

**POLITECNICO DI TORINO**

**Master of Science in Mechatronic Engineering**



**Master's Degree Thesis**

**Machine Learning Techniques for  
Collaborative, Multi-agent GNSS  
Positioning in IoT devices**

**Supervisors**

**Dr. Alex MINETTO (DET)**

**Dr. Daniele JAHIER PAGLIARI (DAUIN)**

**Candidate**

**Angela ROTUNNO**

**October 2023**



# Summary

Nowadays, Global Navigation Satellite System (GNSS) receivers are embedded in a variety of electronic devices, and a growing number of users depend on them to navigate themselves to a destination. GNSS technology allows the user to estimate its position through multilateration, which is based on satellite detection, the estimation of the signal Time-of-Arrival (ToA), and the subsequent measurement of the receiver-to-satellite ranges (commonly referred to as “pseudoranges”). The low power of GNSS signals makes the continuous satellite’s signal tracking a challenging task for the receiver, especially considering the tight constraints on resource usage. Internet of Things (IoT) electronics host “by definition” low-power-consumption network connectivity, and they could enable new patterns for the Position Velocity and Time (PVT) estimation, based on collaborative, multi-agent Position Navigation and Time (PNT) methods that would not imply a continuous operation of the embedded GNSS receiver.

This study aims to understand whether Machine Learning (ML) techniques could support such paradigms for the PVT estimation in IoT devices, in the attempt to avoid the need for costly continuous signal tracking, demodulation of the navigation message, and pseudorange construction and correction, by exploiting the information made available by a set of networked collaborative users. In particular, we aim to share multi-satellite delay-Doppler matrices and their associated position estimates gathered by conventional GNSS receivers operating within a network.

The work has been developed in two different stages. The first step was generating an experimental environment, building a dataset inclusive of the IoT receiver’s and its surrounding networked receivers’ position information. In detail, we considered 4 networked receivers, randomly distributed within a 200m radius of the IoT receiver. For each receiver, the pseudoranges and the delay-Doppler matrices were simulated, as well as the PVT solutions. The next step was to use machine learning techniques to estimate the IoT receiver’s position, based on the information shared by the networked receivers. Two different machine learning open-source libraries have been tested: XGBoost (eXtreme Gradient Boosting), and Keras, to implement a

neural network (namely, a multi-layer perceptron). Finally, the estimation of the IoT position obtained using the two machine learning tools has been compared to a simplified “reference” model, where the IoT receiver’s position is approximated by the arithmetic average of the networked receivers’ positions.

The estimation error on the IoT receiver position obtained using machine learning tools is lower (typically, 10 to 20%) than the estimation error shown by the simplified reference model. This observation is confirmed when doing a further validation of the models, over geographic regions different from the one used to generate the training dataset. Concerning, instead, the positioning error, the position estimates are in general showing a 50m offset compared to the “true” position. Although such a distance is not negligible, and considering also that the work has been developed in an experimental environment (with all the limitations that this entails), this preliminary study suggests that the PVT estimation via machine learning could work, and its use in support of PVT estimation might be further investigated.

# Acknowledgements

Non sarei mai riuscita a chiudere questo lavoro senza l'aiuto e il sostegno di tante persone, che hanno fatto e fanno tutti i giorni la differenza.

Per primi, devo ringraziare i miei relatori, dr. Alex Minetto e dr. Daniele Jahier Pagliari, per essere stati sempre presenti, attenti e molto pazienti: grazie di cuore!

Grazie ai bravi insegnanti che ho avuto la fortuna di incontrare nel corso della mia carriera (anche precedente), per tutto il lavoro che sta dietro una singola ora di lezione, per l'entusiasmo che hanno saputo trasmettermi, e soprattutto, grazie per avermi resa una persona meno incolta.

Grazie a mamma e a papà, a fratm (che sarà qui per la cerimonia!), grazie a Chiara e alle nostre gite EE(A) in montagna, e a Patrick per i bei vecchi tempi insieme.

Grazie a Denise, la persona più autentica che io conosca, e grazie al super team TAT e a tutti i colleghi pettegoli e polemici che rendono divertenti anche le giornate di lavoro devastante.

Grazie al caro vecchio QBO e alla McFit, per avermi aiutata a mantenere l'equilibrio mentale. Grazie alla bellissima Torino, che è diventata la mia casa, e a questa regione meravigliosa.



# Table of Contents

<b>List of Tables</b>	VIII
<b>List of Figures</b>	IX
<b>Acronyms</b>	XIII
<b>1 Introduction</b>	1
<b>2 Background</b>	3
2.1 Global Navigation Satellite Systems . . . . .	3
2.1.1 GNSS receiver operations and architecture . . . . .	3
2.1.2 GNSS constellations and signals . . . . .	5
2.1.3 GNSS observables and PVT estimation . . . . .	8
2.1.4 Baseband signal processing . . . . .	10
2.2 Machine learning tools . . . . .	14
2.2.1 Gradient-boosted decision tree (XGBoost library) . . . . .	15
2.2.2 Neural Networks (Keras library) . . . . .	17
<b>3 Methodology, design and set-up of the experimental scenario</b>	20
3.1 Experimental environment . . . . .	22
3.1.1 Generation of the GNSS constellation . . . . .	22
3.1.2 Definition of the position of the IoT receiver and generation of the networked receivers . . . . .	24
3.1.3 Computation of the pseudoranges and pseudoranges' rates based on visible satellites . . . . .	27
3.1.4 Processing of the dataset before data export . . . . .	29
3.2 Machine Learning . . . . .	31
3.2.1 XGBoost . . . . .	34
3.2.2 Keras . . . . .	40

<b>4 Results</b>	43
4.1 Validation of ML models . . . . .	43
4.2 Position Estimation . . . . .	48
<b>5 Conclusions</b>	51
<b>A GPS signal parameters</b>	52
<b>B Reference Frames</b>	53
<b>C Geometric Dilution of Precision</b>	56
<b>D Matlab function lla2enu.m</b>	60
<b>E Matlab function lookangles.m</b>	64
<b>F Matlab function pseudoranges.m</b>	66
<b>G Alternative calculation of pseudoranges and pseudoranges rates</b>	70
<b>H Check of the integer number of code repetition</b>	72
<b>Bibliography</b>	78



# List of Tables

3.1	Attributes of the Receivers . . . . .	26
3.2	Data export; networked receivers are numbered 1 to 4, while IoT is identified as receiver 5 . . . . .	30
3.3	XGBoost Hyperparameters . . . . .	34
4.1	Edges (Lat, Lon) of the "Spanish" area . . . . .	45
4.2	Edges (Lat, Lon) of the "Polish" area . . . . .	45
4.3	Edges (Lat, Lon) of the "Congolese" area . . . . .	46
4.4	Edges (Lat, Lon) of the "Russian" area . . . . .	46
A.1	GPS code information . . . . .	52

# List of Figures

2.1	GNSS receiver: block diagram of a conventional architecture . . . .	4
2.2	GNSS constellations and their features: GPS, Galileo, GLONASS and BeiDou . . . . .	6
2.3	Exemplification of the GPS signal components . . . . .	7
2.4	A simple geometry: localization of an object in a 2-dimensional space. The ambiguity on the position in the case only 2 emitters are available (left) is resolved when a third emitter is available (right) . . . . .	8
2.5	Signal travelling from the satellite to the user . . . . .	9
2.6	Example of the equations to solve the positioning problem (multilateration) using 4 satellites . . . . .	10
2.7	Signal correlation: the code replica is shifted with respect to the incoming signal until a peak is found at the correlation output . . . . .	11
2.8	Example of Cross Ambiguity Function over the delay - Doppler domain ( <i>search space</i> ) . . . . .	12
2.9	Schematic description of the operations done at tracking stage . . . . .	13
2.10	Tracking stage: carrier and code loops . . . . .	13
2.11	Type of problem: regression (left) vs. classification (right) . . . . .	15
2.12	Example of gradient-boosting decision tree architecture . . . . .	16
2.13	A single-neuron network . . . . .	18
2.14	Exemplification of a multi-layer perceptron . . . . .	19
3.1	GNSS Receiver: use of machine learning to support PVT estimation . . . . .	20
3.2	Work flowchart . . . . .	21
3.3	Matlab semread_function: parameters and description . . . . .	23
3.4	Edges (Lat, Lon) of the area where the scenario is developed . . . . .	24
3.5	Distribution over Germany of the dataset and detail of a single set including the IoT receiver ("IoT") and the 4 surrounding networked receivers ("Rec") . . . . .	25
3.6	Example of mask angles: (a) distribution of the satellites with the elevation angles and (b) LOS and NLOS satellites for the user . . . . .	27

3.7	Illustration of code replicas meaning: for a receiver that has in view $n$ satellites, the integer number of code repetitions for all of these satellites must be the same . . . . .	29
3.8	Distance from the true position of the IoT receiver using LLA coordinates as output target and no normalization: the estimation is totally out of span. . . . .	33
3.9	Distance from the true position of the IoT receiver using ECEF coordinates as output target and no normalization: the estimation is still out of span. . . . .	33
3.10	Distance from the true position of the IoT receiver using normalized ECEF coordinates as output target: the estimation is within the correct range ( $< 200m$ ). . . . .	33
3.11	Illustration of data split for XGBoost - use of Optuna library for hyperparameters optimization . . . . .	35
3.12	Illustration of cross validation for XGBoost . . . . .	36
3.13	Incorrect use of machine learning techniques: average position predicted by XGBoost is extremely close to the real position (zero) only due to data overfitting (training dataset = validation dataset) . . .	37
3.14	Hyperparameter importance in Optuna . . . . .	38
3.15	Results over training and validation dataset with XGBoost . . . . .	39
3.16	Results when using 4 hidden layers, with 20 nodes each. The behavior of loss and validation loss over the iterations show a high level of overfitting on the training dataset . . . . .	41
3.17	Results when using no hidden layers. The ability of learning of the network is negligible (underfitting) . . . . .	41
3.18	Results when using one hidden layer with 5 nodes. Learning and prediction are well balanced . . . . .	41
4.1	Comparison of results from XGBoost and Keras over 10 additional validation sets created in the German area . . . . .	43
4.2	Distribution of the sample points generated over different geographic areas with respect to the experimental scenario: "Polish" and "Spanish" areas . . . . .	45
4.3	Distribution of the sample points generated over different geographic areas with respect to the experimental scenario: "Russian" and "Congolese" areas . . . . .	46
4.4	Keras model is tested on the different validation dataset; the prediction seems strongly improved when moving right below the Equator	47
4.5	Keras script is validated on Spanish and Polish sets, then validated on other regions . . . . .	47

4.6	Distance from true IoT position based on simplified average model (red) and ML model (blue) . . . . .	49
4.7	Sample A: distance from the true position is $47.65m$ when using reference simplified model and $25.08m$ when using XGBoost model . . . . .	49
4.8	Sample B: distance from the true position is $29.02m$ when using reference simplified model and $15.52m$ when using XGBoost model . . . . .	50
4.9	Sample C: distance from the true position is $105.46m$ when using reference simplified model and $68.58m$ when using XGBoost model . . . . .	50
4.10	Sample D: distance from the true position is $67.62m$ when using reference simplified model and $57.15m$ when using XGBoost model . . . . .	50
B.1	Latitude and Longitude . . . . .	53
B.2	Geopotential surface for Earth Gravitational Model (EGM 96) . . . . .	54
B.3	Parameters of WGS 84 . . . . .	54
B.4	Geodetic, ENU and ECEF coordinates . . . . .	55
C.1	Example of high and low GDOP for different emitters geometries . . . . .	58



# Acronyms

**IoT**

Internet of Things

**ToA**

Time-of-Arrival

**GNSS**

Global Navigation Satellite System

**PVT**

Position Velocity and Time

**PNT**

Position Navigation and Time

**ML**

Machine Learning

**TEC**

Total Electron Content

**LOS**

Line Of Sight

**NLOS**

Non Line Of Sight

**PRN**

Pseudo-Random Noise

**SIS**

Signal In Space

**LLA**

Latitude Longitude Altitude

**ENU**

East North Up

**NED**

North East Down

**ECEF**

Earth Centered Earth Fixed

**EGM**

Earth Gravitational Model

**WGS**

World Geodetic System

**CAF**

Cross Ambiguity Function

**GLONASS**

GLObal NAvigation Satellite System

**GPS**

Global Positioning System

**ESA**

European Space Agency

**CDMA**

Code Division Multiple Access

**FDMA**

Frequency Division Multiple Access

**PLL**

Phase Lock Loop

**FLL**

Frequency Lock Loop

**DLL**

Delay Lock Loop

**UTC**

Universal Time Coordinated

**ReLU**

Rectified Linear Unit

**SSM**

Single-Scalar Metric

**MLA**

Machine Learning Asset

**USERE**

User Equivalent Range Error

**GDOP**

Geometric Dilution of Precision

**PDOP**

Position Dilution of Precision

**TDOP**

Time Dilution of Precision

**HDOP**

Horizontal Dilution of Precision

**GBDT**

Gradient-boosting Decision Tree



**MLP**

Multi-Layer Perceptron

**SGD**

Stochastic Gradient Descent

# Chapter 1

## Introduction

The purpose of a GNSS receiver is to determine the user's Position Velocity and Time (PVT) by processing the signals transmitted by satellites. To do that, the receiver must be capable of continuously acquiring and tracking the signals from the satellites in view, then demodulate and process these signals, to translate them into meaningful information for the user.

This work will be especially focused on the determination of the receiver's position. GNSS technology allows the user to estimate its position by means of multilateration, which is based on the satellite detection, the estimation of the signal Time-of-Arrival (ToA) and the subsequent measurement of the receiver-to-satellite ranges. These latter measurements are commonly affected by the misalignment between on-board satellites clocks and receivers' local oscillator, for this reason they're named *pseudoranges*.

The low power of GNSS signals makes the continuous satellite's signal tracking a demanding task for the receiver, especially considering the tight constraints on resources' usage. The reduction of the computational load on the device hosting the receiver and, more in general, power-saving techniques, represent a significant topic when dealing with GNSS receiver intended for a mass market audience[1]. In the last couple of decades, strategies for power saving have been explored with special reference to smartphones, with solutions going from powering on and off the chip [2], to switching to alternative location sensors [3], to reducing the number of GNSS receiver channels [4].

On top of that, the recent growth of user applications requiring small size, low cost, low power consumption devices, is making even more challenging the implementation of GNSS receivers, pushing the current technology to its limits, and the search for new approaches to positioning is increasing in importance [5]. Along with studies

intended to implement new hardware solutions for extra-small devices, such as, for instance, wearables [6], there are others focusing on the possibilities offered by Cloud computing platforms [7].

IoT electronics host “by definition” low power-consumption network connectivity, and they could represent the ideal application to enable new patterns for the PVT estimation, by mean of collaborative, multi-agent PNT methods that would not imply a continuous operation of the embedded GNSS receiver. The use of ML techniques could hence be an efficient solution to support such paradigms for the PVT estimation in IoT devices.

As a matter of fact, in the last decades, ML tools have been successfully adopted across a large number of disciplines, going from engineering to finance to healthcare, and have become part of our everyday life. Tasks like image recognition, speech recognition, fraud detection, but also medical diagnosis and stock market forecasting are enhanced by machine learning tools [8]. The significant advantage of these approaches with respect to traditional methods of data processing is in their ability to *learn* from a given set of data, detecting patterns that can be used to forecast and / or detect a specific behavior on different (or future) sets of data. When it comes to IoT devices, the growing number of computing applications relying on them is constantly pushing towards the use of ML models, with a special regards to all those tools and solutions meant to combine accuracy with low energy consumption[9].

In GNSS domain, there have been different studies exploring the potential of machine learning methodologies, ranging from the forecast of atmospheric phenomena that have an impact on the signals [10] [11] to the improvement of the accuracy of the PVT solution in urbanized areas [12] [13]. In this study, the aim is to understand if and in which extent ML techniques could be exploited to support the position estimation for IoT devices. Previous works have already explored the possibilities of collaborative methods for positioning, with the aim to improve the positioning performance in case of limited availability and lack of continuity of the GNSS service[14].

After an introductory chapter concerning GNSS receivers and ML techniques (Chapter 2), the methodology, the design and set up of the experimental environment will be displayed (Chapter 3), followed by the results on ML models and position estimation (Chapter 4).

# Chapter 2

## Background

### 2.1 Global Navigation Satellite Systems

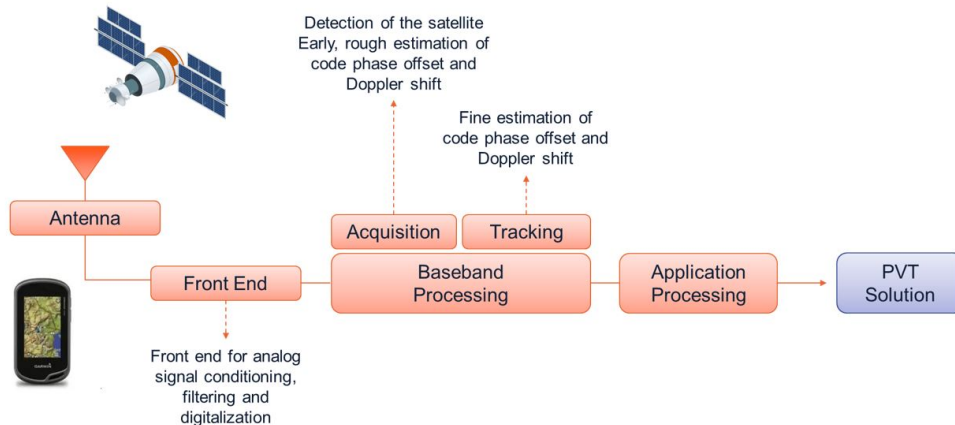
This section is dedicated to the illustration of the basic concepts concerning GNSS. Due to the extent of the topic, the intent here is to give a high-level overview on how a conventional receiver works, with a special reference to the methodology used to estimate the user's position based on satellites' signals detection. A thorough description of GNSS principles and applications can be found in [15].

#### 2.1.1 GNSS receiver operations and architecture

For a GNSS receiver, the basic concept used to establish the user position is founded on measuring the time it takes for the signal transmitted by the satellite, whose location is known, to reach the receiver. From a high-level point of view, the user's PVT solutions are the result of a sequence of operations, which can be described as follows:

1. identification of the satellite;
2. estimation of the Time-of-Arrival (e.i., the time instant when the signal broadcast by the satellite reaches the user);
3. construction of the receiver-to-satellite ranges (commonly referred to as "pseudoranges");
4. estimation of the PVT solutions.

These operations are carried out at different unit stages within the receiver. Figure 2.1 shows a conventional GNSS receiver architecture [16]:



**Figure 2.1:** GNSS receiver: block diagram of a conventional architecture

With reference to Figure 2.1, four main functions can be identified:

- Antenna and front-end processing, where the received signals are amplified, filtered and then digitalized;
- Acquisition, where the satellite is identified and a rough, early estimation of code phase offset and Doppler shift is performed;
- Tracking, where there is a refinement of code phase offset and Doppler shift;
- Application processing, where the navigation message demodulation is completed, followed by pseudorange construction and estimation of the PVT solution.

Each satellite within a GNSS constellation has a unique Pseudo-Random Noise (PRN) code, that is broadcast as part of the navigation message (see subsection 2.1.2). This code allows any receiver to identify exactly which satellite is 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 digital stream of samples.

Baseband processing, instead, include all those algorithms required to detect and follow a visible GNSS signal. The operations here are identified as "acquisition" and "tracking".

The acquisition stage has the overall function 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 initiated. The tracking stage uses 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 acquisition and tracking operations for a conventional GNSS receiver are described in subsection 2.1.4.

The output of the baseband processing is a pair of *code delay* and *Doppler phase or frequency*, which represent the actual measurement of a GNSS receiver. The replica code phase is converted into satellite transmit time, required to compute the pseudorange measurement, while the replica carrier Doppler phase or frequency is converted into delta pseudorange. Pseudorange and pseudorange rate are then used to estimate the PVT solutions (see subsection 2.1.3).




## 2.1.2 GNSS constellations and signals

A Global Navigation Satellite System provides signals from a constellation of satellites, which are distributed in such a way to guarantee an almost global coverage on the Earth surface. There are four sets of GNSS constellations:<sup>1</sup>

- GPS (Global Positioning System), owned and operated by the United States Government;
- Galileo (named after the Italian astronomer *Galileo Galilei*), created upon the initiative of the European Union (EU) and the European Space Agency (ESA);
- GLONASS (GLOBAL NAVIGATION Satellite System), developed and operated by the Russian Federation Government;
- BeiDou (named after the Great Bear Constellation), developed and operated by the Chinese Popular Republic.

---

<sup>1</sup>In addition to the GNSS systems listed above, other two can be mentioned, even though they're not *global* systems: the QZSS (*Quasi-Zenith Satellite System*), operated by the Japanese government, that only covers the Asia-Oceania regions, and the NavIC (*Navigation with Indian Constellation*), developed by the Indian government, that only covers the Indian region.

System	GPS 	Galileo 	GLONASS 	BeiDou 
<b>Owner</b>	United States	European Union	Russian Federation	China
<b>Semi-Major Axis</b>	26560 km	29994 km	25508 km	42164 km GEO 42164 km IGSO 27878 km MEO
<b>Orbital Altitude</b>	20180 km	23222 km	19130 km	21150 km
<b>Period</b>	11h 58min	14h 08min	11h 16min	12h 38min
<b>Number of satellites</b>	31 (24 by design)	27 (plus 3 spares)	28 (24 by design)	5 GEO 2 IGSO 27 MEO
<b>Frequency</b>	1575.42 MHz (L1) 1227.60 MHz (L2)	1164-125 MHz (E5a-E5b) 1260-1300 MHz (E6) 1559-1592 MHz (E2-L1-E11)	1602 MHz (SP) 1246 MHz (SP)	1561.098 MHz (B1) 1589.742 MHz (B1-2) 1207.14 MHz (B2) 1268.52 MHz
<b>Number of orbital planes</b>	6	3	3	1 GEO 3 IGSO 3 MEO
<b>Orbit inclination</b>	55°	56°	64.8°	0° GEO 55° IGSO 55° MEO

**Figure 2.2:** GNSS constellations and their features: GPS, Galileo, GLONASS and BeiDou

Figure 2.2 shows the difference in GNSS features, like the orbital altitude, the period, the number of satellites and the frequency of the signal <sup>2</sup>. The reference constellation for this work is the GPS; it is understood that the same approach can be applied to any of the other systems.

The signals transmitted by the satellites must carry specific data, such that for the user it is possible to identify each satellite in a unique way. One of the most common technique for GNSS to allow the satellites' identification without ambiguity is using mutually orthogonal codes. In this way, the receiver can easily separate the signal of the satellite of interest from the others.

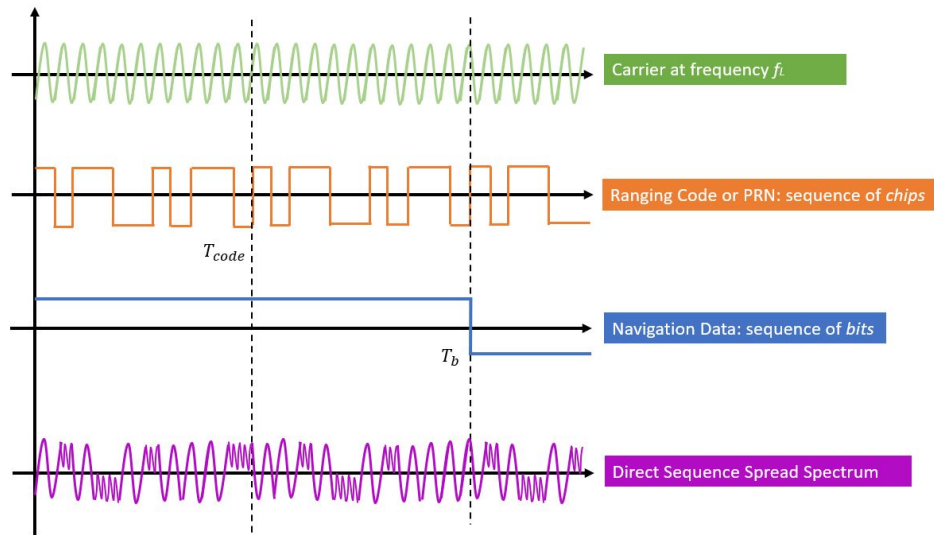
For legacy GPS, the signal in space consists of 3 components:

1. Carrier: a radio frequency sinusoidal signal;
2. Ranging Code: a sequences of zeroes and ones, which allow the receiver to determine the travel time of the radio signal from the satellite to the receiver. They are called Pseudo-Random Noise sequences or PRN codes;

<sup>2</sup>Referenced information have been collected from <https://gssc.esa.int/navipedia>.

- Navigation Data: a binary-coded message, carrying information on the satellite ephemeris <sup>3</sup>, clock bias parameters, almanac, satellite health status, and other information.

The result of the modulation of such signal is the Direct Sequence Spread Spectrum.



**Figure 2.3:** Exemplification of the GPS signal components

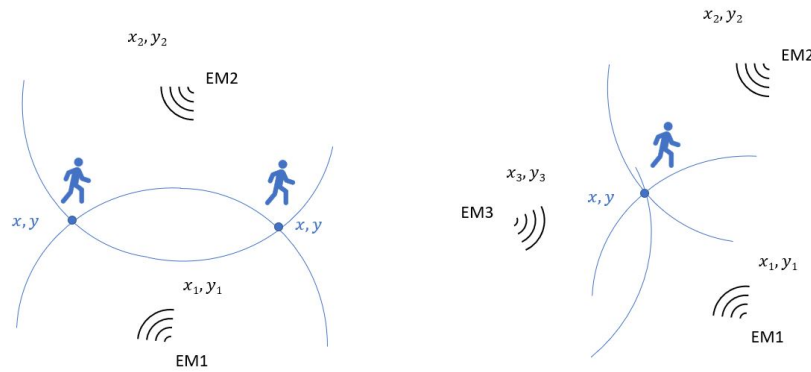
The transmission of multiple PRN signals on a common carrier frequency is referred to as Code Division Multiple Access (CDMA). A different technique, referred to as Frequency Division Multiple Access (FDMA) consists, instead, in assigning each satellite with a specific carrier frequency. As a note, the main advantage of FDMA when compared to CDMA is that it guarantees signal separation, since each signal is transmitted in a dedicated frequency slot. On the other hand, it requires a higher complexity (and cost) regarding antenna and receiver design.

<sup>3</sup>The ephemeris are a collection of Keplerian elements or satellite position and velocity



### 2.1.3 GNSS observables and PVT estimation

Based on their geometry, Global Satellite Systems can be classified as *spherical systems*, where the sources of the signals (i.e., the satellites) are placed at known locations in the space and the user's position can be obtained by intersection of spheres. The estimation of the position is done by the receiver via *multilateration*. From a strictly geometrical point of view, in a 2-dimensional space, the coordinates of an object can be computed without ambiguity if at least 3 reference emitters' locations are known: the user's position will be computed as the intersection of the 3 circumferences.



**Figure 2.4:** A simple geometry: localization of an object in a 2-dimensional space. The ambiguity on the position in the case only 2 emitters are available (left) is resolved when a third emitter is available (right)

However, when dealing with satellite navigation, not only the circumferences are replaced by spheres, but there is a further unknown to be taken into account: time. Let's assume, for instance, that a satellite transmits a signal at time instant  $T_{tx}$ . The same signal will be received by the user at time  $T_{rx} = T_{tx} + \tau$ , where  $\tau$  is the *delay* or *propagation time*. The distance between the transmitter and the receiver can be thus estimated as:

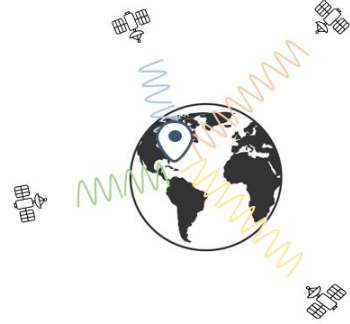
$$d = c \cdot (T_{rx} - T_{tx}) = c \cdot \tau, \quad (2.1)$$

where  $c$  is the speed of light.

The 2.1 holds true only if both the transmitter's and the receiver's clocks are synchronous. Given the magnitude of the speed of light constant, any gap between the two would result in a non-negligible error on the final distance.



$$\begin{aligned}\rho_1 &= \sqrt{(x_1 - x_{rec})^2 + (y_1 - y_{rec})^2 + (z_1 - z_{rec})^2} + b_{rec} \\ \rho_2 &= \sqrt{(x_2 - x_{rec})^2 + (y_2 - y_{rec})^2 + (z_2 - z_{rec})^2} + b_{rec} \\ \rho_3 &= \sqrt{(x_3 - x_{rec})^2 + (y_3 - y_{rec})^2 + (z_3 - z_{rec})^2} + b_{rec} \\ \rho_4 &= \sqrt{(x_4 - x_{rec})^2 + (y_4 - y_{rec})^2 + (z_4 - z_{rec})^2} + b_{rec}\end{aligned}$$



**Figure 2.6:** Example of the equations to solve the positioning problem (multilateration) using 4 satellites

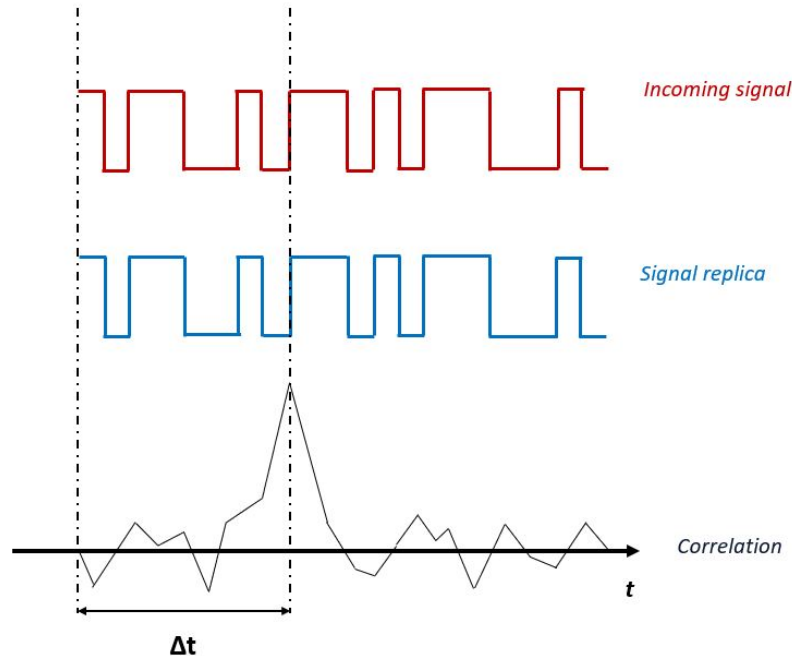
To be able to estimate its position, a receiver must have at least 4 satellites *in view*, which means that the satellites must be necessarily in Line Of Sight (LOS), e.i., there must be no obstacles in between the user and the satellite (buildings, trees, etc.) - see Figure 3.6. If a larger number of satellites is in view, a better estimation is possible. Modern receivers use at least to 12 channels in order to perform the position estimation. Appendix C illustrates further details to understand the importance of how the satellites' geometry with respect to the user can affect the estimation of the position.

### 2.1.4 Baseband signal processing

Code delay and Doppler frequency or phase represent the natural measurement of a GNSS receiver, made available at the output of the baseband processing operations. The baseband operations are divided into "acquisition" and "tracking" stages. In particular, the receiver is said to be in acquisition mode when searching for signals, and then transits to tracking mode once the signal is found. The basic principle of GNSS baseband processing is founded on the correlation process, made possible by the satellites' signals design (see subsection 2.1.2). The ranging codes are built to have:

- high auto-correlation properties: when the code is compared with an *aligned* replica of itself, the correlation output is maximum;
- low cross-correlation properties: when the code is correlated with another code of the same family, the correlation output is low.

Figure 2.7 shows an exemplification of the correlation process. The receiver shifts the code replica (by changing  $\Delta t$ ) until it finds a peak in the correlation output.

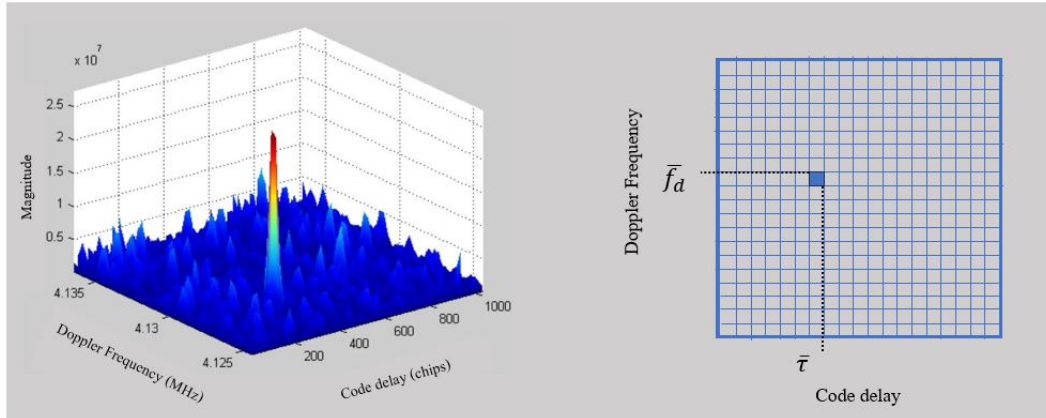


**Figure 2.7:** Signal correlation: the code replica is shifted with respect to the incoming signal until a peak is found at the correlation output

In detail, the receiver first assigns each channel with a PRN code. Then, a local replica is generated, in such a way that its code delay and phase characteristics vary. The correlation function, referred to as Cross Ambiguity Function (CAF), will hence vary over a two-dimensional *search space*, defined by code delay offset and Doppler shift. Over the search space, the value of the CAF is compared against appropriate thresholds to understand whether the  $i^{th}$  satellite is in view or not. Figure 2.8 shows an example of CAF over the search space <sup>4</sup>.

When a channel is first set up, usually there are no estimates available for code delay and Doppler phase or frequency. Therefore, each channel will launch a *cold* search for the signal (see more at pp.231-232 [15]). When in acquisition mode, each channel is looking for all possible pairs of code delay, Doppler frequency, by

<sup>4</sup>The figure of the CAF is extracted from [17]

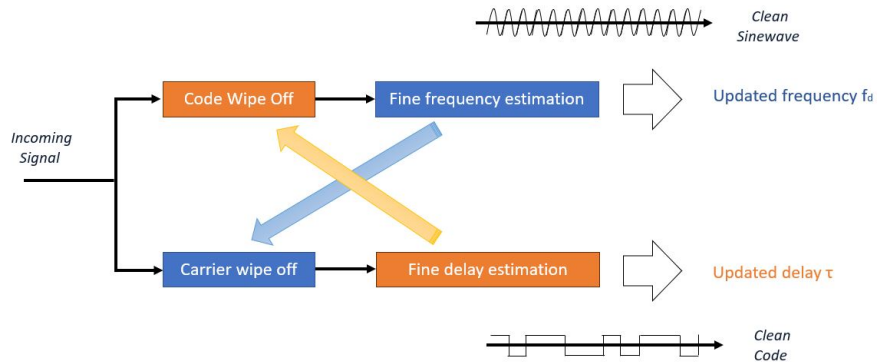


**Figure 2.8:** Example of Cross Ambiguity Function over the delay - Doppler domain (*search space*)

generating a set of possible local replicas, and correlating each of them with the incoming signal. In this context, the power of the correlation output is somehow a measure of how close to the real signal the estimates of the code delay and carrier phase are. It must be considered that real-life systems are very noisy, and as a consequence it is complex not only to detect the peak in CAF, but also to define an appropriate threshold to assess whether the signal is present or not (see more at pp. 219-223 [15]).

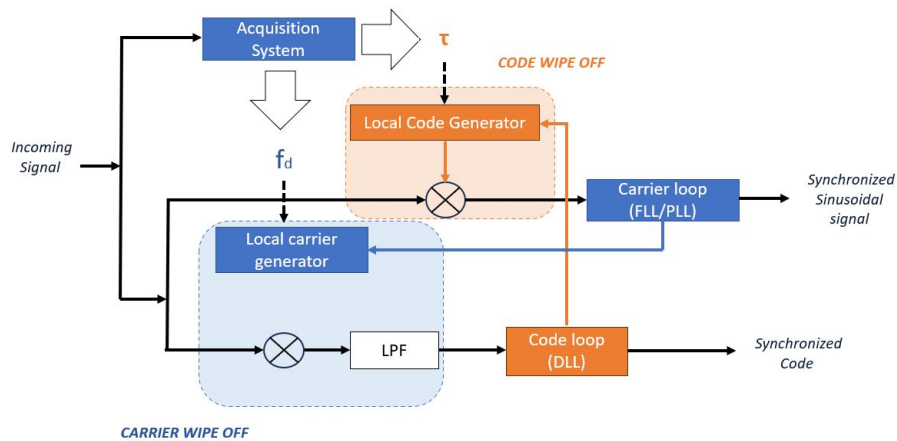
Once the satellite is identified and a earlyt, rough estimation of the code delay and Doppler shift are available, the signals processing moves to the tracking stage. Here, the incoming signal is first cleared of its Doppler frequency (according to the current estimation), and then compared with PRN code replicas generated locally (according to the current estimation of code delay). At each iteration, based on the output of this correlation process, the estimators provide fresh values of Doppler and delay to the wiping blocks, which progressively improve the quality of the signals. The patterns are usually replicated over different channels, and each of these processes a given signal from a given satellite. A high level description of the operations performed on a single channel is shown in Figure 2.9.

A good estimation of the residual carrier frequency and phase is not possible until the code is wiped off; at the same time, a good estimation of the code delay is not possible until the residual modulation is present. For this reason, the architecture of the tracking stage is commonly built using feedback loops on both carrier frequency / phase and code delay.



**Figure 2.9:** Schematic description of the operations done at tracking stage

The carrier tracking loop is meant to refine the estimation of the frequency and the phase of a noisy sinusoidal signal, and to track the frequency changes while the satellite is moving. To this purpose, the Phase Lock Loop (PLL) or Frequency Lock Loop (FLL) are employed. For what concerns instead the Delay Lock Loop (DLL), the delay information is contained in the correlation peak. The DLL is based on two correlators and two local replicas of the code. The signal at the input of the DLL requires that the residual Doppler modulation has been almost totally removed.



**Figure 2.10:** Tracking stage: carrier and code loops

## 2.2 Machine learning tools

The core of the work is based on the use of machine learning techniques to process data from the experimental scenario and estimate the IoT receiver position based on the information shared by the networked receivers. More precisely, the idea is to understand if and how a machine learning approach could improve the estimation of the IoT receiver position with respect, for instance, to an approximation calculated from the average of the networked receivers' positions.

As already mentioned in section 1, ML represents an expanding branch of computational algorithms and has become part of our everyday life, being applied successfully in diverse fields, ranging from pattern recognition, to finance, to medical applications and so on. The designation of Machine Learning gathers a complex of different tools, which all have in common the ability to *learn* from the surrounding environment.

Based on the approach, ML problems can be classified in *supervised learning*, *unsupervised learning* and *reinforcement learning*.

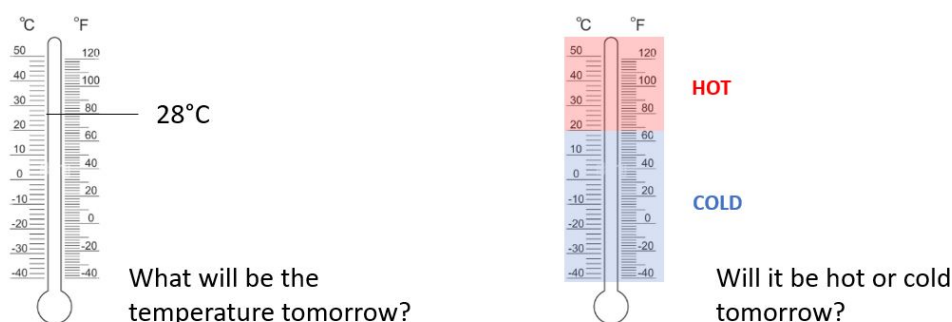
Supervised learning will map an input to an output based on a known dataset, inclusive of input-output pairs "examples". The input-output relation can be found by defining some *loss function* to measure the error between the model's prediction and the "true" value. The algorithm will be set in such a way that the loss function is minimized. This is the case of the work discussed in this thesis: we build a dataset, comprehensive of both input and output features' measurements, and we aim to find models to estimate the output on new datasets. As explained in detail in section 3.2, when approaching a supervised learning problem it is essential to divide the initial dataset into *train*, *validation* and *test* dataset. Using the whole data to build the model could result in a good data-fitting on the current dataset, while we need to assess also the generalization of the model on new data.

As opposite to the supervised learning, there are cases where the input features' measurements are the only available data (i.e., no information on the output), and the task is to describe how the data are organized or clustered. In this case, we talk about unsupervised learning, since *there are no correct answers, and there is no teacher* [18]. It will be the algorithm to suggest how the data are meant to be structured. When new data is introduced, the features already learned will be used to address them to the appropriate class.

Reinforcement learning is a self-teaching approach that learns from its own experience. At every iteration, the decisions are taken based on previous feedback that

represent either *rewards* or *penalties*. The objective is to maximize the cumulative reward over time.

For supervised learning problems, a further distinction can be done based on the expected output, depending on whether we need to predict continuous numerical values or to assign a certain label or attribute the output variable. In the first case, the task is to address a *regression* problem, while in the second case, we aim to solve a *classification* problem. A simple example would be, to forecast the temperature. We might be interested in knowing what the temperature will be (numeric value), or whether the weather will be "hot" or "cold" (attribute) - see Figure 2.11.



**Figure 2.11:** Type of problem: regression (left) vs. classification (right)

Based on the considerations done so far, the problem we're going to solve is a supervised regression problem where we're dealing with numeric features. Amongst the possible tools available, we have chosen to test two open-source libraries: XGBoost (*eXtreme Gradient Boosting*)<sup>5</sup> and Keras<sup>6</sup>.

### 2.2.1 Gradient-boosted decision tree (XGBoost library)

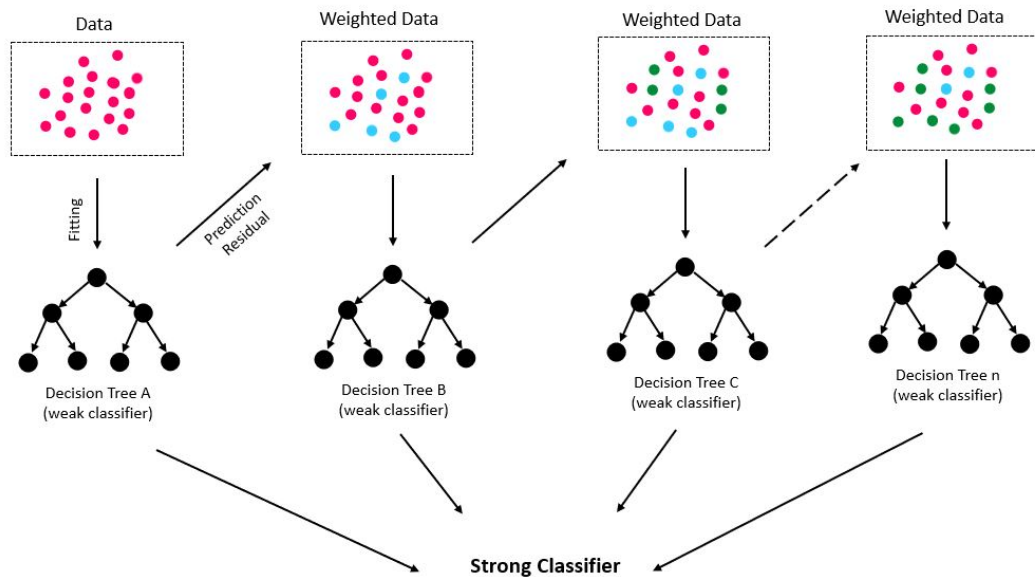
XGBoost is a gradient-boosted decision tree machine learning library. It implements parallel tree boosting and can be used not only for regression, but also for classification problems. In the past years, it has gained popularity for being the winning solution of an important ML challenge[19], and has since then been used for a large number of competitions.

<sup>5</sup>Free download at <https://anaconda.org/anaconda/py-xgboost>

<sup>6</sup>Free download at <https://anaconda.org/conda-forge/keras>



In all supervised learning problems, the goal is to find the relation that best approximates the output variables from the values of input variables, exploiting the information available in the train dataset. The gradient boosting technique is based on the idea of “boosting” or improving a single, weak model, by combining it with a number of other weak models, to generate a collectively strong model - see Figure 2.12.



**Figure 2.12:** Example of gradient-boosting decision tree architecture

The Gradient-boosting Decision Tree (GBDT) method seeks an approximation in the form of a weighted sum of functions, which are called *weak* learners. The method tries to find an approximation that minimizes the average value of the loss function on the training set, by applying the steepest descent step to this minimization problem. The training is done on an ensemble of weak decision trees, with each iteration using the error residuals of the previous model to fit the next model. The final prediction will then be a weighted sum of all the "weak" trees predictions.

The working principle of the GBDT method is founded on the basic concepts of decision trees and ensemble learning.

A decision tree is a classifier expressed as a recursive partition of the instance space.

The structure evolves in the fashion of a growing tree, and is made-up of *nodes* (features), *branches* (decisions) and *leaves* (outcome). The "root" node represents the feature identified as the most significant of all when starting the process of decision-making. Due to the fact that its structure recalls a flowchart diagram (and due to the analogy with the logical thinking), the decision tree method can be very intuitive to comprehend. The problem, however, is that this approach is extremely sensible to data and generally shows good fitting on the training dataset, as opposite to a poor accuracy on a test dataset. *Ensemble* learning is one of the techniques used to reduce this undesired behavior. Through the combination of a number of different models, an ensemble learning will result in a better performance on new data. The two most popular ensemble learning methods are *bagging* and *boosting*:

- with bagging, several models are trained in parallel, and each of them learns from a random portion or subset of the data;
- with boosting, different models are trained in sequence, and the residual error of the previous model is exploited to fit the next model.

GBDT algorithms leverage different *hyperparameters* that, appropriately tuned, will make the difference on the performance and the accuracy of the model. Two of the most significant hyperparameters are *learning rate* and the *number of estimators*. The learning rate denotes how quickly the model learns: the lower the learning rate, the better the model will perform, even though the decision process will take longer. The number of estimators, instead, is nothing but the number of trees used in the model. As a note, if the learning rate is low, probably more trees will be required to train the model; at the same time, using too many trees can lead to overfitting on the training dataset (with consequent poor fitting on new data), which is the the opposite of what we're trying to do when applying ensemble methods to decision trees.

On top of GBDT algorithms, XGBoost implements a number of optimizations, that can be found described in the documentation <sup>7</sup>.

## 2.2.2 Neural Networks (Keras library)

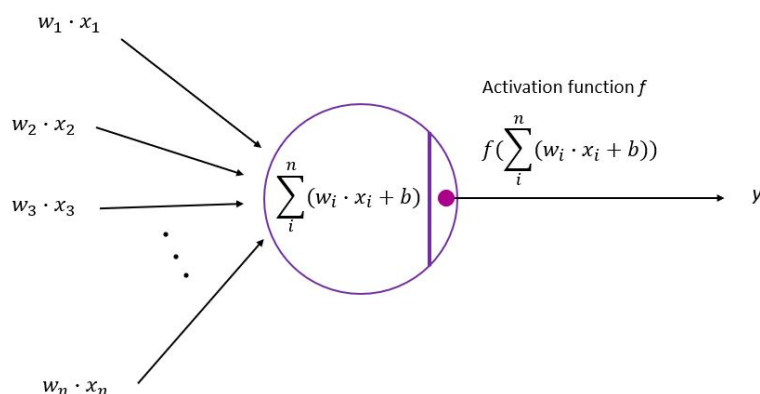
Keras is a library that eases the definition, training and evaluation of neural network models. Neural networks can be considered a *universal approximator*, due to their ability to reproduce and model nonlinear processes. Their name is inspired by the

---

<sup>7</sup>see <https://xgboost.readthedocs.io/en/stable/>

similitude to the human brain, since they emulate the way biological neurons make connections with each other.

The simplest network we can think of is formed by a single unit, namely, the neuron. The function implemented by a neuron consists of a linear combination of its inputs, plus a final non-linearity, similar, for example, to logistic regression. Each neuron has a set of input data ( $x_i$ ) and their weights ( $w_i$ ), a bias or threshold  $b$  and an output  $y$  - see Figure 2.13.



**Figure 2.13:** A single-neuron network

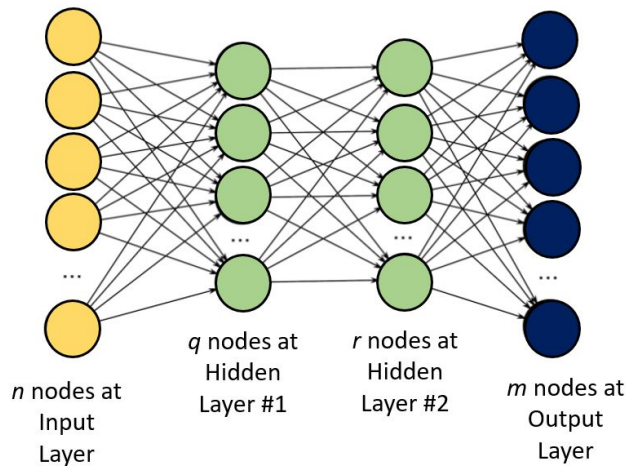
The non-linearity  $f$  that produces the output  $y$  is called "activation function" and is typically a sigmoid, a hyperbolic tangent or a Rectified Linear Unit (ReLU).

As intuitive, a single neuron is quite limited in terms of tasks that can perform. By gathering several neurons in parallel, it is possible to create *layers* and, by combining these layers, an actual *network* can be built. This architecture is referred to as Multi-Layer Perceptron (MLP).

With reference to Figure 2.14, 3 types of layers can be identified:

- Input layer, that represents the initial data for the neural network;
- Hidden layers, that are intermediate layers between input and output;
- Output layer, where the prediction for the given inputs is produced.

During the training process, at every iteration, based on the difference between the prediction of the network and the target output (i.e., the loss or error function value),



**Figure 2.14:** Exemplification of a multi-layer perceptron

the network adjusts its weighted associations according to a certain learning rule, which is typically based on some variant of gradient descent. These adjustments are repeated either for a fixed number of passes ("epochs"), or until the prediction is sufficiently close to the target output, which means, until meeting defined criteria.

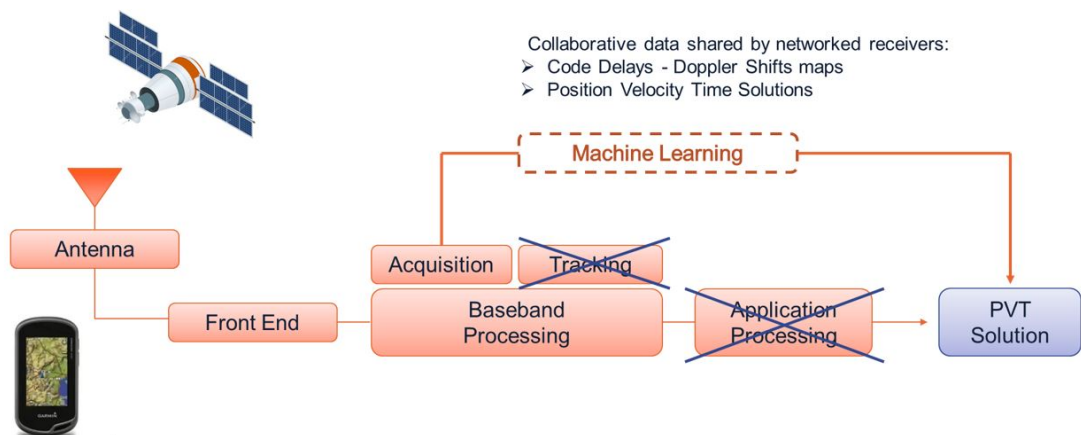
Stochastic Gradient Descent (SGD) is one of the most common optimization algorithm used to train neural models. The approach consists in computing the gradient of the error with respect to the weights, then updating the weights in such a way that the error is minimized. The gradient represents the direction of the steepest descent in the error space, and the step size determines the magnitude of the update. The step size is nothing but the *learning rate*. As seen for the GBDT, setting an appropriate learning rate is very important. High values of learning rate could cause the algorithm to diverge, while low values could require a long time to reach acceptable results (i.e., a high number of epochs).

The method is called "stochastic" because it only uses one sample from the training set to calculate the gradients at each iteration. This speeds-up the process, even though it could introduce a sort of instability: the loss, in fact, can temporarily increase and decrease with respect to the previous iteration. As a countermeasure to this behavior, in SGD smaller learning rate are preferred, and used in conjunction with other techniques, like "mini-batch". With "mini-batch", instead of iterating through the entire dataset or one observation, the dataset is split into into small subsets (batches) and for each batch the gradient is computed. Using a subset of data results in a lower number of iterations with respect to SGD.

## Chapter 3

# Methodology, design and set-up of the experimental scenario

With reference to the standard architecture (see Section 2, Figure 2.1), the purpose of this work is to understand if both tracking stage and application processing could be replaced by machine learning techniques, supported by collaborative users sharing their data (delay-Doppler mapping and estimated position) - see Figure 3.1.

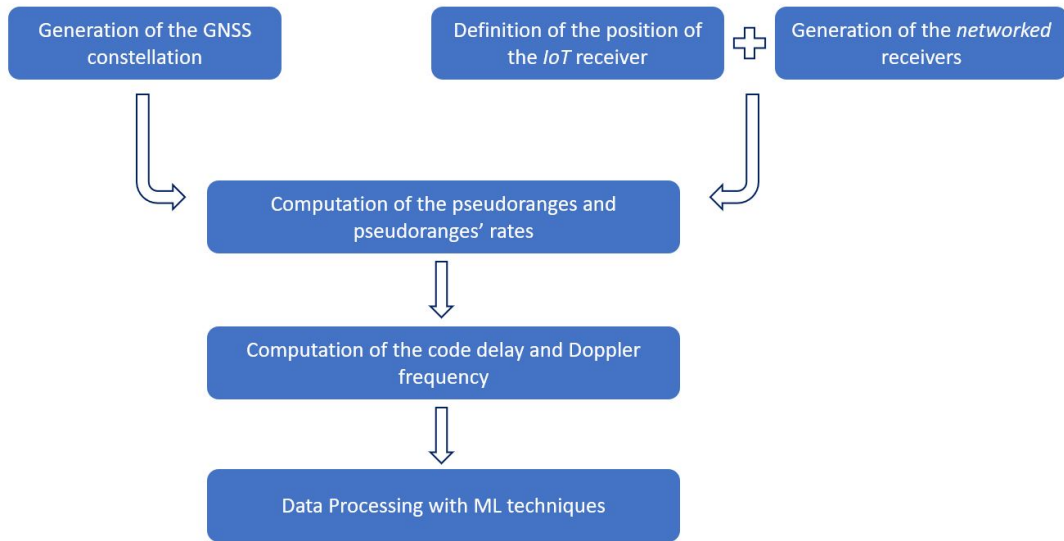


**Figure 3.1:** GNSS Receiver: use of machine learning to support PVT estimation

The first part of the work has been dedicated to the creation of an experimental environment in Matlab. We assume to have an IoT receiver, surrounded by a set of  $n$  networked receivers, randomly located within a radius  $R_{tol}$ . The IoT receiver has no chance to estimate its position, unless assisted by "standard" receivers. The idea is to build a dataset where the following information are available for each receiver:

- positions in geodetic, local and ECEF coordinates (see Appendix B for a description of the reference frames),
- satellites in view,
- delay-Doppler frequency pairs (with respect to each satellite in view).

The second part of the work has been focused on the data processing via ML tools in Python environment, using two different machine learning open-source libraries, XGBoost and Keras. The aim is to understand whether or not is it possible to estimate the IoT receiver position by using the data shared by the networked receivers. The work flowchart is shown in Figure 3.2.



**Figure 3.2:** Work flowchart

Finally, the estimation of the IoT position obtained using the two machine learning tools has been compared to a simplified “reference” model, where the IoT receiver’s

position is approximated by the arithmetic average of the networked receivers' positions.

## **3.1 Experimental environment**

This section describes the steps followed to build the experimental scenario in Matlab, namely:

1. Generation of the GNSS constellation;
2. Definition of the position of the IoT receiver and generation of the networked receivers;
3. Computation of the pseudoranges and pseudoranges' rates based on visible satellites;
4. Computation of code delay and Doppler frequency.

### **3.1.1 Generation of the GNSS constellation**

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 consists of:

1. coarse orbit information;
2. health status;
3. satellite vehicle identification;
4. clock corrections;
5. IoT time (for correlation with UTC).

These are the information needed by the receiver to identify a specific satellite and determine its position in space. The list of parameters returned by `semread_function.m` and their description is shown in Figure 3.3.

Parameters	Description
Time	GPS clock time, calculated using GPSWeekNumber and GPSTimeOfApplicability.
GPSWeekNumber	GPS week number, continuous, not mod(1024).
GPSTimeOfApplicability	Number of seconds since the beginning of the GPS week number.
PRNNumber	Satellite pseudorandom noise number.
SVN	Space vehicle reference number of the satellite.
AverageURANumber	Average URA number of the satellite.
Eccentricity	Eccentricity of the satellite.
InclinationOffset	Inclination angle offset from 54 degrees, in semicircles.
RateOfRightAscension	Rate of change in the measurement of the angle of right ascension, in semicircles per second.
SqrtOfSemiMajorAxis	Square root of the semimajor axis, in meters <sup>1/2</sup> .
GeographicLongitudeOfOrbitalPla	Geographic longitude of the orbital plane at the weekly epoch, in semicircles.
ArgumentOfPerigee	Angle from the equator to perigee, in semicircles.
MeanAnomaly	Angle of the position of the satellite in its orbit relative to perigee, in semicircles.
ZerothOrderClockCorrection	Satellite almanac zeroth-order clock correction term, in seconds.
FirstOrderClockCorrection	Satellite almanac first-order clock correction term, in seconds per second.
SatelliteHealth	Satellite vehicle health data code.
SatelliteConfiguration	Satellite vehicle configuration code.

**Figure 3.3:** Matlab semread\_function: parameters and description

Based on these data, the Matlab function [gnssconstellation.m](#) returns the satellite positions and velocities at the datetime  $t$ . Positions and velocities are specified in the ECEF coordinate system. If the timezone for the datetime is not specified, it is assumed to be UTC.



### 3.1.2 Definition of the position of the IoT receiver and generation of the networked receivers

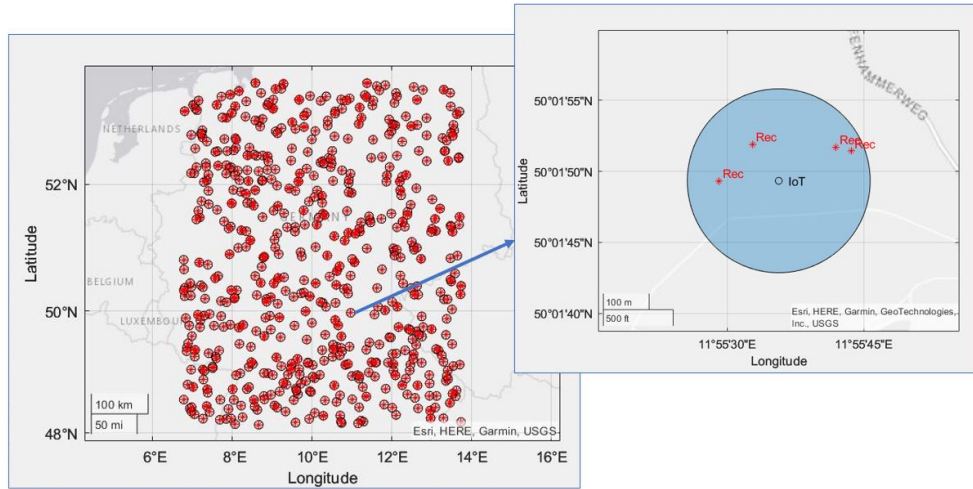
To build the experimental scenario, first of all we need to define the position of the IoT receiver. The position is randomly set within a specified area, identified by its geodetic coordinates (maximum - minimum latitude and longitude, plus average altitude). The initial area for training was chosen to correspond to Germany, given the edges in Figure 3.4. In terms of altitude, the average value over Germany is used:  $263m$  <sup>1</sup>.



Figure 3.4: Edges (Lat, Lon) of the area where the scenario is developed

Once the position of the IoT receiver is determined, it is possible to generate the networked receivers. We chose to consider a worst-case situation, where the number of networked receivers is 4. The idea is to have them randomly distributed within a certain radius  $R_{tol}$  with respect to the IoT receiver. The assumption is to use a  $200m$  radius (see also 3.1.3, where a plausibility check of the maximum radius is done).

<sup>1</sup><https://www.worlddata.info/>



**Figure 3.5:** Distribution over Germany of the dataset and detail of a single set including the IoT receiver ("IoT") and the 4 surrounding networked receivers ("Rec")

A first dataset of 500 points has been created following this approach - see Figure 3.5, where both the distribution of the points on the area and a detail of the single row (IoT plus the 4 networked receivers) is shown. The attributes that need to be defined for the receivers are listed in Table 3.1

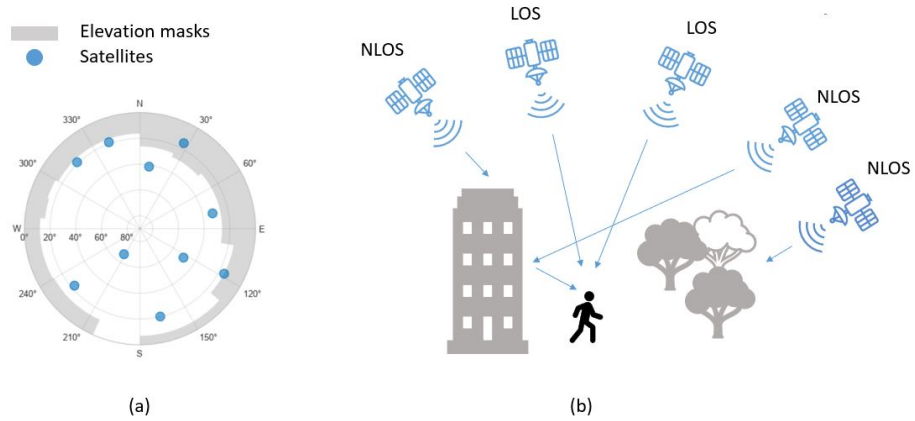
The position of the IoT receiver is defined with respect to geodetic coordinates (Latitude Longitude Altitude, or LLA), then translated into both East North Up (ENU) and Earth Centered Earth Fixed (ECEF) coordinates. The geodetic coordinates are preferred when the main need is to locate a point on the Earth's surface; however, they are less practical when, for instance, the networked receivers must be generated within a certain radius from the IoT "center". 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 using the Matlab function [lla2enu.m](#) (Appendix D) and only afterwards the 4 networked receivers' positions are generated (again, in ENU coordinates).

attribute	unit	description
name	[string]	name of the receiver (1,2,...,n)
mask angle	[deg]	random mask angle with respect to available satellites
azimut	[deg]	azimut angle with respect to satellites in view
elevation	[deg]	elevation angle with respect to satellites in view
visibility	[-]	number of satellites visible from the receiver
sat ID	[-]	satellite identification number
satNum	[-]	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
position dms	[deg,min,sec;deg,min,sec]	position of the receiver in lat,lon
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, EastNothUp coord
position ecef	[m,m,m]	position of the receiver, ECEF coord
posdiff	[m,m,m]	difference in position between satellite and receiver
losVector	[m,m,m]	line of sight between satellite and receiver
velocity enu	[m/s,m/s,m/s]	velocity of the receiver, EastNothUp coord
velocity enu	[m/s,m/s,m/s]	velocity of the receiver, ECEF coord
velocity ecef	[m/s,m/s,m/s]	velocity of teh receiver, EastNothUp 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
pdot matlab	[m/s]	pseudorange rate computed using Matlab toolbox function
bias	[s]	receiver clock bias
integer	[-]	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)

**Table 3.1:** Attributes of the Receivers

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](#) (Appendix E) to calculate azimut and elevation of the satellites in view and then cut out all the non-relevant data (i.e., coming from non-visible satellites).

Figure 3.6 shows an exemplification of satellites position (at a certain time instant  $t$ ) with the elevation masks (a) and a depiction of the LOS and NLOS satellites for the user due to environmental constraints (b).



**Figure 3.6:** Example of mask angles: (a) distribution of the satellites with the elevation angles and (b) LOS and NLOS satellites for the user

### 3.1.3 Computation of the pseudoranges and pseudoranges' rates based on visible satellites

In our environment, the IoT receiver is placed 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](#) (Appendix F) takes as inputs:

- the receiver position, in geodetic coordinates;
- the visible satellites' positions, specified as an  $S$ -by- $3$  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  $S$ -by- $3$  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 (see Appendix G), which has been developed 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).

All the work has been developed under the assumption of having a zero clock bias for all the receivers. Realistic values can be in the order of  $10^{-6}s$ . For what concerns the velocities of the receivers instead, it is made the assumption that the IoT receiver is not moving, while the networked receivers velocities are comparable to those of people walking (3 to 6 *km/hour*).

Once the pseudorange and pseudorange rate have been calculated, it is possible to simulate the information on code delay  $\tau$  and Doppler frequency  $f_d$ :

$$\tau = \rho \% L_{CODE} \quad (3.1)$$

$$f_d = (f_{L1} \cdot \rho\_rate)/c \quad (3.2)$$

where  $\rho$  and  $\rho\_rate$  are the ones previously computed,  $f_{L1}$  is the carrier frequency and  $L_{CODE}$  is calculated as:

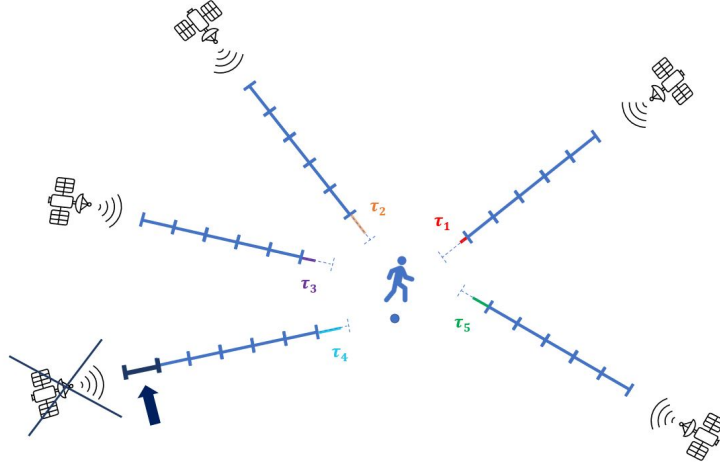
$$L_{CODE} = p \cdot L_{chip}, \quad (3.3)$$

where  $p$  is the code length and  $L_{chip} = c/R_{chip}$ , with  $c$  equal to the speed of light.

The referred values for carrier frequency, code length, code duration, chip duration and chip rate are shown in Appendix A.

### Check of the integer number of code repetition

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.



**Figure 3.7:** Illustration of code replicas meaning: for a receiver that has in view  $n$  satellites, the integer number of code repetitions for all of these satellites must be the same

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 (see Appendix H). This value is well larger than the  $200m$  radius used for the development of the simulation environment.

### 3.1.4 Processing of the dataset before data export

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 3.2 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.

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
$tau_{i1}$	[s]	delay of the i-th receiver to satellite 1
$tau_{i2}$	[s]	delay of the i-th receiver to satellite 2
$tau_{i3}$	[s]	delay of the i-th receiver to satellite 3
$tau_{i4}$	[s]	delay of the i-th receiver to satellite 4
$tau_{i5}$	[s]	delay of the i-th receiver to satellite 5
$tau_{i6}$	[s]	delay of the i-th receiver to satellite 6
$doppler_{i1}$	[1/s]	doppler of the i-th receiver to satellite 1
$doppler_{i2}$	[1/s]	doppler of the i-th receiver to satellite 2
$doppler_{i3}$	[1/s]	doppler of the i-th receiver to satellite 3
$doppler_{i4}$	[1/s]	doppler of the i-th receiver to satellite 4
$doppler_{i5}$	[1/s]	doppler of the i-th receiver to satellite 5
$doppler_{i6}$	[1/s]	doppler 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

**Table 3.2:** Data export; networked receivers are numbered 1 to 4, while IoT is identified as receiver 5

## 3.2 Machine Learning

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 Figure 3.4;
- for each IoT receiver,  $n = 4$  networked receivers are randomly generated to lay within a radius  $R_{tol} = 200m$  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.

As already stated in section 2, 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.



### Definition of the target output

With reference to Figure 3.4, 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  $10^3m$ , totally out of span if compared to the radius of  $200m$ . It must be kept in mind that at the equator  $1^\circ$  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, see Appendix B). 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 "real" IoT position.

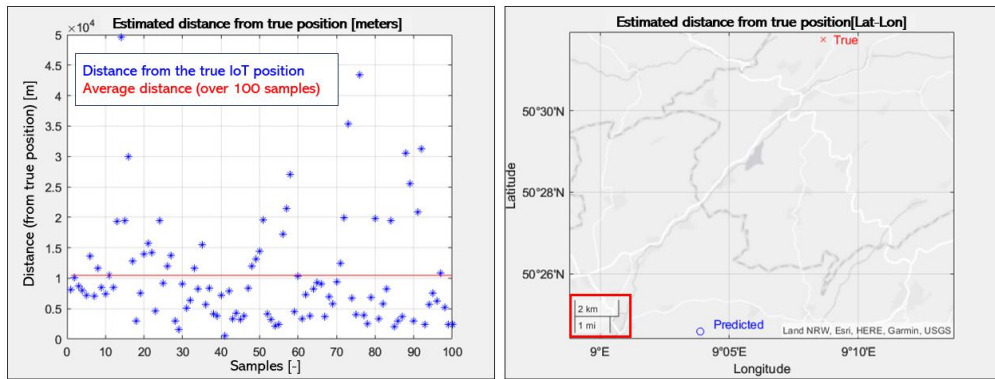
### Data Normalization

In our 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  $10^7$ . Some of the features, like for instance Doppler frequency, can have negative values. Such a discrepancy in numeric values for the different features can greatly affect the performance of the algorithm. 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:

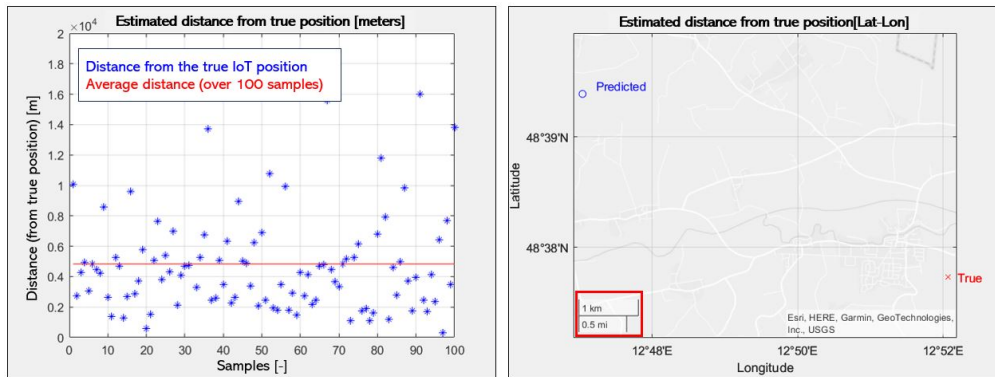
$$x_{normalized} = \frac{x - \mu(x)}{\sigma(x)}; \quad (3.4)$$

where  $\mu$  is the mean of the feature and  $\sigma$  is its standard deviation.

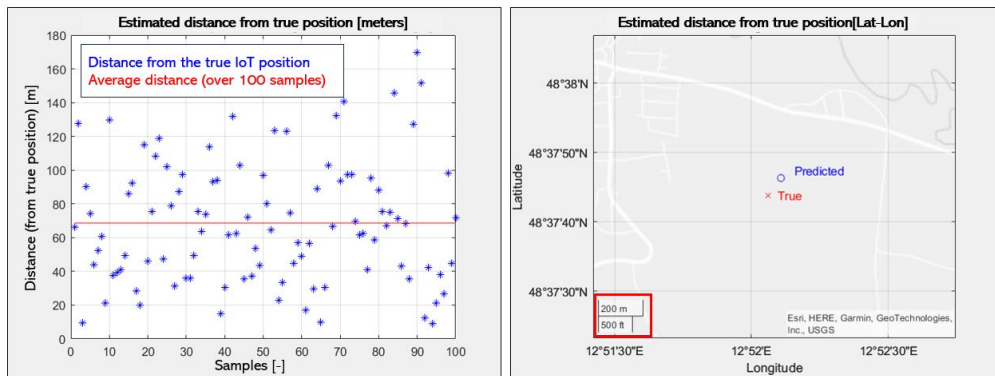
Figures 3.8, 3.9 and 3.10 show how the definition of the target output (LLA coordinates instead of ECEF coordinates) and the normalization of the data have affected the results of the models. In the first experiments, when using as output variables the geodetic coordinates, the IoT receiver position was estimated to be totally far-off the true location (see Figure 3.8, where the vertical axis scale goes up to  $50km$ ). When using instead the ECEF coordinates as outcome variables, without normalizing the data, the models would still return estimates totally out of range (see Figure 3.9, where the vertical axis scale goes up to  $20km$ ). Finally, the choice of ECEF coordinates as output variables and the normalization of the data has proved to give results consistent with the input dataset - see Figure 3.10, where all the results are lying within a  $200m$  radius.



**Figure 3.8:** Distance from the true position of the IoT receiver using LLA coordinates as output target and no normalization: the estimation is totally out of span.



**Figure 3.9:** Distance from the true position of the IoT receiver using ECEF coordinates as output target and no normalization: the estimation is still out of span.



**Figure 3.10:** Distance from the true position of the IoT receiver using normalized ECEF coordinates as output target: the estimation is within the correct range ( $< 200m$ ).

### Single Scalar Metric

The easiest method to make an estimation on the IoT receiver position 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.

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 ( $MSE_{avg}$ ):

$$SSM = \frac{MSE_{ML}}{MSE_{avg}}. \quad (3.5)$$

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.

### 3.2.1 XGBoost

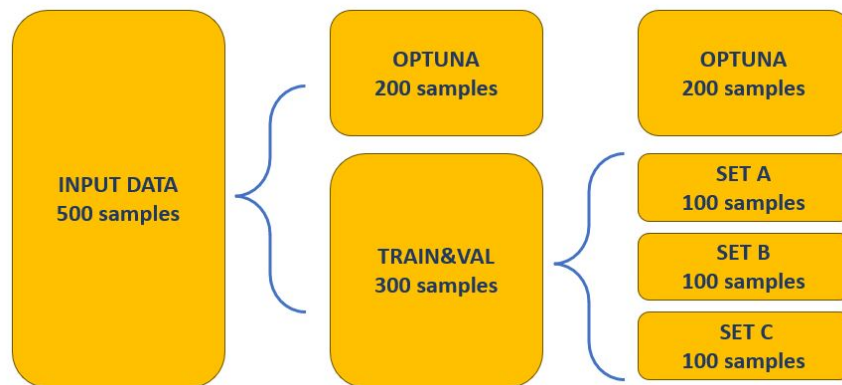
XGBoost provides a number of tuning hyperparameters that must be appropriately modulated to optimize the model. Table 3.3 describes the main hyperparameters that have been leveraged to build the model in XGBoost.

hyperparameter	description	use
objective	determines the loss function to be used	mean squared error 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	percentage of features used per tree	high values could lead to overfitting
subsample	percentage of samples used per tree	low values could lead to underfitting
max depth	determines how deeply a tree is allowed to grow	high values can result in data overfitting
n estimators	number of trees we want to build	to be tested, can't tell a-priori the effect on data fitting
learning rate	step size shrinkage	it is used to prevent overfitting

**Table 3.3:** XGBoost Hyperparameters

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<sup>2</sup>. This decision has led to split the dataset into two portions:

- 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.



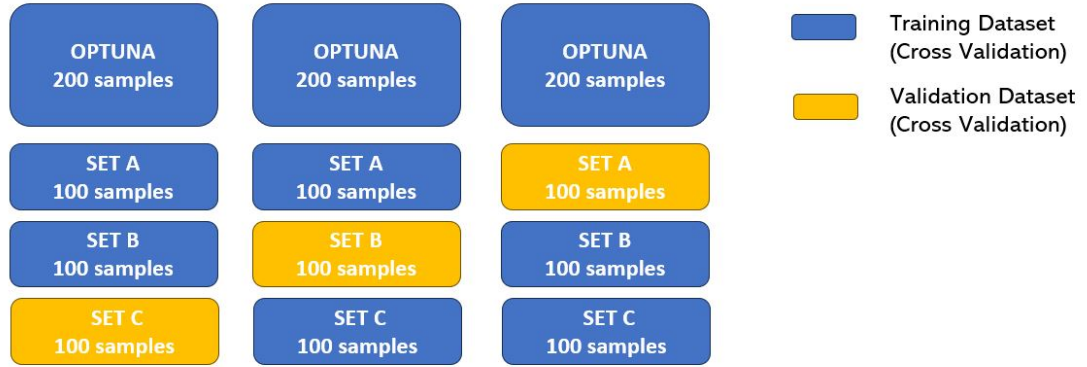
**Figure 3.11:** Illustration of data split for XGBoost - use of Optuna library for hyperparameters optimization

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.

---

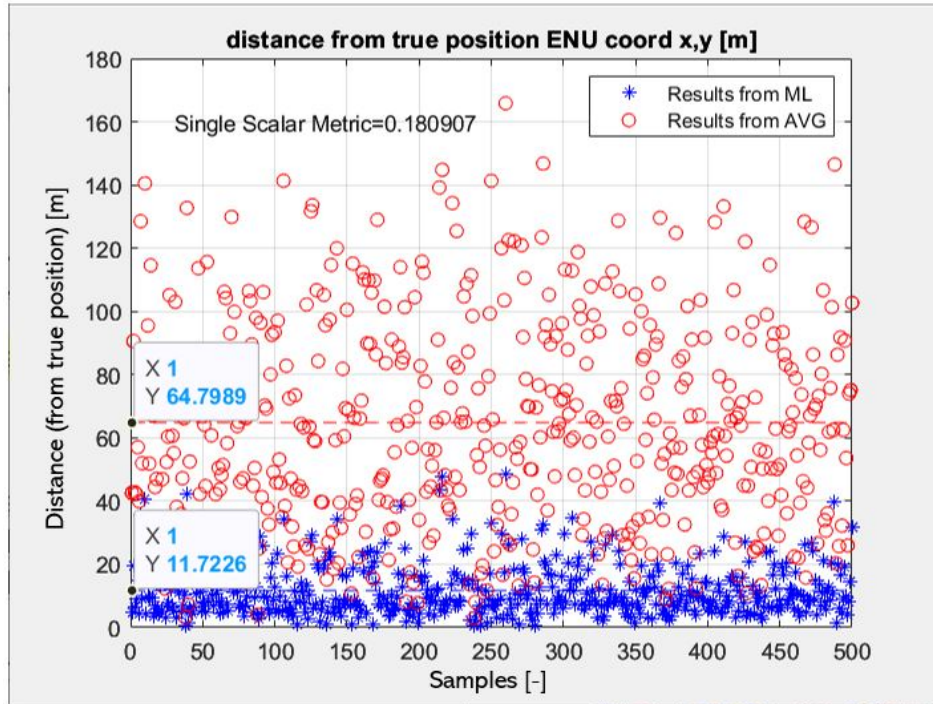
<sup>2</sup>Free download at <https://anaconda.org/conda-forge/optuna>



**Figure 3.12:** Illustration of cross validation for XGBoost

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 (see Figure 3.12, where the sets are identified as  $A, B, C$ ).

To stress the importance of splitting the data into separate training and validation sets, Figure 3.13 shows the results on the  $x, y$  coordinates obtained during the first trials done in XGBoost, where the same dataset was incorrectly used for both training and validation. Data were not even normalized at that time, but the results of the ML estimate look outstanding, with a Single-Scalar Metric of 0.18. Rather than being a "prediction", this is a result of data fitting (overfitting).



**Figure 3.13:** Incorrect use of machine learning techniques: average position predicted by XGBoost is extremely close to the real position (zero) only due to data overfitting (training dataset = validation dataset)

The training in Optuna can be done with reference to a single parameter at a time. Separate runs of Optuna have been done using first  $x$ , and then  $y$ , as target optimization parameters. It has been observed that the importance assigned to the hyperparameters changes consistently, with a predominance, for  $y$  coordinate, of the learning rate, while the same hyperparameter is way less important when optimizing with respect to  $x$  coordinate.

This behavior can be seen in the results over both training and validation datasets. When comparing the two optimization, the Single Scalar Metric is lower on the training set (better fitting), but higher on the validation set - see Figure 3.15.

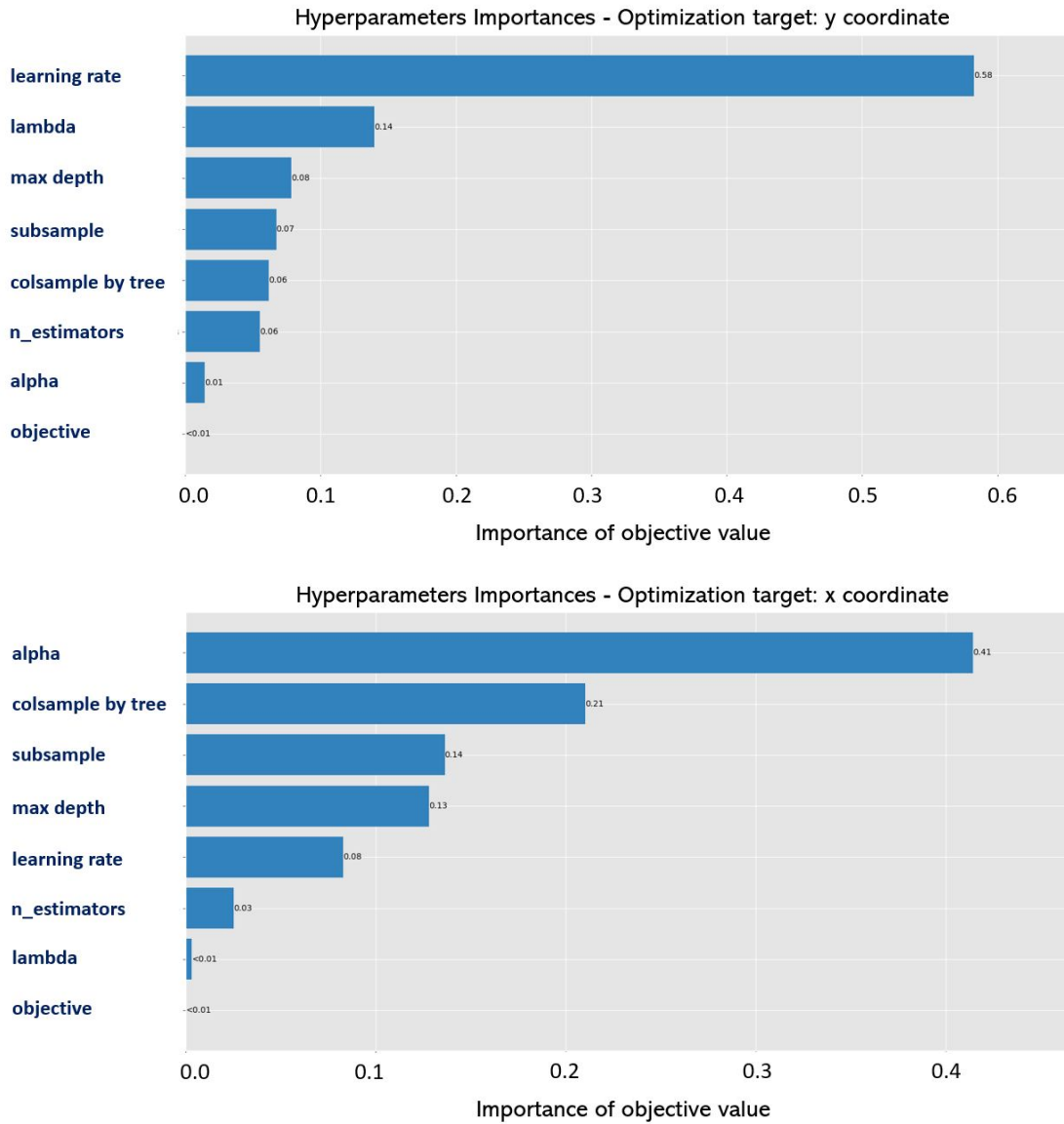
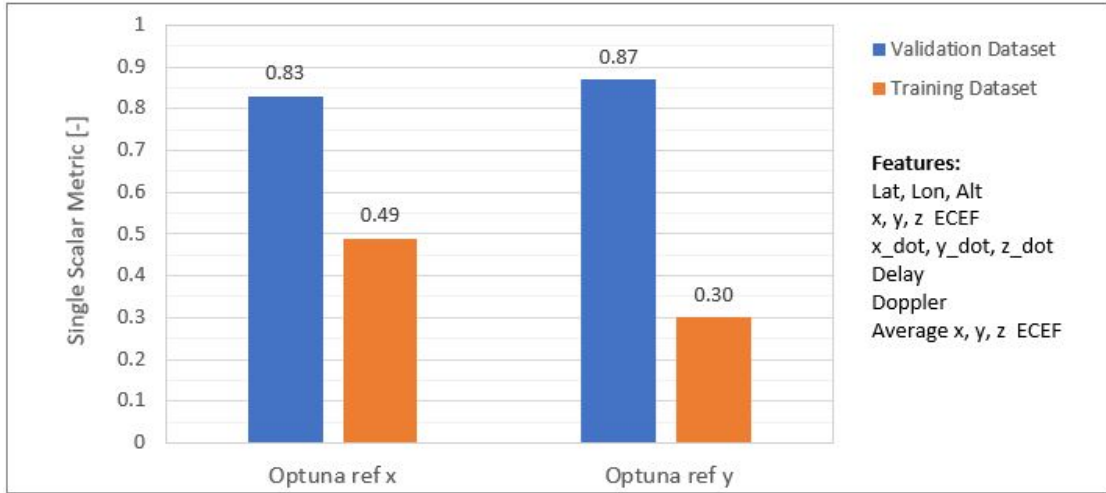


Figure 3.14: Hyperparameter importance in Optuna

With XGBoost 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.



**Figure 3.15:** Results over training and validation dataset with XGBoost

In our case, there isn't a clear relevance of a specific feature for all the receivers, so it happens, for instance, that the code delays of the third receiver with respect to the satellites  $n^{\circ}3$  and  $n^{\circ}5$  have been useful, but there is no general trend of the model particularly relying on code delay information. Doppler frequency instead has been, generally, classified with a lower importance. Based on these considerations, additional trials have been done excluding some features, like Doppler frequency and receiver velocity, but there has been no sensible improvement in the data. The only effect has been a minor worsening of the fitting on the training dataset vs. a minor improvement of the fitting on the validation set.



### 3.2.2 Keras

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:

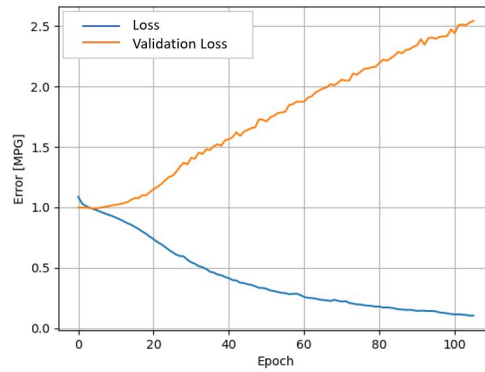
- the number of nodes / neurons at *Input Layer*: the number of features that have to be used as model inputs;
- the number of nodes / neurons at *Output Layer*: the number of features that need to be predicted by the model output;
- the number of *Hidden Layers*: the number of hidden layers between input and output;
- the number nodes / neurons for each *Hidden Layers*
- the activation function: used to determine a non-linearity into the output of a node / neuron. Rectified Linear Unit (ReLU) function is used at input and hidden layers, while linear identity function (no activation) is used at output layer.

Figures 3.16, 3.17 and 3.18 show an example of the difference in results that can be achieved over training and validation datasets when changing the number of hidden layers and of their nodes. As for the model optimisation, the *Adam* optimizer has been chosen<sup>3</sup> and tested with different values of *learning rate*.

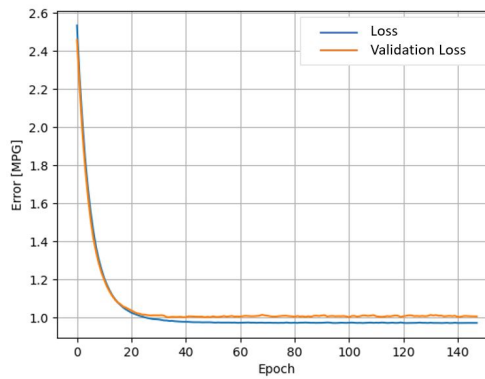
Similarly to what has been done for XGBoost, the data are normalized and split into training and validation (*validation\_split n = 20%*). The loss function (i.e., the function to be minimized during the training phase) is the mean squared error. The training phase in Keras can be advantageously managed 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 ("Epochs") and a *patience* index, which corresponds to the number of iterations with no improvement after which the training will be stopped.

---

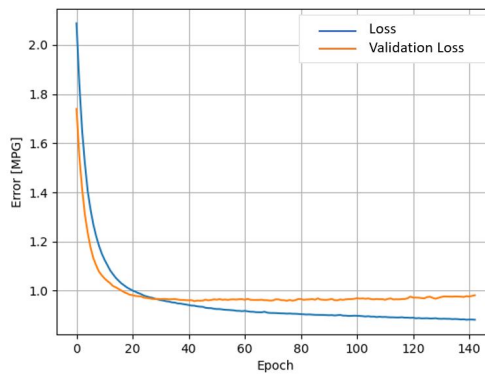
<sup>3</sup>Adam is a stochastic gradient descent method, based on adaptive estimation of first-order and second-order moments, see (<https://keras.io/api/optimizers/adam/>)



**Figure 3.16:** Results when using 4 hidden layers, with 20 nodes each. The behavior of loss and validation loss over the iterations show a high level of overfitting on the training dataset



**Figure 3.17:** Results when using no hidden layers. The ability of learning of the network is negligible (underfitting)



**Figure 3.18:** Results when using one hidden layer with 5 nodes. Learning and prediction are well balanced

## 2-nodes or 1-node output layer

When it comes to define the nodes at the output layer, there are two possible choices:

- using a single network to predict both x and y coordinates (2-nodes output layer), or
- using two separate models, to do the optimization over x and y coordinates separately (1-node 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 there is a slight improvement when using a single network for both outputs.

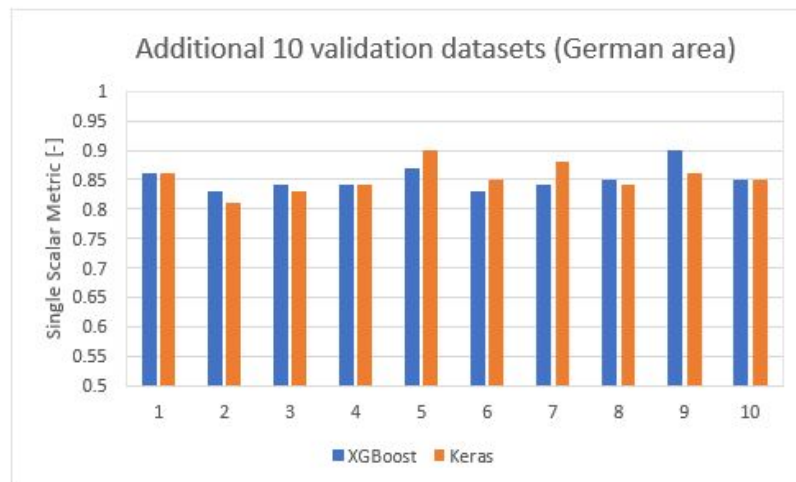
# Chapter 4

## Results

In this chapter, the results from ML processing are first validated, then the overall results on the position estimation are discussed.

### 4.1 Validation of ML 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 Figure 3.4).



**Figure 4.1:** Comparison of results from XGBoost and Keras over 10 additional validation sets created in the German area

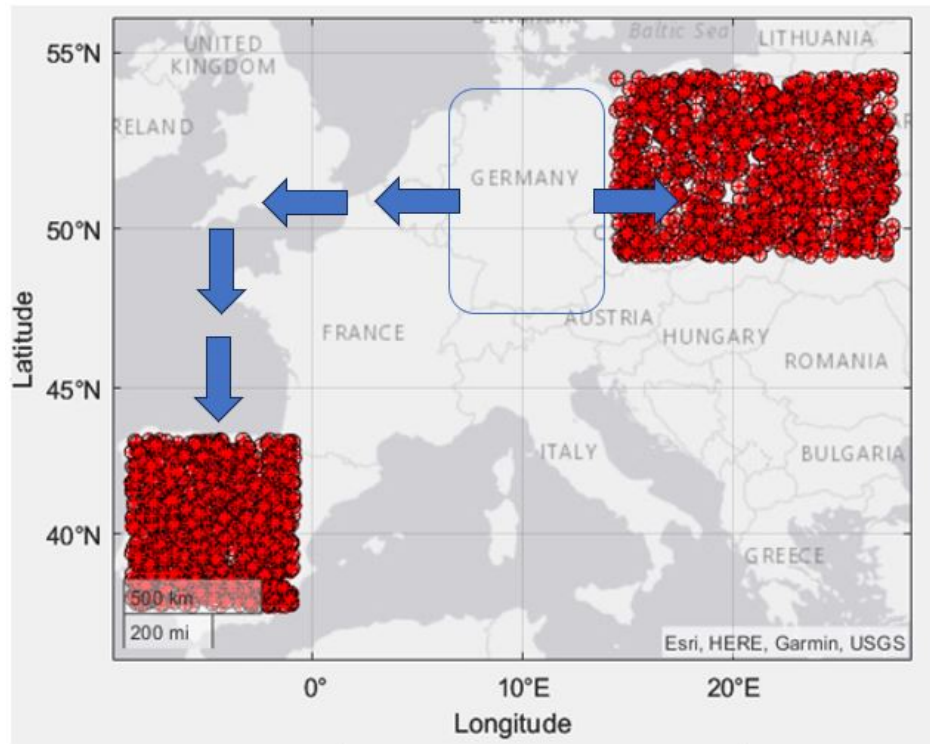
The results do not differ much from what obtained on the reference dataset; also, the performance in XGBoost and Keras are comparable - see Figure 4.1. The SSM is always lower than 1, meaning that the model estimate works better than the simplistic model defined as the average of the networked receivers 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 if we had developed a procedure that could be used everywhere or if it was in some way "tailored" on the features belonging to the original area (Germany). In detail, we have built:

1. 5 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'll refer to this region as "Polish area" (see Table 4.2 and Figure 4.2).
2. 5 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'll refer to this region as "Spanish area" (see Table 4.1 and Figure 4.2).
3. 1 datasets of 500 samples, created in a region delimited by the cities of Lusaka (Zambia) and Kinshasa (Congo Democratic Republic); we'll refer to this region as "Congolese area" (see Table 4.3 and Figure 4.3).
4. 1 datasets of 500 samples, created in a region delimited by the cities of Chelyabinsk (Russian Federation) and Astana (Kazakhstan); we'll refer to this region as "Russian area"(see Table 4.4 and Figure 4.3).

The general idea is to perform, first of all, a validation of the models obtained over the reference dataset on all the other 4 regions. The results are shown in Figure 4.4. The prediction looks like degrading moving further east in longitude with respect to the reference area. For the Congolese region, where we've moved below the equator, the prediction look sensibly improved.

A final model validation is done by training the models over the Spanish and Polish datasets, then performing the validation over the other areas. The results are aligned with what already seen (Figure 4.5).



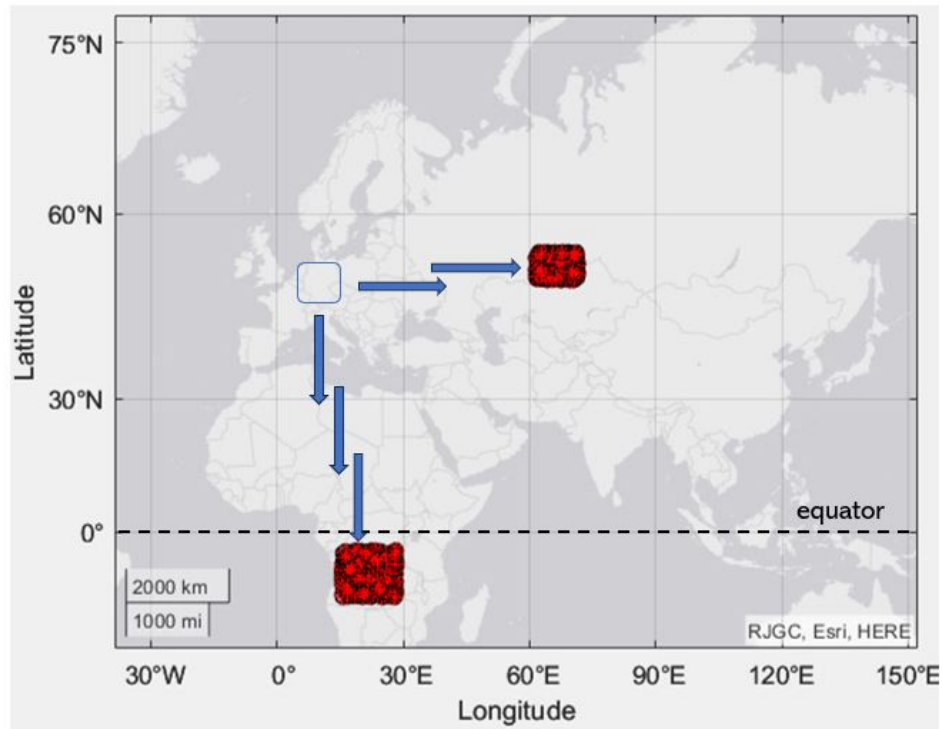
**Figure 4.2:** Distribution of the sample points generated over different geographic areas with respect to the experimental scenario: "Polish" and "Spanish" areas

location	min Lat	max Lat	min Lon	max Lon
Seville (Spain)	37.389092	-	-	-
Bilbao (Spain)	-	43.263013	-	-
Porto (Portugal)	-	-	-8.597684	-
Zaragoza (Spain)	-	-	-	-0.889085

**Table 4.1:** Edges (Lat, Lon) of the "Spanish" area

location	min Lat	max Lat	min Lon	max Lon
Brno (Czech Republic)	49.195060	-	-	-
Gdansk (Poland)	-	54.352025	-	-
Prague (Poland)	-	-	14.437800	-
Minsk (Belarus)	-	-	-	27.561524

**Table 4.2:** Edges (Lat, Lon) of the "Polish" area



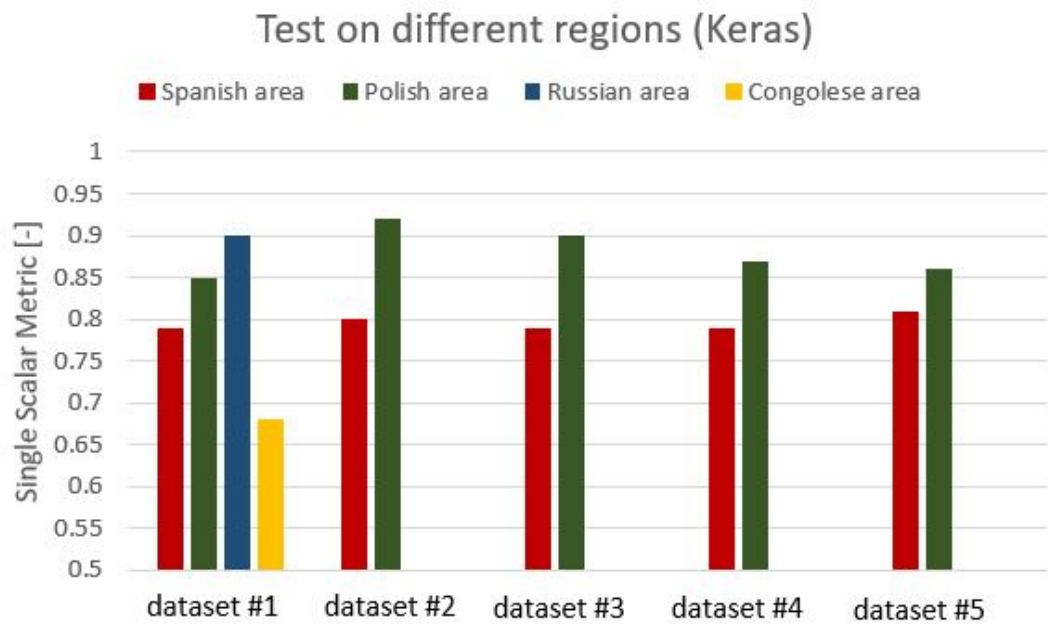
**Figure 4.3:** Distribution of the sample points generated over different geographic areas with respect to the experimental scenario: "Russian" and "Congolese" areas

location	min Lat	max Lat	min Lon	max Lon
Kinshasa (Congo Democratic Republic)	-	-4.441931	15.266293	-
Lusaka (Zambia)	-15.387526	-	-	28.322817

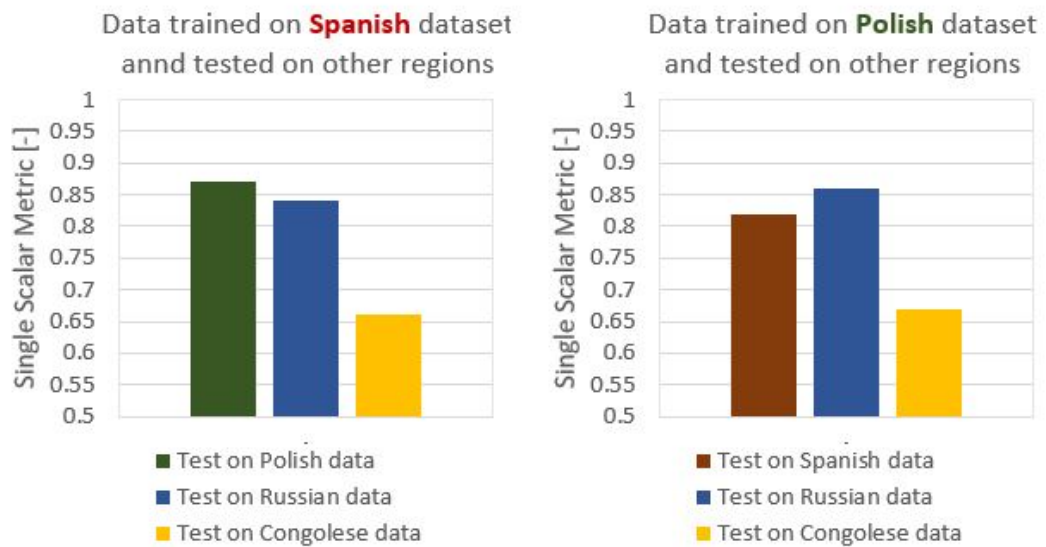
**Table 4.3:** Edges (Lat, Lon) of the "Congolese" area

location	min Lat	max Lat	min Lon	max Lon
Chelyabinsk (Russian Federation)	-	55.164442	61.436843	-
Astana (Kazakhstan)	51.160523	-	-	71.470356

**Table 4.4:** Edges (Lat, Lon) of the "Russian" area



**Figure 4.4:** Keras model is tested on the different validation dataset; the prediction seems strongly improved when moving right below the Equator



**Figure 4.5:** Keras script is validated on Spanish and Polish sets, then validated on other regions



## 4.2 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. Figure 4.6 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.

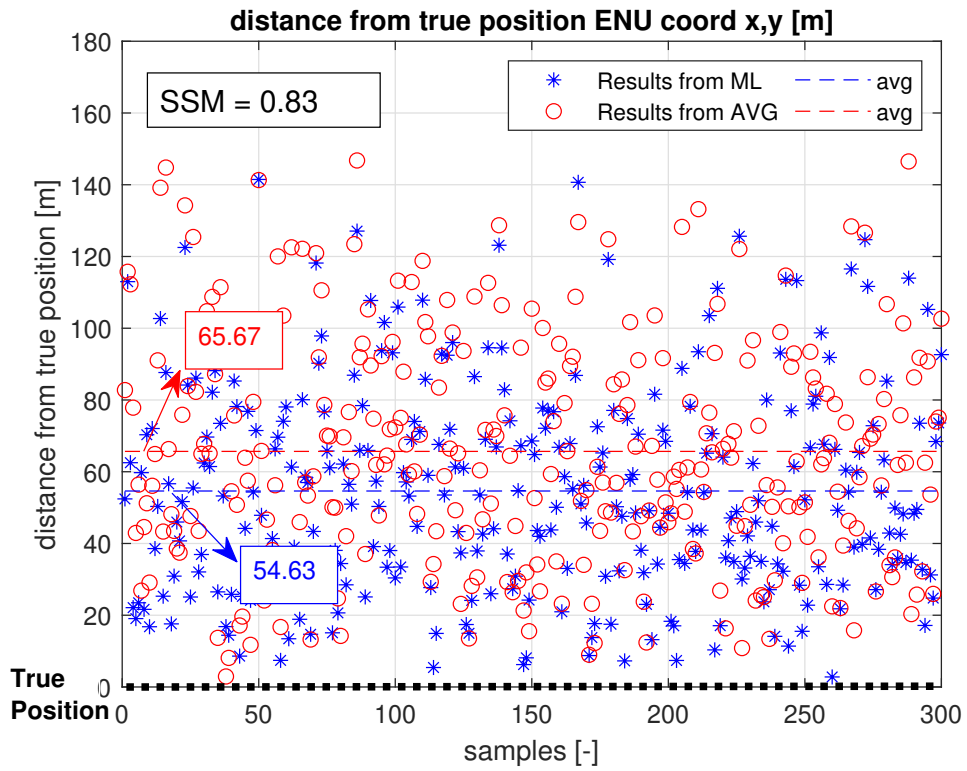
Keeping in mind the definition of SSM index (see 3.5), the values returned by ML model are, *in general*, closer to the true position: the SSM over the tested samples is 0.83. However, by looking at the detail of a single sample, it is clear that the ML models do *not* always perform better than the 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):

$$MLA = 100 \cdot \frac{\sum_{i=1}^N n_{MLi}}{N_{tot}} \quad (4.1)$$

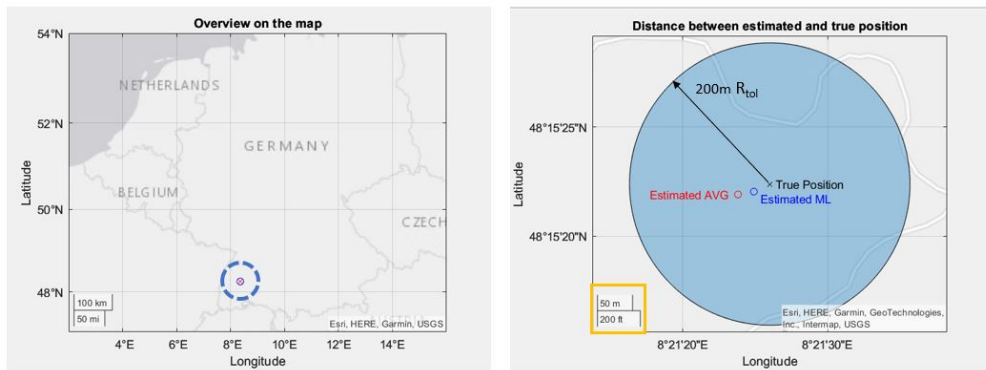
where  $N_{tot}$  is the total number of samples, and  $n_{ML}$  is equal to 1 if the ML model estimates better than the simplified model, and equal to 0 otherwise:

For the samples shown in Figure 4.6, the MLA is around 70%, which is an information to be considered together with the one coming from the SSM.

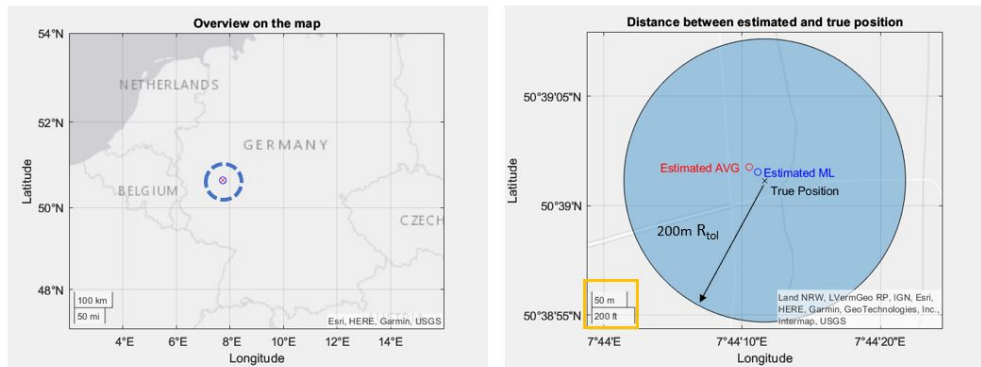
The values shown in Figure 4.6 are computed using local ENU coordinates; it is sufficient to translate them into LLA to get a plot in geodetic coordinates. Figures 4.7, 4.8, 4.9 and 4.10 show three samples randomly chosen within the dataset.



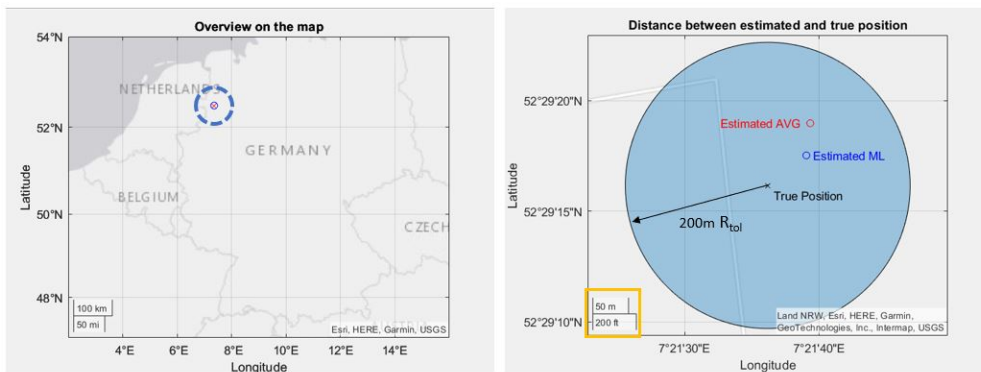
**Figure 4.6:** Distance from true IoT position based on simplified average model (red) and ML model (blue)



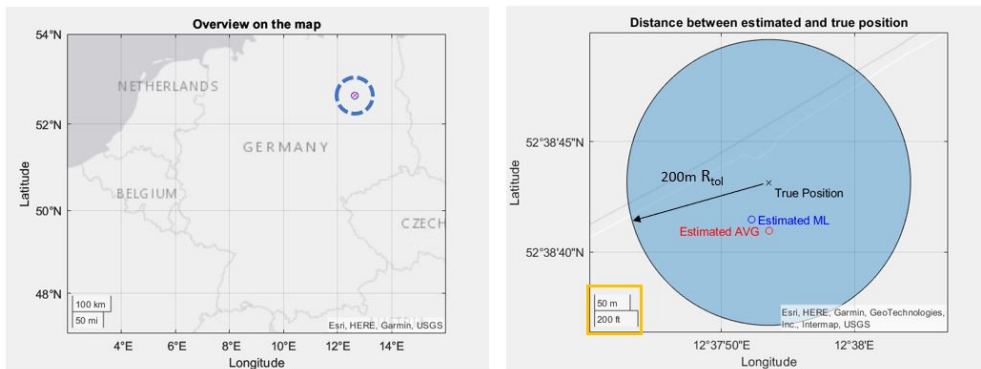
**Figure 4.7:** Sample A: distance from the true position is 47.65m when using reference simplified model and 25.08m when using XGBoost model



**Figure 4.8:** Sample B: distance from the true position is  $29.02m$  when using reference simplified model and  $15.52m$  when using XGBoost model



**Figure 4.9:** Sample C: distance from the true position is  $105.46m$  when using reference simplified model and  $68.58m$  when using XGBoost model



**Figure 4.10:** Sample D: distance from the true position is  $67.62m$  when using reference simplified model and  $57.15m$  when using XGBoost model

# Chapter 5

## Conclusions

In the development of the work, we assumed to have an IoT receiver surrounded by  $n$  networked receivers, sharing relevant information about their position. Based on a dataset inclusive of both networked and IoT receivers position information, different models have been trained, to assess whether machine learning could be able to estimate the IoT position better than the bare average of the networked receivers positions.

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. This observation is confirmed when doing a further validation of the models, over geographic regions different from the one used to generate the training dataset. Concerning, instead, the distance from the IoT receiver, the estimations are in general showing a 50-meters offset with respect to the “true” position. Although such a distance is not negligible, and considering also that the work has been developed in an experimental environment (with all the limitations that this entails), this preliminary study suggests that the PVT estimation via machine learning could work and its use in support of PVT estimation might be further investigated.

# Appendix A

## GPS signal parameters

The values of code length and duration, chip duration and rate, and of the carrier frequencies  $L1$  and  $L2$  are shown in Table A.1.

Code length $p$	1023	-
Code duration $T_{code}$	1	ms
Chip duration $T_{chip}$	997.5	ns
Chip rate $R_{chip}$	1.023	Mchips/s
Frequency $f_{L1}$	$154 \cdot 10.23$	MHz
Frequency $f_{L2}$	$120 \cdot 10.23$	MHz

**Table A.1:** GPS code information

# Appendix B

## Reference Frames

In order to compute the distance between a user and a satellite it is necessary for them to share a common reference system. Positions can be expressed:

- in geodetic coordinates: longitude, latitude, height
- in local coordinates, such as East-North-Up reference frame
- in Earth-Centered-Earth-Fixed coordinates

### Geodetic Coordinates

The position of the user is conventionally expressed in *geodetic coordinates*: longitude, latitude and height.

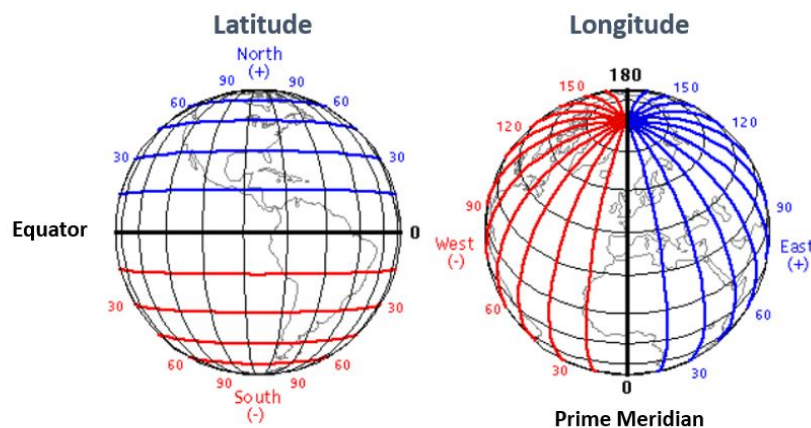
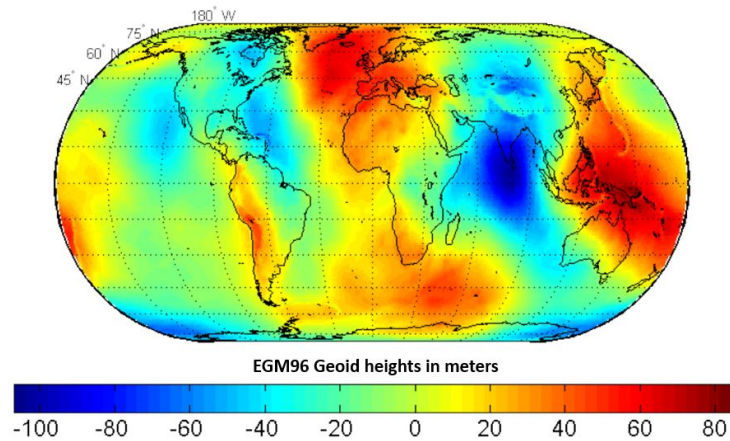


Figure B.1: Latitude and Longitude

Concerning height, or altitude, since the Earth is not spherical<sup>1</sup> and not uniform in density, an absolute height has to be defined with respect to an idealized mean sea level. The surface which serves as the global zero reference for measurement of height is the *Geoid*, which represents the locus of all points with the same gravity potential best fitting the average sea level globally - see Figure B.2.



**Figure B.2:** Geopotential surface for Earth Gravitational Model (EGM 96)

A reference ellipsoid is defined for each navigation systems. For GPS, it is the World Geodetic System (WGS 84).

WGS 84 fundamental parameters (1997 revision)	
Parameter	Value
Ellipsoid	
Semimajor axis (a)	6378137.0 m
Reciprocal flattening	298.257223563
Earth's angular velocity	$7292115.0 \cdot 10^{-11}$ rad/s
Earth's gravitational constant	$3986004.418 \cdot 10^8$ m <sup>3</sup> /s <sup>2</sup>
Speed of light in vacuum	$2.99792458 \cdot 10^8$ m/s

**Figure B.3:** Parameters of WGS 84

<sup>1</sup>The Earth can be approximated as an *ellipsoid of revolution*, obtained by revolving an ellipsoid about its minor axis (oblate ellipsoid)

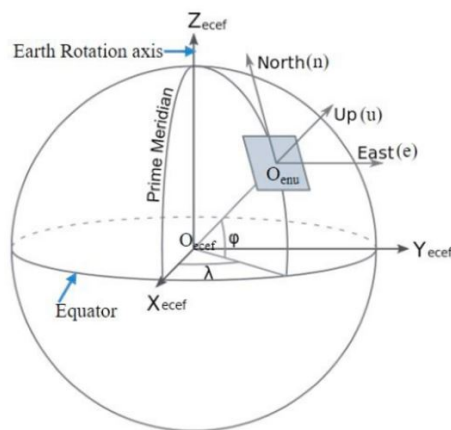
### Local Coordinates (ENU)

When dealing with positioning and tracking application, a local, Cartesian coordinate system might be more practical than Geodetic or ECEF coordinates. By definition, local coordinates are based on the tangent plane determined by the local vertical direction and the Earth's axis of rotation (see Figure B.4). The three coordinates represents the positions along the northern axis, the local eastern axis, and the vertical position. Two right-handed variants exist: *East-North-Up* (ENU) coordinates *North-East-Down* (NED) coordinates.

### ECEF Coordinates

The *Earth-centered, Earth-fixed coordinate system* (ECEF) is a Cartesian reference system meant to represents all the locations in the vicinity of the Earth (including its surface, interior, atmosphere, and surrounding outer space). The origin of the reference system is placed in correspondence of the center of mass of the Earth, while the three axis are:

- $z$ , along the axis of rotation
- $x$ , in the equatorial plane, toward the vernal equinox<sup>2</sup>
- $y$ , defined to complete a right-handed system



**Figure B.4:** Geodetic, ENU and ECEF coordinates

<sup>2</sup>It is the direction of intersection of the earth's equatorial plane with the plane of earth's orbits around the sun



## Appendix C

# Geometric Dilution of Precision

With reference to the definitions given in section 2.1.3, the generic pseudorange equation:

$$\rho_i = \sqrt{(x_i - x_{rec})^2 + (y_i - y_{rec})^2 + (z_i - z_{rec})^2} + b_{rec} \quad (\text{C.1})$$

can be linearized by means of Taylor expansion around a known location (and time) with coordinates  $\hat{x}_{rec}$ ,  $\hat{y}_{rec}$ ,  $\hat{z}_{rec}$  and  $\hat{b}_{rec}$ . The linearized equation will depend on the displacement with respect to the approximation point:

$$\Delta\rho = \hat{\rho}_i - \rho_i = a_{xi} \cdot \Delta x_{rec} + a_{yi} \cdot \Delta y_{rec} + a_{zi} \cdot \Delta z_{rec} - \Delta b_{rec} \quad (\text{C.2})$$

where:

$$\Delta x_{rec} = x_{rec} - \hat{x}_{rec};$$

$$\Delta y_{rec} = y_{rec} - \hat{y}_{rec};$$

$$\Delta z_{rec} = z_{rec} - \hat{z}_{rec};$$

$$\Delta b_{rec} = b_{rec} - \hat{b}_{rec};$$

and the coefficients  $a_{xi}$ ,  $a_{yi}$  and  $a_{zi}$  are defined as:

$$\begin{aligned} a_{xi} &= \frac{x_i - \hat{x}_{rec}}{\hat{r}_i}; \\ a_{yi} &= \frac{y_i - \hat{y}_{rec}}{\hat{r}_i}; \\ a_{zi} &= \frac{z_i - \hat{z}_{rec}}{\hat{r}_i}; \end{aligned}$$

The quantity  $\hat{r}_i$  is the geometrical distance between the linearization point and the satellite:

$$r_i = \sqrt{(x_i - \hat{x}_{rec})^2 + (y_i - \hat{y}_{rec})^2 + (z_i - \hat{z}_{rec})^2} \quad (C.3)$$

In case 4 satellites are used, the equations C.2 can be written in a compact expression as:

$$\Delta\rho = H \cdot \Delta x \quad (C.4)$$

where  $H = \begin{bmatrix} a_{x1} & a_{y1} & a_{z1} & 1 \\ a_{x2} & a_{y2} & a_{z2} & 1 \\ a_{x3} & a_{y3} & a_{z3} & 1 \\ a_{x4} & a_{y4} & a_{z4} & 1 \end{bmatrix}$  is the matrix of the coefficients.

By rewriting the C.4 with respect to  $\Delta x$ , we can define and solve an optimization problem, that in its final expression can be written as:

$$\Delta x = (H^T \cdot H)^{-1} \cdot H^T \Delta\rho \quad (C.5)$$

In general, the matrix combination  $(H^T \cdot H)^{-1} \cdot H^T$ , which is sometimes called the least-squares solution matrix, is a  $4 \times n$  matrix (where  $n$  is the number of the satellites) and depends only on the relative geometry between the user and the satellites at a certain time  $t$ . By applying the definition of covariance to the error in the position and time estimate  $dx$ , after some mathematical operations [15], it can be found that:

$$cov(dx) == (H^T \cdot H)^{-1} \cdot \sigma_{URE}^2, \quad (C.6)$$

with  $\sigma_{URE}^2$  is the square of the satellite User Equivalent Range Error (URE). The elements of the matrix  $(H^T \cdot H)^{-1}$  quantify how pseudorange errors translate into components of the covariance of  $dx$ .

In particular, the elements on the diagonal are the variance of the error in the different dimensions (including time), while the off-diagonal elements indicate the level of cross-correlation between the variables.

Defining:

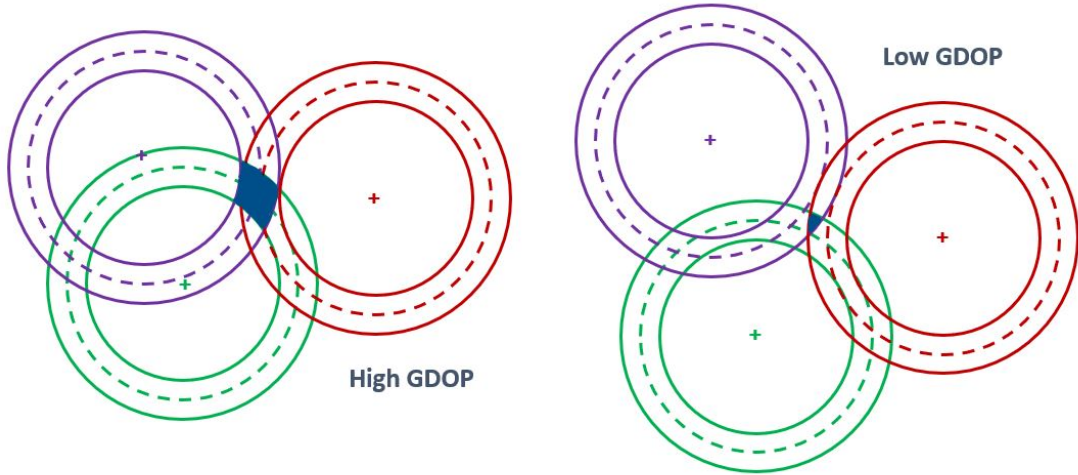
$$G = (H^T \cdot H)^{-1} = g_{ij} \quad (\text{C.7})$$

For each dimension, it will be possible to express the error as:

$$\begin{aligned} \sigma_{x_{rec}}^2 &= g_{11} \cdot \sigma_{URE}^2 \\ \sigma_{y_{rec}}^2 &= g_{22} \cdot \sigma_{URE}^2 \\ \sigma_{z_{rec}}^2 &= g_{33} \cdot \sigma_{URE}^2 \\ \sigma_{b_{rec}}^2 &= g_{44} \cdot \sigma_{URE}^2 \end{aligned}$$

The standard deviation of the positioning error can be obtained as:

$$\sigma = \sqrt{\sigma_{x_{rec}}^2 + \sigma_{y_{rec}}^2 + \sigma_{z_{rec}}^2 + \sigma_{b_{rec}}^2} = GDOP \cdot \sigma_{URE}. \quad (\text{C.8})$$



**Figure C.1:** Example of high and low GDOP for different emitters geometries

The quantity:

$$GDOP = \sqrt{g_{11} + g_{22} + g_{33} + g_{44}} = \sqrt{\text{tr}((H^T \cdot H)^{-1})} \quad (\text{C.9})$$

is the Geometric Dilution of Precision (GDOP) factor. This parameter is, by construction, a function of the satellites' geometry with respect to the user, while the  $\sigma_{URE}$  represents the pseudorange error factor.

To understand the meaning of the GDOP, Figure C.1 shows a simple example of how the distribution of the emitters can impact on the final precision on the position estimate.

## Appendix D

# Matlab function lla2enu.m

```
1 function xyzENU = lla2enu(lla, lla0, method)
2 %lla2enu Transform geodetic coordinates to local East-
  North-Up coordinates
3 % xyzENU = lla2enu(lla, lla0, method) transforms the
  geodetic coordinates, lla, to local East-North-Up (
  ENU) Cartesian coordinates, xyzENU. Specify the
  geodetic coordinates either as a 3-element row vector
  or an N-by-3 matrix of [lat, lon, alt]. Specify the
  origin of the local ENU system with the geodetic
  coordinates, lla0, as a 3-element row vector or an N-
  by-3 matrix of [lat0, lon0, alt0]. The conversion
  method is specified either as 'flat' or 'ellipsoid',
  to specify if earth is assumed to be flat or
  ellipsoidal. The local ENU coordinates are returned
  as a 3-element row vector or an N-by-3 matrix of [
  xEast, yNorth, zUp] in meters. lat and lat0 specify
  the latitude in degrees. lon and lon0 specify the
  longitude in degrees. alt and alt0 specify the
  altitude in meters.
4 %
5 % Notes
6 % -----
7 % - The latitude and longitude values in the geodetic
  coordinate system
8 % uses WGS84 standard.
```

```
9 % - Altitude is specified as height in meters above
WGS84 reference
10 %   ellipsoid.
11 %
12 % Limitations of the Flat Earth approximation
13 % -----
14 % - This transformation assumes the vehicle moves in
parallel to the
15 %   earth's surface.
16 %
17 % - This transformation method assumes the flat Earth z
-axis is normal to
18 %   the Earth at the initial geodetic latitude and
longitude only. This
19 %   method has higher accuracy over small distances
from the initial
20 %   geodetic latitude and longitude, and nearer to the
equator. The
21 %   longitude will have higher accuracy when there are
smaller variations
22 %   in latitude.
23 %
24 % - Latitude values of +90 and -90 may return
unexpected values because of
25 %   singularity at the poles.
26 %
27 % Example:
28 %
29 %   %Define the geodetic coordinates
30 %   lla=[45.976,7.658,4531];
31 %
32 %   %Define the reference geodetic coordinates
33 %   lla0=[46.017 7.750 1673];
34 %
35 %   %Transform from geodetic to local ENU coordinates
using the flat
36 %   %earth approximation
37 %   xyzENU = lla2enu(lla, lla0, "flat");
38 %
39 % See also lla2ned, enu2lla, ned2lla
40
```

```

41 % Copyright 2020 The MathWorks, Inc.
42
43 % References
44 % -----
45 % - [1] Stevens, B. L., and F. L. Lewis, Aircraft
46 %   Control and Simulation,
47 %   Second Edition, John Wiley & Sons, New York, 2003.
48 % - [2] Hofmann-Wellenhof, Bernhard, Herbert
49 %   Lichtenegger, and James Collins.
50 %   Global positioning system: theory and practice.
51 %   Springer Science &
52 %   Business Media, 2012.
53
54 %#codegen
55     narginchk(3,3);
56     validateattributes(l1a, {'double'}, {"real", "
57     nonempty", "2d", "ncols", 3}, "lla2enu", 'l1a', 1);
58
59     validateattributes(l1a0, {'double'}, {"real", "
60     nonempty", "2d", "ncols", 3}, "lla2enu", 'l1a0', 2);
61
62     method=validatestring(method, {'flat', 'ellipsoid'},
63     "lla2enu", "method", 3);
64
65     %Verify that the inputs are within the range
66     matlabshared.internal.latlon.validateLat(l1a0(:,1)
67     ,0);
68     matlabshared.internal.latlon.validateLat(l1a(:,1),1)
69     ;
70     matlabshared.internal.latlon.validateLon(l1a0(:,2)
71     ,0);
72     matlabshared.internal.latlon.validateLon(l1a(:,2),1)
73     ;
74
75     %Verify that input matrix sizes are correct
76     [minSize, idx]=min([size(l1a,1), size(l1a0,1)]);
77     if minSize~=1 && size(l1a,1)~=size(l1a0,1)
78         if idx==1

```

```

71         coder.internal.error("shared_coordinates:
latlonconv:IncorrectllaSize");
72         else
73             coder.internal.error("shared_coordinates:
latlonconv:Incorrectlla0Size");
74         end
75     elseif size(lla,1)==size(lla0,1)
76         idx=0;
77     end
78
79     if idx==1
80         llatmp= repmat(lla, size(lla0,1),1);
81         lla0tmp=lla0;
82     elseif idx==2
83         lla0tmp= repmat(lla0, size(lla,1),1);
84         llatmp=lla;
85     else
86         llatmp=lla;
87         lla0tmp=lla0;
88     end
89
90     xyzENU=nan(size(lla0tmp,1),3);
91
92     switch method
93     case "ellipsoid"
94         %Transform to xyz in ENU frame using [2]
95         xyzENU=fusion.internal.frames.lla2enu(llatmp,
lla0tmp);
96     case "flat"
97         %Transform to xyz in ENU frame using [1]
98         xyzNED = matlabshared.internal.latlon.
lla2nedFlat(llatmp, lla0tmp);
99         rotm=[0 1 0; 1 0 0; 0 0 -1];
100        xyzENU=(rotm*xyzNED')';
101     end

```



## Appendix E

# Matlab function lookangles.m

```
1 function [az, el, vis] = lookangles(recPos, satPos,
   maskAngle)
2 %LOOKANGLES Satellite look angles from receiver and
   satellite positions
3 % [az, el, vis] = LOOKANGLES(recPos, satPos) returns
   the look angles,
4 % azimuth az and elevation el in degrees, of the
   satellites using the
5 % satellite positions satPos in the Earth-Centered-
   Earth-Fixed (ECEF)
6 % coordinate system in meters and the receiver
   position recPos in
7 % geodetic coordinates in (latitude-longitude-altitude
   ) in
8 % (degrees, degrees, meters). The output vis is a
   logical array specifying
9 % the visibility of each satellite. The visibility is
   determined using
10 % the default receiver mask angle of 10 degrees.
11 %
12 % [az, el, vis] = LOOKANGLES(..., maskAngle) returns
   the look angles and
```

```

13 %   visibilities of satellites with a mask angle
    maskAngle.
14 %
15 %   Example:
16 %       recPos = [42 -71 50];
17 %       t = datetime('now');
18 %       gpsSatPos = gnssconstellation(t);
19 %       maskAngle = 5;
20 %       [az, el, vis] = lookangles(recPos, gpsSatPos,
    maskAngle);
21 %       fprintf('%d satellites visible at %s.\n', nnz(
    vis), t);
22 %
23 %   See also gnssconstellation, pseudoranges,
    receiverposition, gnssSensor.
24
25 %   Copyright 2020 The MathWorks, Inc.
26
27 %#codegen
28
29 if (nargin < 3)
30     maskAngle = 10;
31 end
32
33 validateattributes(recPos, {'double', 'single'}, ...
34     {'vector', 'numel', 3, 'real', 'finite'});
35 validateattributes(satPos, {'double', 'single'}, ...
36     {'2d', 'ncols', 3, 'real', 'finite'});
37 validateattributes(maskAngle, {'numeric'}, ...
38     {'scalar', 'real', '>=', 0, '<=', 90});
39
40 [vis, az, el] = nav.internal.gnss.satelliteStatus( ...
41     recPos, maskAngle, satPos);
42 end

```

## Appendix F

# Matlab function pseudoranges.m

```
1 function [p, pdot] = pseudoranges(recPos, satPos,
   varargin)
2 %PSEUDORANGES Pseudoranges between GNSS receiver and
   satellites
3 %   p = PSEUDORANGES(recPos, satPos) returns the
   pseudoranges, in meters,
4 %   between the receiver at position recPos and the
   satellites at positions
5 %   satPos. The receiver position is specified in
   geodetic coordinates
6 %   (latitude-longitude-altitude) in (degrees, degrees,
   meters). The
7 %   satellite positions are specified as an S-by-3
   matrix in meters in the
8 %   Earth-Centered-Earth-Fixed (ECEF) coordinate system.
   S is the number of
9 %   satellites.
10 %
11 %   [p, pdot] = PSEUDORANGES(___, recVel, satVel)
   returns the pseudorange
12 %   rates, pdot, in meters per second, between the
   receiver and satellites.
```

```

13 %   The receiver velocity, recVel, is specified in
14 %   meters per second in the
15 %   North-East-Down (NED) coordinate system. The
16 %   satellite velocities,
17 %   satVel, are specified as an S-by-3 matrix in meters
18 %   per second in the
19 %   ECEF coordinate system. S is the number of
20 %   satellites.
21 %
22 %   [p, pdot] = PSEUDORANGES(___, 'RangeAccuracy',
23 %   rangeStd, ...
24 %   'RangeRateAccuracy', rangeRateStd) returns the
25 %   pseudoranges and
26 %   pseudorange rates with random noises specified by
27 %   rangeStd and
28 %   rangeRateStd, in meters and meters per second,
29 %   respectively. The
30 %   default value of rangeStd and rangeRateStd are 1 and
31 %   0.02,
32 %   respectively.
33 %
34 %   Example:
35 %       recPos = [42 -71 50];
36 %       recVel = [1 2 3];
37 %       t = datetime('now');
38 %       [gpsSatPos, gpsSatVel] = gnssconstellation(t);
39 %       [p, pdot] = pseudoranges(recPos, gpsSatPos,
40 %       recVel, gpsSatVel);
41 %
42 %   See also gnssconstellation, lookangles,
43 %   receiverposition, gnssSensor.
44 %
45 %   Copyright 2020 The MathWorks, Inc.
46 %
47 %   %#codegen
48 %
49 %   narginchk(2,8);
50 %
51 %   validateattributes(recPos, {'double', 'single'}, ...
52 %       {'vector', 'numel', 3, 'real', 'finite'}, ...
53 %       'pseudoranges', 'recPos', 1);

```

```

43 validateattributes(satPos, {'double', 'single'}, ...
44     {'2d', 'ncols', 3, 'real', 'finite'}, ...
45     'pseudoranges', 'satPos', 2);
46
47 % Parse optional inputs.
48 numOptArgs = numel(varargin);
49 validNumOptArgs = any(numOptArgs == [0 2 4 6]);
50 coder.internal.errorIf(~validNumOptArgs, 'MATLAB:minrhs'
51     );
52 if ~isempty(varargin) && isnumeric(varargin{1})
53     recVel = varargin{1};
54     satVel = varargin{2};
55     validateattributes(recVel, {'double', 'single'}, ...
56         {'vector', 'numel', 3, 'real', 'finite'}, ...
57         'pseudoranges', 'recVel', 3);
58     validateattributes(satVel, {'double', 'single'}, ...
59         {'2d', 'nrows', size(satPos, 1), 'ncols', 3, '
60         real', 'finite'}, ...
61         'pseudoranges', 'satVel', 4);
62     optArgsStart = 3;
63 else
64     recVel = zeros(size(recPos), 'like', recPos);
65     satVel = zeros(size(satPos), 'like', satPos);
66     optArgsStart = 1;
67 end
68 numOptArgs = numOptArgs - (optArgsStart-1);
69 defaults = struct('RangeAccuracy', 1, 'RangeRateAccuracy
70     ', 0.02);
71 props = matlabshared.fusionutils.internal.setProperties(
72     defaults, ...
73     numOptArgs, varargin{optArgsStart:end});
74 rangeStd = props.RangeAccuracy;
75 rangeRateStd = props.RangeRateAccuracy;
76 validateattributes(rangeStd, {'double', 'single'}, ...
77     {'scalar', 'real', 'nonnegative'}, ...
78     'pseudoranges', 'RangeAccuracy');
79 validateattributes(rangeRateStd, {'double', 'single'},
80     ...
81     {'scalar', 'real', 'nonnegative'}, ...
82     'pseudoranges', 'RangeRateAccuracy');

```

```
79
80 % Convert input receiver position and velocity.
81 recVel = fusion.internal.frames.ned2ecefv(recVel, recPos
    (:,1), recPos(:,2));
82 recPos = fusion.internal.frames.lla2ecef(recPos);
83
84 % Calculate pseudoranges and pseudorange rates using
    satellite and
85 % receiver positions and velocities.
86 [p, pdot] = nav.internal.gnss.calculatePseudoranges(
    satPos, satVel, ...
87     recPos, recVel);
88
89 % Add measurement noises.
90 p = p + rangeStd .* randn(size(p), 'like', p);
91 pdot = pdot + rangeRateStd .* randn(size(pdot), 'like',
    pdot);
92 end
```

## Appendix G

# Alternative calculation of pseudoranges and pseudorange rates

```
1
2 % OPT2 --> use my function
3   for i=1:num_rec
4     % calculate receiver position in ECEF frame
5     Receiver(i).position_ecef=rotation_lla2ecef(Receiver
6     (i).position_lla(1,1),Receiver(i).position_lla(1,2),
7     Receiver(i).position_lla(1,3));
8     % calculate receiver velocity in ECEF frame
9     Receiver(i).velocity_ecef=rotation_enu2ecef(Receiver
10    (i).position_lla(1,1),Receiver(i).position_lla(1,2),
11    Receiver(i).velocity_enu);
12    % calculate position difference
13    Receiver(i).posdiff=Receiver(i).VisibleSatPos-((
14    Receiver(i).position_ecef).*ones(3,length(Receiver(i)
15    .VisibleSatPos))');
16    % calculate line of sight vector
17    Receiver(i).losVector = (Receiver(i).posdiff)./
18    vecnorm((Receiver(i).posdiff), 2, 2);
19    % compute pseudorange
20    Receiver(i).pseudorange=vecnorm(Receiver(i).posdiff
21    ,2,2)+Receiver(i).bias;
```

```
14     % compute pdot
15     Receiver(i).pdot=zeros(1,length(Receiver(i).
VisibleSatVel));
16     for j=1:length(Receiver(i).VisibleSatVel)
17         [Receiver(i).pdot(j)] = dot((Receiver(i).
VisibleSatVel(j,:)-Receiver(i).velocity_ecef),
Receiver(i).losVector(j,:))';
18     end
19     end
```



## Appendix H

# Check of the integer number of code repetition

```
1 %% Machine Learning techniques for Positioning,  
    Navigation and Timing in IoT devices  
2 % build a simulation scenario including:  
3 % gnss satellites  
4 % n=4 gnss receivers, positioned randomly within a  
    selectable radius  
5 % 1 IoT device that has to identify its own position  
    based on the  
6 % informations on doppler frequency and delay coming  
    from the 4 receivers  
7 %% STEP 1: generate the gnss constellation - later on we  
    'll include Galileo satellites  
8 t=datetime("today","TimeZone","UTC");  
9 d=t-2; % variable "d" is meant to represent a further  
    degree of freedom with respect to "t"  
10 DATA=gnssDATA;  
11 [DATA,TITLE]=semread_function(d);  
12 [satPos,satVel,satID] = gnssconstellation(d,DATA,  
    GNSSFileType="SEM"); % in ECEF coordinates  
13 %% STEP 2: specify the position of the IoT receiver  
14 % For sake of simplicity, we'll define a radius R_tol (  
    calibratable, starting value is 200 meters)
```

```
15 % The gnss receivers will be generated randomly within
    the circle with radius R_tol
16 % The IoT device is located at the center of the circle
17 % The IoT device and the receivers positions are
    specified in geodetic coordinates (latitude,
    longitude)
18 % altitude is based on avg altitude Turin (wrt sea level
    )
19 IoT_receiver=IoT;
20 %prompt = 'specify the IoT receiver in terms of [LAT(deg
    min sec);LON (deg min sec)]: ';
21 %IoT_receiver.position_dms=input(prompt);
22 IoT_receiver.position_dms=[45 04 47;7 37 55];
23 %prompt = 'specify the IoT altitude in meters: ';
24 %IoT_receiver.position_alt=input(prompt);
25 IoT_receiver.position_alt=239;
26 IoT_receiver.position_lla=[dms_angle(IoT_receiver.
    position_dms),IoT_receiver.position_alt];
27 IoT_receiver.position_enu=lla2enu(IoT_receiver.
    position_lla,IoT_receiver.position_lla,'flat');
28 IoT_receiver.position_ecef=rotation_lla2ecef(
    IoT_receiver.position_lla(1,1),IoT_receiver.
    position_lla(1,2),IoT_receiver.position_lla(1,3));
29 IoT_receiver.velocity_enu=[0,0,0]; % initial assumption,
    to be generated randomly
30 IoT_receiver.velocity_ecef=rotation_enu2ecef(
    IoT_receiver.position_lla(1,1),IoT_receiver.
    position_lla(1,2),IoT_receiver.velocity_enu);
31 IoT_receiver.bias=0; % initial assumption
32 %% STEP 3: generate the gnss receivers / repeat from
    here!
33 % specify position and velocity; the receivers must be
    located within a
34 % radius R_tol with respect to the IoT receiver
35 R_tol=23650; % define the circle of radius R_tol, in
    meters
36 num_rec=4; % specify the number of receivers
37 % generate the object receivers
38 for i=1:num_rec
39     Receiver(i).name=['Rec' num2str(i)];
40 end
```

```

41 % STEP 3.1: generate the receivers positions by setting
    randomly angle and distance wtr the IoT
42 % reference (0,0) in xy plane
43 % angles are 0 - 90 - 180 - 270
44 theta=2*pi*[0;1/4;1/2;3/4]; % angle in xy coord ref of
    the receivers
45 % all the receivers are placed on the limit radius
46 dist=R_tol*sqrt(ones(num_rec,1)); % distance of the
    receivers from IoT device
47 x = IoT_receiver.position_enu(1,1) + dist.*cos(theta);
48 y = IoT_receiver.position_enu(1,2) + dist.*sin(theta);
49 % define avg altitude
50 a = 239; % meters, in Turin - to be checked again
51 alt = a*(1+0.1*ones(num_rec,1));
52 % define gnss receivers position in EastNorthUp
    coordinates
53 for i=1:num_rec
54     Receiver(i).position_enu=[x(i),y(i),alt(i)];
55 end
56 % translate gnss receivers position from ENU to Lat-Lon-
    Alt
57 for i=1:num_rec
58     Receiver(i).position_lla=enu2lla(Receiver(i).
    position_enu,IoT_receiver.position_lla,'flat');
59     % correct a "wrong" estimation of the altitude with
    method enu2lla
60     Receiver(i).position_lla(3)=Receiver(i).position_lla
    (3)-a;
61 end
62 % plot the circle of radius R_Tol, the IoT and the
    receivers location in
63 % geoplot
64 numCirclePoints = 360;
65 angle = linspace(0, 2 * pi, numCirclePoints);
66 xcord = IoT_receiver.position_enu(1,1)+R_tol*cos(angle);
67 ycord = IoT_receiver.position_enu(1,2)+R_tol*sin(angle);
68 x_cord_lla=zeros(360,3);
69 for i=1:360
70 x_cord_lla(i,:)=enu2lla([xcord(i),ycord(i),0],
    IoT_receiver.position_lla,'flat');
71 end

```

```
72 circle = geopolyshape(x_cord_lla(:,1),x_cord_lla(:,2));
73 geoplots(circle, '-')
74 hold on
75 % geoplots of the IoT and receivers' positions
76 geoplots(IoT_receiver.position_lla(1,1),IoT_receiver.
    position_lla(1,2), 'ko')
77 text(IoT_receiver.position_lla(1,1),IoT_receiver.
    position_lla(1,2), ' IoT', 'Color','k');
78 hold on,
79 for i=1:num_rec
80 geoplots(Receiver(i).position_lla(1,1),Receiver(i).
    position_lla(1,2), 'r*')
81 text(Receiver(i).position_lla(1,1),Receiver(i).
    position_lla(1,2), ' Rec', 'Color','r', '
    VerticalAlignment','bottom');
82 hold on,
83 end
84 % NOTE: use different colors or at least add legend
85 % STEP 3.2: generate the gnss receivers velocity and
    clock bias
86 % specify a receiver velocity (NorthEastUp)
87 % for sake of simpliccity, we consider people walking
    3-6 km/hour
88 direction=2*pi*rand(num_rec,1);
89 velocity_vect=(3+3*rand(num_rec,1))/3.6;
90 vel_x=velocity_vect.*cos(direction);
91 vel_y=velocity_vect.*sin(direction);
92 vel_z=zeros(num_rec,1);
93 for i=1:num_rec
94     Receiver(i).velocity_enu=[vel_x(i),vel_y(i),vel_z(i)
    ];
95 end
96 % specify clock bias of the receivers (start with zero)
97 % later on, we can use 10e-06*rand
98 for i=1:num_rec
99     Receiver(i).bias=0*10e-06*rand;
100 end
101 % STEP 3.3: specify a mask angle and check which
    satellites are in view
102 % generate a random mask angle for each receiver
103 for i=1:num_rec
```

```

104     Receiver(i).mask_angle=0*20*rand;
105 end
106 % calculate azimuth, elevation and satellites in view
107 for i=1:num_rec
108 [Receiver(i).azimut,Receiver(i).elevation,Receiver(i).
    visibility] = lookangles(Receiver(i).position_lla,
    satPos,Receiver(i).mask_angle);
109 %fprintf('%d satellites visible at %s.\n',nnz(Receiver(i)
    ).visibility),t);
110 end
111 % cut out data from not visible satellites
112 for i=1:num_rec
113 [Receiver(i).azimut,Receiver(i).elevation,Receiver(i).
    VisibleSatPos,Receiver(i).VisibleSatVel,Receiver(i).
    satID]=satview(satPos,satVel,Receiver(i).azimut,
    Receiver(i).elevation,Receiver(i).visibility,length(
    DATA.PRNNumber));
114 end
115 %% STEP4: compute pseudoranges and pdot based on visbile
    satellites
116 c = physconst("Lightspeed");
117 % OPT2 --> use my function
118     for i=1:num_rec
119         % calculate receiver position in ECEF frame
120         Receiver(i).position_ecef=rotation_lla2ecef(Receiver
            (i).position_lla(1,1),Receiver(i).position_lla(1,2),
            Receiver(i).position_lla(1,3));
121         % calculate receiver velocity in ECEF frame
122         Receiver(i).velocity_ecef=rotation_enu2ecef(Receiver
            (i).position_lla(1,1),Receiver(i).position_lla(1,2),
            Receiver(i).velocity_enu);
123         % calculate position difference
124         Receiver(i).posdiff=Receiver(i).VisibleSatPos-((
            Receiver(i).position_ecef).*ones(3,length(Receiver(i)
            .VisibleSatPos))');
125         % calculate line of sight vector
126         Receiver(i).losVector = (Receiver(i).posdiff)./
            vecnorm((Receiver(i).posdiff), 2, 2);
127         % compute pseudorange
128         Receiver(i).pseudorange=vecnorm(Receiver(i).posdiff
            ,2,2)+Receiver(i).bias;

```

```

129     % compute pdot
130     Receiver(i).pdot=zeros(1,length(Receiver(i).
VisibleSatVel));
131         for j=1:length(Receiver(i).VisibleSatVel)
132             [Receiver(i).pdot(j)] = dot((Receiver(i).
VisibleSatVel(j,:)-Receiver(i).velocity_ecef),
Receiver(i).losVector(j,:))';
133         end
134     end
135 %% STEP 5: compute code delay tau
136 % code length
137 p_code=1023;
138 R_chip=1.023e06;
139 L_chip=c/R_chip;
140 L_CODE=L_chip*p_code;
141 % use calculation from OPT1 and OPT2
142 for i=1:num_rec
143     Receiver(i).delay=rem(Receiver(i).pseudorange,L_CODE
);
144     Receiver(i).integer=fix(Receiver(i).pseudorange/
L_CODE);
145 end
146 %% CHECK THE CODE REPLICAS
147 for j=1:length(Receiver(1).integer)
148     if ((Receiver(1).integer(j)+Receiver(2).integer(j)+
Receiver(3).integer(j)+Receiver(4).integer(j))/
num_rec) ~= Receiver(1).integer(j)
149         fprintf('the code replicas are not consistent! do
not use data\n');
150     else
151         fprintf('all the code replicas are consistent\n
');
152     end
153 end

```

# Bibliography

- [1] Nicola Linty, Letizia Lo Presti, Fabio Dovis, and Paolo Crosta. «Performance analysis of duty-cycle power saving techniques in GNSS mass-market receivers». In: May 2014, pp. 1096–1104. ISBN: 978-1-4799-3320-4. DOI: 10.1109/PLANS.2014.6851479 (cit. on p. 1).
- [2] Ismail Zahid, Muna Ali, and Rasheed Nassr. «Android Smartphone: Battery saving service». In: Nov. 2011. DOI: 10.1109/ICRIIS.2011.6125677 (cit. on p. 1).
- [3] Jatinder Pal Singh Zhenyun Zhuang Kyu-Han Kim. «Performance analysis of duty-cycle power saving techniques in GNSS mass-market receivers». In: 2010, pp. 315–330 (cit. on p. 1).
- [4] Dinesh Sathyamorthy, Shalini Shafii, Zainal Fitry M. Amin, Asmariah Jusoh, and Siti Zainun Ali. In: *Evaluation of the trade-off between Global Positioning System (GPS) accuracy and power saving from reduction of number of GPS*. Vol. 8. 67–75. 2016, pp. 67–75. DOI: <https://doi.org/10.1007/s12518-015-0166-z> (cit. on p. 1).
- [5] Nicolaie Fantana, Till Riedel, Jochen Schlick, Stefan Ferber, Jürgen Hupp, Stephen Miles, Florian Michahelles, and Stefan Svensson. «Internet of Things - Converging Technologies for Smart Environments and Integrated Ecosystems». In: Jan. 2013, pp. 153–204. ISBN: ISBN 978-87-92982-73-5 (print) (cit. on p. 1).
- [6] Ken Yamamoto et al. «26.5 A 0.7V 1.5-to-2.3mW GNSS receiver with 2.5-to-3.8dB NF in 28nm FD-SOI». In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 2016, pp. 444–445. DOI: 10.1109/ISSCC.2016.7418098 (cit. on p. 2).
- [7] Vicente Lucas-Sabola, Gonzalo Seco-Granados, José A. López-Salcedo, J.A. Garcia-Molina, and Crisci Massimo. «Cloud GNSS receivers: New advanced applications made possible». In: June 2016, pp. 1–6. DOI: 10.1109/ICL-GNSS.2016.7533852 (cit. on p. 2).

- [8] Pramila P. Shinde and Seema Shah. «A Review of Machine Learning and Deep Learning Applications». In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857 (cit. on p. 2).
- [9] Alessio Burrello, Matteo Risso, Beatrice Alessandra Motetti, Enrico Macii, Luca Benini, and Daniele Jahier Pagliari. «Enhancing Neural Architecture Search with Multiple Hardware Constraints for Deep Learning Model Deployment on Tiny IoT Devices». In: *IEEE Transactions on Emerging Topics in Computing* (2023) (cit. on p. 2).
- [10] Nicola Linty, Alessandro Farasin, Alfredo Favenza, and Fabio Dovis. «Detection of GNSS Ionospheric Scintillations Based on Machine Learning Decision Tree». In: *IEEE Transactions on Aerospace and Electronic Systems* 55.1 (2019), pp. 303–317. DOI: 10.1109/TAES.2018.2850385 (cit. on p. 2).
- [11] I. Mallika, Venkata Ratnam, Saravana Raman, and Sivavaraprasad Gampala. «Machine learning algorithm to forecast ionospheric time delays using Global Navigation satellite system observations». In: *Acta Astronautica* 173 (Apr. 2020). DOI: 10.1016/j.actaastro.2020.04.048 (cit. on p. 2).
- [12] Li-Ta Hsu. «GNSS multipath detection using a machine learning approach». In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–6. DOI: 10.1109/ITSC.2017.8317700 (cit. on p. 2).
- [13] Haosheng Xu, Antonio Angrisano, Salvatore Gaglione, and Li-Ta Hsu. «Machine learning based LOS/NLOS classifier and robust estimator for GNSS shadow matching». In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. Vol. 1. 2020. DOI: 10.1186/s43020-020-00016-w (cit. on p. 2).
- [14] Alex Minetto, Calogero Cristodaro, and Fabio Dovis. «A collaborative method for positioning based on GNSS inter agent range estimation». In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, pp. 2714–2718 (cit. on p. 2).
- [15] Christopher Hegarty and Elliott Kaplan. *Understanding GPS Principles and Applications, Second Edition*. 2005 (cit. on pp. 3, 11, 12, 57).
- [16] [https://gssc.esa.int/navipedia/index.php?title=Generic\\_Receiver\\_Description](https://gssc.esa.int/navipedia/index.php?title=Generic_Receiver_Description) (cit. on p. 3).



- [17] G. Arul Elango, G.F. Sudha, and Bastin Francis. «Weak signal acquisition enhancement in software GPS receivers – Pre-filtering combined post-correlation detection approach». In: *Applied Computing and Informatics* 13.1 (2017), pp. 66–78. ISSN: 2210-8327. DOI: <https://doi.org/10.1016/j.aci.2014.10.002>. URL: <https://www.sciencedirect.com/science/article/pii/S2210832714000271> (cit. on p. 11).
- [18] Batta Mahesh. «Machine learning algorithms-a review». In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386 (cit. on p. 14).
- [19] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. «The Higgs boson machine learning challenge». In: *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*. Ed. by Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. Vol. 42. Proceedings of Machine Learning Research. Montreal, Canada: PMLR, Dec. 2015, pp. 19–55. URL: <https://proceedings.mlr.press/v42/cowa14.html> (cit. on p. 15).