

Black-Boxing GNSS Signals Post-Processing Through Machine Learning for Multi-Agent Collaborative Positioning of IoT Devices

Original

Black-Boxing GNSS Signals Post-Processing Through Machine Learning for Multi-Agent Collaborative Positioning of IoT Devices / Minetto, Alex; Jahier-Pagliari, Daniele; Rotunno, Angela. - ELETTRONICO. - (2024), pp. 589-603. (Intervento presentato al convegno 2024 International Technical Meeting of The Institute of Navigation tenutosi a Long Beach, California (USA) nel January 23 - 25, 2024) [10.33012/2024.19566].

Availability:

This version is available at: 11583/2987532 since: 2024-04-03T14:42:47Z

Publisher:

Institute of Navigation (ION)

Published

DOI:10.33012/2024.19566

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository



Publisher copyright

GENERIC -- per es. Nature : semplice rinvio dal preprint/submitted, o postprint/AAM [ex default]

The original publication is available at <https://www.ion.org/publications/abstract.cfm?articleID=19566> / <http://dx.doi.org/10.33012/2024.19566>.

(Article begins on next page)

Black-Boxing GNSS Signals Post-Processing Through Machine Learning for Multi-Agent Collaborative Positioning of IoT Devices

Alex Minetto¹ , Daniele Jahier-Pagliari¹ , Angela Rotunno^{1,2},
¹Politecnico di Torino Turin, Italy, ²Punch Torino S.p.A., Turin, Italy

BIOGRAPHY

Alex Minetto received the B.Sc., and M.Sc. degrees in Telecommunications Engineering from Politecnico di Torino, Turin, Italy and his Ph.D. degree in Electrical, Electronics and Communications Engineering, in 2020. He joined the Department of Electronics and Telecommunications of Politecnico di Torino in 2021 as researcher and assistant professor. His current research interests cover navigation signal design and processing, advanced Bayesian estimation applied to Positioning and Navigation Technologies (PNT) and applied Global Navigation Satellite System (GNSS) to space weather and space PNT.

Daniele Jahier-Pagliari received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Turin, Italy, in 2014 and 2018, respectively. He is currently an Assistant Professor with the Politecnico di Torino. His research interests are in the computer-aided design and optimization of digital circuits and systems, with a particular focus on energy-efficiency aspects and on emerging applications, such as machine learning at the edge.

Angela Rotunno was born in Salerno, Italy. She received the M.Sc. degree in mechanical engineering in 2010, and the M.Sc. in mechatronic engineering in 2023 from the Politecnico di Torino. Her research work has been focused on the use of machine learning techniques for collaborative positioning solutions. She is currently working as test automation engineer for Punch Torino, an engine development company.

ABSTRACT

Nowadays, GNSS (GNSS) receivers are embedded in a variety of electronics devices, and a growing number of users rely on them to track their position, velocity, and time. The density of Global Navigation Satellite System (GNSS) receiver has especially increased in urban areas with the advent of small-scaled IoT devices. Due to the limited GNSS signal power and the variability of the environment, continuous GNSS signal tracking may represent a demanding task for receivers, which, in addition, have to perform demodulation of the navigation message to provide the user with meaningful information. Furthermore, when operated by low-power platforms, such as Internet of Things (IoT) devices, the aforementioned tasks may quickly drain the battery. On the other hand, the low-power-consumption network connectivity hosted by IoT electronics could represent an ideal environment to enable new patterns for their state estimation, based on collaborative, multi-agent Position Navigation and Time (PNT) methods that would not imply continuous operation of the embedded GNSS receiver. This preliminary study aims to understand whether machine learning techniques could support such paradigms for position estimation in IoT devices. We generated an artificial environment, where conventional GNSS share their multi-satellite delay-Doppler matrices and their positions. These data are meant to be used to estimate a "reference" IoT receiver position. We used two open-source libraries, XGBoost, to implement a gradient-boosted decision tree, and Keras, to implement a multi-layer perceptron. The estimation error on the IoT receiver position obtained using machine learning tools is lower (typically, 10 % to 20 %) than the estimation error returned by a simplistic reference model, based on the arithmetic average of the networked receivers' positions. The results suggest that rudimentary ML algorithms can extract fundamental information from collaborative users, thus opening new frontiers to collaborative GNSS navigation.

I. INTRODUCTION

GNSS technology allows the user to effectively perform receiver state estimation by means of multilateration of radiometric ranges and Doppler shifts, that are observed upon satellites' signal detection and tracking, through the estimation of the signals Time-of-Arrival and the observation of their frequency offsets (Misra and Enge, 2006). Due to the limited signal power and the satellites and receivers' dynamics, as well as the variability of the environment, the continuous GNSS signal detection and tracking may represent a challenging task for receivers. Furthermore, the demodulation of the navigation data is mandatory for the receiver's standalone operational capabilities unless aiding information are gathered through network connectivity. Furthermore, when those tasks are operated by low-power platforms such as Internet of Things (IoT) devices, they may quickly drain the battery by reducing devices' operational life (Banville and van Diggelen, 2016). However, IoT electronics host by

definition low-power-consumption network connectivity (Bembe et al., 2019), and they could enable new patterns for their state estimation, based on collaborative, multi-agent Position Navigation and Time methods that would not imply a continuous operations of the embedded Internet of Things receiver or even a dramatic reduction in the number of required tasks.

Several solutions have been indeed considered in literature to reduce the power consumption of embedded GNSS receivers (Grenier et al., 2023), but network cooperation among the devices is still of interest for sensor networks and their autonomy (Oliveira et al., 2023). It is naturally of interest to evaluate whether ML algorithms can be designed to replace battery-draining hardware or software functional blocks in GNSS receiver architecture. Of course, the use of machine learning is not new in the field of GNSS, as demonstrated by the growing number of scientific contributions on the topic (Caparra et al., 2021; Navarro et al., 2021; Nardin et al., 2023). This work aims at merging this main trends with the collaborative GNSS PNT previously addressed by literature (Wen et al., 2020; Minetto et al., 2019, 2022).

With this study, we aim at understanding whether machine learning techniques (Mahesh, 2020) could support collaborative paradigms for the state estimation in IoT devices, in the attempt to avoid the need of continuous signal tracking, demodulation of the navigation message, and observables construction and correction, by correlating local observables with the code offset and Doppler shift observables made available by a set of nearby networked collaborative users. In particular, we assume to share time-stamped, multi-satellite, multi-frequency delay-Doppler matrices and the associated, anonymized, state estimates gathered by networked GNSS receivers operating within a pre-existent, accessible communication network. These matrices can be resembled to as the collections of peaks coordinates of the respective cross-ambiguity functions of each satellite signal, and they can be characterized by a given uncertainty both at the acquisition and tracking stages of the collaborative receivers.

The pursued approach relies on the correlation domain exploited by well-known Direct Position Estimation (DPE) (Closas et al., 2007) but it offers a complementary view to the problem by leveraging a networked perspective and ML solutions.

The study aims at demonstrating the effectiveness of baseline machine learning techniques in the interpretation and black-boxing of the GNSS signals tracking, message demodulation and observables construction stages of a conventional GNSS receiver architecture. Such an interpretation only relies on the coarse acquisition locally performed by the target IoT receiver and its relationship with the observations of nearby collaborating receivers.

The remainder of the paper is as follows. Section II recalls the fundamental operational tasks of canonical GNSS receivers architecture and highlights the data that will be shared in the proposed approach. It also introduce basics aspects of ML algorithms. Section III describes the proposed approach and describes the simulation environment build for the analysis. In Section IV validation of the ML processing is presented, and the preliminary results on the position estimation are discussed.

II. BACKGROUND

1. GNSS receiver architecture

From a high-level perspective, the tasks carried out by a conventional GNSS receiver can be associated to four main operational tasks, namely:

- **Antenna and front-end assembly**, where the signals received from the satellites are amplified, filtered, down-converted and then digitalized;
- **Acquisition**, where the satellite is identified and a coarse estimation of code phase offset and Doppler shift is performed;
- **Tracking**, through which a refinement of the local code phase offset and Doppler shift is accomplished to ensure accurate timing information and data demodulation;
- **Application processing**, where the navigation message demodulation is completed, followed by pseudorange construction and estimation of the PVT solution.

These tasks are performed according to a conventional receiver architecture as shown in Fig. ???. Each satellite within a GNSS constellation has a unique Pseudo-Random Noise (PRN) code, that is broadcast as part of the navigation message. This code allows any receiver to unambiguously discriminate the satellites in view. However, the satellite signal has an extremely low power at the ground level, in the order of $10^{-5}W$, which means, it can be easily masked out by noise. The receiver front-end stage must be capable, first of all, to amplify and filter the signal to an intermediate frequency, and then convert it to a stream of numerical signal samples. We may refer to this stage as pre-correlation stage. Baseband processing then include all those algorithms required to track the radionavigation signals of the visible GNSS. The operations here are identified as *acquisition* and *tracking* and they both rely on the use of correlation functions. The received signal is repeatedly compared with a replica of what is the expected satellite.

The acquisition stage is in charge of detecting the i -th satellite, based on the correlation of the received signal with several local replicas of the possible "expected" signals: when the local replica and the input signal are aligned, the tracking process is

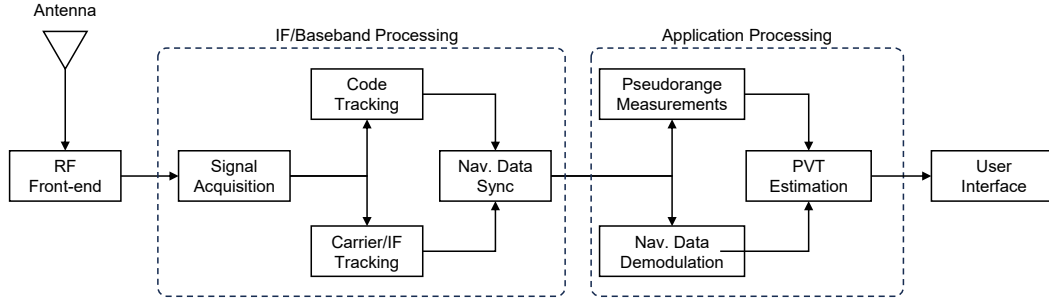


Figure 1: Block diagram of a GNSS receiver conventional architecture.

initialized. The tracking stage uses narrowband correlation functions as well, to refine the local replica generation. In general, the receiver tracks each signal using dedicated channels running in parallel, where each channel tracks one signal, providing pseudorange and phase measurements.

The tracking stage provides a pair of *code delay* and *Doppler shift* for each GNSS signal in tracking. Along with the messages' Handover Words and satellite ephemeris. The code phase is exploited to estimate the overall signal time of flight, that, accounting for the local clock bias, leads to pseudorange measurement, while the Doppler shift is converted into pseudorange rates. Pseudoranges and pseudorange rates hence constitute the observables used to estimate the PVT solutions. For static receivers and by constraining the local geographical area to some extent, each code phase offset - Doppler shift pair allows to identify a specific satellite without ambiguity. This principle is also exploited in DPE (Closas et al., 2007).

2. Machine Learning

The goal of this study is to assess the effectiveness of Machine Learning (ML) techniques for position estimation in IoT devices. In particular, we frame the problem as a *supervised learning* one, in which the ML model is trained on a set of (i, o) pairs, where i is a vector of input features, and o is the corresponding ground truth output, i.e., the IoT receiver's position. In our case, o corresponds to the $[x, y]$ position of the receiver in the Earth Centered Earth Fixed (ECEF) coordinates system, as detailed in Section III. Therefore, the output is real-valued, requiring the use of *regression models*. At test time, the ML solution outputs a predicted position $\hat{o} = f(i)$, where $f()$ is the learned relation between the input features i and the output.

Different ML models impose different priors on the form of $f()$, as well as offering different trade-offs in terms of number of samples required for training convergence, and time and memory complexity for prediction. To explore these trade-offs, in this work we considered two significantly different families of models, namely Gradient Boosted Decision Trees (GBTs) and Multi-Layer Perceptrons (MLPs).

GBTs are shallow, non-parametric ML models, built as ensembles of Decision Trees (DTs). By itself, a single DT, when used for regression, learns a piece-wise constant approximations of the target variable, applying a set of if-then-else rules to each sample. Precisely, each DT node compares one feature of the input with a learned threshold, and passes the sample to its left or right child based on the result. This is repeated recursively until a leaf node is reached, which contains the output prediction. The features considered in each node, and the corresponding thresholds, are optimized at training time with the goal of minimizing a loss function that measures the prediction error on training samples (Maimon and Rokach, 2014). While only requiring a few lightweight branching operations at prediction time, single DTs are extremely prone to over-fitting, thus poorly generalizing on unseen samples. This limitation is addressed thanks to ensemble learning, that is, by aggregating the predictions of multiple independent DTs. GBTs, in particular, are DT ensembles trained sequentially with a *gradient boosting* approach, in which each new DT learns to produce an output that corrects the aggregated prediction error of all previous ones. A local linear approximation of the residual error is used, which relies on the gradient of the loss function. A detailed description of the GBT training algorithm is out of scope of this paper, and we refer readers to (Maimon and Rokach, 2014) for details. In our work, we implement GBT models using the state-of-the-art open source *eXtreme Gradient Boosting* (XGBoost) library, which implements several additions to the basic training algorithm, such as shrinkage and bagging, as well as computational optimizations for parallel and distributed execution (Chen et al., 2015).

MLPs are one of the simplest types of Artificial Neural Network (ANN). Namely, they are built as linear sequences of multiple layers of artificial neurons, each of which implements a nonlinear function of its inputs. The network is *feed-forward*, i.e., each layer is only connected to the following one, without recurrent (backward) connections. Furthermore, MLP layers are *fully-connected*, that is, each neuron's output is a function of *all* the neurons in the previous layer (and for the first layer, of all

the input features). Mathematically, the l -th layer's neurons constitute a vector $z_l = g(W_l \cdot z_{l-1} + b_l)$ where $z_0 = i$, W_l and b_l are the trainable weights matrix and biases vector of the layer, and $g()$ is a non-linear *activation function*. Common functional forms for $g()$ include sigmoid, hyperbolic tangent, and the rectified linear unit (ReLU). For regression problems, the output of each neuron in the last layer of the MLP corresponds to one predicted value, and usually has an identity activation function. The MLP is trained by updating the weights and biases of all layers to minimize a differentiable loss function, using some variant of Stochastic Gradient Descent (SGD). For details on the model and on the training algorithm we refer readers to (Goodfellow et al., 2016). In this work, we leverage the Keras Python-based framework for training our MLP models (Chollet et al., 2015), which exposes a simple interface to define the model architecture, the loss function, the optimizer, etc.

Both types of ML model include several *hyper-parameters*, that is, non trainable configurations that significantly affect their performance, both on training data and on unseen test data. For GBTs, these include the number of DTs in the ensemble, their maximum depth, the learning rate, etc, whereas for MLPs, they include the number of layers, the number of neurons in each internal layer, the type of activation function, etc. As detailed in Section III, the most relevant hyper-parameters for each model type have been tuned by means of an automated framework, called Optuna (Akiba et al., 2019).

III. METHODOLOGY

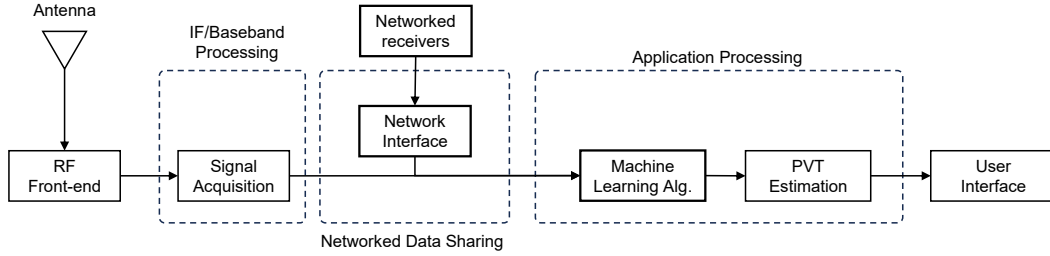


Figure 2: Proposed IoT GNSS receiver architecture: use of machine learning to perform PVT estimation within a network of collaborative receivers.

With reference to the standard architecture shown in Fig. 1, we propose to replace the full post-processing chain by means of ML techniques, supported by collaborative users sharing their data, i.e., delay-Doppler data and estimated position. In order to do so, we included in the diagram a network interface devoted to the exchange of data among the receivers. The concept architecture is then modified as per Fig. 2. The investigation has been pursued through two different stages. First, we set up an artificial, experimental environment in Matlab. We assume a scenario in which an IoT target receiver is surrounded by a set of n networked receivers, randomly located within a radius R_{tol} . The target IoT receiver is intended having no chance to autonomously estimate its own state, unless assisted by surrounding receivers. We hence build a dataset of aiding data where the following information have been shared by each aiding receiver:

- Receivers location in geodetic, local and ECEF coordinates,
- Identifiers of the satellites in view,
- delay-Doppler array (for respect to each satellite in view).

Note that, in the definition of the dataset we worked in a backward direction with respect to the sequence of operations done by a GNSS receiver: we started by defining the *position* for the receivers on the Earth, which is thus a known, noiseless datum, and by simulating a constellation scenario, we computed the receivers-to-satellites ranges (or pseudoranges) and, from these latter, we simulated the code delay - Doppler shift pairs. The second part of the work has been focused on the data processing via

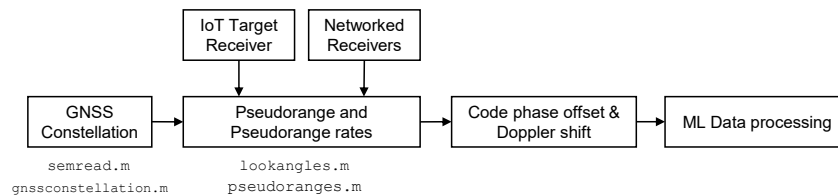


Figure 3: Block diagram describing the workflow of the simulation environment.

ML tools in Python environment, using two different machine learning open-source libraries, XGBoost and Keras. The aim is to understand whether or not it is possible to estimate the IoT receiver position by exploiting the data shared by the networked receivers. The workflow of the presented methodology is shown in Fig. 3. Eventually, the estimation error affecting the IoT state obtained using the two machine learning tools has been compared to an over-simplistic, reference model, where the IoT receiver's position is estimated by the arithmetic average of the networked receivers' positions. A single-scalar metric, defined as the ratio between the Root Mean Square Error (RMSE) of the position estimated by the machine learning tools and the RMSE of the position estimated by the reference model has been used to compare the machine learning models' performance against the simplified reference model. A comparison against nominal GNSS solution has been considered not practicable in this phase, since a legacy PVT solution would have returned the reference position used for the observable generation.

1. EXPERIMENTAL ENVIRONMENT

a) Generation of the GNSS constellation

The built-in Matlab function *semread_function.m* retrieves almanac data for all the available GPS satellites for a specific time and timezone. For each satellite in the constellation, the almanac includes:

1. Coarse orbit information (i.e., orbit ephemeris)
2. Health status
3. Satellite vehicle identifier
4. clock corrections
5. GPS time

Such data are exploited by the receiver to identify a specific satellite and determine its location along its orbit at a given time instant. Based on these data, the Matlab function *gnssconstellation.m* returns the satellite positions and velocities at the desired datetime t . Positions and velocities are specified in the ECEF coordinate system.

b) Generation IoT and networked receivers locations

To build the experimental scenario, we first define the position of the IoT receiver. The position is randomly set within a specified area, identified by its geodetic coordinates (maximum - minimum latitude and longitude, plus average altitude). The initial area for training was chosen to correspond to Germany, given the edges in Fig. 4b.

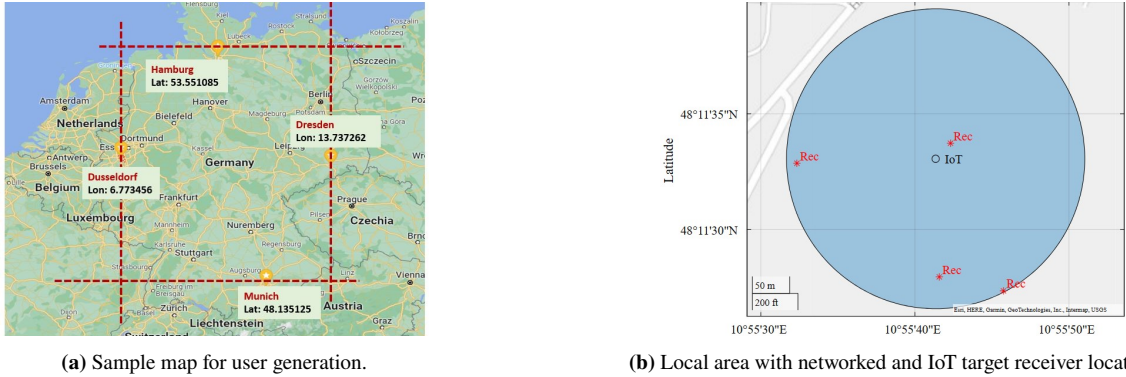


Figure 4: Constraints and output of a generated scenario. The of the area where the scenario is developed (a), and detail of a single set of locations including the IoT receiver ("IoT") and the four surrounding networked receivers (b).

A first dataset of 500 points has been created following this approach - see Fig. 4, where a detail of the single row (IoT plus the four networked receivers) is shown. The attributes that need to be defined for the receivers are listed in Table 1

Normally, due to environmental constrains, such as the presence of tall buildings or trees, a certain receiver might not be able to see all the satellites available at that specific time. By specifying a random *mask angle* for each receiver, it is possible to cut out of the view a certain number of satellites, thus making the scenario more realistic. Once this has been done, we use the Matlab function *lookangles.m* to calculate azimuth and elevation of the satellites in view and then cut out all the non-relevant data (i.e., coming from non-visible satellites).

Note that the position of the IoT receiver is defined with respect to geodetic coordinates (Latitude Longitude Altitude, or LLA),

Table 1: Attributes of the generated GNSS receivers

Attribute	Unit	Description
Name	n.a.	Receiver numerical identifier (e.g., 1,2,...,n)
Mask angle	[deg]	Random elevation mask
Azimuth	[deg]	Azimuth angle with respect to satellites in view
Elevation	[deg]	Elevation angle with respect to satellites in view
Visibility	n.a.	Number of satellites visible from the receiver
Sat ID	n.a.	Satellite identification number
SatNum	n.a.	Number of satellites in view
VisibleSatPos	[m,m,m]	Position of satellites in view, ECEF coord
VisibleSatVel	[m/s,m/s,m/s]	Velocities of satellites in view, ECEF coord
Geodetic position	[deg,min,sec;deg,min,sec]	Receiver location in geodetic coordinates
Position alt	[m]	Altitude of the receiver
Position lla	[deg,deg,m]	Position of the receiver in lat,lon,alt
Position enu	[m,m,m]	Position of the receiver, EastNorthUp coord
Position ecef	[m,m,m]	Position of the receiver, ECEF coord
Posdiff	[m,m,m]	Difference in position between satellite and receiver
LOS Vector	[m,m,m]	LOS vector between satellite and receiver
Velocity enu	[m/s,m/s,m/s]	velocity of the receiver, East-North-Up coord
velocity enu	[m/s,m/s,m/s]	velocity of the receiver, ECEF coordinates
Velocity ecef	[m/s,m/s,m/s]	velocity of the receiver, East-NotH-Up coord
Pseudorange	[m]	pseudorange between satellite and receiver
Pdot	[m/s]	pseudorange rate between satellite and receiver
Pseudorange matlab	[m]	pseudorange computed using Matlab toolbox function
Range rate matlab	[m/s]	pseudorange rate computed using Matlab toolbox function
Clock Bias	[s]	receiver clock bias
Integer	n.a.	number of integer repetitions of codes in pseudorange
Delay	[s]	delay computed from pseudorange
Doppler	[1/s]	doppler frequency computed from pseudorange rate
Delay Matlab	[s]	delay from pseudorange (Matlab toolbox function)
Doppler Matlab	[1/s]	doppler frequency from pseudorange rate (Matlab toolbox function)

then translated into both East North Up (ENU) and Earth Centered Earth Fixed (ECEF) coordinates. The geodetic coordinates are preferred when the main need is to locate a point on the Earth's surface; however, they are less practical when, for instance, the networked receivers must be generated within a certain radius from the IoT location. Also, the velocity of the receivers will be likely available in local ENU coordinates, rather than in Geodetic or ECEF coordinates. For all these reasons, once the IoT receiver coordinates have been defined, they are translated into local coordinates and only afterwards the four networked receivers' locations are generated (again, in ENU coordinates).

c) Computation of the pseudoranges and pseudoranges' rates based on visible satellites

In our environment, the IoT receiver is located at the center of an area of radius R_{tol} . In the same area, there are 4 networked receivers, and each of them has a certain number of satellites in view, which means, has all the relevant information required to compute the pseudorange and the pseudorange rate. The Matlab function *pseudoranges.m* takes as inputs:

- the receiver position, in geodetic coordinates;
- the visible satellites' positions, specified as an $S \times 3$ matrix in ECEF coordinates, where S is the number of satellites;
- the receiver velocity, in local coordinate system;
- the visible satellites' velocities, specified as an $S \times 3$ matrix in ECEF coordinates, where S is the number of satellites.

The clock bias is not taken into account in this formula; for this reason, it is not formally correct to talk about *pseudorange*. To get an effective pseudorange, the data returned from the function must be corrected using the information on the clock bias. An alternative computation, which has been conceived for this work, uses:

- for the calculation of the pseudorange, the difference between receiver and i^{th} satellite positions, both expressed in ECEF coordinates, plus the clock bias;

- for the calculation of the pseudorange rate, the dot product of the difference between receiver and i^{th} satellite positions and the LOS (i.e. the unit vector for position difference).

The current investigation assumes all the receiver have null clock bias. For what concerns the velocities of the receivers instead, it is made the assumption that the IoT receiver is static, while the networked receivers velocities are comparable to those of walking pedestrians (0.83 to 1.39 m/s).

Once the pseudorange and pseudorange rate have been calculated, it is possible to generate the observables based on code delay τ and Doppler frequency f_d :

$$\tau = \rho \bmod L_C \quad (1)$$

$$f_d = (f_{L1} \cdot \dot{\rho})/c \quad (2)$$

where ρ and ρ_{rate} are the ones previously computed, f_{L1} is the carrier frequency and L_{CODE} is calculated as:

$$L_{CODE} = p \cdot L_{chip}, \quad (3)$$

where p is the code length and $L_{chip} = c/R_{chip}$, with c equal to the speed of light. As a note, for a receiver that has in view n satellites, the integer number of code repetitions for all of these satellites is the same. If this condition was not met, the code delay parameter would lose its meaning and role in the satellite's identification process. With reference to the simulation scenario, we must make sure that all of the networked receivers are located within an area that guarantees that the same number of integer code repetitions is shared amongst the pseudoranges. By means of increasing the tolerance radius R_{tol} , it has been verified that the data are always consistent for networked receivers located within a $23km$ radius from the IoT receiver. This value is well larger than the $200m$ radius used for the development of the simulation environment.

d) Processing of the dataset before data processing

Before moving to machine learning tools, the data have been pre-processed by applying some further constraint. In particular:

- for every point of the dataset, all of the receivers (networked and IoT) must have at least 6 satellites in common; where this condition is not met, the data point is discarded;
- for every point, we want to export the same number of features; it has been chosen to limit the data to 6 common satellites in view for each receiver.

Based on the assumptions above, whenever the receivers share more than 6 satellites in view, only the data related to the first 6 are part of the data export, while all the others are not taken into account. Considering the number of features to be processed, a dataset made of 500 points has been created.

Table 2 shows the final set of parameters that will be processed with ML tools. Networked receivers are numbered 1 to 4, while the IoT receiver is identified as receiver 5.

2. Machine Learning solutions

Once that the data are available, we want to assess if it is possible to estimate the IoT receiver position using the information shared by the networked receivers. Below is a recap of the assumptions and constraints applied to the input dataset:

- the IoT receiver can be located anywhere in the area delimited by the coordinates specified in Fig. 4b;
- for each IoT receiver, $n = 4$ networked receivers are randomly generated to lay within a radius $R_{tol} = 200$ m from the center IoT location;
- for every receiver, a random mask angle has been defined, ranging from 0 to 20 degrees;
- the clock bias is always set equal to zero;
- the networked receivers are all moving slowly (e.g., a person walking or running);
- the IoT receiver is not moving;
- for every receiver (networked and IoT), there must be at least 6 satellites in view;
- for every point of the dataset, all of the receivers (networked and IoT) must share at least 6 satellites in view; any additional data is truncated.

Table 2: Data export. Networked receivers are numbered 1 to 4, while IoT target is identified as receiver 5.

Label	Unit	Description
Lat_i	deg	Latitude of the i -th receiver
Lon_i	deg	Longitude of the i -th receiver
Alt_i	[m]	Altitude of the i -th receiver
x_i	[m]	x-ECEF coordinate of the i -th receiver
y_i	[m]	y-ECEF coordinate of the i -th receiver
z_i	[m]	z-ECEF coordinate of the i -th receiver
bias_i	[s]	Clock bias of the i -th receiver
τ_{i1}	[s]	Code phase offset of the i -th receiver to satellite 1
τ_{i2}	[s]	Code phase offset of the i -th receiver to satellite 2
τ_{i3}	[s]	Code phase offset of the i -th receiver to satellite 3
τ_{i4}	[s]	Code phase offset of the i -th receiver to satellite 4
τ_{i5}	[s]	Code phase offset of the i -th receiver to satellite 5
τ_{i6}	[s]	Code phase offset of the i -th receiver to satellite 6
$f_{d,i1}$	[1/s]	Doppler shift of the i -th receiver to satellite 1
$f_{d,i2}$	[1/s]	Doppler shift of the i -th receiver to satellite 2
$f_{d,i3}$	[1/s]	Doppler shift of the i -th receiver to satellite 3
$f_{d,i4}$	[1/s]	Doppler shift of the i -th receiver to satellite 4
$f_{d,i5}$	[1/s]	Doppler shift of the i -th receiver to satellite 5
$f_{d,i6}$	[1/s]	Doppler shift of the i -th receiver to satellite 6
x_{avg}	[m]	x-ECEF average coordinate of the 4 networked receivers
y_{avg}	[m]	y-ECEF average coordinate of the 4 networked receivers
z_{avg}	[m]	z-ECEF average coordinate of the 4 networked receivers

Since we're treating a supervised regression problem, it is essential to split the dataset into *training set*, *validation set* and *test set*. The reason is that, the learning algorithms tends to tailor their learning parameters based on the available information. A very high accuracy in the prediction over the training set (*overfitting*) can result in poor results on a different dataset. The ability to find the best data-fitting is different from the ability to *predict*, which is instead what is expected from a machine learning tool.

a) Definition of the target output

With reference to Fig. 4b, for every receiver the location is specified both in geodetic (or Latitude Longitude Altitude (LLA)) and ECEF coordinates. During the first experiments, the choice of the geodetic coordinates as target output has shown extremely poor results. The error was in fact ranging in the order of 10^3 m, totally out of span if compared to the radius of 200 m. It must be kept in mind that at the equator 1° of latitude corresponds approximately to 111km , which means that a low numerical error on LLA coordinates will lead to high inaccuracy in the prediction. Moreover, the relation between pseudorange and LLA coordinates is highly non-linear, due to the local models for altitude estimation (i.e., Geoid WGS84). For the reasons above, all of the experiments have been conducted setting the ECEF coordinates as target output. The predictions are then translated back into geodetic coordinates, and finally to the ENU frame, to be compared, also graphically, to the true IoT position.

b) Data Normalization

In the foreseen dataset, the features can be extremely different in magnitude: they're ranging from the LLA coordinates, in the order of 10^2 , to the ECEF coordinates, in the order of 10^7 . Such a discrepancy in numeric values for the different features can greatly affect the performance of the ML algorithms, e.g., by creating numerical problems with the propagation of gradients in MLPs (Goodfellow et al., 2016). The best practice in this case is to normalize all of the entries before running the code. In our case, each feature x has been normalized as:

$$x_{normalized} = \frac{x - \mu(x)}{\sigma(x)}; \quad (4)$$

where μ is the mean of the feature and σ is its standard deviation.

c) Single Scalar Metric

The most simplistic method to obtain an approximated position of the IoT target receiver based on the collaborative data and without the need of any machine learning tool is to use an averaged value from the networked receivers' positions:

$$x_{avg} = \frac{1}{N} \cdot \sum_{i=1}^N x_i, y_{avg} = \frac{1}{N} \cdot \sum_{i=1}^N y_i, z_{avg} = \frac{1}{N} \cdot \sum_{i=1}^N z_i,$$

where $N = 4$ is the number of the i networked receivers and the coordinates x, y, z are the ones defined in ECEF domain. This approximation does not exploit any correlation with the target receiver local observations therefore it has too be considered highly suboptimal. However, it still represents an approximated solution to the positioning problem if we can suppose a uniform distribution of the collaborative receivers.

To ease the interpretation of the results, a Single-Scalar Metric (SSM) index has been defined, as the ratio between the mean square error of the position estimated by the machine learning (MSE_{ML}) and the mean square error of the position calculated as the average of the 4 networked receivers' positions (MSE_{avg}):

$$SSM = \frac{MSE_{ML}}{MSE_{avg}}. \quad (5)$$

If the SSM index is lower than 1, the prediction of the ECEF location done with the machine learning is better than the simple average of the networked receivers' positions.

d) XGBoost implementation

XGBoost provides a number of tuning hyperparameters that must be appropriately modulated to optimize the model. Table 3 describes the main hyperparameters that have been leveraged to build the model in XGBoost.

Table 3: XGBoost hyperparameters

Hyperparameter	Description	Use
Objective	Determines the loss function to be used	MSE has been chosen as optimizer in the regression
Lambda	L2-regularization term	Low values could lead to overfitting
Alpha	L1-regularization term	Low values could lead to overfitting
Colsample bytree	Features per tree	High values could lead to overfitting
Subsample	Percentage of samples per tree	Low values could lead to underfitting
Max depth	Allowed tree depth	High values can result in data overfitting
No. of estimators	Amount of desired trees	Used to prevent overfitting
Learning rate	Step size shrinkage	Used to prevent overfitting

Given the number of hyperparameters to be identified (and the ranges to be screened), the tuning has been done using Optuna, an automatic hyperparameter optimization framework. This decision has led to split the dataset into two portions, as depicted in Fig. 5:

- a set of 200 samples, to be used by Optuna for the selection of the best fitting parameters (of these, the 80 % will be used for training and the 20 % for validation)
- a set of 300 samples, to be used for training (200 samples) and validation (100 samples) once the hyperparameters have been selected in Optuna

A common practice in regression problems where data are split into training and validation sets is to divide the overall dataset into k different sub-sets to perform a k -fold Cross-Validation. At every iteration k :

- one sub-set is selected to be the validation set;
- all the others $k - 1$ sets are used for the training.

This approach allows a better estimation of the actual accuracy of the model. For our work, we have used a 3-folds cross validation, plus we have included in the training set the data used for Optuna. Note that, the hyper-parameters optimization in Optuna has been done separately first for the x , and then for the y output targets.

With XGBoost it is also possible to evaluate how significant each feature (i.e., input parameter) has been in the construction of the boosted decision trees within the model. In our case, there isn't a clear relevance of a specific feature for all the receivers, but a general trend is that higher importance is assigned to code delay information. Doppler frequency instead has been, generally,

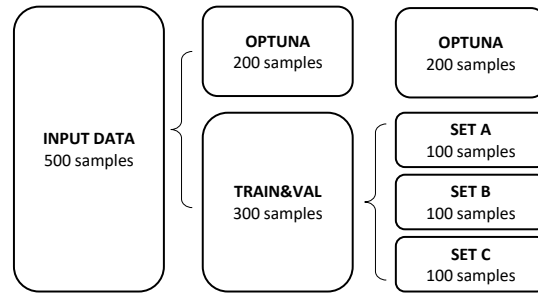


Figure 5: Diagram of the data splitting for XGBoost - use of Optuna library for hyper-parameters optimization.

classified with a lower importance. Based on these considerations, additional trials have been done excluding some features, like Doppler shift and receiver velocity, but there has been no sensible improvement in the data.

e) Keras implementation

In Keras we're dealing with two levels of tuning, one related to the structure to assign to the neural network, and the other referred to the model optimization (for a given network structure). As for the structure, the main parameters to be defined are:

- the number of *hidden Layers*: the number of hidden layers between input and output;
- the number nodes / neurons for each hidden layer
- the type of activation function. In our case, we fixed this hyper-parameter, using a Rectified Linear Unit function for input and hidden layers, while an identity function (no activation) is used for the output layer.

Note that the number of neurons for the input and output layer are always fixed. In fact, the former is equal to the number of input features. For the output layer, instead, there are two possible choices:

- using a single network to predict both x and y coordinates (2-neurons output layer), or
- using two separate models, to do the optimization over x and y coordinates separately (1-neuron output layer).

Given a certain number of input features, an independent network for each parameter is expected to maximize the learning capabilities of the model. On the other hand, using a single network to predict more than one output (i.e., *multi-task learning*) could be a better solution when the number of samples is limited with respect to the number of features. It is difficult to say a priori which approach could fit better. Both of them have been tested, and a slight improvement was noticed when using a single network for both outputs.

Similarly to what has been done for XGBoost, the data are normalized and split into training and validation (*validation_split* $n = 20\%$). The loss function (i.e., the function to be minimized during the training phase) is the mean squared error. The training phase in Keras can be advantageously managed by using an *Early Stopping* function, that terminates the training once that the loss is found to be no longer decreasing. The early stopping requires in input the maximum number of iterations and a *patience* index, which corresponds to the number of iterations with no improvement after which the training will be stopped.

IV. RESULTS

a) Validation of the models

To understand if and in what extent the models in XGBoost and Keras can be trusted, a first check can be done by applying those models to different sets of data. A first validation has been carried out by testing 10 datasets, of 500 samples each, generated over the same geographic region of the reference dataset (see Fig. 4b).

The results do not differ much from what obtained on the reference dataset; also, the performance in XGBoost and Keras are comparable - see Fig. 6. The SSM is always lower than 1, meaning that the model estimate works better than the simplistic model defined as the average of the networked receivers positions. In particular, the estimation error on the IoT receiver position obtained using machine learning tools is lower (typically, 10 to 20%) than the estimation error showed by the simplified reference model.

To go further into models' validation, new datasets have been created for other geographic regions. The aim was to understand

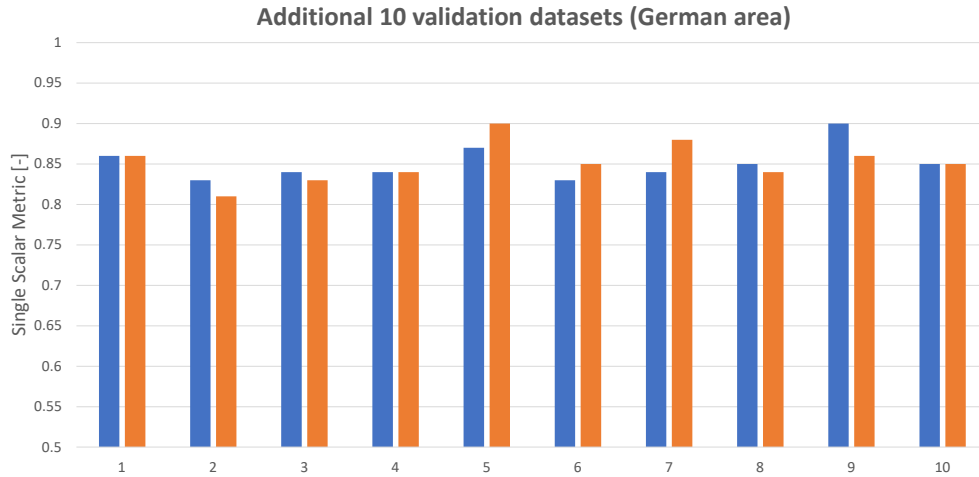


Figure 6: Comparison of results from XGBoost and Keras over 10 additional validation sets created in the German area.

if we had developed a procedure that could be used everywhere or if it was in somehow tailored on the features belonging to the original area (Germany). In detail, we have built:

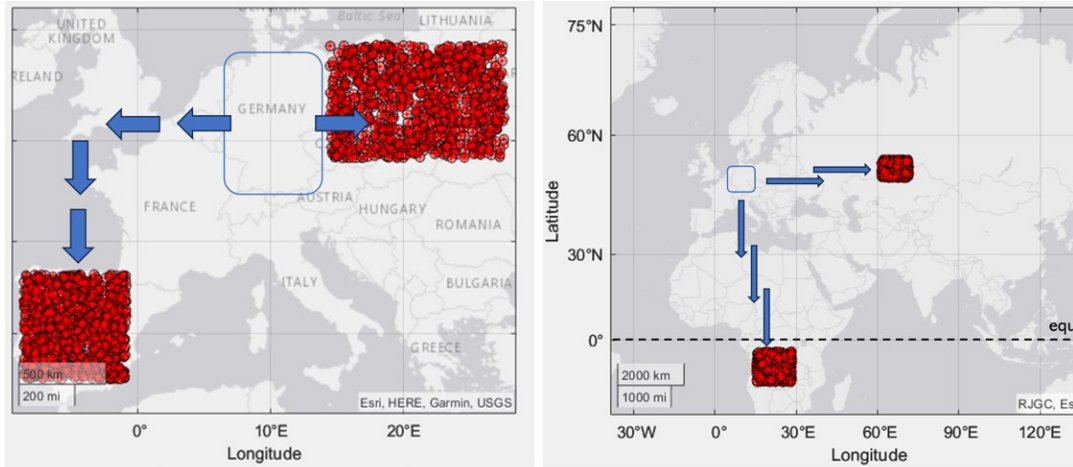


Figure 7: Distribution of the sample points generated over different geographic areas with respect to the experimental scenario: "Polish" and "Spanish" areas (left) and "Russian" and "Congolesse" areas (right).

1. Five datasets of 500 samples each, created in a region shifted in longitude (e.i., further east than the German area), delimited by the cities of Brno (Czech Republic), Gdansk (Poland), Prague (Poland) and Minsk (Belarus); we refer to this region as "Polish area".
2. Five datasets of 500 samples each, created in a region shifted in longitude and latitude (i.e., further west *and* south than the German area), delimited by the cities of Seville (Spain), Bilbao (Spain), Porto (Portugal) and Zaragoza (Spain); we refer to this region as "Spanish area".
3. A datasets of 500 samples, created in a region delimited by the cities of Lusaka (Zambia) and Kinshasa (Congo Democratic Republic); we refer to this region as "Congolesse area".
4. A datasets of 500 samples, created in a region delimited by the cities of Chelyabinsk (Russian Federation) and Astana (Kazakhstan); we refer to this region as "Russian area".

The general idea is to perform, first of all, a validation of the models obtained over the reference dataset on all the other four regions. The results are shown in Fig. 8. The prediction looks like degrading moving further east in longitude with respect to the reference area. For the Congolesse region, where we've moved below the equator, the prediction look sensibly improved.

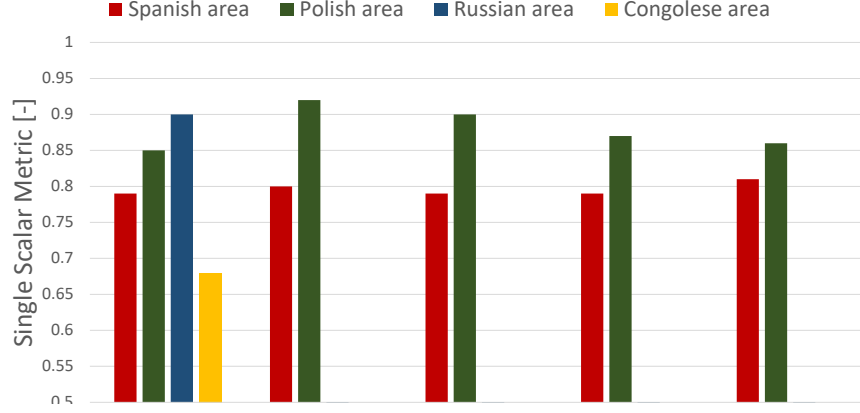


Figure 8: Keras model validation on the different validation dataset.

b) Position Estimation

. A first evaluation of the receiver's position estimate obtained through the ML models can be done by looking at the distribution of their distance from the "true" IoT position. Fig. 9 shows the values of distance from the true IoT position returned by both the reference model ("AVG", in red in the plot) and the ML model ("ML", in blue in the plot - in this case, XGBoost), while the true position of the IoT receiver coincides with the horizontal axis.

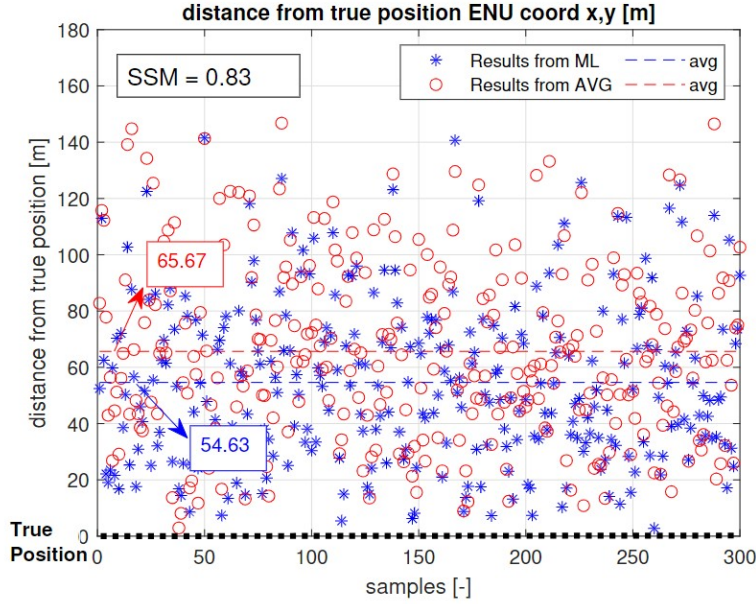


Figure 9: Distance from true IoT position based on simplified average model (red) and ML model (blue).

Keeping in mind the definition of SSM index (see 5), the values returned by ML model are, *in general*, closer to the true position: the SSM over the tested samples is 0.83. However, by looking at the detail of a single sample, it is clear that the ML models do *not* always perform better than the simplified reference model. There are cases, in fact, where the simplified reference model seems to give a better estimate, or where the two models are basically returning the same estimate (and hence there would be no advantages in using ML). A simple check can be done defining a percentage of Machine Learning Asset (MLA):

$$MLA = 100 \cdot \frac{\sum_{i=1}^N n_{MLi}}{N_{tot}} \quad (6)$$

where N_{tot} is the total number of samples, and n_{ML} is equal to 1 if the ML model estimates better than the simplified model,

and equal to 0 otherwise:

For the samples shown in Fig. 9, the MLA is around 70 %, which is an information to be considered together with the one coming from the SSM.

The values shown in Fig. 9 are computed using local ENU coordinates; it is sufficient to translate them into LLA to get a plot in geodetic coordinates. Fig. 10 and Fig. 11 show two samples randomly chosen within the dataset.

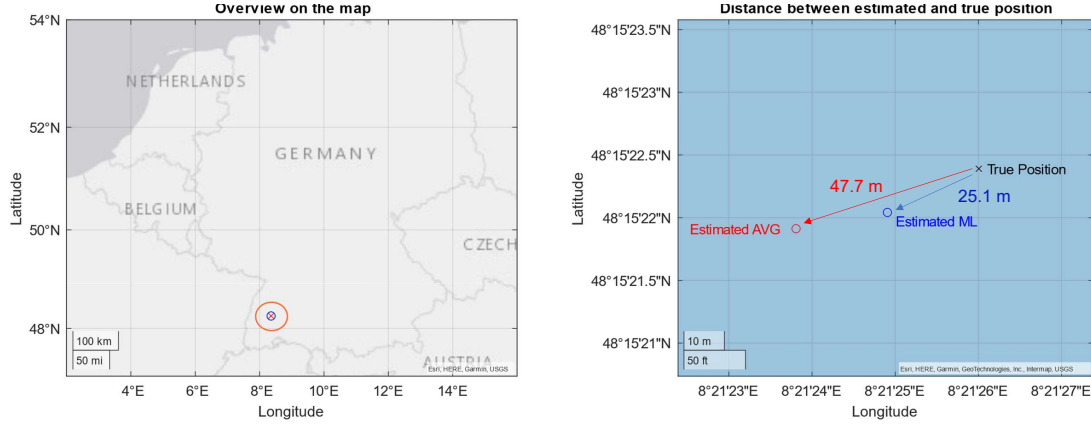


Figure 10: Sample #25: distance from the true position is 47.65 m when using reference simplified model and 25.08 m when using XGBoost model

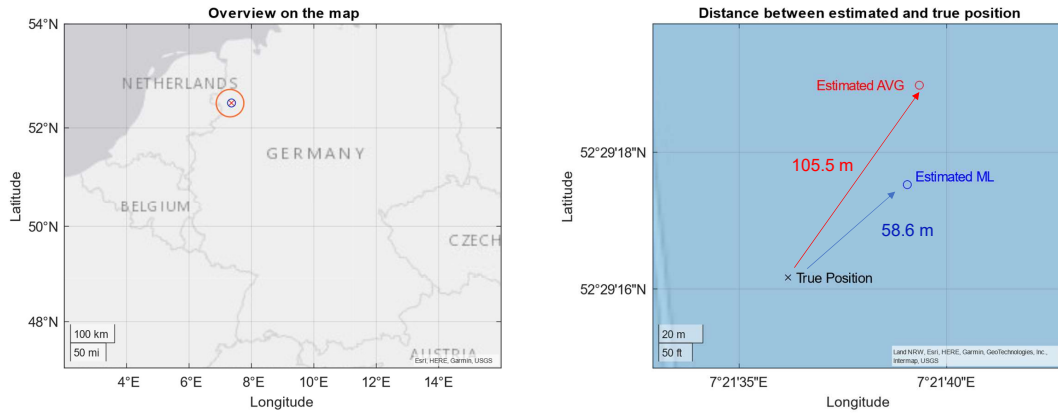


Figure 11: Sample #150: Estimation error (w.r.t. true position) 105.46 m when using reference simplified model and 68.58 m when using XGBoost model

With reference to SSM, the estimation error on the IoT receiver position obtained using machine learning tools is lower, i.e., 10 to 20 % than the estimation error showed by the adopted reference model. This observation is further assessed through a validation of the models, over geographic regions different from the one used to generate the training dataset. Concerning, instead, the magnitude of the target estimation error, the position estimates show on average a RMSE of about 50 m with respect to the true location of the target IoT receiver. Although such a distance is not negligible, and considering also that the study has been conducted in an experimental environment (with all the limitations that this entails), this preliminary study suggests that the collaborative state estimation based on the post-processing of collaborative GNSS data via machine learning could work and its use to support low-power GNSS usage in IoT devices is worth of further investigation.

V. CONCLUSIONS

The results presented in this work can pave the way for alternative GNSS receivers architectures where base-band or IF signal post-processing stages are heavily duty-cycled or switched-off, and machine learning engines are embedded to complement or replace them at a certain extent. The fact that both the ML algorithms provide results that are better than the average location of the networked receivers means that they can extract a certain extent of information.

ACKNOWLEDGEMENTS

This study was carried out within the Ministerial Decree no. 1062/2021 and received funding from the FSE REACT-EU - PON Ricerca e Innovazione 2014-2020. This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA. Association for Computing Machinery.
- Banville, S. and van Diggelen, F. (2016). Precision GNSS for everyone. *GPS World*, 27(11):43–48.
- Bembe, M., Abu-Mahfouz, A., Masonta, M., and Tembisa, N. (2019). A survey on low-power wide area networks for iot applications. *Telecommunication Systems*, 71.
- Caparra, G., Zoccarato, P., and Melman, F. (2021). Machine learning correction for improved pvt accuracy. In *Proceedings of the 34th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2021)*, pages 3392–3401.
- Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., et al. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4.
- Chollet, F. et al. (2015). Keras.
- Closas, P., Fernandez-Prades, C., and Fernandez-Rubio, J. A. (2007). Maximum likelihood estimation of position in gnss. *IEEE Signal Processing Letters*, 14(5):359–362.
- Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. <http://www.deeplearningbook.org>.
- Grenier, A., Lohan, E. S., Ometov, A., and Nurmi, J. (2023). A survey on low-power gnss. *IEEE Communications Surveys & Tutorials*, 25(3):1482–1509.
- Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR)*. [Internet], 9(1):381–386.
- Maimon, O. Z. and Rokach, L. (2014). *Data mining with decision trees: theory and applications*, volume 81. World scientific.
- Minetto, A., Bello, M. C., and Dovis, F. (2022). Dgnss cooperative positioning in mobile smart devices: A proof of concept. *IEEE Transactions on Vehicular Technology*, 71(4):3480–3494.
- Minetto, A., Nardin, A., and Dovis, F. (2019). Gnss-only collaborative positioning among connected vehicles. In *Proceedings of the 1st ACM MobiHoc Workshop on Technologies, MOdels, and Protocols for Cooperative Connected Cars*, TOP-Cars '19, page 37–42, New York, NY, USA. Association for Computing Machinery.
- Misra, P. and Enge, P. (2006). *Global Positioning System: Signals, Measurements and Performance Second Edition*. Lincoln, MA: Ganga-Jamuna Press.
- Nardin, A., Dovis, F., Valsesia, D., Magli, E., Leuzzi, C., Messineo, R., Sobreira, H., and Swinden, R. (2023). On the use of machine learning algorithms to improve gnss products. In *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 216–227. IEEE.
- Navarro, V., Grieco, R., Soja, B., Nugnes, M., Klotek, G., Tagliaferro, G., See, L., Falzarano, R., Weinacker, R., and VenturaTraveset, J. (2021). Data fusion and machine learning for innovative gnss science use cases. In *Proceedings of the 34th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2021)*, pages 2656–2669.

- Oliveira, L. L. d., Eisenkraemer, G. H., Carara, E. A., Martins, J. B., and Monteiro, J. (2023). Mobile localization techniques for wireless sensor networks: Survey and recommendations. *ACM Transactions on Sensor Networks*, 19(2):1–39.
- Wen, W., Bai, X., Zhang, G., Chen, S., Yuan, F., and Hsu, L.-T. (2020). Multi-agent collaborative gnss/camera/ins integration aided by inter-ranging for vehicular navigation in urban areas. *IEEE Access*, 8:124323–124338.