within SATERA

Figure 2 represents the workflow for the CPS and the relationship with the simulations and evaluation activities planned in SATERA. The workflow is structured into three main parts:

Scenario Simulator (WP7)

This module is responsible for generating simulated scenarios by ingesting ADS-B data and producing simulated measurement types (TOA, FOA, AOA). It also incorporates measurement noise generated with the model proposed from other work packages (WP3 and WP5).

CPS Simulator (WP6)

The CPS simulator ingests satellite-based measurements, groups them per aircraft, and tracks multiple targets. Key functionalities are:

- **Measurement ingestion** (TOA/FOA/AOA).
- **ADS-B data extraction** to obtain the reported GNSS-based position data contained in ADS-B messages.
- **Data grouping** to generate PLOTs for each aircraft.
- **Multiple Target Tracking**, which includes:
    - **Track initialisation**.
    - **Track termination**.
    - **Track update, estimation, and prediction** using EKF-IMM (Extended Kalman Filter - Interacting Multiple Model) or other techniques.
- **Integrity check** for data validation.
- **Data output formatting**, which includes MLAT, Integrity, and ADS-B data formatting.

Output Viewer (WP7)

This module is responsible for processing and visualising the generated output. It includes:

- **Data ingestion.**
- **Analysis tools.**
- **A viewer** for visualisation.

Focusing on the data in input and output of the CPS:

- **Input Data:** Satellite measurement reports, containing: CPS time (a local reference time, can be synchronised with Coordinated Universal Time (UTC)), satellite number, TOA/FOA/AOA, ADS-B messages, satellite position, velocity, rotation matrix, and errors.
- **Output Data:** Processed, also in Asterix format, including CPS TOA, data from ADS-B messages, positions (ADS-B and MLAT), and integrity monitoring results.

Moreover, the internal data must be arranged to store, track and plot information:

- A track table storing information for each aircraft, including ICAO ID, time, ADS-B reported vector state, estimated MLAT estimated vector state, covariance matrix, track status, and last update.
- Plot table, containing the plots information for each time instant consisting of satellite measurements for each received ADS-B message.

Inside the CPS, the localisation problem needs to be solved based on the received measurements (TOA, FOA, AOA), i.e. computing the aircraft position and velocity. This will be done with localisation algorithms that will be described later and can be based on simple or sequential estimation, such as:

- Closed Form (CF) algorithms, i.e., Bancroft.
- Open Form (OF) algorithms, i.e., Weighted Least Square (WLS) based or Extended Kalman Filter.

Finally, having the MLAT position for the aircraft it will be possible to compute an integrity parameter to check the ADS-B data and to improve the system safety and security.

Figure 3 better details the main steps of the MTT and plot generation function.

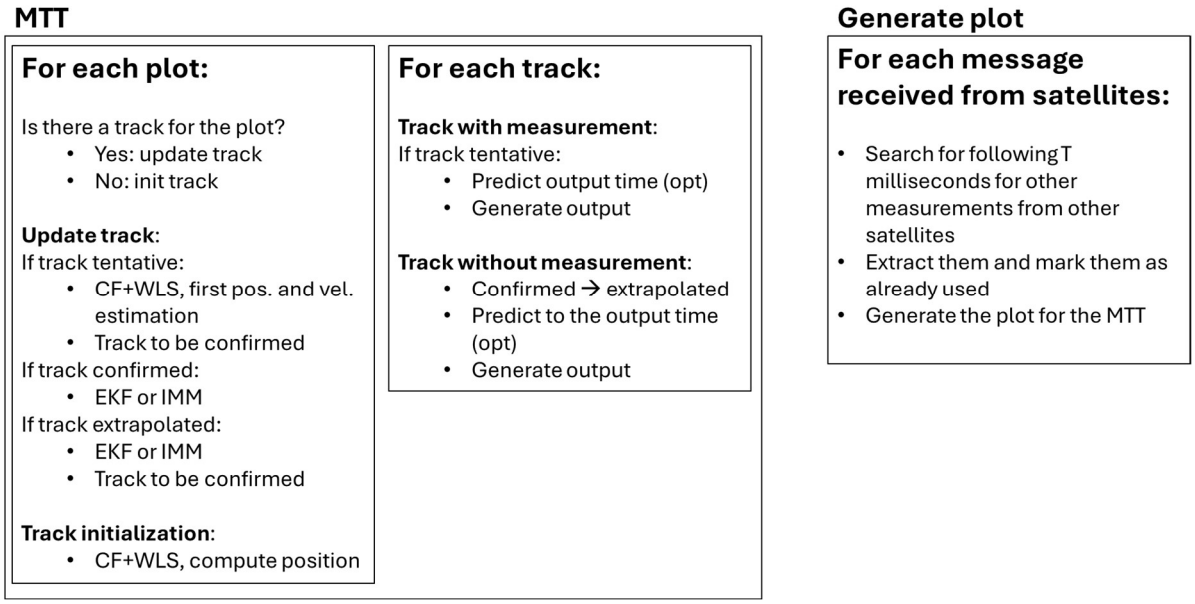

Figure 3: MTT logic.

3.1 Data description

The structure of the data used by the system is described in this paragraph. It is first divided in the following categories:

- **input** data to the CPS, coming from the satellites;
- data **internal** to the CPS, used to keep track of the state of every aircraft and estimate its position;
- data **output** from the CPS, with the results of the estimation.

3.1.1 Input data

In the form of CSV or binary file where each row represents a satellite report with the following columns, an estimate of the minimum size (in Bytes) required for each is given:

1. CPS Time → Timestamp from the CPS system (UTC time or UNIX time).

   nanoseconds precision: 7B [ns]

2. SAT # → Satellite identification number.

   ID: 2B

3. SAT TOA → Time of Arrival measured by the satellite.

   nanoseconds resolution: 7B [ns]

4. SAT FOA → Frequency of Arrival measured by the satellite.

   1Hz resolution: 3B [Hz]

5. SAT AOA → Angle of Arrival measured by the satellite.

   0.1 deg resolution (0.001745 rad): 2B [rad]

6. ADS-B message → ADS-B message received from the aircraft.

   It is standard: 14B

7. SAT pos → Satellite position in Earth-Centered Earth-Fixed (ECEF) coordinates.

   cm resolution: 4B [cm] per coordinate (tot: 12B)

8. SAT vel → Satellite velocity in ECEF coordinates.

   m/s resolution: 4B [m/s] per coordinate (tot: 12B)

9. SAT Rotation matrix → Satellite rotation matrix (for coordinate system transformations). Do not need the entire matrix, ϕ and λ are enough (see [1]).

   deg resolution (0.001745 rad): 2B (2x → 4B)

10. Measurements and SAT errors → Measurement errors associated with TOA, FOA, and AOA and satellite position error.

a. TOA meas. error: nanoseconds resolution : 3B [ns]

b. FOA meas. error: 1Hz resolution: 2B [Hz]

c. AOA meas. error: 0.1 deg resolution (0.001745 rad): 2B [rad]

d. POS error: cm resolution: 3B [cm]

Data coming from ground station:

CPS time

Data coming from initialisation variables:

Measurements and SAT errors

Everything else comes from satellites.

Expected CSV Format:

CPS Time, SAT #, SAT TOA, SAT FOA, SAT AOA, ADS-B message, SAT pos X, SAT pos Y, SAT pos Z, SAT vel X, SAT vel Y, SAT vel Z, SAT Rot Mat (3x3), SAT TOA error, SAT FOA error, SAT AOA error, Sat pos error.

Knowing the size of each element is crucial in the design of the inter-satellite network, since the data described here has to be routed to ground.

3.1.2 Internal data

3.1.2.1 Track Table (Aircraft Tracking)

CSV or binary file in which each entry represents the estimated state of an aircraft at a given timestamp, containing:

1. ICAO ID → Unique identifier of the aircraft.
2. Time → ADS-B message time.
3. ADS-B data → Data extracted from the received ADS-B message.
4. ADS-B position → Aircraft position reported by ADS-B (lat, lon, alt).
5. MLAT estimated data → Position, velocity and acceleration estimated using multilateration.
6. Covariance matrix → Covariance matrix of the estimated state.
7. Track status → Track state (initialisation, tentative, confirmed, extrapolated).
8. Last update → Last update timestamp for this track (CPS update time).
9. Num Satellites Used → Number of satellites contributing to the estimation.

3.1.2.2 Measurement Table (Plots)

CSV or binary file in which each row represents a measurement taken by a satellite at a given timestamp, containing:

1. CPS Time ⟶ Timestamp of the measurement in CPS.
2. ADS-B msg ⟶ Received ADS-B message.
3. TOAs ⟶ List of TOA measurements from satellites.
4. FOAs ⟶ List of FOA measurements.
5. AOAs ⟶ List of AOA measurements.
6. Sat # ⟶ Identification number of the satellite that took the measurement.
7. Sat pos ⟶ Position of the satellites at the time of measurement.
8. Sat vel ⟶ Velocity of the satellites at the time of measurement.
9. RM (Rotation Matrices) ⟶ Rotation matrices used for coordinate transformations.

3.1.3 Output data

CSV or binary file with one row per processed ADS-B message, containing:

1. CPS Time → Processing timestamp in CPS.

   ns resolution
2. ADS-B message → Original ADS-B message.
3. ADS-B position → Position reported by ADS-B (lat, lon, alt).

   cm resolution
4. MLAT position → Estimated position using multilateration (lat, lon, alt).

   cm resolution
5. Integrity monitor information → Integrity information of the estimated position.
6. Asterix formatted data → Data formatted according to ASTERIX standard:
   a. CAT020 (MLAT data)
   b. CAT021/053 (ADS-B data)

Co-funded by the European Union

4 Localisation Algorithms prototype

4.1 Introduction

Given the formulations for all the kinds of measurements considered in each SATERA satellite – TOA, FOA, AOA – this section presents the algorithms that can be used to estimate the aircraft state vector (hence the location and, if possible, the velocity), when there are a minimum number of visible satellites that can receive the ADS-B message broadcasted by the aircraft.

The minimum number of satellites depends on the types of measurements considered, or equivalently, the number of unknown variables in the state vector.

4.2 SATERA localisation algorithm

Recalling deliverable D6.1 [1], closed form algorithms are the fastest and least computationally expensive, but they are much more sensitive to the noise. To achieve the best possible performance and achieve an Root Mean Square (RMS) error closest to the Cramér Rao Lower Bound (CRLB), an open form algorithm can be used in the form of WLS.

The Bancroft algorithm (formulation in [1]) was chosen as it is the most widely known and easily implemented. For the WLS iterations, the Tikhonov regularisation algorithm (formulation in [1]) can be used when ill-conditions are encountered.

To know when this is needed, a Singular Value Decomposition (SVD) spectrum analysis is carried out at each WLS iteration, by checking whether the condition number of the matrix that is being inverted is growing.

Figure 4 explains the functioning of the entire proposed localisation algorithm:

- The measurements taken by each satellite are tied to the aircraft state vector $\boldsymbol{\theta}$ (hence the location) by a characteristic equation.
- The objective is to invert the (non-linear) system of equations to find $\boldsymbol{\theta}$.
- First, the Bancroft algorithm is used to compute an initial guess. This estimate is less accurate than the one achievable by WLS. Remember that regardless of the eventual extra measurement types available (FOA or AOA), for the Bancroft algorithm only the TOA measurements are used.
- If Bancroft cannot be used (i.e.: there are only 3 satellites in view), the initial guess is taken in a point that is the centroid between the available satellites, at a given altitude.
- The WLS algorithm is used to improve the estimate of $\boldsymbol{\theta}$.
- WLS requires a matrix inversion, that can happen to be ill-conditioned. At each WLS iteration, the SVD spectrum of such matrix is analysed: when the condition number increases it means that the solution is not converging and the matrix is ill-conditioned, so the Tikhonov algorithm is used to regularise it.

Co-funded by the European Union

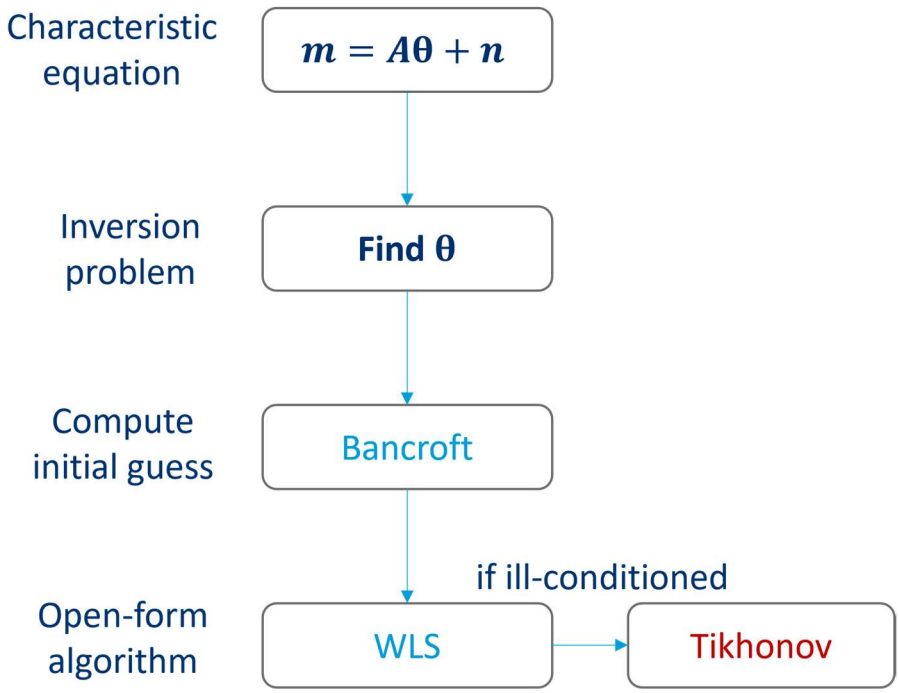

Figure 4: Proposed localisation algorithm

The complete formulation for the entire combination of algorithms is the following:

1. **Compute the initial guess point:**
   - **a.** If there are 4 or more visible satellites, use the Bancroft algorithm using TOA measurements (from [1]): $\hat{\boldsymbol{s}}_a^{1,2} = \Lambda^{1,2}\boldsymbol{d} + \boldsymbol{e}$
   - **b.** If there are less than 4 visible satellites, compute the centroid (horizontally) between the available satellites. The point is taken at a given altitude, typical of an en-route trajectory.
2. **Compute the final estimate (iterative):**
   - **a.** Run the WLS algorithm using all available measurements (from [1]):
     $$\widehat{\boldsymbol{\theta}}^k = \left(\boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \boldsymbol{W}\boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)\right)^{-1} \boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \boldsymbol{W}\widehat{\boldsymbol{m}}_\Delta\left(\widehat{\boldsymbol{\theta}}^{k-1}\right) + \widehat{\boldsymbol{\theta}}^{k-1}, \qquad k = 1, \dots, K$$
     where K is the maximum number of iterations.
   - **b.** The Jacobian matrix depends on the available measurements; it is taken from [1].
3. **Regularisation (for each iteration, if needed):**
   - a. If ill-conditioning is encountered while running the WLS algorithm, the matrix to be inverted is regularised using the Tikhonov algorithm, summarised by the following (from [1]):
     $$\widehat{\boldsymbol{\theta}}^k = \left(\boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right) + \lambda^2\right)^{-1} \boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \widehat{\boldsymbol{m}}_\Delta\left(\widehat{\boldsymbol{\theta}}^{k-1}\right) + \widehat{\boldsymbol{\theta}}^{k-1}, \qquad k = 1, \dots, K$$
4. **Exit condition:**
   - **a.** When $\left(\boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \boldsymbol{W}\boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)\right)^{-1} \boldsymbol{J}\left(\widehat{\boldsymbol{\theta}}^{k-1}\right)^T \boldsymbol{W}\widehat{\boldsymbol{m}}_\Delta\left(\widehat{\boldsymbol{\theta}}^{k-1}\right) < \epsilon$ → exit from iteration and give the state vector in output.

b. If $k=K$ → stop iterating, algorithm is not converging, the state vector of the target is not produced.

All the equations above were derived and described more in detail in [1]. They are reported here as reference for the MATLAB implementation that follows.

4.3 Algorithm prototype

There are multiple functions that make up the complete algorithm, each one implementing one of the single algorithms described above: the Bancroft algorithm (or centroid when needed) is used to compute an initial guess and WLS is the chosen open-form algorithm that uses Tikhonov for regularisation.

4.3.1 Initial guess computation

The function is described in Algorithm 1 and it is characterised by the following inputs and outputs:

Input:

- **initialGuessAlgo**: a string that can be either "*bancroft*" or "*centroid*". Allows choosing the preferred algorithm to compute the initial guess. Note that when the number of visible satellites is less than 4, the centroid algorithm will be used regardless of the value of this parameter.
- **satPositions**: matrix containing the position coordinates of every satellite in view of the aircraft. Each row corresponds to a satellite and each column to a coordinate (**x, y, z**) in ECEF:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}$$

- **aircraftAlt**: Typical aircraft altitude. Used when computing the centroid.
- **measurements**: vector containing the TOA measurements, one for each satellite in view. Not needed if using the centroid algorithm.

Output:

- **startingPoint**: vector containing the coordinates of the first estimate of the point, to be used as initial guess in WLS.

Algorithm 1

```
function computeInitialGuess(initialGuessAlgo, satPositions, aircraftAlt,
                             measurements)
    // if number of visible satellites is lower than 4, use centroid anyway
    if(not_enough_sats(satPositions))
        initialGuessAlgo = 'centroid'
    switch(initialGuessAlgo)
        case 'centroid':
            startingPoint = centroid(satPositions, aircraftAlt)
        case 'bancroft':
            startingPoint = bancroft(satPositions, aircraftAlt, measurements)
    return startingPoint
```

The function `not_enough_sats` checks if the number of visible satellites is lower than 4, in which case the Bancroft algorithm cannot be run.

The function `centroid` is further described in Algorithm 2. It computes the initial guess by averaging the positions of the satellites horizontally (latitude and longitude) and by setting the point to the given altitude. This should be the typical altitude at which we expect the aircraft to be depending on the use case, i.e.: 10000m.

The function `bancroft` is further described in Algorithm 3. It computes the initial guess by running the Bancroft algorithm as described in [1].

Algorithm 2

```
function centroid(satPositions, aircraftAlt)
    // compute horizontal centroid of satellites
    centroid = average_lat_lon(satPositions)
    // set altitude to aircraft altitude
    startingPoint = set_altitude(centroid, aircraftAlt)
    return startingPoint
```

The function `average_lat_lon` computes the average latitude and longitude among all provided satellites positions.

The function `set_altitude` returns a point with the provided latitude and longitude centroid and the given altitude.

Algorithm 3

```
function bancroft(satPositions, aircraftAlt, measurements)
    // run Bancroft algorithm to compute 2 possible solutions
    possibleSolutions = bancroft_algorithm(satPositions, measurements)
    // choose the solution closest to the centroid
    centroid = centroid(satPositions, aircraftAlt)
    startingPoint = choose_closest_solution(possibleSolutions, centroid)
    return startingPoint
```

The function `bancroft_algorithm` computes the two possible solutions as per the Bancroft algorithm, described in [1].

The function `choose_closest_solution` selects the solution which is closest to the provided point.

4.3.2 WLS with Tikhonov

The function is described in Algorithm 4 and it is characterised by the following inputs and outputs:

Input:

- **startingPoint**: vector containing the coordinates of the first estimate of the point, to be used as initial guess in WLS.
- **maxIters**: maximum number of iterations.
- **measurements**: vector containing the measurements taken in each satellite. They can be TOA, FOA, AOA measurements, depending on the measurement type that was configured.
- **type**: string indicating measurement type. Can be one of: ["*toa*", "*foa*", "*aoa*", "*toa_aoa*", "*toa_foa_aoa*"].
- **errors**: [*sigmaT sigmaF sigmaA sigmaP*] vector containing the standard deviations for the errors of respectively time, frequency, angle, satellite position; expressed in [s Hz rad m].
- **satPos**: matrix containing the position coordinates of every satellite in view of the aircraft. Each row corresponds to a satellite and each column to a coordinate (**x, y, z**) in ECEF:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}.$$

- **satVel**: matrix containing the velocity components of every satellite in view of the aircraft. Each row corresponds to a satellite and each column to a velocity component (**vx, vy, vz**) in ECEF:

$$\begin{bmatrix} v_{x_1} & v_{y_1} & v_{z_1} \\ \vdots & \vdots & \vdots \\ v_{x_N} & v_{y_N} & v_{z_N} \end{bmatrix}.$$

- **f0**: frequency of the ADS-B signals received from the satellites.

Output:

- **thetaCur**: current estimate of the aircraft position.

Algorithm 4

```
function taylorWLS(startingPoint, maxIters, measurements, errors, type,
                   satPos, satVel, f0)
    for(iter = 1 to maxIters)
        compute_jacobian_covariance(type, theta, errors, f0, satPos, satVel)
        if(condition_number_increased(jacobian))
            taylorWLSTikhonov()
        compute_next_estimate(jacobian, covariance, measurements, thetaCur)
        if(converged())
            return theta
        else
            return NaN
```

The function `compute_jacobian_covariance` computes the Jacobian and covariance matrices as described in [4].

The function `condition_number_increased` checks whether the conditions number has increased by more than 10%.

The function `taylorWLSTikhonov` restarts the Taylor iterations adding Tikhonov regularisation, as explained in [1].

The function `compute_next_estimate` computes the next estimate of the state vector $\boldsymbol{\theta}$.

The function `converged` checks whether the algorithm has converged. I.e.: the difference between the last two state vectors is small (See [1]).

4.4 Prototype validation

A prototype was developed with the purpose of validating the chosen algorithms. The prototype is capable of running a given number of Montecarlo (MC) simulations to evaluate the RMS error of the localisation algorithm. It is possible to provide an entire trajectory and to perform the position estimation on all the waypoints that make up the trajectory. Furthermore, it is possible to compare the RMS error with the theoretical limit given by the CRLB, reusing the formulation from [4].

Most tests about the achievable performance of the entire system were carried out in [4]. By demonstrating that the chosen algorithm can achieve a performance that converges to the CRLB, the correctness of the implementation is validated.

The following tests were performed:

1 Reducing the errors standard deviation, the localisation performance improves.

2 Increasing the number of Montecarlo iterations makes the RMS error converge towards the CRLB.

3 The performance does not improve when the number of measurements decreases.

4 Adding a new measurement type cannot reduce the overall performance.

5 A higher number of visible satellites (i.e.: due to a higher antenna gain) leads to a better performance.

6 The Bancroft algorithm provides a better estimate of the initial guess, using the centroid instead should not lead to a better overall performance.

4.4.1 Simulation and new results

A qualitative evaluation was done by running the localisation algorithm over the waypoints that form a complete trajectory over one of the typical routes of interest for SATERA.

The following simple scenario was used to validate the code on a real aircraft path. The CRLB is also evaluated as a benchmark to be compared with the RMS error of the algorithm output. A number of Montecarlo iterations are run to calculate the RMS error. The results are then discussed to check whether they reflect the expectations.

4.4.1.1 Scenario description

The scenario reflects the tests carried out in [4].

The chosen LEO satellites constellation is a Walker Star with an inclination of 86°, at an altitude of 500 km. 200 satellites are deployed over 10 orbital planes, which means that there are 20 satellites on each plane.

The simulated aircraft moves on a route that goes from North America (New York) to Europe (Madrid). The chosen route is orthodromic, that is, the shortest route between the two positions. The results show 100 waypoints sampled along the trajectory. At each waypoint, CRLB and RMS error are calculated.

The simulator uses a link budget to compute which satellites are in view of the aircraft, so that the relative measurements can be used. The link budget parameters are the following:

- Receiver antenna gain: 12dB (14dB for FOA)
- Transmitter Power: 125W
- Transmitter antenna gain: 3dB
- Faraday rotation loss: 3dB
- Sensitivity: -97dBm

The measurement errors standard deviations are taken from [4] for TOA (~32.8 ns) and FOA (500 Hz) measurements and for the satellite position error (~0.99 m). For AOA measurements, given the current state of the errors analysis, an error of 1° for the sake of the simulation was assumed.

The receiver antenna gain was increased to 14dB for the FOA-only test, as a higher number of satellites in view is needed to compute a position.

4.4.1.2 TOA

The TOA-based localisation works as expected and the results show a convergence towards the predictions of the CRLB. There is no solution when only 3 satellites are visible and there is only one point where the system has bad performance due to bad geometry. A qualitative analysis can be done using Figure 5. From a quantitative point of view, in the points where the system is available, the average CRLB was $22.537\ m$, the average RMS error is different for every test but tends towards the CRLB as the number of Montecarlo iterations increase. An example sequence of tests gave the following results:

- 1000 Montecarlo iterations: average RMS error was 22.5738.
- 10000 Montecarlo iterations: average RMS error was 22.5207.
- 100000 Montecarlo iterations: average RMS error was 22.5402.

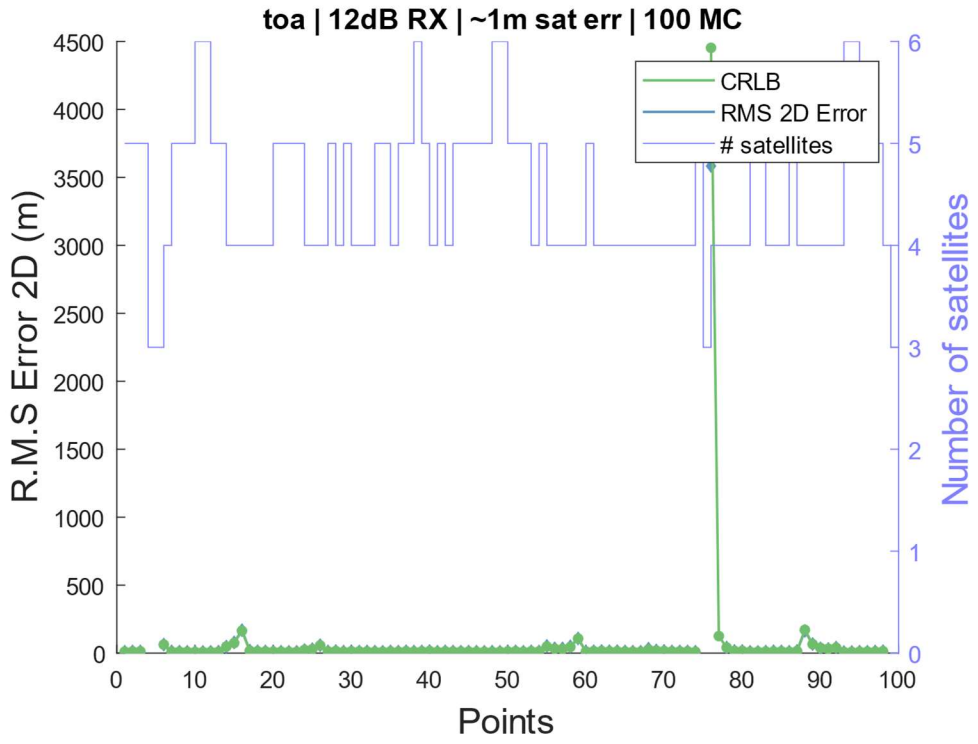

Figure 5: TOA-based localisation along the trajectory.

4.4.1.3 FOA

The FOA-based localisation works as expected and the results show a convergence towards the predictions of the CRLB (see Figure 6). FOA-only localisation requires at least 7 visible satellites, due to the higher number of parameters in the state vector, i.e.: the velocity components are estimated too. For this reason, the test was carried out using a configuration that included an increased receiver antenna gain.

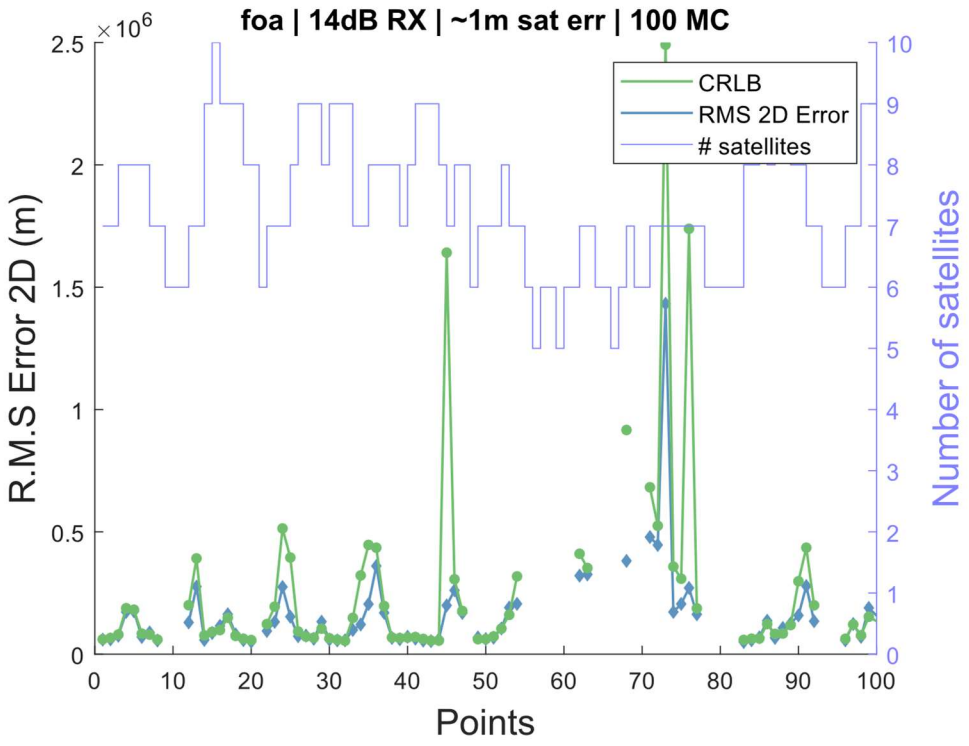

Figure 6: FOA-based localisation along the trajectory.

4.4.1.4 AOA

The AOA-based localisation works as expected and the results show a convergence towards the predictions of the CRLB. In the first test (Figure 7) it seems as if the RMS error could converge to a value that is below the CRLB. However, the measurement errors are high and the estimated position has both a very high RMS and CRLB.

The system is not able to compute a solution in the points of bad geometry, where the CRLB value shows spikes.

In order to validate that the algorithm implementation works correctly, another test was carried out using a lower angle measurement error (Figure 8), which is not realistic, but shows how the RMS error converges towards the CRLB.

Figure 9 and Figure 10 show that with high errors, a small fraction of the Montecarlo iterations had produced a valid solution, the only ones were those with lower RMS, while the rest would not converge in the WLS algorithm. By reducing the measurement errors, in almost every iteration the algorithm converges to a solution and the RMS error is close to the CRLB.

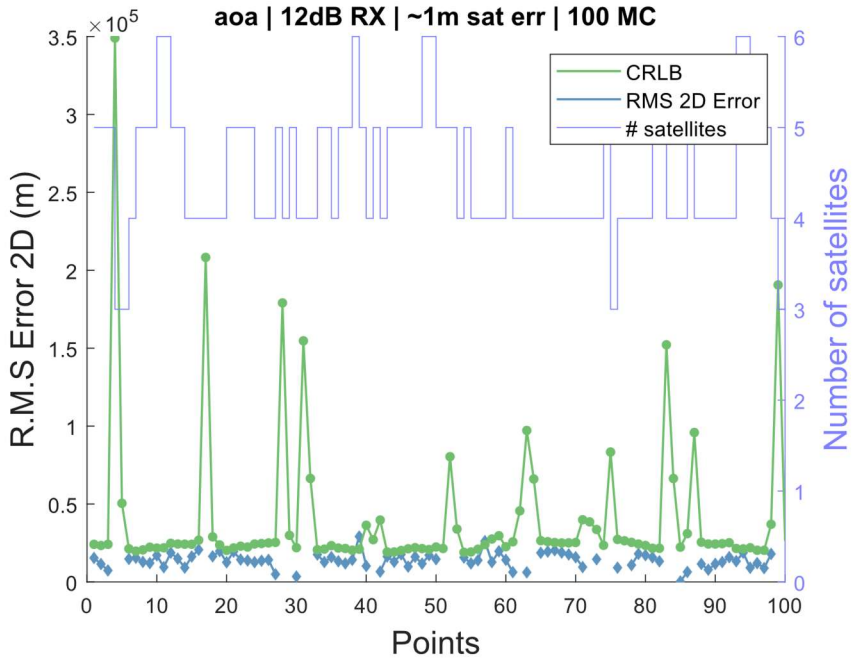

Figure 7: AOA-based localisation along the trajectory.

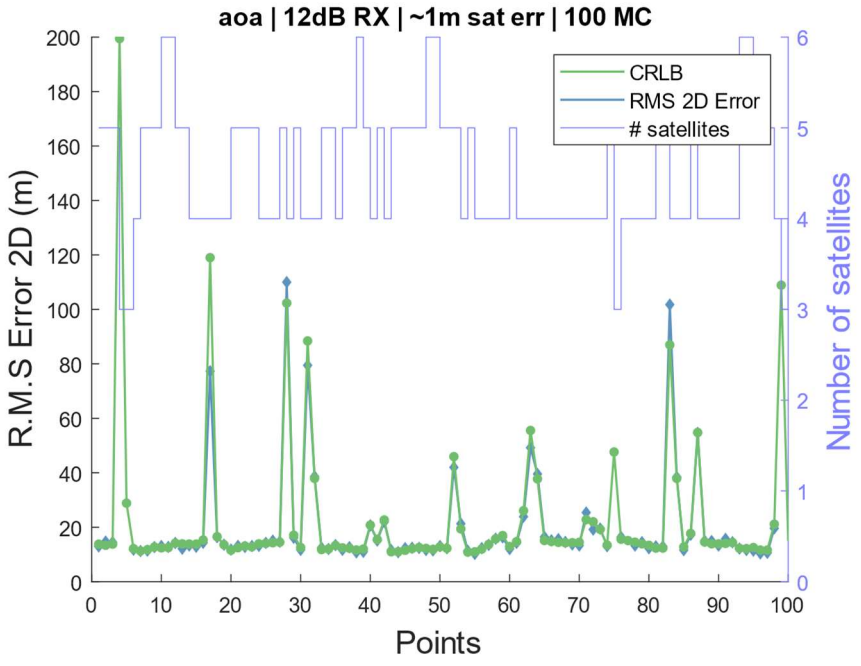

Figure 8: AOA-based localisation along the trajectory, using reduced measurement error.

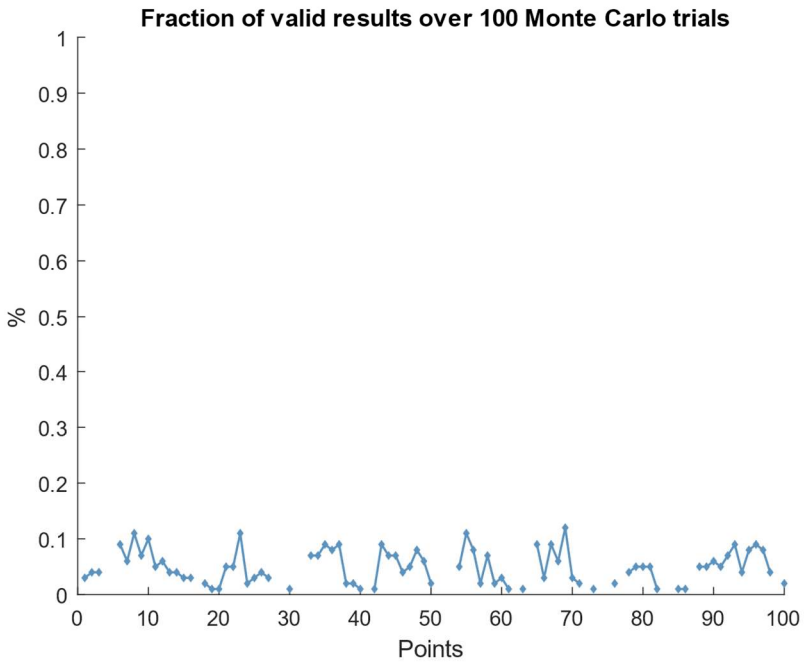

Figure 9: AOA-based localisation along the trajectory, valid results.

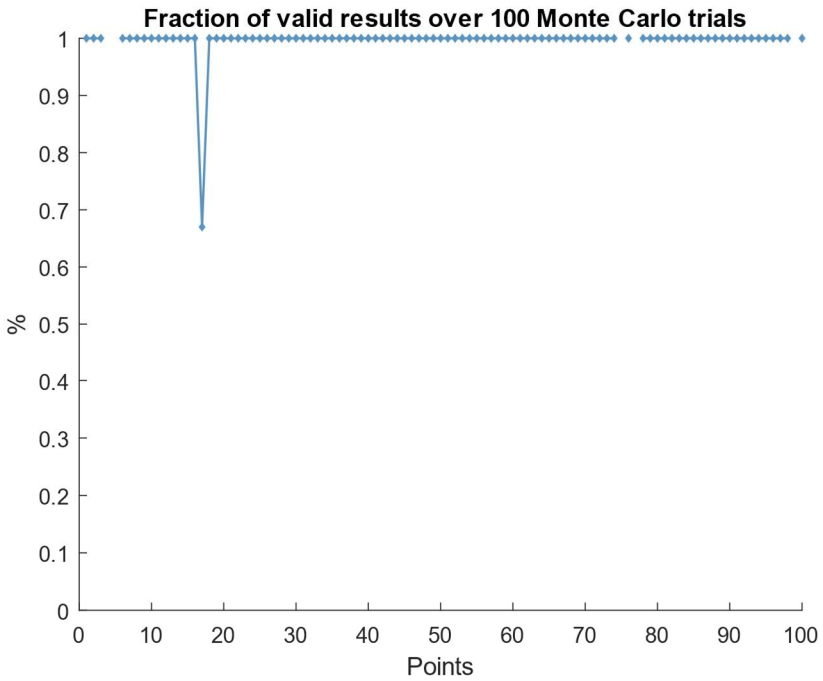

Figure 10: AOA-based localisation along the trajectory, using reduced measurement error, valid results.

4.4.1.5 TOA+AOA

The TOA + AOA localisation works as expected and the results show a convergence towards the predictions of the CRLB.

Figure 11 shows that the addition of AOA measurements makes it possible to find a solution even when the number of satellites is lower than 4, although having higher error.

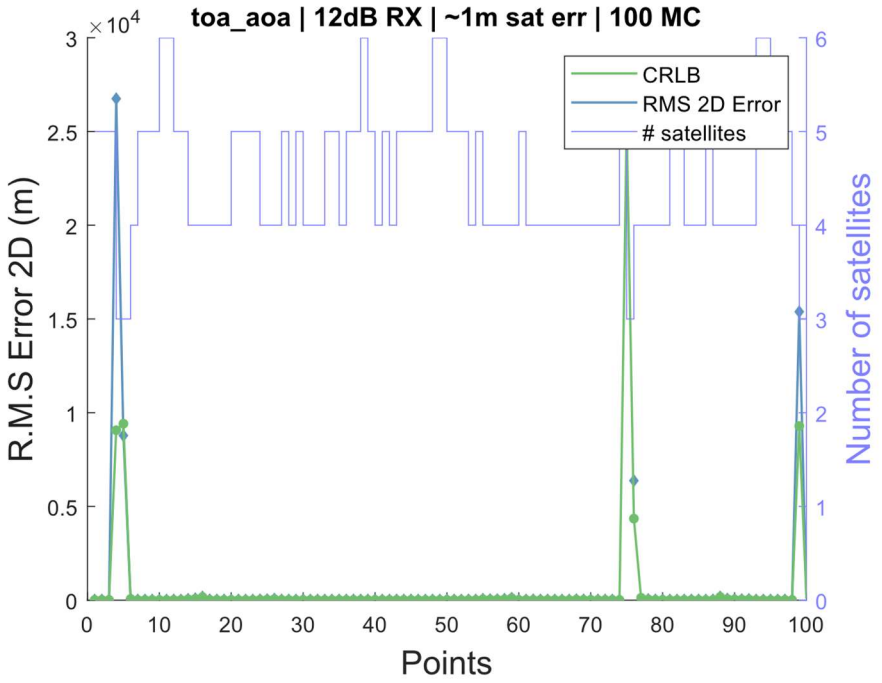

Figure 11: TOA+AOA localisation along the trajectory.

4.4.1.6 TOA+FOA+AOA

The TOA + FOA + AOA localisation works as expected and the results show a convergence towards the predictions of the CRLB.

Figure 12 shows that the addition of FOA and AOA measurements make it possible to find a solution even when the number of satellites is lower than 4, although having higher error.

Figure 13 shows that with unrealistic (in the SATERA scenario) low frequency and angle measurement error values, the RMS error, and CRLB, drop to a few metres even with a number of satellites that is lower than 4.

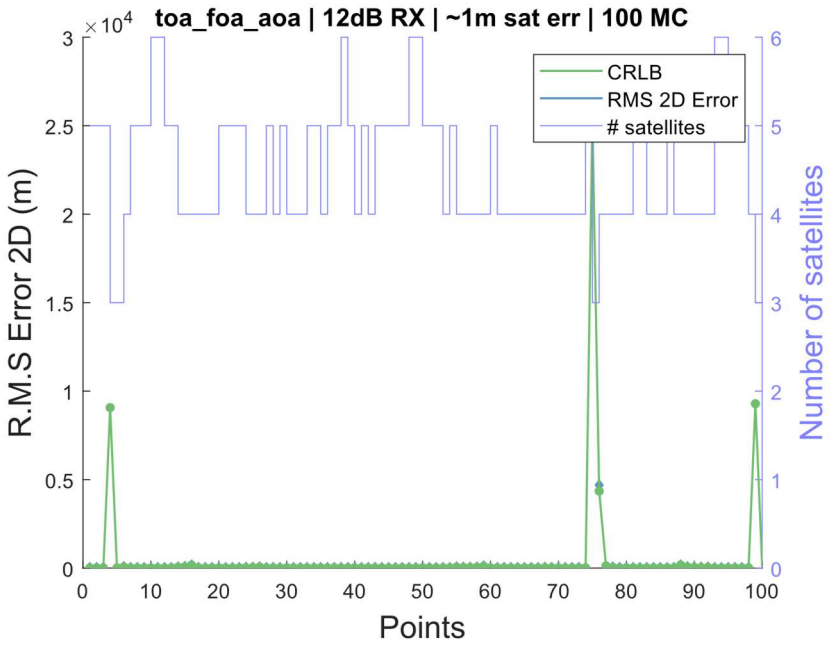

Figure 12: TOA+FOA+AOA localisation along the trajectory.

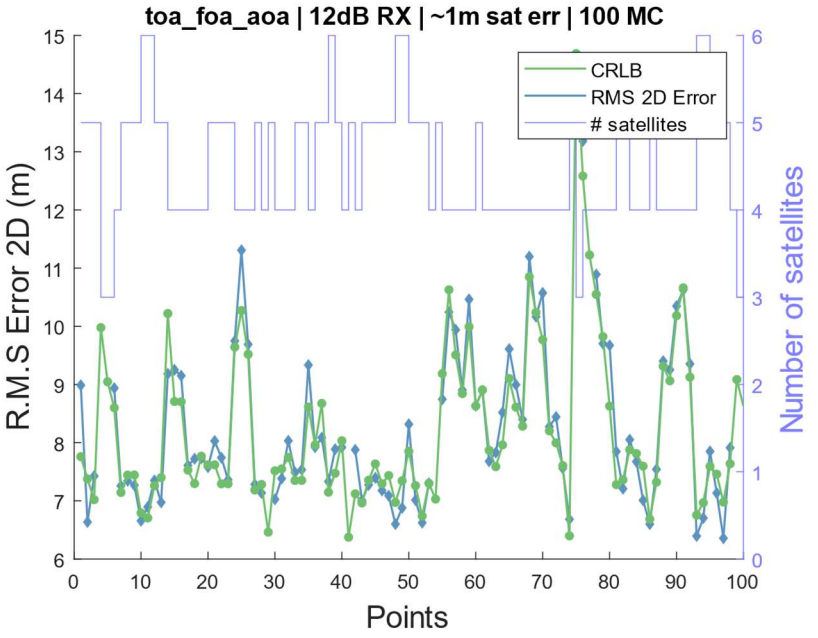

Figure 13: TOA+FOA+AOA localisation along the trajectory, using reduced measurement error.

4.4.1.7 Centroid for initial guess

Figure 14 shows the performance of the localisation algorithm when the centroid is used as initial guess instead of using the Bancroft algorithm.

Most times the WLS converges to the same solution, but there are some cases where the centroid is too far from the real aircraft position and the RMS error is higher than expected.

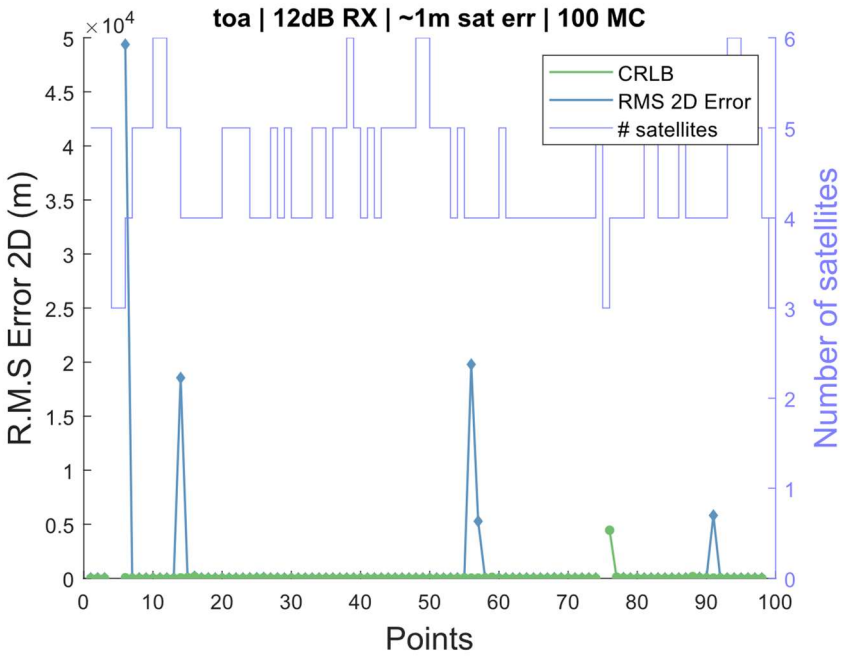

Figure 14: TOA-based localisation along the trajectory, using centroid as initial guess.

4.4.1.8 Increased number of Montecarlo iterations

Figure 15 and Figure 16 show from a qualitative point of view how the RMS error tends to the CRLB when the number of Montecarlo iterations increase.

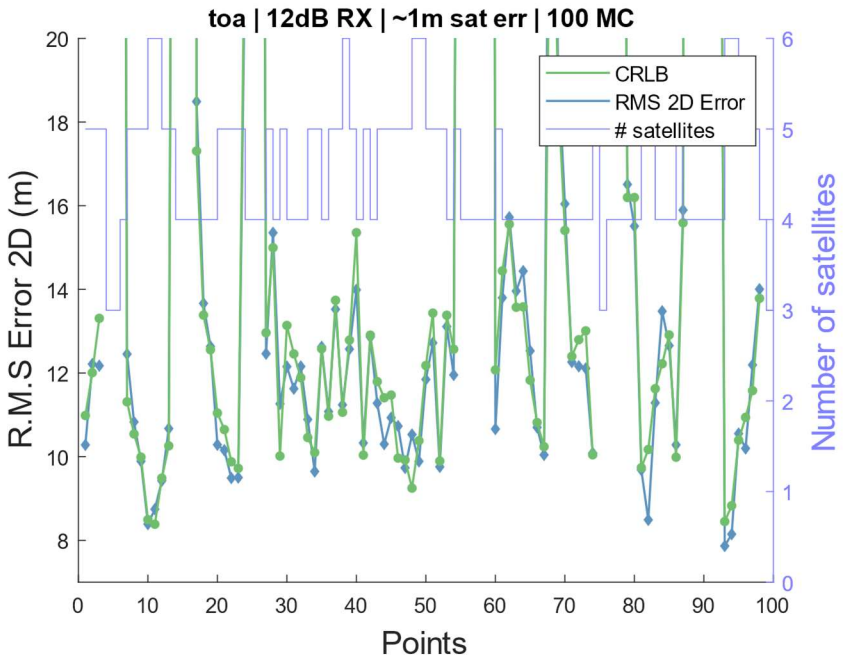

Figure 15: TOA-based localisation along the trajectory, zoomed, 100 Montecarlo iterations.

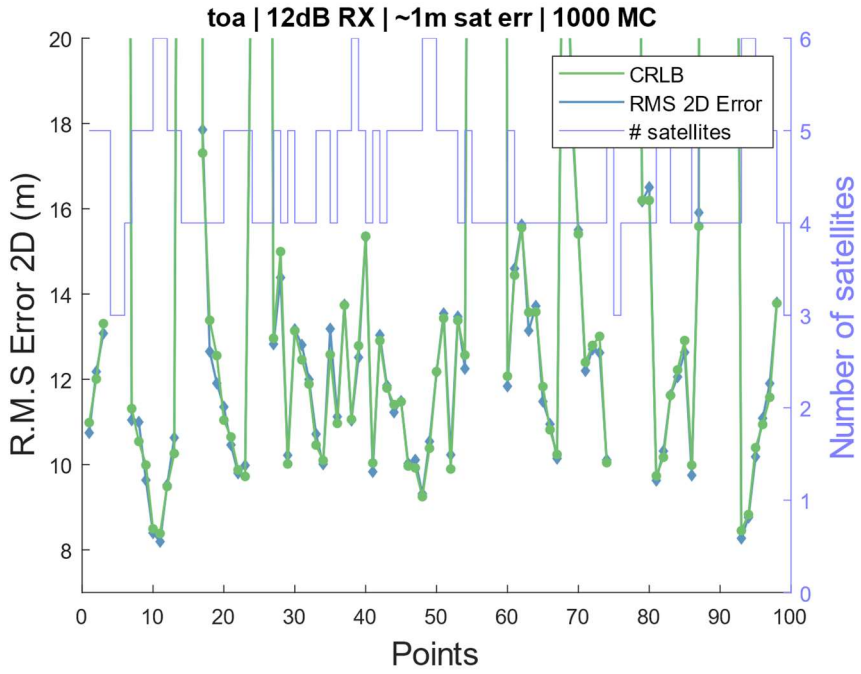

Figure 16: TOA-based localisation along the trajectory, zoomed, 1000 Montecarlo iterations.

4.4.1.9 No Tikhonov regularisation

Figure 17 shows a TOA-based localisation test that was carried out without the use of Tikhonov regularisation within the WLS algorithm. It appears as if the addition of regularisations techniques does not have an impact on the final performance.

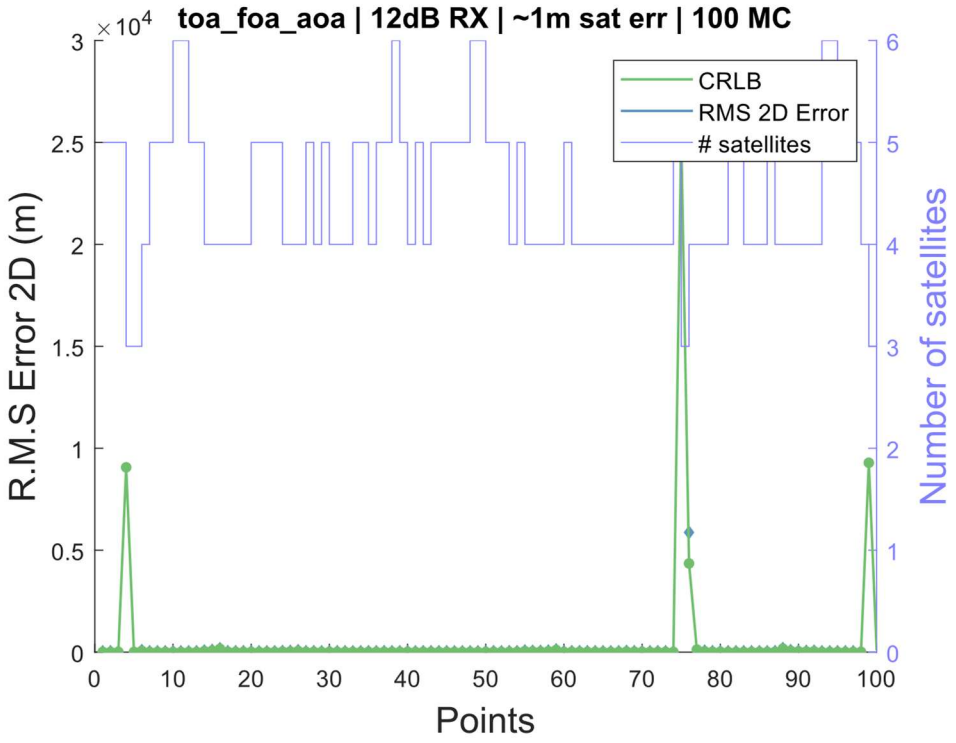

Figure 17: TOA-based localisation along the trajectory, without Tikhonov regularisation.

4.5 Conclusions

The results presented confirm the same trends previously identified in D6.1: No significant differences are observed.

5 Sequential Filtering for MLAT

5.1 Introduction

This section builds on the concepts introduced in D6.1 [1] and focuses on the practical validation of sequential filtering algorithms within the CPS prototype. The aim is to demonstrate how the CPS performs MTT, estimating aircraft trajectories from satellite-based MLAT measurements derived from ADS-B signals. While the overall tracking logic follows the framework defined in D6.1, the emphasis here is placed on the evaluation of different filtering approaches, EKF, UKF, H_∞ filter, and their integration into an Interacting Multiple Model (IMM) scheme.

5.2 Multi Target Tracking

The MTT module is responsible for managing and updating the trajectories of multiple aircraft detected by the CPS. It processes satellite measurement reports, groups them into consistent sets (plots), and recursively estimates the aircraft states using filtering algorithms.

The main workflow can be summarised as follows:

- **Input processing:** satellite reports are grouped by aircraft ICAO address and timestamp of transmission. Temporal consistency checks are applied using provisional thresholds on data age and latency spread.
- **Track management:** for each aircraft, a track table is maintained with state estimates, covariances, and track status (tentative or confirmed). Tracks are initialised, confirmed, updated, or terminated depending on the availability of valid plots.
- **Recursive filtering:** confirmed tracks are propagated using prediction and update steps. The IMM framework supervises multiple dynamic models, ensuring robustness to manoeuvring trajectories.

The current implementation operates in **offline mode**, since no communications network has yet been modelled. Grouping by ICAO and timestamp is therefore adopted as a provisional design choice. Once a network simulation becomes available, the logic may be adapted to support real-time operation.

5.2.1 Input processing and plot generation

The input to the CPS consists of measurement reports generated by multiple satellites, each including an ADS-B message, multilateration observables (TOA, FOA, AOA), and satellite metadata (position, velocity, clock information, etc.). To construct a consistent measurement plot:

- Reports are grouped by ICAO address and transmission timestamp.
- Within each group, data are validated using temporal constraints:
    - **Data age:** maximum time allowed between message emission and reception at the CPS.

- **Latency spread:** maximum allowable difference between reception times across satellites.

The resulting plots constitute the input to the tracking module, ensuring that only temporally coherent information contributes to state estimation

5.2.2 Track processing and management

Each aircraft is represented by an individual track stored in the CPS track table. This includes the latest state estimate, error covariance, track status, and timestamp of the last update. The track lifecycle follows these steps:

- **Initialisation:** a tentative track is created when a new ICAO address is detected, using a single localisation estimate as the initial position.
- **Confirmation:** after accumulating multiple valid estimates (e.g. three), a complete state vector (position, velocity, acceleration) is formed, and the track is upgraded to confirmed status.
- **Update:** confirmed tracks are propagated through prediction and, whenever possible, corrected using new measurement plots.
- **Termination:** if no valid plots are available for a configurable period, the track is terminated and removed from the table.

At the core of the update step, several filtering strategies have been implemented and tested:

- **Extended Kalman Filter:** recursive estimator using first-order linearisation of the measurement model.
- **Unscented Kalman Filter:** estimator based on sigma-point approximation, improving accuracy in strongly nonlinear regimes.
- H_∞ **filter:** robust filter providing guaranteed error bounds in the presence of modelling uncertainties.
- **Interacting Multiple Model:** supervisory framework combining multiple dynamic models (Constant Velocity, Constant Acceleration, and Singer), coordinating the prediction and update stages of the filters.

The inputs to the filters are the state vectors (position, velocity, acceleration) and the measurement plots generated during input processing. The outputs are updated state estimates and associated error covariances, which are stored in the track table and later used for performance assessment.

5.3 Algorithm prototype

The algorithm prototype implements the sequential tracking of aircraft trajectories based on position estimates obtained from the localization algorithms described in Section 4. Once three valid position estimates are available for a given aircraft, the CPS starts the tracking procedure. The prototype is organized in three main functional blocks: *position_tracking*, *IMM*, and the sequential filters (*EKF*, *UKF*, H_∞).

5.3.1 Position tracking

The *position_tracking* function (Algorithm 5) coordinates the tracking of each aircraft. Its main role is to initialise the track if it is the first iteration, manage the historical state and covariance, and call the IMM filter for recursive state estimation.

The function takes as input a flag indicating whether this is the first iteration, three previous position estimates to construct the initial state vector, the incoming data block containing ADS-B messages and TOA/FOA/AOA measurements together with satellite ephemeris, the sampling period, and (once the track has been initialised) the previous block with the state, covariance and manoeuvre model probabilities from the last step.

Additional parameters specify the type of measurements considered and the standard deviations associated with them. Its outputs are the updated state vector, the associated covariance matrix, and auxiliary variables required by the IMM framework.

Algorithm 5

```
function position_tracking (flag, estimation_1, estimation_2, estimation_3,
bloque, T, bloque_anterior, measures, SIGMA_vec)
      if (flag == 0)
          IMM (modeProb, Transprob, F, Q, xm, xp, pos_satellites,
vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec)
      else
          IMM (bloque_anterior.mode_Probability{1}, Transprob, F, Q,
bloque_anterior.pos_vel_acc{1}, bloque_anterior.individual_cov_matrix{1},
sat_pos, sat_vel, TOAs, FOAs, AOAs, measures, SIGMA_vec)
      end
return state_vector,cov_matrix
```

Recursive state estimation is therefore carried out by delegating the computation to the IMM filter, described in the following section.

5.3.2 IMM Filtering

The IMM filter (Algorithm 6) combines the predictions of different manoeuvre models and updates their probabilities according to the received measurements. Its inputs are the initial or updated model probabilities, the transition probability matrix, the state transition and process noise matrices for each model, and the current measurement set (TOA, FOA, AOA), together with satellite information and the measurement noise statistics. Depending on whether it is the first iteration or the track is already

active, the function either uses default initial conditions or the state and covariance from the previous block.

Algorithm 6

```
function IMM (modeProb, Transprob, F, Q, xm, xp, pos_satellites,
vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec)
for (iter = 1 to nModels)
      EKF /UKF / H_inf (x, P, Fa, Q, satelites, vel_satelites, TOAs, FOAs,
AOAs, measure, SIGMA_vec)
 end
return state_vector, cov_matrix, modeProb
```

The outputs of the IMM are the fused state estimate (position, velocity, acceleration), the updated covariance matrix, and the updated model probabilities. Each call to the filter (EKF, UKF, or H_∞) is detailed in the next section.

5.3.3 Sequential Filtering (EKF, UKF, H_∞)

Within each iteration of the IMM, a dedicated filter is executed for each motion model (Algorithm 7). The filter can be selected among the EKF, the UKF, or the H_∞ filter, depending on the configuration. The procedure follows the standard two-step structure: in the prediction step, the state and covariance are propagated using the manoeuvre model and process noise; in the update step, the prediction is corrected with the available measurements (TOA, FOA, AOA). For nonlinear cases, the required linearisation (EKF Jacobian) or transformation (UKF sigma points) is applied internally.

Inputs to the filter are the previous state and covariance, the model matrices, the satellite metadata, and the measurement set with their noise parameters. Its outputs are the updated state estimate (including position, velocity, and acceleration), the updated covariance, and the innovation terms, later used for likelihood evaluation.

Algorithm 7

```
function EKF (x, P, Fa, Q, satelites, vel_satelites, TOAs, FOAs, AOAs,
measure, SIGMA_vec)
    # Prediction step
    x_pred = Fa * x
    P_pred = Fa * P * Fa' + Q

    # Update step (if measurements available)
    if (enough_measurements_available)
        x_upd = x_pred
        P_upd = P_pred
    else
        H = compute_jacobian()
        K = P_pred * H' * inv(H * P_pred * H' + R)
        x_upd = x_pred + K * (z - h(x_pred))
        P_upd = (I - K * H) * P_pred
    end

return x_upd, P_upd, innovation, S
```

5.4 Prototype evaluation

The evaluation of the prototype builds upon the results already presented in Deliverable D6.1 [1], where the performance of the IMM-based tracking system was analysed in two representative transoceanic scenarios, namely the North Atlantic Tracks (NAT) and the Europe to South America (EUR/SAM) corridor. In those tests, the tracking filters (EKF, UKF, and H_∞) were shown to provide similar performance, consistently achieving 2D RMS errors below the CRLB. This behaviour is expected, as the filters exploit both measurement data and motion models, and are therefore less sensitive to geometric dilution or instantaneous satellite visibility than pure localisation algorithms. The study also demonstrated that:

- The filters remain operational even with fewer than four satellites in view, unlike closed-form positioning methods.
- Increasing satellite ephemeris error from ~ 1 m to ~ 100 m leads to a proportional increase in RMS error, confirming the robustness and expected sensitivity of the filters.
- The manoeuvre models correctly reflect the simulated dynamics, with the CV model largely dominating in nominal cruise conditions.
- Hybrid measurements (TOA combined with FOA or AOA) do not yield a systematic improvement in positioning accuracy under the tested conditions.

Building on this initial validation, the prototype has been extended to process real ADS-B trajectories obtained from the OpenSky Network. This represents a key step towards more realistic evaluation, since actual flight data introduces irregularities and operational conditions that cannot be fully captured by idealised great-circle simulations. The prototype is now capable of ingesting ADS-B messages, generating the corresponding satellite measurements (TOA, FOA, AOA), and performing multi-target tracking with the IMM framework.

5.4.1 Evaluation with real ADS-B data (OpenSky Network)

To complement the results obtained in D6.1 with simulated trajectories, the prototype has been extended to process real ADS-B data from the OpenSky Network. This step introduces additional realism into the evaluation, since operational flight trajectories exhibit irregularities and variations that cannot be captured by idealised great-circle simulations.

The evaluation process consists of four main stages. First, raw ADS-B messages are retrieved from the OpenSky dataset [5] and pre-processed to select representative aircraft trajectories (see Figure 18 and Figure 19). Second, these trajectories are embedded into a simulated satellite scenario, where each aircraft transmission is mapped to the satellites in view. For every received message, synthetic measurements of TOA, FOA and AOA are generated. The resulting dataset is stored in the InputData file, which replicates the information flow expected in the operational CPS, including the ADS-B message, TOA, FOA and AOA (horizontal) measurements and satellite data (e.g. ephemeris, velocity, etc.) (see Figure 20), to which a nominal communication network delay of 250 ms is added (still pending a realistic communication network simulation).

states_2017-06-05-00.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
								states2017060500								
	time	icao24	lat	lon	velocity	heading	vertrate	callsign	onground	alert	spi	squawk	baroaltitude	geoaltitude	lastposupda...	lastcontact
	Number	Text	Number	Number	Number	Number	Number	Text	Categorical	Categorical	Categorical	Number	Number	Number	Number	Number
1	time	icao24	lat	lon	velocity	heading	vertrate	callsign	onground	alert	spi	squawk	baroaltitude	geoaltitude	lastposupd...	lastcontact
2	1496620800	4bccb9	52.084915638	11.4453935...	175.320556...	305.321044...		SXS2WY	False	False	False	1000	7459.98	7581.9	1496620800.0	1496620800.0
3	1496620800	502cb2	49.6373748...	2.87184062...	232.93635757	305.199349...	-0.32512	MON55BR	False	False	False	2770	10988.04	11033.76	1496620797...	1496620800.0
4	1496620800	4bccaf	49.8504638...	12.5227832...	196.684570...	288.764648...		SXS7R	False	False	False	3215	11582.4	11772.9	1496620800.0	1496620800.0
5	1496620800	4008e1	50.7808470...	9.262611866	214.205322...	292.747192...		TCX229	False	False	False	3462	10363.2		1496620800.0	1496620800.0
6	1496620800	4070e3	50.7229924...	3.82369995...	221.822621...	297.634765...	0.32512	EXS14D	False	False	False	7775	10972.8	11033.76	1496620797...	1496620800.0
7	1496620800	44d1cc	50.1474165...	11.2171912...	242.690673...	94.8669433...	0.0	TAY289S	False	False	False	7174	10668.0		1496620800.0	1496620800.0
8	1496620800	c04081	51.4994001...	7.58133888...	222.683105...	310.127563...	-293.37	TFL230	False	False	False	1000	11414.76	11452.86	1496620800.0	1496620800.0
9	1496620800	405f09	49.6737670...	-1.5797906...	225.925273...	10.3631924...	-8.45312	EZY12VM	False	False	False	1056	9669.78	9685.02	1496620798...	1496620800.0
10	1496620800	34310d	51.8420577...	12.5282979...	218.726806...	37.353515625	681.99	FAH5102	False	False	False	3611	5433.06		1496620800.0	1496620800.0
11	1496620800	4bccab	49.9548768...	12.0416164...	204.484130...	288.479003...	0.0	SXS4XY	False	False	False	3207	11582.4	11719.56	1496620800.0	1496620800.0
12	1496620800	400bdb	50.7147992...	-2.7101898...	175.305426...	13.7507864...	-7.80288	EZY53TU	False	False	False	7370	4023.36	4046.22	1496620798...	1496620800.0
13	1496620800	406696	50.7993846...	4.70260620...	216.956960...	296.929600...	0.0	TOM66A	False	False	False	7771	11574.78	11650.98	1496620797...	1496620800.0

Figure 18. Example of data from OpenSky Network.

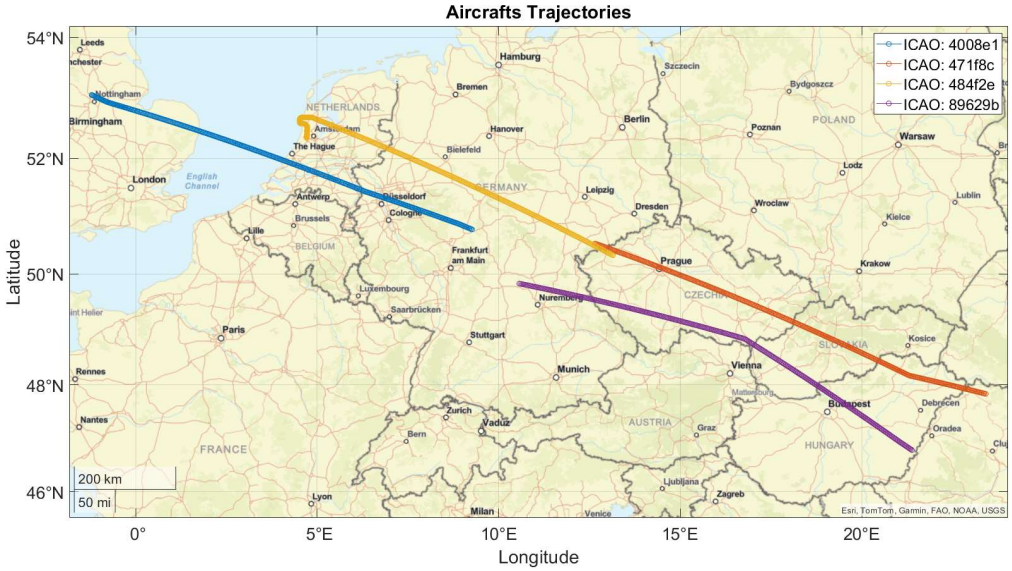

Figure 19. Selected aircraft trajectories.

InputData.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
																		InputData									
	CPS_Time	CPS_Unix_T...	SAT_ID	ToA	FoA	AoAH	Unix_ti...	ICAO	Callsign	Latitude	Longi...	Geoalt...	baro...	ongro...	velo...	hea...	vert...	lastp...	last...	SAT_Positio...	SAT_Positio...	SAT_Positio...	SAT_Veloci...	SAT_Veloci...	SAT_Veloci...	SAT_Attitu...	SAT_Attitu...
	Datetime	Number	Num...	Number	Nu...	Num...	Num...	Num...	Nu...	Nu...	Nu...	Nu...	Nu...	Cat...	N...	N...	Nu...	Nu...	N...	Number	Number	Number	Number	Number	Number	Number	Number	Number
1	CPS_Time	CPS_Unix_T...	SAT_ID	ToA	FoA	AoA-H	Unix_t...	ICAO	Callsi...	Latitu...	Long...	Geoal...	baro...	ongr...	velo...	he...	vert...	lastp...	las...	SAT_Positio...	SAT_Positio...	SAT_Positio...	SAT_Velocit...	SAT_Velocit...	SAT_Velocit...	SAT_Attitu...	SAT_Attitu...
2	05-Jun-201...	1496620800...	108	0.00210...	15643....	275.18...	14966...	4008e1	TCX2...	50.780...	9.262...	NaN	1036...	False	214....	29...	NaN	1496...	14...	3983415.09...	773934.898...	5544733.53...	5870.02236...	1777.42273...	-4465.2001...	0.94197653...	0.19189851...
3	05-Jun-201...	1496620800...	108	0.00222...	17311....	91.707...	14966...	484f2e	CND...	50.329...	13.14...	12382.5	12192	False	222....	29...	-19.05	1496...	14...	3983415.09...	773934.898...	5544733.53...	5870.02236...	1777.42273...	-4465.2001...	0.94197653...	0.19189851...
4	05-Jun-201...	1496620800...	108	0.00230...	18708....	348.62...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	3983415.09...	773934.898...	5544733.53...	5870.02236...	1777.42273...	-4465.2001...	0.94197653...	0.19189851...
5	05-Jun-201...	1496620800...	108	0.00230...	18121....	347.31...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	3983415.09...	773934.898...	5544733.53...	5870.02236...	1777.42273...	-4465.2001...	0.94197653...	0.19189851...
6	05-Jun-201...	1496620800...	128	0.00433...	3450.9...	293.19...	14966...	484f2e	CND...	50.329...	13.14...	12382.5	12192	False	222....	29...	-19.05	1496...	14...	3546643.56...	1965810.00...	5546850.58...	5035.51794...	3505.51286...	-4462.0562...	0.94249743...	0.50611817...
7	05-Jun-201...	1496620800...	88	0.00434...	7883.7...	92.712...	14966...	4008e1	TCX2...	50.780...	9.262...	NaN	1036...	False	214....	29...	NaN	1496...	14...	4030499.29...	-494575.61...	5542662.84...	6129.67318...	-123.96466...	-4468.4214...	0.94146739...	-0.1220978...
8	05-Jun-201...	1496620800...	88	0.00475...	7760.9...	89.154...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	4030499.29...	-494575.61...	5542662.84...	6129.67318...	-123.96466...	-4468.4214...	0.94146739...	-0.1220978...
9	05-Jun-201...	1496620800...	88	0.00475...	7086.1...	87.957...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	4030499.29...	-494575.61...	5542662.84...	6129.67318...	-123.96466...	-4468.4214...	0.94146739...	-0.1220978...
10	05-Jun-201...	1496620800...	128	0.00494...	4809.2...	293.24...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	3546643.56...	1965810.00...	5546850.58...	5035.51794...	3505.51286...	-4462.0562...	0.94249743...	0.50611817...
11	05-Jun-201...	1496620800...	128	0.00494...	3720.6...	293.56...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	3546643.56...	1965810.00...	5546850.58...	5035.51794...	3505.51286...	-4462.0562...	0.94249743...	0.50611817...
12	05-Jun-201...	1496620800...	128	0.00507...	295.58...	288.93...	14966...	4008e1	TCX2...	50.780...	9.262...	NaN	1036...	False	214....	29...	NaN	1496...	14...	3546643.56...	1965810.00...	5546850.58...	5035.51794...	3505.51286...	-4462.0562...	0.94249743...	0.50611817...
13	05-Jun-201...	1496620800...	88	0.00522...	5803.5...	92.732...	14966...	484f2e	CND...	50.329...	13.14...	12382.5	12192	False	222....	29...	-19.05	1496...	14...	4030499.29...	-494575.61...	5542662.84...	6129.67318...	-123.96466...	-4468.4214...	0.94146739...	-0.1220978...
14	05-Jun-201...	1496620800...	109	0.00560...	-2385....	203.38...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	5409096.45...	1312520.25...	4028598.66...	4260.24894...	1307.14048...	-6145.9946...	0.62946814...	0.23804966...
15	05-Jun-201...	1496620800...	109	0.00560...	-2434....	202.91...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	5409096.45...	1312520.25...	4028598.66...	4260.24894...	1307.14048...	-6145.9946...	0.62946814...	0.23804966...
16	05-Jun-201...	1496620800...	109	0.00571...	-2510....	194.79...	14966...	484f2e	CND...	50.329...	13.14...	12382.5	12192	False	222....	29...	-19.05	1496...	14...	5409096.45...	1312520.25...	4028598.66...	4260.24894...	1307.14048...	-6145.9946...	0.62946814...	0.23804966...
17	05-Jun-201...	1496620800...	109	0.00600...	-2431....	204.26...	14966...	4008e1	TCX2...	50.780...	9.262...	NaN	1036...	False	214....	29...	NaN	1496...	14...	5409096.45...	1312520.25...	4028598.66...	4260.24894...	1307.14048...	-6145.9946...	0.62946814...	0.23804966...
18	05-Jun-201...	1496620800...	89	0.00692...	-2095....	142.69...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	5552045.02...	-422994.35...	4025731.33...	4452.48112...	-73.819352...	-6148.3489...	0.62895210...	-0.0760402...
19	05-Jun-201...	1496620800...	89	0.00692...	-2059....	143.19...	14966...	89629b	ETD946	49.830...	10.56...	10736....	1058...	False	258....	10...	371....	1496...	14...	5552045.02...	-422994.35...	4025731.33...	4452.48112...	-73.819352...	-6148.3489...	0.62895210...	-0.0760402...
20	05-Jun-201...	1496620800...	89	0.00698...	-2173....	142.74...	14966...	4008e1	TCX2...	50.780...	9.262...	NaN	1036...	False	214....	29...	NaN	1496...	14...	5552045.02...	-422994.35...	4025731.33...	4452.48112...	-73.819352...	-6148.3489...	0.62895210...	-0.0760402...
21	05-Jun-201...	1496620810...	108	0.00197...	12771....	252.03...	14966...	4008e1	TCX2...	50.788...	9.234...	NaN	1036...	False	214....	29...	NaN	1496...	14...	4041883.35...	791618.732...	5499741.78...	5823.50725...	1759.31834...	-4533.0576...	0.93099186...	0.19340579...
22	05-Jun-201...	1496620810...	108	0.00207...	16227....	117.14...	14966...	484f2e	CND...	50.338...	13.11...	12374....	12192	False	222....	29...	0	1496...	14...	4041883.35...	791618.732...	5499741.78...	5823.50725...	1759.31834...	-4533.0576...	0.93099186...	0.19340579...
23	05-Jun-201...	1496620810...	108	0.00214...	18137....	298.12...	14966...	89629b	ETD946	49.825...	10.60...	10812....	1065...	False	257....	10...	409....	1496...	14...	4041883.35...	791618.732...	5499741.78...	5823.50725...	1759.31834...	-4533.0576...	0.93099186...	0.19340579...

Figure 20. Example of expected information arriving to the CPS.

In the third stage, the InputData file is processed by the MTT logic implemented in the CPS. Since the current simulations are performed offline, the messages are grouped by aircraft ICAO address and timestamp of emission, ensuring that each trajectory is handled independently. Finally, the IMM-based tracking framework (with EKF, UKF, or H_∞ filters) is applied to produce state estimates for each aircraft, which are then compared against the reference ADS-B trajectory to obtain the RMS error.

This setup allows assessing the behaviour of the different tracking filters under realistic flight conditions, while maintaining a controlled and repeatable evaluation framework.

5.5 Simulation and new results

The following results present the 2D (horizontal) RMS position errors obtained for the four trajectories described earlier, simulated within the same satellite scenario defined in Scenario description 4.4.1.1. As in previous analyses, the results are compared against the corresponding CRLB.

Figure 21 shows the number of satellites available along each trajectory. It can be observed that the number of visible satellites remains equal to or greater than four, which is the minimum required for positioning based on TDOA measurements.

Figure 22 compares the tracking performance of the different filters (EKF, UKF, and H_∞) across the same trajectories. At a first glance, the 2D RMS error remains close to the CRLB, with typical values in the order of a few tens of metres. Despite being derived from real flight data, the three filters exhibit very similar behaviour, producing nearly overlapping performance curves.

This observation is confirmed in Table 1, which provides a statistical summary of the horizontal error distributions. Each column in the table offers a complementary view: the **mean** captures the overall tendency but is sensitive to outliers; the **median** offers a more robust measure of central tendency; the **standard deviation** quantifies variability; and the **25th and 75th percentiles** define the interquartile range, showing how tightly most of the estimates are distributed.

Overall, the results confirm that all three filters achieve comparable tracking accuracy, with mean RMS errors around 11 m and similar variability across the tested flights. The UKF provides slightly better performance in most cases, with mean and median errors reduced by approximately $\sim 1-2\%$.

Furthermore, the IMM framework was tested with different motion models (Constant Velocity, Constant Acceleration, and Singer). No clear dominance was observed among them, as their mode probabilities fluctuated over time without a consistent trend. This behaviour is likely related to the relatively low sampling rate of the available ADS-B data (10 s), which limits the filter's ability to capture short-term manoeuvres.

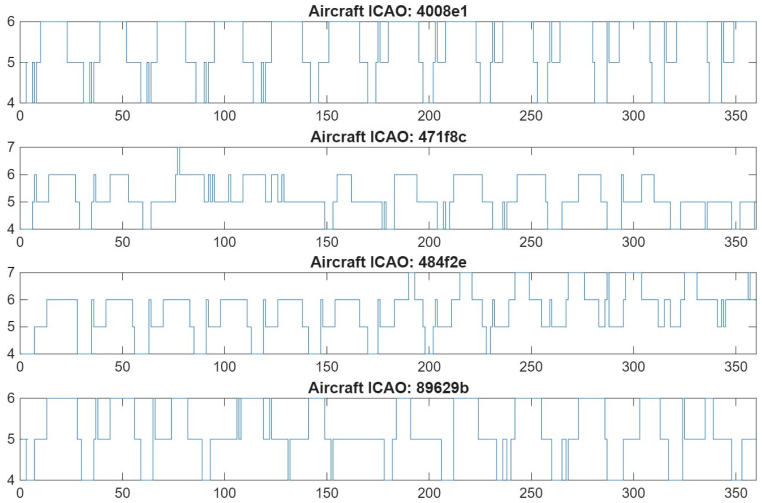

Figure 21. Number of available satellites along each trajectory.

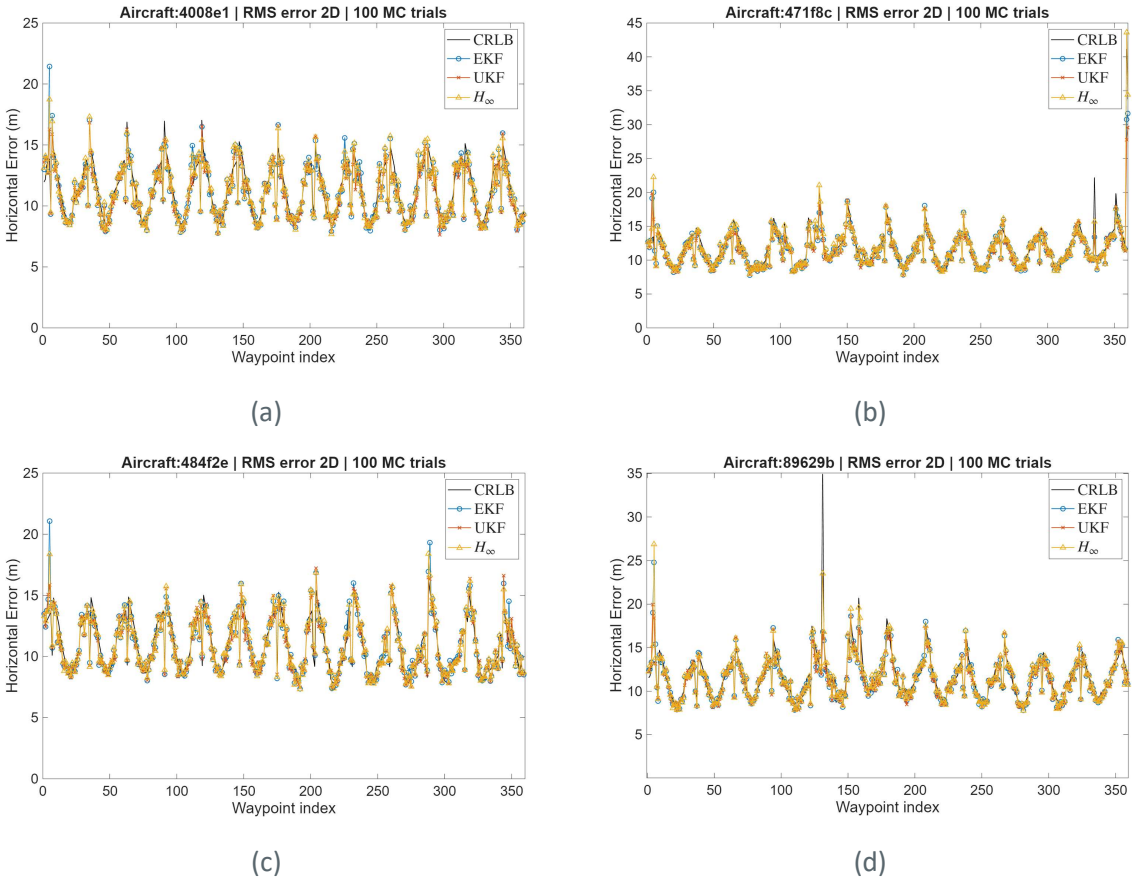

Figure 22. Horizontal RMS error for each trajectory.

ICAO ID	Filter	Mean	Median	Std. Dev.	25th Petl.	75th Petl.
4008e1	EKF	11.2090	11.0178	2.1784	9.2684	12.9099
	UKF	11.0603	10.7787	2.0422	9.2449	12.6982
	H_∞	11.2106	11.0189	2.1236	9.3532	12.9176
471f8c	EKF	11.6560	11.3377	2.6611	9.7338	13.1169
	UKF	11.5656	11.2523	2.5338	9.6572	13.0332
	H_∞	11.7814	11.4848	3.0937	9.7377	13.1908
484f2e	EKF	10.9475	10.4056	2.2447	9.1260	12.6106
	UKF	10.8672	10.3885	2.1577	9.0999	12.5948
	H_∞	10.9604	10.4415	2.1937	9.2200	12.7207
89629b	EKF	11.4828	11.3284	2.3471	9.6353	12.7965
	UKF	11.4022	11.1953	2.2364	9.6348	12.6603
	H_∞	11.5735	11.3839	2.4825	9.6483	12.9621

Table 1. Statistical error metrics (in meters) for each trajectory.

5.5.1 Evaluation with hybrid measurements

Additionally, hybrid measurements were evaluated using the EKF to analyse whether the inclusion of Frequency of Arrival (FDOA) or AOA information could improve tracking accuracy. Figure 23 compares the resulting 2D RMS errors for Time Difference of Arrival (TDOA) only, TDOA+FDOA, and TDOA+AOA configurations, and Table 2 summarises their statistical indicators.

As observed previously in the simulation results reported in D6.1, the inclusion of hybrid measurements does not lead to a significant improvement in accuracy. The RMS errors and their distributions remain practically identical across all configurations, suggesting that under the tested geometry and measurement noise conditions, the contribution of FDOA and AOA is negligible compared to the dominant TDOA component.

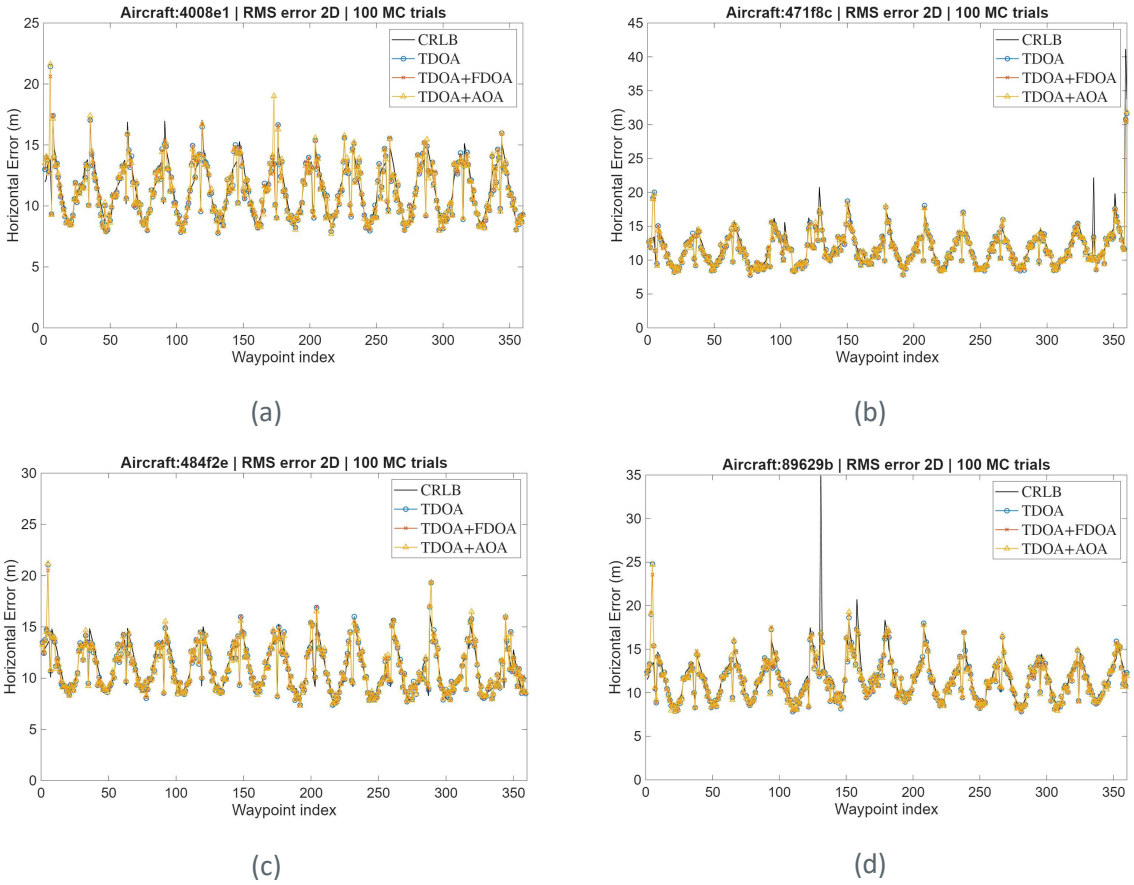

Figure 23. Horizontal RMS error with hybrid measurements for each trajectory.

ICAO ID	Measurements	Mean	Median	Std. Dev.	25th Petl.	75th Petl.
4008e1	TDOA	11.2090	11.0178	2.1784	9.2684	12.9099
	TDOA+FDOA	11.2044	11.0467	2.1732	9.2408	12.9065
	TDOA+AOA	11.2246	10.9208	2.2152	9.3168	12.9244
471f8c	TDOA	11.6560	11.3377	2.6611	9.7338	13.1169
	TDOA+FDOA	11.6461	11.3627	2.6243	9.7687	13.1098
	TDOA+AOA	11.6425	11.2794	2.6518	9.7648	13.0253
484f2e	TDOA	10.9475	10.4056	2.2447	9.1260	12.6106
	TDOA+FDOA	10.9507	10.3885	2.2251	9.1455	12.6237
	TDOA+AOA	10.9564	10.3633	2.2315	9.1385	12.6904
89629b	TDOA	11.4828	11.3284	2.3471	9.6353	12.7965
	TDOA+FDOA	11.4775	11.3431	2.3230	9.6173	12.8062
	TDOA+AOA	11.4853	11.3499	2.3540	9.6114	12.7748

Table 2. Statistical error metrics (in meters) with hybrid measurements for each trajectory

5.6 Conclusions

The results presented confirm the same trends previously identified in D6.1. No significant differences are observed among the three evaluated filters, as clearly shown in the statistical analysis, and the inclusion of hybrid measurements provides only a marginal benefit under the current conditions. Overall, the MTT framework implemented within the CPS demonstrates stable and consistent tracking performance, which will be further refined and extended in the next phases of the project, once the communication network is completed.

6 Integrity indicator

6.1 Introduction

As discussed in D6.1 [1], ensuring the accuracy and reliability of aircraft position data is of paramount importance for maintaining the safety, efficiency, and robustness of modern air traffic management systems. With the increasing reliance on automated surveillance technologies and data-driven decision-making in aviation, even minor discrepancies in position information can lead to significant operational risks or degraded system performance. Therefore, developing advanced methods to assess and ensure the integrity of aircraft positioning data has become a critical objective in the evolution of air traffic surveillance infrastructures.

In this context, this section recalls the design and implementation of a comprehensive integrity monitoring framework tailored to enhance situational awareness for both air traffic controllers and automated systems. The goal is to proactively detect, quantify, and isolate anomalies or inconsistencies in aircraft position estimates, thereby mitigating the potential consequences of erroneous data on flight operations and surveillance decision-making processes.

A top-level view of the operational diagram is shown in Figure 24.

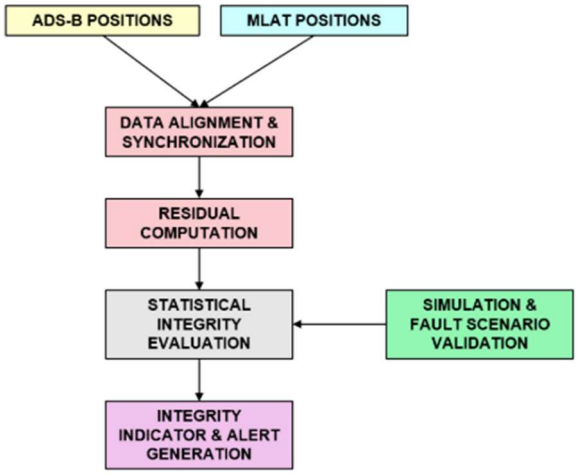

Figure 24: Operational diagram overview.

More specifically, the framework will statistically assess the integrity of aircraft position information using ADS-B data as the primary input. The integrity monitoring framework evaluates the consistency of the filtered ADS-B-derived positions, using statistical tests to detect anomalies that may indicate sensor degradation, data corruption, communication errors, or system faults.

A key feature of the proposed approach is its reliance on comparative analysis between ADS-B position data and GNSS-independent MLAT estimates.

ADS-B data, while offering high temporal resolution and wide coverage, can be affected by both internal and external error sources, such as GNSS spoofing or transmission delays. In contrast, MLAT systems provide position estimates based on the TOA (in the SATERA case also on additional measurements) of aircraft signals received by a set of stations (satellites), which do not depend on satellite navigation systems.

By comparing these two independent position sources, the integrity framework can identify discrepancies that may indicate an integrity issue. Therefore, the integration of these methodologies aims to deliver a more resilient surveillance system capable of identifying erroneous inputs before they compromise the broader traffic management process.

The integrity indicator proposed in [1] will thus play an important role in enhancing trust in surveillance data and will support the transition toward more safety-critical applications in future airspace operations. An overview of the detailed operational diagram is shown in Figure 25.

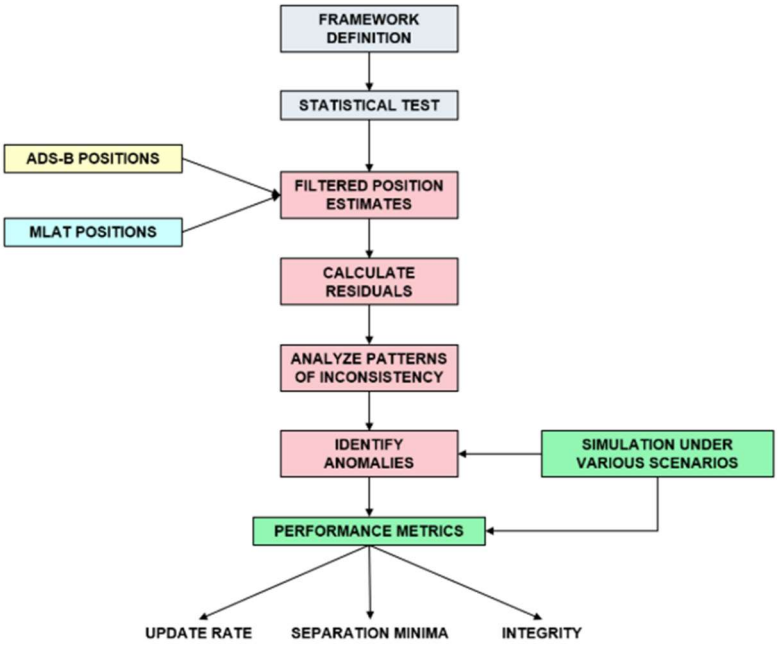

Figure 25: Detailed system workflow highlighting the output of key performance indicators.

This workflow encompasses the entire process from defining the integrity indicator framework to identifying and evaluating performance metrics within the SATERA system. It begins with the establishment of a structured framework that outlines the integrity indicators relevant for assessing the reliability and consistency of surveillance data.

Both ADS-B and MLAT position data are fed into a filtering stage. In this stage, residuals for each surveillance source are calculated—these residuals represent the differences between expected and observed values. The purpose of this calculation is to uncover any discrepancies between the ADS-B and MLAT data streams, enabling a detailed analysis of potential inconsistencies or mismatches in aircraft positioning.

Following this, any anomalies detected in the data are examined in relation to their effect on system performance. These anomalies are systematically linked to specific performance metrics, allowing for a quantifiable understanding of how data inconsistencies may degrade surveillance integrity.

To ensure comprehensive validation, the workflow incorporates simulations conducted across a broad range of scenarios, including both nominal (fault-free) conditions and those involving various types of faults. These simulated scenarios serve as a basis for testing the system's robustness and are used to train and refine anomaly detection mechanisms throughout the process.

Overall, this workflow defines the architecture of the integrity monitoring system and details each stage involved in the evaluation and monitoring of performance metrics. It provides a foundation for ensuring that the surveillance system maintains a high level of integrity, even under degraded or anomalous operating conditions. Further details about the theory were discussed in D6.1 [1].

6.2 Composite surveillance and System level integrity check

6.2.1 Introduction

Composite MLAT–ADS-B surveillance is a technique that integrates position data from two independent sources—MLAT and ADS-B—to form a unified and more reliable aircraft surveillance output. Each system has distinct characteristics: ADS-B relies on onboard GNSS-derived position reports transmitted by the aircraft, while MLAT uses TDOA measurements from ground-based sensors to independently compute an aircraft's position. By combining the two, the system benefits from the strengths of both and increases surveillance robustness.

The same concept can be used in SATERA, using E-MLAT measurements. The core concept behind this composite approach is to compare the ADS-B position with the independently calculated E-MLAT position, within a predefined time frame and spatial tolerance.

If the two positions agree within acceptable bounds, the surveillance system can be confident in the integrity of the reported aircraft position. This cross-verification helps detect anomalies, such as undetected GNSS faults in ADS-B or data anomalies in E-MLAT increasing the overall system's ability to identify errors that might go unnoticed if only one surveillance method were used.

Further theoretical details were discussed in D6.1 [1].

Note that SATERA propose the use of the AWIC algorithm initially considered by EUROCAE WG-51 SG-5 when drafting early versions of ED-142A **[6].**

6.3 Algorithm prototype

The integrity monitoring algorithms implemented in this section use synthetic ADS-B and MLAT measurements over a representative trajectory. The methodology builds upon deterministic and statistical bounds, ensuring robust detection of anomalies while controlling false alarm rates.

6.3.1 Algorithm Description

First, we describe the inputs and outputs that are used for the simulation of the AWIC detection algorithm.

Inputs:

- $traj$: selected trajectory having T points (New York → Madrid or São Paulo → Milan)
- data files: 'ny_madrid.mat', 'saopaulo_milan.mat'
- $\sigma_{H,ADSB}$: horizontal ADS-B position error (m)
- $\sigma_{H,MLAT}$: horizontal MLAT position error (m)
- N_{MC}: number of Montecarlo runs
- OP: observation period (s)
- $P_{FA,pfh}$: probability of false alarm per flight hour
- P_{MD}: probability of missed detection
- τ: ADS-B latency (s)
- v_{nom}: nominal aircraft speed (m/s)
- a_{max} : maximum horizontal acceleration (m/s²)

Outputs:

- δ_r: horizontal deviation between MLAT and ADS-B
- γ: adaptive alarm threshold
- β: containment bound
- A: alarm events (boolean per point)
- Figures:
    - Trajectory (true vs noisy)
    - δ_r vs γ with alarm markings
    - number of alarms per MC run

Algorithm Steps:

Here, the sequence of steps is outlined that are used to load real or synthetic flight trajectory data, add measurement noise and biases to simulate ADS-B and MLAT errors, compute residuals and adaptive thresholds, detect alarms, and visualize results.

1. Load and prepare the trajectory

- Select data file according to $traj$.
- Extract true latitude, longitude, and height arrays.
- Combine into LLA_{coord} ($T \times 3$ matrix).
- Convert full trajectory to East North Up (ENU) coordinates relative to the first point (ENU_{coord}).

2. Define measurement noise and system parameters

- Set standard deviations $\sigma_{H,ADSB}$ and $\sigma_{H,MLAT}$.

- Compute combined measurement uncertainty $\sigma_{res} = \sqrt{\sigma_{H,ADSB}^2 + \sigma_{H,MLAT}^2}$.

3. Initialize Montecarlo simulation

- Allocate matrices to store residuals, thresholds, containment bounds, and alarms for all T samples and N_{MC} runs.

4. Loop over Montecarlo runs

For each run:

- Generate Gaussian random noise for MLAT and ADS-B horizontal components.
- Optionally add bias or "spoofing/jamming" effects (e.g., double ADS-B noise, constant offset).
- Compute noisy positions in ENU coordinates:
- $MLAT_{ENU} = ENU_{coord} + noise_{MLAT}$
- $ADSB_{ENU} = ENU_{coord} + noise_{ADSB} + Bias$

5. Compute detection parameters

- From $P_{FA,pfh}$, OP, and P_{MD}, derive constants:
- P_{FA} , k_R, k_M using Gaussian approximations.
- These govern statistical thresholds for detection.

6. Model systematic biases

- BM: fixed MLAT bias (e.g., 5 m).
- $speed = v_{nom} \pm 10\%$ random variation.
- $BA = speed \times \tau$ (speed-dependent ADS-B latency bias).
- Introduce time-jitter $dt_{eff} = OP + random\ error$.
- $BE = 0.5 \times a_{max} * dt_{eff}^2$ (time extrapolation bias).
- $BT = BM + BA + BE$ (total expected bias).

7. Compute deviations

- $res_E = MLAT_{ENU,east} - ADSB_{ENU,east}$
- $res_N = MLAT_{ENU,north} - ADSB_{ENU,north}$
- $\delta r = \sqrt{res_E^2 + res_N^2}$ for each trajectory point.

8. Compute thresholds and containment bounds

- $\gamma = (k_R \times \sigma_{res}) + BT$
- $CB = 2 \times BT + (k_R + k_M) \times \sigma_{res}$.

9. Evaluate alarms

Co-funded by the European Union

- Mark points where $\delta_r > \gamma$.
- Store alarms per point and Montecarlo run.
- Count total alarms and identify which trajectory segments trigger detections.

10. Aggregate Montecarlo statistics

- Compute mean deviations $\langle \delta_r \rangle$, mean thresholds $\langle \gamma \rangle$, mean containment bounds $\langle CB \rangle$.
- Determine fraction of alarms per trajectory point and per run.

11. Visualize results

- Plot full trajectory: true, ADS-B, and MLAT paths.
- Highlight zoom regions with deviations.
- Bar chart of number of alarms in each MC run.
- Mean of alarms, considering all MC runs.

6.3.2 Evaluation

The prototype evaluation considers several scenarios (nominal and faulty) for a trajectory in the NAT corridor between New York and Madrid.

In this situation, there is the true trajectory, the ADS-B-derived trajectory, and position estimates by means of an MLAT approach. To evaluate the performance of the integrity monitoring algorithm, both the ADS-B and MLAT estimates are slightly perturbed by injecting correlated Gaussian noise into the latitude and longitude coordinates of the perfect trajectory, ensuring realistic spatial correlation between latitude and longitude errors, and then added to the corresponding coordinates of the true positions. Then, correlated Gaussian noise is generated for each time step in the trajectory, which is added to the latitude and longitude components of the ADS-B and MLAT positions.

With respect to the biases, BM (MLAT-derived bias), BA (ADS-B-derived bias), and BE (extrapolation-derived bias), which all combined produce BT (total bias), the following values have been used:

- **BM -** MLAT residual bias due to system calibration.
- **BA -** uncompensated ADS-B latency bound.
- **BE -** extrapolation/time-alignment bias due to aircraft acceleration. Typical values for aircraft range from 3m to 10m.

However, for the purposes of these simulations and this phase of validation, it is assumed that systematic errors can be neglected. Consequently, the bias term, BT, is set to zero when computing the detection threshold. In other words, any persistent offset or bias between the ADS-B and MLAT measurements is disregarded at this stage, without affecting the validation of the algorithm.

The objective is to evaluate the performance of the detection algorithm under idealized conditions, where only random noise and stochastic variations are considered. Therefore, no systematic differences between the ADS-B and MLAT trajectories are introduced or corrected for in this part of the analysis. In WP7, the value of BT will be defined according to the system simulation and included in the threshold computation.

6.3.2.1 Example Trajectory: New York - Madrid (NAT corridor)

6.3.2.1.1 Nominal scenario

The New York – Madrid trajectory consists of 5400 points. The full trajectory is displayed on the left side of Figure 26, while two zoomed-in sections on the right highlight specific regions of the trajectory, illustrating the deviations and errors between the true path and the trajectories derived from ADS-B and MLAT measurements. All simulation results below are based on 1000 Montecarlo runs, used to assess trajectory deviations, thresholds, and alarms for ADS-B and MLAT measurements under different noise and bias conditions. The standard deviations for the MLAT and ADS-B errors used in the simulation are respectively 10m and 5m.

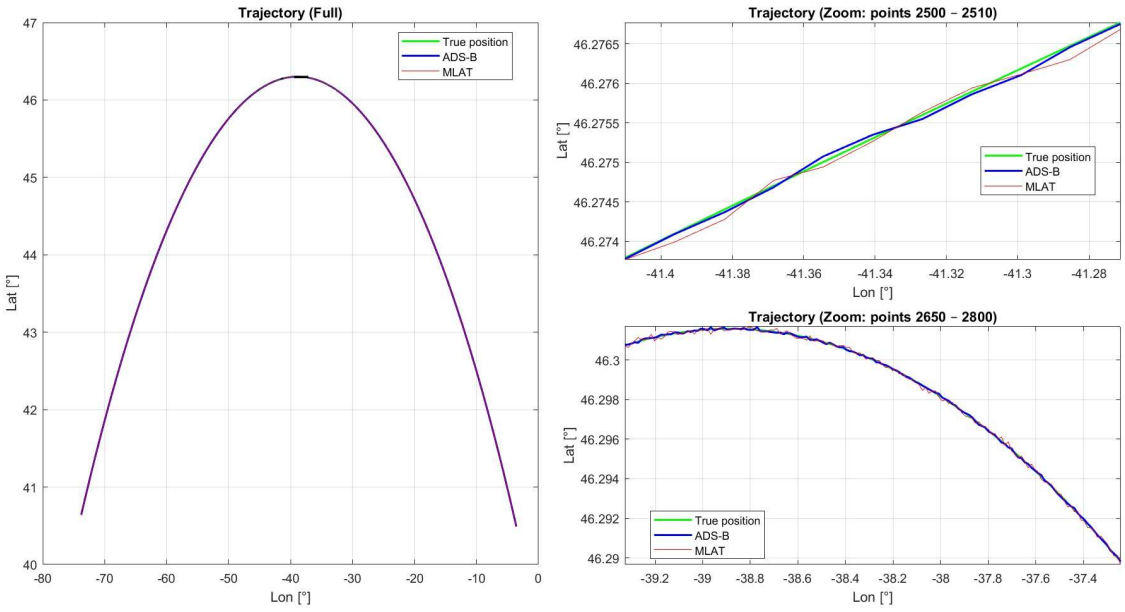

Figure 26: Full trajectory (New York – Madrid) and zoomed-in regions of the trajectory.

For a nominal situation, a Probability of False Alarm (PFA) (per flight hour) of 2e-3, which results in a PFA of approximately 6.1e-4, is considered. An alarm is triggered whenever the distance, or deviation, between an ADS-B point and a MLAT point (δ_r) is greater than the AWIC threshold found, γ.

The mean of δr and γ are 14.01 m and 43.02 m, respectively. In this test, δ_r surpassed γ 3329 times out of 5400000 (result of analysing 5400 trajectory points for 1000 MC runs). The resulting PFA, 6.1648e-04 (derived from 3329/5400000), aligns very well with the expected value of 6.1e-4.

The number of alarms that triggered every MC run are shown in Figure 27:

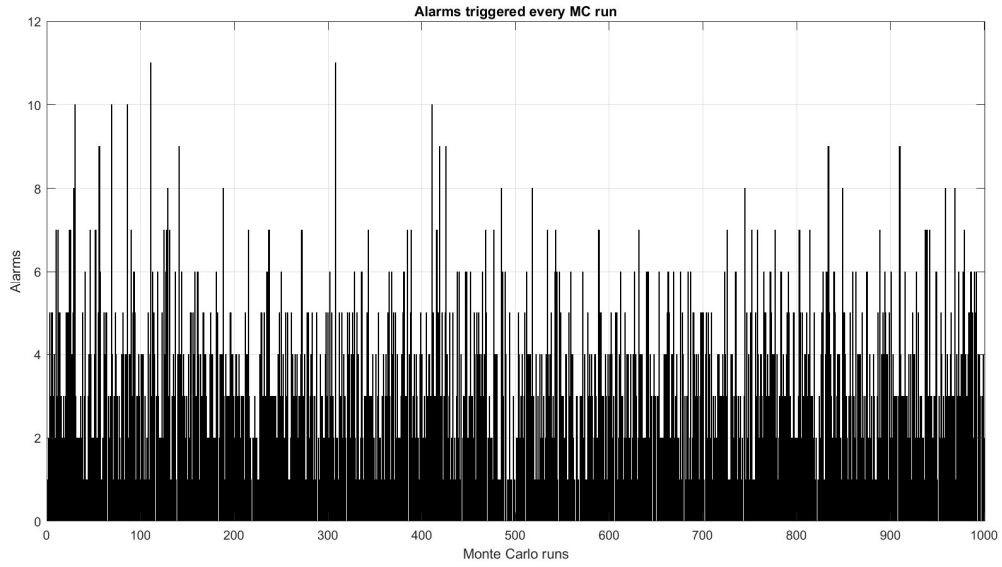

Figure 27: Number of alarms for the nominal, faultless scenario.

As an example, MC run 241 triggered 3 alarms. The ADS-B/MLAT deviation, the AWIC Threshold, the AWIC Containment Bound, and the alarms raised are shown in Figure 28:

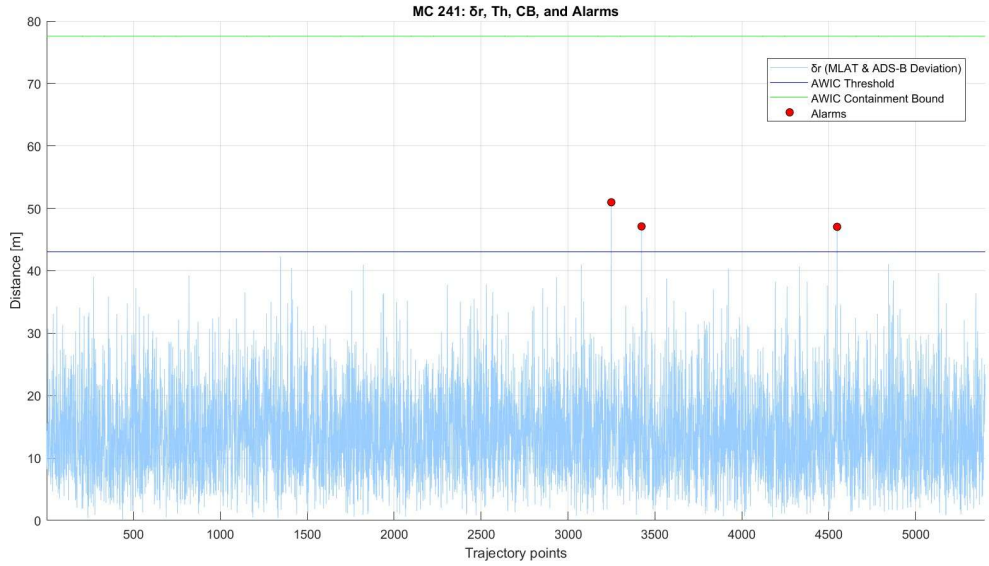

Figure 28: Alarms triggered during Montecarlo run 241.

6.3.2.1.2 Faulty scenario

The following cases have been proposed to test the (ADS-B) faulty scenario:

Case 1. Faulty ADS-B caused by jamming:

A higher level of noise injection is achieved adding an additive noise, by scaling the ADS-B noise by a multiplication factor (tested with values 1 and 2), as shown below, where 0 is replaced with 1 or 2 to simulate different jamming intensities:

No jamming: noiseADSB2 = 0 * noiseADSB;
Jamming 1: noiseADSB2 = 1 * noiseADSB (Figure 29);
Jamming 2: noiseADSB2 = 2 * noiseADSB (Figure 30);
Jamming 3: noiseADSB2 = 3 * noiseADSB (Figure 31).

This results in increased noise in the ADS-B data: adsb_ENU = perfect_ENU + [noiseADSB + noiseADSB2].

Jamming 1:

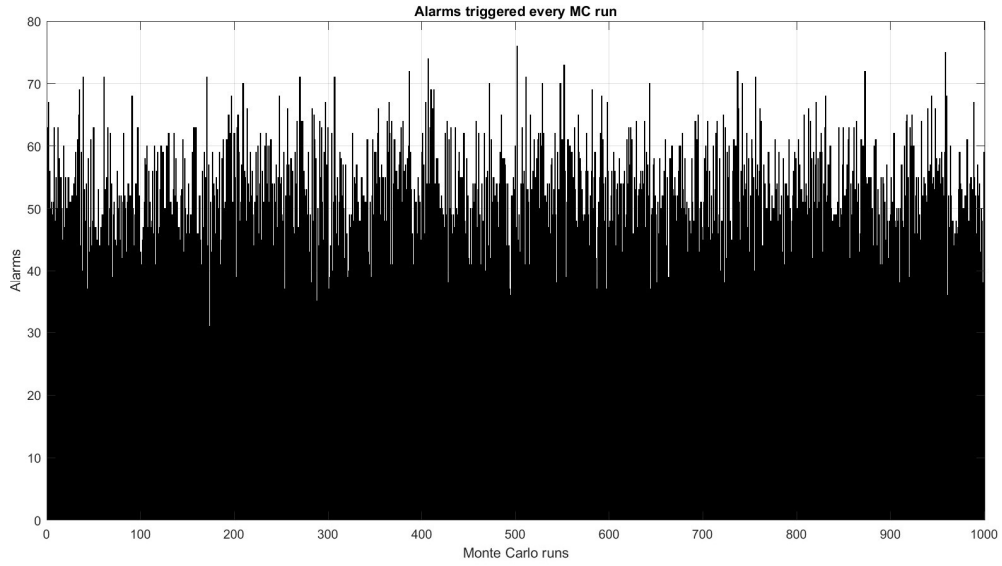

Figure 29: Number of alarms for the Jamming 1 scenario.

The mean of δr and Th are 17.72 m and 43.02 m, respectively. Alarms: 53051.

Jamming 2:

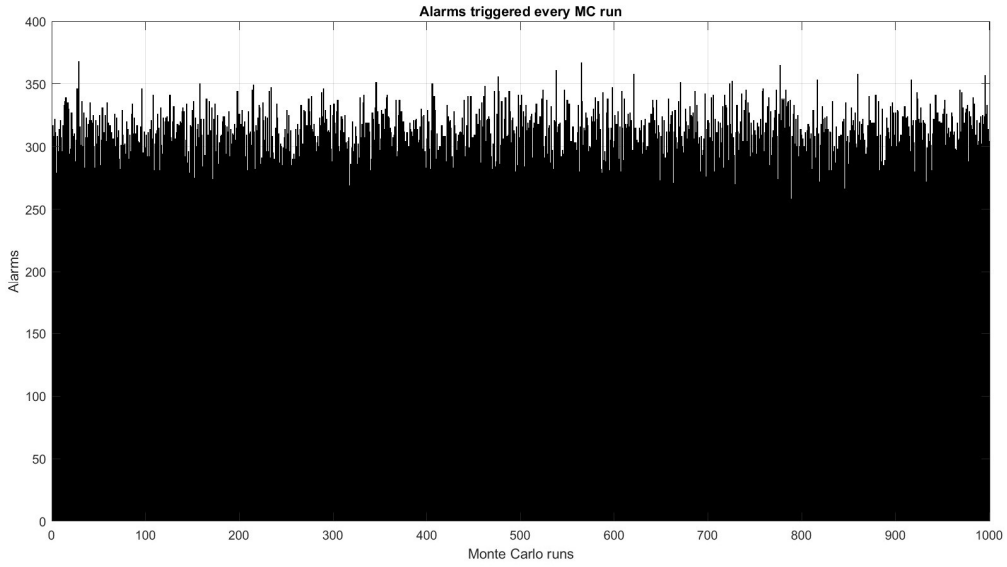

Figure 30: Number of alarms for the Jamming 2 scenario.

The mean of δr and Th are 22.59 m and 43.02 m, respectively. Alarms: 313122.

Jamming 3:

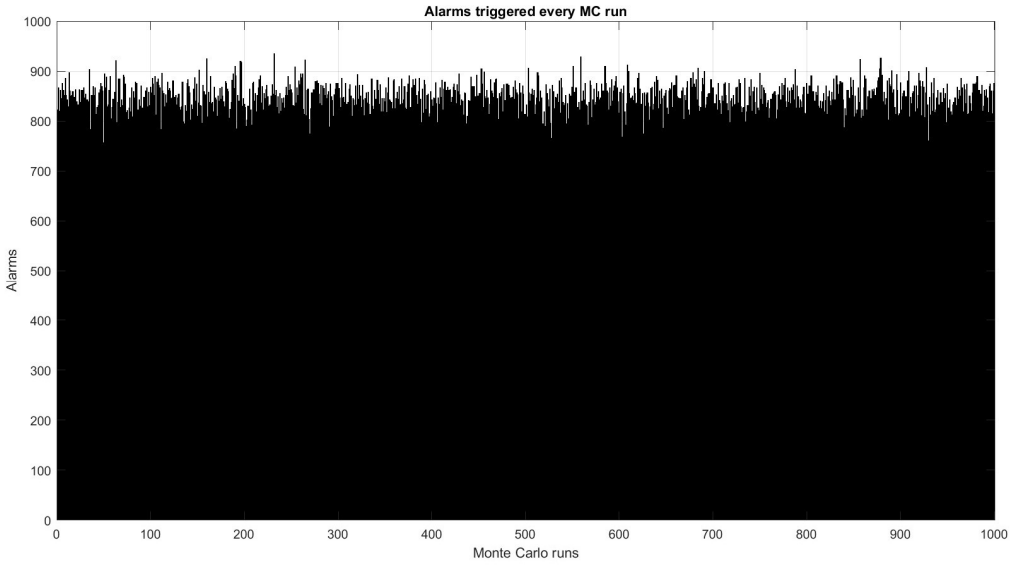

Figure 31: Number of alarms for the Jamming 3 scenario.

The mean of δr and Th are 28.03 m and 43.02 m, respectively. Alarms: 848765.

Case 2. Faulty ADS-B caused by spoofing:

Spoofing is simulated by adding a fixed two-dimensional offset (in meters) to the trajectory coordinates. Three cases are used for testing and validation purposes, with offsets of 10 m, 25 m, and 50 m in both East and North coordinates:

No bias: `bias_ENU = [0, 0];`

Spoofing 1: `bias_ENU = [10, 10]` (Figure 32);

Spoofing 2: `bias_ENU = [25, 25]` (Figure 33);

Spoofing 3: `bias_ENU = [50, 50]` (Figure 34);

Spoofing 1:

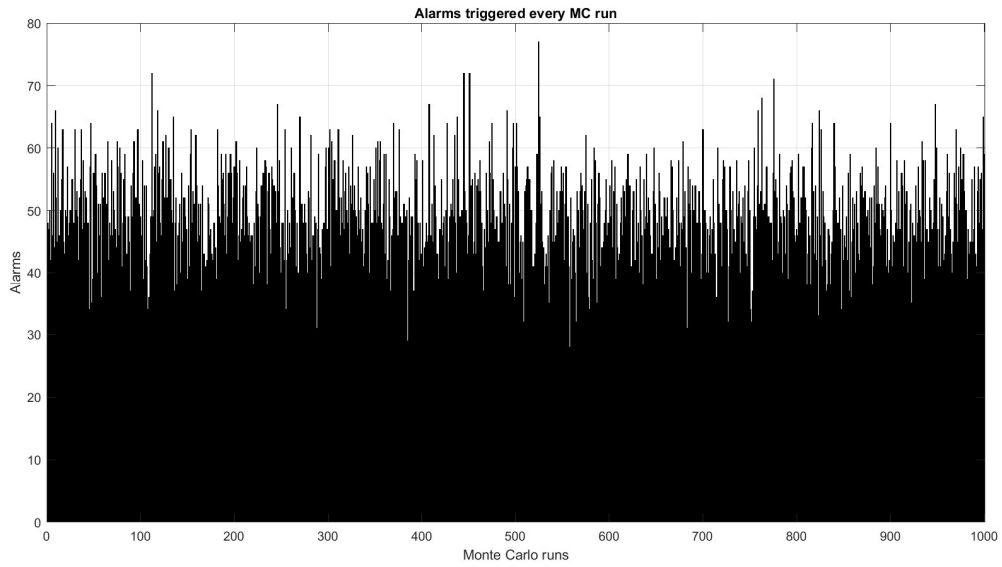

Figure 32: Number of alarms for the Spoofing 1 scenario.

The mean of δr and Th are 19.12 m and 43.02 m, respectively. Alarms: 49185.

Spoofing 2:

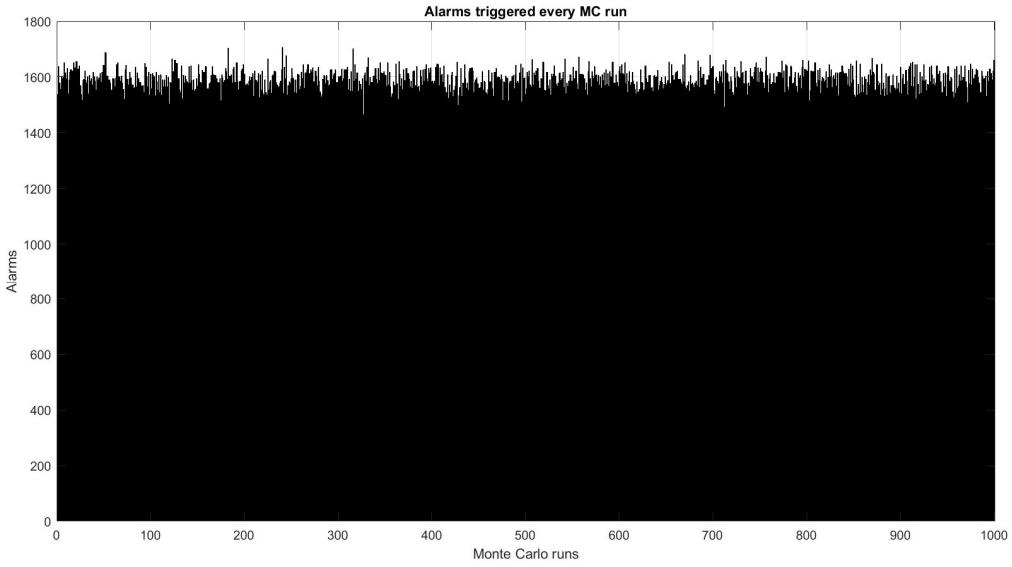

Figure 33: Number of alarms for the Spoofing 2 scenario.

The mean of δr and Th are 37.19 m and 43.02 m, respectively. Alarms: 1591373.

Spoofing 3:

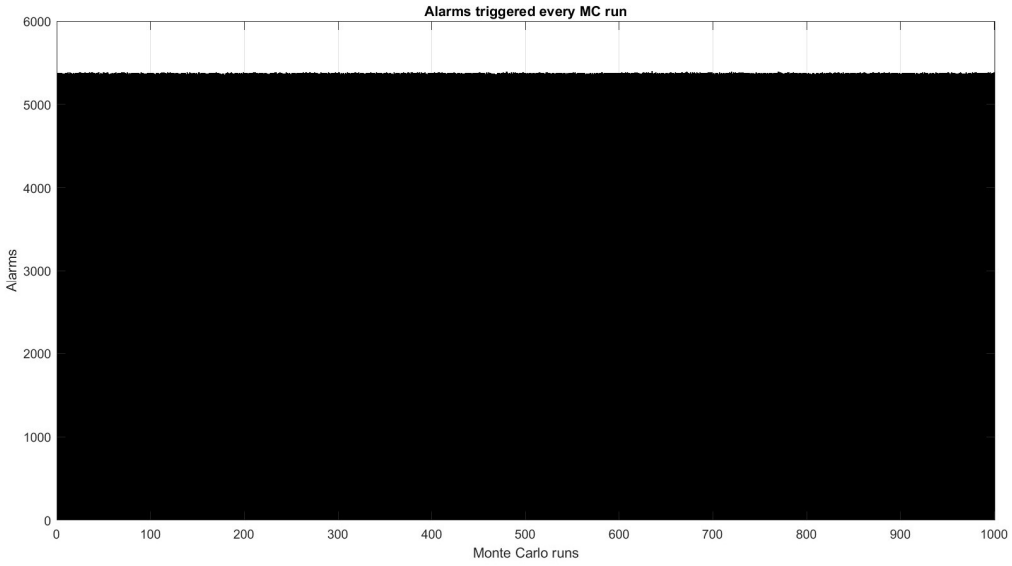

Figure 34: Number of alarms for the Spoofing 3 scenario.

The mean of δr and Th are 71.61 m and 43.02 m, respectively. Alarms: 5373253.

Key Points

A series of scenarios were evaluated to assess system performance under varying conditions, ranging from nominal operation without biases or faults to cases where deliberate errors were introduced into the ADS-B positions to simulate jamming and spoofing. For the faulty scenarios, three independent simulation runs were conducted for each case to ensure the robustness of the results.

Case 0. Nominal scenario

Under nominal conditions, with no biases or faults present, 3329 alarms were triggered across 1000 Montecarlo simulation runs. This outcome aligns with the expected nominal behaviour and is consistent with the defined probability of false alarms, confirming the system's reliability under normal operating conditions.

Case 1. Faulty ADS-B caused by jamming

- **Case 1.1:** When noise equal in magnitude to the initial system noise was injected, **53051** alarms were triggered.
- **Case 1.2:** When the injected noise was doubled relative to the initial noise, the number of triggered alarms increased substantially to **313122**.
- **Case 1.3:** When the injected noise was tripled relative to the initial noise, the number of triggered alarms increased substantially to **848765**.

These results illustrate the sensitivity of the system to increasing levels of jamming interference and demonstrate a nonlinear escalation in alarm frequency as the noise magnitude grows.

Case 2. Faulty ADS-B caused by spoofing

- **Case 2.1:** For an injected bias of 10 meters in both the East and North directions, **49185** alarms were triggered.
- **Case 2.2:** For an injected bias of 25 meters in both the East and North directions, **1591373** alarms were triggered.
- **Case 2.3:** For an injected bias of 50 meters in both the East and North directions, **5373253** alarms were triggered.

These findings indicate that even relatively small positional biases caused by spoofing can significantly affect the system, with alarm frequency rising sharply as the magnitude of the spoofing bias increases.

Note how an attacker introducing a bias of 50m is detected almost 100% of the times. This is well below the ATC requirements for the en-route application (350m for horizontal error) [7].

Table 3 summarizes the results obtained for the different simulated scenarios, all of them considering 1000 Montecarlo runs. Since the trajectory is composed of 5400 points, the total number of analysis points for each simulation is 5400000. Thus, the alarms triggered are referenced to this upper limit.

Case	Alarms Triggered	Alarms / Total Trials
Nominal: Faultless scenario	3329	6.16e-4
Jamming 1: Total Noise: 2x ADS-B Noise	53051	0.0098
Jamming 2: Total Noise: 3x ADS-B Noise	313122	0.058
Jamming 3: Total Noise: 3x ADS-B Noise	848765	0.1572
Spoofing 1: East, North bias of 10 m	49185	0.0091
Spoofing 2: East, North bias of 25 m	1591373	0.2947
Spoofing 3: East, North bias of 50 m	5373253	0.995

Table 3. Alarms triggered for the nominal and faulty scenarios.

Co-funded by the European Union

7 References

[1] SATERA Consortium. “Space-based MLAT algorithms description,” SATERA Deliverable 6.1, 2025.

[2] Vo, Ba-ngu, et al. "Multitarget tracking." Wiley encyclopedia of electrical and electronics engineering 2015 (2015).

[3] Blackman, Samuel S. "Multiple-target tracking with radar applications." Dedham (1986).

[4] SATERA Consortium. “CRLB for MLAT,” SATERA Deliverable 5.2, 2025.

[5] OpenSky. https://opensky-network.org/data/scientific.

[6] EUROCAE. (draft). Technical specification for a wide area multilateration ground system with composite surveillance functionality working version v0.0 (ED-142A). European Organisation for Civil Aviation Equipment.

[7] EUROCAE. “Technical Specification for Wide Area Multilateration (WAM) Systems,” EUROCAE, ED-142, September 2010.

8 List of acronyms

Acronym	Description
ADS-B	Automatic Dependent Surveillance – Broadcast
AOA	Angle of Arrival
ATC	Air Traffic Control
ATM	Air Traffic Management
BA	Uncompensated ADS-B latency bound
BE	Extrapolation/time-alignment bias
BM	MLAT residual bias
BT	Total bias
CF	Closed Form
CNS	Communication, Navigation and Surveillance
CPS	Central Processing Station
CRLB	Cramér Rao Lower Bound
EASA	European Union Aviation Safety Agency
ECEF	Earth-Centered Earth-Fixed
EKF	Extended Kalman Filter
EKF-IMM	Extended Kalman Filter - Interacting Multiple Model
E-MLAT	Enhanced Multilateration
ENU	East North Up
EUR/SAM	Europe to South America
FDOA	Frequency Difference Of Arrival
FOA	Frequency of Arrival
ICAO	International Civil Aviation Organization
IMM	Interacting Multiple Model
KF	Kalman Filter
LEO	Low Earth Orbit
MC	Montecarlo
MLAT	Multilateration
MTT	Multi Target Tracking
NAT	North Atlantic

OF	Open Form
PFA	Probability of False Alarm
RM	Rotaation Matrix
RMS	Root Mean Square
SVD	Singular Value Decomposition
TDOA	Time Difference of Arrival
TOA	Time of Arrival
UKF	Unscented Kalman Filter
UNOOSA	United Nations Office for Outer Space Affairs
UTC	Coordinated Universal Time
WLS	Weighted Least Squares

Table 4: List of acronyms

9 Appendix (MATLAB code)

9.1 Localisation algorithm

The provided code is composed of the following functions:

- `computeInitialGuess`: calculates the initial guess for the aircraft position estimation using either the centroid or the Bancroft algorithm.
- `taylorWLS`: runs the WLS algorithm and if needed restarts using Tikhonov regularisation.
- `getHN`: computes Jacobian (H) and error covariance matrix (N) given theta (state vector) and errors.
- `taylorWLSTikhonov`: runs the WLS algorithm using Tikhonov regularisation.
- `getExactMeasurements`: calculates the exact measurements (in absence of noise) observed in each satellite.

9.1.1 computeInitialGuess

The inputs and outputs are described in Section 4.3.1.

Algorithm 8: computeInitialGuess

```
function [startingPoint] = computeInitialGuess( ...
    initialGuessAlgo, satPositions, aircraftAlt, measurements)
%{
--------------------------------------------------------------------------
Description: Bancroft's algorithm to estimate aircraft position
Paper: SOLVING PASSIVE MULTILATERATION EQUATIONS USING BANCROFT'S ALGORITHM
--------------------------------------------------------------------------
- INPUTS
    initialGuessAlgo - string, can be either "bancroft" or "centroid"
    satPositions - satellites positions in ECEF coordinates
    aircraftAlt - aircraft altitude
    measurements - Time Of Arrival measurements
- OUTPUTS:
     point - estimation of the aircraft position
--------------------------------------------------------------------------
%}
% if number of visible satellites is lower than 4, use centroid anyway
if length(satPositions) < 4 & initialGuessAlgo == "bancroft"
    % fprintf("only %d visible satellites, using centroid for initial " + ...
    %     "guess\n", length(satPositions));
    initialGuessAlgo = "centroid";
```

```
end
switch initialGuessAlgo
    case "bancroft"
        startingPoint = bancroft( ...
            satPositions, aircraftAlt, measurements);
    case "centroid"
        startingPoint = lla2ecef(getCentroid(satPositions, aircraftAlt))';
    otherwise
        fprintf('initial guess algo "%s" not supported', ...
            initialGuessAlgo);
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [point] = getCentroid(satPos, altitude)
%{
---------------------------------------------------------------------------
Description: simple script to find average point between satellites in view
of the aircraft. Horizontally, the centroid is considered (average over lat
and lon); vertically, the typical aircraft altitude is used.
---------------------------------------------------------------------------
- INPUTS
    satPos - satellites positions in ECEF coordinates
    altitude - aircraft altitude
- OUTPUTS:
     point - calculated central point [lat, lon, alt]
---------------------------------------------------------------------------
%}
satPos2lla = ecef2lla(satPos);
centroid = mean(satPos2lla, 1);
point = [centroid(1) centroid(2) altitude];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [point] = bancroft(satPos, altitude, TOAi)
%TODO refactor variable names
%{
---------------------------------------------------------------------------
Description: Bancroft's algorithm to estimate aircraft position
Paper: SOLVING PASSIVE MULTILATERATION EQUATIONS USING BANCROFT’S ALGORITHM
---------------------------------------------------------------------------
- INPUTS
    sat_pos - satellites positions in ECEF coordinates
    altitude - aircraft altitude
```

```
    TOAi - Time Of Arrival measurements
- OUTPUTS:
     point - estimation of the aircraft position
--------------------------------------------------------------------------
%}
c0 = physconst("Lightspeed");

AlMatrix = [satPos(:,1) satPos(:,2)  satPos(:,3) -c0*TOAi];
bVector  = (satPos(:,1)).^2+(satPos(:,2)).^2+(satPos(:,3)).^2 - ...
    c0^(2).*TOAi.^2;

AlMatrixPseudo = pinv(AlMatrix);
dVector = 0.5*AlMatrixPseudo*ones(length(AlMatrix),1);
eVector = 0.5*AlMatrixPseudo*bVector;

alpha = dVector(1)^2+dVector(2)^2+dVector(3)^2-dVector(4)^2;
beta = 2*eVector(1)*dVector(1)+2*eVector(2)*dVector(2)+ ...
    2*eVector(3)*dVector(3)-2*eVector(4)*dVector(4)-1;
gamma = eVector(1)^2+eVector(2)^2+eVector(3)^2-eVector(4)^2;

delta = roots([alpha beta gamma]);

saHat1 = delta(1)*dVector+eVector;
saHat1 = saHat1(1:end-1);
saHat2 = delta(2)*dVector+eVector;
saHat2 = saHat2(1:end-1);

pointINTEREST = lla2ecef(getCentroid(satPos, altitude));
saHat1 = real(saHat1);
saHat2 = real(saHat2);

% choose one of the 2 solutions by taking the one closer to the centroid
dist1 = norm(saHat1 - pointINTEREST);
dist2 = norm(saHat2 - pointINTEREST);
if dist1 < dist2
    point = saHat1;
else
    point = saHat2;
end

end
```

9.1.2 taylorWLS

The inputs and outputs are described in Section 4.3.2.

Algorithm 9: taylorWLS

```
function [thetaCur,condH,condA] = taylorWLS(startingPoint, maxIters, measurements, ...
    errors, type, satPos, satVel, f0)
%{
---------------------------------------------------------------------------
Description: Taylor (LS) algorithm to estimate aircraft position
Paper: Statistical Theory of Passive Location Systems (DON J. TORRIERI)
---------------------------------------------------------------------------
- INPUTS
    startingPoint - aircraft initial guess point in ECEF coordinates
    satPos - satellites positions in ECEF coordinates
    measurements - satellites measurements
    maxIters - maximum number of iterations
    errors - measurement errors standard deviations
    type - string indicating measurement type ["toa", "toa_foa_aoa"]
    f0 - nominal frequency
    satVel - satellites velocities in ECEF coordinates
- OUTPUTS:
    thetaCur - estimation of the aircraft position using Taylor method
    condH - condition number of the Jacobian at each iteration
    condA - condition number of the matrix A at each iteration
---------------------------------------------------------------------------
%}
condH = [];
condA = [];
startCond = 0;
thetaCur = startingPoint;
% start Taylor iterations
for k = 1:maxIters
    [H, N] = getHN(type, thetaCur, errors, f0, satPos, satVel);
    % check cond number
    if k == 1
        % initialise on first iteration
        startCond = cond(H);
    elseif cond(H) > startCond * 1.1
        % if cond number grew more than 10%, restart Taylor adding
        % Tikhonov regularisation
        [thetaCur,condH,condA] = taylorWLSTikhonov( ...
            startingPoint, maxIters, measurements, ...
            errors, type, satPos, satVel, f0);
        return
    end
```

```
    % extract data from theta depending on type
    switch type
        case "toa"
            airVel = [];
            freqBias = 0;
            clockBias = thetaCur(4);
        case "foa"
            airVel = thetaCur(4:6)';
            freqBias = thetaCur(7);
            clockBias = 0;
        case "aoa"
            airVel = [];
            freqBias = 0;
            clockBias = 0;
        case "toa_aoa"
            airVel = [];
            freqBias = 0;
            clockBias = thetaCur(4);
        case "toa_foa_aoa"
            airVel = thetaCur(4:6)';
            freqBias = thetaCur(8);
            clockBias = thetaCur(7);
        otherwise
            fprintf('taylorWLS: type "%s" not supported', type);
    end
    f = getExactMeasurements(type, thetaCur(1:3)', airVel, clockBias, ...
        freqBias, f0, satPos, satVel);
    % transpose without changing imaginary parts (unlike ')
    Ht = transpose(H);
    mDelta = measurements - f;
    % check cond number of H
    condH = [condH cond(H)];
    % precalculate matrix A to be inverted (Ht*W*H) and check cond number
    A = Ht / N * H;
    condA = [condA cond(A)];
    thetaNext = A^(-1) * Ht / N * mDelta + thetaCur;
    deltaTheta = norm(thetaCur - thetaNext);
    thetaCur = thetaNext;
    %TODO parametrise this value somehow... ?
    if deltaTheta < 1e-4
        break
    end
end
```

```
% return NaN values when the solution has not converged
if deltaTheta >= 1e-4
    thetaCur = nan(length(thetaCur), 1);
end
end
```

9.1.3 getHN

Function called while running the WLS algorithm.

Algorithm 10: getHN

```
function [H,N] = getHN(type, theta, errors, f0, satPos, satVel)
%{
--------------------------------------------------------------------------
Description: Function to compute Jacobian (H) and Cov. matrix (N)
given theta (state vector) and errors
--------------------------------------------------------------------------
- INPUTS
    type - string indicating measurement type ["toa", "toa_foa_aoa"]
    theta - state vector, depends on type
    errors - [sigmaT sigmaF sigmaA] standard deviations for the errors of
        each measurement type, expressed in [s Hz rad]
    f0 - nominal frequency
    satPos - [x1 y1 z1; x2 y2 z2...] ECEF coordinates of satellites
    satVel - [vx1 vy1 vz1; vx2 vy2 vz2...] ECEF velocities of satellites
- OUTPUTS:
    H - Jacobian calculated in given theta for given type
    N - Cov. matrix built with given error standard deviations
--------------------------------------------------------------------------
%}
% C0 = physconst("Lightspeed");
variances = errors.^2;

switch type
    case "toa"
        [H,N] = toaHN(theta, variances, satPos);
    case "foa"
        [H,N] = foaHN(theta, variances, f0, satPos, satVel);
    case "aoa"
        [H,N] = aoaHN(theta, variances, satPos);
    case "toa_aoa"
        [Htoa,Ntoa] = toaHN(theta, variances, satPos);
        [Haoa,Naoa] = aoaHN(theta, variances, satPos);
        N = blkdiag(Ntoa, Naoa);
```

```
        HaoaPad = [Haoa, zeros(height(Haoa),1)];
        H = [Htoa; HaoaPad];
    case "toa_foa_aoa"
        [Htoa,Ntoa] = toaHN(theta, variances, satPos);
        [Hfoa,Nfoa] = foaHN(theta, variances, f0, satPos, satVel);
        [Haoa,Naoa] = aoaHN(theta, variances, satPos);
        N = blkdiag(Ntoa, Nfoa, Naoa);
        HtoaPad = [Htoa(:,1:3), zeros(height(Htoa),3), ...
            Htoa(:,4), zeros(height(Htoa),1)];
        HfoaPad = [Hfoa(:,1:6), zeros(height(Hfoa),1), Hfoa(:,7)];
        HaoaPad = [Haoa, zeros(height(Haoa),5)];
        H = [HtoaPad; HfoaPad; HaoaPad];
    otherwise
        fprintf('getHN: type "%s" not supported\n', type);
end
end

function [H,N] = toaHN(theta, variances, satPos)
    C0 = physconst("Lightspeed");
    % N matrix
    N = diag(variances(:,1));
    % H matrix
    numSat = height(satPos);
    H = zeros(numSat,4);
    for i = 1:numSat
        dx = theta(1) - satPos(i,1);
        dy = theta(2) - satPos(i,2);
        dz = theta(3) - satPos(i,3);
        R = sqrt(dx^2 + dy^2 + dz^2);
        H(i,1) = dx / (C0 * R);
        H(i,2) = dy / (C0 * R);
        H(i,3) = dz / (C0 * R);
        H(i,4) = 1;
    end
end

function [H,N] = foaHN(theta, variances, f0, satPos, satVel)
    C0 = physconst("Lightspeed");
    % N matrix
    N = diag(variances(:,2));
    % H matrix
    numSat = height(satPos);
    H = zeros(numSat,7);
    for i = 1:numSat
```

```
        dx = theta(1) - satPos(i,1);
        dy = theta(2) - satPos(i,2);
        dz = theta(3) - satPos(i,3);
        R = sqrt(dx^2 + dy^2 + dz^2);
        dvx = theta(4) - satVel(i,1);
        dvy = theta(5) - satVel(i,2);
        dvz = theta(6) - satVel(i,3);
        vRel = ...
            dot((satPos(i,:)-theta(1:3)'),(theta(4:6)'-satVel(i,:)))/R;
        H(i,1) = -f0/C0*(dvx/R+vRel*dx/R^2);
        H(i,2) = -f0/C0*(dvy/R+vRel*dy/R^2);
        H(i,3) = -f0/C0*(dvz/R+vRel*dz/R^2);
        H(i,4) = -f0/C0*dx/R;
        H(i,5) = -f0/C0*dy/R;
        H(i,6) = -f0/C0*dz/R;
        H(i,7) = 1;
    end
end

function [H,N] = aoaHN(theta, variances, satPos)
    % N matrix
    N = diag(variances(:,3));
    % H matrix
    numSat = height(satPos);
    H = zeros(numSat,3);
    satLla = ecef2lla([satPos(:,1) satPos(:,2) satPos(:,3)], 'WGS84');
    wgs84 = wgs84Ellipsoid('meter');
    % aircraft to NED
    [xAir,yAir,] = ecef2ned(theta(1), theta(2), theta(3), ...
        satLla(:,1), satLla(:,2), satLla(:,3), wgs84);
    for i = 1:numSat
        H(i,1) = -yAir(i)/(xAir(i)^2 + yAir(i)^2);
        H(i,2) = xAir(i)/(xAir(i)^2 + yAir(i)^2);
        H(i,3) = 0;
        % apply rotation matrix
        dcm = dcmecef2ned(satLla(i,1), satLla(i,2));
        H(i,1:3) = H(i,1:3) * dcm;
    end
end
```

9.1.4 taylorWLSTikhonov

The inputs and outputs are described in Section 4.3.2. Function called while running the WLS algorithm, when regularisation is needed.

Algorithm 11: taylorWLSTikhonov

```
function [thetaCur,condH,condA] = taylorWLSTikhonov( ...
    startingPoint, maxIters, measurements, ...
    errors, type, satPos, satVel, f0)
%{
--------------------------------------------------------------------------
Description: Taylor (LS) algorithm to estimate aircraft position
Paper: Statistical Theory of Passive Location Systems (DON J. TORRIERI)
--------------------------------------------------------------------------
- INPUTS
    startingPoint - aircraft initial guess point in ECEF coordinates
    satPos - satellites positions in ECEF coordinates
    measurements - satellites measurements
    maxIters - maximum number of iterations
    errors - measurement errors standard deviations
    type - string indicating measurement type ["toa", "toa_foa_aoa"]
    f0 - nominal frequency
    satVel - satellites velocities in ECEF coordinates
- OUTPUTS:
    thetaCur - estimation of the aircraft position using Taylor method
    condH - condition number of the Jacobian at each iteration
    condA - condition number of the matrix A at each iteration
--------------------------------------------------------------------------
%}
w = 0.3;

condH = [];
condA = [];
thetaCur = startingPoint;
% start Taylor iterations
for k = 1:maxIters
    [H, N] = getHN(type, thetaCur, errors, f0, satPos, satVel);
    % extract data from theta depending on type
    switch type
        case "toa"
            airVel = [];
            freqBias = 0;
            clockBias = thetaCur(4);
        case "foa"
            airVel = thetaCur(4:6)';
            freqBias = thetaCur(7);
```

```
            clockBias = 0;
        case "aoa"
            airVel = [];
            freqBias = 0;
            clockBias = 0;
        case "toa_aoa"
            airVel = [];
            freqBias = 0;
            clockBias = thetaCur(4);
        case "toa_foa_aoa"
            airVel = thetaCur(4:6)';
            freqBias = thetaCur(8);
            clockBias = thetaCur(7);
        otherwise
            fprintf('taylorWLS: type "%s" not supported', type);
    end
    f = getExactMeasurements(type, thetaCur(1:3)', airVel, clockBias, ...
        freqBias, f0, satPos, satVel);
    % transpose without changing imaginary parts (unlike ')
    Ht = transpose(H);
    mDelta = measurements - f;
    % check cond number of H
    condH = [condH cond(H)];
    % derive lambda
    HS = svd(H);
    lambda = HS(3) + w * (HS(2) - HS(3));
    % precalculate matrix A to be inverted (Ht*W*H) and check cond number
    A = Ht / N * H + lambda^2 * eye(width(H));
    condA = [condA cond(A)];
    thetaNext = A^(-1) * Ht / N * mDelta + thetaCur;
    deltaTheta = norm(thetaCur - thetaNext);
    thetaCur = thetaNext;
    %TODO parametrise this value somehow... ?
    if deltaTheta < 1e-4
        break
    end
end
% return NaN values when the solution has not converged
if deltaTheta >= 1e-4
    thetaCur = nan(length(thetaCur), 1);
end
end
```

9.1.5 getExactMeasurements

Function called while running the WLS algorithm.

Algorithm 12: getExactMeasurements

```
function [measurements] = getExactMeasurements( ...
    type, airPos, airVel, userClockBias, freqBias, f0, satPos, satVel)
%{
function that calculates the exact measurements (in absence of noise)
observed in each satellite

type - string indicating measurement type, can be one of the following:
    - toa
    ...

airPos - [x y z] ECEF coordinates of the aircraft
airVel - [vx vy vz] ECEF velocity of the aircraft
userClockBias - [s]
freqBias - [Hz]
f0 - nominal frequency [Hz]
satPos - [x1 y1 z1; x2 y2 z2...] ECEF coordinates of satellites
satVel - [vx1 vy1 vz1; vx2 vy2 vz2...] velocities of satellites
%}

C0 = physconst("Lightspeed");

measurements = [];
R = sqrt( ...
        (satPos(:,1) - airPos(1)).^2 + ...
        (satPos(:,2) - airPos(2)).^2 + ...
        (satPos(:,3) - airPos(3)).^2 ...
        );
toa = R ./ C0 + userClockBias;
switch type
    case "toa"
        measurements = toa;
    case "foa"
        vRel = dot((satPos - airPos), (airVel - satVel), 2) ./ R;
        foa = f0 .* vRel ./ C0 + freqBias;
        measurements = foa;
    case "aoa"
        wgs84 = wgs84Ellipsoid('meter');
        % satellites to lla
        lla = ecef2lla([satPos(:,1) satPos(:,2) satPos(:,3)], 'WGS84');
```


Co-funded by the European Union

```
        % aircraft to NED
        [xAir,yAir,] = ecef2ned(airPos(1), airPos(2), airPos(3), ...
            lla(:,1), lla(:,2), lla(:,3), wgs84);
        aoa = atan2(yAir, xAir);
        measurements = aoa;
    case "toa_aoa"
        % aoa
        wgs84 = wgs84Ellipsoid('meter');
        % satellites to lla
        lla = ecef2lla([satPos(:,1) satPos(:,2) satPos(:,3)], 'WGS84');
        % aircraft to NED
        [xAir,yAir,] = ecef2ned(airPos(1), airPos(2), airPos(3), ...
            lla(:,1), lla(:,2), lla(:,3), wgs84);
        aoa = atan2(yAir, xAir);
        % merge together
        measurements = [toa; aoa];
    case "toa_foa_aoa"
        % foa
        vRel = dot((satPos - airPos), (airVel - satVel), 2) ./ R;
        foa = f0 .* vRel ./ C0 + freqBias;
        % aoa
        wgs84 = wgs84Ellipsoid('meter');
        % satellites to lla
        lla = ecef2lla([satPos(:,1) satPos(:,2) satPos(:,3)], 'WGS84');
        % aircraft to NED
        [xAir,yAir,] = ecef2ned(airPos(1), airPos(2), airPos(3), ...
            lla(:,1), lla(:,2), lla(:,3), wgs84);
        aoa = atan2(yAir, xAir);
        % merge together
        measurements = [toa; foa; aoa];
    otherwise
        fprintf('getExactMeasurements: type "%s" not supported', type);
end
end
```

9.2 Sequential filtering

The code presented below is composed of the following main functions:

- `position_tracking`: manages the overall target tracking process, including the reading of input data, initialisation of tracks, and update of estimated positions over time.
- `IMM`: implements the Interacting Multiple Model framework, which fuses the outputs of multiple filters running under different motion models to improve robustness and accuracy. It also includes the gaussian pdf function to calculate the likelihood.
- `EKF`: performs the Extended Kalman Filter estimation for a single model. It includes the prediction and update steps, as well as specific subfunctions to compute measurement predictions and Jacobians for TDOA, FDOA and AOA.

For simplicity, only the EKF implementation is included here. The UKF and H_∞ filters are omitted, as they follow analogous structures.

9.2.1 Position tracking

```
function [x_est, cov_matrix, mode_Probability, pos_vel_acc, xp] =
position_tracking(flag, estimation_1, estimation_2, estimation_3, bloque, T,
bloque_anterior, measures, SIGMA_vec)
% POSITION_TRACKING: IMM-based tracking using CV, CA and Singer models.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ipunts:
% - flag: to know if the filter is initialize yet
% - estimation_1, estimation_2, estimation_3: three initial estimations
% - T: sampling time
% - bloque_anterior: previous estimation (used once track is available)
% - measures: to choose which measurements are available
% - SIGMA_vec: cointains the standard deviations considered in the measurement
errors
%
% Outputs:
% - x_est: estimated state vector (x,vx,ax,y,vy,ay,z,vz,az)
% - cov_matrix: covarianza matrix
% - mode_Probability: manouver model probability
% - pos_vel_acc: means of r models (individual state vectors)
% - xp: individual covarianze matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c0 = physconst('Lightspeed');

% Filter values
alpha_temporal= 16;
tau = 1/alpha_temporal;
% Measurements
TOAs = bloque.ToA';
```

```
FOAs = bloque.FoA';
AOAs = bloque.AoA_H';

% Satellites position and velocity
sat_pos = bloque.SAT_Position; % ecef
sat_vel = bloque.SAT_Velocity; %ecef

if flag == 0
% Generate state vector

% Position
x0=estimation_1(1);
y0=estimation_1(2);
z0=estimation_1(3);

% Velocity
vx1 = (estimation_2(1)-estimation_1(1))/T; % m/s
vx2 = (estimation_3(1)-estimation_2(1))/T; % m/s
vx0 = vx2;
vy1 = (estimation_2(2)-estimation_1(2))/T; % m/s
vy2 = (estimation_3(2)-estimation_2(2))/T; % m/s
vy0= vy2;
vz1 = (estimation_2(3)-estimation_1(3))/T; % m/s
vz2 = (estimation_3(3)-estimation_2(3))/T; % m/s
vz0= vz2;

% Aceleration
ax0=(vx2-vx1)/T; % m/s^2
ay0=(vy2-vy1)/T; % m/s^2
az0=(vz2-vz1)/T; % m/s^2

% Initial state
X = [x0; vx0; ax0; y0; vy0; ay0; z0; vz0; az0];
else %if the filter is already initialise
 X = bloque_anterior.Estimated_Position{1};
end


%%%%%%%%%%%% Manouver models %%%%%%%%%%%%%
nFilters= 3; % CV, CA, Singer
nStates = 9; % x,V_x,A_x, y,V_y,A_y, z,V_z,A_z
Q=zeros(nStates,nStates,nFilters); % Initialization of Covariance Matrix
F=zeros(nStates,nStates,nFilters); % Initialization of Transition Matrix

%------- Constant velocity (CV)  -------%
s_a=1; % process noise spectral density
eps_acc = 1e-2; % small variance for acceleration RW (m/s^2)^2

% Process noise covariance matrix

% per-axis 3x3 block for CV with acceleration as random-walk (a_{k+1}=a_k)
Phi3_CV = [1, T, 0;
           0, 1, 0;
```

```
          0, 0, 1];    % note: accel persists (identity) con esto a_k existe como
estado coherente, se mantiene al siguiente paso y tiene algo de incertidumbre
(evitas degeneración de covarianza).

% process noise per axis (use standard CV block for pos/vel + small var for acc)
Q3_CV = s_a^2 * [T^3/3, T^2/2, 0;
                 T^2/2, T,     0;
                 0,     0,     0] ;

Q3_CV(3,3) = Q3_CV(3,3) + eps_acc; % small injection on accel

% build 9x9
Q_MLU = blkdiag(Q3_CV, Q3_CV, Q3_CV);
phi_MLU = blkdiag(Phi3_CV, Phi3_CV, Phi3_CV);

Q(:,:,1) = Q_MLU;
F(:,:,1) = phi_MLU;


%------- Constant acceleration (CA)  -------%

s_j=1; % Variance of the jitter
% Process noise covariance matrix
Q_MLA= s_j^2*...
    [T^5/20, T^4/8, T^3/6, 0, 0, 0, 0, 0, 0;
     T^4/8, T^3/3, T^2/2, 0, 0, 0, 0, 0, 0;
     T^3/6, T^2/2, T, 0, 0, 0, 0, 0, 0;
       0,  0,  0, T^5/20, T^4/8, T^3/6, 0, 0, 0;
       0,  0,  0, T^4/8, T^3/3, T^2/2, 0, 0, 0;
       0, 0, 0, T^3/6, T^2/2, T, 0, 0, 0;
       0,  0,  0, 0,  0,  0, T^5/20, T^4/8, T^3/6;
       0,  0,  0, 0,  0,  0, T^4/8, T^3/3, T^2/2;
       0,  0,  0, 0, 0, 0, T^3/6, T^2/2, T];

 Q(:,:,2)=Q_MLA;

% Transition Matrix
phi_MLA= [1, T, T^2/2, 0, 0, 0, 0, 0, 0;
          0, 1, T, 0, 0, 0, 0, 0, 0;
          0, 0, 1, 0, 0, 0, 0, 0, 0;
          0, 0, 0, 1, T, T^2/2, 0, 0, 0;
          0, 0, 0, 0 ,1, T, 0, 0, 0;
          0, 0, 0, 0 ,0, 1, 0, 0, 0;
          0, 0, 0, 0, 0, 0, 1, T, T^2/2;
          0, 0, 0, 0, 0, 0, 0 ,1, T;
          0, 0, 0, 0, 0, 0, 0 ,0, 1];
F(:,:,2)=phi_MLA;

%------- Singer  -------%

% Varianza del ruido blanco w_k
Amax=19.62;
Pmax=0.1;
P0=0.6;
```

```
varianza_aceleracion_Singer = Amax^2*(1+4*Pmax-P0)/3; %Acceleration total variance

%Process noise covariance matrix

% X
q11=1/(2*alpha_temporal^5)*(1-exp(-
2*T*alpha_temporal)+2*alpha_temporal*T+2*alpha_temporal^3*T^3/3-
2*alpha_temporal^2*T^2-4*alpha_temporal*T*exp(-alpha_temporal*T));
q12=1/(2*alpha_temporal^4)*(exp(-2*T*alpha_temporal)+1-2*exp(-
T*alpha_temporal)+2*alpha_temporal*T*exp(-alpha_temporal*T)-
2*alpha_temporal*T+alpha_temporal^2*T^2);
q13=1/(2*alpha_temporal^3)*(1-exp(-2*T*alpha_temporal)-2*alpha_temporal*T*exp(-
alpha_temporal*T));
q22=1/(2*alpha_temporal^3)*(4*exp(-T*alpha_temporal)-3-exp(-
2*alpha_temporal*T)+2*alpha_temporal*T);
q23=1/(2*alpha_temporal^2)*(exp(-2*T*alpha_temporal)+1-2*exp(-alpha_temporal*T));
q33=1/(2*alpha_temporal)*(1-exp(-2*alpha_temporal*T));
% Y
q44=q11;q45=q12;q46=q13;q55=q22;q56=q23;q66=q33;
% Z
q77=q11;q78=q12;q79=q13;q88=q22;q89=q23;q99=q33;
% Error covariance Q
Q_SIN=2*(1/tau)*varianza_aceleracion_Singer*...
[q11, q12, q13, 0, 0 ,0, 0, 0 ,0;
 q12, q22, q23, 0, 0, 0, 0, 0 ,0;
 q13, q23, q33, 0, 0, 0, 0, 0 ,0;
  0,   0,   0, q44,q45,q46, 0, 0 ,0;
  0,   0,   0, q45,q55,q56, 0, 0 ,0;
  0,   0,   0, q46,q56,q66, 0, 0 ,0;
  0,   0 ,  0,  0,  0 , 0,q77,q78,q79;
  0,   0 ,  0,  0,  0 , 0,q78,q88,q89;
  0,   0 ,  0,  0,  0 , 0,q79,q89,q99];

Q(:,:,3)=Q_SIN;

% Transition Matrix

phi_SIN = [1 T (alpha_temporal*T-1+exp(-alpha_temporal*T))/(alpha_temporal^2) 0 0
0 0 0 0;
           0 1 (1-exp(-alpha_temporal*T))/(alpha_temporal) 0 0 0 0 0 0;
           0 0 exp(-alpha_temporal*T) 0 0 0 0 0 0;
           0 0 0 1 T (alpha_temporal*T-1+exp(-
alpha_temporal*T))/(alpha_temporal^2) 0 0 0;
           0 0 0 0 1 (1-exp(-alpha_temporal*T))/(alpha_temporal) 0 0 0;
           0 0 0 0 0 exp(-alpha_temporal*T) 0 0 0;
           0 0 0 0 0 0 1 T (alpha_temporal*T-1+exp(-
alpha_temporal*T))/(alpha_temporal^2)
           0 0 0 0 0 0 0 1 (1-exp(-alpha_temporal*T))/(alpha_temporal)
           0 0 0 0 0 0 0 0 exp(-alpha_temporal*T)];

F(:,:,3)=phi_SIN;

% Markov chain transition matrix
Transprob=[0.95  0.025  0.025;
```

```
            0.025 0.95 0.025;
            0.025 0.025 0.95];

% Initial probability model
modeProb_init=[1/3;1/3;1/3];


% Initial covariance matrix of state vector

P = diag([1e4,1e2,1e0,1e4,1e2,1e0,1e4,1e2,1e0]);
% Position x (100 m)^2
% Velocity x (10 m/s)^2
% Aceleration x (1 m/s²)^2


% Initialization of manouver models

% Velocidad constante (CV)
XaMLU = X;
P_MLU_init = P;

%%%%%%%%%%%% Aceleración constante (CA) %%%%%%%%%%%

XaMLA = X;
P_MLA_init=P;

%%%%%%%%%%%%%%%%% Singer %%%%%%%%%%%%%%%%%

XaSinger = X;

s_t = 15e-9;                    % seconds (15 ns) (receiver error)
sigma_r = c0 * s_t;             % meters
sigmaR2 = sigma_r^2;            % m^2
sigma_M2 = varianza_aceleracion_Singer; % your precomputed maneuver variance
alpha = alpha_temporal;

% auxiliary terms
exp_at = exp(-alpha * T);

term_P22 = (2 - alpha^2 * T^2 + (2 * alpha^3 * T^3 / 3) - 2 * exp_at - 2 * alpha *
T * exp_at);

P11 = sigmaR2;
P12 = sigmaR2 / T;
P13 = 0;
P22 = 2*sigmaR2 / T^2 + ( sigma_M2 / (alpha^4 * T^2) ) * term_P22;
P23 = ( sigma_M2 / (alpha^2 * T) ) * ( exp_at + alpha * T - 1 );
P33 = sigma_M2;

% Build single-axis 3x3 block
Paxis = [ P11,  P12,  P13;
          P12,  P22,  P23;
          P13,  P23,  P33 ];
```

```
% Compose full 9x9 (x, y, z)
P_SIN_init = blkdiag(Paxis, Paxis, Paxis);



% State vector for each manouver model
X_M=[XaMLU, XaMLA, XaSinger];
% Covariance matrix for each manouver model
xp(:,:,1)= P_MLU_init;
xp(:,:,2)= P_MLA_init;
xp(:,:,3)= P_SIN_init;

% IMM FILTERING
if flag == 0
  [x_est,cov_matrix,mode_Probability,pos_vel_acc,xp] = IMM(modeProb_init,
Transprob, F, Q, X_M, xp, sat_pos, sat_vel, TOAs, FOAs, AOAs, measures,SIGMA_vec);
else
  [x_est,cov_matrix,mode_Probability,pos_vel_acc,xp] =
IMM(bloque_anterior.mode_Probability{1},Transprob,F,Q,bloque_anterior.pos_vel_acc{
1},bloque_anterior.individual_cov_matrix{1},sat_pos,sat_vel, TOAs, FOAs, AOAs,
measures, SIGMA_vec);
end


end
```

9.2.2 IMM

```
function[MM,PP,modeProb,xm,xp] = IMM(modeProb,Transprob, F, Q, xm, xp,
pos_satellites, vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inputs:
% modeProb: weights or likelihood (mode probability)
% Transprob: Transition Probability matrix
% F: evolution matrix r models (transition matrices)
% Q: process noise covariance r models
% xm: means of r models  (state vectors)
% xp: error covariances of r models (covariance matrices)
% pos_satellites: satellite position
% vel_satellites: satellite velocity
% TOAs: TOA values
% FOAs: FOA values
% AOAs: AOA values
% measures: string to select which measures are available
% SIGMA_vec: vector with standar deviation for each measurement used
%           (to generate the measurement noise covariance (R matrix)) and
%            for the satellite positions
% Outputs:
%
```

```
% MM = mean of the model (estimated position)
% PP = covariance of the model (estimated covariance matrix)
% modeProb: mode probability
% xm: means of r models  (state vectors)
% xp: error covariances of r models (covariance matrices)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% getting no. of rows and coloumns for transprob matrix
[r,nFilters] = size(Transprob);
%S_M=zeros(numel(Z),numel(Z),r);

%% 1 - Joint prob calculation

c_j = Transprob*modeProb;

if any(c_j <= 0)
    warning('Some c_j <= 0 in IMM mixing; regularizing small values');
    c_j(c_j <= 0) = 1e-300;
end
c_j_inv = 1 ./ c_j; % r x 1, inverso elemento a elemento
MU_ij = Transprob .* (modeProb * c_j_inv'); % r x r

% Each column j of MU_ij should sum to 1 (sum over i)
if any(abs(sum(MU_ij,1) - 1) > 1e-6)
    warning('MU_ij columns not exactly normalized — renormalizing');
    MU_ij = MU_ij ./ sum(MU_ij,1);
end

%% 2 - Mixing input estimates and covariance matrices

temp_x = xm * MU_ij;

[m, n, d] = size(xp);
Pk_1k_1 = zeros(m, n, d);

for j = 1:d % for each model
    P_mix = zeros(m, n);
    for i = 1:d
        diff_x = xm(:,i) - temp_x(:,j); % \bar{x}_i - \bar{x}_j
        P_mix = P_mix + MU_ij(i,j) * (xp(:,:,i) + diff_x * diff_x');
    end
    % symmetrize
    P_mix = 0.5*(P_mix + P_mix');
    Pk_1k_1(:,:,j) = P_mix; % Guardar la covarianza mixta del modelo j
end

temp_MU = zeros(d,1);


% Filtering
for i = 1:nFilters
```

```
    [xk,Pk,nuk,S] = EKF(temp_x(:,i), Pk_1k_1(:,:,i), F(:,:,i), Q(:,:,i),
pos_satellites, vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec);
    %[xk,Pk,nuk,S] = UKF(temp_x(:,i), Pk_1k_1(:,:,i), F(:,:,i), Q(:,:,i),
pos_satellites, vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec);
    %[xk, Pk, nuk, S] = Hinf(temp_x(:,i), Pk_1k_1(:,:,i), F(:,:,i), Q(:,:,i),
pos_satellites, vel_satellites, TOAs, FOAs, AOAs, measures, SIGMA_vec, gamma);

    xm(:,i) = xk;
    xp(:,:,i) = Pk;


    if ~isnan(nuk)
        assert(all(eig(S) > 0), 'S no es definida positiva');
        temp_MU(i) = max(gauss_pdf(nuk, S), 1e-300);  % evita ceros exactos;
calculating likelihood (nuk: innovation measurement)
    else
        temp_MU(i) = modeProb(i); % Mantener la probabilidad anterior
    end

end


%% 4 - Mode probability update
if ~isnan(nuk)
    modeProb = (temp_MU .* c_j) / (c_j' * temp_MU);
else
    modeProb = modeProb; % explicit (not needed)
end
% enforce non-negativity and normalization
modeProb(modeProb < 0) = 0;
modeProb = modeProb / sum(modeProb);

if ~(abs(sum(modeProb) - 1) < 1e-6)
    assert(abs(sum(modeProb) - 1) < 1e-6, 'Las probabilidades no están
normalizadas');
end
%% 5 - Combined estimation

% Output Estimate
MM = xm*modeProb;

for i= 1:nFilters
     temp_p(:,:,i)=(xp(:,:,i) + (xm(:,i)-MM)*(xm(:,i)-MM)');
end
P = zeros(m*n,d);

P(:) = reshape(temp_p,[m*n d]);
PP = zeros(m,n);
PP(:) = P * modeProb;

assert(isequal(PP, PP'), 'PP is not symmetric');
if any(eig(PP) < 0)
    assert(all(eig(PP) > 0), 'PP is not positive definite);
end
```

```
assert(abs(sum(modeProb) - 1) < 1e-6, 'The probabilities are not normalized');

end


function likelihood = gauss_pdf(nuk,S)
%   Multivariate Gaussian Normal Function
% -------------------------------------
% nuk is Innovation mean in Kalman Filter
% S is Innovation Covariance in Kalman Filter
% likelihood is the normal pdf

    n = length(nuk);
    E = -0.5 * (nuk' / S * nuk);
    likelihood = exp(E) / sqrt((2*pi)^n * det(S));  % PDF multivariante
end
```

9.2.3 EKF

```
function [X_k_k, P_k_k, innovation_measurement, S] = EKF(x, P, Fa, Q,
satelites,vel_satelites,TOAs, FOAs, AOAs, measure,SIGMA_vec)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inputs:
% x: state vector
% P: Covariance matrix
% Fa: Transition matrix
% Q: process noise covariance matrix
% satellites: satellite position
% vel_satellites: satellite velocity
% TOAs: TOA values
% FOAs: FOA values
% AOAs: AOA values
% measures: string to select which measures are available
% SIGMA_vec: vector with standar deviation for each measurement used
%           (to generate the measurement noise covariance (R matrix)) and
%            for the satellite positions
% Outputs:
%
% X_k_k: tracked position
% P_k_k: covariance matrix
% innovation_measurement: nuk (needed for the PDF)
% S: innovation matrix (needed for the PDF)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c0 = physconst('Lightspeed');
sat_errors = [SIGMA_vec(end), SIGMA_vec(end), SIGMA_vec(end)] .*
randn(length(satelites),3);

% Prediction
% -------------------------------------
```

```
% Estado predicho
x_pred = Fa * x;

% Covarianza predicha
P_pred = Fa * P * Fa' + Q;

% Check if enough TOAs for update
if length(TOAs) < 2
    X_k_k = x_pred;
    P_k_k = P_pred;
    innovation_measurement = NaN;
    S = NaN;
    return;
end

% Update
% ---------------------------------------

% Initialization
H_k = [];
z = [];
z_pred = [];
R = [];

% 1) TDOA

% Prediction
z_pred_tdoa = c0*predict_tdoa(x_pred, satelites, c0, sat_errors);
% Jacobian
H_TDOA = c0*calc_jacobiano_tdoa(x_pred, satelites, c0, sat_errors);
% Measurements
z_tdoa = c0*(TOAs(2:end) - TOAs(1))'; % TDOAs relativos
% Innovation covariance
m = length(z_tdoa);
R_tdoa = (c0*SIGMA_vec(1))^2 * (ones(m) + eye(m));  % diagonal 2*sigma^2, off-diag
sigma^2
R = blkdiag(R, R_tdoa);

H_k = [H_k; H_TDOA];
z = [z; z_tdoa];
z_pred = [z_pred; z_pred_tdoa];


% --- TDOA + FDOA ---
if isequal(measure(1), 'tdoa_fdoa')
    f0= 1090e6;
    z_fdoa = (FOAs(2:end) - FOAs(1))';
    z_pred_fdoa = predict_fdoa(x_pred, satelites, vel_satelites, f0, c0);
    [~, ~, H_FDOA, ~] = compute_H_TDOA_FDOA(x_pred(1), x_pred(4), x_pred(7),
x_pred(2), x_pred(5), x_pred(8), satelites, vel_satelites);
    R_fdoa = SIGMA_vec(2)^2 * (ones(length(z_fdoa)) + eye(length(z_fdoa)));
    R = blkdiag(R, R_fdoa);
    H_FDOA_full = zeros(size(H_FDOA,1), length(x_pred));  % size(H_FDOA,2) = 6
    H_FDOA_full(:,[1,4,7,2,5,8]) = H_FDOA; % Map to x,y,z,vx,vy,vz
```

```
    H_k=[H_k; H_FDOA_full];
    z = [z; z_fdoa];
    z_pred = [z_pred; z_pred_fdoa];
end

% --- TDOA + AOA (horizontal) ---
if isequal(measure(1), 'tdoa_aoah')

    z_aoa = AOAs';  % azimuth angles
    z_pred_aoa = predict_aoa_h(x_pred, satelites);
    [~, ~, ~, H_AOA_H] = compute_H_FOA_TOA_AOA(x_pred(1), x_pred(4), x_pred(7),
x_pred(2), x_pred(5), x_pred(8), satelites, vel_satelites);
    R_aoa = (SIGMA_vec(2)^2) .* eye(length(z_aoa));
    R = blkdiag(R, R_aoa);
    H_AOA_H_full = zeros(size(H_AOA_H,1), length(x_pred));
    H_AOA_H_full(:,[1,4,7]) = H_AOA_H;  % AOA depends only on position

    H_k = [H_k; H_AOA_H_full];
    z = [z; z_aoa];
    z_pred = [z_pred; z_pred_aoa];
end

% ---TDOA + FDOA + AOA (horizontal) ---
if isequal(measure(1), 'tdoa_fdoa_aoah')
    f0= 1090e6;
    z_fdoa = (FOAs(2:end) - FOAs(1))';
    z_pred_fdoa = predict_fdoa(x_pred, satelites, vel_satelites, f0, c0);
    [~, ~, H_FDOA, ~] = compute_H_TDOA_FDOA(x_pred(1), x_pred(4), x_pred(7),
x_pred(2), x_pred(5), x_pred(8), satelites, vel_satelites);
    H_FDOA_full = zeros(size(H_FDOA,1), length(x_pred));  % size(H_FDOA,2) = 6
    H_FDOA_full(:,[1,4,7,2,5,8]) = H_FDOA;
    R_fdoa = SIGMA_vec(2)^2 * (ones(length(z_fdoa)) + eye(length(z_fdoa)));
    R = blkdiag(R, R_fdoa);

    z_aoa = AOAs';  % azimuth angles
    z_pred_aoa = predict_aoa_h(x_pred, satelites);
    [~, ~, ~, H_AOA_H] = compute_H_FOA_TOA_AOA(x_pred(1), x_pred(4), x_pred(7),
x_pred(2), x_pred(5), x_pred(8), satelites, vel_satelites);
    R_aoa = (SIGMA_vec(3)^2) .* eye(length(z_aoa));
    R = blkdiag(R, R_aoa);
    H_AOA_H_full = zeros(size(H_AOA_H,1), length(x_pred));
    H_AOA_H_full(:,[1,4,7]) = H_AOA_H;

    H_k=[H_k; H_FDOA_full; H_AOA_H_full];
    z = [z; z_fdoa; z_aoa];
    z_pred = [z_pred; z_pred_fdoa; z_pred_aoa];
end

    S = H_k * P_pred * H_k' + R;
    % numerical hygiene: symmetrize
    S = 0.5 * (S + S');

    % Kalman gain with Cholesky (stable)
    P12 = P_pred * H_k';
```

```
    [R_chol, p] = chol(S);
    if p > 0
        % fallback to pseudo-inverse with warning
        warning('Cholesky failed on S; using pseudo-inverse (numerical risk)');
        K = P12 * pinv(S);
    else
        U = P12 / R_chol; % P12 * inv(R_chol)
        innovation_measurement = z - z_pred;
        X_k_k = x_pred + U * (R_chol' \ innovation_measurement);
        P_k_k = P_pred - U * U';
        P_k_k = 0.5 * (P_k_k + P_k_k');  % symmetrize
        return;
    end

    % If fallback branch reached:
    innovation_measurement = z - z_pred;
    X_k_k = x_pred + K * innovation_measurement;
    P_k_k = P_pred - K * H_k * P_pred;
    P_k_k = 0.5 * (P_k_k + P_k_k');
end


function z_pred = predict_tdoa(x, satelites, c0, errores_sat)
    % Predict TDOAs
    pos = [x(1); x(4); x(7)];  % aircraft position
    pos_sat = satelites + errores_sat; % satellites with error (3xN)
    diff = pos - pos_sat';               % difference pos - sat (3xN)
    ranges = sqrt(sum(diff.^2, 1));     % norm column by column (1xN)
    z_pred = (ranges(2:end) - ranges(1))' / c0; % column
end

function H = calc_jacobiano_tdoa(x, satelites, c0, errores_sat)
    % TDOA jacobian
    pos = [x(1); x(4); x(7)];
    pos_sat = satelites + errores_sat; % 3xN
    diff = pos - pos_sat';              % 3xN
    d = sqrt(sum(diff.^2, 1));         % 1xN

    J = zeros(length(d), numel(x));
    J(:,[1,4,7]) = (diff ./ d)';       % Nx3

    H = (J(2:end,:) - J(1,:)) / c0;    % TDOA w.r.t. 1st sat
end

function z_pred_fdoa = predict_fdoa(x, satelites, vel_satelites, f0, c)
    % Predict FDOAs
    pos = [x(1); x(4); x(7)];
    vel = [x(2); x(5); x(8)];

    dx = (pos - satelites')';        % Nx3
    dv = (vel_satelites - vel');     % Nx3
    R = sqrt(sum(dx.^2, 2));         % Nx1
    rel_vel = sum(dx .* dv, 2) ./ R; % Nx1
    FOAs = (f0 / c) * rel_vel;       % Nx1
```

```
    z_pred_fdoa = FOAs(2:end) - FOAs(1); % Nx1
end


function az = predict_aoa_h(x, sat_pos)
    % Predict horizontal AOAs
    pos = [x(1); x(4); x(7)];
    Ns = size(sat_pos,1);
    az = zeros(Ns,1);
    for i=1:Ns
        dx = pos(1) - sat_pos(i,1);
        dy = pos(2) - sat_pos(i,2);
        az(i) = atan2(dy, dx); % rad
    end
end
```


Co-funded by the European Union

9.3 Integrity indicator

The following code is used to check the behaviour and performance of the integrity indicator for a given trajectory.

```
 % ------------------------------ Trajectory -------------------------------

traj = 1;

switch traj
       case 1, data = load('ny_madrid.mat'); % New York → Madrid
       case 2, data = load('saopaulo_milan.mat'); % São Paulo → Milan
end

lat_true = data.latitudes{1,1};
lon_true = data.longitudes{1,1};
h_true = data.heights{1,1};
T = length(lat_true);
perfect_LLA = [lat_true(:), lon_true(:), h_true(:)];

% full trajectory once to ENU relative to first point

ENU0 = perfect_LLA(1,:);
perfect_ENU = zeros(T,3);

for i = 1:T
       perfect_ENU(i,:) = lla2enu_local(perfect_LLA(i,:), ENU0);
end

% MLAT & ADS-B standard deviation (standards), in meters

sigma_H_MLAT = 10;
sigma_H_ADSB = 5;
sigma_res = sqrt(sigma_H_ADSB^2 + sigma_H_MLAT^2);

% ---------------- Monte Carlo setup ----------------

N_MC = 1000;
fprintf('Monte Carlo runs - %d\n\n', N_MC);

delta_r_MC = zeros(T,N_MC);
Th_MC = zeros(T,N_MC);
CB_MC = zeros(T,N_MC);
A_MC = zeros(T,N_MC);

% ---------------- MC loop ----------------

for mc = 1:N_MC

       % rng(12345); % reproducibility
```

```
% [East, North] horizontal noise, in meters
noiseMLAT = sigma_H_MLAT * randn(T,2);
noiseADSB = sigma_H_ADSB * randn(T,2);
noiseADSB2 = 0 * noiseADSB; % jamming

% [East, North] bias, in meters
bias_ENU = [0, 0]; % static spoofing

% adding noise and bias, in ENU
mlat_ENU = perfect_ENU + [noiseMLAT, zeros(T,1)]; % MLAT (no bias)
adsb_ENU = perfect_ENU + [noiseADSB + noiseADSB2, zeros(T,1)]; % ADS-B noise
adsb_ENU(:,1:2) = adsb_ENU(:,1:2) + bias_ENU; % horizontal bias, ADS-B faulty

% ---------------- Biases & thresholds ----------------

OP = 2;
PFA_pfh = 2e-3;
PMD = 1e-3;

Pfa_comp = 1 - (1 - PFA_pfh)^(OP/3600);
PFA = sqrt(Pfa_comp/3);

k_G = norminv(1-PFA);
k_R = sqrt(-2*log(PFA));
k_M = norminv(1-PMD);

BM = 5*ones(T,1); % MLAT bias (set to 5 meters, constant)

tau = 0.01;
v_nom = 250;
sigma_v = 0.1*v_nom;
speed = v_nom + sigma_v*randn(T,1);

BA = speed * tau; % ADS-B bias (aircraft's speed dependent)

dt_nom = OP*ones(T,1);
error_time = randn(T,1);
error_time = 0.125 * error_time/max(abs(min(error_time)),max(error_time));
dt_eff = dt_nom + error_time;
a_max = 3;

BE = 0.5 * a_max .* (dt_eff.^2); % Time extrapolation bias

% For the purposes of testing these simulations and this phase of validation,
it is assumed that systematic errors can be neglected. Therefore, the total
bias is set to 0:

BT = 0; % BM + BA + BE; % Total bias

% residuals [East, North]
resE = mlat_ENU(:,1) - adsb_ENU(:,1);
resN = mlat_ENU(:,2) - adsb_ENU(:,2);
delta_r = hypot(resE, resN);
```

```
        Th = (k_R * sigma_res) * ones(T,1) + BT;
        CB = 2*BT + ((k_R + k_M) * sigma_res) * ones(T,1);

        A = delta_r > Th;

        delta_r_MC(:,mc) = delta_r;
        Th_MC(:,mc) = Th;
        CB_MC(:,mc) = CB;
        A_MC(:,mc) = A;

end

% ---------------- MC statistics ----------------

mean_delta_r = mean(delta_r_MC,2);
mean_Th = mean(Th_MC,2);
mean_CB = mean(CB_MC,2);
mean_alarms = mean(A_MC,2);

fprintf('Mean δr: %.2f m\n', mean(mean_delta_r));
fprintf('Mean Th: %.2f m\n', mean(mean_Th));

% alarms per trajectory point and MC run

A_any = delta_r_MC > Th_MC;
fprintf('\nδr > Th => %d out of 5 400 000\n', sum(A_any(:)));

% -------------------------- Trajectory plots (LLA) ----------------------

% Reference point for ENU → LLA
LLA0 = perfect_LLA(1,:);

% ENU to LLA
adsb_LLA_plot = zeros(T,3);
mlat_LLA_plot = zeros(T,3);

for i = 1:T
        adsb_LLA_plot(i,:) = enu2lla(adsb_ENU(i,:), LLA0, ‘ellipsoid’);
        mlat_LLA_plot(i,:) = enu2lla(mlat_ENU(i,:), LLA0, ‘ellipsoid’);
end

figure;
tiledlayout(2,2,'TileSpacing','Compact','Padding','Compact');

% Left: full trajectory
ax1 = nexttile([2 1]);
plot(perfect_LLA(:,2), perfect_LLA(:,1), 'g-', 'LineWidth', 1.5);
hold on;
plot(adsb_LLA_plot(:,2), adsb_LLA_plot(:,1), 'b-', 'LineWidth', 1.5);
plot(mlat_LLA_plot(:,2), mlat_LLA_plot(:,1), 'r-', 'LineWidth', 0.5);
xlabel('Lon [°]');
ylabel('Lat [°]');
legend('True position', 'ADS-B', 'MLAT', 'Location', 'best');
title('Trajectory (Full)');
```

```
grid on;

% zoom regions
idx1 = 2500:2510;
idx2 = 2650:2800;
margin = 0.00005;

% zoom 1 region rectangle
x1 = [min(perfect_LLA(idx1,2)); max(perfect_LLA(idx1,2))];
y1 = [min(perfect_LLA(idx1,1)); max(perfect_LLA(idx1,1))];
rectangle('Position',[x1(1)-margin,        y1(1)-margin,        diff(x1)+2*margin,
diff(y1)+2*margin],...
'EdgeColor','k','LineStyle','-','LineWidth',1.2');

% zoom 2 region rectangle
x2 = [min(perfect_LLA(idx2,2)); max(perfect_LLA(idx2,2))];
y2 = [min(perfect_LLA(idx2,1)); max(perfect_LLA(idx2,1))];
rectangle('Position',[x2(1)-margin,        y2(1)-margin,        diff(x2)+2*margin,
diff(y2)+2*margin],...
'EdgeColor','k','LineStyle','-','LineWidth',1.2');

% Right-top
ax2 = nexttile;
plot(perfect_LLA(idx1,2), perfect_LLA(idx1,1), 'g-', 'LineWidth', 1.5); hold on;
plot(adsb_LLA_plot(idx1,2), adsb_LLA_plot(idx1,1), 'b-', 'LineWidth', 1.5);
plot(mlat_LLA_plot(idx1,2), mlat_LLA_plot(idx1,1), 'r-', 'LineWidth', 0.5);
xlabel('Lon [°]');
ylabel('Lat [°]');
title(sprintf('Trajectory (Zoom: points %d – %d)', idx1(1), idx1(end)));
grid on;
axis tight;
legend('True position', 'ADS-B', 'MLAT', 'Location', 'best');

% Right-bottom
ax3 = nexttile;
plot(perfect_LLA(idx2,2), perfect_LLA(idx2,1), 'g-', 'LineWidth', 1.5); hold on;
plot(adsb_LLA_plot(idx2,2), adsb_LLA_plot(idx2,1), 'b-', 'LineWidth', 1.5);
plot(mlat_LLA_plot(idx2,2), mlat_LLA_plot(idx2,1), 'r-', 'LineWidth', 0.5);
xlabel('Lon [°]');
ylabel('Lat [°]');
title(sprintf('Trajectory (Zoom: points %d – %d)', idx2(1), idx2(end)));
grid on;
axis tight;
legend('True position', 'ADS-B', 'MLAT', 'Location', 'best');

% alarms for each MC run
num_alarms_per_MC = sum(A_MC, 1);

figure;
bar(1:N_MC, num_alarms_per_MC, 'FaceColor', 'k', 'EdgeColor', 'k');
xlabel('Monte Carlo runs');
ylabel('Alarms');
title('Alarms triggered every MC run');
grid on;
```

```
fprintf('\nMean Alarms per MC run: %.4f \n', mean(num_alarms_per_MC(:)));
```