European Commission

JRC TECHNICAL REPORTS

Implementation of MATLAB libraries and related technical documents for Accurate Positioning in Intelligent Transport System

Technical Report for contract

CCR.IPSC.C259769

Baldini Gianmarco

ARES2T

Joint Research Centre

Contact information
Name: Gianmarco Baldini
Address: Via Enrico Fermi 2749
Email: gianmarco.baldini@ec.europa.eu
Tel: +39 0332 78 6618

JRC Science Hub
https://ec.europa.eu/jrc

JRC97140

PDF	ISBN 978-92-79-67955-1	doi:10.2760/027883
Print	ISBN 978-92-79-67956-8	doi:10.2760/25848

Luxembourg: Publications Office of the European Union, 2015

How to cite this report: Baldini Gianmarco, ARES2T, *Implementation of MATLAB libraries and related technical documents for Accurate Positioning in Intelligent Transport System,* doi:10.2760/027883

Title *Implementation of MATLAB libraries and related technical documents for Accurate Positioning in Intelligent Transport System*.

Abstract
A proof-of-concept project was started in the JRC in early 2014 to investigate and design a framework to improve positioning accuracy in Intelligent Transport Systems (ITS) on the basis of the IMU sensors. In the first phase of the project, a theoretical framework was defined and described in Report EUR 27042 EN. This technical report describes in detail the technical solution adopted to implement the theoretical framework as a MATLAB library providing dedicated functionalities to apply for an Accurate Positioning in Intelligent Transport System (which is the second phase of the project). The mathematical framework developed in the first phase of the project has been implemented into a working and self-standing MATLAB library. The MATLAB library is composed by three functional blocks: the Extended Kalman Filter, the vehicle's status estimation module and the Particle Filter.
The MATLAB library is capable of ingesting data in batch or real-time fashion. It elaborates the acquired data and displays the results by means of easy to understand graphs. It interoperates with the OpenStreetMap maps to provide georeferenced results.
The report describes the main modules of the MATLAB library, how they are connected and how they must be used. The report also describes the testing scenarios and the results of the tests. Finally, the technical report provides recommendations on the future developments of this MATLAB library.

Summary

2 Index of Figures 4
3 Table of Acronyms 6
Executive summary 7
4 Introduction 8
5 The mathematical framework (EKF, PF) 10
5.1 Coordinate Systems 10
5.2 Modelling 12
5.2.1 IMU and GPS Sensors' Models 12
5.2.2 System Dynamics 12
5.2.3 Measurement models 17
6 Architecture of the Integrated INS/GPS Navigation system and modules' description 18
6.1 Extended Kalman Filter 20
6.2 $\boldsymbol{\Delta p(N), \Delta \psi(N)}$ pdf Estimation Module 22
6.3 Particle Filter 25
7 Implementation 29
7.1 Supporting tools 29
7.1.1 MATLAB 29
7.1.2 OpenStreetMap and Overpass-turbo 29
7.2 Data Sources and Data Model 32
7.2.1 GPS 33
7.2.2 Gyroscope 37
7.2.3 Accelerometer 39
7.3 Positioning Modules 40
7.3.1 Extended Kalman Filter- Code (& code description) 41
7.3.2 Particle Filter- Code (& code description) 47
7.4 Integration 50
7.5 Test cases 56
7.5.1 Scenario 1: Straight Motion at Constant Velocity – Description 56
7.5.2 Scenario 2: Straight Motion under Constant Acceleration – Description 59
7.5.3 Scenario 3: Circular Turn at Constant Velocity – Description 62

7.5.4 Scenario 4: Junction at Constant Velocity – Description 64
7.5.5 Scenario 5: Road Hump at Constant Velocity – Description 65
8 Simulation and results 68
8.1 Scenario 1: Straight Motion at Constant Velocity – Test Result 68
8.2 Scenario 2: Straight Motion at Constant Acceleration – Test Result 74
8.3 Scenario 3: Circular Turn at Constant Velocity – Test Result 78
8.4 Scenario 4: Junction at Constant Velocity – Test Result 79
8.5 Scenario 5: Road Hump at Constant Velocity – Test Result 80
9 Conclusions 82
10 Recommendations 83
11 Bibliography 86

2 Index of Figures

Figure 1. The employed coordinate systems (navigation and body frames). 11
Figure 2. Architecture of the implemented integrated INS/GPS Navigation system. 19
Figure 3. The Overpass-turbo graphical user interface. 30
Figure 4. Map information visualised with OpenStreetMap. 32
Figure 5. Route followed in the measurement campaigns at ISPRA. 32
Figure 6. The DAISY-7 - GPS/MEMS module employed in measurement campaigns. 33
Figure 7 Measured angular velocity along x axis 38
Figure 8 Measured angular velocity along y axis 38
Figure 9 Measured angular velocity along z axis 38
Figure 10 Measured acceleration along x axis 39
Figure 11 Measured acceleration along y axis 40
Figure 12 Measured acceleration along z axis 40
Figure 13. MATLAB integration workflow 51
Figure 14. Artificial acceleration measurement along x axis. 58
Figure 15. Artificial acceleration measurement along z axis. 59
Figure 16. Artificial gyroscope measurement along x axis. 59
Figure 17. Artificial acceleration measurement along x axis. 61
Figure 18. Artificial acceleration measurement along y axis. 64
Figure 19. Artificial acceleration measurement along z axis. 64
Figure 20. Scheme of a junction 65
Figure 21. Simplified acceleration and velocity profiles for road hump scenario 66
Figure 22. Artificial acceleration measurement along z axis. 68
Figure 23. Scenario 1: estimated path on xy plane. 69
Figure 24. Scenario 1: estimated path on xy plane, zoom on the initial simulation times. 70
Figure 25. Scenario 1: estimated path on xy plane, zoom on the final simulation times. 70
Figure 26. Scenario 1: estimated path on xy plane, error with respect to the nominal path. 71
Figure 27. Scenario 1: estimated path on xy plane – with EKF restart. 71
Figure 28. Scenario 1: estimated path on xy plane, zoom on the initial simulation times – with EKF restart. 72
Figure 29. Scenario 1: estimated path on xy plane, zoom on the final simulation times – with EKF restart. 72
Figure 30. Scenario 1: error with respect to the nominal path – with EKF restart. 73
Figure 31. Scenario 2: estimated path on xy plane – with EKF restart. 74
Figure 32. Scenario 2: estimated path on xy plane, zoom on the initial simulation times – with EKF restart. 75
Figure 33. Scenario 2: estimated path on xy plane, zoom on the final simulation times – with EKF restart. 75
Figure 34. Scenario 2: error with respect to the nominal path – with EKF restart. 76
Figure 35. Scenario 2: estimated path on xy plane – with EKF restart and map constraints included. 76

Figure 36. Scenario 2: estimated path on xy plane, zoom on the initial simulation times – with EKF restart and map constraints included. 77
Figure 37. Scenario 2: estimated path on xy plane, zoom on the final simulation times – with EKF restart and map constraints included. 77
Figure 38. Scenario 2: error with respect to the nominal path – with EKF restart and map constraints included. 78
Figure 39. Scenario 3: estimated path on xy plane – with EKF restart 79
Figure 40. Scenario 4: error with respect to the nominal path – with EKF restart. 79
Figure 41. Scenario 3: estimated path on xy plane – with EKF restart 80
Figure 42. Scenario 4: error with respect to the nominal path – with EKF restart. 80
Figure 43. Velocity of the vehicle as estimated by the EKF. 81

3 Table of Acronyms

Acronym	Definition
AGN	Additive Gaussian Noise
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
IMU	Inertial Measurements Unit
INS	Inertial Navigation System
ITS	Intelligent Transportation Systems
LBS	Location Based Services
pdf	Probability Density Function
OSM	OpenStreetMap
PF	Particle Filter
RF	Radio Frequency

Executive summary

Research in positioning technologies has a long history because of the unquestionable benefits in various domains, but its application in the everyday life has increased enormously in recent years with the use of Global Navigation Satellite Systems (GNSS) for road transportation, Location Based Services (LBS) and other positioning technologies. GNSS is not the only available positioning technology; in fact, also Wireless Communication based on ranging and Inertial Measurement Unit (IMU) positioning can provide accurate vehicle positioning. A proof-of-concept project was started in the JRC in early 2014 to investigate and design a framework to improve positioning accuracy in Intelligent Transport Systems (ITS) on the basis of the IMU sensors identified above, to determine the limits of the existing theoretical frameworks and to propose new approaches.

In the first phase of the project, the analysis of the existing theoretical frameworks was completed and a new framework has been defined with the contribution of Prof Vitetta, CNIT/UNIMORE. In a second phase of the project, the theoretical framework was implemented in MATLAB and tested with data derives from measurements on the driving scenario.

This technical report describes in detail the results of the activities carried out during the second phase of the project results and illustrates the technical solution adopted to implement the theoretical framework as a MATLAB library providing dedicated functionalities to apply for an Accurate Positioning in Intelligent Transport System. The mathematical framework developed in the first phase of the project has been implemented into a working and self-standing MATLAB library. The MATLAB library is composed by three functional blocks: the Extended Kalman Filter, the vehicle's status estimation module and the Particle Filter. The MATLAB library is capable of ingesting data in batch or real-time fashion. It elaborates the acquired data and displays the results by means of easy to understand graphs. It interoperates with the OpenStreetMap maps to provide georeferenced results. The report describes the main modules of the MATLAB library, how they are connected and how they must be used. The report also describes the testing scenarios and the results of the tests. Finally, the technical report provides recommendations on the future developments of this MATLAB library.

The intended audience of this technical report are engineers or technicians that are familiar with the basics of mathematical models, system engineering and the MATLAB scripting language.

4 Introduction

Research in positioning technologies has a long history because of the unquestionable benefits in various domains, but its application in the everyday life has increased enormously in recent years with the use of Global Navigation Satellite Systems (GNSS) for road transportation, Location Based Services (LBS) and other positioning technologies.

Even if the GNSS is the predominant positioning technology, it has limitations, which makes its use in many applications not viable: the accuracy of GNSS is limited to 10 meters or more, GNSS signal may not be available in underground areas or urban areas (e.g. due to urban canyons) and it may be subject to attacks like jamming or spoofing.

GNSS is not the only available positioning technology; in fact, also Wireless Communication based on ranging and Inertial Measurement Unit (IMU) positioning can provide accurate vehicle positioning.

Positioning systems using IMU technology use orientation, and gravitational forces through a combination of accelerometers and gyroscopes. A proof-of-concept project was started in the JRC in early 2014 to investigate and design a framework to improve positioning accuracy in Intelligent Transport Systems (ITS) on the basis of the IMU sensors identified above, to determine the limits of the existing theoretical frameworks and to propose new approaches.

In an initial phase of the project, the analysis of the existing theoretical frameworks was completed and a new framework has been defined with the contribution of Prof Vitetta, CNIT/UNIMORE.

The objective of this second phase of the project was to implement the theoretical framework already defined in the first phase of the proof-of-concept project into MATLAB code, which can be integrated in the proof-of-concept system, which uses the sensors identified above.

This phase of the project had the following task breakdown structure:

1. MATLAB implementation of the theoretical framework including related simulation manual.
2. Unit Test of the MATLAB implementation with simulated (computer-generated) data.
3. Support to the integration of the MATLAB implementation with the JRC proof-of-concept.

This document describes in detail the results of the activities carried out during the second phase of the project results and illustrates the technical solution adopted to implement the theoretical framework as a MATLAB library providing dedicated functionalities to apply for an Accurate Positioning in Intelligent Transport System.

The authors of this report developed a MATLAB library implementing the mathematical framework described in [1] with the aim to improve the vehicle's state (position, velocity and direction) estimation ingesting real-time accelerator, gyroscope and GPS data. The mathematical framework presented in [1] has been adjusted to be implementable into a working and self-standing MATLAB library. The MATLAB library is composed by three functional blocks: the Extended Kalman Filter, the vehicle's status estimation module and the Particle Filter. The MATLAB library is capable of ingesting data in batch or real-time fashion. It elaborates the acquired data and displays the results by means of easy to understand graphs. It interoperates with the OpenStreetMap maps to provide georeferenced results.

To test and validate the developed solution, five scenarios have been identified:

(i) straight motion at constant velocity,
(ii) straight motion under constant acceleration,
(iii) circular turn at constant velocity,
(iv) junction at constant velocity
(v) road hump at constant velocity.

For each scenario, the IMU and GPS data have been simulated taking into account the typical characteristics of the sensors equipping the Daisy7 board [3].

The simulation results shows that an open-loop application of the positioning framework described in [1] behaves better than a GPS during the initial phase of the simulation, when the covariance provided by the EKF is relatively small. Unfortunately the information coming from the EKF deteriorates very fast (due to the approximations of nonlinear dynamics), up to the point in which the PF ignores the EKF estimates, and relies only on the GPS information.

To overcome this issue, the authors designed, developed and integrated into the MATLAB library a solution to lower the effect of the EKF estimates degradation, introducing a *EKF restart* that periodically resets the EKF so that the new starting initial state is set to the last EKF state estimation, while the covariance is restarted to the initial covariance value. This closed-loop workaround provides very good results, but requires some tuning to perform in a satisfactory way.

The intended audience of this document are engineers or technicians that are familiar with the basics of mathematical models, system engineering and the MATLAB scripting language.

The document is organised as follows.

Section 5 provides the mathematical foundations to understand the model and the logical framework behind the implemented MATLAB libraries.

Section 6 describes the architecture of the logical framework and details its functional components: the extended Kalman filter, the estimation module and the particle filter.

Section 7 details the implementation of the functional components in a MATLAB simulation environment and describes several test cases used to validate the developed solution.

Section 8 shows the simulations results and analyses the performance of the presented solution with respect to the most representative test cases.

Section 9 and 10 report the authors' conclusions and considerations on the obtained results and on potential future works.

5 The mathematical framework (EKF, PF)

This Section briefly recalls from [1] the mathematical framework based on which the integrated navigation solution has been developed. In particular, the coordinate systems used and the related conventions adopted are clarified; the mathematical modelling of the sensors and of the system at study is detailed. The information provided in this section is preliminary and functional to the definition of the architecture of the integrated navigation solution and of the description of the mathematical details regarding the constituent EKF and PF modules.

5.1 Coordinate Systems

This Section details the coordinate systems used to describe vehicle's position and orientation.

We make reference throughout the course of this document to two distinct reference frames (i.e. coordinate systems) jointly used to univocally define vehicle's position and orientation in space:

1. ***The Navigation Frame.*** The navigation frame is a coordinate system rigidly attached to some fixed point of the space where the navigation task takes place. It can be used to univocally define the position of objects in the navigation environment (e.g. any chosen point of the vehicle, like its center of gravity).
   It is chosen as navigation frame a right-handed coordinate system with the z-axis pointing away from earth.
   The information on the (estimated) position and orientation of the vehicle can be provided to the user making reference to the navigation frame.
2. ***The Body Frame.*** The body frame (also called vehicle frame) is a coordinate system rigidly attached to the vehicle.
   We chose as body frame a right-handed coordinate system with its origin at the center of gravity of the vehicle and with the x-axis pointing forward, the y-axis pointing sideway to the right and the z-axis pointing downward.
   The information provided by on-board inertial sensors (accelerometer and gyroscope) is typically expressed in body frame, since sensors are mounted rigidly attached to the vehicle.

Figure 1 provides a graphical representation of the two employed reference frames.

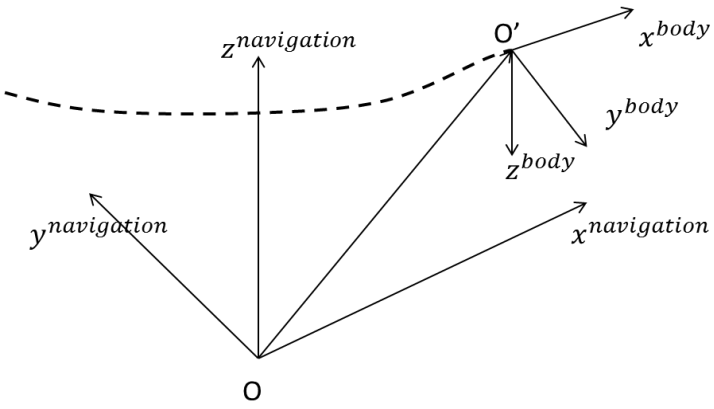

Figure 1. The employed coordinate systems (navigation and body frames).

In this report, the orientation of the body frame with respect to the navigation frame is described by means of the roll, pitch, yaw Euler angles ϕ, θ, ψ, describing, respectively, rotations around the x, y and z axis of the navigation frame.

Based on the Euler angles, the rotation matrix

$$C_b^n = \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta sin\psi - cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \quad (1)$$

can be defined, describing a change of coordinates mapping the representation of a vector from body frame to navigation frame, that is

$$x^n = C_b^n x^b \quad (2)$$

where x^b are the coordinates of a generic vector with respect to the body frame, and x^n are the coordinates of the same vector with respect to the navigation frame.

Matrix C_b^n is orthogonal (i.e., $C_b^{n-1} = C_b^{nT}$).

5.2 Modelling

This section specifies the employed mathematical models of the sensors and of the navigation system at study, as detailed in the following.

5.2.1 IMU and GPS Sensors' Models

This Section reports the mathematical models considered for the IMU sensors (accelerometer and gyroscope) and the GPS. These models appear in the systems dynamics formulated in the next Section as well as in the formulation of the EKF and the PF.

Gyroscope Model

The 3D gyroscope is employed to measure the angular velocity along the three axis. The measure is expressed in body frame and is provided by the gyro in raw data format, as an integer (refer to Section 7.2.2 for the conversion procedure to the international system of units).

The mathematical model relating the actual angular velocity ω_b, expressed in body frame, to the measured one, ω_{bm}, also expressed in body frame, is given by the equation

$$\omega_{bm} = G_\omega \omega_b + \hat{b}_\omega + n_\omega \tag{3}$$

where G_ω is a gain matrix accounting for scaling factors, $\hat{b}_\omega$ is a vector of deterministic sensor bias and n_ω is a measurement noise term modelled as AGN (with covariance $\sigma_\omega^2 I_3$).

Accelerometer Model

The employed 3D accelerometer senses the accelerations along the three axis. Acceleration measurement are expressed in body frame and are provided by the accelerometer in row data format, as integer numbers (refer to Section 7.2.3 for details on the procedure for converting the data format to the international systems of units).

The following model is employed to express the relation between the *true acceleration* in body frame a_b and the *measured acceleration* in body frame a_{bm}

$$a_{bm} = G_a a_b + \hat{b}_a + n_a \tag{4}$$

where G_a is a matrix of gains accounting for scaling, $\hat{b}_a$ is the deterministic bias of the sensor [1] and n_a is the noise affecting the measure (noise modelled as AGN, with covariance $\sigma_a^2 I_3$).

5.2.2 System Dynamics

This Section details the mathematical models of the vehicle's dynamics and the measurement models (i.e. models relating the measured quantities to the state space variables) which enter the Extended Kalman Filter mathematical formulation.

First of all, according to [1], the state of the vehicle at the generic time $t = kT_s$ is described by the 21-dimensional vector

$$x_k = \left[v_k^T,\ a_k^T,\ \Psi_k^T,\ a_k^{bT},\ \omega_k^{b,n\to bT},\ b_k^{aT},\ b_k^{\omega T}\right] \in R^{21} \quad (5)$$

where

- $v_k = \left[v_{x,k},\ v_{y,k}, v_{z,k}\right]^T \in R^3$ represents the vehicle speed (measured in m/s) at time $t = kT_s$, written in the navigation frame.
- $a_k = \left[a_{x,k},\ a_{y,k}, a_{z,k}\right]^T \in R^3$ represents the vehicle acceleration (measured in m/s^2) at time $t = kT_s$, written in the navigation frame.
- $\Psi_k = [\phi_k,\ \theta_k,\ \psi_k]^T \in R^3$ represents the Euler angles (expressed in radiants – rad) describing the orientation of the body frame with respect to the navigation frame at time $t = kT_s$.
- $a_k^b = \left[a_{x,k}^b,\ a_{y,k}^b, a_{z,k}^b\right]^T \in R^3$ represents the vehicle acceleration (measured in m/s^2) expressed in body frame. A *measurement* of a_k^b is available from the on-board accelerometer.
- $\omega_k^{b,n\to b} = [\omega_{x,k}^{b,n\to b},\ \omega_{y,k}^{b,n\to b},\ \omega_{z,k}^{b,n\to b}] \in R^3$ represents the angular velocity (expressed in rad/s) of the vehicle expressed in body frame. A *measurement* of $\omega_k^{b,n\to b}$ is available from the on-board gyroscope.
- b_k^{aT}, $b_k^{\omega T} \in R^3$ represents, respectively, the bias instability of the accelerometer and the gyroscope, expressed in body frame. b_k^{aT} is measured in m/s^2, $b_k^{\omega T}$ is expressed in rad/s.

In the following, we first derive the mathematical models specifying the dynamics of each of the above seven components of the state vector, and then provide the collective dynamics governing the evolution of the state vector.

Dynamics of v_k

The dynamics of v_k is given by the first order Taylor expansion of the equation of the motion of a rigid body

$$v_{k+1} = v_k + a_k T_s + n_{v,k} \quad (6)$$

where $n_{v,k}$ is an Average Gaussian Noise (AGN) vector capturing modelling errors.

Dynamics of a_k

The dynamics of a_k can be derived starting from the equation

$$a_{k+1} = C_b^n(\Psi_{k+1})a_{k+1}^b - g \quad (7)$$

which relates the acceleration in the body frame with the acceleration in the navigation frame. According to [1] we the choose a random walk model for the dynamics of a_k^b

$$a_{k+1}^b = a_k^b + n_{a^b,k} \tag{8}$$

where $n_{a^b,k}$ is an AGN term characterized by covariance matrix $C_{a^b} = \sigma_{a^b}^2 I_3$, being I_n the identity matrix of dimension n. By plugging (8) into (7) and posing the definition

$$\Delta C_b^n(\Psi_k, \Psi_{k+1}) \triangleq C_b^n(\Psi_{k+1}) - C_b^n(\Psi_k) \tag{9}$$

we get

$$a_{k+1} = C_b^n(\Psi_k) a_k^b - g + n_{a,k} \tag{10}$$

being $n_{a,k}$ an additive noise term given by

$$n_{a,k} = \Delta C_b^n(\Psi_k, \Psi_{k+1}) a_k^b + C_b^n(\Psi_{k+1}) n_{a_k^b,k} \tag{11}$$

According to [1], we model $n_{a,k}$ as an AGN term, with covariance matrix $C_a = \sigma_a^2 I_3$, with $\sigma_a^2 > \sigma_{a^b}^2$.

Concluding, the dynamics of a_k is given by

$$a_{k+1} = C_b^n(\Psi_k) a_k^b - g + n_{a,k} \tag{12}$$

Dynamics of Ψ_k

Dynamics of Ψ_k is given by the standard discrete time equation for the angular motion of a rigid motion

$$\Psi_{k+1} = \Psi_{\mathrm{k}} + T_s A_{s\omega}(\Psi_{\mathrm{k}})\omega_k^{b,n\to b} + n_{\Psi,k} \tag{13}$$

where

$$A_{s\omega}(\Psi_{\mathrm{k}}) \triangleq \begin{bmatrix} 1 & sin\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi/cos\theta & cos\phi/cos\theta \end{bmatrix} \tag{14}$$

and $n_{\Psi,k}$ ia an AGN term accounting for model inaccuracies, with covariance matrix $C_\Psi = \sigma_\Psi^2 I_3$.

Dynamics of a_k^b

As already specified in the above while discussing about the dynamics of a_k, the dynamics of a_k^b is modelled as a random walk

$$a_{k+1}^b = a_k^b + n_{a^b,k} \tag{15}$$

where $n_{a^b,k}$ is an AGN term characterized by covariance matrix $C_{a^b} = \sigma_{a^b}^2 I_3$.

Dynamics of $\omega_k^{b,n\to b}$

The dynamics of $\omega_k^{b,n\to b}$ is modelled as a random walk

$$\omega_{k+1}^{b,n\to b} = \omega_k^{b,n\to b} + n_{\omega^b,k} \tag{16}$$

where $n_{\omega^b,k}$ is an AGN term characterized by covariance matrix $C_{\omega^b} = \sigma_{\omega^b}^2 I_3$.

Dynamics of b_k^a

The following autoregressive model is employed to model the dynamics of $b_k^{a^T}$

$$b_{k+1}^a = diag\{\beta^a\} b_k^a + n_{b^a,k} \tag{17}$$

where β^a is the three-dimensional vector of AR(1) coefficients and $n_{b^a,k}$ an AGN term characterized by covariance matrix $C_{b^a} = \sigma_{b^a}^2 I_3$.

Dynamics of $b_k^{\omega^T}$

Similarly to b_k^a, the following autoregressive model is employed to model the dynamics of b_k^ω

$$b_{k+1}^\omega = diag\{\beta^\omega\} b_k^\omega + n_{b^\omega,k} \tag{18}$$

where β^ω is the three-dimensional vector of AR(1) coefficients and $n_{b^\omega,k}$ an AGN term characterized by covariance matrix $C_{b^\omega} = \sigma_{b^\omega}^2 I_3$

Collective dynamics of the system

The collective dynamics of the system is nonlinear (see (12) and (13)) and is given by the collection of equations (6), (12), (13), (15), (16), (17) and (18)

$$x_{k+1} = f_x(x_k) - Mg + n_k \tag{19}$$

where

$$f_x(x_k) = \begin{bmatrix} I_3 & T_s I_3 & 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & C_b^n(\Psi_k) & 0_{3,3} & 0_{3,3} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & I_3 & 0_{3,3} & T_s A_{s\omega}(\Psi_k) & 0_{3,3} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & I_3 & 0_{3,3} & 0_{3,3} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & I_3 & 0_{3,3} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & diag\{\beta^a\} & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & diag\{\beta^\omega\} \end{bmatrix} \tag{20}$$

$$M = \left[0_{3,3}, I_{3,3}, 0_{3,3}, 0_{3,3}, 0_{3,3}, 0_{3,3}, 0_{3,3}\right]^T \tag{21}$$

$$n_k = \left[n_{v,k}, n_{a,k}, n_{\Psi,k}, n_{a^b,k}, n_{\omega^b,k}, n_{b^a,k}, n_{b^\omega,k}\right]^T \tag{22}$$

Based on the above discussion about the system dynamics, it can be stated that the pdf of x_{k+1} conditioned to x_k is given by

$$f(x_{k+1}|x_k) = N(x_{k+1}, f_x(x_k), C_n) \tag{23}$$

where $N(x_{k+1}, f_x(x_k), C_n)$ denotes the Gaussian pdf with mean $f_x(x_k)$ and covariance C_n (i.e. C_n denotes the covariance matrix of the AGN term n_k, assumed diagonal).

5.2.3 Measurement models

Measurement models are employed to write the IMU-sensed quantities as a function of the state space (5). The two models (the first for the acceleration, the second for the angular velocity) are given by

$$z_k^a = a_k^b + b_k^a + diag\{w_a\}m_{a,k} \tag{24}$$

$$z_k^\omega = \omega_{k+1}^{b,n\to b} + b_k^\omega + diag\{w_\omega\}m_{\omega,k} \tag{25}$$

where z_k^a and z_k^ω denote the calibrated acceleration and angular velocity measurement, $m_{a,k}$ and $m_{\omega,k}$ are AGN terms with covariance matrices $C_{z,a} = \sigma_{z,a}^2 I_3$ and $C_{z,\omega} = \sigma_{z,\omega}^2 I_3$, respectively, and w_a and w_ω are three-dimension vectors accounting for possible asymmetries along the three measurement axes of the accelerometer and the gyroscope.

The resulting collective measurement model is given by

$$z_k = \left[\, z_k^{aT} \quad z_k^{\omega T}\right]^T = f_z(x_k) + m_k \tag{26}$$

where

$$f_z(x_k) \triangleq \begin{bmatrix} 0_{3,3} & 0_{3,3} & 0_{3,3} & I_3 & 0_{3,3} & I_3 & 0_{3,3} \\ 0_{3,3} & 0_{3,3} & 0_{3,3} & 0_{3,3} & I_3 & 0_{3,3} & I_3 \end{bmatrix} x_k \tag{27}$$

and

$$m_k \triangleq \left[\left(diag\{w_a\} m_{a,k} \right)^T \quad \left(diag\{w_\omega\} m_{\omega,k} \right)^T \right]^T \tag{28}$$

Based on the above discussion about the measurement model, it can be stated that the pdf of z_k conditioned to x_k is given by

$$f(z_k|x_k) = N(z_k, f_z(x_k), C_m) \tag{29}$$

where $N(z_k, f_z(x_k), C_m)$ denotes the Gaussian pdf with mean $f_z(x_k)$ and covariance C_m (i.e. C_m denotes the assumed diagonal covariance matrix of the AGN term m_k).

6 Architecture of the Integrated INS/GPS Navigation system and modules' description

The aim of this section is to present the architecture of the implemented Integrated INS/GPS Navigation system (depicted in Figure 2) and to give the fundamental mathematical details of the modules appearing in the architecture.

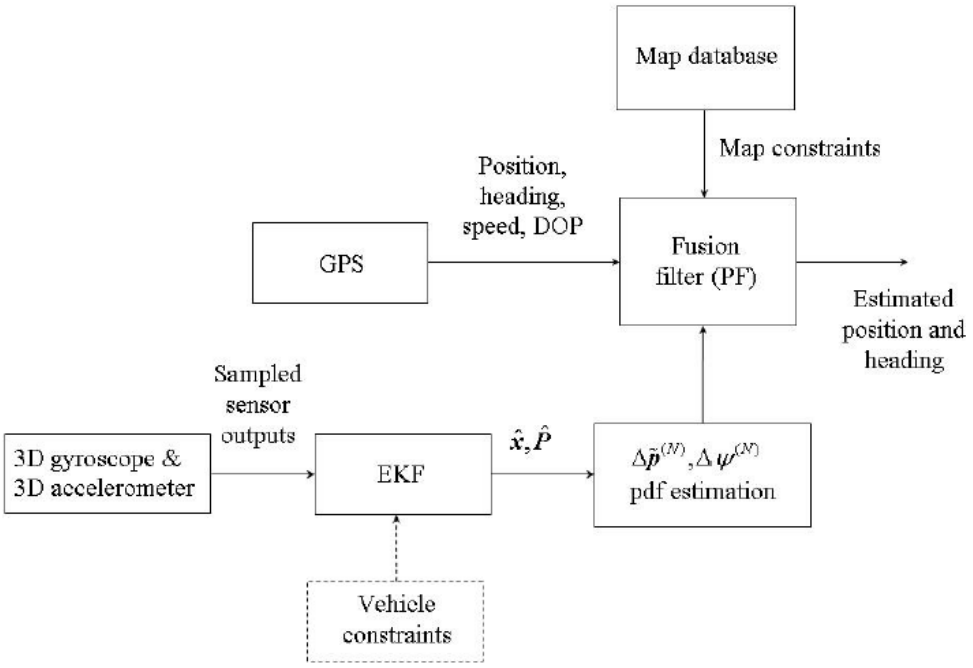

Figure 2. Architecture of the implemented integrated INS/GPS Navigation system.

First of all, the navigation systems takes as input both a priori information on the environment in which the navigation task takes place (i.e. knowledge from map database) and information sensed in near-real through on board sensors (3D gyroscope, 3D accelerometer, GPS). Also, as explained in [1] pag. 50-51, vehicle nonholonomic constraints can be exploited as an additional source of information (a sort of virtual measure), and for this reason they appear in the architecture, in dashed line (to mean they are a source of virtual information). The main navigation modules implemented are, as reported in the figure:

(1) The EKF module, elaborating information (i.e. measurements coming from the gyroscope and the accelerometer).
(2) The "pdf estimation" module, elaborating the estimates coming from the EKF module and providing adaptation between the EKF module and the Fusion filter module.
(3) The Fusion filter module (also called in this report particle filter module), in charge of deriving the actual estimation of vehicle position and attitude, and taking inputs from the "pdf estimation" module and the GPS.

The EKF module works at a faster smapling time with respect to the Fusion filter module. It is responsible for estimating the current state of the system (among the others, of the vehicle velocity, acceleration and attitude) based on the measurements coming from the gyroscope and the accelerometer (and also taking into account, in an advanced configuration, the information from vehicle constraints). The EKF estimate is accompanied by a measure of its accuracy given by the covariance matrix associated to the estimated state vector. EKF output is input to the "pdf estimation" module, which elaborates an estimate of the linear and angular displacement of the vehicle occurred between to elaborations of the Fusion filter module (which runs slower than the EKF module). It is important to recall and note that the EKF output consists of incremental, differential information, as it does not refer to an estimate of the absolute position of the vehicle, but rather it allows estimating the *displacement* of the vehicle occurred

during each time step of EKF elaboration. So, the "pdf estimation" module is in a certain sense in charge of integrating the EKF information in order to derive the estimation of vehicle position and attitude change between two consecutive executions of the Fusion filter module. The Fusion filter module then "grounds" the differential information provided by the "pdf estimation" module by fusing it with the absolute position information coming from the GPS.

In the following, the mathematical details of the three modules introduced above are briefly discussed.

6.1 Extended Kalman Filter

The Extended Kalman Filter is employed to achieve a sub-optimal estimation of the first two moments (i.e. mean value and covariance) of the posterior pdf $f(x_k|z_{0:k})$, where $z_{0:k}$ denotes the cumulative information from IMU sensors (i.e. $z_{0:k} = \{z_l, l = 1, \dots, k\}$).

It is an iterative procedure in which, at the generic time step $k+1$, the mean value of the state $\hat{x}_k$ and the associated covariance matrix $\hat{P}_k$ are estimated by performing in sequence a *prediction step* and an *update step*, as detailed in the following.

Generic EKF iteration at time step $k+1$

1. **Prediction Step**

In the prediction step, *a priori* approximations (denoted with, respectively, $\hat{x}_{k+1|k}$ and $\hat{P}_{k+1|k}$) of the mean state vector $\hat{x}_{k+1}$ and the associated covariance matrix $\hat{P}_{k+1}$ are computed based on the approximations $\hat{x}_k$ and $\hat{P}_k$ available from the previous time step k and by using the state transition model, as follows

$$\hat{x}_{k+1|k} = f_x(\hat{x}_k) - Mg \tag{30}$$

$$\hat{P}_{k+1|k} = J_x(\hat{x}_k)\hat{P}_k\big(J_x(\hat{x}_k)\big)^T + C_n \tag{31}$$

where $J_x(\hat{x}_k)$ denotes the Jacobian of f_x, for whose expression the reader is referred to [1].

2. **Update Step**

In the update step, $\hat{x}_{k+1}$ and $\hat{P}_{k+1}$are derived starting from the a priori approximations $\hat{x}_{k+1|k}$ and $\hat{P}_{k+1|k}$, and tacking into account the additional information coming at time step from the sensors, as follows.

Computation of $\hat{x}_{k+1}$

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + K_{k+1|k} s_{k+1|k} \tag{4}$$

where:

- $s_{k+1|k}$ is the so called *innovation residual*

$$s_{k+1|k} \triangleq z_k - f_z(\hat{x}_{k+1|k}) \tag{32}$$

with estimated covariance matrix given by

$$S_{k+1|k} = J_z(\hat{x}_{k+1|k}) \hat{P}_{k+1|k} \left(J_z(\hat{x}_{k+1|k})\right)^T + C_m \tag{33}$$

where $J_z(\hat{x}_{k+1|k})$ denotes the Jacobian of $f_z(x_k)$.

- $K_{k+1|k}$ is the so called *Kalman gain*

$$K_{k+1|k} = \hat{P}_{k+1|k} \left(J_z(\hat{x}_{k+1|k})\right)^T S_{k+1|k}^{-1} \tag{34}$$

Computation of $\hat{P}_{k+1}$

$$\hat{P}_{k+1} = [I - K_{k+1|k} J_z(\hat{x}_{k+1|k})] \hat{P}_{k+1|k} \tag{35}$$

Two additional fundamental steps to be performed regard the initialization of the EKF, which deals with the choice of the initial value x_0 (i.e. *initialization of the EKF*), and the tuning of the many covariance matrices appearing in the above formulas.

Initialization of the EKF

The initial state x_0 clearly depends on the initial configuration of the vehicle. As an example, assuming that the vehicle is still at time step $k = 0$, we have:

- $v_0 = [0\ 0\ 0]^T$.
- $a_0 = [0\ 0\ 0]^T$.
- Ψ_0 can be derived by employing a magnetometer or an orientation filter processing the data from the accelerometer (to infer the orientation of the vehicle with respect to the gravity vector).
- $a_0^b = [0\ 0\ 0]^T$.
- $\omega_k^{b,n\to b} = [0\ 0\ 0]^T$.
- $b_0^a = [0\ 0\ 0]^T$, see [1].
- $b_0^\omega = [0\ 0\ 0]^T$, see [1].

Selection of Covariance Matrices

- *Covariance matrix C_n associated to the state vector* (covariance matrix of the AGN term n_k in (19)).
  An empirical selection of C_n has been carried out, since, as stated in [1], the indications available in literature about the selection of C_n are difficultly applicable due to the model approximations and simplifications performed.
- *Covariance matrix C_m associated to the state vector* (covariance matrix of the AGN term m_k in (26)).
  C_m is the covariance characterizing the AGN affecting the measured acceleration and angular velocity. It has been therefore estimated based on the real measurement data available, through the method described in [2].

6.2 $\Delta\tilde{p}^{(N)}, \Delta\psi^{(N)}$ pdf Estimation Module

Based on the inputs from the EKF module, the "$\Delta\tilde{p}^{(N)}, \Delta\psi^{(N)}$ pdf Estimation Module" (also shortly referred to as "pdf Estimation Module" in this report) is in charge of providing to the particle filter module, with a periodicity of NT_s seconds, information about the displacement of the vehicle (change in vehicle position and attitude) occurred between in the interval $[(k-N)T_s, kT_s]$. The displacement of the vehicle on the moving plane in the interval $[(k-N)T_s, kT_s]$ is denoted with $\Delta\tilde{p}_k^{(N)} = \left[\Delta p_{x,k}^{(N)}, \Delta p_{y,k}^{(N)}\right] \triangleq \tilde{p}_k - \tilde{p}_{k-N}$, with $\tilde{p}_k = [\tilde{p}_{x,k}, \tilde{p}_{y,k}]$ being the position of the vehicle on the plane at time kT_s.

The change of attitude of the vehicle occurred in the interval $[(k-N)T_s, kT_s]$ is denoted with $\Delta\psi_k^{(N)} \triangleq [\psi_k - \psi_{k-N}]$.

Linear and angular displacements are collected in the vector $\Delta r_k = \left[\Delta p_{x,k}^{(N)}, \Delta p_{y,k}^{(N)}, \Delta \psi_k^{(N)}\right]^T$.

Variable Δr_k is modelled as a Gaussian vector $N(\Delta r_k, \overline{\Delta r_k}, C_{\Delta \mathrm{r,k}})$. The "$\Delta\tilde{p}^{(N)}, \Delta\psi^{(N)}$ pdf Estimation Module" has to compute and provide to the PF module the mean value $\overline{\Delta r_k}$ and the covariance $C_{\Delta \mathrm{r,k}}$ of Δr_k.

It is straightforward to see that the mean value of the linear displacement, $\overline{\Delta\tilde{p}^{(N)}}$ can be evaluated as

$$\overline{\Delta\tilde{p}^{(N)}} \triangleq E\left\{\Delta\tilde{p}^{(N)}\right\} = \sum_{l=k-N-1}^{k} E\{\Delta\tilde{p}_l\} \tag{36}$$

The displacement $\Delta\tilde{p}_l$ can be approximated as

$$\Delta\tilde{p}_l = \tilde{v}_l T_s + \frac{1}{2}\tilde{a}_l T_s + n_{p,l} \tag{37}$$

being $\tilde{v}_l = \left[v_{x,l}, v_{y,l}\right]^T$, $\tilde{a}_l = \left[a_{x,l}, a_{y,l}\right]^T$ and $n_{p,l}$ an AGN term of covariance $C_{\tilde{\mathrm{p}}} = \sigma_p I_2$.

Therefore $\overline{\Delta\tilde{p}^{(N)}}$ can be written as

$$\overline{\Delta\tilde{p}^{(N)}} = \sum_{l=k-N-1}^{k} \left(\bar{\tilde{v}}_l T_s + \frac{1}{2}\bar{\tilde{a}}_l T_s\right) \tag{38}$$

where the mean values $\bar{\tilde{\mathrm{v}}}_\mathrm{l}$ and $\bar{\tilde{\mathrm{a}}}_\mathrm{l}$ are estimated by the EKF (velocity and acceleration are included in the state of the EKF).

Similarly, under proper independence hypothesis, the covariance $C_{\Delta\tilde{p}^{(N)},k}$ of $\Delta\tilde{p}^{(N)}$ can be evaluated as

$$C_{\Delta\tilde{p}^{(N)},k} = \sum_{l=k-N-1}^{k} C_{\Delta\tilde{p},l} = \sum_{l=k-N-1}^{k} \left(C_{\tilde{v},l}T_s^2 + \frac{1}{4}C_{\tilde{a},l}T_s^4 + C_{\tilde{v}\tilde{a},l}T_s^3 + C_{\tilde{p}}\right) \tag{39}$$

where estimates of all the covariances appearing are available from the EKF.

As far as concerns the change of the vehicle attitude, $\Delta\psi_k^{(N)}$, it can be evaluated again as

$$\Delta\psi_k^{(N)} = \sum_{l=k-N-1}^{k} \Delta\psi_l \tag{40}$$

and its expectation $\overline{\Delta\psi_k^{(N)}}$ as

$$\overline{\Delta\psi_k^{(N)}} = \sum_{l=k-N-1}^{k} \overline{\Delta\psi_l} = E\{\psi_l\} - E\{\psi_{l-1}\} \tag{41}$$

where estimates of $E\{\psi_l\}$ and $E\{\psi_{l-1}\}$ are available from the EKF (since the yaw angle is included in the state space).

Also the variance $\sigma^2_{\Delta\psi_k^{(N)},k}$ of $\Delta\psi_k^{(N)}$ can be evaluated in a similar way

$$\sigma^2_{\Delta\psi_k^{(N)},k} = \sum_{l=k-N-1}^{k} \sigma^2_{\Delta\psi,l} \tag{42}$$

Again, an approximation of $\sigma^2_{\Delta\psi,l}$ can be worked out based on information from the EKF. However, the procedure is rather complex, given the fact that $\Delta\psi$ does not belong to the EKF state space. The reader is referred to [1] for details.

Finally, the reader should note that the tuning of $C_{\tilde{p}}$ is needed.

6.3 Particle Filter

The mean state vector $\hat{x}_k$ and the state covariance matrix $\hat{P}_k$ generated by the EKF are processed to estimate the joint pdf of $\Delta\hat{p}_k^{(N)}$ and $\Delta\psi_k^{(N)}$, i.e. the pdf of the vector $\Delta r_k \triangleq [\,\Delta p_{x,k}^{(N)},\ \Delta p_{y,k}^{(N)},\ \Delta\psi_k^{(N)}]^T$. However, since the particle filter runs N times more slowly than the EKF, the parameters of these joint pdf are updated every NT_s; in the following we assume that this occurs when the time index *k* of the EKF is a multiple of *N*. These statistical information are then fused with the information about position and heading coming from the GPS module and from those coming from the available map to generate the final estimates $\hat{p}_l = [\hat{p}_{x,l}, \hat{p}_{y,l}]^T$ and $\hat{\psi}_l$ of the 2D position vector $\tilde{p}_l = [\mathrm{p}_{x,l}, \mathrm{p}_{y,l}]^T$ and heading ψ_l, respectively, with $l = \lfloor k/N \rfloor$. In this document, the following state vector has been used:

$$x_l^{PF} \triangleq [r_l^T, L_l]^T \tag{43}$$

where

$$r_l \triangleq [p_{x,l}, p_{y,l}, \psi_l]^T \tag{44}$$

is a vector collecting the 2D position and the yaw, and L_l represents a link of the map. In the following is discussed how the pdf of r_l can be estimated via particle filtering; in fact, L_l can be easily related to r_l. Estimating such a pdf raises the problem of how the differential information extracted from the EKF can be combined with the absolute information coming from the GPS module. To show how this can be accomplished, we need to define a measurement model and a state model for the particle filter. In principle, in the considered problem the *l*-th measurement vector z_l consists of two contributions, i.e. the 3D vector

$$z_l^{INS} = \overline{\Delta r_k} = r_l - r_{l-1} + n_l^{INS} \tag{45}$$

coming from the INS (always available) and the 3D vector

$$z_l^{GPS} = r_l + n_l^{GPS} \tag{46}$$

from the GPS module (sometimes unavailable); here n_l^{INS} and n_l^{GPS} denote the zero mean (Gaussian) noise contributions affecting INS and GPS data and characterized by the covariance matrices $C_{\Delta r,k}$ (evaluated in the first stage) and $C_{GPS,l}$ respectively (statistical independence can be assumed between these noise vectors). In this case defining a state model requires developing a probabilistic model for the vehicle movements. If a state transition $r_{l-1} \rightarrow r_l$ violates some constraint originating from the map (e.g., it is associated with a path intersecting the walls of a building), $p(r_l \mid r_{l-1}) = 0$ is selected. On the contrary, if no constraint is violated, a simple Markovian model is assumed; in practice, it is assumed that:

- the sequence $\{r_l - r_{l-1}\}$ is a Gaussian process, consisting of independent vectors;
- for any l $(r_l - r_{l-1})$ is characterized by a zero mean and a covariance matrix $C_{\mathrm{n}r,l}$, and its elements are independent (their variances can be adapted to the vehicle speed, estimated by $|\overline{\Delta r_k}|/\,T_s'$).

It is worth mentioning that particle filtering is employed to approximate the posterior pdf $p(r_l \mid z_{0:l})$ of $r_l$at step l, given the vector $z_{0:l} \triangleq [z_0^T, z_1^T, \ldots., z_l^T]^T$ (which contains the available noisy observations available up to that step), as a sum of weighted delta functions associated with a set of Np distinct particles,
i.e. as

$$p(r_l|Z_l) \approx \sum_{j=0}^{N_p-1} w_l^{(j)} \cdot \delta(r_l - r_l^{(j)}) \tag{47}$$

For any j the j-th particle is drawn according to a so-called proposal (or importance) density $(r_l\,|r_{l-1}^{(j)}, z_l)$, which, in principle, should be also used for the evaluation of the particle weigths; in fact, the weight $w_l^{(j)}$ for the j-th particle $r_l^{(j)}$ should be evaluated as

$$w_l^{(j)} \propto w_{l-1}^{(j)} \frac{p\left(z_l \middle| r_l^{(j)}\right) p\left(r_l^{(j)} \middle| r_{l-1}^{(j)}\right)}{q(r_l\,|r_{l-1}^{(j)}, z_l)} \tag{48}$$

where $p(z_l|r_l)$ ($p(r_l|r_{l-1})$) denotes the pdf of z_l (r_l) conditioned on r_l (r_{l-1}).
A proper choice (even if not the optimal one) for the proposal density is often the prior

$$q(r_l\,|r_{l-1}^{(j)}, z_l) = p\left(r_l \middle| r_{l-1}^{(j)}\right) \tag{49}$$

So that (47)yields

$$w_l^{(j)} \propto w_{l-1}^{(j)}\, p\left(z_l \middle| r_l^{(j)}\right) \tag{50}$$

In the considered problem, in formulating the proposal density we assume that z_l consists of the INS differential information only; this choice is motivated by the fact that INS data are always available and, consequently, are able to drive the navigation system continuosly. In practice, this means that INS data only are exploited in the generation of new particles according to a given proposal density; GPS data, however, can still play an important role, since when available, can be used, for instance, to define a confidence region, such that particles falling outside it are discarded, as discussed in more detail below. Under this assumption the proposal density (48) becomes

$$q(r_l\,|r_{l-1}^{(j)}, z_l^{INS}) = p\left(r_l \middle| r_{l-1}^{(j)}, z_l^{INS}\right) = \mathcal{N}\left(r_l; r_{l-1}^{(j)} + z_l^{INS}, C_{\Delta r,k}\right) = \mathcal{N}\left(r_l; r_{l-1}^{(j)} + \overline{\Delta r_k}, C_{\Delta r,k}\right) \tag{51}$$

Moreover, the weight update equation becomes

$$\begin{aligned} w_l^{(j)} &\propto w_{l-1}^{(j)}\, p\left(z_l^{INS} \middle| r_{l-1}^{(j)}\right) = w_{l-1}^{(j)} \int p\left(z_l^{INS} \middle| \tilde{r}_l, r_{l-1}^{(j)}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l \\ &= w_{l-1}^{(j)} \int \mathcal{N}\left(\overline{\Delta r_k}; \tilde{r}_l - r_{l-1}^{(j)}, C_{\Delta r,k}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l \\ &= w_{l-1}^{(j)} \int \mathcal{N}\left(\tilde{r}_l; \overline{\Delta r_k} + r_{l-1}^{(j)}, C_{\Delta r,k}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l \end{aligned} \tag{52}$$

The integral

$$\int \mathcal{N}\left(\tilde{r}_l; \overline{\Delta r_k} + r_{l-1}^{(j)}, C_{\Delta r,k}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l = \int \mathcal{N}\left(\tilde{r}_l; \overline{z_l^{INS}} + r_{l-1}^{(j)}, C_{\Delta r,k}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l \tag{53}$$

appearing in the right hand side of (51) does not admit a closed form solution, but can be approximated with an arbitray accuracy by means of particle filtering methods. This requires generating $\widetilde{N}_p$ particles $\{y_l^{(q)}, q = 0,1,\dots,\widetilde{N}_p - 1\}$ (independent of the index j of $r_{l-1}^{(j)}$ and consequently to be computed only once for any l), which are drawn from the pdf $\mathcal{N}(y_l; 0, C_{\Delta r,k})$; in fact, given these particles, the integral (52) is approximated as

$$\int \mathcal{N}\left(\tilde{r}_l; \overline{\Delta r_k} + r_{l-1}^{(j)}, C_{\Delta r,k}\right) p\left(\tilde{r}_l \middle| r_{l-1}^{(j)}\right) d\tilde{r}_l \cong \frac{1}{\widetilde{N}_p} \sum_{q=0}^{\widetilde{N}_p - 1} p\left(y_l^{(q)} + \overline{\Delta r_k} + r_{l-1}^{(j)} \middle| r_{l-1}^{(j)}\right) \tag{54}$$

where

$$p\left(y_l^{(q)} + \overline{\Delta r_k} + r_{l-1}^{(j)} \middle| r_{l-1}^{(j)}\right) = 0 \tag{55}$$

if the state transition $r_{l-1} \rightarrow r_l$ does not satisfy map constraints and

$$p\left(y_l^{(q)} + \overline{\Delta r_k} + r_{l-1}^{(j)} \middle| r_{l-1}^{(j)}\right) = \mathcal{N}\left(y_l^{(q)} + \overline{\Delta r_k}; 0, C_{\mathrm{n}r,k}\right) \tag{56}$$

if satisfies them.

Given a proposal density and weight update equation, particle filtering operates in a recursive fashion; the l-th recursion (with l = 1,2,…) evolves through the following steps for the j-th particle (with $j = 0,1,\dots,N_p - 1$)

(1) Draw $r_l^{(j)} \sim q(r_l | r_{l-1}^{(j)}, z_l)$;
(2) Assign the weight $w_l^{(j)}$ to the particle $r_l^{(j)}$ according to (51).

Once the set $\{r_l^{(j)}, w_l^{(j)}\}$ collecting all the N_p new particles and their weights is available, the total weight

$$W = \sum_{j=0}^{N_p - 1} w_l^{(j)} \tag{57}$$

is computed, and weight normalization is accomplished evaluating

$$\tilde{w}_l^{(j)} = w_l^{(j)} W^{-1} \tag{58}$$

and assigning the new weight $\tilde{w}_l^{(j)}$ to $r_l^{(j)}$ (with $j = 0,1,\dots,N_p - 1$). A common problem with the particle filter described above is the so called degeneracy phenomenon; this means that, after a few

iterations, all but one particle are characterized by negligible weights. Degeneracy can be measured by evaluating the estimate

$$\widehat{N_{eff}} = \frac{1}{\sum_{j=0}^{N_p-1}(w_l^{(j)})^2} \tag{59}$$

of the so called effective sample size $\widehat{N_{eff}}$, whose value cannot exceed N_p; in fact, a small $\widehat{N_{eff}}$ indicates severe degeneracy. This problem can be mitigated by properly choosing the proposal density and/or by using resampling when $\widehat{N_{eff}}$ drops below a given threshold N_T. Resampling eliminates particles with small weights and concentrates on particles on large weights. In practice, if a resampling step is added to the particle filter described above (after weight normalization, evaluating $\widehat{N_{eff}}$ and checking if the inequality $\widehat{N_{eff}} < N_T$ holds), a new set of particles $\{\check{r}_l^{(j)}, j = 0,1,\dots,N_p-1\}$ is generated by resampling (with replacement) N_p times from the density

$$\sum_{j=0}^{N_p-1} \tilde{w}_l^{(j)} \cdot \delta(r_l - r_l^{(j)}) \tag{60}$$

so that $\Pr\left\{\check{r}_l^{(j)} = r_l^{(j)}\right\} = \tilde{w}_l^{(j)}$; then, all the particle weights are reset to $1/N_p$.

In the particle _lter described above INS information only has been exploited to derive the proposal density and the weight update equation. GPS data have not been exploited; in addition map information has played a limited role, being employed only in the evaluation of the integral. Let us now discuss how these additional information can be employed. First of all, GPS data are needed for particle initialization, since information about absolute position are available through GPS only. In other words, it is assumed that for $l = 0$ the j-th sample $r_0^{(j)}$ is drawn from $\mathcal{N}\left(r_l^{(j)}; z_0^{GPS}, C_{GPS,0}\right)$ and that the weight $\tilde{w}_l^{(j)} = 1/N_p$ is assigned to it. For $l > 0$, when reliable GPS information are available, they can be exploited to evaluate the weight

$$w_{l,G}^{(j)} = \mathcal{N}\left(r_l^{(j)}; z_l^{GPS}, C_{GPS,l}\right)$$

to be multiplied by $w_l^{(j)}$; in this way particle weighting is influenced by GPS data and particles falling out of a specific confidence region loose relevance. In addition, map information are considered in order to modify the particle weigths, accordingly to (54) and (55). In particular, in this work we model a map constraint as an area outside of which is very unlikely that the vehicle is (for example, the traffic lane).

7 Implementation

This Section reports details on the implementation of the integrated navigation solution. First of all, the tools employed for the purpose are described; then the data sources and the data models (i.e. accelerometer, gyroscope and GPS) are detailed, including details on the needed conversion processes; then the implementation of the three main modules of the navigation solution is discussed, with the MATLAB code of each module reported and commented separately; then the integration of three above modules is detailed, with the MATLAB "orchestration" code (file main.m) reported and commented; finally, the test cases are described and their actual implementation in MATLAB is reported.

7.1 Supporting tools

This Section lists and briefly describes the software tools employed in the implementation of the navigation system.

7.1.1 MATLAB

The MATLAB technical computing language has been employed to write the code library implementing the integrated navigation solution. The library consists of a bundle of .m files implementing the three main modules detailed above and a main file for the launch of the simulations and for the orchestration of the execution of the three modules. More details are provided in the following.

MATLAB has been as well employed to build the simulation environment at the base of the test results shown next in the document (in particular, to process the measurement dataset provided by JRC, to build the artificial signals employed for testing, etc.).

7.1.2 OpenStreetMap and Overpass-turbo

As shown in the previous sections, in order to improve the accuracy of the estimate vehicle positioning, information coming from map should be considered. There are many different ways of using map information and several ways to obtain it. In the following, it is reported a procedure to export free map information (both topological and graphical) provided by OpenStreetMap project, using a web based data mining tool called Overpass-Turbo.

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world (https://wiki.openstreetmap.org). Two major driving forces behind the establishment and growth of OSM have been restrictions on use or availability of map information across much of the world and the advent of inexpensive portable satellite devices.

There are several tools allowing to export OSM maps and one of them is Overpass-Turbo. This tool allows to easily filter data and inspect the results that underlie the open source maps, by using a very practical graphical interface and supporting a lot of different data formats (JSON, KML, OSM, etc).

For more information about Overpass-Turbo, please refer to:http://wiki.openstreetmap.org/wiki/Overpass_turbo.

In the following are listed the steps required to export a OSM map using Overpass-Turbo:

1) Go to http://overpass-turbo.eu/

2) Insert the area of interest
3) Zoom in to visualize only the area that has to be exported
4) Insert the query to select the data to export. For example to export all the streets, insert the following query:

```
<osm-script output="xml" timeout="25">
  <!-- gather results -->
  <union>
    <bbox-query {{bbox}}/>
  </union>
  <!-- print results -->
  <print mode="body"/>
  <recurse type="down"/>
  <print mode="skeleton" order="quadtile"/>
</osm-script>
```

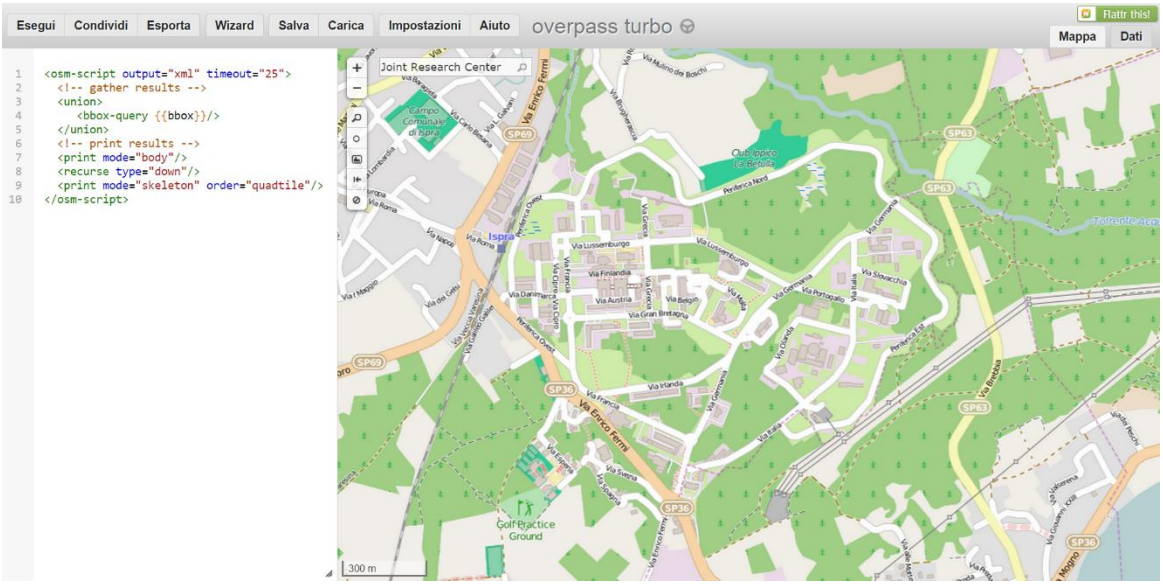

Figure 3. The Overpass-turbo graphical user interface.

5) Execute the query
6) Export the result (the MATLAB libraries proposed in this document takes as input a raw data format file with a .osm extension or a KML file (with .kml extension), but it is also possible to export the result in other formats)
7) To visualize on MATLAB not only the map topology but the image as well, it is possible to export a .png image of the selected area (optional).

In order to import the OSM map and to visualize it on MATLAB, it is possible to download from http://www.mathworks.com/matlabcentral/fileexchange/35819-openstreetmap-functions the OpensStreetMap Functions (and its dependencies), developed by Ioannis Filippidis.

In order to visualize a map on MATLAB using this library, the user has only to modify the map data filename in the "debug_openstreetmap.m" file, as shown below.

```
% example script for using the OpenStreetMap functions
% to illustrate usage of the OpenStreetMap functions for MATLAB
%
% See also PARSE_OPENSTREETMAP, PLOT_WAY, EXTRACT_CONNECTIVITY,
%          GET_UNIQUE_NODE_XY.
%
% 2010.11.25 (c) Ioannis Filippidis, jfilippidis@gmail.com

% download an OpenStreetMap XML Data file (extension .osm) from the
% OpenStreetMap website:
%   http://www.openstreetmap.org/
% after zooming in the area of interest and using the "Export" option to
% save it as an OpenStreetMap XML Data file, selecting this from the
% "Format to Export" options. The OSM XML is specified in:
%   http://wiki.openstreetmap.org/wiki/.osm
openstreetmap_filename = 'map.osm';

%% convert XML -> MATLAB struct
% convert the OpenStreetMap XML Data file donwloaded as map.osm
% to a MATLAB structure containing part of the information describing the
% transportation network
[parsed_osm, osm_xml] = parse_openstreetmap(openstreetmap_filename);

%% plot
% plot the network, optionally a raster image can also be provided for the
% map under the vector graphics of the network
ax = gca;
plot_way(ax, parsed_osm)
%plot_way(ax, parsed_osm, map_img_filename) % if you also have a raster image

%% find connectivity
[connectivity_matrix, intersection_node_indices] = extract_connectivity(parsed_osm);
[uniquend] = get_unique_node_xy(parsed_osm, intersection_node_indices);
```

The map information will be stored in proper data structure (namely parsed_osm, connectivity_matrix, etc) and the area will be shown as reported in Figure 4.

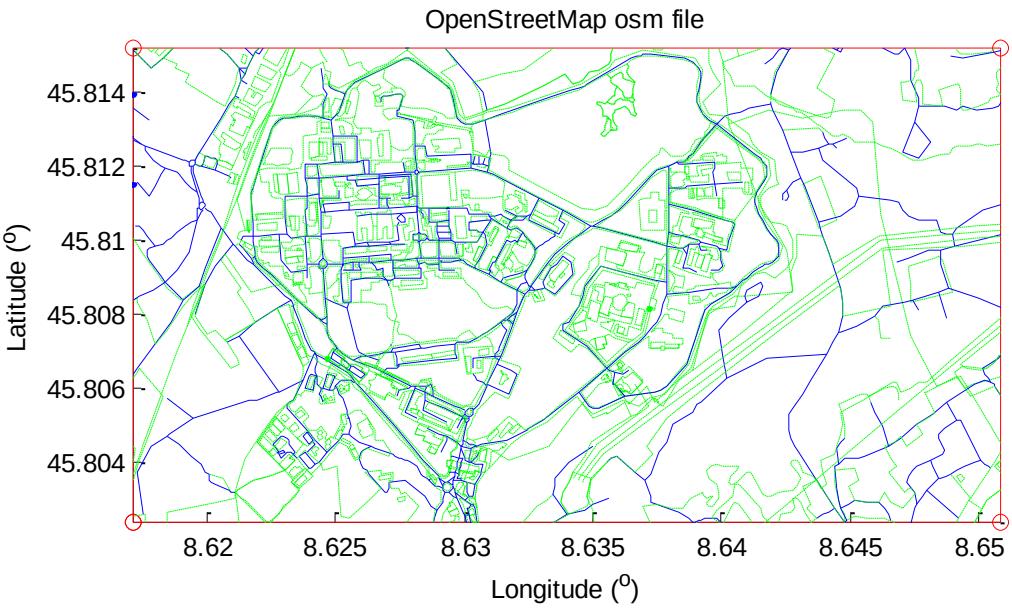

Figure 4. Map information visualised with OpenStreetMap.

7.2 Data Sources and Data Model

Real data from INS and GPS sensors gathered by JRC team have been considered in this study in order to extract the information needed to build proper artificial signal for the testing of the integrated navigation solution.

The accelerometer, gyroscope and GPS measurements refer to the route reported in the Figure below.

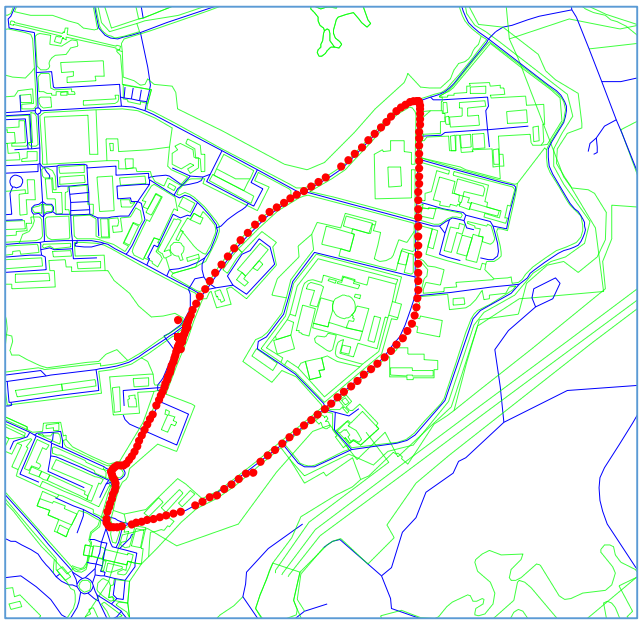

Figure 5. Route followed in the measurement campaigns at ISPRA.

Data have been provided in .cvs files gathering the output of the inertial board (the DAISY-7 - GPS/MEMS module has been employed, depicted in the figure below).

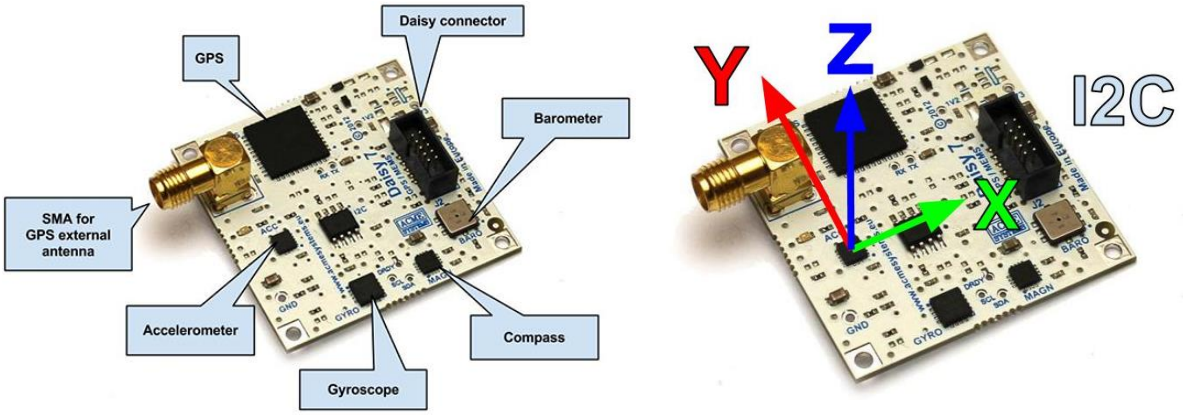

Figure 6. The DAISY-7 - GPS/MEMS module employed in measurement campaigns.

The measurement data have been first converted from the particular row data format to the International System of Units (according to the details reported in the following subsections) and then analysed in order to extract information on the covariance characterising the noise affecting the measurement. As a matter of fact, in this report it is assumed that the measured quantities can be represented as a useful signal corrupted by additive, zero-mean Gaussian noise.

The covariance values of the noise affecting the accelerometer and gyroscope measurements have been estimated following the method described in [2][1]. To the purpose, the file "evar.m" has to be downloaded from http://www.mathworks.com/matlabcentral/fileexchange/25645-noise-variance-estimation/content/evar.m.

7.2.1 GPS

The MATLAB function "importfile" reported below allows importing the data included in the .csv file.

```
function [CPUtime,GPStimestamp,Latitude,NSindicator,Longitude,EWindicator,Qualityindicator,GPSAltitude] = importfile(filename, startRow, endRow)
%IMPORTFILE Import numeric data from a text file as column vectors.
%
[CPUTIME,GPSTIMESTAMP,LATITUDE,NSINDICATOR,LONGITUDE,EWINDICATOR,QUALITYINDICATOR,GPSALTITUDE]
%   = IMPORTFILE(FILENAME) Reads data from text file FILENAME for the
%   default selection.
%
%
[CPUTIME,GPSTIMESTAMP,LATITUDE,NSINDICATOR,LONGITUDE,EWINDICATOR,QUALITYINDICATOR,GPSALTITUDE]
%   = IMPORTFILE(FILENAME, STARTROW, ENDROW) Reads data from rows STARTROW
%   through ENDROW of text file FILENAME.
```

1 A dedicated MATLAB library for the purpose is available at http://www.mathworks.com/matlabcentral/fileexchange/25645-noise-variance-estimation/content/evar.m

```
%
% Example:
%   [CPUtime,GPStimestamp,Latitude,NSindicator,Longitude,EWindicator,Qualityindicator,GPSAltitude]
%   = importfile('gps1.csv',1, 326);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2015/02/24 13:45:34

%% Initialize variables.
delimiter = ',';
if nargin<=2
    startRow = 1;
    endRow = inf;
end

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter, 'HeaderLines',
startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1, 'Delimiter', delimiter,
'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,5,7,8]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
```

```
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-
]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [1,2,3,5,7,8]);
rawCellColumns = raw(:, [4,6]);


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); % Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
CPUtime = cell2mat(rawNumericColumns(:, 1));
GPStimestamp = cell2mat(rawNumericColumns(:, 2));
Latitude = cell2mat(rawNumericColumns(:, 3));
NSindicator = rawCellColumns(:, 1);
Longitude = cell2mat(rawNumericColumns(:, 4));
EWindicator = rawCellColumns(:, 2);
Qualityindicator = cell2mat(rawNumericColumns(:, 5));
GPSAltitude = cell2mat(rawNumericColumns(:, 6));
```

The MATLAB function "visualizeCSV" reported below allows to graphically map the GPS data included in the .csv file on the map image included in "map_img_filename" and the network topology specified by the data structure "parsed_osm", obtained from OpenStreetMap as explained in Section 7.1.2.

```
function [ output_args ] = visualizeCSV( CSV_filename,parsed_osm, map_img_filename )

[gpsTrack.CPUtime,gpsTrack.GPStimestamp,gpsTrack.Latitude,gpsTrack.NSindicator,gpsTrack.Longitude,gpsTrack.EWindicator,gpsTrack.Qualityindicator,gpsTrack.GPSAltitude]= importCSV(CSV_filename,2); %parse the GPX file in a suitable data structure



    a_lat=gpsTrack.Latitude(2);
    b_lat=fix(a_lat/100);
    c_lat=(a_lat-(b_lat*100))*100/60;

    a_lon=gpsTrack.Longitude(2);
    b_lon=fix(a_lon/100);
    c_lon=(a_lon-(b_lon*100))*100/60;



    gpsTrack_clean.Latitude(1) = (b_lat + ((c_lat)/100));    %without column header
    gpsTrack_clean.Longitude(1) = (b_lon + ((c_lon)/100));

    num_clean_coordinates = 1;
    threshold = 0.002;

for i=3:size(gpsTrack.Latitude,1)
    a_lat=gpsTrack.Latitude(i);
    b_lat=fix(a_lat/100);
    c_lat=(a_lat-(b_lat*100))*100/60;

    a_lon=gpsTrack.Longitude(i);
    b_lon=fix(a_lon/100);
    c_lon=(a_lon-(b_lon*100))*100/60;

    if (abs((b_lat + ((c_lat)/100))- gpsTrack_clean.Latitude(num_clean_coordinates))<threshold)  && (abs((b_lon + ((c_lon)/100)) - gpsTrack_clean.Longitude(num_clean_coordinates))<threshold)

            num_clean_coordinates = num_clean_coordinates+1;
            gpsTrack_clean.Latitude(num_clean_coordinates) = (b_lat + ((c_lat)/100));
            gpsTrack_clean.Longitude(num_clean_coordinates) = (b_lon + ((c_lon)/100));
    end

end

%% plot
% plot the network, optionally a raster image can also be provided for the
% map under the vector graphics of the network

ax = gca;  %inizialize axes with current axes handle

if nargin < 3
    map_img_filename = [];  %check if the last input arg isset
end

plot_way(ax, parsed_osm, map_img_filename) % plot the parsed map and if isset also the raster image
```

```
%%
scatter(gpsTrack_clean.Longitude(1),gpsTrack_clean.Latitude(1),'r','filled');  %plot the first GPX point
for i=2:size(gpsTrack_clean.Latitude,2) %for each point contained in the GPX data structure

    %plot a line connecting the current point with the previous one
    plot([gpsTrack_clean.Longitude(i-1),gpsTrack_clean.Longitude(i)],[gpsTrack_clean.Latitude(i-
1),gpsTrack_clean.Latitude(i)],'r','LineWidth',3);
    %plot the current GPX point
    scatter(gpsTrack_clean.Longitude(i),gpsTrack_clean.Latitude(i),'r','filled');

   hold on

   pause(); %pause after each iteration
end

end
```

7.2.2 Gyroscope

The following information characterises the gyroscope row measurement data[2] read from the sensor and provided in the .csv file:

- Measurements are 16 bit integers with sign (i.e. minimum -32768, maximum +32767).
- The accelerometer is operated with a selected full scale of $\pm 250 dps$[3].
- The accelerometer data format is 16 bit.
- The sensitivity of the accelerometer is 8.75/1000 dps per digit.

Given the above information, the conversion from the raw data format (let us denote with ω^{output} the generic sample of angular velocity) to the International System of Units (let us denote with ω^{SI} the corresponding value) can be performed according to the following formula

$$\omega^{SI} = 8.75 \frac{\omega^{output}}{1000} \frac{\pi}{180°} \tag{61}$$

The three figures below report the resulting measured acceleration values, along the three axis of the accelerometer, expressed in the International System of Units.

[2] See the datasheet available at http://www.acmesystems.it/DAISY-7.

[3] dps means degree per second.

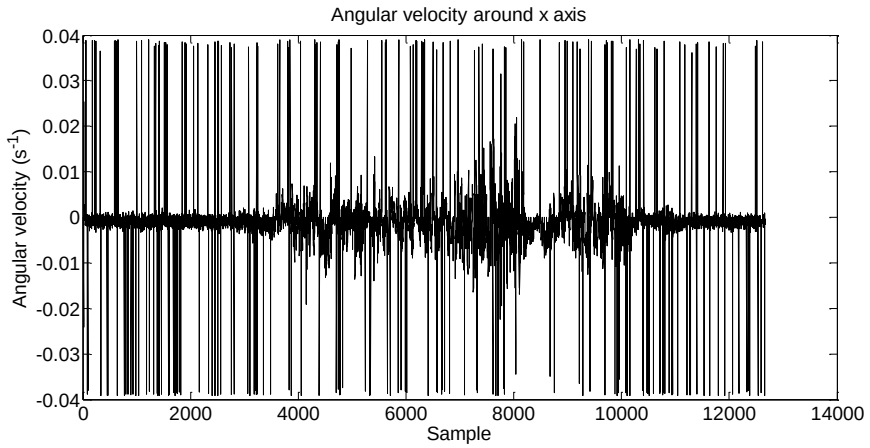

Figure 7 Measured angular velocity along x axis

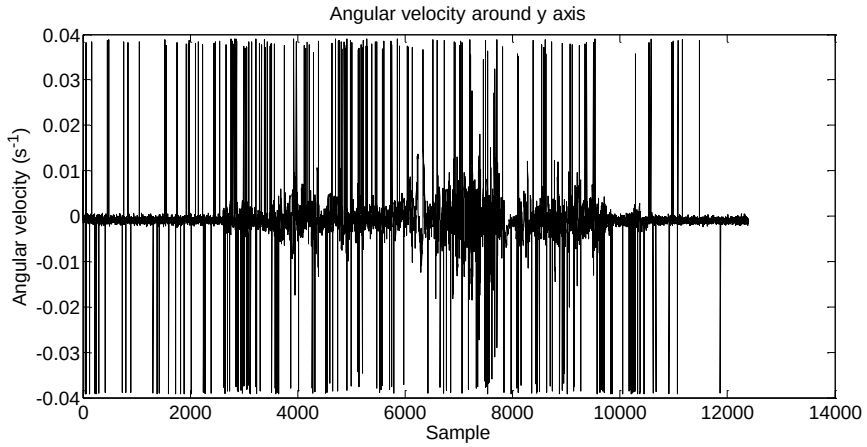

Figure 8 Measured angular velocity along y axis

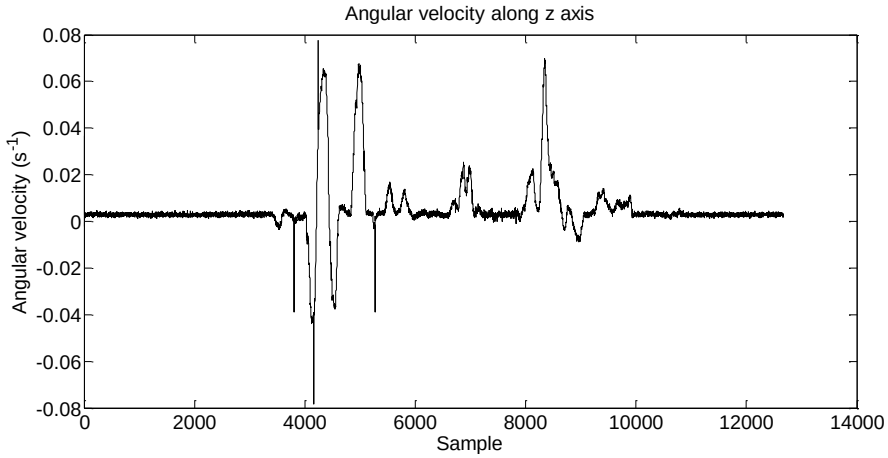

Figure 9 Measured angular velocity along z axis

The estimated covariance values characterizing the additive noises are:

- Along the gyroscope x axis: 3.02e-05.
- Along the gyroscope y axis: 3.20e-05.
- Along the gyroscope z axis: 7.47e-07.

7.2.3 Accelerometer

The following information characterises the accelerometer row measurement data[4] read from the sensor and provided in the .csv file:

- Measurements are 16 bit integers with sign (i.e. minimum -32768, maximum +32767).
- The accelerometer is operated with a selected full scale of $\pm 2g$.
- The accelerometer data format is 12 bit.
- The sensitivity of the accelerometer is g/1000 per digit.

Given the above information, the conversion from the raw data format (let us denote with a^{output} the generic acceleration sample) to the International System of Units (let us denote with a^{SI} the corresponding value) can be performed according to the following formula

$$a^{SI} = \frac{1}{1000}\frac{a^{output}}{16} \tag{62}$$

The three figures below report the resulting measured acceleration values, along the three axis of the accelerometer, expressed in the International System of Units.

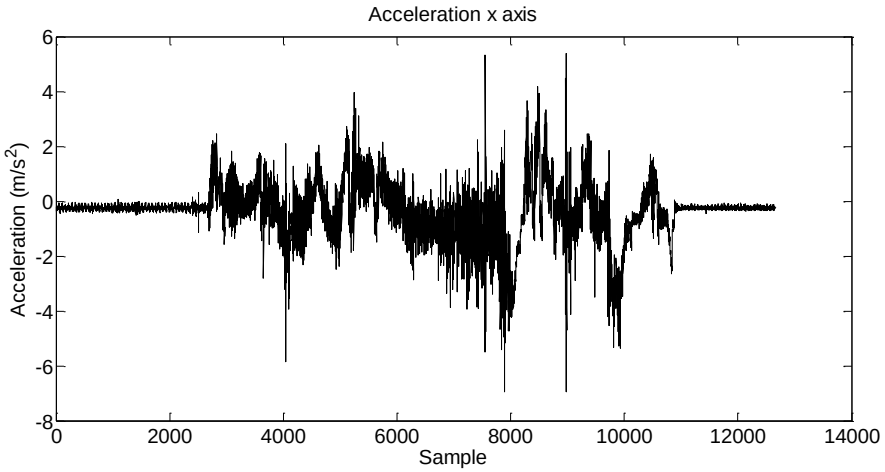

Figure 10 Measured acceleration along x axis

[4] See the datasheet available at http://www.acmesystems.it/DAISY-7.

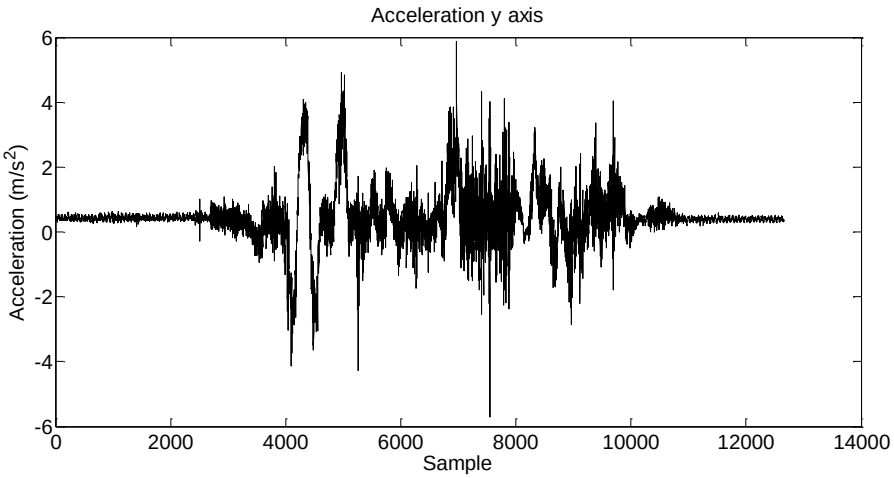

Figure 11 Measured acceleration along y axis

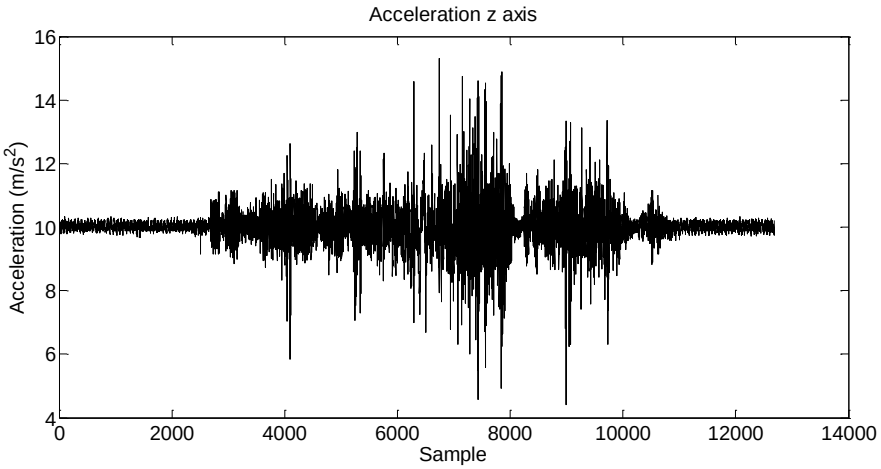

Figure 12 Measured acceleration along z axis

The estimated covariance values characterizing the additive noise are:

- Along the accelerometer x axis: 0.271.
- Along the accelerometer y axis: 0.137.
- Along the accelerometer z axis: 0.258.

7.3 Positioning Modules

Constraints

(the theoretical model is for a terrestrial ground vehicle, not applicable for 3D vehicles because the altitude is not part of the theoretical model)

7.3.1 Extended Kalman Filter- Code (& code description)

The following Section reports the MATLAB code of the function written to implement the EKF module.

1. The script "EKF.m", which implements the Extended Kalman Filter in case the pseudo measurement are not considered (see [1], pag. 50-51).
2. The script "EKF_pseudo_measurement", which implements the Extended Kalman Filter in case the pseudo measurement are considered (see [1], pag. 50-51).

In both cases, the EKF function has the following inputs and outputs:

Inputs:

1. The estimate of the state of the system at time step k (**xhatk**).
2. The covariance matrix associated to the estimate at k (**Phatk**).
3. The vector of measurement available at k (**zk**).

Outputs:

1. The updated estimate of the state of the system at time step k (**xhatkplus1**).
2. The updated covariance matrix associated to the estimate at k (**Phatkplus1**).

The MATLAB script "EKF.m" is reported in the following:

```
% EKF function
function [xhatkplus1, Phatkplus1]=EKF(xhatk,Phatk,zk)

   % Global Parameters, initialized in main.m file
   global g;                              % gravitational acceleration in navigation frame
   global Ts;                             % sampling time
   global Betaa;                          % three-dimentional vectors of AR(1) coefficients in (103)
   global Betaomega;                      % three-dimentional vectors of AR(1) coefficients in (104)
   global Cn;                             % diagonal covariance matrix of the vector nk in (121), see also (127) and
comments after (112)
   global Cm;                             % diagonal covariance matrix of vector mk in (124)

   % Components of the state vector
   Psik=xhatk(7:9)';            % Euler angles, see (52)
   ab=xhatk(10:12)';            % acceleration in body frame, see (52)
   omegabnb=xhatk(13:15)';      % angular velocity, see (52)

   % Euler angles
   phik=Psik(1);
   thetak=Psik(2);
   psik=Psik(3);

   % Acceleration components
   ab0=ab(1);
   ab1=ab(2);
   ab2=ab(3);

   % angular velocity components
   omegabnby=omegabnb(2);
```

```
    omegabnbz=omegabnb(3);

    % Computation of trigonometric functions
    sinphik=sin(phik);
    sinthetak=sin(thetak);
    sinpsik=sin(psik);
    cosphik=cos(phik);
    costhetak=cos(thetak);
    cospsik=cos(psik);
    tanthetak=tan(thetak);

    % Computation of roll-pitch-yaw rotation matrix Cnb(Psik), see (106)
    Cnb_Psik=[costhetak*cospsik                                    sinphik*sinthetak*cospsik-cosphik*sinpsik
cosphik*sinthetak*cospsik+sinphik*sinpsik;
            costhetak*sinpsik              sinphik*sinthetak*sinpsik+cosphik*cospsik          cosphik*sinthetak*sinpsik-
sinphik*cospsik;
            -sinthetak            sinphik*costhetak                  cosphik*costhetak];

    % Computation of matrix Asomega(Psik), see (100)
    Asomega_Psik=[1      sinphik*tanthetak     cosphik*tanthetak;
                  0      cosphik               -sinphik;
                  0      sinphik/costhetak     cosphik/costhetak];

    % Computation of matrix  A(Psik,ab), see (137)
    d11=[cosphik*sinthetak*cospsik+sinphik*sinpsik]*ab1+[-sinphik*sinthetak*cospsik+cosphik*sinpsik]*ab2;
    d12=[-sinthetak*cospsik]*ab0+[sinphik*costhetak*cospsik]*ab1+[cosphik*costhetak*cospsik]*ab2;
    d13=[-costhetak*sinpsik]*ab0+[-sinphik*sinthetak*sinpsik-cosphik*cospsik]*ab1+[-
cosphik*sinthetak*sinpsik+sinphik*cospsik]*ab2;
    d21=[cosphik*sinthetak*sinpsik-sinphik*cospsik]*ab1+[-sinphik*sinthetak*sinpsik-cosphik*cospsik]*ab2;
    d22=[-sinthetak*sinpsik]*ab0+[sinphik*costhetak*sinpsik]*ab1+[cosphik*costhetak*sinpsik]*ab2;
    d23=[costhetak*cospsik]*ab0+[sinphik*sinthetak*cospsik-
cosphik*sinpsik]*ab1+[cosphik*sinthetak*cospsik+sinphik*sinpsik]*ab2;
    d31=[cosphik*costhetak]*ab1+[-sinphik*costhetak]*ab2;
    d32=[-costhetak]*ab0+[-sinphik*sinthetak]*ab1+[-cosphik*sinthetak]*ab2;
    d33=0;
    A_Psik_ab=[d11 d12 d13;
               d21 d22 d23;
               d31 d32 d33];

    % Computation of B(Psik,omegabnb), see (147)
    App=[tanthetak*(omegabnby*cosphik-omegabnbz*sinphik)
(omegabnby*sinphik+omegabnbz*cosphik)/(costhetak^2)            0;
         -omegabnby*sinphik-omegabnbz*cosphik                  0                                                0;
        (omegabnby*cosphik+omegabnbz*sinphik)/costhetak                                                (omegabnby*sinphik-
omegabnbz*cosphik)*sinthetak/(costhetak^2)    0];
    B_Psik_omegabnb=eye(3)+Ts*App;

    % Computation of state transition function fx(xhatk), see (119) - (see also (118): xk+1=fx(xk)xk-Mg+nk)
    fx_xhatk=[eye(3)    Ts*eye(3)   zeros(3)   zeros(3)   zeros(3)        zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   Cnb_Psik   zeros(3)        zeros(3)     zeros(3);
              zeros(3)  zeros(3)    eye(3)     zeros(3)   Ts*Asomega_Psik  zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   eye(3)     zeros(3)        zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   zeros(3)   eye(3)          zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   zeros(3)   zeros(3)        diag(Betaa)  zeros(3);
```

```
        zeros(3)  zeros(3)   zeros(3)   zeros(3)   zeros(3)        zeros(3)     diag(Betaomega)]*xhatk;

    % Computation of the Jacobian of the transition function fx(xhatk), see (136)
    Jx=[eye(3)    Ts*eye(3)   zeros(3)        zeros(3)   zeros(3)        zeros(3)     zeros(3);
       zeros(3)  zeros(3)    A_Psik_ab        Cnb_Psik   zeros(3)        zeros(3)     zeros(3);
       zeros(3)  zeros(3)    B_Psik_omegabnb    zeros(3)   Ts*Asomega_Psik   zeros(3)     zeros(3);
       zeros(3)  zeros(3)    zeros(3)         eye(3)     zeros(3)        zeros(3)     zeros(3);
       zeros(3)  zeros(3)    zeros(3)         zeros(3)   eye(3)          zeros(3)     zeros(3);
       zeros(3)  zeros(3)    zeros(3)         zeros(3)   zeros(3)        diag(Betaa)  zeros(3);
       zeros(3)  zeros(3)    zeros(3)         zeros(3)   zeros(3)        zeros(3)     diag(Betaomega)];

    % Matrix appearing in the state transition equation, see (120)
    M=[zeros(3),eye(3),zeros(3),zeros(3),zeros(3),zeros(3),zeros(3)]';

    % EKF prediction step (132)-(133)
    xhatkplus1atk=fx_xhatk-M*g;
    Phatkplus1atk=Jx*Phatk*Jx'+Cn;

    % Computation of measurement model fz(xhatkplus1atk), see (125)
    fz_xhatkplus1atk=[xhatkplus1atk(10:12)+xhatkplus1atk(16:18);
                      xhatkplus1atk(13:15)+xhatkplus1atk(19:21)];

    % Computation of the Jacobian Jz(xhatkplus1atk) of fz(xhatkplus1atk), see (161)
    Jz_xhatkplus1atk=[zeros(3) zeros(3) zeros(3) eye(3)   zeros(3) eye(3)   zeros(3);
                      zeros(3) zeros(3) zeros(3) zeros(3) eye(3)   zeros(3) eye(3)];

    % EKF update step, see (158)-(163)
    skplus1atk=zk-fz_xhatkplus1atk;                          % innovation residual, eq. (158)
    Skplus1atk=Jz_xhatkplus1atk*Phatkplus1atk*Jz_xhatkplus1atk'+Cm;        % estimated covariance matrix of the
innovation residual, eq. (159)
    Kkplus1atk=Phatkplus1atk*Jz_xhatkplus1atk'/Skplus1atk;              % Kalman gain, eq. after (161)

    xhatkplus1=xhatkplus1atk+Kkplus1atk*skplus1atk;                    % new estimate, eq. (162)
    Phatkplus1=[eye(21)-Kkplus1atk*Jz_xhatkplus1atk]*Phatkplus1atk;            % covariance of the estimate, eq.
(163)
end
```

The MATLAB script “EKF_pseudo_measurement.m” is reported in the following:

```
% EKF function
function [xhatkplus1, Phatkplus1]=EKF_pseudo_measurement(xhatk,Phatk,zk)

% % Inputs
% xhatk       % Estimate of the state of the EKF at previous iteration
% Phatk       % Covariance matrix associated to xhatk
% zk          % Measurements vector
%
% %Outputs
% xhatkplus1    % Updated estaimate
% Phatkplus1    % Updated covariance matrix associated to the estimate
%
% All equation numbers below refer to [ref 1] = Giorgio M. Vitetta, In-Car
```

```
% Navigation Based on Integrated GPS/IMU Technologies (13.0_v1)

   % Global Parameters, initialized in main.m file
   global Ts;                        % EKF sampling time
   global Sigmaza;                   % covariance of the AGN affecting the acceleration measurement, see text after (123)
   global Sigmazomega;               % covariance of the AGN affecting the angular velocity measurement, see text after (123)
   global Cn;                        % diagonal covariance matrix of the vector nk in (121), see also (127) and comments after (112)
   global Cm;                        % diagonal covariance matrix of vector mk in (124)
   global g;                         % vector of acceleration gravity in navigation frame
   global Betaa;                     % three-dimentional vectors of AR(1) coefficients in (103)
   global Betaomega;                 % three-dimentional vectors of AR(1) coefficients in (104)

   % Overwrites previous definitions in main script - TO BE PROPERLY SELECTED
   Cm=blkdiag(Sigmaza,Sigmazomega,[0.01 0;0 0.01]);

   % Components of the state vector *** xhatk ***
   Psik=xhatk(7:9)';            % Euler angles, see (52)
   ab=xhatk(10:12)';            % acceleration in body frame, see (52)
   omegabnb=xhatk(13:15)';      % angular velocity, see (52)

   % Euler angles
   phik=Psik(1);
   thetak=Psik(2);
   psik=Psik(3);

   % Acceleration components
   ab0=ab(1);
   ab1=ab(2);
   ab2=ab(3);

   % angular velocity components
   omegabnby=omegabnb(2);
   omegabnbz=omegabnb(3);

   % Computation of trigonometric functions
   sinphik=sin(phik);
   sinthetak=sin(thetak);
   sinpsik=sin(psik);
   cosphik=cos(phik);
   costhetak=cos(thetak);
   cospsik=cos(psik);
   tanthetak=tan(thetak);

   % Computation of roll-pitch-yaw rotation matrix Cnb(Psik), see (106)
   Cnb_Psik=[costhetak*cospsik                                    sinphik*sinthetak*cospsik-cosphik*sinpsik  cosphik*sinthetak*cospsik+sinphik*sinpsik;
             costhetak*sinpsik          sinphik*sinthetak*sinpsik+cosphik*cospsik     cosphik*sinthetak*sinpsik-sinphik*cospsik;
             -sinthetak                 sinphik*costhetak                             cosphik*costhetak];

   % Computation of matrix Asomega(Psik), see (100)
```

```
    Asomega_Psik=[1      sinphik*tanthetak     cosphik*tanthetak;
                  0      cosphik               -sinphik;
                  0      sinphik/costhetak     cosphik/costhetak];

    % Computation of matrix  A(Psik,ab), see (137)
    d11=[cosphik*sinthetak*cospsik+sinphik*sinpsik]*ab1+[-sinphik*sinthetak*cospsik+cosphik*sinpsik]*ab2;
    d12=[-sinthetak*cospsik]*ab0+[sinphik*costhetak*cospsik]*ab1+[cosphik*costhetak*cospsik]*ab2;
    d13=[-costhetak*sinpsik]*ab0+[-sinphik*sinthetak*sinpsik-cosphik*cospsik]*ab1+[-
cosphik*sinthetak*sinpsik+sinphik*cospsik]*ab2;
    d21=[cosphik*sinthetak*sinpsik-sinphik*cospsik]*ab1+[-sinphik*sinthetak*sinpsik-cosphik*cospsik]*ab2;
    d22=[-sinthetak*sinpsik]*ab0+[sinphik*costhetak*sinpsik]*ab1+[cosphik*costhetak*sinpsik]*ab2;
    d23=[costhetak*cospsik]*ab0+[sinphik*sinthetak*cospsik-
cosphik*sinpsik]*ab1+[cosphik*sinthetak*cospsik+sinphik*sinpsik]*ab2;
    d31=[cosphik*costhetak]*ab1+[-sinphik*costhetak]*ab2;
    d32=[-costhetak]*ab0+[-sinphik*sinthetak]*ab1+[-cosphik*sinthetak]*ab2;
    d33=0;
    A_Psik_ab=[d11 d12 d13;
               d21 d22 d23;
               d31 d32 d33];

    % Computation of B(Psik,omegabnb), see (147)
    App=[tanthetak*(omegabnby*cosphik-omegabnbz*sinphik)
(omegabnby*sinphik+omegabnbz*cosphik)/(costhetak^2)            0;
        -omegabnby*sinphik-omegabnbz*cosphik                   0                                                    0;
       (omegabnby*cosphik+omegabnbz*sinphik)/costhetak                                                             (omegabnby*sinphik-
omegabnbz*cosphik)*sinthetak/(costhetak^2)    0];
    B_Psik_omegabnb=eye(3)+Ts*App;

    % Computation of state transition function fx(xhatk), see (119) - (see also (118): xk+1=fx(xk)xk-Mg+nk)
    fx_xhatk=[eye(3)    Ts*eye(3)   zeros(3)   zeros(3)   zeros(3)         zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   Cnb_Psik   zeros(3)         zeros(3)     zeros(3);
              zeros(3)  zeros(3)    eye(3)     zeros(3)   Ts*Asomega_Psik  zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   eye(3)     zeros(3)         zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   zeros(3)   eye(3)           zeros(3)     zeros(3);
              zeros(3)  zeros(3)    zeros(3)   zeros(3)   zeros(3)         diag(Betaa)  zeros(3);
              zeros(3)  zeros(3)    zeros(3)   zeros(3)   zeros(3)         zeros(3)     diag(Betaomega)]*xhatk;

    % Computation of Jacobian matrix, see (136)
    Jx=[eye(3)    Ts*eye(3)   zeros(3)          zeros(3)   zeros(3)         zeros(3)     zeros(3);
        zeros(3)  zeros(3)    A_Psik_ab         Cnb_Psik   zeros(3)         zeros(3)     zeros(3);
        zeros(3)  zeros(3)    B_Psik_omegabnb   zeros(3)   Ts*Asomega_Psik  zeros(3)     zeros(3);
        zeros(3)  zeros(3)    zeros(3)          eye(3)     zeros(3)         zeros(3)     zeros(3);
        zeros(3)  zeros(3)    zeros(3)          zeros(3)   eye(3)           zeros(3)     zeros(3);
        zeros(3)  zeros(3)    zeros(3)          zeros(3)   zeros(3)         diag(Betaa)  zeros(3);
        zeros(3)  zeros(3)    zeros(3)          zeros(3)   zeros(3)         zeros(3)     diag(Betaomega)];

    % Matrix appearing in the state transition equation, see (120)
    M=[zeros(3),eye(3),zeros(3),zeros(3),zeros(3),zeros(3),zeros(3)]';

    % EKF prediction step (132)-(133)
    xhatkplus1atk=fx_xhatk-M*g;
    Phatkplus1atk=Jx*Phatk*Jx'+Cn;

    % Computation of measurement model fz(xhatkplus1atk), see (125), (175)
```

```
Psihatkplus1atk=xhatkplus1atk(7:9)';          % Euler angles in vector *** xhatkplus1atk ***, see (52)
% Euler angles
phihatkplus1atk=Psihatkplus1atk(1);
thetahatkplus1atk=Psihatkplus1atk(2);
psihatkplus1atk=Psihatkplus1atk(3);
% Computation of trigonometric functions related to Euler angles in
% vector *** xhatkplus1atk ***THIS OVERWRITES THE PREVIOUS COMPUTATION
% OF TRIGONOMETRIC FUNCTIONS
sinphik=sin(phihatkplus1atk);
sinthetak=sin(thetahatkplus1atk);
sinpsik=sin(psihatkplus1atk);
cosphik=cos(phihatkplus1atk);
costhetak=cos(thetahatkplus1atk);
cospsik=cos(psihatkplus1atk);
% Computation of roll-pitch-yaw rotation matrix Cnb(Psik), with Euler angles in vector *** xhatkplus1atk ***,
see (106)
Cnb_Psik=[costhetak*cospsik                                    sinphik*sinthetak*cospsik-cosphik*sinpsik
cosphik*sinthetak*cospsik+sinphik*sinpsik;
        costhetak*sinpsik          sinphik*sinthetak*sinpsik+cosphik*cospsik          cosphik*sinthetak*sinpsik-
sinphik*cospsik;
        -sinthetak          sinphik*costhetak                  cosphik*costhetak];
vbkatkplus1atk=Cnb_Psik'*xhatkplus1atk(1:3);                % Last two components of velocity in body
frame, see (171)
fz_xhatkplus1atk=[xhatkplus1atk(10:12)+xhatkplus1atk(16:18);      % Measurement model, see (176)
            xhatkplus1atk(13:15)+xhatkplus1atk(19:21);
            vbkatkplus1atk(2:3)];

% Computation of matrix Mv, see (179)
Mv=[sinphik*sinthetak*cospsik-cosphik*sinpsik                  sinphik*sinthetak*sinpsik+cosphik*cospsik
sinphik*costhetak;
    cosphik*sinthetak*cospsik+sinphik*sinpsik                  cosphik*sinthetak*sinpsik-sinphik*cospsik
cosphik*costhetak];

% Computation of matrix MPsi, see (180)-(186)
MPsi11=xhatkplus1atk(1)*(cosphik*sinthetak*cospsik          +          sinphik*sinpsik)                +
xhatkplus1atk(2)*(cosphik*sinthetak*sinpsik - sinphik*cospsik)   +   xhatkplus1atk(3)*(cosphik*costhetak);
MPsi12=xhatkplus1atk(1)*sinphik*costhetak*cospsik      +      xhatkplus1atk(2)*sinphik*costhetak*sinpsik     -
xhatkplus1atk(3)*sinphik*sinthetak;
MPsi13=xhatkplus1atk(1)*(-sinphik*sinthetak*sinpsik          -          cosphik*cospsik)                +
xhatkplus1atk(2)*(sinphik*sinthetak*cospsik - cosphik*sinpsik);
MPsi21=xhatkplus1atk(1)*(-sinphik*sinthetak*cospsik   +   cosphik*sinpsik)        +        xhatkplus1atk(2)*(-
sinphik*sinthetak*sinpsik - cosphik*cospsik)   +   xhatkplus1atk(3)*(-sinphik*costhetak);
MPsi22=xhatkplus1atk(1)*(cosphik*costhetak*cospsik)   +   xhatkplus1atk(2)*(cosphik*costhetak*sinpsik)   +
xhatkplus1atk(3)*(-cosphik*sinthetak);
MPsi23=xhatkplus1atk(1)*(-cosphik*sinthetak*sinpsik          +          sinphik*cospsik)                +
xhatkplus1atk(2)*(cosphik*sinthetak*cospsik + sinphik*sinpsik);
MPsi=[MPsi11 MPsi12 MPsi13;
      MPsi21 MPsi22 MPsi23];

% Computation of the Jacobian Jz(xhatkplus1atk) of fz(xhatkplus1atk), see (161), (175)
Jz_xhatkplus1atk=[zeros(3)  zeros(3)   zeros(3)   eye(3)    zeros(3)   eye(3)    zeros(3);
            zeros(3)  zeros(3)   zeros(3)   zeros(3)   eye(3)    zeros(3)   eye(3);
            Mv        zeros(2,3)  MPsi       zeros(2,3)  zeros(2,3)  zeros(2,3)  zeros(2,3)];

% EKF update step, see (158)-(163)
```

```
    skplus1atk=[zk ;[0;0]]-fz_xhatkplus1atk;                    % innovation residual, eq. (158), (187)
    Skplus1atk=Jz_xhatkplus1atk*Phatkplus1atk*Jz_xhatkplus1atk'+Cm;                    % innovation estimated coviarance matrix, eq. (159)
    Kkplus1atk=Phatkplus1atk*Jz_xhatkplus1atk'/Skplus1atk;                    % Kalman gain, eq. after (161)

    xhatkplus1=xhatkplus1atk+Kkplus1atk*skplus1atk;                    % new estimate, eq. (162)
    Phatkplus1=[eye(21)-Kkplus1atk*Jz_xhatkplus1atk]*Phatkplus1atk;                    % covariance of the estimate, eq. (163)
end
```

7.3.2 Particle Filter- Code (& code description)

The following Section reports the MATLAB code implementing the PF module, as described in Section 6.2.5 of [1].

The PF function has the following inputs and outputs:

Inputs:

3. The estimate state vector at the previous iteration (**rkm1**).
- The estimate state variation (**deltark**).
- The covariace matrix of the estimate state variation (**Cdelta_r**).
- Covariace Matrix (**C_n**).
- The number of particles (**Np**).
- The GPS meauserement (zeros if not available) (**z_GPS**).
- The GPS covariance matrix (**C_GPS**).
- The particles weights at the previous iteration (**wkm1**).
- The particles at the previous iteration (**r_Pm1**).
- The number of the current PF iteration (**PF_iteration**).
- The x coordinates of the map constraint polygon (**xv**) [optional].
- The y coordinates of the map constraint polygon (**yv**) [optional].

Outputs:

- The current state estimation (**rk**).
- The current set of particles (**r_P**).
- The current set of particles weights (**wk**).

The MATLAB function “PF.m” is reported in the following.

```
%% PARTICLE FILTER function
function [rk r_P wk] = PF( rkm1,deltark,Cdelta_r,C_n,Np,z_GPS,C_GPS,wkm1,r_Pm1,PF_iteration,xv,yv)
% %Input
% rkm1       %State vector at the previous iteration
% deltark    %estimation of state variation
% Cdelta_r   %Covariace Matrix of deltark
% C_n        %Covariace Matrix
```

```
% Np         %Number of Particles
% z_GPS      %GPS Meauserement (if available)
% C_GPS      %GPS Covariance matrix
% wkm1       %particles weights at the previous iteration
% r_Pm1      %particles at the previous iteration
% PF_iteration  %current PF iteration (PF_iteration == 1 => initialization)
% xv         %[optional] x coordinates of map constraint polygon
% yv         %[optional] y coordinates of map constraint polygon
%
% %Output
% rk         %state estimation
% r_P        %current particles
% wk         %current weights
%%
% [ref 1] = Giorgio M. Vitetta, In-Car Navigation Based on Integrated
% GPS/IMU Technologies (13.0_v1)

% if a map constraint isn't defined, we set the map constraint as a wide area
if isempty(xv) || isempty(yv)
    xv = [-10^8;10^8;10^8;-10^8];
    yv = [-10^8;-10^8;10^8;10^8];
end



nr = size(rkm1,1);   % state vector dimensionality
r_P = r_Pm1;         % define the vector of particles and we initialize them as the particles obteined in the previous
iteration
Np_tilde =30;       % Number of particles to estimate the weigth of each particle  (defined in pag. 54 of [ref 1])



if PF_iteration == 1   % Particles Initialization
       for i = 1:Np
          r_P(i,1:2) = mvnrnd(z_GPS,C_GPS);  % Initialization using GPS information (see pag 56 [ref 1])
          r_P(i,3) = rkm1(3);
          wkm1(i) = 1/Np;                % weights initialization (see pag 56 of [ref 1])
       end
end

%generate the observations from the randomly selected particles
r_est = rkm1; % initialize current state estimation

%Here, we do the particle filter
for i = 1:Np
    r_P_update(i,:) = mvnrnd(r_P(i,:)' + deltark,C_n);
    %with these new updated particle locations, update the observations
    %for each of these particles.
    %Generate the weights for each of these particles.

    r_tilde=zeros(nr,Np_tilde);   % sub-particles vector initialization
    cumulative_weight = 0;

   for s=1:Np_tilde
        y(:,s)= mvnrnd(zeros(nr,1),Cdelta_r);   % generate sub-particles (see pag 55 in [ref 1])
        r_tilde(:,s) = y(:,s) + deltark;
```

```
      if inpolygon(r_tilde(1,s),r_tilde(2,s),xv,yv)  %verify if the map constraint is respected or not
         weight = mvnpdf(r_tilde(:,s),zeros(nr,1),C_n);    %see equation (204) in [ref 1])
      else
         weight = 0;                  %see equation (203) in [ref 1])
      end
      cumulative_weight = cumulative_weight + weight;
   end

   if isempty(z_GPS)       %updated weights if no GPS meausures are available
     wk(i) =  wkm1(i)*(1/Np_tilde)*cumulative_weight;    %see equation (202) in [ref 1])
   else                  %updated weights considering GPS meausures
     wkG(i)=  mvnpdf(r_P_update(i,1:2),z_GPS,C_GPS);     %weight contribution due to GPS measures     %see
equation (209) in [ref 1])
     if wkG(i)== 0 %invalid GPS measurement
        wkG(i)=1;
     end
     wk(i) = wkm1(i)*(1/Np_tilde)*cumulative_weight*wkG(i);    %see equation (210) in [ref 1])
   end

end

    % Normalize to form a probability distribution (i.e. sum to 1).
    wk = wk./sum(wk);   %see equation (205) and (206) in [ref 1])

    % Resampling: From this new distribution, now we randomly sample from it to generate our new estimate
particles
   Neff = 1/sum(wkm1.^2);       % Effective sample size -> used to trigger the resapling strategy [formula (207) of
ref1]
   Nt = Np *0.5;              % resampling threshold (defined in pag.55 of [ref 1])

   if Neff < Nt || sum(wkm1.^2)==0   % in this case a new set of particles have to be choosen (and consequently their
weights) (see pag 56 of [ref 1])
     for i = 1 : Np
        pos = find(rand <= cumsum(wk),1);
        if isempty(pos)
           pos = ceil(1 + (Np-1).*rand);
        end
        r_P(i,:) = r_P_update(pos,:);
        wk(i) = 1/Np;             % the particle weights are setted (see pag 56 of [ref 1])
     end
     disp('Resampling...');
   else
     disp('Not Resampling...');
     r_P = r_P_update;
   end



    %The final estimate is some metric of these final resampling, uncomment
    %only one of the three different strategies

%    %%%select particle with maximum probability
%     pos_max = find(max(wk)==wk);
%     r_est = r_P(pos_max,:)';
%
```

```
%    %%simple average
%     r_est = mean(r_P,1)';

   %%weighted average
    r_est = [wk*r_P/sum(wk)]';

    rk = r_est;  %return estimation

return;
```

7.4 Integration

The EKF and PF functions described above constitute the core modules of the architecture shown in Figure 2. The developed library contains these key functions as well as other functions necessary for the proper functioning of the first ones. This section reports and explains the script "main.m" written in order to orchestrate the execution of all these functions.

As shown in Figure 13, the script "main.m" is in charge of configuring the simulation parameters. First of all, "main.m" runs the script "estimate_covariance.m" which returns the covariance estimation obtained from the real data stored in the structure "integratedData.mat". After that, the selected simulation scenario is activated simply uncommenting it, and the corresponding function is called. For example by selecting the scenario *scenario_circular_turn* the function "scenario_circular_turn.m" is executed: this function returns the simulated sensors measurements (gyroscope and accelerometer) representing the selected scenario (such simulated sensors measurements are given by nominal profiles with superimposed a noise characterized by the covariance estimated by "estimate_covariance.m").

Later in "main.m" all the other parameters (the initial estimate position, the number of particles, the frequency to call the PF, etc…) are initialized.

After this initial configuration phase, the core modules can be properly executed. In fact, for each IMU sample the "EKF.m" function is called in order to evaluate the vehicle status estimation. The code below the EKF call represents the pdf estimator that is in charge of processing the "EKF.m" outputs and calculating the vehicle linear/angular displacement and the corresponding covariance values.

These data are taken as inputs each *PF_freq* IMU samples by the Particle Filter, which is executed calling the function "PF.m". The Particle Filter is assumed to run *PF_freq* times slower than the EKF, where *PF_freq* is a parameter properly initialized in "main.m". "PF.m" uses the PDF estimations to calculate the vehicle position and yaw estimations. In addition, when the GPS measurements and/or the Map constraints are available, the PF merges this information with the PDF estimations in order to improve the quality of its estimation.

The last lines of code in "main.m" are used to visualize the results by tracing the vehicle trajectory.

In addition, if the *EKF_restart_freq* is set in the "main.m" with an integer value, each time that the EKF iteration is a multiple of *EKF_restart_freq*, the EKF covariance matrix is re-initialized to avoid divergence from the nominal trajectory. This functionality can be easily removed by setting the *EKF_restart_freq* with a non-integer number.

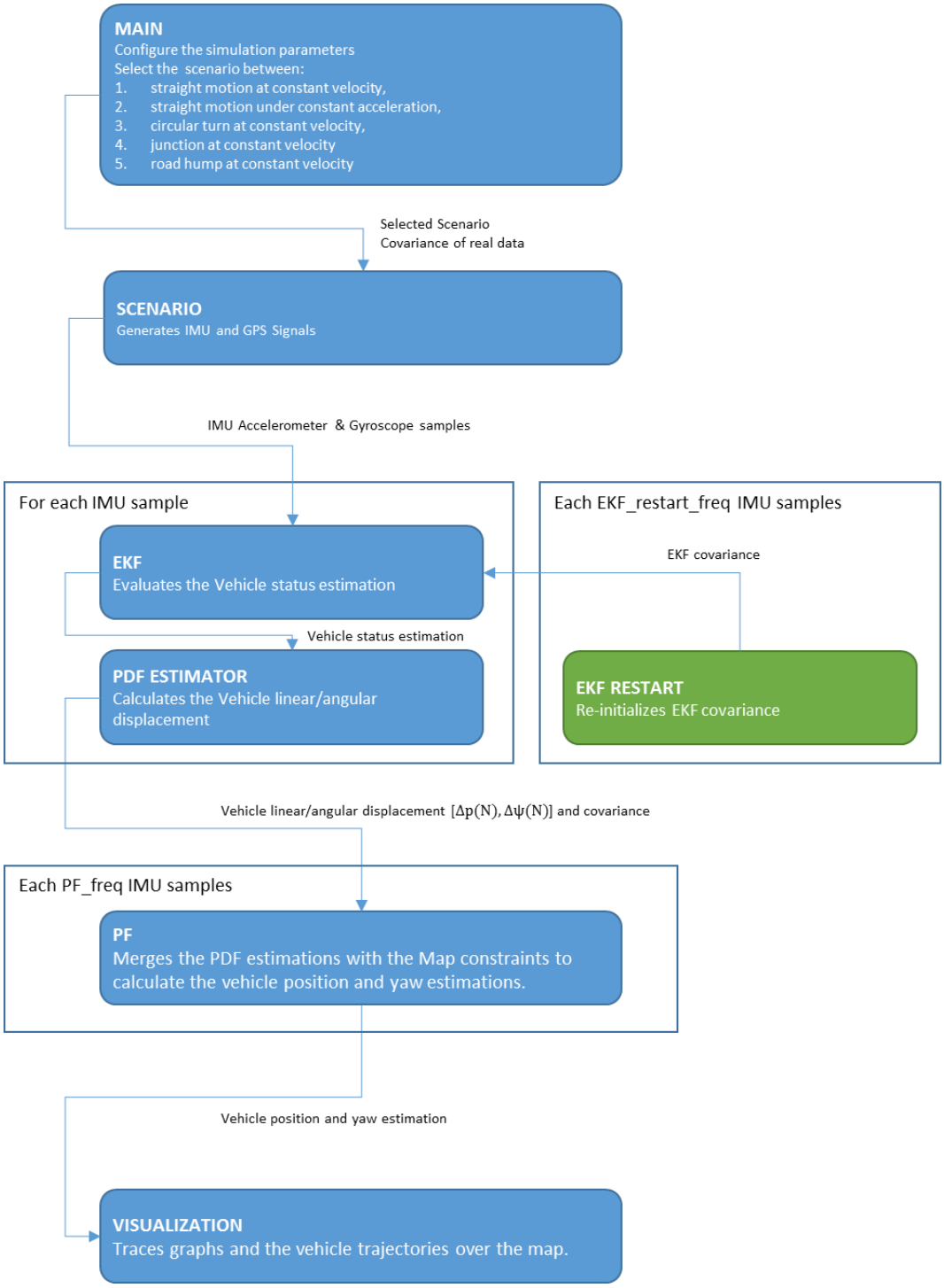

Figure 13. MATLAB integration workflow

The MATLAB script “main.m” is reported in the following:

```
%% main file

% All equation numbers below refer to [ref 1] = Giorgio M. Vitetta, In-Car Navigation Based on Integrated
GPS/IMU Technologies (13.0_v1)

%% Clear workspace, close all, clear command line
clear all
close all
clc

%% Parameters declaration
global Ts;                          % EKF sampling time
global N;                           % number of simulation steps
global Sigmaza;                     % covariance of the AGN affecting the acceleration measurement, see text after
(123)
global Sigmazomega;                 % covariance of the AGN affecting the angular velocity measurement, see text
after (123)
global Cn;                          % diagonal covariance matrix of the vector nk in (121), see also (127) and comments
after (112)
global Cm;                          % diagonal covariance matrix of vector mk in (124)
global g; g=-[0;0;9.8];             % vector of acceleration gravity in navigation frame (assigned here since needed
in estimate_covariance)
global Betaa;                       % three-dimentional vectors of AR(1) coefficients in (103)
global Betaomega;                   % three-dimentional vectors of AR(1) coefficients in (104)
global C_GPS;                       % covariance of the noise affecting GPS measures
global x0;                          % initial position of the vehicle
global intial_yaw;                  % initial yaw of the vehicle
global nominal_trajectory;          % nominal trajectory of the vehicle built in the script which generates the
scenario

%% ESTIMATION OF THE COVARIANCE OF THE NOISE AFFECTING ACCELEROMETER,
GYROSCOPE, GPS MEASUREMENTS
estimate_covariance

%% PARAMETERS ASSIGNMENT - PARAMETERS TO BE PROPERLY SELECTED/TUNED
T=45;                               % simulation time window [seconds]
Ts=0.01;
N=round(T/Ts);
Sigmaza=[estimated_variance_add_noise_acc_x 0 0;0 estimated_variance_add_noise_acc_y 0;0 0
estimated_variance_add_noise_acc_z];
Sigmazomega=[estimated_variance_add_noise_gyro_x 0 0;0 estimated_variance_add_noise_gyro_y 0;0 0
estimated_variance_add_noise_gyro_z];
%     Cn=0.06*eye(21);
Cn=blkdiag(Ts^2*Sigmaza*10,Sigmaza,Ts^2*Sigmazomega*10,Sigmaza,Sigmazomega,0.01*eye(3),0.01*eye(3));
Cm=blkdiag(Sigmaza,Sigmazomega);
Betaa=[0.9999; 0.9999; 0.9999];
Betaomega=[0.9999; 0.9999; 0.9999];
C_GPS = [3 0;0 3];                  %GPS covariance matrix

%% SELECTION OF THE SCENARIO TO BE SIMULATED (uncomment the scenario to be simulated). Creates
artificial test signals and save data in the "integratedData" structure
%       scenario_vehicle_still
```

```
%       scenario_constant_velocity_x
%       scenario_constant_acceleration_x
        scenario_circular_turn
%       scenario_constant_velocity_x_hump

%% INITIALIZATIONS

% Initialization of the EKF and of the pdf Estimator module
estimated_position=[];          % matrix 2Xtime storing the final position estimates (x and y) from the PF
estimated_orientation=[];       % vector storing the final orientation estimates from the PF
DeltapN=[0;0];                  % change in the vehicle position in the motion plane occurred from the last call of the PF
module, see (53)
CDeltapN=[0 0;0 0];             % estimate of DeltapN covariance, see (60)
Cptilde=[0.01 0;0 0.01];        % covariance of the AGN vector describing model inaccuracies in (60)-(62) - TO BE
PROPERLY SELECTED/TUNED
DeltapsiN=0;                    % change in the vehicle orientation occurred from the last call of the PF module, see (54)
sigmasquaredDeltapsiN=0;        % estimate of DeltapN covariance, see (74)
intial_yaw=0;

% Initialization Particle Filter
Cdelta_n = [0 0;0 0];           % noise covariance in the system (i.e. process noise in the state update, here, we'll use a
gaussian.)
C_n = [0 0;0 0];                % noise covariance in the measurement
Np = 100;                       % the number of particles the system generates. The larger this is, the better your
approximation, but the more computation you need.
rkm1 = [x0(1:2);intial_yaw];    % initial actual state
wkm1 = [];                      % initial particle weights
r_Pm1=[];                       % initial particles
PF_freq = 25;                   % frequency of PF calling (PF is called each 'PF_freq' EKF iteration)
PF_iteration = 0;               % number of current PF iterations
EFK_restart_freq = 250;         % frequency of EKF restarting (EKF is restarted each 'EFK_restart_freq' EKF
iteration)

%% Initial state of the EKF filter (parameters v0, a0, Psi0, ab0, omegab0, ba0, bomega0 defined in the scenario_
scripts)
xhatkplus1{1}=[v0; a0; Psi0; ab0; omegab0; ba0; bomega0];
Phatkplus1{1}=zeros(21);            % TO BE PROPERLY SELECTED/TUNED
Phatkplus1{1}=blkdiag(Ts^2*Sigmaza/100,Sigmaza/10,Ts^2*Sigmazomega/100,Sigmaza,Sigmazomega,0.01*eye(
3),0.01*eye(3));
traj_nom = rkm1;
error = 0;
error_x = 0;
error_y = 0;
traj_EKF=zeros(2,N);
traj_EKF(:,1) = x0(1:2);

GPS_plot = [];
xv = [];
yv = [];

%%Map constraint
%  xv = [0;130;130;0];
%  yv = [-1.5;-1.5;1.5;1.5];  %%bounds map constraints
```

```
%% Scan inputs from structure "integratedData" containing the artificialtest signals
for i=1:length(integratedData.timestamp)

    % ith measured acceleration and angular velocity
    zi=[integratedData.ACC_Xaxis(i);  integratedData.ACC_Yaxis(i);  integratedData.ACC_Zaxis(i);
integratedData.GYR_Xaxis(i);  integratedData.GYR_Yaxis(i);  integratedData.GYR_Zaxis(i)];

    % Launch simple EKF or EKF with pseudo measurements (uncomment one of the two lines accordingly)
    [xhatkplus1{i+1}, Phatkplus1{i+1}]=EKF(xhatkplus1{i},Phatkplus1{i},zi);
%     [xhatkplus1{i+1}, Phatkplus1{i+1}]=EKF_pseudo_measurement(xhatkplus1{i},Phatkplus1{i},zi);     % with
pseudo measurements

    % Computation of joint statistical information inputs to PF (change in position and orientation occurred from the
last call of the PF)
    DeltapN = DeltapN + xhatkplus1{i+1}(1:2)*Ts + 1/2*xhatkplus1{i+1}(4:5)*Ts^2;
% see (58), (59), (62) and (63)
    CDeltapN = CDeltapN + Ts^2*Phatkplus1{i+1}(1:2,1:2) + 1/4*Ts^4*Phatkplus1{i+1}(4:5,4:5) +
Ts^3*Phatkplus1{i+1}(1:2,4:5) + Cptilde;          % see (60), (61) and (66)
    DeltapsiN = DeltapsiN+(xhatkplus1{i+1}(9)-xhatkplus1{i}(9));
% see (70) and (72)
    phis=sin(xhatkplus1{i+1}(7))*[1-1/2*Phatkplus1{i+1}(7,7)];
% see (80)
    phic=cos(xhatkplus1{i+1}(7))*[1-1/2*Phatkplus1{i+1}(7,7)];
% see (81)
    phissquared=sin(xhatkplus1{i+1}(7))^2 + (cos(xhatkplus1{i+1}(7))^2 -
sin(xhatkplus1{i+1}(7))^2)*Phatkplus1{i+1}(7,7);                    % see (83)
    phicsquared=cos(xhatkplus1{i+1}(7))^2 + (sin(xhatkplus1{i+1}(7))^2 -
cos(xhatkplus1{i+1}(7))^2)*Phatkplus1{i+1}(7,7);                    % see (84)
    phisc=1/2*sin(2*xhatkplus1{i+1}(7))*[1-2*Phatkplus1{i+1}(7,7)];
% see (85)
    thetacmone    =[1 + (1/2 +   tan(xhatkplus1{i+1}(8))^2)*Phatkplus1{i+1}(8,8)]/cos(xhatkplus1{i+1}(8));
% see (86)
    thetacmsquared=[1 + (1   + 3*tan(xhatkplus1{i+1}(8))^2)*Phatkplus1{i+1}(8,8)]/cos(xhatkplus1{i+1}(8))^2;
% see (87)
    sigmasquaredX=[(xhatkplus1{i+1}(14)^2+Phatkplus1{i+1}(14,14))*phissquared*thetacmsquared-
(xhatkplus1{i+1}(14)*phis*thetacmone)^2];          % see (88)
    sigmasquaredY=[(xhatkplus1{i+1}(15)^2+Phatkplus1{i+1}(15,15))*phicsquared*thetacmsquared-
(xhatkplus1{i+1}(15)*phic*thetacmone)^2];          % see (89)
    covXlYl=xhatkplus1{i+1}(14)*xhatkplus1{i+1}(15)*[phisc*thetacmsquared-phis*phic*(thetacmone^2)];
% see (90)
    sigmasquaredDeltapsiN = sigmasquaredDeltapsiN + [sigmasquaredX+sigmasquaredY+2*covXlYl]*Ts^2;
% see (74)

    if i~=1
        traj_EKF(:,i)= traj_EKF(:,i-1) + xhatkplus1{i+1}(1:2)*Ts + 1/2*xhatkplus1{i+1}(4:5)*Ts^2;
    end
    % Launch PF with frequency PF_freq
    if  mod(i,PF_freq)== 0

        string_iter = sprintf('Iteration %d of %d',i,T*100);
        disp(string_iter)

        % Estimate position and orientation in navigation frame
        deltark=[DeltapN;DeltapsiN];
```

```
        Cdelta_r=blkdiag( (CDeltapN+CDeltapN')/2 , sigmasquaredDeltapsiN)

    %  Cdelta_r = [0.6 0;0 0.6];
        C_n = Cdelta_r;   %we assume C_n equal to Cdelta_r
        C_n(3,3) =0.001;
        if mod(i,25)== 0
            z_GPS=[integratedData.Longitude(i),integratedData.Latitude(i)];
        else
            z_GPS = [];
        end

        PF_iteration = PF_iteration + 1;
        [rk, r_P, P_w] = PF( rkm1,deltark,Cdelta_r,C_n,Np,z_GPS,C_GPS,wkm1,r_Pm1,PF_iteration,xv,yv);

        if mod(i,EFK_restart_freq) == 0
            Phatkplus1{i+1} = zeros(21);
        end

        r_Pm1 = r_P;
        wkm1 = P_w;
        rkm1 = rk;

        % Write result (absolute position, orientation and covariance on the position)
        estimated_position=[estimated_position [rk(1:2)]];
        estimated_orientation=[estimated_orientation rk(3)];

        % Reset differential information
        DeltapN=[0;0];
        CDeltapN=[0 0;0 0];
        DeltapsiN=0;
        sigmasquaredDeltapsiN=0;

        error_x = [error_x ; (rk(1) - nominal_trajectory(1,i))];
        error_y = [error_y ; (rk(2) - nominal_trajectory(2,i))];
      %  error = [error ; norm(rk - nominal_trajectory(1:2,i))];

        GPS_plot = [GPS_plot; z_GPS];
    end
end

% Plot the position, attitude estimates
hold on;
plot(nominal_trajectory(1,:), nominal_trajectory(2,:),'-r');
hold on;
plot(GPS_plot(:,1), GPS_plot(:,2),'-g');
hold on;
plot(traj_EKF(1,:), traj_EKF(2,:),'-b');
hold on;
plot(estimated_position(1,:), estimated_position(2,:),'-k','LineWidth',3);
scale = 0.1;
quiver(estimated_position(1,1:3:end),estimated_position(2,1:3:end),cos(estimated_orientation(1:3:end)),sin(estimate
d_orientation(1:3:end)),scale)
hold on;
title(['Estimated Position on xy plane'],'FontSize', 25)
```

```
xlabel('x', 'FontSize', 25)
ylabel('y', 'FontSize', 25)
hold on;
if ~isempty(xv) && ~isempty(yv)
plot(xv,yv,'LineWidth',2,...
        'LineStyle','--');
end
set(gca,'fontsize',25)
h_legend = legend('Nominal Trajectory', 'GPS Measures', 'EKF Trajectory','Position estimation');
set(h_legend,'FontSize',15)
xlim([0 250]);
ylim([-125 125]);

% Plot the estimation error
figure;
title([' Estimation error'],'FontSize', 25);
xlabel('PF iterations', 'FontSize', 25)
ylabel('meters', 'FontSize', 25)
hold on;
plot(abs(error_x),'b');
plot(abs(error_y),'r');
%plot(error,'g');
%legend('Error in X', 'Error in Y', 'Total Error');
h_legend = legend('Error in X', 'Error in Y');
set(h_legend,'FontSize',15)
set(gca,'fontsize',25)
```

7.5 Test cases

The following Sections report the description of the simulation scenarios considered in this report to test the implementation of the integrated EKF-PF navigation solution.

In particular, the simulation scenarios allow building the synthetic accelerometer, gyroscope and GPS measurement profiles used to test the solution. Such artificial measurement profiles are built starting from the nominal profiles corresponding to the scenario in question (e.g. vehicle moving at constant velocity along x axis) and then adding a Gaussian noise with a covariance approximating the one characterizing actual accelerometer, gyroscope and GPS samples.

7.5.1 Scenario 1: Straight Motion at Constant Velocity – Description

In this scenario, the vehicle proceeds in straight motion (e.g. along the x-axis) at constant velocity. It is useful to consider this simple scenario in order to evaluate the impact of disturbances and noise in (1) and (2) on EKF performances.

In this scenario, $\boldsymbol{a_b}$ is given by[5]

[5] We assume the accelerometer measures the inertial acceleration plus the gravity vector **g** (for this reason the gravity acceleration g appears in (1), which refers to motion at constant velocity).

$$\boldsymbol{a_b^1} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

and $\boldsymbol{\omega_b}$ by

$$\boldsymbol{\omega_b^1} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

7.5.1.1 Scenario 1: Straight Motion at Constant Velocity – MATLAB Code

Below we report the MATLAB script "scenario_constant_velocity_x.m", which computes the nominal trajectory on the plane and the nominal orientation of the vehicle, along with the corresponding artificial acceleration, angular velocity and GPS measurement signals.

```
% Scenario: vehicle proceeding at constant velocity along x axis

%% Parameters - see main.m for explanations
v_x=10;                    % linear velocity along x axis
global N;
global x0;
global Sigmaza;
global Sigmazomega;
global Ts;
global nominal_orientation;
global nominal_trajectory;
global g;
global C_GPS;

%% Construction of an artificial data set from accelerometer and gyroscope (notice the noisy term "randn")
accelerometer_data=[0;0;-g(3)]*ones(1,N)+sqrt(Sigmaza)*randn([3,N]);
gyro_data=zeros(3,N)+sqrt(Sigmazomega)*randn([3,N]);

%% Initial state of the EKF (see pagg. 49-50)
x0=[0;0;0];
v0=[v_x;0;0];
a0=zeros(3,1);
Psi0=[pi 0 0]';
ab0=Cnb(Psi0)'*g;
omegab0=zeros(3,1);
ba0=zeros(3,1);
bomega0=zeros(3,1);

%% Computation of the nominal orientation and trajectory of the vehicle
nominal_orientation=Psi0;
nominal_trajectory=zeros(3,N+1);
nominal_trajectory(:,1)=x0;
for i=1:N
   nominal_trajectory(:,i+1)=nominal_trajectory(:,i)+v0*Ts;
end

%% Structure "integratedData" stores the generated artificial accelerometer, gyroscope and GPS measurement
```

```
profiles
integratedData={};
integratedData.timestamp=(1:N)';
integratedData.ACC_Xaxis=accelerometer_data(1,:)';
integratedData.ACC_Yaxis=accelerometer_data(2,:)';
integratedData.ACC_Zaxis=accelerometer_data(3,:)';
integratedData.GYR_Xaxis=gyro_data(1,:)';
integratedData.GYR_Yaxis=gyro_data(2,:)';
integratedData.GYR_Zaxis=gyro_data(3,:)';
integratedData.Longitude=nominal_trajectory(1,:)' +  sqrt(C_GPS(1,1))*randn(length(nominal_trajectory(1,:)),1);
integratedData.Latitude=nominal_trajectory(2,:)'  +  sqrt(C_GPS(2,2))*randn(length(nominal_trajectory(2,:)),1);
```

Test signals samples - Accelerometer

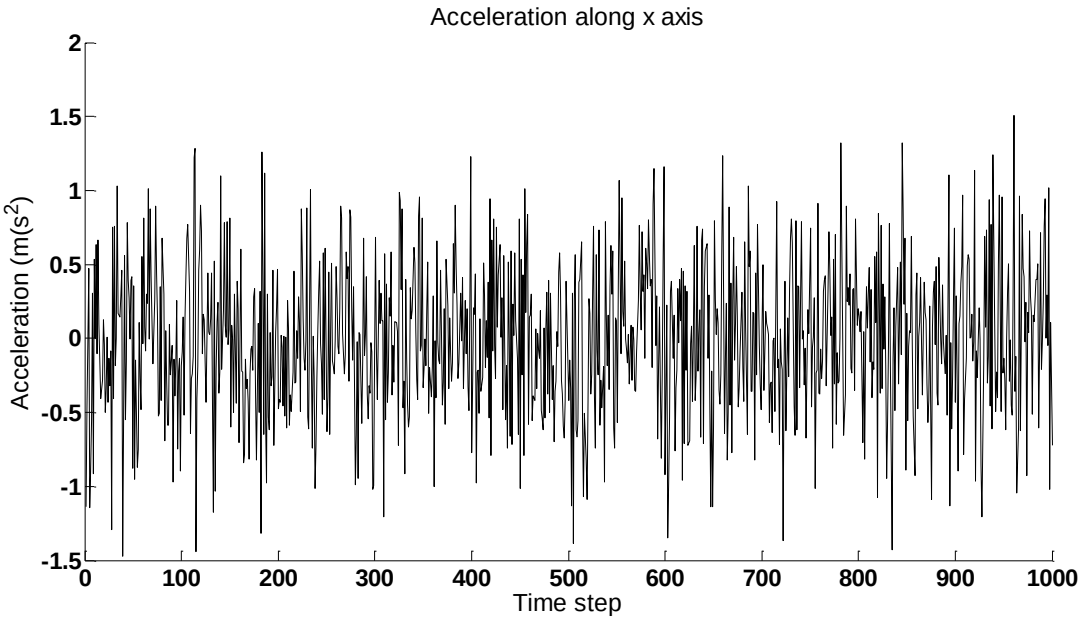

Figure 14. Artificial acceleration measurement along x axis.

The sample along the y axis is similar to the previous one and is not reported here for the sake of brevity.

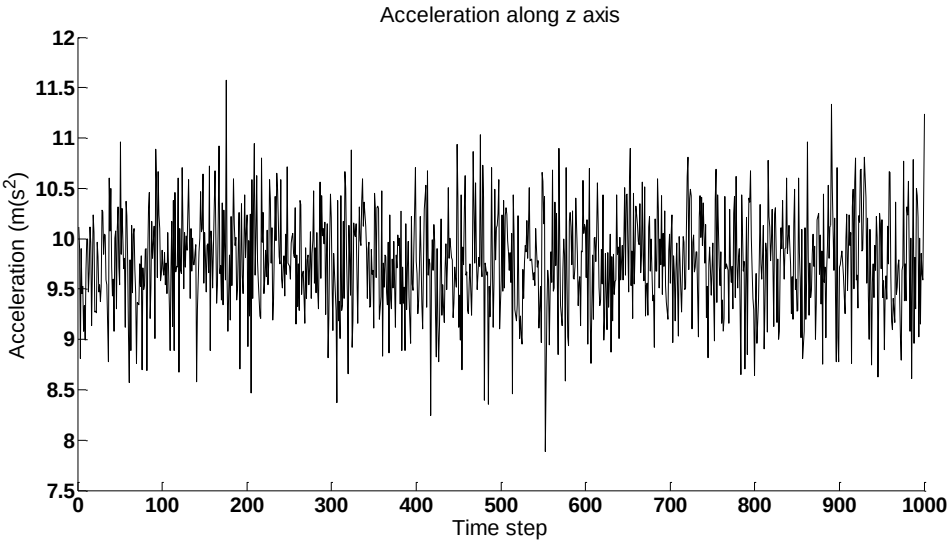

Figure 15. Artificial acceleration measurement along z axis.

Test signals samples – Gyroscope

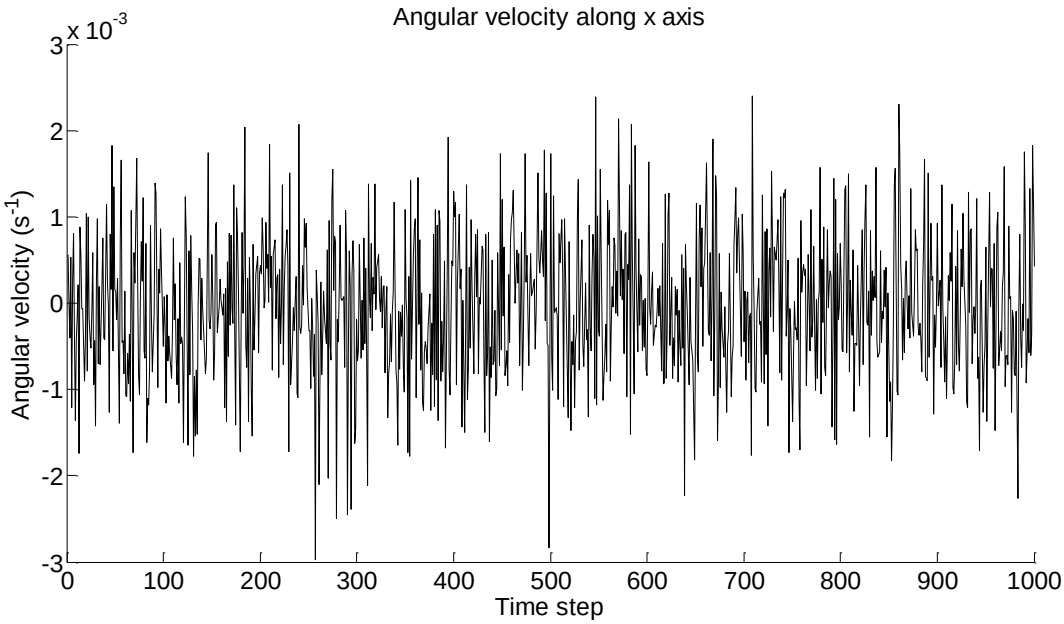

Figure 16. Artificial gyroscope measurement along x axis.

The artificial profiles for the y and z axis are similar to the above one.

7.5.2 Scenario 2: Straight Motion under Constant Acceleration – Description

In this scenario, the vehicle proceeds in straight motion (along the x-axis) with constant acceleration.

In this scenario, $\boldsymbol{a_b}$ is given by

$$\boldsymbol{a_b^2} = \begin{pmatrix} a_x \\ 0 \\ g \end{pmatrix}$$

being a_x the linear acceleration of the vehicle.

The true angular velocity $\boldsymbol{\omega_b}$ is given by

$$\boldsymbol{\omega_b^2} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Similar scenarios with non-constant acceleration profiles $a_x(t)$ could be considered, to simulate acceleration or breaking.

7.5.2.1 Scenario 1: Straight Motion at Constant Acceleration – MATLAB Code

Below we report the MATLAB script "scenario_constant_acceleration_x.m", which computes the nominal trajectory on the plane and the nominal orientation of the vehicle, along with the corresponding artificial acceleration, angular velocity and GPS measurement signals.

```
% Scenario: vehicle proceeding at constant acceleration along x axis

%% Parameters - see main.m for explanations
a_x=0.1;                    % linear acceleration along x axis
global N;
global x0;
global Sigmaza;
global Sigmazomega;
global Ts;
global nominal_orientation;
global nominal_trajectory;
global g;
global C_GPS;

%% Construction of an artificial data set from accelerometer and gyroscope (notice the noisy term "randn")
accelerometer_data=[a_x;0;-g(3)]*ones(1,N)+sqrt(Sigmaza)*randn([3,N]);
gyro_data=zeros(3,N)+sqrt(Sigmazomega)*randn([3,N]);

%% Initial state of the EKF (see pagg. 49-50)
x0=[0;0;0];
v0=zeros(3,1);
a0=[a_x;0;0];
Psi0=[pi 0 0]';
ab0=Cnb(Psi0)'*(g+[a_x;0;0]);          % **** ADDED TRANSPOSE SIGN
omegab0=zeros(3,1);
ba0=zeros(3,1);
bomega0=zeros(3,1);

%% Computation of the nominal orientation and trajectory of the vehicle
nominal_orientation=Psi0;
nominal_trajectory=zeros(3,N+1);
nominal_trajectory(:,1)=x0;
for i=1:N
```

```
    nominal_trajectory(:,i+1)=v0*(i*Ts)+1/2*a0*(i*Ts)^2;
end

%% Structure "integratedData" stores the generated artificial accelerometer, gyroscope and GPS measurement profiles
integratedData={};
integratedData.timestamp=(1:N)';
integratedData.ACC_Xaxis=accelerometer_data(1,:)';
integratedData.ACC_Yaxis=accelerometer_data(2,:)';
integratedData.ACC_Zaxis=accelerometer_data(3,:)';
integratedData.GYR_Xaxis=gyro_data(1,:)';
integratedData.GYR_Yaxis=gyro_data(2,:)';
integratedData.GYR_Zaxis=gyro_data(3,:)';
integratedData.Longitude=nominal_trajectory(1,:)' +  sqrt(C_GPS(1,1))*randn(length(nominal_trajectory(1,:)),1);
integratedData.Latitude=nominal_trajectory(2,:)'  +  sqrt(C_GPS(2,2))*randn(length(nominal_trajectory(2,:)),1);
```

Test signals samples - Accelerometer

The nominal constant acceleration is set to 0.1 m/s^2.

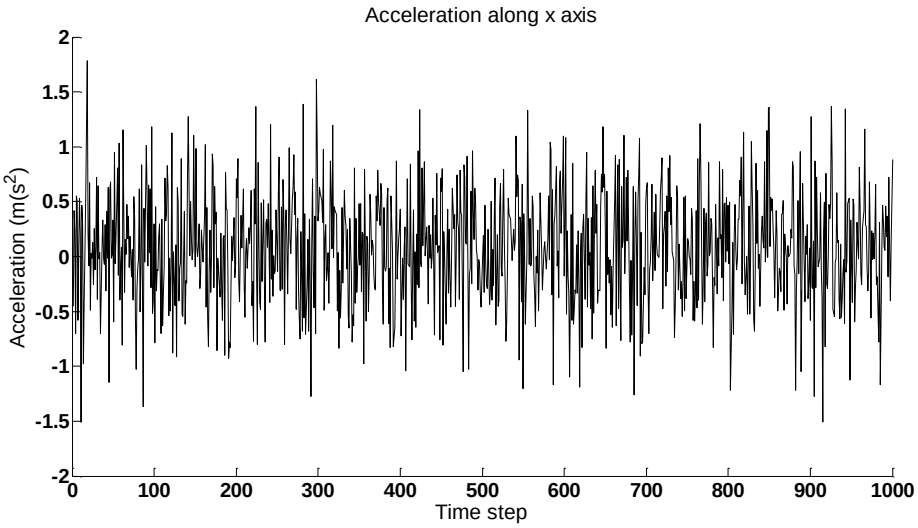

Figure 17. Artificial acceleration measurement along x axis.

The artificial acceleration profiles along y and z axis, as well as the gyroscope profiles are similar to the ones shown for the previous scenario.

7.5.3 Scenario 3: Circular Turn at Constant Velocity – Description

In this scenario, the vehicle proceeds at constant linear velocity v along a turn of circular shape and radius r, in clockwise direction.

In this scenario, $\boldsymbol{a_b}$ is given by

$$\boldsymbol{a_b^3} = \begin{pmatrix} 0 \\ v^2/r \\ g \end{pmatrix}$$

while $\boldsymbol{\omega_b}$ is given by

$$\boldsymbol{\omega_b^3} = \begin{pmatrix} 0 \\ 0 \\ v/r \end{pmatrix}$$

7.5.3.1 Scenario 1: Circular Turn at Constant Velocity – MATLAB Code

Below we report the MATLAB script "scenario_circular_turn.m", which computes the nominal trajectory on the plane and the nominal orientation of the vehicle, along with the corresponding artificial acceleration, angular velocity and GPS measurement signals.

```
% Scenario: vehicle proceeding along a circular turn, at constant velocity

%% Parameters
r=50;                       % Radius of the turn
v=8.3;                      % Velocity of the vehicle
global N;
global x0;
global Sigmaza;
global Sigmazomega;
global Ts;
global nominal_orientation;
global nominal_trajectory;
global C_GPS;

%% Construction of an artificial data set from accelerometer and gyroscope (notice the noisy term "randn")
accelerometer_data=[0.0;0;0]*ones(1,N)    +      [0;v^2/r;-g(3)]*ones(1,N)+sqrt(Sigmaza)*randn([3,N]);              %
******  METTERE QUI IL BIAS  ******
gyro_data=[0;0;v/r]*ones(1,N)+sqrt(Sigmazomega)*randn([3,N]);

%% Initial state of the EKF (see pagg. 49-50)
x0=[0;0;0];
v0=[v;0;0];
a0=[0;-v^2/r;0];
Psi0=[pi 0 0]';
ab0=Cnb(Psi0)'*[0;-v^2/r;-9.8];        % **** ADDED TRANSPOSE SIGN
omegab0=[0;0;v/r];                  % In body frame
ba0=zeros(3,1);
bomega0=zeros(3,1);
```

```
%% Computation of the nominal orientation and trajectory of the vehicle
nominal_orientation=zeros(3,N+1);
nominal_orientation(:,1)=Psi0;
for i=1:N
   nominal_orientation(:,i+1)=nominal_orientation(:,i)+[0;0;v/r*Ts];
end
nominal_trajectory=zeros(3,N+1);
nominal_trajectory(:,1)=x0;
for i=2:N+1
   nominal_trajectory(:,i)=[r*sin(nominal_orientation(3,i)); -r+r*cos(nominal_orientation(3,i)); 0];
end

%% Structure "integratedData" stores the generated artificial accelerometer, gyroscope and GPS measurement profiles
integratedData={};
integratedData.timestamp=(1:N)';
integratedData.ACC_Xaxis=accelerometer_data(1,:)';
integratedData.ACC_Yaxis=accelerometer_data(2,:)';
integratedData.ACC_Zaxis=accelerometer_data(3,:)';
integratedData.GYR_Xaxis=gyro_data(1,:)';
integratedData.GYR_Yaxis=gyro_data(2,:)';
integratedData.GYR_Zaxis=gyro_data(3,:)';
integratedData.Longitude=nominal_trajectory(1,:)' +  sqrt(C_GPS(1,1))*randn(length(nominal_trajectory(1,:)),1);
integratedData.Latitude=nominal_trajectory(2,:)'  +  sqrt(C_GPS(2,2))*randn(length(nominal_trajectory(2,:)),1);
```

Test signals samples - Accelerometer

The vehicle proceeds along a roundabout of radius r=50 m, at a velocity of v=8.3 m/s. As a result, the acceleration test signals report a lateral centripetal acceleration (nominal value of v^2/r=1.37 m/s^2), as shown in the following figure (acceleration along the x and z axis are not reported as they are similar to the what illustrated in the previous scenarios).

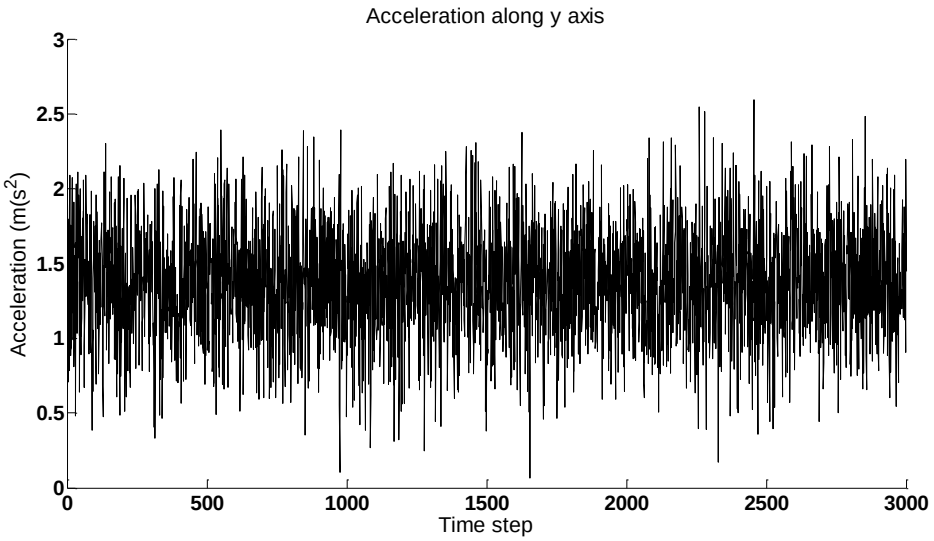

Figure 18. Artificial acceleration measurement along y axis.

Test signals samples – Gyroscope

The gyroscope test signal around the z axis report a nominal angular velocity of v/r=0.166 rad/s, with superimposed AGN noise. The signal is shown in the following figure.

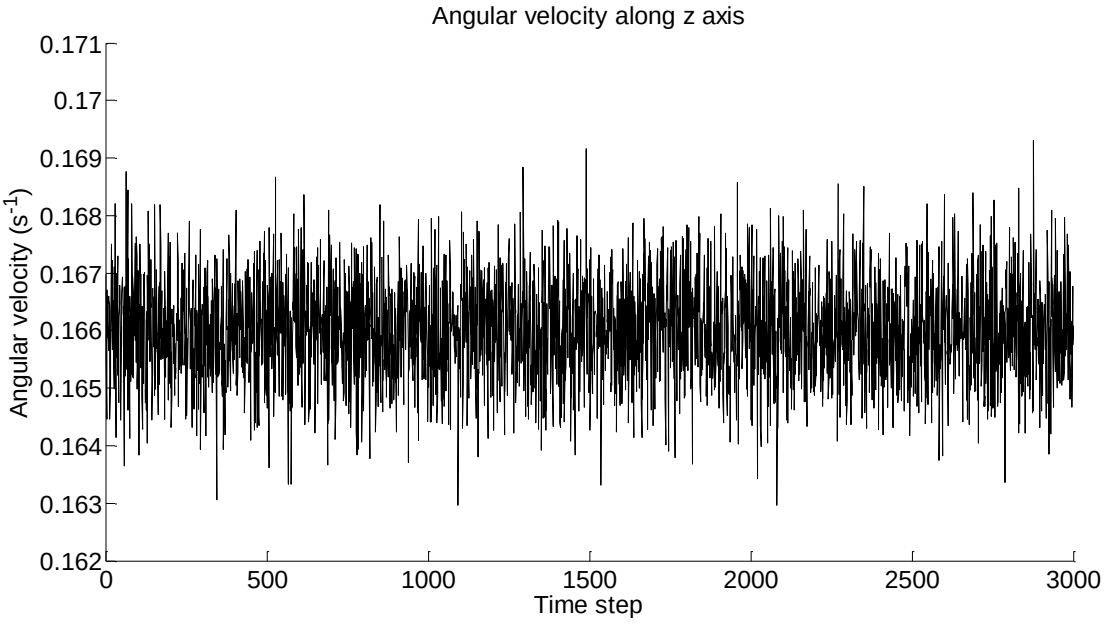

Figure 19. Artificial acceleration measurement along z axis.

7.5.4 Scenario 4: Junction at Constant Velocity – Description

For simplicity, we consider in the following a junction modelled as a portion of a circular turn with long radius (see Figure 20).

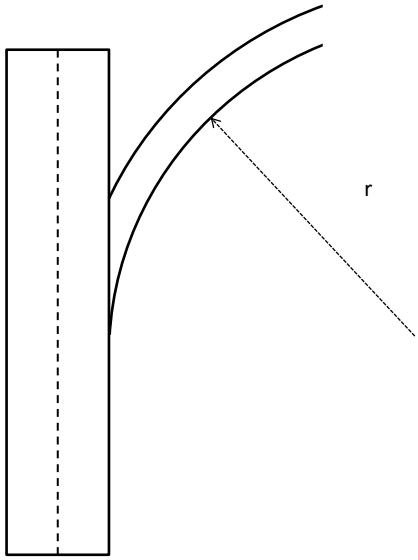

Figure 20. Scheme of a junction

Acceleration and rotation profiles are therefore again

$$\boldsymbol{a}_{\boldsymbol{b}}^{\boldsymbol{4}} = \begin{pmatrix} 0 \\ v^2/r \\ g \end{pmatrix}$$

$$\boldsymbol{\omega}_{\boldsymbol{b}}^{\boldsymbol{4}} = \begin{pmatrix} 0 \\ 0 \\ v/r \end{pmatrix}$$

with variable values of r depending on the curvature of the junction.

The MATLAB code and the test signals are not reported as this case can be modelled as a particular case of the above scenario.

7.5.5 Scenario 5: Road Hump at Constant Velocity – Description

In this scenario, the vehicle passes over a road hump while proceeding in straight line (along the x-axis) at constant speed.

The hump causes sudden accelerations along the z-axis, resulting in an acceleration profiles for $\boldsymbol{a}_{\boldsymbol{b}}$ of the kind exemplified in Figure 21, in which it is possible to recognize two rectangular impulses modelling acceleration caused by the impact with the hump.

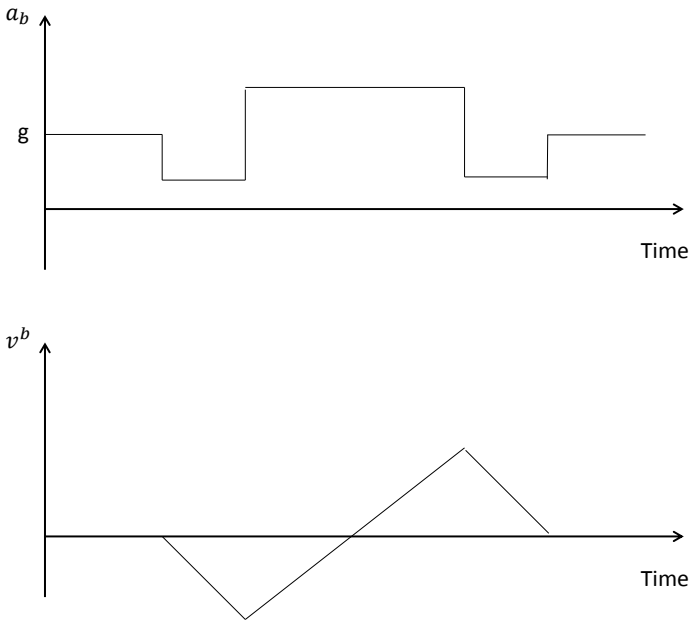

Figure 21. Simplified acceleration and velocity profiles for road hump scenario

For simplicity we assume that the vehicle does not experience any rotation

$$\boldsymbol{\omega}_{\boldsymbol{b}}^{\mathbf{5}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

7.5.5.1 Scenario 5: Road Hump at Constant Velocity – MATLAB Code

Below we report the MATLAB script "scenario_hump_constant_velocity_x.m", which computes the nominal trajectory on the plane and the nominal orientation of the vehicle, along with the corresponding artificial acceleration, angular velocity and GPS measurement signals.

```
% Scenario: vehicle encountering a road hump while proceeding at constant velocity along x axis

%% Parameters - see main.m for explanations
v_x=10;                     % linear velocity along x axis
global N;
global x0;
global Sigmaza;
global Sigmazomega;
global Ts;
global nominal_orientation;
global nominal_trajectory;
global g;
global C_GPS;
n1=200;          % car proceeding at constant velocity along x axis (duration in time steps)
n2=5;            % parameter to model the duration (in time steps) of the hump
n3=2*n2;         % parameter to model the duration (in time steps) of the hump

%% Construction of an artificial data set from accelerometer and gyroscope (notice the noisy term "randn")
accelerometer_data=[ [0;0;-g(3)]*ones(1,n1) , [0;0;-g(3)/2]*ones(1,n2) , [0;0;-1.5*g(3)]*ones(1,n3) , [0;0;-
g(3)/2]*ones(1,n2) , [0;0;-g(3)]*ones(1,N-n1-2*n2-n3)] + sqrt(Sigmaza)*randn([3,N]);
gyro_data=zeros(3,N)+sqrt(Sigmazomega)*randn([3,N]);

%% Initial state of the EKF (see pagg. 49-50)
x0=[0;0;0];
v0=[v_x;0;0];
a0=zeros(3,1);
Psi0=[pi 0 0]';
ab0=Cnb(Psi0)'*g;
omegab0=zeros(3,1);
ba0=zeros(3,1);
bomega0=zeros(3,1);

%% Computation of the nominal orientation and trajectory of the vehicle
nominal_orientation=Psi0;
nominal_trajectory=zeros(3,N+1);
nominal_trajectory(:,1)=x0;
for i=1:N
   nominal_trajectory(:,i+1)=nominal_trajectory(:,i)+v0*Ts;
end

%% Structure "integratedData" stores the generated artificial accelerometer, gyroscope and GPS measurement
profiles
integratedData={};
integratedData.timestamp=(1:N)';
integratedData.ACC_Xaxis=accelerometer_data(1,:)';
integratedData.ACC_Yaxis=accelerometer_data(2,:)';
integratedData.ACC_Zaxis=accelerometer_data(3,:)';
integratedData.GYR_Xaxis=gyro_data(1,:)';
integratedData.GYR_Yaxis=gyro_data(2,:)';
```

```
integratedData.GYR_Zaxis=gyro_data(3,:)';
integratedData.Longitude=nominal_trajectory(1,:)' +  sqrt(C_GPS(1,1))*randn(length(nominal_trajectory(1,:)),1);
integratedData.Latitude=nominal_trajectory(2,:)'  +  sqrt(C_GPS(2,2))*randn(length(nominal_trajectory(2,:)),1);
```

Test signals samples - Accelerometer

Below a proposed artificial acceleration measurement profile along the z axis is reported for this scenario. The other accelerometer and gyroscope profiles are not reported since they are similar to the ones discussed previously.

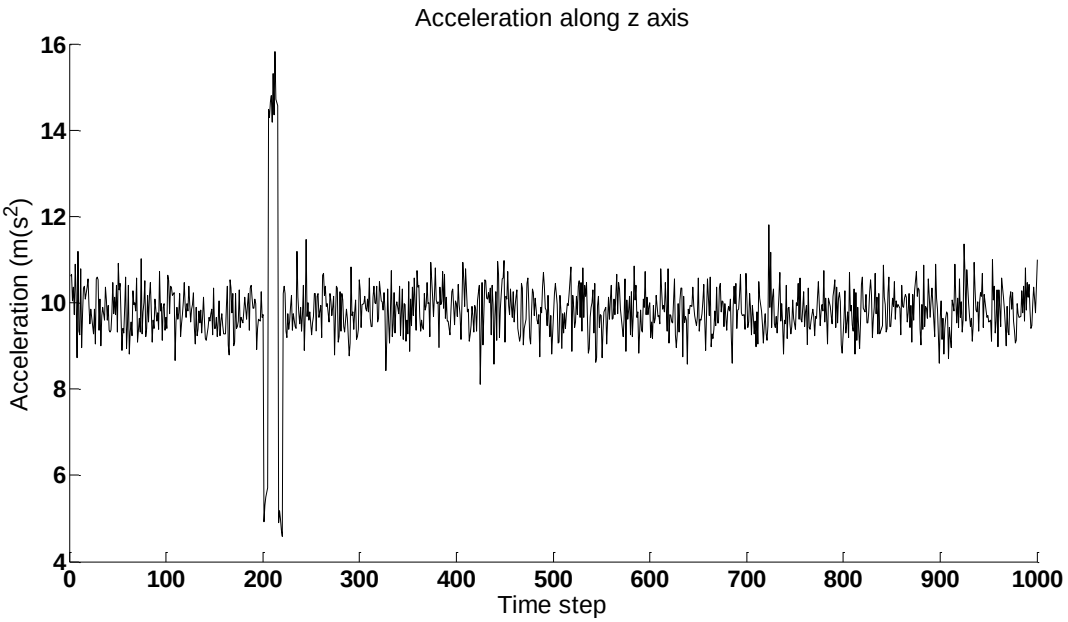

Figure 22. Artificial acceleration measurement along z axis.

8 Simulation and results

Below the implemented navigation solution is tested on the artificial test signals built in Section 0.

8.1 Scenario 1: Straight Motion at Constant Velocity – Test Result

Figure below reports the result of the test considering a vehicle moving on the horizontal plane (along the x axis of the navigation frame) at constant velocity (10 m/s). The figure reports the following information:

- *The nominal trajectory (in red).* This curve describes the path actually followed by the vehicle, and is computed considering the nominal values of angular/linear velocity and acceleration used to compute the artificial test signals (the test signals are computed by superimposing AGN noise to the nominal values of acceleration and angular velocity. The AGN is characterized by a covariance comparable to the one of the noise affecting the real accelerometer and gyroscope measurements that were available from measurement campaigns conducted at ISPRA).

- *GPS Measures (in green).* The artificial GPS measures are generated starting from the nominal trajectory and superimposing AGN noise characterized by a covariance comparable to the one of the noise affected the real GPS measurements available from the field experience at ISPRA.
- *EKF Trajectory (in blue).* The path that is obtained starting from the initial nominal position of the vehicle and then considering the estimation of the displacement of the vehicle at each time step, as estimated based on EKF information (the displacement undergone by the vehicle in a time step can be estimated based on the EKF estimations of the vehicle velocity and acceleration). It is worth noting that this curve is reported only to the purpose of evaluating the performances of the EKF position estimation (by comparing it to the nominal trajectory), as the initial actual position of the vehicle is not known (only a noisy measurement of the initial position is available from the GPS, and is actually taken into account by the PF).
- Position estimation (in black). The final estimation of the vehicle position as computed by the PF, fusing differential positioning information from the EKF (actually, from the "pdf estimation module") and absolute positioning information from the GPS.

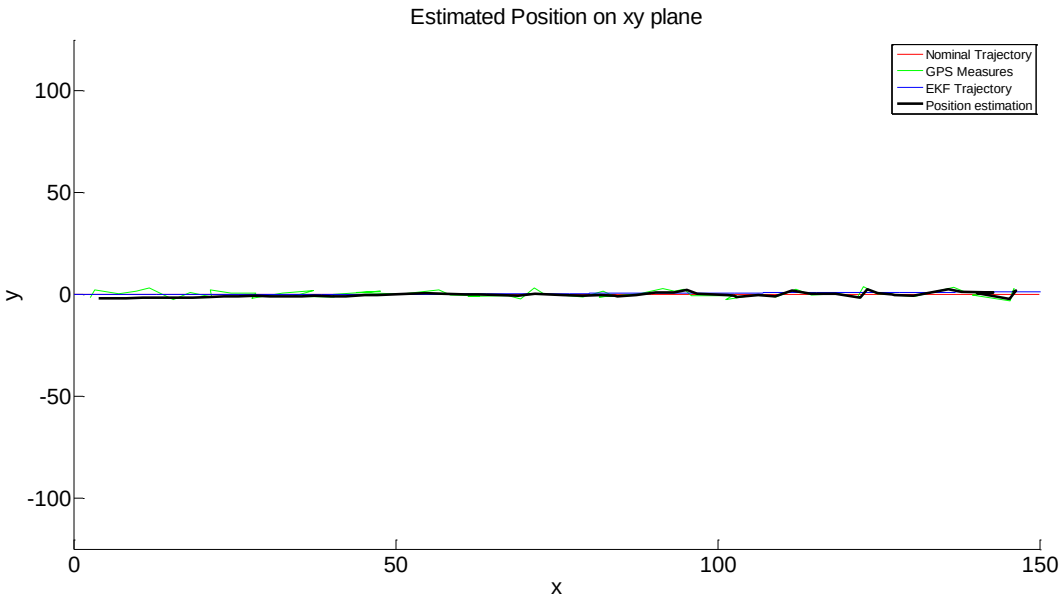

Figure 23. Scenario 1: estimated path on xy plane.

Figure 24 reports a zoom centered in the initial phase of the simulation. At the initial time, the PF estimate is influenced by the measure of the GPS position. In fact, it is seen that the error of the estimate is comparable with the error on the GPS. Then, as GPS measures and information from the EKF are progressively fused, the error is reduced, going below 0.5 m after 30 meters of run. Then, from Figure 25 (which refers to the final phases of the simulations) it can be seen that the error starts growing from the minimum, mainly because the information coming from the EKF deteriorates. As a matter of fact, the covariance characterising the EKF estimates progressively grows, up to the point in which the PF practically ignores EKF estimates and only relies on GPS information (notice in fact that the PF estimate at initial times is much more smoother than the estimates at the final times, because it is driven mainly by the EKF, which provides accurate estimates at the first samples). Figure 26 finally reports the positioning error, evaluated by comparison of the PF position estimate with the nominal trajectory.

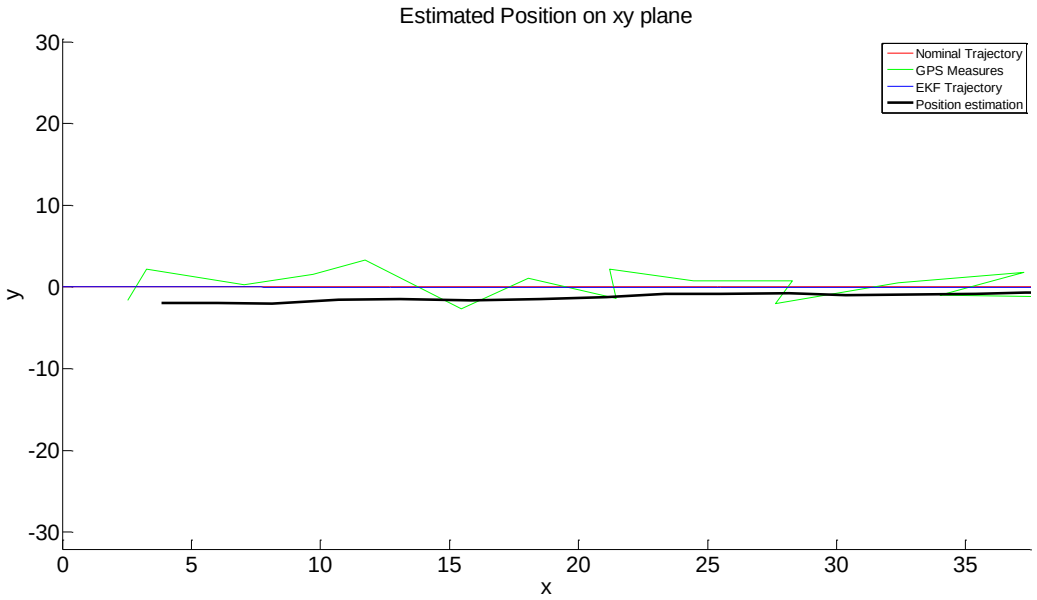

Figure 24. Scenario 1: estimated path on xy plane, zoom on the initial simulation times.

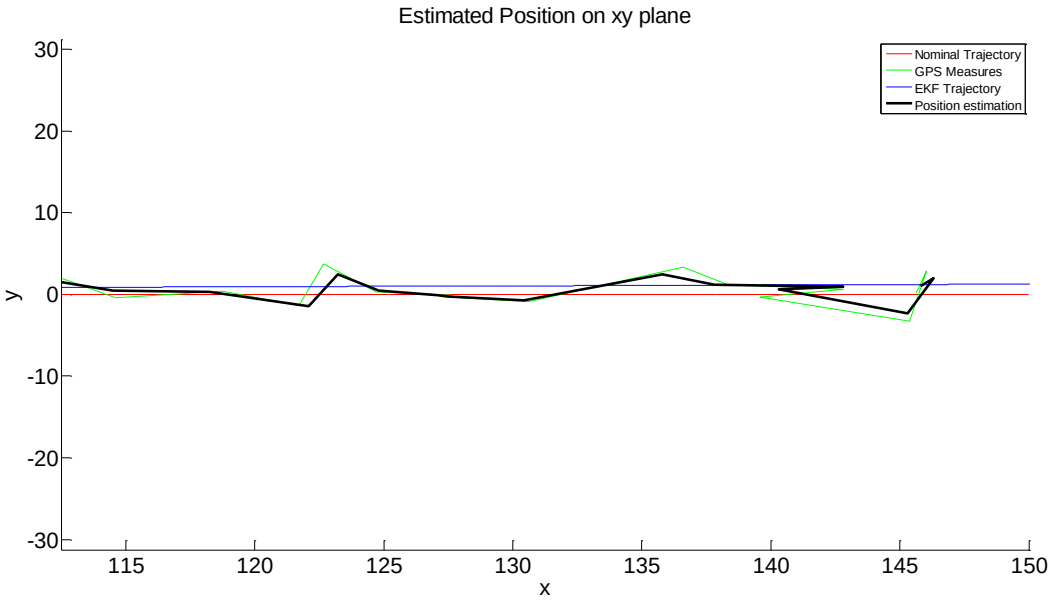

Figure 25. Scenario 1: estimated path on xy plane, zoom on the final simulation times.

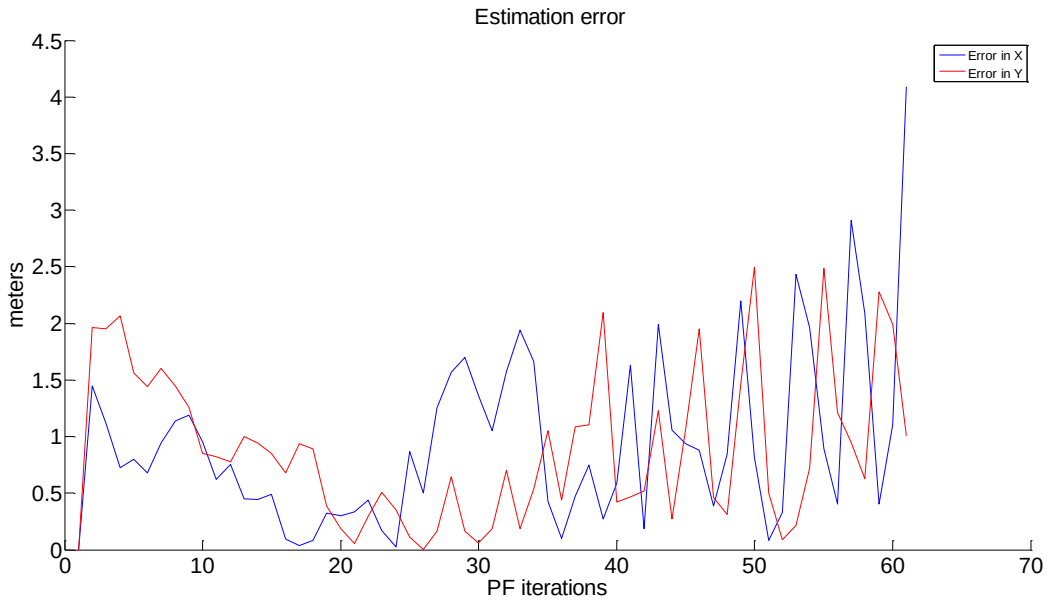

Figure 26. Scenario 1: estimated path on xy plane, error with respect to the nominal path.

To remedy to the degradation of EKF estimates (which we argue is due to the approximations of nonlinear dynamics), we propose an *EKF restart approach* according to which the EKF estimate and the associated covariance are periodically (each 250 samples in the simulations, i.e. each 10 PF executions) reset (the new starting initial state is set to the last EKF state estimation, while the covariance is restarted from to the initial covariance value). From Figure 27, Figure 28 and Figure 29 it can be seen that performances improve notably, and the estimate remains smooth also at the final times. Figure 30 furthermore shows that the error remains significantly low during all the simulation.

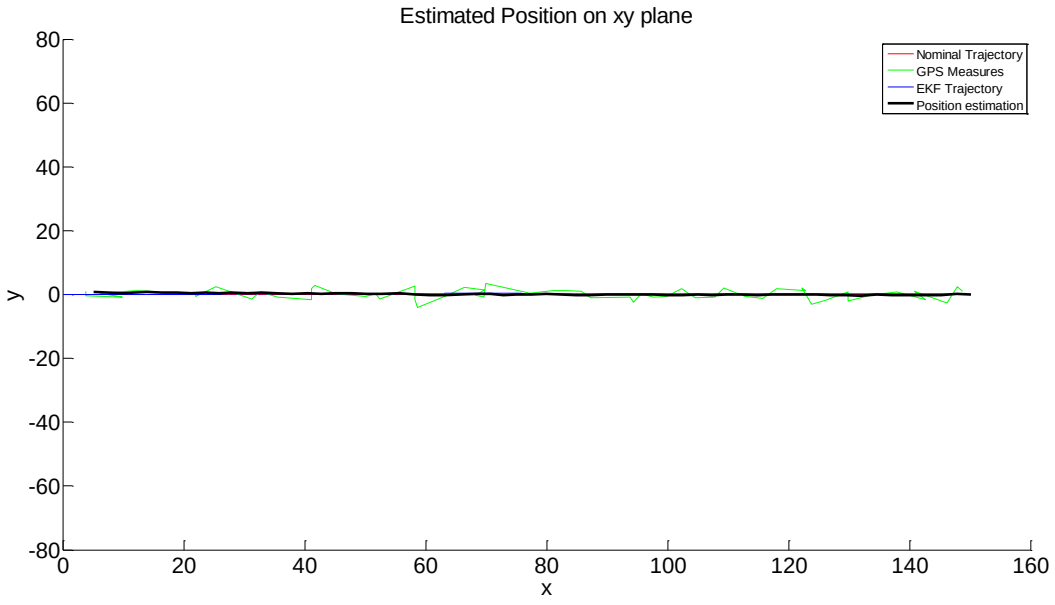

Figure 27. Scenario 1: estimated path on xy plane – with EKF restart.

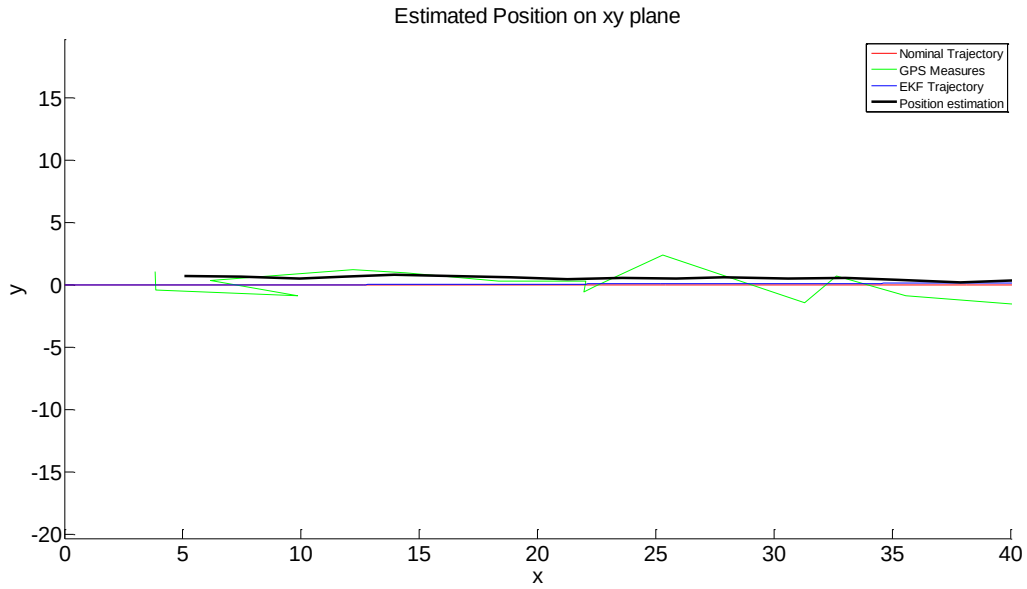

Figure 28. Scenario 1: estimated path on xy plane, zoom on the initial simulation times – with EKF restart.

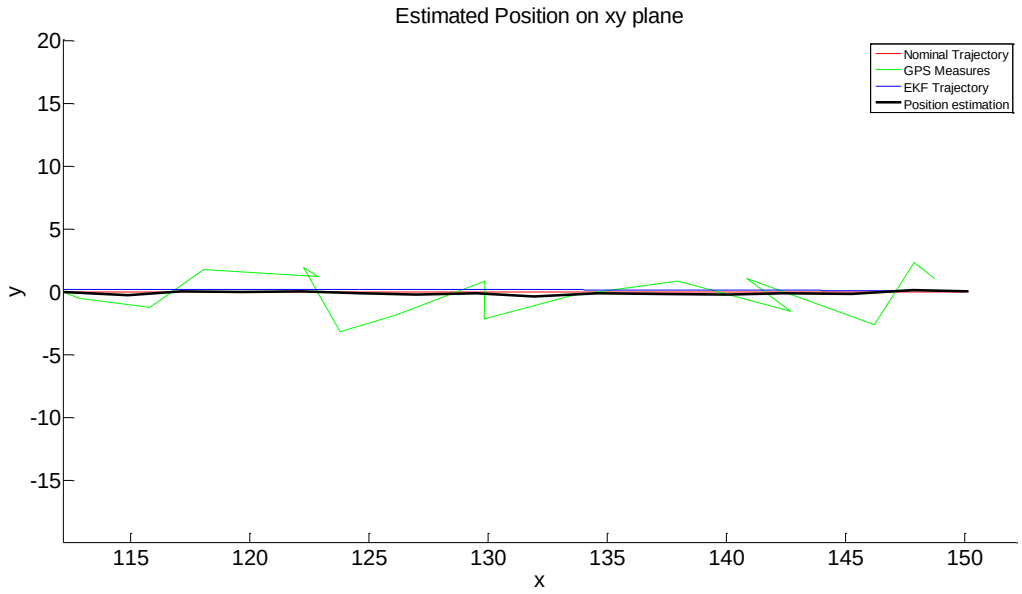

Figure 29. Scenario 1: estimated path on xy plane, zoom on the final simulation times – with EKF restart.

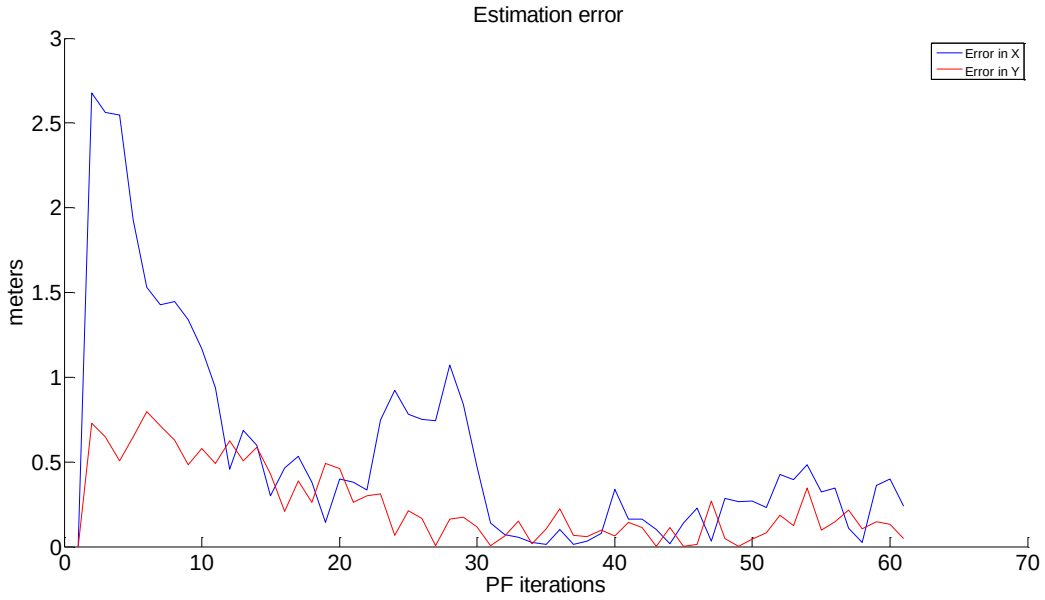

Figure 30. Scenario 1: error with respect to the nominal path – with EKF restart.

8.2 Scenario 2: Straight Motion at Constant Acceleration – Test Result

Figure 31, Figure 32, Figure 33 and Figure 34 below reports the result of the test considering a vehicle moving on the horizontal plane (along the x axis of the navigation frame) at constant acceleration (0.1 m/s^2) and zero initial velocity. Figures refer to navigation system with EKF periodic restart, with the same restart periodicity as in the simulation above.

The figures clearly show that also in this case the solution significantly improves the quality of positioning information available from the GPS alone. The estimation remains smooth and close to the nominal path.

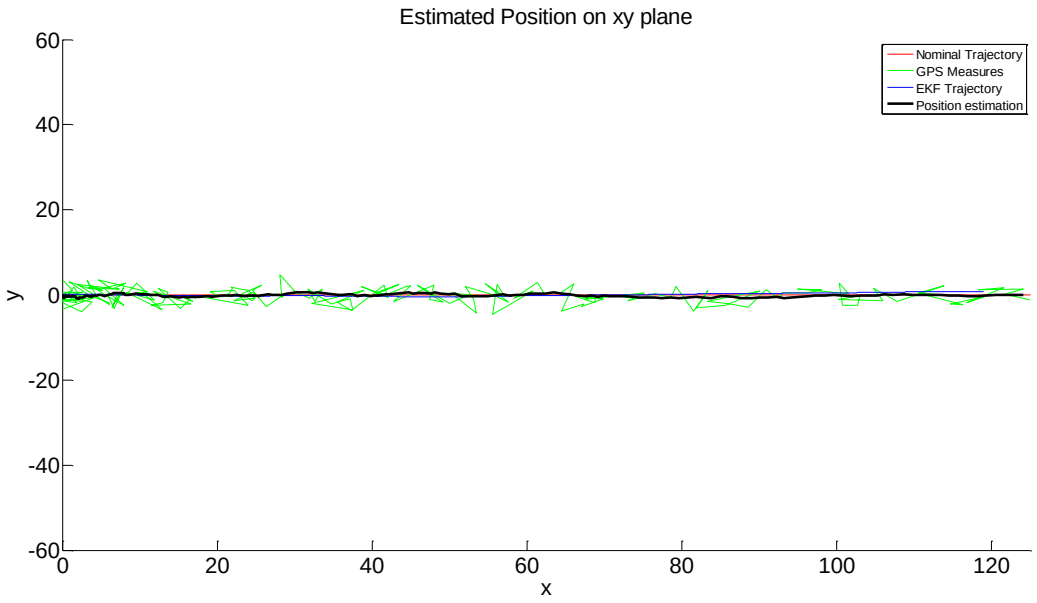

Figure 31. Scenario 2: estimated path on xy plane – with EKF restart.

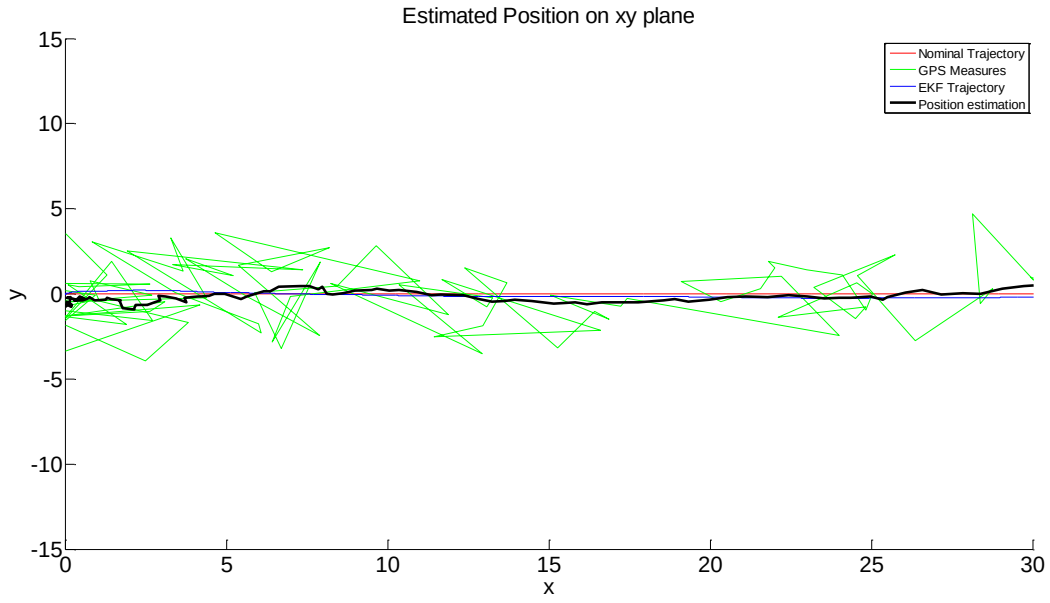

Figure 32. Scenario 2: estimated path on xy plane, zoom on the initial simulation times – with EKF restart.

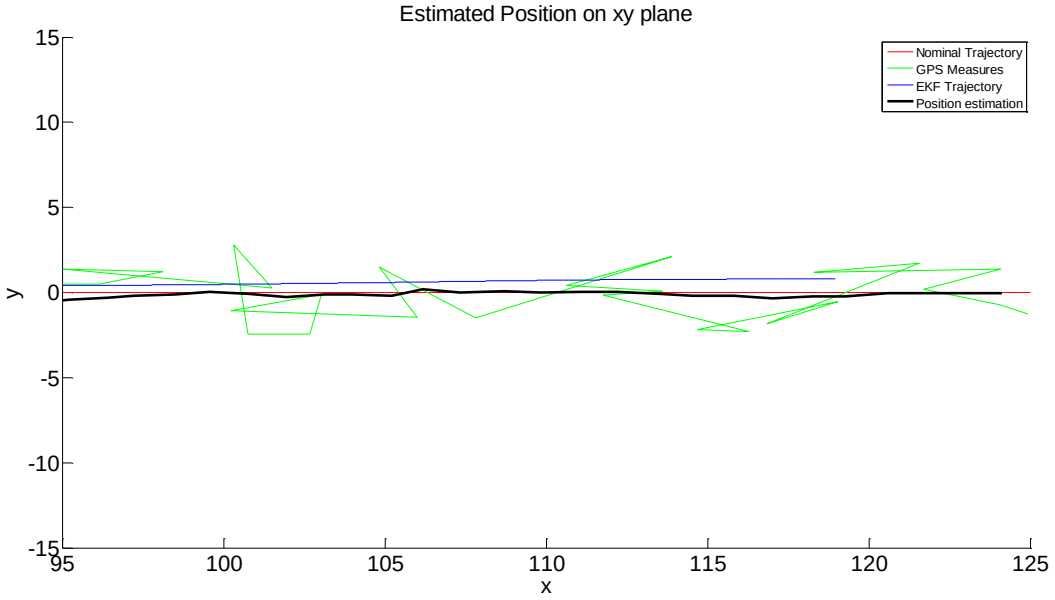

Figure 33. Scenario 2: estimated path on xy plane, zoom on the final simulation times – with EKF restart.

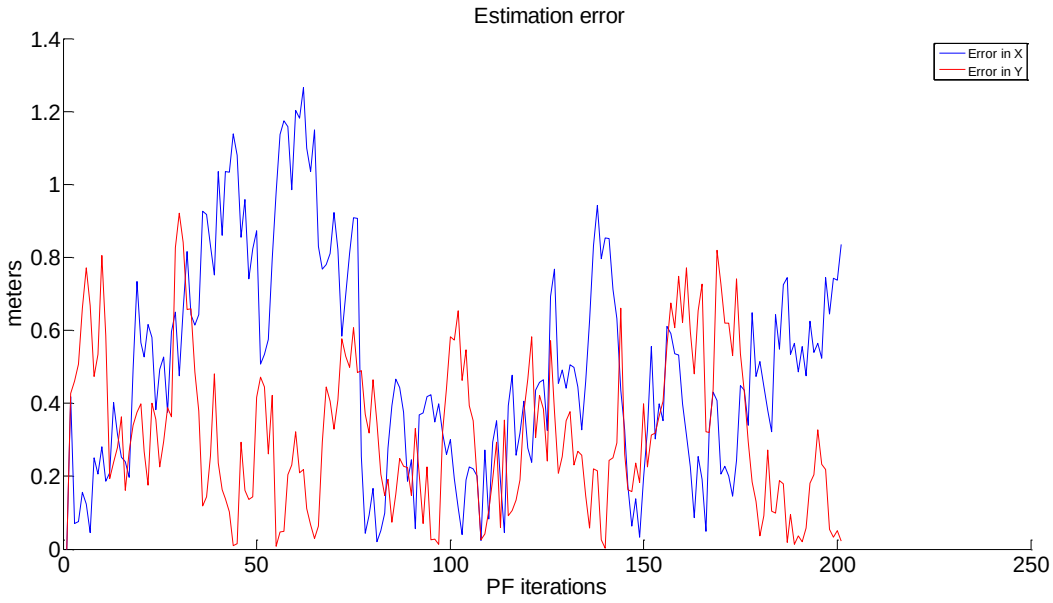

Figure 34. Scenario 2: error with respect to the nominal path – with EKF restart.

In the following the same kind of test is repeated including information on map constraint (a carriageway 3 meters wide). Such information is processed by the PF, to give different weights to the particles depending on if they violate or not the constraints. Results of the simulation are provided in the figures below.

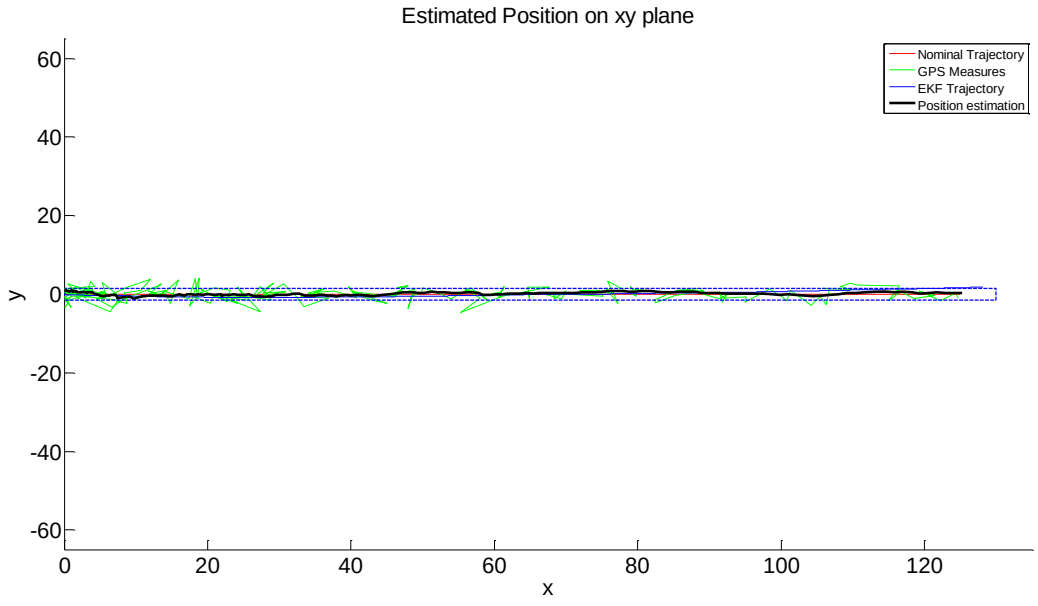

Figure 35. Scenario 2: estimated path on xy plane – with EKF restart and map constraints included.

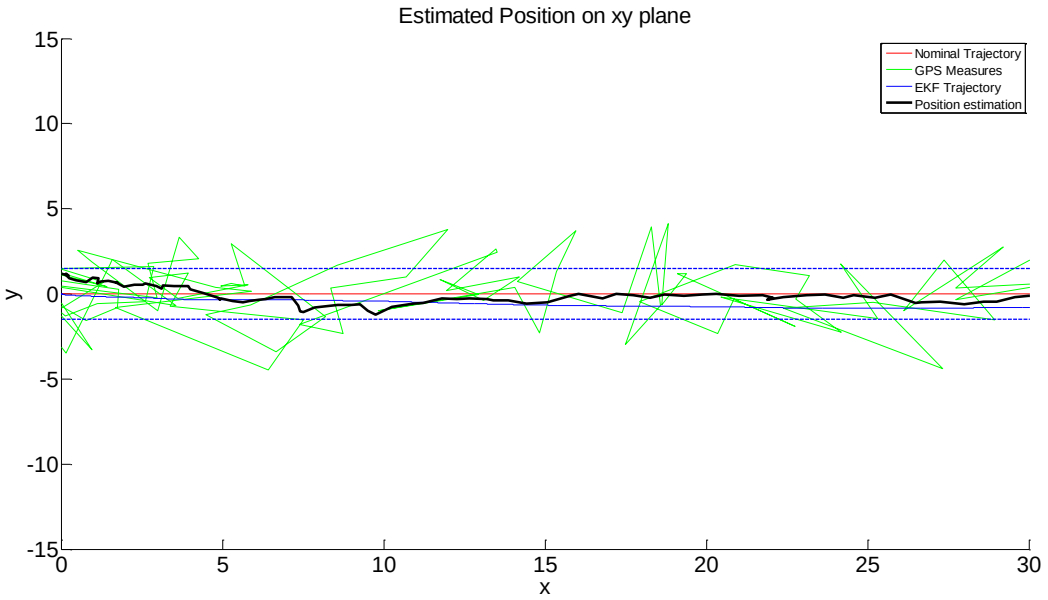

Figure 36. Scenario 2: estimated path on xy plane, zoom on the initial simulation times – with EKF restart and map constraints included.

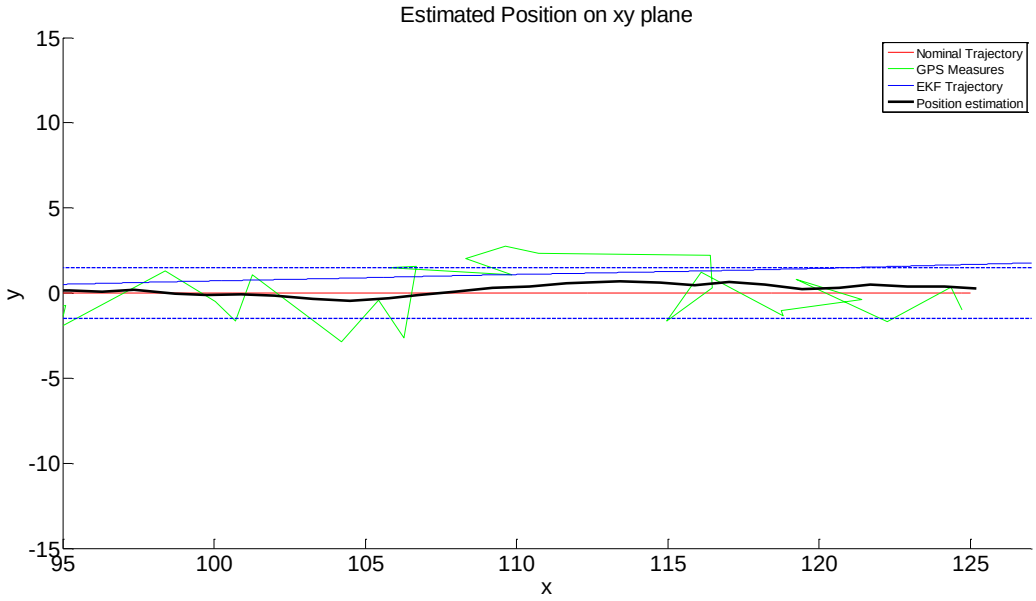

Figure 37. Scenario 2: estimated path on xy plane, zoom on the final simulation times – with EKF restart and map constraints included.

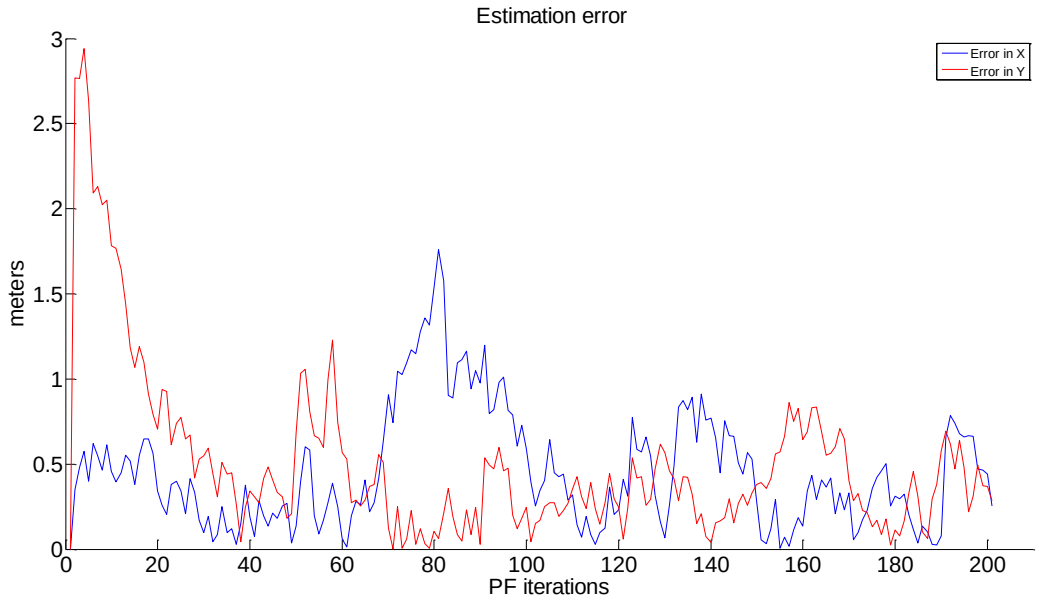

Figure 38. Scenario 2: error with respect to the nominal path – with EKF restart and map constraints included.

8.3 Scenario 3: Circular Turn at Constant Velocity – Test Result

The Figures below report the result of the test considering a vehicle moving along a circular turn (a roundabout with radius r=50 m) at constant velocity (v=10 m/s).

The result reported in Figure 39 is interesting under different perspectives. First of all, it can be noticed how, again, the integrated solution is able to recover from the error on the initial estimate provided based on GPS information. Then, the EKF (actually, the estimate built by the "pdf estimation module" based on EKF information) keeps tracking accurately the nominal trajectory for a certain amount of time (before it separates from the nominal trajectory). The PF estimate is smooth and tracks the nominal trajectory with a low error (see Figure 40; notice also from the figure how the initial error is quickly reduced).

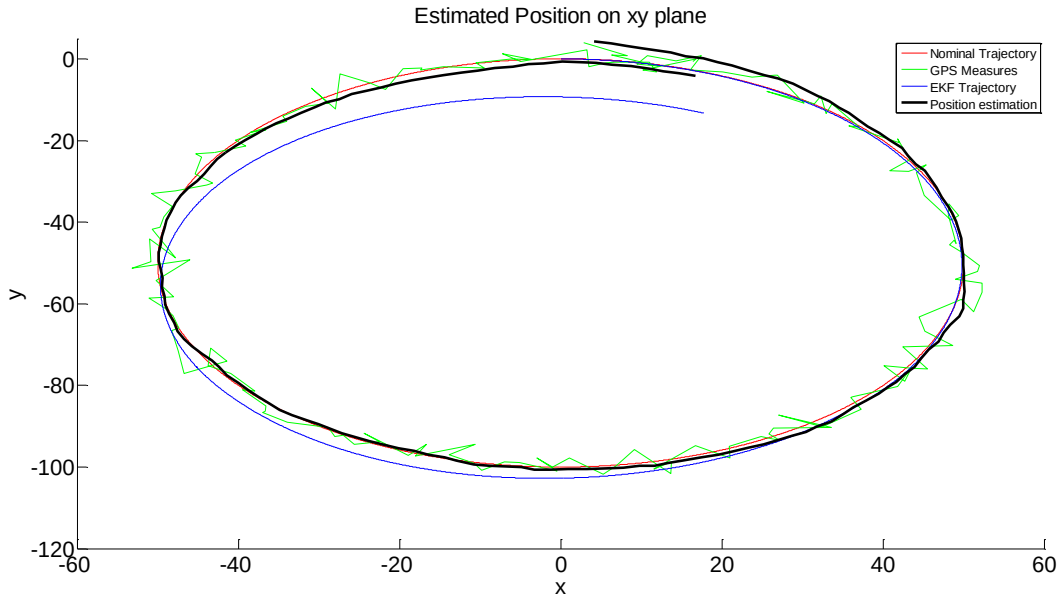

Figure 39. Scenario 3: estimated path on xy plane – with EKF restart

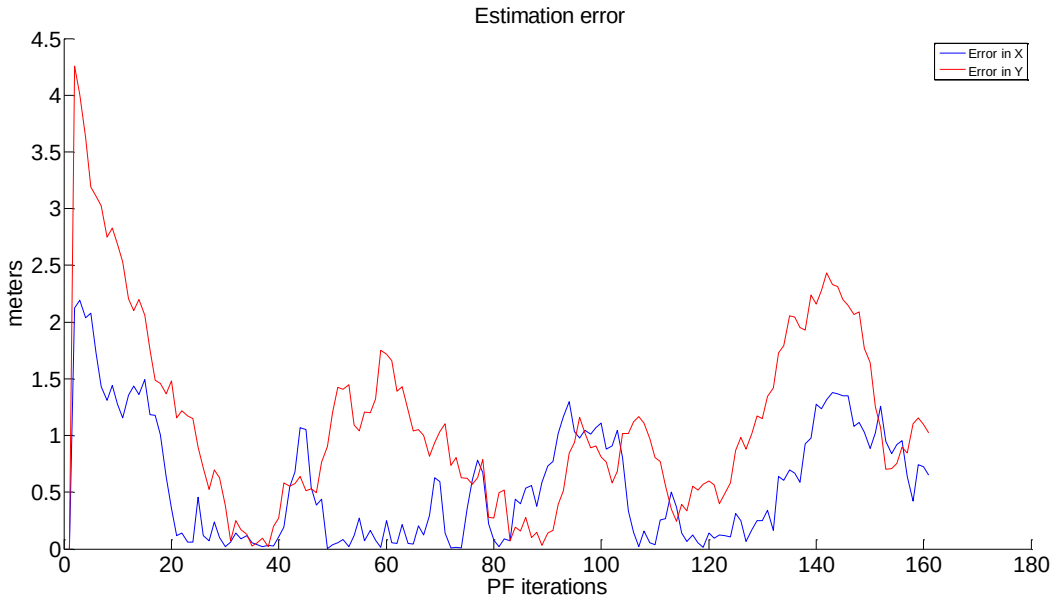

Figure 40. Scenario 4: error with respect to the nominal path – with EKF restart.

8.4 Scenario 4: Junction at Constant Velocity – Test Result

This scenario is the composition of scenario 1 (Straight Motion at Constant Velocity) and scenario 3 (Circular Turn at Constant Velocity), please refer to the relative sections to analyse the test results.

8.5 Scenario 5: Road Hump at Constant Velocity – Test Result

Figures below refer to the test of scenario 5. Also in this case the solution performs well, with a small estimation error (see Figure 41 and Figure 42).

In particular, Figure 43 reports the velocity of the vehicle, in navigation frame, as estimated by the EKF. Along the z axis it can be recognized the oscillation caused by the hit of the hump.

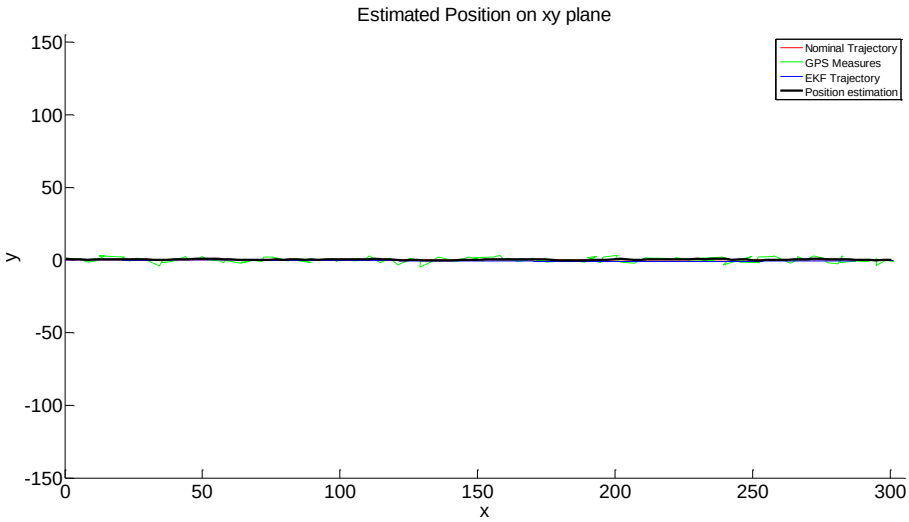

Figure 41. Scenario 3: estimated path on xy plane – with EKF restart

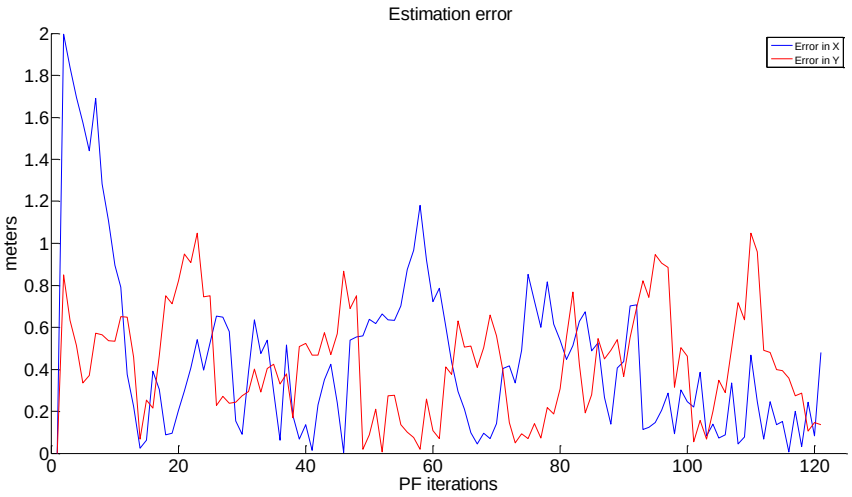

Figure 42. Scenario 4: error with respect to the nominal path – with EKF restart.

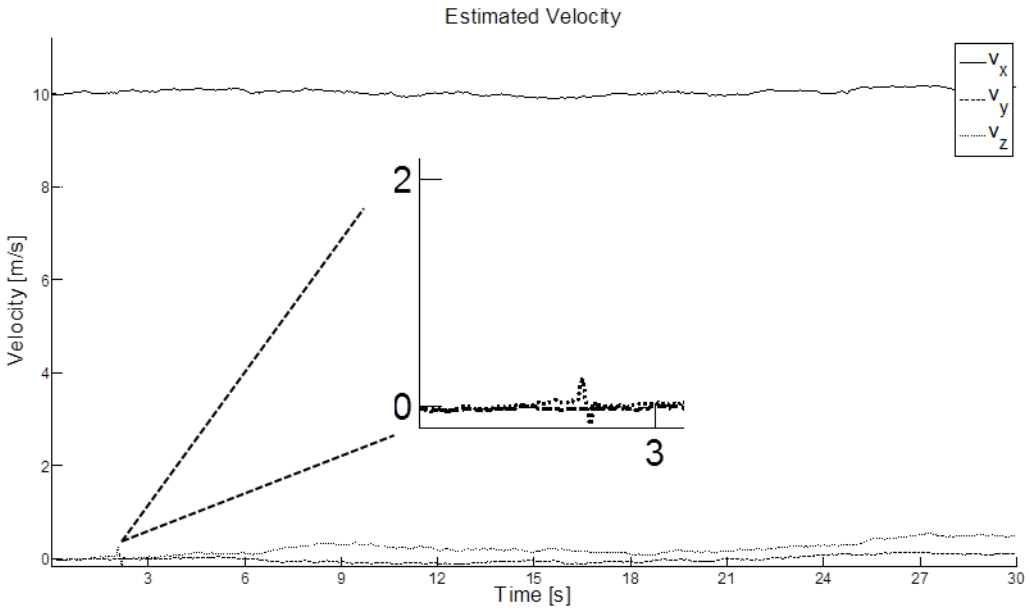

Figure 43. Velocity of the vehicle as estimated by the EKF.

9 Conclusions

This work describes the design and the implementation of a MATLAB library providing accurate positioning functionalities based on the cooperation between an extended Kalman filter, an estimation module and a particle filter. The developed solution has been tested in several scenarios that simulate meaningful vehicle situations.

The simulation results shows that an open-loop application of the positioning framework behaves better than a GPS during the initial phase of the simulation, when the covariance provided by the EKF is relatively small. Unfortunately the information coming from the EKF deteriorates very fast (due to the approximations of nonlinear dynamics) and up to the point in which the PF ignores the EKF estimates and relies only on the GPS information.

The authors provided a solution to lower the effect of the EKF estimates degradation, introducing a *EKF restart* that periodically resets the EKF so that the new starting initial state is set to the last EKF state estimation, while the covariance is restarted from to the initial covariance value. This closed-loop workaround provides very good results, but requires some tuning to perform in a satisfactory way.

10 Recommendations

This section discusses a number of recommendations about potential improvements of the ITS Mathematical Model developed in the project.

The first set of recommendations regards suggested additional steps that could be performed to further test the implemented ITS solution:

- The following additional scenarios could be considered to test the system in normal conditions:
  - Vehicle proceeding along a horizontal plane and then encountering a ramp with constant inclination. This scenario entails a certain complexity in building corresponding artificial test signals, and it is for this reason suggested that it is tested relying on real field data.
  - Improve length and complexity of itinerary (e.g. up to 20 minutes driving time) to test the resilience of the model on longer track and different sequence of signals.
- The following scenarios could be considered to test the system in abnormal vehicle situations, like for example:
  - Loss of adherence: vehicle sliding.
  - Loss of adherence: car spin round.
  - Car accident: car crashing against a fixed structure.
  - Car accident: lateral collisions.
  - Car accident: rear-end collisions.
  - Car going out of the road path.

  It may be significant to test the performances of the systems in these "abnormal" scenarios, since they are relevant for possible important practical applications (e.g. accident detection, fraud detection, violation of traffic rules, etc.). On the other hand, it is expected that such scenarios are challenging ones for the developed ITS system, since they stress some of the assumptions and approximations on which the ITS systems itself relies (e.g. the linearization of the many dynamics involved, violation of the non-holonomic constraints, violation of the map constraints, etc.).

- Test the system on real field data. In this report, real field data have been considered to derive a characterization of the noise affecting accelerometer, gyroscope and GPS measurements (in particular, to derive an estimate of the noise covariance). Such noise characterization has been used to build realistic synthetic test signals for the different scenarios considered in the report (i.e. test signals built starting from nominal acceleration/angular velocity profiles corresponding to the chosen scenarios, corrupted with noise characterized by the estimated covariance).
- Managing real-time measurements: the following model assumes that data is recorded, stored, synchronized and processed and is not real time. There are several difficulties to be faced when testing the system on real field data. For example, there is not typically an high synchronization of the data acquisition from the different sensors, so that data from the different sensors are associated with different timestamps. This implies that the sampling step considered by the Extended Kalman Filter may not be assumed constant, but it has rather to be measured from the data. Furthermore, experimental data may be in general affected by wrong, faulty measurements, so that they have to be in some way pre-processed before they are analysed by the positioning modules.

- Test the systems on different road tracks (e.g. concrete roads, asphalt roads, rural unpaved road).
- Conduct an accurate tuning campaign. A very relevant aspect of future work could regard the accurate tuning of the many parameters needed to setup the MATLAB implementation of the systems (in particular the covariance values). From experimental trials on the matter, it is expected that a careful tuning may improve in particular the performance of the EKF module, and hence impact on the performance of the overall ITS solution.

The second set of recommendations regards possible improvements to the integration/implementation of the proposed ITS system (i.e. improvements not targeting the mathematical framework but rather the way the different modules are implemented and/or integrated), in particular:

- Introduction of EKF restart strategies. It has been observed experimentally that, while the estimate computed by the EKF remains accurate for a fairly prolonged time interval, the covariance of the EKF estimate rapidly grows, so that in practice the Particle Filter rapidly disregards the information coming from IMU sensors. To remedy to this problem, an EKF restarting procedure has been proposed, such that the covariance of the EKF estimate is periodically restarted to a default value. Other restarting criteria may be considered, for example criteria based on checking the current EKF covariance against a pre-defined restarting covariance threshold. The restarting procedure considered only affects the EKF covariance. Other restarting procedures may be considered to affect also the EKF estimate (the EKF state could be restarted also on the ground of the measurements from the IMU/GPS sensors).

A third set of recommendations regards possible modifications of the mathematical framework itself. In particular,

- It is expected that letting the EKF directly process also positioning measurements (e.g. GPS and/or odometry data) would improve results, especially helping to keep small the covariance associated to the final position and attitude estimate.
- In literature there exist interesting applications of the Kalman filter to the integrated navigation systems. As an example, in [5] Kalman filter (KF), extended Kalman filter (EKF), unscented Kalman filter (UKF) and Adaptive unscented Kalman filter (AUKF) have been evaluated showing that, in vehicle integrated navigation system, UKF is superior to the EKF and AUKF is better than UKF. In particular, AUKF is better in reducing sensitivity of the process and the initial value of the statistical characteristics of the noise and the accuracy, reliability of the navigation solution. The authors recommend to extend the developed MATLAB library with other state of the art algorithms. This will allow the library end users to test and compare different solution and to choose the best for the given scenario.
- Compare resilience of this IMU EKF/PF based positioning system with other different models proposing a Data Fusion (DF) approach.
- Another modification to the mathematical formulation could consist in utilizing the vehicle velocity and orientation estimation to refine the process of particles generation in the Particle Filter module. In particular, the above information could be used to define a proper region in the space from which generating the particles used by the Particle Filter to estimate the position.

Finally, another recommendation is related to a potentially interesting field of future works, which regards the possibility of exploiting the IMU and GPS sensors that are embedded in the latest smartphones or portable devices. In this case there are some additional issues related to the low quality of the sensors, and the additional delay due to the telecommunication network (Bluetooth, wifi, 3G/4G). It could be also interesting to study the integration of these data with other data sources available from smartphones regarding vehicle positioning like, for example, road side units, Wi-Fi spots, 3G/4G base stations, etc.

11 Bibliography

[1] G. M. Vitetta. In-Car Navigation Based on Integrated GPS/IMU Technologies. November 2014.

[2] Garcia, Damien. "Robust smoothing of gridded data in one and higher dimensions with missing values." Computational Statistics & Data Analysis 54.4 (2010): 1167-1178.

[3] ACME Systems. The DAISY-7 - GPS/MEMS module. Website: http://www.acmesystems.it/DAISY-7.

[4] Garcia, Damien. EVAR MATLAB library to estimate the noise variance from 1-D to N-D data. Website: http://www.mathworks.com/matlabcentral/fileexchange/25645-noise-variance-estimation/content/evar.m.

[5] Wu Xiao-yan ; Sch. of Autom., Beijing Inst. of Technol., Beijing, China ; Song Chun-lei ; Chen Jia-bin ; Han Yong-qiang, "An adaptive UKF algorithm and its application for vehicle integrated navigation system", Published in Control Conference (CCC), July 2014 33rd Chinese, Doi: 10.1109/ChiCC.2014.6896727.

Europe Direct is a service to help you find answers to your questions about the European Union.

Freephone number (*)**:**

00 800 6 7 8 9 10 11

(*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

More information on the European Union is available on the internet (http://europa.eu).

HOW TO OBTAIN EU PUBLICATIONS

Free publications:

- one copy:
  via EU Bookshop (http://bookshop.europa.eu);
- more than one copy or posters/maps:
  from the European Union's representations (http://ec.europa.eu/represent_en.htm);
  from the delegations in non-EU countries (http://eeas.europa.eu/delegations/index_en.htm);
  by contacting the Europe Direct service (http://europa.eu/europedirect/index_en.htm) or
  calling 00 800 6 7 8 9 10 11 (freephone number from anywhere in the EU) (*).

  (*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

Priced publications:

- via EU Bookshop (http://bookshop.europa.eu).

KJ-02-17-390-EN-N

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub
ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

Joint Research Centre

EU Science Hub

Publications Office

doi:10.2760/027883

ISBN 978-92-79-67955-1