Analysis and development of algorithms for data fusion in sensor arrays

Josué Manuel Rivera Velázquez

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Microélectronique

École doctorale I2S

Unité de recherche LIRMM

Analysis and development of algorithms for data fusion in sensor arrays

Présentée par Josué Manuel RIVERA VELÁZQUEZ

Le 17 juillet 2020

Sous la direction de Pascal NOUET

Devant le jury composé de

Jean-Yves FOURNIOLS, Professeur, LAAS-CNRS	Rapporteur
Jérôme JUILLARD, Professeur, Centrale-Supélec	Rapporteur
Suzanne LESECQ, Directeur de Recherche, CEA	Examinatrice
Laurent LATORRE, Professeur, Université de Montpellier	Président du jury
Frédéric MAILLY, Maître de Conférences, Université de Montpellier	Encadrant
Pascal NOUET, Professeur, Université de Montpellier	Directeur de thèse

UNIVERSITÉ DE MONTPELLIER

Analysis and development of algorithms for data fusion in sensor arrays

Author: Josué Manuel Rivera Velázquez

josue.rivera-velazquez@lirmm.fr

Publications resulting from this thesis:

"A generic model for sensor simulation at system level"
DOI: 10.1109/DTIP.2018.8394198

"System-level simulations of multi-sensor systems and data fusion algorithms"
DOI: 10.1007/s00542-018-4204-8

"Dynamic weighted average in multisensory systems"
DOI: 10.1109/DTIP.2019.8752804

This research work was supported by CONACYT, Mexico (Unique Curriculum Vitae (acronym in Spanish CVU) – 420024) in collaboration with the University of Montpellier and the LIRMM-CNRS, France.

Acknowledgments

My deep gratitude goes first to my advisor Pascal Nouet and my co-advisor Frédérick Mailly, who expertly guided me through these four years of research work. It was a great honor to work under their supervision. Moreover, their generosity and kindness helped to make my time at LIRMM enjoyable.

My appreciation also extends to my laboratory colleges and outside friends, whose support and friendship made my stay in Montpellier unforgettable.

I am indebted to my family, whose value to me grows with age. Finally, I acknowledge my wife and my daughter. They are my home.

Résumé

Actuellement, la plupart des capteurs sont de nature " intelligente ", ce qui signifie que les éléments de détection et l'électronique associée sont intégrés sur le même circuit. Parmi ces capteurs de nouvelle génération les systèmes micro-électro-mécaniques (MEMS) utilisent les technologies microélectroniques pour la fabrication par lots de capteurs à des volumes sans précédent et à des prix bas. Si ces composants sur étagère sont satisfaisants pour de nombreuses applications nécessitant un niveau de précision faible à moyen, ils ne peuvent toujours pas répondre pleinement aux besoins de performances de nombreuses applications de haute précision.

Cependant, en raison de leur prix décroissant, de leur faible encombrement et de leur faible consommation d'énergie, il est désormais possible de mettre en œuvre des systèmes avec des dizaines ou même des centaines de capteurs. Ces systèmes amènent une solution possible au manque de performances des capteurs individuels et peuvent en outre améliorer la fiabilité et la robustesse de la détection. Les matrices de capteurs sont l'une de ces méthodes de mesures redondantes qui surviennent en réponse aux problèmes susmentionnés. Le développement d'algorithmes de fusion de données pour ces systèmes est un sujet de recherche fréquemment étudié dans la littérature. Néanmoins, il reste encore beaucoup de recherches à faire dans ce domaine de plus en plus important. L'émergence de nouvelles applications aux besoins de plus en plus complexes accroît la nécessité de nouveaux algorithmes avec des propriétés telles que la facilité d'intégration, l'adaptabilité, la robustesse, le faible coût de calcul et la généricité, entre autres.

Dans cette thèse, nous présentons un nouvel algorithme pour les systèmes multi-capteurs qui propose une solution viable pour surmonter les contraintes mentionnées précédemment. La proposition est une méthode on-line basée sur une estimation quadratique sans biais de norme minimale (acronyme en Anglais: MINQUE) qui est capable de calculer les variances des capteurs sans connaître les entrées. Cet algorithme est capable de suivre les changements de variances des capteurs causés principalement par les effets du bruit basse fréquence, ainsi que de détecter et de signaler les capteurs affectés par des erreurs permanent ou transitoires. Cette approche est générique, ce qui signifie qu'elle peut être mise en œuvre pour différents types de systèmes de capteurs. De même, cet algorithme peut être implémenté dans des systèmes de réseaux de capteurs.

Deux autres contributions de cette thèse peuvent être répertoriées. La première est un modèle de capteur générique pour les simulations de capteurs au niveau système. Cet outil créé dans l'environnement Matlab Simulink permet l'analyse des implémentations d'algorithmes de fusion de données dans des systèmes multi-capteurs. Contrairement aux modèles existant auparavant dans la littérature, ce modèle présente des caractéristiques telles que la généricité et l'inclusion de bruits

basse fréquence, ainsi que le paramétrage à travers des graphiques d'analyse spectrale (graphique de Densité Spectrale de Puissance) et des graphiques d'analyse de stabilité dans le temps (graphique de l'écart Allan). La seconde est une étude visant à comparer les performances et la faisabilité de la mise en œuvre de différents algorithmes de fusion de données dans les systèmes multi-capteurs. Cette étude contient une analyse de la complexité de calcul, de la mémoire requise et de l'erreur d'estimation. Les algorithmes analysés sont : algorithme d'étalonnage aveugle, la méthode des moindres carrés, le réseau de neurones artificiels, le filtre de Kalman et la pondération aléatoire.

Aperçu

De nombreux appareils électroménagers tels que les machines à laver, les cafetières, les détecteurs de fumée, les montres connectées, les téléphones portables et les ordinateurs portables remplissent d'innombrables fonctions utiles. Cependant, aucun appareil électronique ne fonctionne sans recevoir d'informations externes. Même si ces informations proviennent d'un autre appareil électronique, quelque part dans la chaîne, il y a au moins un composant sensible aux signaux d'entrée externes. Ce composant est un capteur.

Dans le cadre de ce travail, un capteur est un dispositif qui délivre un signal électrique, la sortie du capteur, dont la valeur dépend de la valeur d'une grandeur physique à observer, l'entrée du capteur. Pour illustrer, section tirée de [1]: *"une thermopile produit une tension positive lorsque l'objet est plus chaud que le capteur ; la tension devient négative lorsque l'objet est plus froid que le capteur, et lorsque les deux sont à la même température, la tension de sortie est nulle"*. Le signal de sortie du capteur peut être affiché, enregistré ou utilisé comme signal d'entrée pour un dispositif ou un système secondaire. Dans un capteur basique, le signal est transmis à un dispositif d'affichage ou d'enregistrement ou la mesure peut être lue par un observateur humain. Le signal peut également être utilisé directement par un système plus important dont le capteur fait partie.

Les capteurs ont toujours été un élément clé des systèmes électroniques ou mécaniques, et pour cette raison, un vaste travail de recherche a été effectué dans ce domaine. Il est intéressant de voir dans une perspective historique comment les capteurs ont ensuite évolué [2]. Des premiers appareils sans ou avec peu d'électronique embarquée aux capteurs intelligents dotés de nombreuses fonctionnalités de sécurité et de communication, y compris des capteurs connectés sans fil pour l'Internet des objets (acronyme en anglais: IOT).

Avec les progrès des technologies silicium, la technologie des capteurs a progressé à une vitesse rapide. Actuellement, la plupart des capteurs sont de nature "intelligente", ce qui signifie que les éléments de détection et l'électronique associée sont intégrés sur la même puce [3]. Les capteurs intelligents présentent les principaux avantages suivants : conditionnement rapide du signal, autotest, faible consommation d'énergie, petite taille physique et, surtout, prix très bas par rapport à leur homologue macroscopique [2]. Les plus grands exemples de cette nouvelle génération de capteurs intelligents sont les dispositifs basés sur la technologie MEMS.

L'acronyme MEMS signifie microsystème électromécanique, et il est désignent généralement des dispositifs à micro-échelle ou des systèmes embarqués miniatures (échelle de longueur caractéristique comprise entre 1mm et 1µm [3]).

Tiré de [4]: *"Les MEMS ont été identifiés comme l'une des technologies les plus prometteuses de ce siècle en raison de leur potentiel à rendre abordables des dispositifs à haute performance et à fonctionnalité améliorée"*. Cette industrie est déjà évaluée à des centaines de milliards de dollars, et selon les prévisions, il continuera d'augmenter au cours des années suivantes. Plusieurs sociétés renommées sont impliquées dans la production de tels appareils : accéléromètres pour les automo-

biles (Analog Devices, Motorola, Bosch), des micro-miroirs pour les appareils de vidéo-projection (Texas Instruments), et des capteurs de pression pour les industries automobile et médicale (NovaSensor).

En 2019, la production de MEMS et ses revenus étaient respectivement de 100 milliards d'unités et de 60 milliards de dollars ; 185 milliards d'unités et plus de 100 milliards de dollars de revenus sont prévus d'ici 2023 [5]. Ce marché présente des revenus élevés même si les prix de vente moyens des capteurs MEMS sont inférieurs à 1 dollar par unité depuis 2013. La Figure 1 montre le graphique relatif à ces chiffres rapportés et prévus, où le terme ASP est l'acronyme de Average Selling Price (prix de vente moyen). Cette figure montre clairement la tendance à la production en masse de capteurs dont les caractéristiques sont le faible coût, la faible consommation d'énergie et la petite taille.

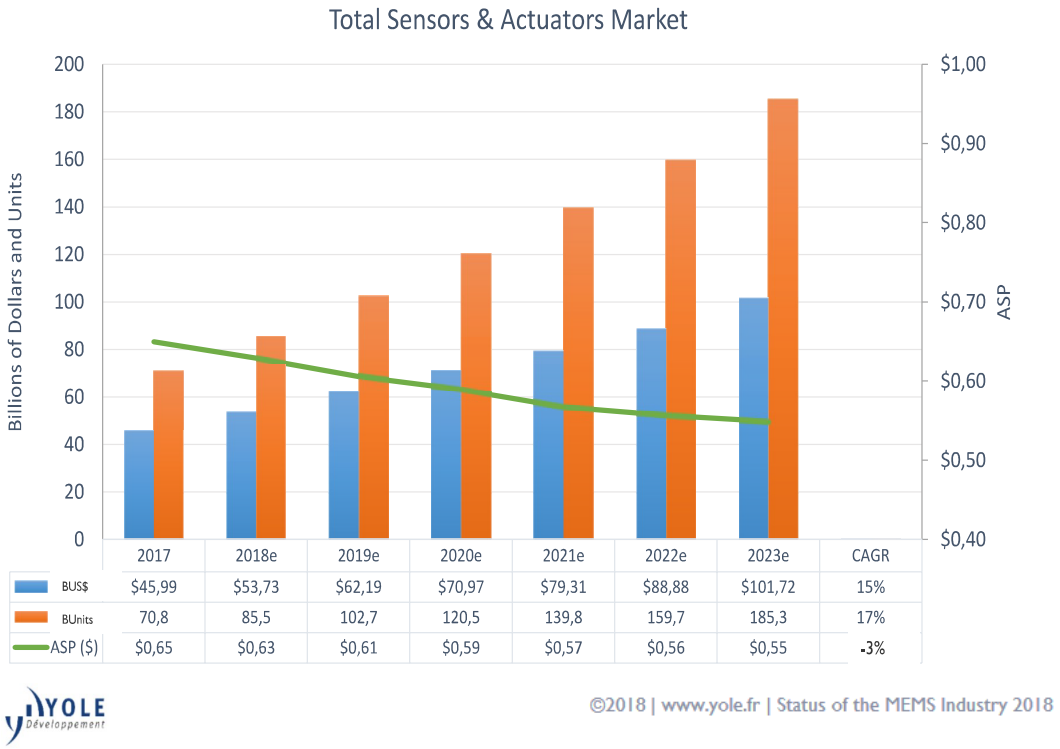


Figure 1 – Situation de l'industrie MEMS [5].

Si ces composants sur étagère sont satisfaisants pour de nombreuses applications nécessitant un niveau de précision faible à moyen, ils ne peuvent toujours pas répondre pleinement aux besoins de performances de nombreuses applications de haute précision ; principalement en raison de la présence d'erreurs déterministes et stochastiques, telles que les biais, les instabilités thermiques et les non-linéarités [6]. Cependant, en raison de leur prix décroissant, de leur faible encombrement et de leur faible consommation d'énergie, il est désormais possible de mettre en œuvre des systèmes comportant des dizaines, voire des centaines de capteurs, proposant ainsi une solution possible à leur manque de performance. L'idée d'exploiter des données redondantes apparaît dans de nombreuses applications différentes où des sous-matrices capturent la géométrie du problème, allant du traitement des réseaux, de la segmentation du mouvement, et des communications sans

fil à entrées multiples et sorties multiples (MIMO) [7].

C'est dans ce domaine de recherche de données redondantes que les matrices de capteurs se présentent comme une proposition intéressante au problème du manque de robustesse et de précision des systèmes d'acquisition de données utilisant la technologie MEMS. Une matrice de capteurs peut être définie comme une collection de capteurs, généralement déployés selon un certain modèle géométrique et utilisés pour mesurer la même entrée physique. Une fois les mesures obtenues, une méthode de fusion de données est appliquée pour tirer parti de la redondance des mesures. Ces méthodes permettent de traiter les données de plusieurs capteurs, ce qui présente des avantages tels que l'amélioration de la précision et de la fiabilité, ainsi que la détection des défauts [8].

Le développement d'algorithmes de fusion de données pour ces systèmes est un sujet de recherche fréquemment étudié dans la littérature [9], où différentes méthodes proposées ont été appliquées à divers types de systèmes tels que les réseaux d'antennes [10], les matrices de capteurs magnétiques [11], les matrices de capteurs acoustiques [12] et les matrices de capteurs chimiques [13].

Malgré cela, il reste encore beaucoup de travail de recherche à faire dans ce domaine de plus en plus important. L'émergence de nouvelles applications aux besoins de plus en plus complexes accroît la nécessité de nouveaux algorithmes présentant des caractéristiques telles que la facilité d'intégration, l'adaptabilité, la robustesse, le faible coût de calcul et la généricité, entre autres. Satisfaire ces demandes est une priorité, car la popularisation des dispositifs basés sur les MEMS dans les applications hautes performances en dépend.

Description du problème

L'étude de l'état de l'art nous apprend que la plupart des solutions existantes pour la fusion des mesures dans les matrices de capteurs présentent les restrictions suivantes :

i) manque de généricité (solutions développées pour des systèmes spécifiques)

ii) une grande complexité de calcul des algorithmes, même pour les systèmes temps réel

iii) absence de réaction contre la présence de capteurs défectueux

iv) absence de comportement adaptatif pour l'ajout ou le retrait de capteurs dans le système

v) pas de traitement du bruit basse fréquence

Il est facile de noter que ces contraintes limitent l'utilisation des matrices de capteurs dans un grand nombre d'applications.

Par exemple, le manque de généricité ne permet pas l'intégration de deux systèmes différents même si les deux comprennent des capteurs du même type. Une grande complexité de calcul des algorithmes, telle que celle des algorithmes d'apprentissage machine, limite leur mise en œuvre dans les systèmes embarqués, où des algorithmes complexes diminuent la durée de vie de la batterie en raison de la consommation d'énergie. Un comportement adaptatif pour l'ajout et le retrait de capteurs dans les systèmes est idéal pour les applications de pointe d'un grand intérêt dans le monde scientifique, telles que les réseaux de capteurs. Enfin, le traitement des bruits basse fréquence permet des applications robustes de longue durée. Cette caractéristique est rarement prise en compte par les algorithmes actuels.

En surmontant ces restrictions, une interopérabilité à grande échelle entre les capteurs pourrait être réalisée, rendant possible des applications plus sophistiquées.

Plan de la thèse et contributions

L'objectif général de cette thèse est d'introduire un nouvel algorithme pour les systèmes multi-capteurs qui propose une solution viable pour surmonter les contraintes mentionnées ci-dessus. En conséquence de ce travail de recherche, différents résultats intéressants ont été obtenus. Nous présentons ci-dessous les principales contributions développées dans le cadre de ce travail :

- Un modèle de capteur générique pour les simulations de capteurs au niveau du système. Cet outil développé dans Matlab Simulink permet l'analyse des implémentations d'algorithmes de fusion de données dans des systèmes multi-capteurs. Contrairement aux modèles existant précédemment dans la littérature, ce modèle de capteur présente des caractéristiques telles que la généricité (il peut être utilisé pour simuler différents types de capteurs), l'évolutivité (il peut être utilisé pour simuler des systèmes avec un grand nombre de capteurs), la portabilité (du fait qu'il est réalisé dans l'environnement Matlab Simulink, il peut être directement traduit dans d'autres langages de programmation), et le paramétrage à travers des graphiques d'analyse spectrale (graphique de Densité Spectrale de Puissance) ou de stabilité dans le temps (graphique de variance d'Allan).

- Une étude visant à comparer les performances et la faisabilité de la mise en œuvre de différents algorithmes pour la fusion de données dans les systèmes multi-capteurs. Cette étude contient une analyse de la complexité de calcul, de la mémoire requise et de l'erreur d'estimation, ce qui permet une comparaison équitable entre les algorithmes. Les algorithmes à analyser sont les suivants : méthode des moindres carrés, réseaux neuronaux artificiels, filtre de Kalman, et pondération aléatoire.

- Le développement d'un nouvel algorithme adaptatif pour les systèmes multi-capteurs. La proposition est une méthode en ligne basée sur l'estimation quadratique minimale sans biais (MINQUE) qui est capable de calculer les variances des capteurs sans connaître les entrées. Cet algorithme est capable de suivre les changements dans les variances des capteurs causés principalement par les effets du bruit basse fréquence, ainsi que de détecter et de signaler les capteurs affectés par des échecs / erreurs d'étalonnage. Cette approche est générique, ce qui signifie qu'elle peut être mise en œuvre pour différents types de systèmes multi-capteurs. De même, cet algorithme peut être implémenté dans des systèmes de réseaux de capteurs.

Description du reste du document

Cette thèse est composée de six chapitres organisés comme suit :

- Le premier chapitre présente le cadre théorique nécessaire à la compréhension du travail développé ici. Nous commençons par la definition d'un capteur et nous présentons quelques exemples de classification de capteurs selon différentes propriétés. Ensuite, d'autres aspects liés aux capteurs tels que les erreurs déterministes, les bruits stochastiques et la caractérisation du bruit sont développés. Enfin, les concepts de mesures redondantes et de fusion de données sont introduits à la fin de ce chapitre.

- Dans le deuxième chapitre, un modèle de capteur générique pour les simulations au niveau système est présenté. Ici, une description complète du modèle mathématique utilisé pour

décrire le comportement générique des capteurs est donnée, ainsi que l'explication complète de chaque bloc qui compose l'outil de simulation développé dans Matlab Simulink. Quelques exemples d'implémentation de cet outil sont présentés à la fin de ce chapitre, où sa généricité et son utilité sont démontrées.

- Au cours du troisième chapitre, nous présentons une étude de faisabilité de l'implémentation de différents algorithmes dans des systèmes multi-capteurs. Ces algorithmes sont comparés en termes de complexité de calcul, de ressources mémoire utilisées et d'erreur d'estimation. Les algorithmes pris en compte dans cette étude sont les suivants : étalonnage aveugle, régression des moindres carrés ordinaires, réseau de neurones perceptron multicouche, filtre de Kalman, et moyenne de pondération aléatoire. Cette analyse est réalisée en mettant en œuvre le modèle de capteur générique du chapitre deux.

- Le quatrième chapitre est consacré à la présentation de notre proposition d'algorithme pour la fusion de données dans les systèmes multi-capteurs. Ce chapitre commence par l'état de l'art des algorithmes de fusion de données pour les systèmes multi-capteurs. Après cela, le support théorique de cet algorithme est présenté.

- Dans le cinquième chapitre, nous présentons l'évaluation de l'algorithme proposé à travers des simulations, ainsi que sa mise en œuvre dans un véritable système matriciel composé de 12 accéléromètres MEMS. Ici, les performances et l'exactitude de notre algorithme sont testées dans des scénarios réels.

- Enfin, dans le sixième chapitre, nous présentons les commentaires finaux, où nous résumons les résultats obtenus durant ce travail de recherche. Quelques propositions générales pour de futures implémentations sont également mentionnées dans ce chapitre.

Le code Matlab généré lors du développement de ce travail de recherche est inclus en Annexe.

Abstract

Currently, most of the sensors are "smart" in nature, which means that sensing elements and associated electronics are integrated on the same chip. Among these new generation of sensors, the Micro-Electro-Mechanical-Systems (MEMS) make use of Microelectronics technologies for batch manufacturing of small footprint sensors to unprecedented volumes and at low prices. If those components of the shelf are satisfactory for many consumer and low- to medium-end applications, they still cannot fully meet the performance needs of many high-end applications.

However, due to their decreasing price, their small footprint, and their low-power consumption, it is now feasible to implement systems with tens and even hundreds of sensors. Those systems give a possible solution to the lack of performance of individual sensors and additionally they can also improve dependability and robustness of sensing. Sensor array systems are one of these methods of redundant measurements that arise in response to the aforementioned problems. The development of data fusion algorithms for sensor array systems is a research topic frequently studied in the literature. Even so, it still remains a lot of research work to do in this increasingly important area. The emergence of new applications with increasingly complex needs is growing the requirement for new algorithms with features such as integration, adaptability, dependability, low computational cost, and genericity among others.

In this thesis we present a new algorithm for sensor array systems that propose a viable solution to overcome constraints mentioned before. The proposal is an on-line method based on the MInimum Norm Quadratic Unbiased Estimation (MINQUE) that is able to compute sensors' variances without the knowledge of the inputs. This algorithm is capable to track changes in sensors' variances caused principally by the low-frequency noise effects, as well as to detect and point out sensors affected by permanent or transitory errors. This approach is generic, which means that it can be implemented for different types of sensor array systems. In addition, this algorithm can be also implemented in sensor network systems.

Two more contributions of this thesis can be listed. The first is a generic sensor model for sensor simulations at system level. This tool created inside the Matlab Simulink environment permits the analysis of implementations of data fusion algorithms in multi-sensor systems. Unlike the models previously existing in the literature, this sensor model has characteristics such as genericity and inclusion of low-frequency noises. The second is a study to compare the performance and feasibility in the implementation of different algorithms for data fusion in sensor array systems. This study contains an analysis of computational complexity, memory required, and the error in estimation. The analyzed algorithms are : blind calibration algorithm, method of least squares, an artificial neural network, Kalman filter, and Random weighting.

Overview

Many household appliances such as washer machines, coffee makers, smoke detectors, smart-watches, cell phones, and laptops and perform endless useful functions. However, no electronic device operates without receiving external information. Even if such information comes from another electronic device, somewhere in the chain, there is at least one component sensitive to external input signals. This component is a sensor.

In the context of this work, a sensor is a device that delivers an electrical signal, the sensor output, which value depends on the magnitude of a physical quantity to be observed, the sensor input. To illustrate, from [1]: *"a thermopile will produce a positive voltage when the object is warmer than the sensor; voltage becomes negative when the object is cooler than the sensor, and when both are at exactly the same temperature the output voltage will be zero"*. Sensor's output can be displayed, recorded, or used as an input signal in other devices or systems. For example, the reported output can be transmitted to a display or recording device where the measurement can be read by an observer, or it can be used directly by some larger system of which the sensor is a part.

Sensors have always been a key component in electronic or mechanical systems, and because of this, wide research work has been done in this area. It is interesting to see in historical perspective how sensors have subsequently developed [2]. From the first devices with none or few embedded electronics up to smart-sensors with a lot of functionalities for safety and communication including wireless connected sensors for the Internet Of Things (IOT).

With the advancement in silicon technology, sensors technology has progressed at a rapid speed. Currently, most of the sensors are "smart" in nature, which means that sensing elements and associated electronics are integrated on the same chip [3]. Smart sensors have the main advantages of fast signal conditioning, self-testing, low-power consumption, small physical size, and above all, very low-price compared with their macroscopic counterparts [2]. The biggest example of this new generation of smart sensors are the devices based on MEMS technology.

The acronym MEMS stands for Micro-Electro-Mechanical System, and its generally used to refer to micro-scale devices or miniature embedded systems with a characteristic length scale between 1mm and 1µm [3].

From [4]: *"MEMS has been identified as one of the most promising technologies of this century because of its potential for making affordable enhanced-functionality high-performance devices"*. This industry is already valued in hundreds of billions of dollars, and according to the forecasts, it will continue to increase in the following years. Several well-known companies are involved in the production of such devices: accelerometers for automobiles (Analog Devices, Motorola, Bosch), micro-mirrors for digital projection displays (Texas Instruments), and pressure sensors for the automotive and medical industries (NovaSensor).

In 2019, production of MEMS and its incomes were about 100 billion units and $60 billion dollars, respectively. Forecasts of 185 billion units and more than $100 billion in revenues are

reported to 2023 [5]. This market presents high revenues even when the average selling prices of based-MEMS sensors are under $1 dollar per unit since 2013. Figure 2 shows the graph related to these reported and predicted numbers, where term ASP is the acronym of Average Selling Price. This figure clearly exemplifies the trend towards mass production of sensors whose characteristics are low cost, low power consumption, and small size.

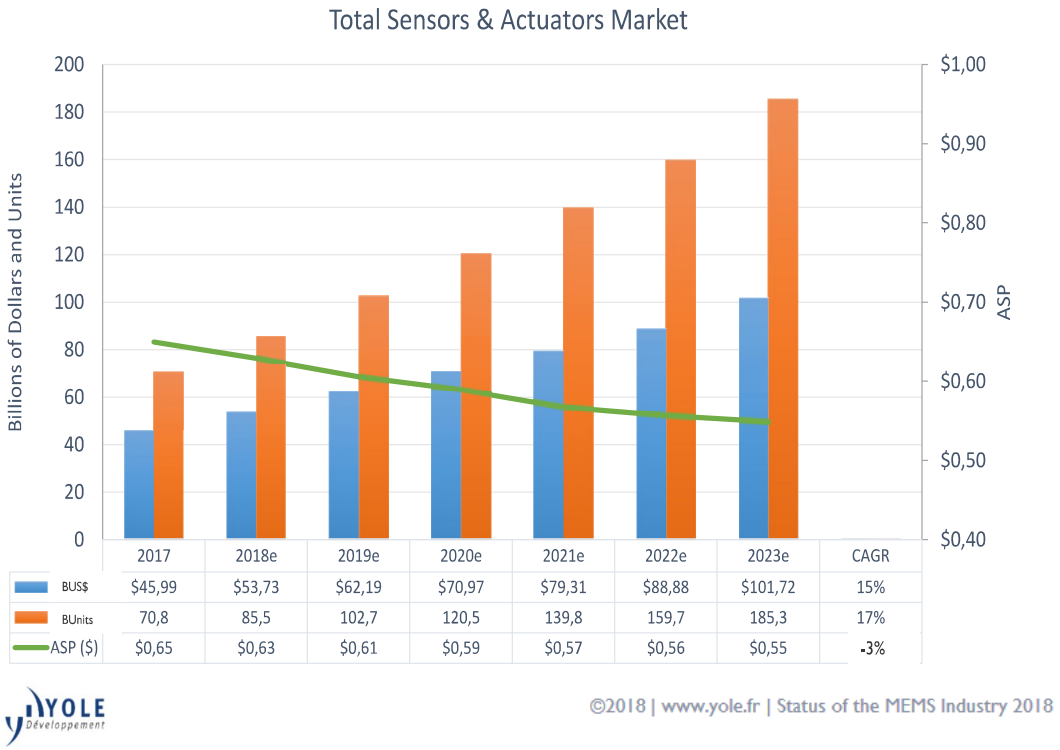


Figure 2 – Status of the MEMS industry. Figure taken from [5].

If those components of the shelf are satisfactory for many consumer and low- to medium-end applications, they still cannot fully meet the performance needs of many high-end applications; primarily due to the presence of deterministic and stochastic errors, such as bias, thermal instabilities and non-linearities [6]. However, due to their decreasing price, their small footprint, and their low-power consumption, it is now feasible to implement systems with tens and even hundreds of sensors, giving a possible solution to their lack of performance. The idea of exploiting redundant data appears in many different applications where low-dimensional subspaces capture the intrinsic geometry of the problem, ranging from array processing, motion segmentation, and Multiple-Input Multiple-Output (MIMO) wireless communications [7].

It is in this research area of redundant data where sensor array systems arise as an interesting proposal to the problem of lack of robustness and accuracy in MEMS-based data acquisition systems. A sensor array system is defined as a collection of sensors, usually deployed in a certain geometry pattern and used to measure the same physical input. Once measurements are obtained, a data fusion method is applied to leverage the redundancy in measurements. These methods permit to process data from several sensors obtaining advantages such as improvement in accuracy and reliability, as well as fault detection [8].

The develop of data fusion algorithms for these systems is a research topic frequently studied in the literature [9], where different proposed methods have been applied to various types of systems such as antenna arrays [10], magnetic sensor arrays [11], acoustic sensor arrays [12], and chemical sensor arrays [13].

Despite that, there is still a lot of research work to do in this increasingly important area. The emergence of new applications with increasingly complex needs is growing the requirement for new algorithms with features such as integration, adaptability, sturdiness, low computational cost, and genericity among others. Satisfying these demands is a priority since the popularization of MEMS-based devices into high-end applications depends on this.

Problem description

From the study of the state of art, we learn that most of the existing solutions for measurement fusion in sensor array systems present the following restrictions:

i) lack of genericity (solutions developed for specific systems)

ii) high computational complexity of the algorithms, even for real-time systems

iii) lack of response against fault-in-sensors presence

iv) absence of adaptive behavior for addition/removal of sensors to the array

v) no low-frequency noise treatment

It is straightforward to note that these constraints limit the application of sensor arrays in a large number of applications.

For example, the lack of genericity can limit the integration of more sensors to the data fusion process even if such sensors measure the same input signal. A high computational complexity of the algorithms, such as those shown by the machine learning algorithms, limit their implementation in embedded systems, where complex algorithms decrease the lifetime of the battery due to the power consumption. An adaptive behavior for addition and removal of sensors in systems is ideal for cutting-edge applications of great interest in the scientific world, such as sensor networks. Besides, treatment of low frequency noises permits long-time robust applications. Such characteristic is rarely taken into account by the current existing algorithms.

By overcoming these restrictions, an interoperability at large scales between sensors could be achieved, making more sophisticated applications possible.

Thesis outline and contributions

The general goal of this thesis is to introduce a new algorithm for systems that propose a viable solution to overcome the constraints mentioned above. As a consequence of this research work, different interesting results were obtained. Next, we present a list of the contributions developed in this work:

- A generic sensor model for sensor simulations at system level. This tool developed in Matlab Simulink permits the analysis of implementations of data fusion algorithms in multi-sensor systems. Unlike the models previously existing in the literature, this sensor model has characteristics such as genericity (it can be used to simulate different types of sensors), scalability

(it can be used to simulate systems with a large number of sensors), portability (due to the fact that it is made in Matlab Simulink environment, it can be straightforwardly translated to others programming languages), and the parameter setting through spectral analysis graphs (Power Spectral Density graph) or stability over time (Allan variance graph).

- A study to compare the performance and feasibility in the implementation of different algorithms for data fusion in sensor array systems. This study contains an analysis of computational complexity, memory required, and the error in estimation; which permits a fair comparison between the algorithms. The algorithms to analyze are : blind calibration algorithm, method of least squares, an artificial neural network, Kalman filter, and Random weighting.

- The development of a new adaptive algorithm for sensor array systems. The proposal is an on-line method based on the MInimum Norm Quadratic Unbiased Estimation (MINQUE) that is able to compute sensors' variances without the knowledge of the inputs. This algorithm is capable to track changes in sensors' variances caused principally by the low-frequency noise effects, as well as to detect and point out sensors affected by faults / uncalibrations. This approach is generic, which means that it can be implemented for different types of sensor array systems. Likewise, this algorithm can be implemented in sensor network systems.

Description of the rest of the document

This thesis is composed of six chapters organized as follows:

- The first chapter introduces the theoretical framework needed for the comprehension of the work developed here. We start with the definition of a sensor and we present some examples of classification of sensors according to different attributes. After, other aspects related to sensors such as deterministic errors, stochastic noises, and characterization of noise are developed. Finally, concepts of redundant measurements and data fusion are introduced at the end of this chapter.

- In the second chapter, a generic sensor model for simulations at the system level is presented. Here, a complete description of the mathematical model used for describing the generic behavior of sensors is given, as well as the complete explanation of each block that composes the simulation tool developed in Matlab Simulink. Some examples of implementation of this tool are shown at the end of this chapter, where its genericity and utility are shown.

- In the third chapter, we present a feasibility study of the implementation of different algorithms in sensor array systems. These algorithms are compared in terms of computational complexity, used memory resources, and estimation error. The algorithms considered to this study are: blind calibration algorithm, ordinary least square regression, multi-layer perceptron neural network, Kalman filter, and random weighting average. This analysis is carried out by implementing the generic sensor model described in chapter two.

- The fourth chapter is dedicated to the presentation of our proposed algorithm for data fusion in sensor array systems. This chapter begins with the state of the art in data fusion algorithms for sensor array systems. After that, the theoretical support of this algorithm is presented.

- In the fifth chapter, we present the assessment of the proposed algorithm through simulations, as well as its implementation in a real array system composed of 12 MEMS accelerometers. Here, the performance and correctness of our algorithm are tested under real scenarios.
- Finally, in the sixth chapter, we present the final comments, where we summarize the results obtained through this research work. Some general propositions for future implementations are also mentioned in this chapter.

The Matlab code files generated during the development of this research work are included in Appendix.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and background knowledge

In this first chapter, we introduce the theoretical framework needed for the comprehension of work developed throughout this thesis. First, a brief introduction to the measurement process is given. Then, a general overview regarding sensors is presented, where some of the fundamental terminologies, which are frequently encountered in the sensor field are defined. The importance and applications of sensors are also highlighted. Finally, concepts of redundant measurements and data fusion are introduced at the end of this chapter.

1.1 Measurement process

Measurement is the process by which relevant information about a physical phenomenon of interest is collected. This information may be obtained for purposes of controlling the behavior of the phenomenon (as in engineering applications) or for learning more about it (as in scientific investigations). The collecting data process is carried out through sensors that deliver data related to the physical phenomenon to be observed.

Figure 1.1 illustrates an overview of the measurement process. The physical phenomenon to measure is in the left of the figure, and the measurand (the physical quantity being sensed) is represented by an observable variable x. Note that x needs not necessarily be the measurand but simply related to the measurand in some known way. For example [14]: *"the mass of an object is often measured by the process of weighting, where the measurand is the mass, but the measured physical variable is the downward force the mass exerts in the Earth's gravitational field"*.

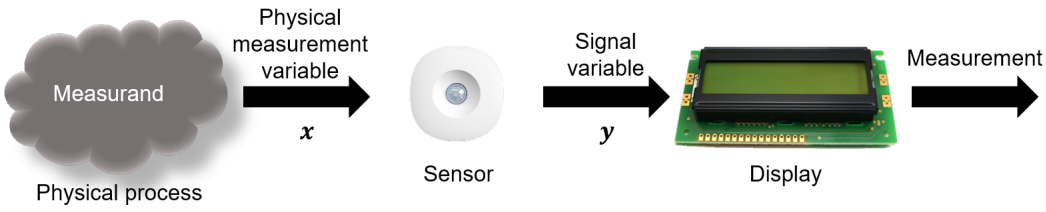


Figure 1.1 – Measurement process. Figure taken from [14].

As shown in Figure 1.1, the key functional element of the measurement process is the sensor.

Next, the definition of a sensor, different classifications according to its attributes, as well as a description of its characteristics are presented.

1.2 Sensors

In the context of this work, a sensor is a device that delivers an electrical signal, the sensor output, which value depends on the magnitude of a physical quantity to be observed, the sensor input. Output is linearly and functionally related to the input stimulus which is generally referred to as measurand. Transducer is the other term that is sometimes interchangeably used instead of the term sensor, although there are subtle differences. The term transducer generally implies a conversion of energy between input and output of the device and can be used for the definition of many devices such as sensors, actuators, or transistors [15].

Another widely used definition of a sensor is depicted in Figure 1.2, where a sensor is commonly made of two major components: a sensitive element and a transducer. The following is taken from [15], which describes the operation of a sensor in the given context: *"the sensitive element interacts with the physical phenomena and cause a change in the transducer. Affected by this change, the transducer produces an output (usually electrical or mechanical), which is translated into readable information by a data acquisition system"*.

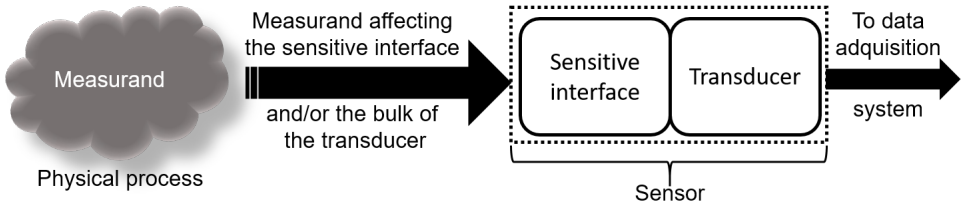


Figure 1.2 – Schematic depiction of a sensing system. Figure taken from [15].

Also, in the literature it is common to find the term sensor used to refer to the sensitive element itself, and the term transducer used to refer to the sensitive element plus any associated peripherals (the overall system). For example [15]: *"a temperature sensor is called a sensor, while together with the data acquisition circuit (to convert the signal into a measurable electrical voltage) is called a transducer"*.

Artificial 'man-made' systems are generally composed with elementary functions such as [15]:

- observation of the surrounding environment as a set of acquired data,
- processing those data to determine actions to be taken,
- act on this surrounding environment

The sensors' role in such systems is the acquisition of data. This data is sent to a processing system, where it is converted into meaningful information. The processed information can be either the desired output or used to feed another system. The more complex the system, the larger number of sensors is required for its operation.

Sensors have always been a key component in electronic or mechanical systems, and because of this, wide research work has been done in this area. It is interesting to see in historical perspective how sensors have subsequently developed [2]. From the first devices with none or few embedded

electronics up to smart-sensors with a lot of functionalities for safety and communication including wireless connected sensors for the Internet Of Things (IOT). With the advancement in silicon technology, instrument technology has progressed at a rapid speed. Nowadays, most of the sensors are "smart" in nature, which means that sensing elements and associated electronics are integrated on the same chip. Smart sensors have main advantages of fast signal conditioning, self testing, auto calibration, small physical size, high reliability, failure prevention and detection [2]. One example of these smart sensors are devices based on Micro-Electro-Mechanical System (MEMS) technology, which will be discussed in more detail later. Next, a study on the classification of sensors according to different characteristics is presented.

1.2.1 Classification

Inside the literature, an extensive variety of sensor classifications can be found. For instance, in [1] sensors are classify into actives and passives. Passive sensors do not require any external power source and directly generates an output response, while active sensors require either add or consume energy to carry out the measurement process.

Another classification of sensors [17] divides sensors in absolute and relative sensors. An absolute sensor measures the absolute value of the measurand, e.g. a thermistor allows to know the temperature from the value of the resistance, where a relative sensor measures the difference between the measurand and a reference, e.g. a thermocouple delivers an electric voltage proportional to the temperature difference between the thermocouple wires. Thus, a thermocouple output signal cannot be related to any particular temperature without referencing to a selected baseline.

In [18] sensors are classified according to the nature of their output signal: analog sensors that deliver an electrical output that may vary continuously in magnitude and in time and digital sensors that deliver an electrical output that are discretized in both magnitude and time.

Finally, we mention the classification done in [16], which divides sensors according to its sensing principle. Table 1.1 shows a summary of this classification.

Sensing principle	Examples
Mechanical motion (including mechanical resonance)	Pendulum-clock, quartz clock, spring balance, odometer, piezoresistive pressure sensor, accelerometer, gyrometer
Thermal (including temperature differences)	Thermometer, thermocouple, thermistor, transistor built-in voltage, air flow sensors
Optical energy (photons)	Photodiode, color sensor
Magnetic field	Compass, magnetoresistance, inductive proximity sensor
Electric field	Electrostatic voltmeter, field-effect transistor

Table 1.1 – Classification of sensors according to its sensing principle. Table taken from [16].

Regardless of their type, electronic sensors present similar physical characteristics, which are commonly classified into two groups: static and dynamic [15]. Understanding the dynamic and static characteristics are essential for a good description of the input-output relationship of a sensor. In the following sections, the static and dynamic characteristics will be presented, as well as their impact in sensing systems.

1.2.2 Static characteristics

Static characteristics of a sensor are characteristics that does not depend on time. They are measured in absence of transient variations to link output value to measurand magnitudes. The most important static characteristics are the following [15]:

- Accuracy. It represents the correctness of sensor's output in comparison to the actual value of a measurand. To assess its accuracy, a sensor may be calibrated with respect to a known measurand or a higher accuracy measurement system.
- Precision. In [15], it is defined as: *"the capacity of a sensing system to give the same reading when repetitively measuring the same input under the same conditions"*. The precision of a sensor is usually quantified by means of probabilistic methods (such as the standard deviation), which assess the degree of dispersion in a sensor's outputs given a constant input. Difference between accuracy and precision is illustrated in Figure 1.3.
- Repeatability. From [15]: *"repeatability is the sensing system's ability to produce the same response for successive measurements when all operating and environmental conditions remain constant"*.
- Reproducibility. It is the sensing system's capacity to report the same output under different environmental conditions. For example [15]: *"if a temperature sensing system shows similar responses; over a long time period, or when readings are performed by different operators, or at different laboratories, the system is reproducible"*.
- Stability. It is a sensing system's ability to report the same output under the presence of the same input over a long period of time.
- Sensitivity. In the literature, it is also known as scale factor [19] that relates a difference of output signal to a given difference of the input signal. Sensitivity can be calculated using the slope of the curve $y = f(x)$. An ideal sensor has a constant sensitivity in its operating range. A real sensor may exhibit non-linearities and/or saturation, a state in which it can no longer follow the input.
- Linearity. From [15]: *"the closeness of the calibration curve to a specified straight line shows the linearity of a sensor. Its degree of resemblance to a straight line describes how linear a sensor is"*.
- Hysteresis. Figure 1.4 represents the relation between output and input of a system with hysteresis. As it can be seen, depending on whether path 1 or 2 is taken, two different output values may be obtained for the same input magnitude.
- Measurement Range. It is the maximum and minimum values of the input that can be measured with the sensing system. All sensing systems are designed to perform over a specified range, values outside of this range cannot be measured by the system.
- Error. It represents the difference between the actual value of the input and the value reported by the sensing system. Error can be caused by a variety of internal and external sources. Absolute and relative error can be computed as follows [15]:

$$\text{Absolute error} = \text{Output - True value} \tag{1.1}$$

$$\text{Relative error} = \frac{\text{Output - True value}}{\text{True value}} \tag{1.2}$$

While the absolute error has the same unit as the measured input, the relative error is unitless. Throughout this thesis, absolute error will be used for assessing the accuracy of a produced estimation.

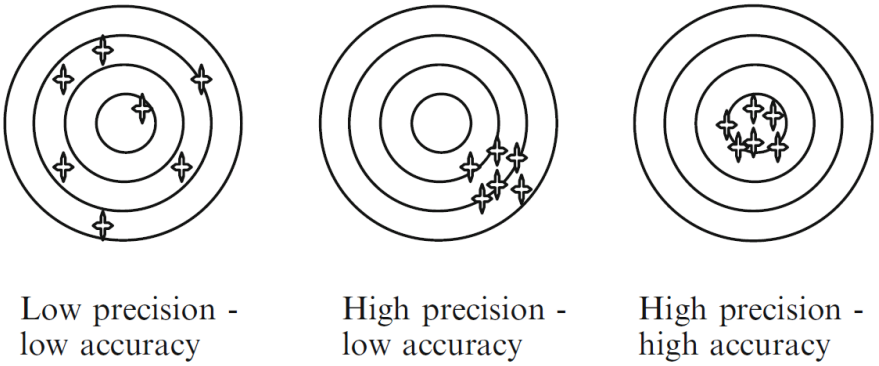


Figure 1.3 – Difference between accuracy and precision. Figure taken from [15].

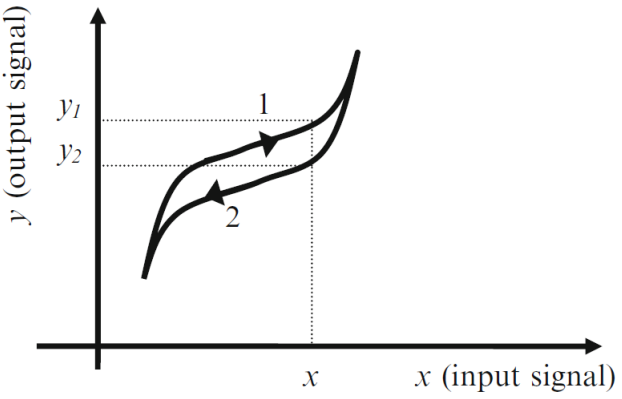


Figure 1.4 – Example of hysteresis curve. Figure taken from [15].

1.2.3 Dynamic characteristics

Dynamic characteristics of a sensor are used to define how the output will follow measurand changes with time [17]. The reason for the presence of dynamic characteristics is the existence of energy-storing elements in a sensing system. These elements can be electrical (capacitance and inductance), mechanical (spring and mass) or thermal (heat capacity). The most common method of assessing the dynamic characteristics is by defining a system's mathematical model and deriving the relationship between the input and output signal. Consequently, such a model can be used for

analyzing the response to variable input waveforms such as impulse, step, ramp, sinusoidal, and white noise signals among others.

Linear Time Invariant (LTI) systems are generally assumed [17]. In that systems, three main assumptions can be done:

- System properties and/or parameters are not changing over time,
- Superposition theorem applies, i.e. two different inputs simultaneously applied to the system leads to an output equal to the sum of outputs obtained for each individual input,
- Linear scaling is obtained, i.e. when input is increased, the output is increased linearly.

The relationship between the input and output of any LTI sensing system can be described as [20]:

$$a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_m \frac{d^m x(t)}{dt^m} + b_{m-1} \frac{d^{m-1} x(t)}{dt^{m-1}} + \cdots + b_1 \frac{dx(t)}{dt} + b_0 x(t) \quad (1.3)$$

Where $x(t)$ is the measured input, $y(t)$ is the reported output, and $a_0, \ldots, a_n, b_0, \ldots, b_m$ are constants defined by the system's parameters.

The two most common dynamic models used in sensing systems are the zero-order and first-order systems [20], which are briefly explained below.

- Zero-order systems. A system is zero-order if its output shows no delay response with respect to the input signal [15]. In such case, all a_i and b_i coefficients are zero, excepting a_0 and b_0. Equation (1.3) can then be simplified to:

  $$a_0 y(t) = b_0 x(t) \qquad \text{or simply:} \qquad y(t) = K x(t) \quad (1.4)$$

  where $K = b_0/a_0$ is defined as the static sensitivity for a linear system (also called the scale factor).

- First-order systems. A system is first-order if its output approaches its final value gradually. A first-order system is mathematically described as:

  $$a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_0 x(t) \quad (1.5)$$

  or after rearranging:

  $$\tau \frac{dy(t)}{dt} + y(t) = K x(t) \quad (1.6)$$

  where $\tau = a_1/a_0$ is defined as the time constant. Assuming a step input of the measurand ($x(0) = 0$, and $x(t) = 1\ \forall t > 0$), $y(t)$ will reach a steady-state of K at an exponential rate. τ is the time required for the output to reach approximately 63% of the final steady-state [$(1 - 1/e^{-1})$=0.6321]. This is illustrated in Figure 1.5.

Throughout this work, sensors' dynamics are assumed to be zero-order and first-order LTI systems. However, this assumption is not restrictive, that is, the analysis carried out here can be applied to sensors whose behavior is described by a greater order system.

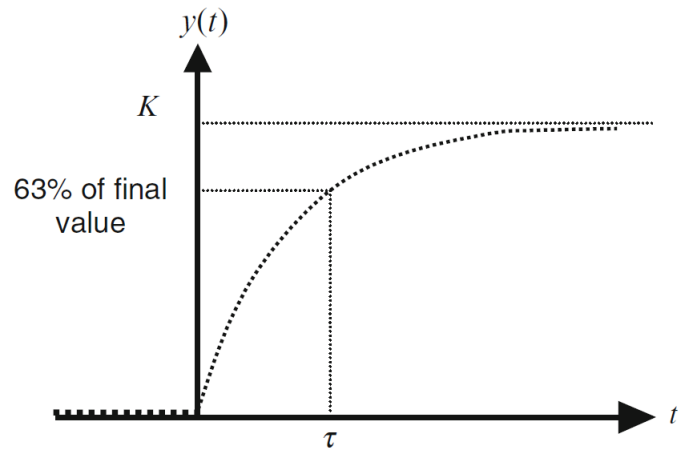


Figure 1.5 – Response of a first order system to a step function. Figure taken from [15].

1.3 Errors in sensors measurements

Sensor measurements always have a certain degree of uncertainty, which means they can only give an estimate of the measured physical property. This uncertainty is usually classified according to nature and/or the cause of error in the input estimation. About this, the following classification is taken from [17]:

- Systematic Errors. These errors are constant and repeatable. There are many different types of systematic errors:
  - Calibration errors. These are a result of an error in the calibration process, and they are often due to linearization of the calibration for devices exhibiting non-linear characteristics.
  - Environmental errors. These arise from the measurement device being affected by environmental factors which are not taken into account.
  - Common representation format errors. These occur when we transform from the original measurement space to a common representational format.
- Stochastic Errors. These errors are characterized by a lack of repeatability.
- Spurious Readings. These errors are infrequent, however when they appear they considerably affect the input estimate.

Next, we present a detailed study on systematic and stochastic errors, which are a determining factor inside the topic of sensors.

1.3.1 Systematic errors

A systematic or deterministic error can be defined as an error which is reproducible under the same input and environmental conditions [15]. The following list of the most common systematic errors affecting sensors is taken from [19]:

- Bias. It is also known as short-term deterministic offset. It is the sensor output observed in the absence of an applied physical input.
- Scale factor error. It is the ratio of the output error (deviation from the fitted straight line slope) over the input, and is typically expressed as a percentage or ppm (parts per million).
- Linearity error, also known as non-linearity. It characterizes the difference between the straight line that relates output to input using the scale factor with the actual output. The linearity error is normally specified as a percentage of the full-scale.
- Cross-coupling error, or cross-sensitivity errors. It characterizes undesired variations of the output to other physical magnitudes, e.g. sensitivity of an x-axis accelerometer to an y-axis acceleration due to misalignments between axes with respect to the sensor case frame.

1.3.2 Stochastic errors

A stochastic error can be defined as a random phenomenon that alters the sensor's output. Usually inside the literature, stochastic errors affecting sensors' outputs are referred as noise, due to noise is a random signal that carries no useful information [1]. Different types of stochastic errors are classified according to their characteristics, such as their nature, their origin or their spectral behavior [21]. Next, we present a concise explanation of stochastic errors with respect to their spectral behavior.

1. **White noise**. In [1], this type of noise is attributed to thermal fluctuations mainly observed in mechanical and electronic components. Thermal noise in electronics circuits depends on bandwidth, temperature and resistance values [22]. It is called white noise due to its power spectral density is almost the same for all frequencies in a given bandwidth. Also, it is possible to define a white noise signal in statistical terms, as in [23], where it is defined as an infinite number of random variables (one for each instant of time) statistically independent and with a zero-mean Gaussian distribution.
2. $1/f$ **Noise**. Also known as flicker noise or pink noise, it is usually attributed to fluctuations in the conductivity of metals and semiconductors. This noise is present in most doped electronic devices, such as resistors, diodes and transistors. Its power spectral density is proportional to $1/f$ over a wide range of frequencies, resulting in a slope of -1 in a log-log graph in the frequency domain.
3. $1/f^2$ **Noise**. Also known as brown noise or Brownian motion process [24]. It generates measurement errors characterized by variances that grow linearly with time and by power spectral densities that fall off as $1/f^2$, i.e. -40 dB per decade, or a slope of -2 in a log-log graph in the frequency domain. This noise is also known as random walk error due to resemblance to this phenomenon [23]. In [24] $1/f^2$ noise is defined as a zero mean stochastic process with undefined correlation time, where increments are independent and stationary.

In the sensors' domain, magnitudes of these stochastic errors are usually quantified by analyzing data records from a sensor's output in absence of an input (or a constant input equal to zero). Commonly, this is done by means of power spectral analysis and Allan variance method. These two methods are explained below.

1.4 Power Spectral Density (PSD) graph

Power spectral density describes how the power of a signal (also called time series) is distributed in the frequency domain [25]. It is computed by the Fourier transform of the autocorrelation in a signal, and it is expressed in Units^2/Hz, where "Unit" is the input signal unit (m s^{-2} for an accelerometer, °/s for a gyrometer…). Figure 1.6 presents an overview of a two-sided PSD graph, which is frequently plotted in a log-log form (Units^2/Hz versus Hz). As mentioned before, this graph is obtained by analyzing an ensemble of measurements given by a sensor over a long period of time with an input equal to zero (i.e. a null input signal). Through this plot it is possible to determine the power density of white noise (slope 0), $1/f$ noise (slope -1) and $1/f^2$ noise (slope -2). PSD graph does not contain any information about systematic errors.

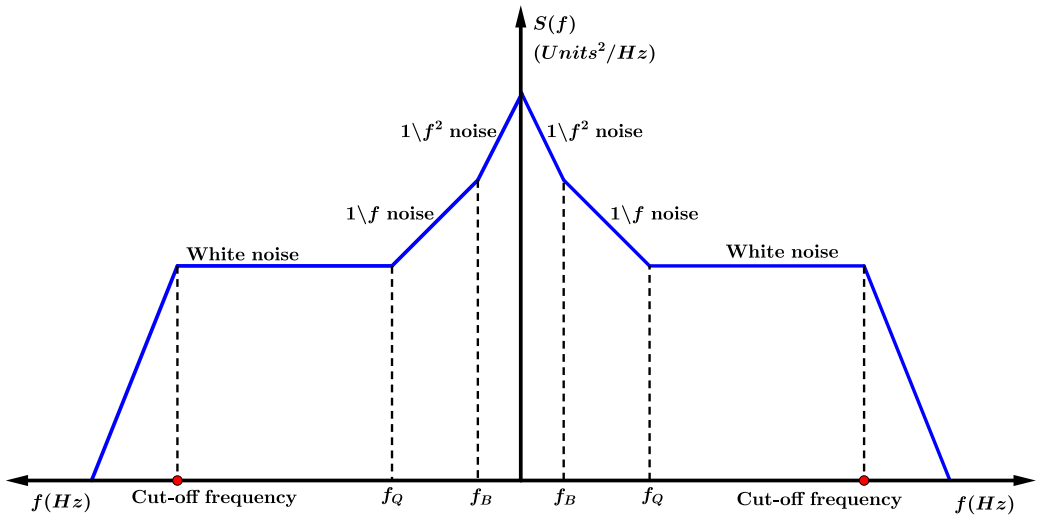


Figure 1.6 – A general example of a two-sided PSD graph. This is a log-log graph with Units^2/Hz vs Hz.

- White noise has a constant PSD that is described by [21]:

$$S(f)_{wn} = Q^2 \tag{1.7}$$

where $S(f)_{wn}$ is the two-sided PSD of white noise in function of frequency f, and Q is the constant related to the power of this noise. PSD is scaled in Units^2/Hz, and for sensors, the square root of this value can be found in the datasheets but it is typically converted into an equivalent physical input and given for a one-sided PSD graph. For example, a noise equivalent acceleration is often given in μg/Hz$^{1/2}$ for an accelerometer. This value corresponds to $2Q$. $S(f)_{wn}$ can be obtained from this value by dividing by 2 and then squared the result.

- In the case of $1/f$ noise, PSD function is defined by [21]:

$$S(f)_{pn} = \left(\frac{B^2}{2\pi}\right)\frac{1}{f} \tag{1.8}$$

where $S(f)_{pn}$ is the two-sided PSD of $1/f$ noise, and B is the bias instability coefficient. Often constant B is not included into sensor data sheets.

- Finally, PSD function of $1/f^2$ noise is described by [21]:

$$S(f)_{bn} = \left(\frac{K}{2\pi}\right)^2\frac{1}{f^2} \tag{1.9}$$

where $S(f)_{bn}$ is the two-sided PSD of $1/f^2$ noise, and K is the random walk coefficient. As a note, parameter K is usually not included in sensor datasheets.

Consider a sensor with a null input signal (i.e., equal to zero). Then, the PSD from measurements of this sensor will be the PSD of its noise. Taking equations 1.7, 1.8 and 1.9, the PSD of the noise present in a sensor can be defined as:

$$S(f) = S(f)_{wn} + S(f)_{pn} + S(f)_{bn} \tag{1.10}$$

Note that effects produced by lower frequency noises are neglected. Figure 1.6 shows how each noise is dominant at certain frequencies. For example, white noise is dominant from f_Q to the cut-off frequency given by the anti-aliasing filter at the output of the sensor. $1/f$ noise is dominant between f_B (the corner frequency with $1/f^2$ noise) and f_Q (the corner frequency with white noise). Below f_B $1/f^2$ noise is dominant. Above f_Q white noise is dominant. Finally, $1/f^2$ noise is dominant below f_B.

It is important to observe, that equations (1.7), (1.8) and (1.9) are related with a two-sided PSD representation. In an one-sided PSD graph, those powers are doubled [25].

1.4.1 Quantifying stochastic errors by means of a PSD graph

To exemplify this process of extracting parameters from a PSD graph, figure 1.7 will be used. Here, it is assumed that horizontal axis f is given in Hz, and vertical axis $S(f)$ is given in Units^{2}Hz^{-1}. It is important to note however, that figure 1.7 shows now a one-sided PSD graph, which means that the values read directly from this graph will be multiply by a factor of two. Quantification of white, $1/f$ and $1/f^2$ noises from this graph is described below.

- As equation (1.15) shows, power spectral density of white noise $S(f)_{wn}$ presents a constant behavior for all f in a bandwidth. The magnitude of this noise is measured in Units^2/Hz by reading the flat region of the graph and then (because Figure 1.7 is a one-sided representation) divided by 2. Parameter Q can also be obtained by reading the flat region of the graph, divided by two and then computing the square root of it. Parameter Q is given in Units$\cdot\sqrt{\text{Hz}}$.

- Power spectral density of $1/f$ noise $S(f)_{pn}$ presents a slope -1 behavior in a log-log graph. The magnitude of this noise is measured in Units^2/Hz by reading the value of this slope at $f = 1$Hz and then divided by two (due to Figure 1.7 is a one-sided representation). If the graph does not present a slope -1 at $f =$1Hz, then the $1/f$ noise behavior is extrapolated with a slope -1, or read at $f = 10^{-n}$ Hz (where $n \in \mathbb{Z}$) and then divide it by 10^n (as shown

in Figure 1.7). Also, parameter B can be extracted directly from this graph by reading the value of the slope -1 at $f = 1$Hz, multiply this value by π and then computing the square root of that. Parameter B is given in Units.

- Power spectral density of $1/f^2$ noise $S(f)_{bn}$ presents a slope -2 behavior in a log-log graph. The magnitude of this noise is measured in Units^2/Hz by reading the value of this slope at f =1Hz and then divided by two (due to Figure 1.7 is a one-sided representation). Again, if the graph does not present a slope -2 at f =1Hz, then the $1/f^2$ noise behavior is extrapolated with a slope -2, or read at $f = 10^{-m}$ Hz (where $m \in \mathbb{Z}$) and then divide it by 10^{2m} (as illustrated in figure 1.7). Parameter K can also be extracted directly by reading the value of this slope at f =1Hz, multiply it by $2\pi^2$ and then computing the square root of this. Parameter K is given in Units$/\sqrt{\text{Hz}}$.

The extraction of white noise, $1/f$ noise and $1/f^2$ noise magnitudes is illustrated in Figure 1.7.

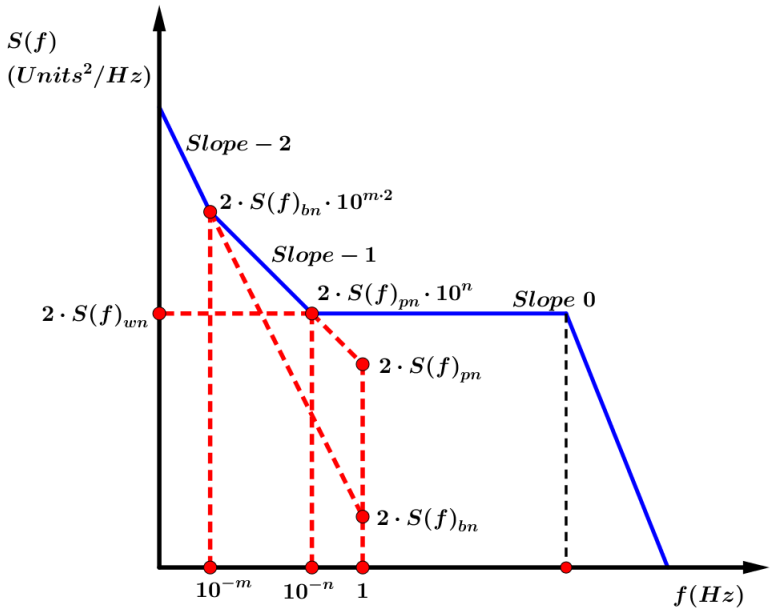


Figure 1.7 – Extraction of white noise, $1/f$ noise and $1/f^2$ noise magnitudes from a PSD graph.

1.5 Allan variance method and Allan deviation (ADEV) graph

The Allan variance is an analysis of a sequence of data in the time domain [21]. This method is generally used to quantify noise present in a system as a function of the averaging time. Allan variance $\sigma^2(\tau)$ is related to the two-sided PSD $S(f)$ by [21] [26]:

$$\sigma^2(\tau) = 4\int_0^\infty S(f)\frac{\sin^4(\pi f\tau)}{(\pi f\tau)^2}df \tag{1.11}$$

where f is the frequency and τ is the cluster's length time. Usually, a log-log plot of $\sigma(\tau)$ versus τ is computed to direct measure parameters related to random process errors. This plot is known

as Allan DEViation (ADEV) graph. As the PSD graph, this graph is obtained by analyzing an ensemble of measurements given by a sensor over a long period of time with an input equal to zero (i.e. a null input signal).

Figure 1.8 presents the general overview of an ADEV graph. This plot is commonly used for identifying and quantifying different stochastic phenomena affecting sensor measurements. The five contributions generally found are quantization noise (corresponding to the slope -1), white noise (slope -1/2), $1/f$ noise (slope 0), $1/f^2$ noise (slope +1/2), and drift rate ramp (slope +1).

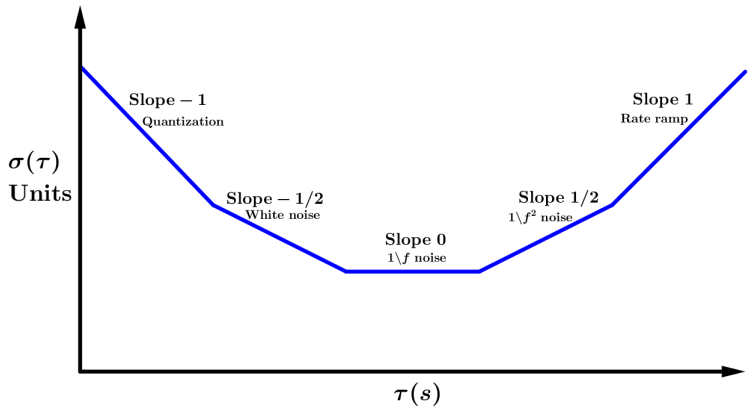


Figure 1.8 – General example of an ADEV graph. This is a log-log graph with Units vs s.

By replacing equations (1.7), (1.8), and (1.9) in (1.11), and performing the integration, it is possible to obtain variances for white noise (σ^2_{wn}), $1/f$ noise (σ^2_{pn}), $1/f^2$ noise (σ^2_{bn}) as follows [26]:

$$\sigma^2_{wn}(\tau) = \frac{Q^2}{\tau} \tag{1.12}$$

$$\sigma^2_{pn}(\tau) = \frac{B^2 2\ln 2}{\pi} \tag{1.13}$$

$$\sigma^2_{bn}(\tau) = \frac{K^2\tau}{3} \tag{1.14}$$

ADEV graph presents information in terms of the standard deviation of the output when the input is null. Relationships between information given by an ADEV graph and the stochastic errors are given by the following equations:

$$\sigma_{wn}(\tau) = \frac{Q}{\sqrt{\tau}} \tag{1.15}$$

$$\sigma_{pn}(\tau) = \frac{B\sqrt{2\ln 2}}{\sqrt{\pi}} \tag{1.16}$$

$$\sigma_{bn}(\tau) = \frac{K\sqrt{\tau}}{\sqrt{3}} \tag{1.17}$$

1.5.1 Quantifying stochastic errors by means of an ADEV graph

To illustrate the extraction process of parameters from an ADEV graph, figure 1.9 is used. Here, it is assumed that τ is in s and $\sigma(\tau)$ is in Units. Due to the fact that this thesis focuses on the study of effects generated by white, $1/f$ and $1/f^2$ noises, only these stochastic errors are considered during this example (quantization noise and drift rate ramp can be consulted in [26] and [21]). Next, quantifications of white, $1/f$ and $1/f^2$ noises from an ADEV graph are explained in detail.

- As it can be noted from (1.15), magnitude of white noise Q can be directly obtained by reading the value of the graph at $\tau = 1$s. From equation (1.15), it can be deduced that Q is given in Units$\cdot\sqrt{\text{s}}$.

- Bias instability coefficient B (related to the $1/f$ noise magnitude) is obtained by reading the value in the flat region of the graph (slope $= 0$) and then divide it by 0.664. Factor 0.664 comes from $(2\ln 2/\pi)^{1/2}$. From equation (1.16) it can be noted that B is given in Units.

- Random walk coefficient K (related to the $1/f^2$ noise magnitude) can be extracted by reading the slope 1/2 of the graph either directly at $\tau = 3$s, or at $\tau = 3 \cdot 60^2 = 10800$s (i.e, $\tau = 3$h) and then divide it by 60. Factor $1/60$ arises from converting K from Units$/\sqrt{\text{s}}$ to Units$/\sqrt{\text{h}}$. Figure 1.9 shows extraction of coefficient K at both $\tau = 1$s and $\tau = 3$h. For the first, an extrapolation of slope 1/2 is carried out (dotted line). From equation (1.17) it can be deduced that K is given in Units$/\sqrt{\text{s}}$.

Figure 1.9 illustrates extraction of coefficients Q, B, and K as it is described above.

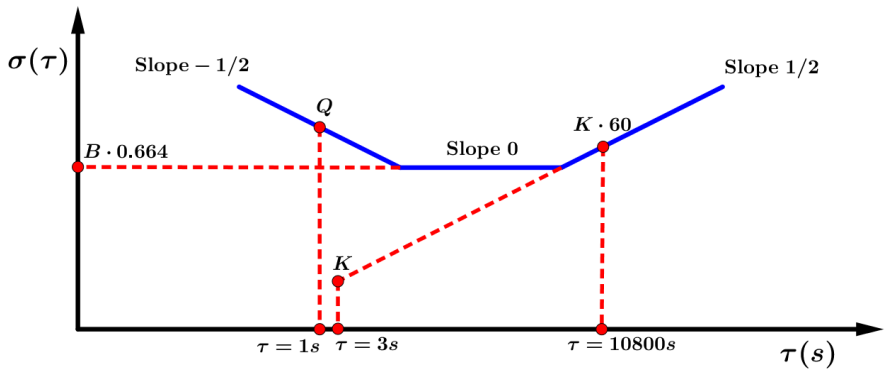


Figure 1.9 – Extraction of parameters Q, B and K from an ADEV graph.

1.6 Relationship between PSD graph and ADEV graph

The relationship between a PSD graph (magnitudes of $S(f)_{wn}$, $S(f)_{pn}$ and $S(f)_{bn}$) and an ADEV graph (coefficients Q, B and K) is illustrated in Figure 1.10. A summary of how to quantify white, $1/f$ and $1/f^2$ noises by means of ADEV and PSD graphs is presented in Table 1.2.

Table 1.2 – Quantifying stochastic errors by a PSD or an ADEV graph.

	White noise	$1/f$ noise	$1/f^2$ noise
Two-sided PSD graph	$S(f)_{wn} = Q^2$	$S(f)_{pn} = \left(\frac{B^2}{2\pi}\right)\frac{1}{f}$	$S(f)_{bn} = \left(\frac{K}{2\pi}\right)^2 \frac{1}{f^2}$
Parameter / Units	$S(f)_{wn}$ - Units^2/Hz	$S(f)_{pn}$ - Units^2/Hz	$S(f)_{bn}$ - Units^2/Hz
Read it at	flat region (slope = 0)	$f = 1$Hz or $f = 10^{-m}$Hz and divide it by 10^m	$f = 1$Hz or $f = 10^{-m}$Hz and divide it by 10^{2m}
ADEV graph	$\sigma_{wn}(\tau) = \frac{Q}{\sqrt{\tau}}$	$\sigma_{pn}(\tau) = \frac{B\sqrt{2\ln 2}}{\sqrt{\pi}}$	$\sigma_{bn}(\tau) = \frac{K\sqrt{\tau}}{\sqrt{3}}$
Parameter / Units	Q - Units$\cdot\sqrt{\text{s}}$	B - Units	K - Units$/\sqrt{\text{s}}$
Read it at	$\tau = 1$s	flat region (slope = 0)	$\tau = 3$s or $\tau = 3$h and divide it by 60

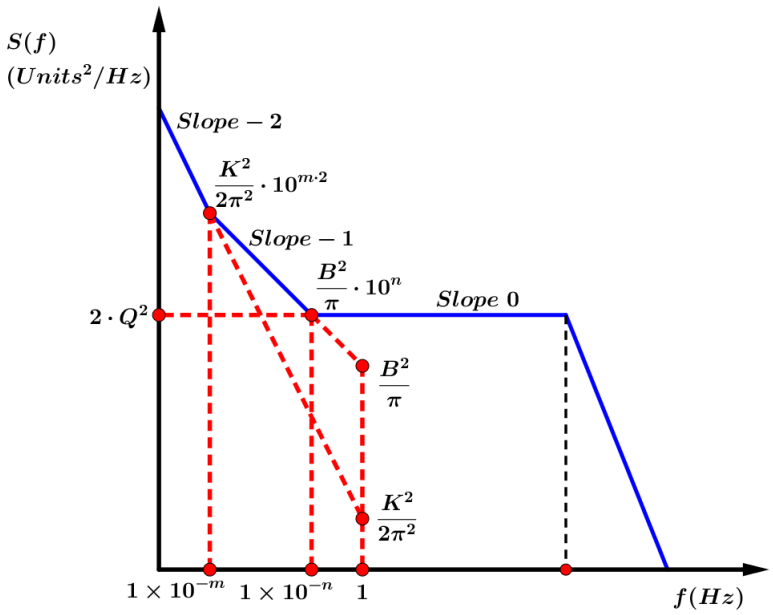


Figure 1.10 – Relationship between a PSD graph and coefficients Q, B and K.

1.7 Simulation of colored noise

In this section, a method for simulating $1/f^\gamma$ noise is presented. This method was first introduced by N. Kasdin in [24]. Here, we give a brief summary about this method and how it can be implemented into Matlab simulink environment.

The general idea presented in [24] to generate a discrete simulation of $1/f^\gamma$ noise is to pass a white noise sequence through a filter and then, depending on the transfer function, a sequence of colored noise is obtained at the output. This method works with both Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) filters. For the FIR filter, the transfer function to generate $1/f^\gamma$ noise is:

$$H(z) = 1 + \frac{\gamma}{2}z^{-1} + \frac{\gamma/2(\gamma/2+1)}{2!}z^{-2} + \dots \tag{1.18}$$

This results in a pulse response, by the definition of the z-transform. Coefficients of this transfer function can be easily computed using the following recursive algorithm:

$$\begin{aligned} H(z) &= h_0 + h_1 z^{-1} + h_2 z^{-2} + \dots \\ h_0 &= 1 \\ h_k &= \big(\frac{\gamma}{2} + k - 1\big)\frac{h_{k-1}}{k} \end{aligned} \tag{1.19}$$

In the case of the IIR filter, transfer function used to generate $1/f^{\gamma}$ noise is:

$$H(z) = \frac{1}{1 - \frac{\gamma}{2}z^{-1} - \frac{(\gamma/2)(1-(\gamma/2))}{2!}z^{-2} + \dots} \tag{1.20}$$

This transfer function is equivalent to a recursive autoregressive filter. Filter coefficients can be easily found from an iterative formula described below:

$$\begin{aligned} H(z) &= \frac{1}{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots} \\ b_0 &= 1 \\ b_k &= \big(k - 1 - \frac{\gamma}{2}\big)\frac{b_{k-1}}{k} \end{aligned} \tag{1.21}$$

From equations (1.19) and (1.21) it could be noted that for $1/f^2$ noise (i.e., $\gamma = 2$), coefficients where $k \geq 2$ are equal to zero. Then, transfer function results in:

$$H(z) = 1 + z^{-1} \qquad \text{FIR filter} \tag{1.22}$$

$$H(z) = \frac{1}{1 - z^{-1}} \qquad \text{IIR filter} \tag{1.23}$$

In contrast, coefficients for generating $1/f$ noise can be computed indefinitely. The size of the $1/f$ behavior in the output sequence depends on the number of coefficients used in the filter [24]. Hereafter, it is assumed that an IIR filter is used for the colored noise simulation. The same can be done by using a FIR filter.

It can be seen from the above analysis that colored noise can be simulated into Matlab simulink environment by using a *Gaussian Noise Generator* block, and then, pass the white noise generated through a *Discrete Filter* block, as it is shown in figure 1.11.

Output colored noise magnitude and frequency are related to the parameters of input noise sequence. Indeed, relationship between the variance of the input white noise sequence (σ^2_{input}) and the power spectral density of the colored noise at the output ($S(f)$) is described as [24]:

$$S(f) \approx \frac{\sigma^2_{input}\Delta t^{1-\gamma}}{(2\pi f)^{\gamma}} \tag{1.24}$$

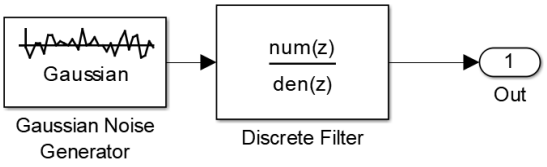


Figure 1.11 – Colored noise ($1/f$ or $1/f^2$ noise) is generated by passing a white noise sequence through a FIR / IIR filter. Color of the noise depends of coefficients in the filter.

where Δt is the sample time of the output. For $1/f$ noise (i.e., $\gamma = 1$), equation (1.24) results in:

$$S(f) \approx \frac{\sigma_{input}^2}{2\pi f} \tag{1.25}$$

σ_{input} and parameter B are related as follows (see equations (1.25) and (1.8)):

$$\sigma_{input}^2 = B^2 \tag{1.26}$$

Therefore, to obtain a $1/f$ noise sequence with a specific magnitude given by B, configuration of blocks shown in Figure 1.11 should be:

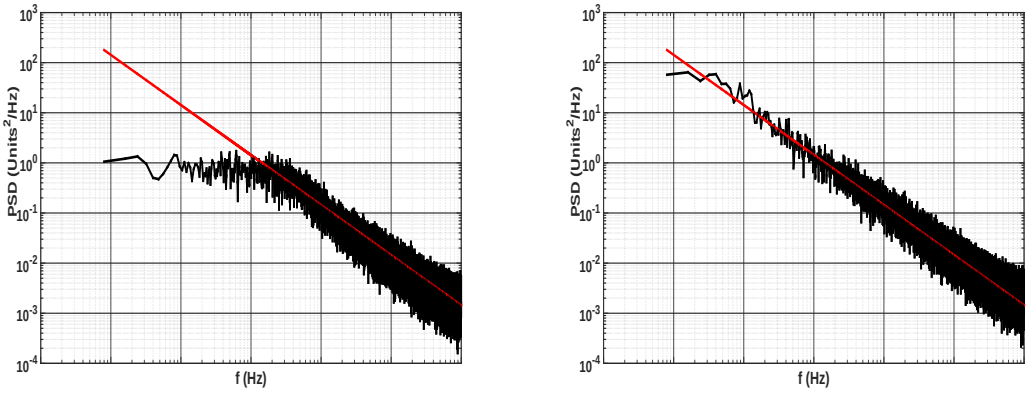
Gaussian Noise Generator
mean: 0
variance: B^2
sample time: Δt_{pn}

Discrete Filter
numerator: 1
denominator: $b_0, \ldots, b_n$

Here, b_n denotes the n-th coefficient in the transfer function. The number of coefficients is limited by the *Discrete Filter* block, which allows an implementation of up to 1000 coefficients. It is important to note however, that due to the fact that the number of coefficients is limited, the $1/f$ behavior of the output noise sequence is restricted to certain period of time and / or frequency. For example, figure 1.12 shows the difference between spectral densities obtained from analysis of two different vectors of 10^6 points. The first vector was created using Matlab Simulink *Colored noise* block which implements an IIR filter with 63 coefficients [27]. Here, incomplete $1/f$ behavior is obtained (figure 1.12a). The second vector was created using an IIR filter with 1000 coefficients. In this case, the complete spectral density shows a $1/f$ behavior (figure 1.12b). If a more extensive $1/f$ behavior were required, then the implementation shown in figure 1.11 should be replaced by a noise sequence created with function presented in Appendix, where a greater number of coefficients can be implemented.

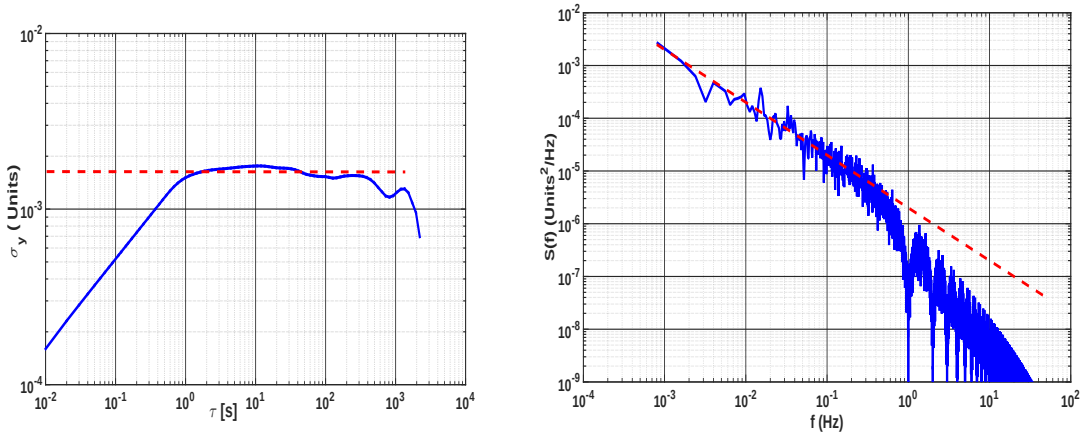
In addition, Δt_{pn} denotes the sampling time of *Gaussian Noise Generator* block, which is related to the $1/f$ behavior. This value is the time at which noise sequence starts a $1/f$ behavior (in terms of an ADEV graph); before that, it presents a $1/2$ slope, as shown in figure 1.13a. The same analysis can be done for a PSD graph, where Δt_{pn} denotes the inverse of frequency at which noise sequence ends a $1/f$ behavior; after that, its behavior presents a slope = -2, as shown in figure 1.13b. As it was already mentioned, size of $1/f$ behavior is restricted by the number of coefficients used in the filter.

On the other hand, for $1/f^2$ noise ($\gamma = 2$), equation (1.24) results in:



(a) (b)

Figure 1.12 – Impact of the number of coefficients used in the IIR filter for generating $1/f$ noise. (a) PSD graph of a vector of 10^6 points generated with an IIR filter with 63 coefficients (Matlab Simulink block *Colored Noise*). (b) PSD graph of a vector of 10^6 points generated with an IIR filter with 1000 coefficients.



(a) Flat region start is related to sample time of noise generator.

(b) $1/f$ behavior ending is related to sample time of noise generator.

Figure 1.13 – Generation of $1/f$ noise using a IIR filter.

$$S(f) \approx \frac{\sigma_{input}^2 \Delta t^{-1}}{(2\pi f)^2} \tag{1.27}$$

σ_{input} and parameter K are related by the following relationship (see equations (1.27) and (1.9)):

$$\sigma_{input}^2 = \Delta t K^2 \tag{1.28}$$

Therefore, for obtaining a $1/f^2$ noise sequence with a specific magnitude given by K, configuration of blocks shown in figure 1.11 should be:

Gaussian Noise Generator
mean: 0
variance: $\Delta t K^2$
sample time: Δt

Discrete Filter
numerator: 1
denominator: b_0, b_1

1.8 Sensor array systems

A sensor array system can be defined as a group of similar and/or dissimilar sensors, embedded in the same chip or in different interconnected chips, usually deployed in a fixed topological pattern, and used to measure one or several physical inputs [28]. Once measurements are obtained, a data fusion method is applied to merge all information reported by such devices. These methods permit to process data from several sensors obtaining advantages such as improvement in accuracy and reliability, as well as fault detection [8]. These systems, as well as providing a high performance to price ratio, can also provide new measurement capabilities and can enable the development of smart systems able to self-adapt to the usage conditions [11].

Sensor array systems are becoming increasingly important in a variety of military and civilian applications. Since a single sensor generally can only perceive limited partial information about the environment, multiple similar and/or dissimilar sensors are required to provide sufficient local pictures with different focus and from different viewpoints in an integrated manner. Further, information from heterogeneous sensors can be combined using data fusion algorithms to improve dependability and robustness of sensing [29]. Thus, the benefits of sensor array systems are to broaden system perception and enhance awareness of the environment compared to what could be acquired by a single sensor.

1.8.1 Classifications

There are different ways to classify sensor array systems. For example, depending on the system architecture they can be classified into two groups: centralized or distributed. The following definitions of those groups are taken from [30]:

- Centralized. Each sensor is physically attached to a central processing unit. All the data streams from sensors are received in parallel, making it easy to correlate the absolute time correspondence of the information. The ability to measure and compensate for the delay between each sensor's information streams makes them easy to correlate during the data fusion stage. One strength of this type of multi sensor systems is that the central processing unit can control each sensor. This means that it can quickly adjust sensor parameters in reaction to an event as well as detect faults.

- Distributed. This systems can usually cover the observation of more properties than their centralized counterparts. Data processing directly performed on the sensors also helps to lighten the processing load on the central processing unit.

A different classification can be given according to the group operation of the sensors. Here, the classification taken from [29]:

- Complementary. A configuration is called complementary when sensors do not directly depend on each other, but can be combined in order to give a more complete image of the phenomenon under observation.

- Competitive. A configuration is competitive when each sensor delivers an independent measurement of the same property. The aim of competitive fusion is to reduce the effects of stochastic errors.

- Cooperative. A configuration is called cooperative when information provided by two, or more, independent sensors is used to derive information that would not be available from the single sensors.

For the purposes of this work, and using classifications shown above, we catalog sensor array systems according to the homogeneity of its elements, as shown below:

- Same-type sensor systems. Such systems permit to observe and report same property from several similar sensors, taking advantage of redundancy in measurements. This classification is similar to the competitive configuration given in [29]. Elements in same-type sensor systems present different levels of accuracy, and in consequence, usage of data fusion algorithms is required. As an example, an implementation of a sensor array system composed of 32 Inertial Measurement Units (IMUs) is presented in [31]. Even if each IMU contains different types of sensors (accelerometers, gyroscopes and magnetometers), the system is classified as a same-type sensor system, since all elements in the system are IMUs, as illustrated in Figure 1.14a.

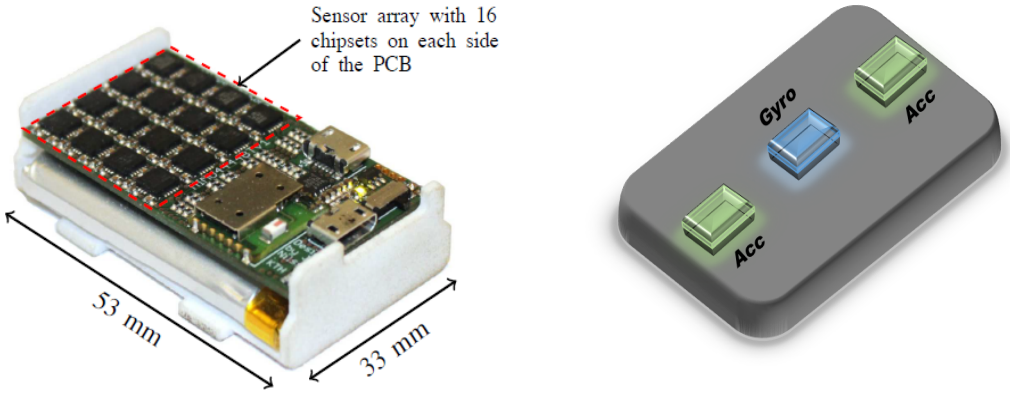
- Different-type sensor systems. These systems are designed for measuring complex phenomena which cannot be observed directly, or for complex environments with several correlated variables. This classification is similar to the cooperative configuration given in [29]. As example, in [32] an implementation of a system composed of two accelerometers and one gyroscope is presented. Here, two different types of sensors are used to estimate directly and indirectly the same property. This system is illustrated in Figure 1.14b.

This thesis is oriented to the treatment of redundant data, and therefore, throughout this work centralized same-type sensor array systems are generally considered.

1.9 Multi sensor data fusion

A multi sensor data fusion can be defined as the theory, techniques and tools which are used to combine measurements coming from different sensors into a single estimation of the input [18]. Data fusion makes use of multiple sensors' performances, analyze and integrate the data detected by each sensor in order to extract the best information. By performing sensors' data fusion the aim is to improve the quality of the information so that it is better than would be if the data sources were used individually.

The main motivation for multi sensor data fusion is the improvement of the quality of information obtained during the measurement process. However, employing more than one sensor may enhance some other aspects of the sensing system, such as better noise suppression, increased reliability, and increased robustness to sensors failures.



(a) Same-type multi sensor system composed of 32 Inertial Measurement Units (configuration presented in [31]).

(b) Different-type multi sensor system composed of two accelerometers and one gyroscope (configuration presented in [32]).

Figure 1.14 – Classification of multi sensor systems according to the homogeneity in its elements.

The following is taken from [33], where it is enunciated four ways in which multi sensor data fusion may improve the performance of a sensing system:

- Representation. The information obtained at the end of the fusion process has an abstraction level higher than each input data set.
- Accuracy. The standard deviation on the data after the fusion process is smaller than the standard deviation obtained directly by the sources. If data is noisy or erroneous, the fusion process aims to reduce or eliminate noise and errors.
- Adaptability. If during the run-time one or more sensors present faults, it is possible to continue with the measurement process using the fault free sensors.
- Completeness. Bringing new information to the current knowledge on an environment allows a more complete view on this environment. In general, if the information is redundant and concordant, we could also have a gain in accuracy.

1.9.1 Data fusion type

The process of data fusion carried out in a multi sensor system can be classified according to different aspects. One of the most used classifications is the one presented in [34], which divides data fusion processes according to the way of implementing sensors measurements. The following is taken from [34], which describes such classification:

- Fusion across sensors. Sensors measure the same physical attribute. For example, a set of temperature sensors measuring the temperature of the same room.
- Fusion across attributes. Sensors measure different attributes associated to the same phenomenon. For example, measurement of air temperature, pressure and humidity to determine air refractive index.

- Fusion across domains. Sensors measure the same physical attribute over a number of different ranges or domains. For example, traffic in a given area can be measured through the concurrence of vehicles and the speed at which they move [35].
- Fusion across time. Sensors measure the same physical attribute, but at different instants of time. For example, a system composed of sensors with different sampling frequencies.

Estimation of position carried out by inertial navigation systems is an example of fusion across attributes, since acceleration and angular rate measurements are used to compute this property. In contrast, sensor array system presented in Figure 1.14a is an example of data fusion across sensors, since all system elements are used to measure the same properties. The work developed in this thesis is aimed at sensor array systems based on fusion across sensors.

1.9.2 Data fusion methods

Most of the current data fusion methods are based on probabilistic methods, which are considered as the standard approach in most of the robotics applications [36]. Indeed, these probabilistic methods are generally based on Bayes' rule, which combines prior knowledge about the input with an update given by observation. This may be implemented in a number of ways: through the use of Kalman filters, through sequential Monte Carlo methods, or through the use of functional density estimates [36]. On the other hand, there are a number of alternatives to probabilistic methods, such as those offered by methods based on the theory of evidence and delimitation for intervals. Such alternative techniques are not as widely used, especially inside the multi sensor's field. Despite this, such alternatives are worth mentioning as they present some special features that can be advantageous in specific problems [36].

Bayes' Rule and Bayesian inference

Bayes' rule is the base of most probabilistic data fusion methods. Broadly speaking, Bayes' rule provides a way to make inferences about an state x, given an observation z. For this, it is required that the relationship between x and z be encoded as a joint probability or a joint probability distribution (denoted by $P(x, z)$) for discrete or continuous variables, respectively. The chain rule of conditional probabilities can be used to expand the joint probability as:

$$P(x, z) = P(x|z)P(z) = P(z|x)P(x) \tag{1.29}$$

Here $P(x|z)$ denotes the conditional probability of a state x for a given observation z, $P(z|x)$ denotes the conditional probability of having an observation z for a given state x, and $P(z)$ and $P(x)$ denote the marginal probabilities of observations and states, respectively. Rearranging expression (1.29) in terms of $P(x|z)$, Bayes' rule is expressed as:

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)} \tag{1.30}$$

Thus, the estimation of a state x given the observation z lies in the interpretation of the probabilities $P(x|z)$, $P(z|x)$, and $P(x)$. Note that observations z are modeled in the form of a conditional probability $P(z|x)$ that describes the probability of z given x. Thus, likelihood associated with the state x is computed from the product of the original prior information of x and the information

gained by observation z (i.e., $P(z|x)P(x)$). This results in the probability $P(x|z)$, which describes the likelihood associated with x given the observation z.

The marginal probability $P(z)$ is used to normalize the posterior and usually is omitted when implementations are carried out [36]. However, probability $P(z)$ plays an important role in model validation: it provides a measure of how well the observation is predicted by the prior. One of the data fusion methods that implement this normalization is the well known Kalman filter.

Multi sensor Bayesian inference

The conditional probability $P(z|x)$ serves the role of a sensor model. This can be illustrated in two ways. From [36]: *"first, the probability is constructed by fixing the value of $x = c_x$ and then asking what probability density $P(z|x = c_x)$ on z results. Conversely, when this sensor model is used and observations are obtained, $z = c_z$ is fixed and a likelihood function $P(z = c_z|x)$ on x is inferred"*. Likelihood $P(z|x)$ models how observed values z correspond to different values of x. The product of this likelihood with the prior likelihood of x, gives the posterior $P(x|z)$.

To be implemented in multi sensor systems, the multi sensor form of Bayes' rule requires conditional independence:

$$P(z_1, \dots, z_n|x) = \prod_{i=1}^{n} P(z_i|x) \tag{1.31}$$

So that

$$P(x|Z^n) = P(x) \prod_{i=1}^{n} P(z_i|x) \tag{1.32}$$

where

$$Z^n = [z_1, \dots, z_n] \tag{1.33}$$

From equation (1.32) it follows that probability on x given the n observations (denoted by Z^n), is proportional to the product of prior probability (i.e. $P(x)$) and individual likelihoods from each information source [36]. Thus, the recursive form of Bayes' rule can be expressed as:

$$P(x|Z^k) = \frac{P(z_k|x)P(x|Z^{k-1})}{P(z_k|Z^{k-1})} \tag{1.34}$$

From (1.34) we observe that no big storage is required, since $P(x|Z^{k-1})$ takes the role of a summary of all past information. When new observations are available, $P(x|Z^{k-1})$ becomes the current prior and the product of the two becomes the new posterior. This process is illustrated in chapter 3, by means of the Kalman filter.

Alternatives to probabilistic methods

Alternative modeling techniques have been proposed to deal with perceived limitations in probabilistic methods. To list the three main perceived limitations, the following is taken from [36]:

- Complexity. The need to specify a large number of probabilities.

- Inconsistency. Difficulties involved in specifying a consistent set of beliefs in terms of probability and using these to obtain consistent deductions about states of interest.
- Precision of models. The need to be precise in the specification of probabilities.

There are many different approaches to overcome these issues. Next, three of the most popular techniques are depicted: interval calculus, fuzzy logic, and the theory of evidence (also known as Dempster–Shafer methods).

In interval calculus, it is used a representation of uncertainty using an interval to bound true parameter values. From [36]: *"intervals provide a good measure of uncertainty in situations where there is a lack of probabilistic information, but in which sensor and parameter error is known to be bounded"*. In interval techniques, the uncertainty in a state x is simply described by a statement that the true value of x is known to be inside of a given range, i.e., $x \in [a, b]$ such that $a, b \in \mathcal{R}$.

Another option is the fuzzy logic, which has found widespread popularity as a method for representing uncertainty [36]. Broadly speaking, this method creates sets and membership functions which will assign a value between 0 and 1 indicating the degree of membership of every element (e.g. every sensor or reported measurement) to each set. Just as an example, if it is seeking to create a fault-free multi sensor system, a set called "fault-free" can be formed, as well as the corresponding membership function, which will determine the degree of certainty that a sensor is fault-free. The complexity of this method lies in forming the appropriate membership function. In addition, in many cases the membership function requires a partial knowledge of the input signal.

Finally, evidential reasoning (often called the Dempster–Shafer method) has seen intermittent success particularly in automated reasoning applications [36]. In order to illustrate this method, let's retake the example of the multi sensor system free of faults. Now, consider the mutually exclusive set $\mathcal{X} =$ faulty, fault-free. In probability theory we might assign a probability to each possible event, for example, P(faulty) $= 0.3$, and thus P(fault-free) $= 0.7$. In evidential reasoning, we construct the set of all subsets $2^{\mathcal{X}} = \{\{\text{faulty}\}, \{\text{free-fault}\}, \{\text{faulty, free-fault}\}, \emptyset\}$, and belief mass is assigned to all elements of this set as: m(faulty,free-fault) $= 0.5$, m(faulty) $= 0.3$, m(free-fault) $= 0.2$, $m(\emptyset) = 0.0$. Evidential reasoning provides a method of capturing the inability to distinguish between alternatives, which allows a richer representation of beliefs. However, this comes at the cost of an increase in complexity, as explained in [36]: *"if there are n elements in the original set $\mathcal{X}$, then there will be 2^n possible subsets on which a belief mass will be assigned. For large n, this is clearly intractable. Further, when the set is continuous, the set of all subsets is not even measurable"*.

1.9.3 Applications

Multi sensor fusion systems have been applied to a wide variety of problems in several different areas of research [37]. In [36] a division into two general areas is given. This two divisions are dynamic system controls and environment modeling. To present this classification, the following is taken from [36]:

- Dynamic system control. It is the usage of appropriate models and sensors to control the state of a dynamic system (e.g., industrial robot, mobile system, autonomous vehicle, surgical tool, etc.). Usually such systems involve real-time feedback control loops for steering, acceleration, and behavior selection. In addition to state estimation, uncertainty models are required. Sensors may include force/torque sensors, gyros, global positioning system (GPS), position encoders, cameras, range finders, etc.

- Environment modeling. It is the usage of appropriate sensors to construct a model of some aspect of the physical environment. This may be a particular object, or a larger part of the surroundings. Typical sensors include cameras, radar, 3-D range finders, infrared (IR), tactile sensors and touch probes (CMMs), etc. The result is usually expressed as geometry (points, lines,surfaces), features (holes, sinks, corners, etc.), or physical properties. Part of the problem includes the determination of optimal sensor placement.

At the beginning of chapter 4, some examples from the literature of applications of sensor arrays to problems such as inertial navigation or magnetic field measurement are presented. Now, in the next chapter, we introduce the generic model used to simulate sensors present in an array.

Chapter 2

Generic sensor model for simulations at system level

System-level simulations of sensors are valuables for optimizing device and system parameters and validating data-processing algorithms. Nowadays, the tendency of multi-sensor systems has increased the necessity of this type of simulations. In this chapter, we present a generic model for simulations of sensors at system level. First, a brief introduction is given in section 2.1. Then, the proposed generic model is presented in section 2.2, where each parameter considered is detailed. Our model allows implementing the complete behavior of a sensor including uncertainties, tolerances, nonlinearities, various noises and so on; as it is presented in section 2.3. Also, this sensor model can be customized by extracting information from a datasheet, a power spectral density graph or an Allan deviation graph; as it is illustrated in section 2.4. Thereby, simulations of a single sensor or multi-sensor systems can be performed and data fusion algorithms can be tested for different applications. This is validated by means of simulations in section 2.5.

Most of the content of this chapter has been published in [38] and [39]. Additional material has been added in order to enrich presented information.

2.1 Introduction

A sensor is a device that interacts with its environment, and it is capable of perceiving and measuring a physical property, such as heat, light, sound, pressure, magnetism, or motion. The sensor output is the translation of this physical magnitude into an electrical signal [1]. This output can be displayed, recorded, or used as an input signal to some secondary devices or systems. Sensors have become crucial elements in several areas of application and development that involve environmental monitoring (physics), control (space, industry, and robotic), and human health monitoring (medicine). Due to their wide variety, sensors are powerful tools for measuring different physical phenomena and extracting essential data.

Sensor measurements are uncertain, which means that they can only give an estimate of the measured physical property [18]. Usually, the factors that cause this uncertainty are classified into systematic and stochastic phenomena [17]. Systematic errors include constant biases, scale factor errors, thermal drift and nonlinearities. The stochastic part contains random errors, which cannot be removed by calibration and should be modeled as stochastic processes [40]. Sensor simulation accuracy depends on which systematic and stochastic parameters are considered [17].

The need for accurate system-level simulations arises across almost all disciplines of science and engineering. For example, mechanical and aerospace engineers often simulate scenarios considering noisy environments for testing the performance of sensors [41,42]. In electrical engineering and physics it is common to simulate sensors and actuators affected by different types of colored noises [43, 44]. In computer science and electronic engineering, design and implementations of sensor networks and multi-sensor platforms have increased in the last years [45, 46], creating the need for accurate and fast simulations.

In the literature, different sensor models have been proposed; most of them relates to a specific type of sensor and have been developed into very different software environments [47–50], making hard or even impossible their usage for simulations of systems composed of two or more different-type sensors.

In other cases, proposed models do not consider the essential features of real sensor systems. For example, in [49] a model for a 6-degree of freedom multi-sensor system is developed. This model includes systematic errors such as bias, scale factor, and misalignment, but low-frequency noises are omitted; even when these are the main problem inside inertial navigation systems composed of gyroscopes [51, 52].

Another example of a proposed model is shown in [53], where modeling of a capacitive humidity sensor is carried out by using an artificial neural network (ANN). The main limitation of the presented model, despite the fact that this is a specific model, is the high computational complexity required to characterize one single sensor (complexity estimated to $\mathcal{O}(n^4m^5)$, where n is related to the number of neurons and m to the number of layers [54]). This makes impractical the implementation for simulations of multi-sensor systems.

Thereby, we have decided to develop a new generic sensor model based on Matlab Simulink blocks. This model is presented as a viable solution to limitations observed in existing models in the literature, such as generality, portability, and parametrization. Our proposed model describes a non-ideal sensor without any assumption about its type, which makes feasible its deployment for simulating multi-sensor systems. As a Matlab-made, this model has great portability, which means; it can be easily exported to other programming languages or software environments. Such model has also the advantage to be customizable by taking information from the sensor datasheet, a power spectral analysis of its output and/or an Allan deviation graph, which makes it suitable for simulations of real sensors.

2.2 Sensor behavioral modeling

Modeling in engineering consists in translating a physical device or system into a set of mathematical equations. In the case of sensors, this model describes the relationship between sensor physical input and its electrical output. Due to the non-ideal behavior of sensors, it is necessary to consider different parameters that represent errors affecting sensor measurements. Thus, independently of the sensor type, a generic measurement model can be described as:

$$y_t = \alpha x_t + c_1 x_t^2 + c_2 x_t^3 + \beta + \eta_t \tag{2.1}$$

Where:

x_t	sensor input at time step t,
y_t	sensor output at t,
α	scale factor (typically equal to 1),
β	bias (typically equal to 0),
c_1 , c_2	nonlinearity coefficients (typically equal to zero),
η_t	stochastic errors affecting measurements.

Each parameter relates to the physical characteristics of a sensor, which usually may change depending on the temperature and/or time [17]. Equation (2.1) can be divided into systematic and stochastic phenomena. Systematic errors are scale factor, nonlinearity coefficients, and bias. Stochastic errors are white noise and low-frequency noises, which are included in η_t.

From (2.1), a more specific model that relates physical input signal and the correspondent electrical output is given as:

$$y_t = f(\alpha x_t + c_1 x_t^2 + c_2 x_t^3 + \beta + \eta_t) \tag{2.2}$$

where f denotes a sensitivity function for continuous systems, or a quantization function for discrete systems.

Through this behavioral modeling, simulations of different kinds of sensors can be carried out, as long as their behavior is somehow classical [38]. One important advantage of this model is the inclusion of low-frequency noises ($1/f$ noise and $1/f^2$ noise), whose effects can not be neglected in electronic systems [51]. Also, we have decided to include temperature coefficients of each systematic error to allow analysis under specific environmental conditions. This inclusion is carried out in the next section, where the implementation of this behavioral model is developed into the Matlab Simulink environment.

2.3 Sensor modeling in Matlab Simulink

The proposed sensor model is based on equation (2.2) and implemented using Matlab Simulink blocks. Matlab Simulink environment was chosen for the development of this simulation-tool because, apart from being a user-friendly software, Matlab offers high portability compared with other programs; i.e., it is straightforward to export implementations and data files from Matlab to different software environments.

Before starting with Matlab Simulink environment, we present in Figure 2.1 translation of equation (2.1) in terms of a block diagram. Here, input x represents the ground-truth signal (i.e., the physical input that sensor measures). Then, x is altered by the scale factor and nonlinearity effects, as well as the addition of constant bias. After that, white noise and low-frequency noises are added. Finally, bandwidth limitation and output saturation are applied. These later blocks set the bandwidth and full scale of the sensor, respectively.

Figure 2.1 illustrates a generic sensor model where all blocks outputs are in the same units as the physical input signal. For example, if sensor model is used to simulate an accelerometer with physical input x given in $\mathrm{m\,s^{-2}}$, then, outputs of all model blocks will be given in $\mathrm{m\,s^{-2}}$.

Block diagram representation of equation (2.2) is presented in Figure 2.2. Unlike figure 2.1, output of sensor model is given in V for both continuous and discrete mode.

Thanks to the simplicity of its graphical environment, it is straightforward to translate elements of the diagram in Figure 2.2 as independent blocks into Matlab Simulink. Assuming that the input signal is scaled in a given unit (x in Units), the translated blocks are:

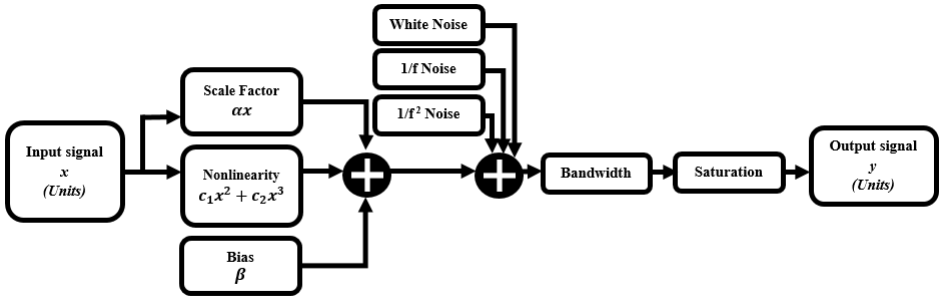


Figure 2.1 – Diagram of sensor model given in (2.1).

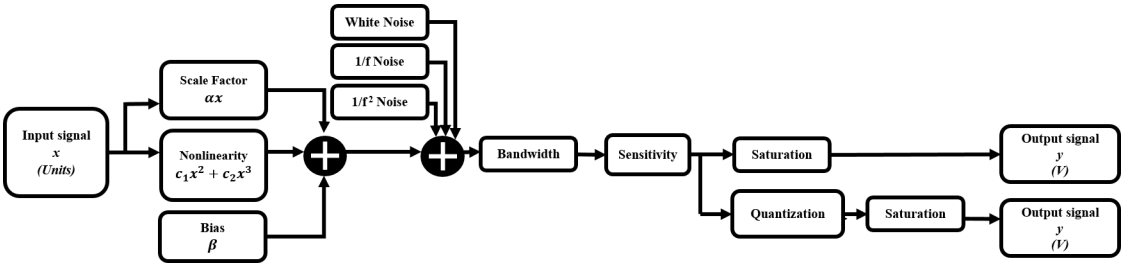


Figure 2.2 – Diagram of sensor model given in (2.2).

1. Scale Factor. This block multiplies the system input by a factor, which may be temperature-dependent and/or affected by random variations. The required parameters to this block are then the typical scale factor constant α, its temperature coefficients ($TC1_\alpha$ in $°C^{-1}$, $TC2_\alpha$ in $°C^{-2}$) and its standard deviation σ_α. Assuming that the reference temperature is 0 °C, the output of this block is then:

$$SF = x_t(normrnd(\alpha, \sigma_\alpha))(1 + TC1_\alpha T + TC2_\alpha T^2) \tag{2.3}$$

where function $normrnd(\alpha, \sigma_\alpha)$ generates a random number from the normal distribution with mean α and standard deviation σ_α. Note that if temperature of reference T_{ref} is different from 0 °C, T should be replaced by $T-T_{ref}$ in (2.3), (2.4) and (2.5). Also, temperature in °K instead of °C can be considered.

2. Nonlinearity. This block implements second-order and third-order nonlinearities of the scale factor using two coefficients c_1 (for x^2) and c_2 (for x^3). Both factors can be temperature-dependent and/or affected by random variations. For this block, the required parameters are c_1, c_2 and their linear temperature coefficients (TC_{c_1} and TC_{c_2} in $°C^{-1}$). Note that quadratic effects of the temperature and random variations are disregarded in this block but they could be easily implemented. The output of this block is then:

$$NL = c_1(1 + TC_{c_1} T)x_t^2 + c_2(1 + TC_{c_2} T)x_t^3 \tag{2.4}$$

3. Bias. This block adds a constant value to the input signal. This bias can be temperature-dependent and/or affected by random variations. Parameters required for this block are the

typical bias (β), given as an equivalent input signal (in Units) and typically equal to zero, standard deviation σ_β (in Units), and its temperature coefficients ($TC1_\beta$ in Units/°C^{-1}, $TC2_\beta$ in Units/°C^{-2}). The output of this block is then:

$$\text{Bias} = normrnd(\beta, \sigma_\beta) + TC1_\beta T + TC2_\beta T^2 \tag{2.5}$$

4. White Noise. Matlab Simulink block *Band-Limited White Noise* is used to generate white noise added to measurements. This block is parametrized using information from an ADEV graph (see 1.5.1), a PSD graph (see 1.4.1) or a datasheet. Consequently, the required inputs to generate a white noise sequence are:

ADEV graph / datasheet	PSD graph / datasheet
Q in Units $\cdot \sqrt{\text{s}}$	$S(f)_{wn}$ in Units^2/Hz assuming a two-sided PSD graph

Thus, by means of the Graphical User Interface (GUI) of sensor model (shown in Figure 2.4), *Band-Limited White Noise* block is fed using this information. For example, input *Noise power* specifies the power of a two-sided spectrum in a PSD graph. Therefore, relationship between this input and coefficient Q obtained from an ADEV is (see equation (1.7)):

$$Noise\ power = Q^2$$

In the case of $S(f)_{wn}$, this parameter is related to a two-sided PSD representation. Therefore, its relationship with the input *Noise power* is described by :

$$Noise\ power = S(f)_{wn}$$

Band-Limited White Noise block also requires a sampling time and the seed. This two parameters are assigned by default as:

Parameter	ADEV graph / datasheet	PSD graph / datasheet
Sampling frequency	(Bandwidth $\cdot 10)^{-1}$ (factor 10 is used as it is suggested in [55]).	
Seed	randi(9999).	

5. $1/f$ noise and $1/f^2$ Noise. For simulating these colored noises, the method presented in [24] is used. In general, this algorithm explains how colored noise can be generated by filtering a white noise sequence with an infinite impulse response (IIR) filter. Then, depending on the denominator of the transfer function, $1/f$ or $1/f^2$ noise is obtained. This method is presented in section 1.7, where it is explained in detail how *Gaussian Noise Generator* and *Discrete IIR filter* Matlab Simulink blocks (figure 1.11) can be used to obtain a colored noise sequence with a specific magnitude. In Appendix it is included the Matlab code of the method to simulate color noise. This code can replace the Matlab Simulink blocks if higher precision for simulating pink noise is required.

Typically, quantifications of $1/f$ and $1/f^2$ noises are carried out by means of ADEV graphs (section 1.5.1) or PSD graphs (section 1.4.1). Colored noise blocks can be parametrized with either of these graphs. Thus, the required inputs for the $1/f$ noise block are:

ADEV graph	PSD graph
B in Units	$S(f)_{pn}$ in Units^2/Hz assuming a two-sided PSD graph
Corner frequency with white noise (Hz)	
Seed: Default = randi(9999). It can be set up manually.	

For the $1/f^2$ noise block, input parameters are:

ADEV graph	PSD graph
K in Units$/\sqrt{\text{s}}$	$S(f)_{bn}$ in Units^2/Hz assuming a two-sided PSD graph
Sampling frequency of sensor's output (s)	
Seed: Default = randi(9999). It can be set up manually.	

6. Bandwidth. To limit the bandwidth, a first-order low-pass filter is used in the proposed sensor model but higher order filters could be implemented. The cut-off frequency of this filter can be the cut-off frequency of the sensor itself or, more likely, the cut-off frequency of the anti-aliasing filter included before the analog to digital converter. Therefore, the parameter of this block is the cut-off frequency given in Hz.

7. Saturation. *Saturation* Matlab Simulink block is used to set the sensor full scale (i.e. the measurement range of input signal). Parameters are upper and lower bounds given in equivalent input signal (in Units).

8. Sensitivity. This block permits to map sensor model measurements into electrical domain. This is carried out by applying a gain function which relates physical signal units to its electrical representation. Usually, this output is required in signal processing at hardware level. The output of this block is typically given in V or A but Ω, F and so on could be also used. Moreover, the output could be also converted in the time domain in the form of a variable frequency (Hz) or a variable duty cycle of a PWM signal for example.

9. Quantization. This block gives a discrete output to the system. For this, *Quantizer* Matlab Simulink block is used. The parameter required for this block is the quantization interval given in Units. The output of this block is also given in Units.

Figure 2.3 presents the Simulink implementation of the system model with all blocks introduced before. This model is the transformation of equation (2.2) into a Matlab tool that can be

used for behavioral simulation of a given sensor, whatever the nature of the physical input is. Numerical values of each parameter are set through the user interface of the system showed in figure 2.4. This tool can be accessed freely from [56].

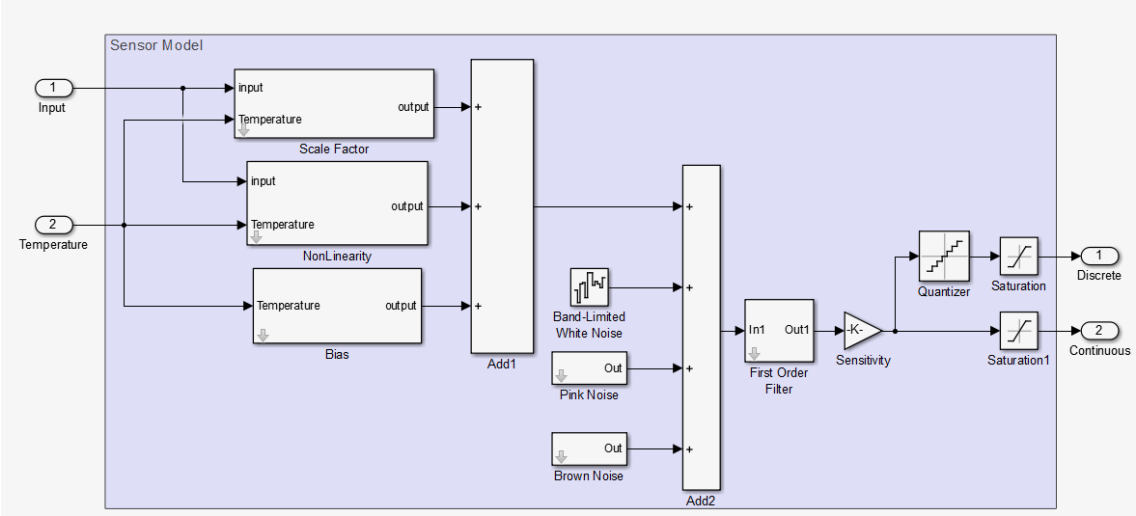


Figure 2.3 – Sensor model implemented in MATLAB Simulink.

2.4 Single sensor modeling

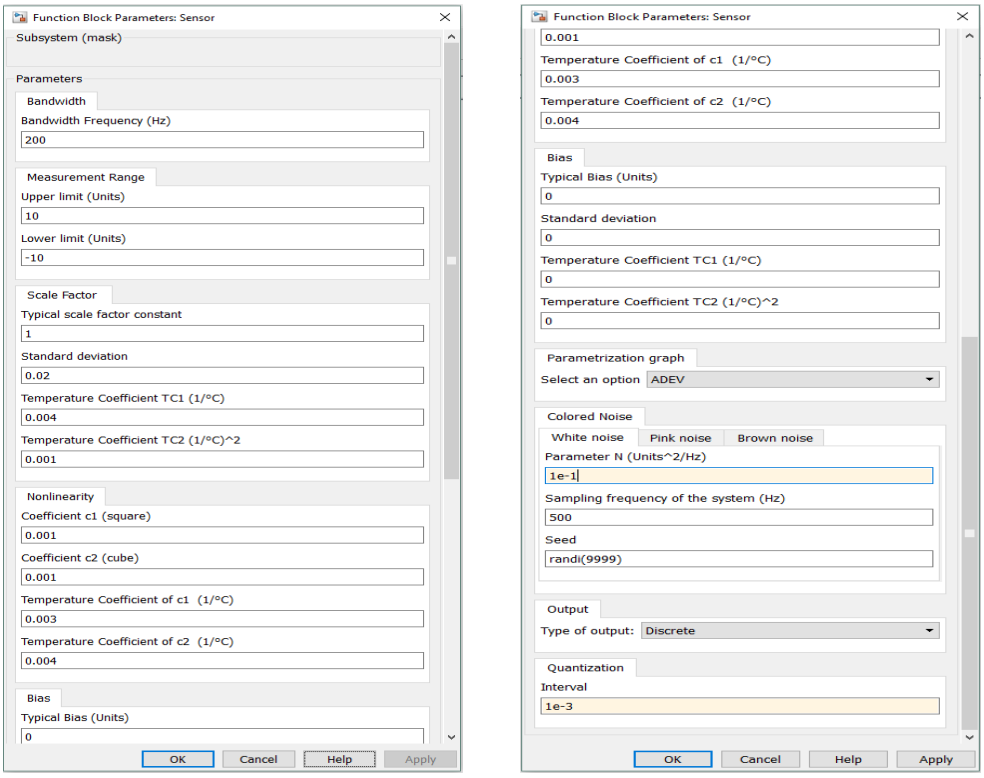
The proposed model can be customized to reflect the behavior of a given sensor using parameters extracted from its datasheet or experimental characterization results such as a PSD graph or an ADEV graph. Parameters from a datasheet can be implemented straightforwardly into the sensor model by using the graphical interface shown in figure 2.4. In the case of experimental data, an extraction procedure is required. Examples of how the sensor model may be populated from ADEV or PSD data are shown below.

2.4.1 Simulating a sensor from an ADEV graph

An example of using the sensor model for simulating an accelerometer is developed hereafter. The considered sensor is the accelerometer FXLN8372 from Freescale Semiconductor, Inc. (now NXP Semiconductors) [57]. For this example, the x-axis accelerometer with a range of $\pm 16g$ and a bandwidth of 1.1 kHz is considered.

For this case, information used to feed the sensor model comes from an ADEV graph. An analysis of the FXLN8372 output using the Allan variance method is carried out in [58] where the ADEV graph and thus, the parameters required by the sensor model can be found. Here, we present an example of how to extract white noise, $1/f$ noise, and $1/f^2$ noise parameters from the ADEV graph. This procedure is also explained in detail in section 1.5.1. To simplify our example, a copy of this ADEV graph is presented in figure 2.5. Results of this extraction are shown in table 2.1.

- Magnitude of white noise Q is measured by reading the slope $-1/2$ of the graph at $\tau = 1$s.



(a) (b)

Figure 2.4 – Graphical interface of proposed sensor model developed in Matlab Simulink environment.

Parameter Q is measured in Units $\cdot \sqrt{\text{s}}$. For this example, value $Q = 0.016\ (\text{m/s}^2) \cdot \sqrt{\text{s}}$ is read directly from Figure 2.5. This value is used as an input in the sensor model interface.

- Bias instability coefficient B (related to the $1/f$ noise) is measured by reading the flat region of the graph (slope $= 0$) and then, divide it by 0.664 (see section 1.5.1). This value is given in Units. For example, value $B = 0.0017/0.664\ \text{m s}^{-2}$ at $\tau = 1000\text{s}$ is read from Figure 2.5. These two values (B and τ) are used as inputs in the sensor model interface.

- Random walk coefficient K (related to the $1/f^2$ noise) is measured by reading the slope $1/2$ of the graph at $\tau = 3\text{s}$. Also, K can be extracted by reading the value at $\tau = 3 \cdot 60^2 = 10800\text{s}$ and then divide it by 60 (see section 1.5.1). This value is given in Units$/\sqrt{\text{s}}$. For this example, value $K = 0.004/60\ \text{m/s}^2/\sqrt{\text{s}}$ at $\tau = 10800\text{s}$ is read from Figure 2.5. This is used to populate the sensor model interface.

For this example just stochastic errors are of interest, therefore systematic parameters such as scale factor, bias, and nonlinearities are set up as typical values described in section 2.2. Also, a range of $\pm 16g$ and bandwidth of 1.1 kHz are set up in the interface, as well as a sensitivity gain of 1V/g just to simplify the analysis. Sensor's output is taken from the continuous output.

Simulations of sensor model are carried out, and then, ADEV graph is computed from 6×10^7 samples with a sampling time of 0.5 ms. For this analysis, AVAR Matlab function [59] is used. The

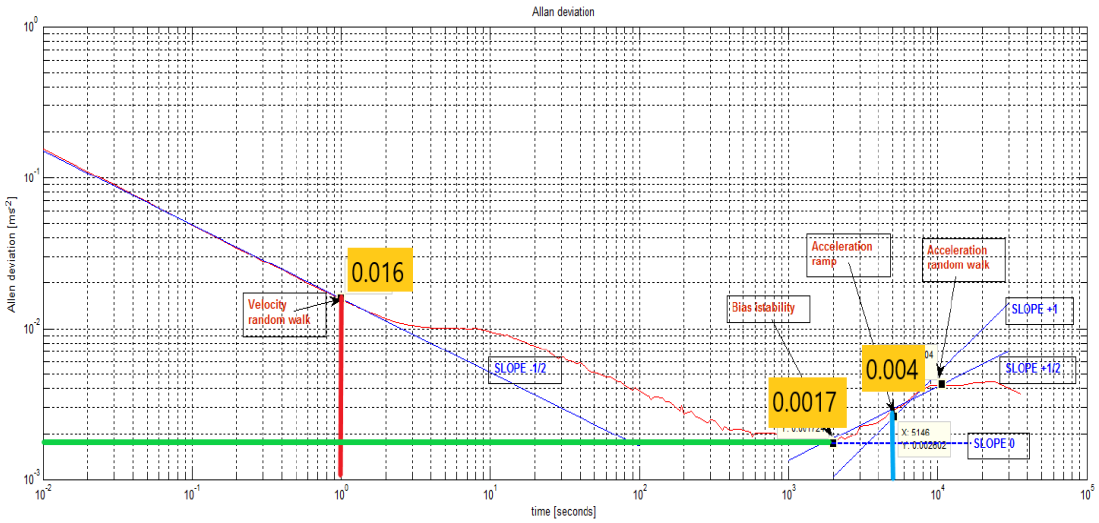


Figure 2.5 – Extraction of coefficients Q, B and K from ADEV graph of an accelerometer FXLN8372 [57].

result is the graph shown in Figure 2.6. Noise parameters are estimated from simulation results at $\tau = 1$s, $\tau = 1000$s and $\tau = 3$h and presented in Table 2.1. Comparing with parameters extracted from [58], it is possible to appreciate that the proposed sensor model is able to reproduce the behavior described by the ADEV graph.

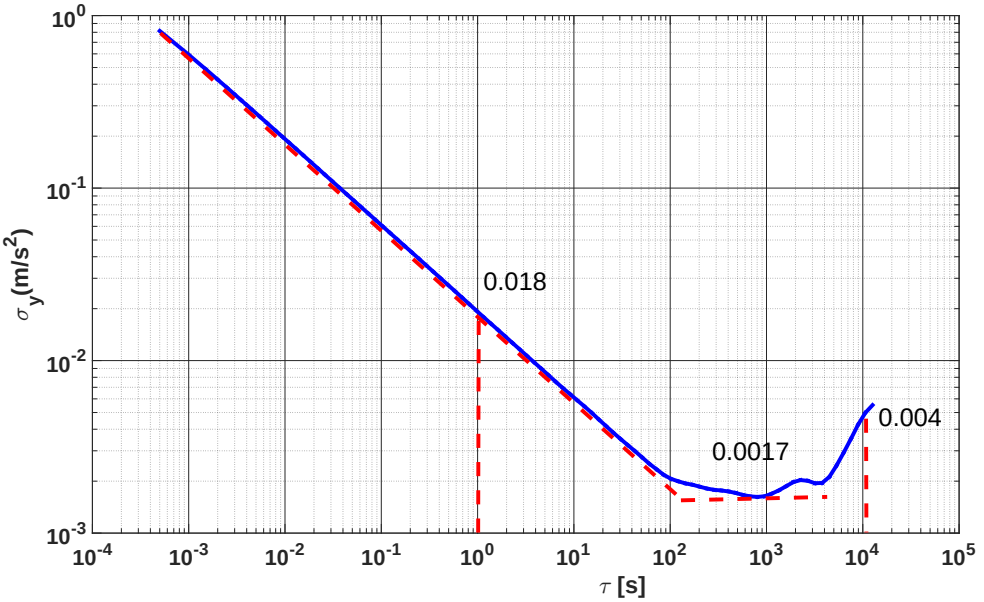


Figure 2.6 – ADEV graph extracted from simulations of accelerometer FXLN8372 [57]. Parameters for the sensor model were obtained from [58].

Table 2.1 – Assessment of sensor model parametrized from an ADEV graph.

	White Noise Slope $-1/2$ at $\tau = 1$s	$1/f$ Noise Slope 0 at $\tau = 1000$s	$1/f^2$ Noise Slope $1/2$ at $\tau = 3$h
Parameters extracted from [58]	$0.016\,(\mathrm{m/s^2})\cdot\sqrt{\mathrm{s}}$	$0.0017/0.664\,\mathrm{m\,s^{-2}}$	$0.004/60\,\mathrm{m/s^2}/\sqrt{\mathrm{s}}$
Results obtained by simulations using sensor model	$0.018\,(\mathrm{m/s^2})\cdot\sqrt{\mathrm{s}}$	$0.0017/0.664\,\mathrm{m\,s^{-2}}$	$0.004/60\,\mathrm{m/s^2}/\sqrt{\mathrm{s}}$

2.4.2 Simulating a sensor from a PSD graph

A second example of using the sensor model for simulating an accelerometer is developed hereafter. The considered sensor is an element of the Inertial Measurement Unit (IMU) 3/3/3 PhidgetSpatial Precision ID 1044_0 [60]. For this example, the accelerometer x-axis with a bandwidth of 500 Hz is considered.

In this case, the information used to feed the sensor model is a PSD graph. An analysis of the IMU 1044_0 is carried out in [61], where the one-sided PSD graph of the accelerometer can be found. Here, we present an example of how to extract white noise, $1/f$ noise, and $1/f^2$ noise parameters from this graph. This procedure is also explained in detail in section 1.4.1. To simplify our example, a copy of this PSD graph is presented in Figure 2.7 (the graph of interest is the blue one which corresponds to the accelerometer x-axis). Results of this extraction are shown in Table 3.1.

- Power of white noise $S(f)_{wn}$ is measured by reading the flat region of the graph. For example, value $S(f)_{wn} = 3\times10^{-7}\ (\mathrm{m/s^2})^2/\mathrm{Hz}$ is read directly from Figure 2.7. This value is divided by two (due to the graph is a one-sided representation) and then used as input parameter into the sensor model interface.

- Power of $1/f$ noise $S(f)_{pn}$ is measured by reading slope -1 (assuming a log-log plot) at 1Hz. For example, value $S(f)_{pn} = 1\times10^{-7}\ (\mathrm{m/s^2})^2/\mathrm{Hz}$ is read from Figure 2.7. This value is divided by two (because the graph is a one-sided representation) and used as input into the sensor model interface.

- Power of $1/f^2$ noise $S(f)_{bn}$ is measured from the slope -2 at 1Hz in a log-log plot. Also, $S(f)_{bn}$ can be obtained by reading the value at 1×10^{-2}Hz and then divide it by 100^2. For example, value $S(f)_{bn} = 5\times10^{-6}/100^2\ (\mathrm{m/s^2})^2/\mathrm{Hz}$ is read at 1×10^{-2}Hz. This value is divided by two (due to the graph is a one-sided representation) and used to populate the sensor model interface.

For this example just stochastic errors are of interest, therefore systematic errors such as scale factor, bias, and nonlinearities are set up as typical values described in section 2.2. A range of $\pm2g$ and bandwidth of 500 Hz are set up in the interface, as well as a sensitivity gain of $1\mathrm{V/g}$ just to simplify the analysis. Sensor's output is taken from the continuous output.

Simulations of the sensor model are carried out, where a PSD graph is computed from 3×10^7 samples with a sampling time of 0.2 ms. For spectral analysis, the function presented in Appendix is used. The result is the graph shown in Figure 2.8. To compare the desired behavior with the simulated by sensor model, it has been added dotted lines that correspond to parameters extracted from Figure 2.7. Analyzing the graph resulting from the sensor model with dotted lines, it is

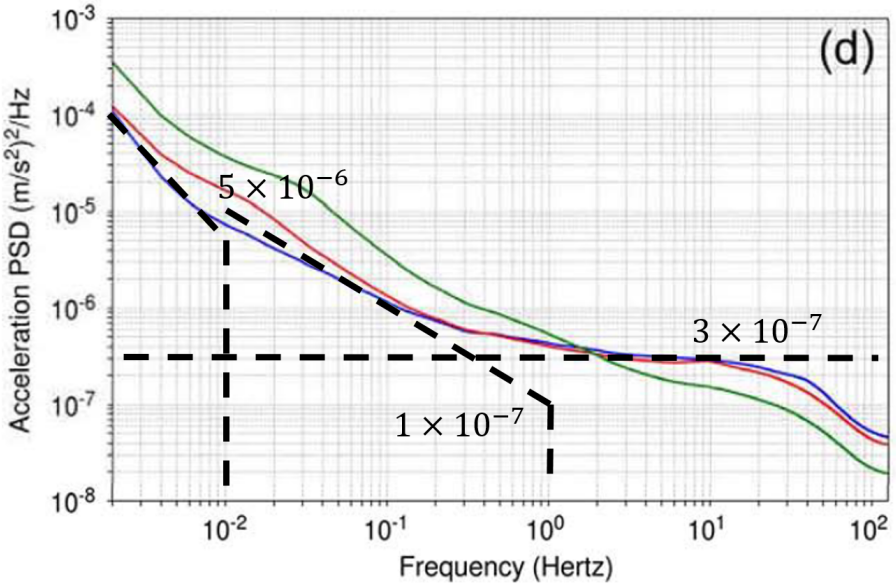


Figure 2.7 – Extraction of parameters PSD_{wn}, PSD_{pn} and PSD_{bn} from one-sided PSD graph of accelerometer ID 1044_0 [60].

possible to appreciate that the proposed sensor model is able to reproduce the behavior described by the PSD graph.

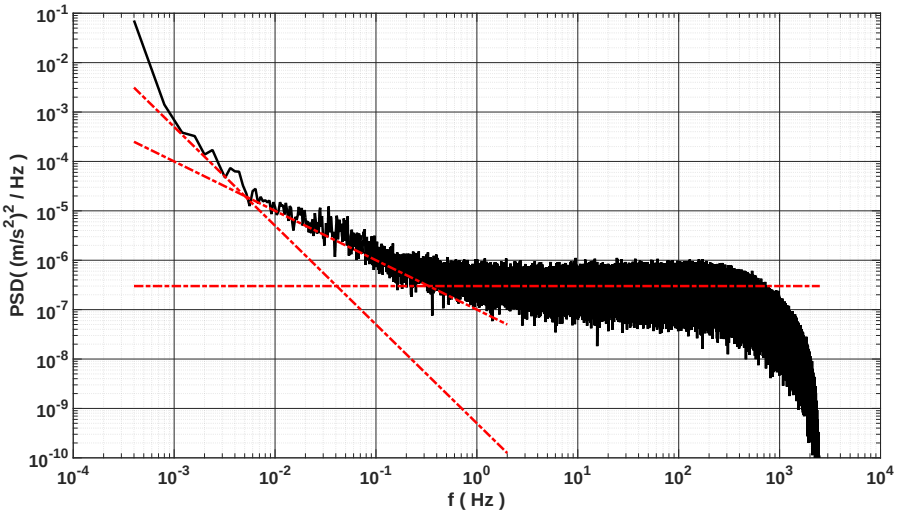


Figure 2.8 – One-sided PSD graph extracted from simulations of an accelerometer ID 1044_0 [60]. Parameters for the sensor model were obtained from the PSD graph presented in [61].

Table 2.2 – Parameters extracted from [61] and used to populate the sensor model interface.

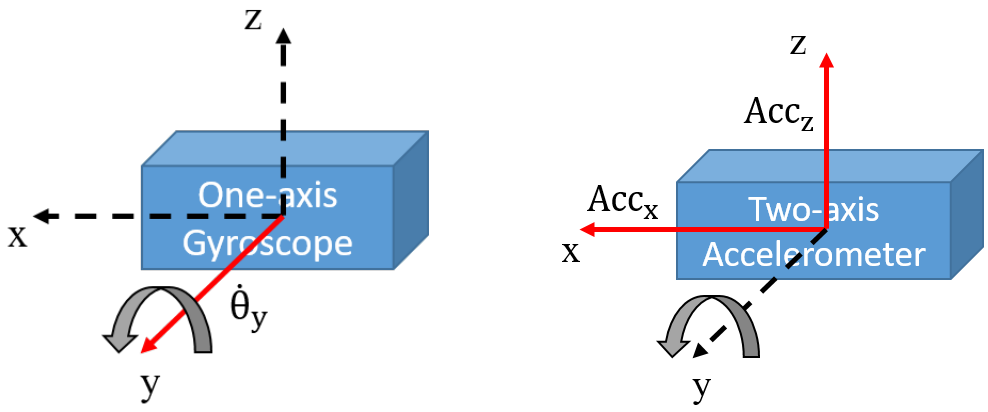
	White Noise	$1/f$ Noise		$1/f^2$ Noise
	PSD	PSD at 1 Hz	Intersection between Pink noise and White noise (Hz)	PSD at 1 Hz
Parameters extracted from [61] (One-sided PSD graph)	$3 \times 10^{-7}\,(m/s^2)^2/Hz$	$1 \times 10^{-7}\,(m/s^2)^2/Hz$	3×10^{-1} Hz	$5 \times 10^{-10}\,(m/s^2)^2/Hz$
Parameters used to populate sensor's model GUI	$1.5 \times 10^{-7}\,(m/s^2)^2/Hz$	$0.5 \times 10^{-7}\,(m/s^2)^2/Hz$	3×10^{-1} Hz	$2.5 \times 10^{-10}\,(m/s^2)^2/Hz$

2.5 Multisensor system modeling

Now, we present an example of the implementation of our sensor model for simulating a multisensor system. This implementation is based on the study presented in [62] about complementary filter and Kalman filter for tilt sensing. For this deployment, simulations of two different types of sensors are carried out by using the proposed sensor model. The problem statement is briefly described below.

Due to their nature, gyroscopes are used for tilt sensing. However, their drift causes a sharp increase in the orientation angle error, which is calculated by the integral of the gyro signal over time. On the other hand, accelerometers and trigonometric function relationship can be used to measure the tilt of an object in a static environment [63]. Contrary to gyroscopes, accelerometer output is stable and without long term drift, but at the same time, it is in general noisy and susceptible to external acceleration interference. Therefore, a tilt sensing using a fusion of both gyroscope and accelerometer advantages is generally used as an improved solution.

In [62] a solution for tilt sensing by merging gyroscope and accelerometer measurements is presented. Measurements are merged by using a complementary filter and a Kalman filter, and then results are compared. Hereafter, simulation of this application considering just one angle to measure is presented (from here on, this angle will be called θ). For this, only a one-axis gyroscope and a two-axis accelerometer are considered (Figure 2.9).



(a) (b)

Figure 2.9 – Configuration on the positions of sensors. A one-axis gyroscope (a) and a two-axis accelerometer (b) are used for simulations.

Estimations of θ from gyroscope and accelerometer outputs are obtained from equations (2.6)

and (2.7) respectively:

$$\hat{\theta}_t = \int \dot{\theta}_{y,t} dt \tag{2.6}$$

$$\hat{\theta}_t = \tan^{-1}\left(\frac{Acc_{x,t}}{Acc_{z,t}}\right) \tag{2.7}$$

2.5.1 Kalman filter for tilt sensing

Figure 2.10 shows the standard linear Kalman filter algorithm, where ϕ_t represents the state vector at step t, z_t is the measurement vector, u_t is the control input vector, F is the state transition matrix, H is the state-to-measurement matrix, B is the control matrix, P_t is the error covariance matrix, w_t is the state transition noise, v_t is the measurement noise, and Q_t and R_t are the covariance matrices related to those noises.

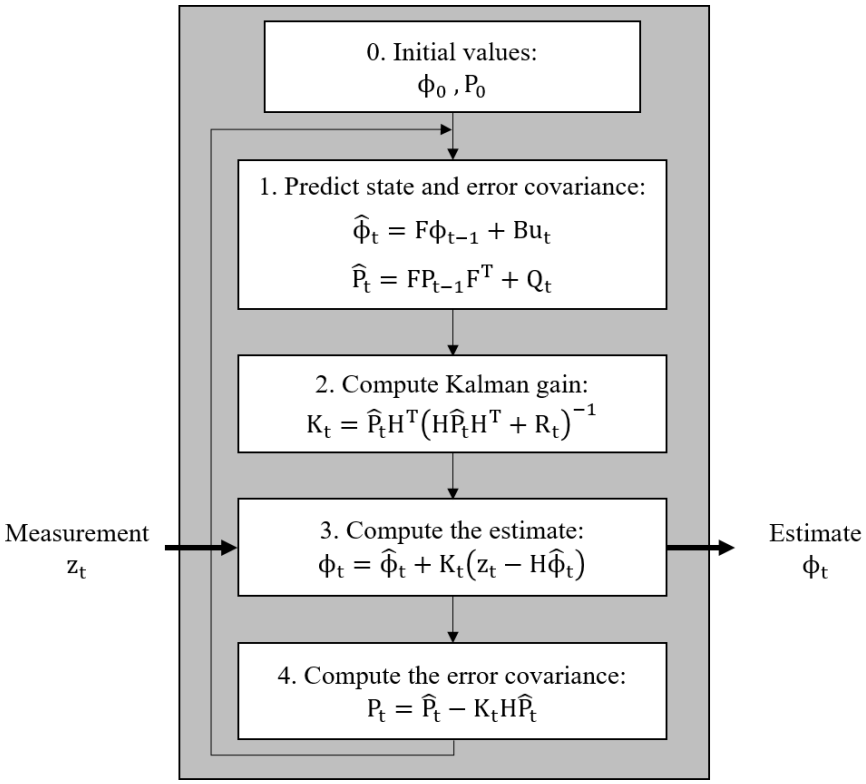


Figure 2.10 – Linear Kalman filter algorithm.

For the particular case of tilt sensing, the state vector at time t is defined as:

$$\phi_t = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix} \tag{2.8}$$

Where θ is the angle to estimate and $\dot{\theta}_b$ is the gyrometer bias (the amount that the gyroscope has drifted). The implementation of Kalman filter includes two main steps: prediction and update. In the predict process, the filter will first estimate the current state and the error covariance matrix

at time t. Equation (2.9) describes the estimation of the current state based on the previous state and the gyroscope measurement.

$$\phi_t = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \phi_{t-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_{y,t} \tag{2.9}$$

Where, $\dot{\theta}_{y,t}$ is the angular rate measured by the gyroscope at time t and Δt is the sampling time. Covariance matrices from state transition noise (matrix Q) and measurement noise (matrix R) are defined as:

$$Q_t = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \tag{2.10}$$

$$R_t = var(v_t) \tag{2.11}$$

Where Q_θ is the angle variance and $Q_{\dot{\theta}_b}$ is the bias variance. Next step is to estimate the a priori error covariance matrix $\hat{P}_t$ based on the previous error covariance matrix, i.e.:

$$\hat{P}_t = F P_{t-1} F^T + Q_t \tag{2.12}$$

Matrix $\hat{P}_t$ is used to indicate the reliability in the prediction of the state vector. Measurement vector is related with state vector by the matrix H. For our example of tilt sensing, measurement vector is defined as:

$$z_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \phi_t + v_t \tag{2.13}$$

Definition of the matrix H indicates that only accelerometer output is considered into the measurement vector. Before the update process, Kalman gain is computed as:

$$K_t = \hat{P}_t H^T (H \hat{P}_t H^T + R_t) \tag{2.14}$$

The update process consist in computing the difference between measurement vector z_t and the prediction of state vector $\hat{\phi}_t$:

$$\phi_t = \hat{\phi}_t + K_t (z_t - H \hat{\phi}_t) \tag{2.15}$$

Finally, update of the error covariance matrix is done by:

$$P_t = (I - K_t H) \hat{P}_t \tag{2.16}$$

More details about the configuration of the Kalman filter can be consulted in [62].

2.5.2 Complementary filter for tilt sensing

The complementary filter takes the advantage of both accelerometer and gyroscope. On the short term it uses the gyroscope measurement which is not susceptible to external forces, on the long term it relies on the accelerometer output to prevent measurement drift [62]. Complementary filter consists in filtering estimations of angle θ obtained from the accelerometer and the gyroscope with a low-pass filter and a high-pass filter respectively. The aim is to remove short term fluctuations

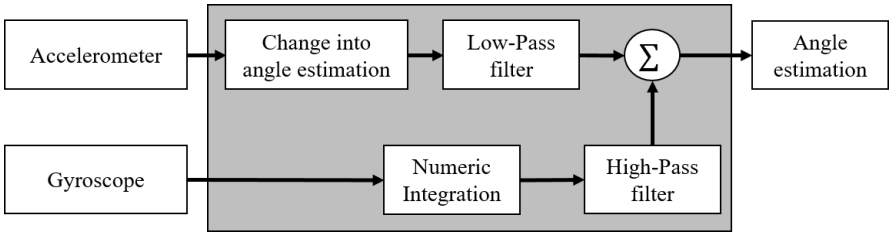


Figure 2.11 – Diagram of complementary filter.

of accelerometers and log-term fluctuations of the gyroscope. Then, both filtered estimations are added obtaining an improved estimation. Figure 2.11 illustrates this process.

Output of the complementary filter is given by:

$$\theta_t = \lambda(\theta_{t-1} + \dot{\theta}_{y,t}\Delta t) + (1-\lambda)\hat{\theta}_t \tag{2.17}$$

Where θ_t and θ_{t-1} are the angle estimations at time t and $t-1$ respectively, $\dot{\theta}_{y,t}$ is the angular rate measured by gyroscope at time t, Δt is the sampling time of the system, λ is the filter coefficient and $\hat{\theta}_t$ is the angle estimated from accelerometer measurements $Acc_{x,t}$ and $Acc_{z,t}$. Coefficient λ is determined by:

$$\lambda = \frac{\tau}{\tau + \Delta t} \tag{2.18}$$

Where τ is the time constant of the filter. Performance of complementary filter is strongly related to λ and therefore time constant τ, which tuning heavily relies on sensor performances.

2.5.3 Configuration of sensor model and simulations

The proposed sensor model is used for both accelerometers and gyroscope simulation. To carry out these simulations, three sensor-model-blocks are implemented, two blocks for the accelerometers (one for each axis), and one block for the gyroscope.

Because stochastic errors are the main interest in this example, it is assumed that all sensors are pre-calibrated. Thus, only noise configuration is adjusted into sensor-model-blocks. For all sensors, a bandwidth of 200Hz is set up. For accelerometers, a range of $\pm 16\ \mathrm{m\,s^{-2}}$ is used. In the case of the gyro, a range of 400 °/s is set up. In all sensors, a sensitivity of 1V/Unit is set up to simplify the analysis. The rest of parameters used for those blocks are presented in Table 2.3.

Table 2.3 – Gyroscope and accelerometer noise parameters

Gyroscope axis	Angle random walk	Bias instability	Rate random walk
Y	$0.01\,(^\circ/\mathrm{s})\cdot\sqrt{\mathrm{s}}$	$5\times10^{-3}\,^\circ/\mathrm{s}$	$6\times10^{-6}\,^\circ/\mathrm{s}/\sqrt{\mathrm{s}}$
Accelerometer axis	Velocity random walk	Bias instability	Rate random walk
X	$0.05\,(\mathrm{m/s^2})\cdot\sqrt{\mathrm{s}}$	$0.004\,\mathrm{m/s^2}$	$1\times10^{-5}\,\mathrm{m/s^2}/\sqrt{\mathrm{s}}$
Z	$0.02\,(\mathrm{m/s^2})\cdot\sqrt{\mathrm{s}}$	$0.003\,\mathrm{m/s^2}$	$6\times10^{-5}\,\mathrm{m/s^2}/\sqrt{\mathrm{s}}$

For the angle θ, we consider an oscillation between 2.09 rad and -2.09 rad with a frequency of 1 rad/s. The sample time at the sensor's output and the total simulation time are 2 ms and 50 s, respectively.

Figure 2.12 shows estimations of angle θ given by the accelerometers and the gyroscope, as well as the complementary and Kalman filters. It can be noted that gyroscope and accelerometers outputs are mainly affected by drift and white noise respectively. Complementary filter is implemented with $\lambda = 0.85$. For Kalman filter, $Q_\theta = 1 \times 10^{-5}$, $Q_{\dot{\theta}_b} = 1 \times 10^{-3}$ and $R = 1 \times 10^{-2}$ are found to be the best options for this particular case. The comparison of all the resulting estimations in terms of the Root Mean Squared Error (RMSE) is presented in Table 2.4.

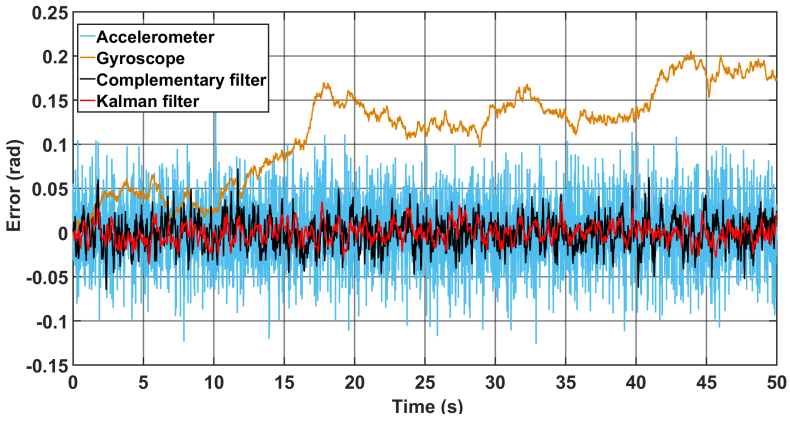


Figure 2.12 – Error in estimation of angle θ. Assessment of Complementary filter and Kalman filter.

Table 2.4 – Performance in terms of the RMSE.

	Gyroscope	Accelerometer	Complementary Filter	Kalman Filter
Root Mean Squared Error	0.18 rad	0.012 rad	0.0068 rad	0.0053 rad

The presented results illustrate the efficiency of both filters to improve the angle estimation, as well as the usefulness of the model to evaluate the performances of data fusion algorithms. Perhaps more complex modifications can be done to the complementary filter and/or the Kalman filter in order to obtain better performances, however to find the optimal solution is not the goal of this exercise. The proposed sensor model show to be an excellent tool for simulations of different sensors at system level. As shown in this example, the inclusion of low-frequency noises in our sensor model permits the analysis of stochastic errors such as bias instability, which is nowadays the main drawback present in MEMS gyroscopes.

Chapter 3

State of the art - Data fusion algorithms for sensor arrays

In this chapter, we present a study of different algorithms for data fusion in sensor array systems existing in the literature. These algorithms are generic, which means that their implementation is not constrained for the nature (i.e., the type) of sensors. The selected algorithms for the study are: the blind calibration algorithm, the least squares method, single-layer artificial neural networks, Kalman filter, and the random weighting method. For a better understanding, these algorithms are evaluated by means of simulations inside the Matlab Simulink environment. For these simulations, the sensor model presented in chapter 2 is used. At the end of this chapter, a summary with a comparative table between the analyzed algorithms is presented.

3.1 Introduction

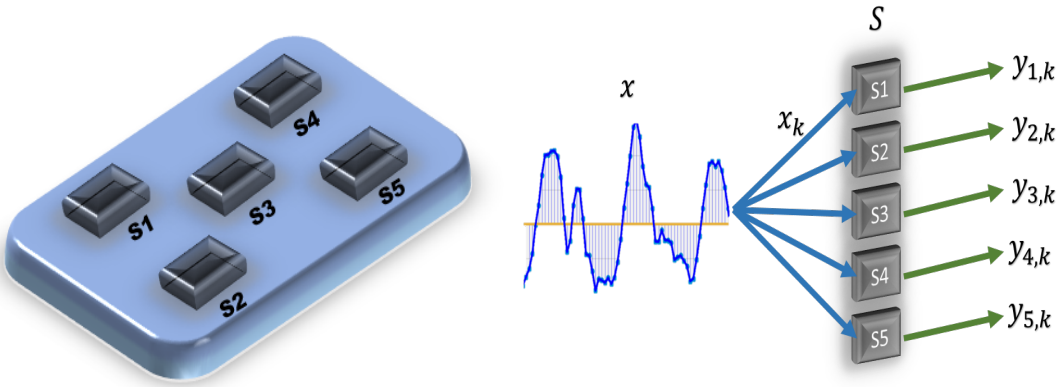
From [64]: *"data fusion in multi sensor systems is defined as the use of techniques that combine data from multiple sensors to achieve improved accuracies and more specific inferences than those that could be obtained by using a single sensor*. In a sensor array system, several sensors are used to measure the same properties, resulting in a high degree of redundancy in the obtained measurements. Since noise in observations of each sensor are independent of each other, redundancy can be used to reduce system uncertainty, enlarge its range of observation, improve its precision, and enhance its reliability. This can be used to create systems composed of several low-performance low-cost sensors that can offer similar performance levels that a single high-performance high-price sensor. Further, because of redundancy, an array system may still operate even when one or more of its sensors stop working, extending its lifetime.

In the following sections, we present a study of different algorithms for data fusion in sensor array systems. The idea is to analyze different methods existing in the literature in order to identify advantages / disadvantages proposed by each of them. The experience acquired throughout this study was used as a basis for the formulation of the proposed algorithm presented in the next chapter.

First, we define a generic system and a simulation environment that will be used for the study and evaluation of these algorithms.

3.2 Generic sensor array system

Let's consider a discrete-time sensor array system S composed of $n = 5$ sensors that measure the same input x, all of them with the same intensity. The type of sensors in S has not been defined, this is done with the intention of generalizing the analysis performed throughout this chapter. It should be mentioned that the constraint that all sensors measure x with the same intensity is not directly met for all existing array systems. For example, in an array of accelerometers, measurements are linked to positions of sensors in the array, as explained in [31]. In cases like this, the constraint can be fulfilled by using some mathematical transformations. In addition, there are many other cases where this constraint is valid, for example in arrays of gyroscopes [31]. System S is illustrated in Figure 3.1.



(a) Sensor array system composed of 5 same-type sensors.

(b) All sensors in the system measure signal x with the same intensity.

Figure 3.1 – Generic sensor array system for simulations.

Let's define x_k as the value of signal x at time step k, and let $y_{i,k}$ be the measurement reported by sensor i at the same time step. Relationship between the input and output of a sensor is generally described as:

$$y_{i,k} = (\alpha_T + \alpha_i)x_k + \beta_i + \varepsilon_{i,k} \tag{3.1}$$

where:

- α_T is the typical value for the scale factor (usually normalized to one and assumed to be constant)
- α_i is the scale factor error in sensor i (typical value equal to zero and assumed to be constant)
- β_i is the bias of sensor i (typical value equal to zero and assumed to be constant)
- $\varepsilon_{i,k}$ denotes the noise present in sensor output (white noise, flicker noise, brown noise, etc.)

If sensor i is calibrated, then the values of its systematic errors (scale factor error and bias) are known and corrected, leaving only the presence of stochastic effects affecting their outputs. So, assuming $\alpha_T = 1$, corrected measurement reported by sensor i at time step k is defined as:

$$y_{i,k} = x_k + \varepsilon_{i,k} \tag{3.2}$$

For all simulations that will be carried out in this chapter, the presence of stochastic errors is assumed. However, not all the algorithms presented here can work under the presence of systematic errors. So, to evaluate performances and characteristics of algorithms under analysis, we decided to create two different scenarios. Such scenarios are:

- Uncalibrated sensors. It is considered that measurements of sensors are affected by systematic and stochastic errors. Therefore, behavior of each device is described by equation (3.1).
- Calibrated sensors. It is assumed that just probabilistic phenomena affect sensors' measurements. Behavior of each device is described by equation (3.2).

3.3 Simulation environment

Simulations of system S are carried out using the sensor model presented in chapter 2. For these simulations, sensors GUIs are populated using parameters presented in Table 3.1. The implementation of this multi sensor system into Matlab Simulink environment is illustrated in Figure 3.2, where a golden device (ideal sensor) is used as a reference to quantify errors present in measurements coming from sensors, as well as the errors in estimates generated from any of the data fusion algorithms.

Table 3.1 – Parameters set up for sensors in system S.

Parameter	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
Bandwidth	500Hz				
Sample time	1ms				
Range	$\pm$10 Units				
Scale factor ($\alpha_T - \alpha_i$) (1 if calibrated)	1.0050	1.0100	0.9600	0.9300	0.9000
Bias β_i in Units (0 if calibrated)	-0.1000	-0.1300	-0.2700	0.3500	-0.4000
$S(f)_{wn}$ in Units^2/Hz	5×10^{-3}	1×10^{-3}	5×10^{-4}	1×10^{-4}	5×10^{-5}
$S(f=1)_{pn}$ in Units^2/Hz	5×10^{-3}	1×10^{-3}	5×10^{-4}	1×10^{-4}	5×10^{-5}
Intersection between $1/f$ noise and White noise (Hz)	1 Hz				
$S(f=1)_{bn}$ in Units^2/Hz	5×10^{-9}	1×10^{-9}	5×10^{-10}	1×10^{-10}	5×10^{-11}

From Table 3.1 it can be noted that noise level in sensors is assigned in such a way that it is decreasing according to the sensor tag-number. For example, sensor 1 has the highest noise level while sensor 5 has the lowest. Thus, for the scenario where sensors are calibrated, sensor 5 will offers the best estimation. On the other hand, levels of systematic errors affecting the scale factor and bias, are increasingly assigned according to the sensor tag-number. Thus, sensor 1 presents the lowest values for systematic errors, while sensor 5 presents the highest values.

For all simulations, a total time of 100 s is considered, where sample time at the output of each sensor is 1ms. Therefore, each simulation generates 10^5 samples for each sensor. In addition, it is assumed that sampling at the output of all sensors is synchronized, which means that all sensors report measurements from the same time step at the same time.

Finally, for the input signal we decided to use a non-periodic function. The chosen input is a white noise signal with values within the range of $\pm$10 Units. For this, we use the Matlab Simulink block *Signal Generator* which is able to simulate random signals with all their values within a certain range. This is equivalent to passing the output of a white noise generator through a *Saturation* block. The desired signal is generated with the following parameters: 1) Wave form: random

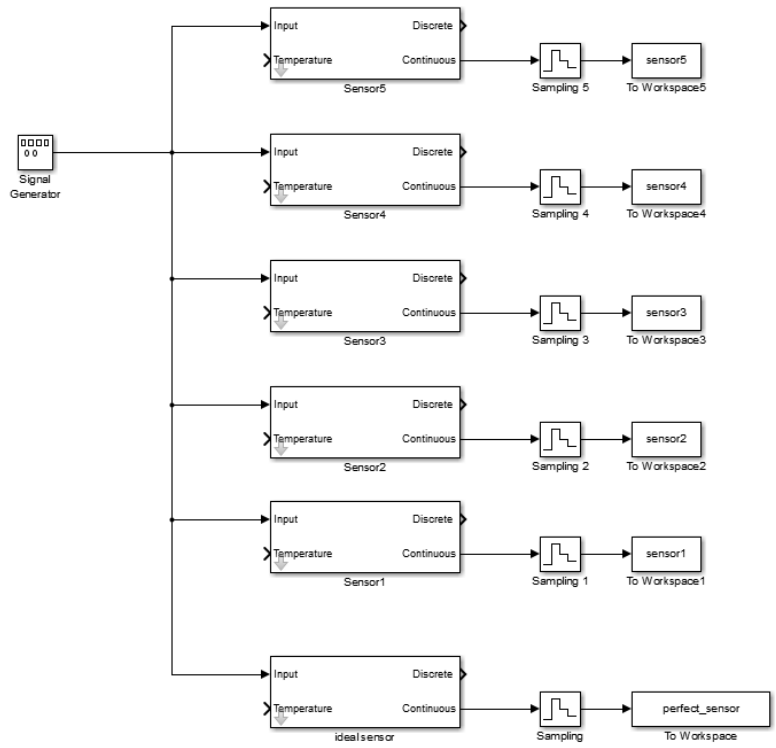


Figure 3.2 – Implementation of generic sensor array system into Matlab Simulink environment.

signal, and 2) Amplitude: 10 Units. The result is a white noise signal with a variance equal to 5.78 Units^2 and whose values are all contained within the range of ± 10 Units.

3.4 Algorithms to evaluate

Once defined the environment for simulations, we now proceed to define algorithms that will be evaluated throughout this chapter. All chosen algorithms are generic, i.e., they do not depend on the nature of the sensors. This feature is essential for the objectives of this thesis since we seek to create a solution applicable to almost any sensor array system, regardless of the type of sensors.

The chosen algorithms for this study are: blind calibration algorithm, the least squares method, single-layer artificial neural network, Kalman filter, and the random weighting method. As mentioned before, all these algorithms were designed to counteract the presence of stochastic errors, however, not all of them are able to work under the presence of systematic errors. The blind calibration algorithm (as its name indicates) is specially designed for scenarios where sensors are uncalibrated. The Kalman filter and the random weighting method are made for implementations where sensors are calibrated, while the least squares method and artificial neural networks can be implemented in both scenarios. This is illustrated in Figure 3.3.

In addition, some of these algorithms require the knowledge of the input signal (see Figure 3.3). Let's define a learning phase as a set of time steps in which the input signal x is known. For the algorithms that require a learning phase, data obtained from the first 10 s of simulation are used for their implementation. The remaining data (from 10 s to the end of the simulation) is used for

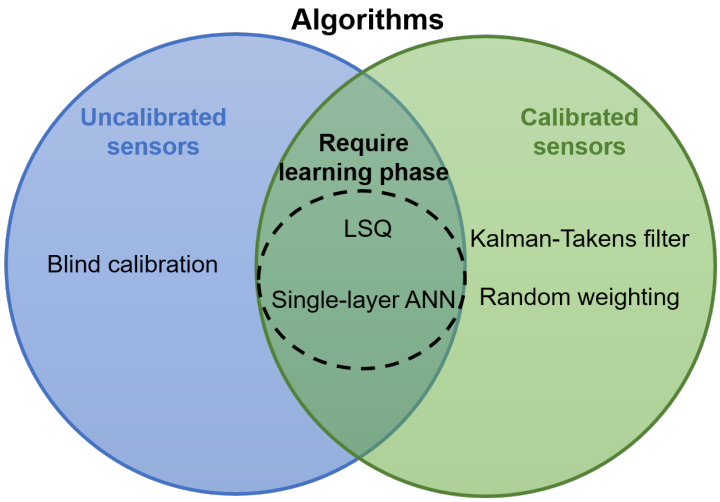
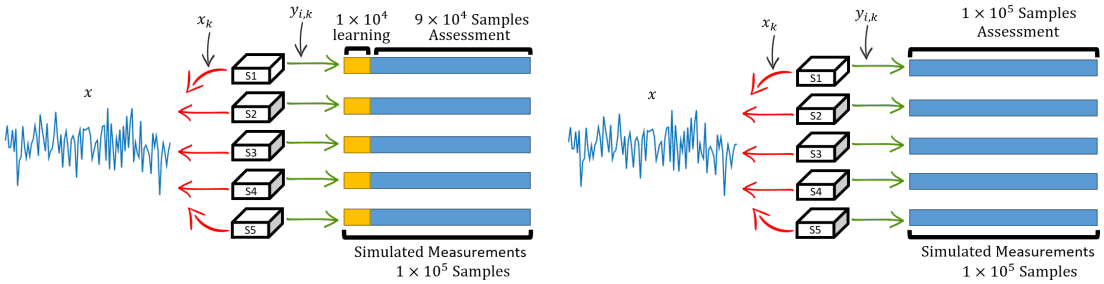


Figure 3.3 – Diagram of evaluated algorithms.

their evaluation. In the case of algorithms that do not require a learning phase, the assessment is carried out from the first measurement until the end of the simulation. Figure 3.4 illustrates this.

Finally, to quantify the error we compute the Root Mean Square (RMS) error (in Units) between x and the estimation given by each algorithm. The Matlab code used for computing the RMS error can be consulted in [65]. The order in which studied algorithms are presented is the following: blind calibration, least squares method, single-layer artificial neural network, Kalman filter, and random weighting method. At the end of this chapter, a summary with a comparative table between these algorithms is presented.



(a) Algorithm with a learning phase.

(b) Algorithm without learning phase.

Figure 3.4 – Illustration of how data from simulations is used to implement and evaluate algorithms.

3.5 Blind calibration algorithm

The first algorithm to analyze is the blind calibration method. This algorithm was designed for dynamic calibration of sensors. It does not require the knowledge of the input signal and requires only the storage of a small number of variables. Consider system S described in section 3.2, but now, assume that S measures a set of r different signals, and let's call this set as X. At a given instant

k, each sensor makes a measurement of $X_k = [x_{1,k}, \dots, x_{r,k}]^T$, where vector of n measurements is denoted as $Y_k = [y_{1,k}, \dots, y_{n,k}]^T$. Relationship between Y_k and X_k is described by:

$$\begin{bmatrix} y_{1,k} \\ \vdots \\ y_{n,k} \end{bmatrix} = \begin{pmatrix} H_{1,1} & \cdots & H_{r,1} \\ \vdots & & \vdots \\ H_{1,n} & \cdots & H_{r,n} \end{pmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_r \end{bmatrix} \begin{bmatrix} (\alpha_T - \alpha_1) \\ \vdots \\ (\alpha_T - \alpha_n) \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,k} \\ \vdots \\ \varepsilon_{n,k} \end{bmatrix} \tag{3.3}$$

This can be expressed in a matrix form as follows:

$$Y_k = H X_k A + \beta + \varepsilon_k \tag{3.4}$$

Where H is the observation matrix of size $n \times r$, $A = [(\alpha_T - \alpha_1), \dots, (\alpha_T - \alpha_n)]^T$, and A_i is used to denote $(\alpha_T - \alpha_i)$. It is important to note that H is different from A; H corresponds to the intensity with which sensors receive input signals, while A corresponds to the scale factor coefficients linked to the physical properties of sensors.

The blind calibration algorithm proposed in [66] is based on the algebraic concept of orthogonal projection. To explain this, let's reformulate equation (3.4) as follows:

$$H X_k = \frac{Y_k - \beta - \varepsilon_k}{A} \tag{3.5}$$

Assume that the number of sensors in S is bigger than the number of measured signals r, (i.e., $n > r$); this implies that H lies in a r-dimensional subspace of $\mathcal{R}^n$. Let $H^\perp$ the orthogonal complement of H, and let P be the orthogonal projection matrix onto $H^\perp$, therefore:

$$PH = 0 \tag{3.6}$$

Multiplying equation (3.5) by P we obtain:

$$0 = P(Y_k - \beta - \varepsilon_k)A^{-1} \tag{3.7}$$

Now, assuming that $\frac{1}{k}\sum_{\nu=1}^{k} \varepsilon_{i,\nu} \to 0$ when $k \to \infty$, from (3.7) it can be derived the following relationship:

$$P\beta = P\bar{Y} \tag{3.8}$$

where $\bar{Y} = \frac{1}{k}\sum_{j=1}^{k} Y_j$. Now, equation (3.7) can be expressed as:

$$P(Y_k - \bar{Y})A^{-1} = 0 \tag{3.9}$$

This is the main result presented in [66], where under certain assumptions, scale factor coefficients A and constant biases β can be estimated from equations (3.9) and (3.8), respectively. These estimations can be carried out each time step without any knowledge about X.

On the other hand, matrix $P(Y_k - \bar{Y})$ has a rank $n - r$, and therefore, for estimating A and β a group of r constraints must be defined. For example, it can be assumed that S contains r calibrated sensors. Another option is to use this algorithm to evaluate the performance of all sensors considering a given reference, which avoids the need for a perfect estimation of A and β. In this work, we will focus on the scenario described in section 3.2 where all sensors are uncalibrated and measure a single signal x with the same intensity. Next, we show how to implement blind calibration algorithm under these conditions.

3.5.1 Blind calibration for data fusion in sensor array systems

Let's retake definition of system S given in section 3.2. Implementation of blind calibration algorithm is performed by adapting equations (3.8) and (3.9) to the system S. First, we define vectors Y_k and $\bar{Y}$ as follows:

$$Y_k = [y_{1,k}, \dots, y_{n,k}]^T \tag{3.10}$$

$$\bar{Y} = \left[\sum_{i=1}^{k} y_{1,k}, \dots, \sum_{i=1}^{k} y_{n,k}\right]^T \tag{3.11}$$

Due to the fact that all sensors measure a single signal, H results in a $n \times 1$ vector where all its elements are ones.

$$H_{n\times 1} = [1, \dots, 1]^T \tag{3.12}$$

Consequently, H and $H^{\perp}$ lie in subspaces of size 1 and $n-1$, respectively. First, we find basis U for the subspace spanned by $H^{\perp}$. This process is shown below:

$$\begin{aligned}
H &= [1, \dots, 1]^T \\
H^{\perp} &= \{u = [u_1, \dots, u_n]^T \in \mathcal{R}^n | H^T u = 0\} \\
H^T u = u &\quad \therefore\ u = 0 \\
u_1 &= -u_2 - \dots - u_n \qquad \%\text{Choose } u_1 \text{ as independent variable, then find special solutions} \\
U_1 &= [-1, 1, 0, \dots, 0]^T \quad \%\text{Special solution: } u_2 = 1, u_3 = 0 \dots\ u_n = 0 \\
U_2 &= [-1, 0, 1, \dots, 0]^T \quad \%\text{Special solution: } u_2 = 0, u_3 = 1 \dots\ u_n = 0 \\
&\vdots \\
U_n &= [-1, 0, 0, \dots, 1]^T \quad \%\text{Special solution: } u_2 = 0, u_3 = 0 \dots\ u_n = 1 \\
U &= [U_1, U_2, \dots, U_n] = [[-1, \dots, -1]_{1\times n-1}; I_{n-1}]
\end{aligned} \tag{3.13}$$

Now, orthogonal projection matrix P can be computed as follows:

$$P_{n\times n} = U(U^T U)^{-1} U^T \tag{3.14}$$

Due to the definition of H, matrix P can also be expressed in function of n, as below [65]:

$$P_{n\times n} = \begin{cases} 1 & , \text{for } P_{i,i} \\ \frac{-1}{n-1} & , \text{for } P_{i,j} \mid i \neq j \end{cases} \tag{3.15}$$

Rank of matrix $P(Y_k - \bar{Y})$ is $n-1$, which means that by adding a single constraint it can become invertible. For example, to estimate scale factors it can be assumed that at least one known sensor in S is calibrated (or at least, it can be used as a reference). For estimation of biases, it can be assumed that the average of these biases converge to the typical value. These constrains are not so far from realistic scenarios; sometimes in sensor array systems different grades of sensors are used, and then, the most reliable sensor can be chosen as a reference. Also, same-type sensors usually have the same typical values. Thus, the more sensors a system has, the more congruent are the

assumptions that average of all sensors results in the convergence of systematic errors into their typical values. Once formulated, constraints are added to matrix $P(Y_k - \bar{Y})$ giving it a full rank n.

To add such constraints, matrix $P(Y_k - \bar{Y})$ is changed by $P(\text{diag}(Y_k) - \text{diag}(\bar{Y}))$ as shown below:

$$P(\text{diag}(Y_k) - \text{diag}(\bar{Y})) = \begin{bmatrix} 1 & \cdots & \frac{-1}{n-1} \\ \vdots & \ddots & \vdots \\ \frac{-1}{n-1} & \cdots & 1 \end{bmatrix} \left(\begin{bmatrix} y_{1,k} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & y_{n,k} \end{bmatrix} - \begin{bmatrix} \sum_{i=1}^{k} y_{1,i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sum_{i=1}^{k} y_{n,i} \end{bmatrix} \right) \tag{3.16}$$

Now, we add a constraint which gives the solution to the system of equations. For this, sensor 1 is used as a reference, i.e., it is assumed that $\alpha_1 = 0$, and therefore $A_1 = 1$, which results in $A_1^{-1} = 1$. This constraint is added to matrix $P(\text{diag}(Y_k) - \text{diag}(\bar{Y}))$, which now is a $(n+1) \times n$ matrix, and more important, the right side is not more the null space (i.e., different of zero). Hence, using equation (3.9), estimation of the inverse of scale factor vector at time step k is computed as follows:

$$\begin{bmatrix} [1, 0, \dots, 0] \\ P(\text{diag}(Y_k) - \text{diag}(\bar{Y})) \end{bmatrix} \begin{bmatrix} A_1^{-1} \\ \vdots \\ A_n^{-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.17}$$

Accuracy in estimation of the inverse of scale factor vector depends on the constraint given in (3.17). In this case, it depends on how close is the scale factor of sensor 1 to the typical value. If $A_1^{-1} \neq 1$, then estimation of scale factor obtained for a given sensor i (with $i \neq 1$) will be:

$$\hat{A}_i^{-1} \approx A_i^{-1} * A_1^{-1} \tag{3.18}$$

Where the approximation symbol is due to the presence of noise in sensors measurements. On the other hand, it is important to point out that (3.17) computes $\hat{A}^{-1}$. If needed, scale factor vector can be obtained as $\hat{A} = \left[1/\hat{A}_1^{-1}, \dots, 1/\hat{A}_n^{-1}\right]$.

Now, vector β is estimated by using equation (3.8). Again, the definition of a constraint is needed. To this case, we use the assumption: $\frac{1}{n}\sum_{i=1}^{n} \beta_i = 0$ (i.e., the average of biases converges to the typical value). Thus, estimation of biases at time step k is computed as follows:

$$\begin{bmatrix} \left[\frac{1}{n}, \dots, \frac{1}{n}\right] \\ P \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} = \begin{bmatrix} 0 \\ P\bar{Y} \end{bmatrix} \tag{3.19}$$

This concludes the implementation of this algorithm in system S, where equations (3.17) and (3.19) are used each time step to estimate parameters A and β, respectively. Once calculated, these parameters are used to correct individual sensor measurements, and then, estimation of the measured signal at time step k is performed by averaging corrected measurements, as shown below:

$$\hat{x}_k = \frac{1}{n} \sum_{i=1}^{n} \hat{A}_i^{-1} (y_{i,k} - \hat{\beta}_i) \tag{3.20}$$

3.5.2 Simulations

To evaluate this implementation, simulations of the scenario with uncalibrated sensors described in section 3.3 are carried out. Note that this algorithm does not require any learning phase, and therefore, the assessment is executed from the beginning to the end of the simulation. Matlab code for this implementation is included in Appendix.

Table 3.2 shows estimated scale factor vector. Note that sensor 1 has the lowest scale factor error in the system, and therefore, it leads to the best calibration. To corroborate this, we execute a second time the blind calibration algorithm, but this time using sensor 2 as the reference. Results of estimation of scale factor vector are also shown in Table 3.2. Here, we observe that error in estimation $\left(\sum |\hat{A}_i - A_i|\right)$ using sensor 2 is bigger than the one obtained using sensor 1, as expected. In addition, note that estimation of β_i does not depend on the reference sensor (equation (3.19)), and therefore, it is the same in both cases.

Table 3.2 – Blind calibration of sensor array system.

Parameter	Reference	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
A_i		1.0050	1.0100	0.9600	0.9300	0.9000
$\hat{A}_i$	Sensor 1	1.0000	1.1119	0.9540	0.9466	0.9575
$\sum \lvert\hat{A}_i - A_i\rvert$		0.1869				
$\hat{A}_i$	Sensor 2	1.2989	1.0000	0.9839	0.9861	0.9885
$\sum \lvert\hat{A}_i - A_i\rvert$		0.4723				
β_i		-0.1000	-0.1300	-0.2700	0.3500	-0.4000
$\hat{\beta}_i$		0.0978	-0.0370	-0.1875	0.4391	-0.3124

Finally, we observe that even when sensor 1 (sensor reference) has a scale factor close to 1, error in estimation of scale factor vector is high. This is mainly due to the high noise level present in sensor 1.

Table 3.3 shows the RMS error in the signal estimation obtained with this implementation. To compare this result, we include in this table the RMS error obtained with calibrated sensors (from the scenario where sensors are calibrated), as well as the simple average of all of them. In theory, error obtained with the blind calibration should be the same as error obtained with the average of calibrated sensors. Nevertheless, due to the imprecision in the constraint used in (3.17), error in signal estimation is higher than expected. Even, we observe that errors in sensors 4 and 5 are lower than the one from the blind calibration. Here, we pointed out that error obtained with this algorithm depends completely on the accuracy of constraint used to full rank the system of equations presented in (3.17).

3.5.3 Discussion

This algorithm was originally created for calibrating dynamically groups of sensors which oversample a group of signals. One of the main strengths of this algorithm is that it can be executed without any knowledge about input signal. Also, estimations of parameters α and β can be computed each time step, which is a desirable feature in dynamic applications where parameters change constantly. Another advantage of this algorithm is the low computational complexity, which is related to matrix operations required to find parameters α and β, resulting in a complexity of $\mathcal{O}(n^3)$ [67].

Table 3.3 – Assessment of blind calibration algorithm.

Signal estimator	Sensor reference	RMS Error (Units)
		Scenario: Uncalibrated sensors
Blind Calibration algorithm	Sensor 1	0.4228
	Sensor 2	0.4333
		Scenario: Calibrated sensors
Average		0.4095
		Scenario: Calibrated sensors
Sensor 1		1.7694
Sensor 2		0.7831
Sensor 3		0.5820
Sensor 4		0.2680
Sensor 5		0.1957

On the other hand, the biggest disadvantage of this algorithm is that it is not a data fusion method, and therefore, its performance (in terms of the RMS error) is limited to the method used for merging the corrected data. For example, in this implementation a simple average was used for this task ; this implies that the best performance that the algorithm can achieve is the same as the average of sensors from the calibrated scenario. Another disadvantage is its dependency on constraints, whose number is linked to the number of measured signals. This disadvantage comes with the fact that every assumption added to the system brings an unknown error. This error results from the difference between the constraint assumed and the true value.

As a partial conclusion, this method is useful only if a golden device is available in the array system.

3.6 The Least Squares (LSQ) method

The second method to analyze is the Least SQuares (LSQ) method, which is an algorithm that, unlike the previous one, can be used for data fusion in both scenarios: calibrated and uncalibrated sensors. This is because the LSQ is a method used to determine the best fit line (or polynomial function) to a set of given data [68]. Suppose a data set x be the input of a system, and a data set y be its output. Then, for the given data $\{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$, the relationship between x and y can be approximated by

$$y = ax + b \tag{3.21}$$

Even if (3.21) is a very simple relationship, it is widely used to approximate the behavior of a system, especially in the field of sensors. If higher grade functions are required to fit given data, then equation (3.21) can be generalized as

$$y = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_2 x^2 + a_1 x + b \tag{3.22}$$

3.6.1 LSQ for calibration of sensors

Let's analyze the simplest form of LSQ method, using the equation (3.21). Then, error associated to the fit line is

$$E(a,b) = \sum_{k=1}^{\ell} (y_k - (ax_k + b))^2 \tag{3.23}$$

Note that this error is the sum of the square of distances between data points $\{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ and line described in (3.21). The sum of these distances is equal to ℓ times the variance of data set $\{y_1 - (ax + b), \ldots, y_\ell - (ax_\ell + b)\}$. The goal of LSQ algorithm is to find values a and b that minimize this error. This minimization related to variables (a, b) is carried out as follows:

$$\frac{\partial E}{\partial a} = 0, \ \frac{\partial E}{\partial b} = 0$$

Differentiating $E(a, b)$ yields:

$$\frac{\partial E}{\partial a} = \sum_{k=1}^{\ell} -2(y_k - (ax_k + b)) \cdot x_k$$

$$\frac{\partial E}{\partial b} = \sum_{k=1}^{\ell} -2(y_k - (ax_k + b))$$

Then, setting $\partial E/\partial a = \partial E/\partial b = 0$ and dividing by -2 yields:

$$\sum_{k=1}^{\ell} (y_k - (ax_k + b)) \cdot x_k = 0$$

$$\sum_{k=1}^{\ell} (y_k - (ax_k + b)) = 0$$

It is possible to rewrite these equations as:

$$a \sum_{k=1}^{\ell} x_k^2 + b \sum_{k=1}^{\ell} x_k = \sum_{k=1}^{\ell} x_k y_k$$

$$a \sum_{k=1}^{\ell} x_k + b \sum_{k=1}^{\ell} 1 = \sum_{k=1}^{\ell} y_k$$

This can be written in a matrix form as:

$$\begin{pmatrix} \sum_{k=1}^{\ell} x_k^2 & \sum_{k=1}^{\ell} x_k \\ \sum_{k=1}^{\ell} x_k & \ell \end{pmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{\ell} x_k y_k \\ \sum_{k=1}^{\ell} y_k \end{bmatrix}$$

As the first matrix is invertible (see proof in [68]), coefficients a and b can be estimated as:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{pmatrix} \sum_{k=1}^{\ell} x_k^2 & \sum_{k=1}^{\ell} x_k \\ \sum_{k=1}^{\ell} x_k & \ell \end{pmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{\ell} x_k y_k \\ \sum_{k=1}^{\ell} y_k \end{bmatrix} \tag{3.24}$$

Thus, the best fit values of a and b are obtained by solving the linear system of equations presented in (3.24). This is the LSQ method for linear fitting. The same calculations can be carried out for a higher grade function.

LSQ method presented in (3.24) can be implemented to estimate the relationship between the input and output of a sensor. Thus, given a data set $\{(x_1, y_{i,1}), \dots, (x_\ell, y_{i,\ell})\}$ where x_ℓ denotes the value of the input signal at step ℓ, $y_{i,\ell}$ denotes the measurement reported by sensor i corresponding to such value, and ℓ is the size of what we have previously defined as a learning phase, estimation of systematic errors (scale factor error and bias) presented in this sensor can be computed using the solution above. The way to do this is explained in detail in section 3.6.3.

3.6.2 Data fusion in sensor array systems using LSQ method

The previous result can be used to correct the systematic errors in all the sensors and then merge the corrected measurements. For the data fusion a simple average could be executed, however, this would be equivalent to what was done in the previous section with the blind calibration algorithm. To avoid this, a different implementation of the LSQ method is used for the data fusion process in both scenarios: 1) uncalibrated sensors once that measurements are corrected, and 2) calibrated sensors.

The main idea is to estimate the best way to merge measurements coming from n calibrated (or corrected) sensors. We can do this by means of a weighted average, where individual weights are related to individual performance of sensors. Next equation describes the weighted average process, where w_i denotes the assigned weight to sensor i:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}^T \approx x \tag{3.25}$$

where, $\sum_{i=1}^{n} w_i = 1$. Consider the given set of data $\{(x_1, y_{1,1}, \dots, y_{n,1}), \dots, (x_\ell, y_{1,\ell}, \dots, y_{n,\ell})\}$. The goal is to fix values of individual weights which minimize the following error:

$$E(w_1, \dots, w_n) = \sum_{k=1}^{\ell} (x_k - (y_{1,k} w_1 + \dots + y_{n,k} w_n))^2 \tag{3.26}$$

Observe that due to the assumption that all sensors are calibrated, then scale factors are normalized to 1 and a constant bias to 0. Now, differentiating equation (3.26) yields:

$$\begin{aligned} \frac{\partial E}{\partial w_1} &= \sum_{k=1}^{\ell} -2y_{1,k}(x_k - y_{1,k} w_1 - \dots - y_{n,k} w_n) \\ &\vdots \\ \frac{\partial E}{\partial w_n} &= \sum_{k=1}^{\ell} -2y_{n,k}(x_k - y_{1,k} w_1 - \dots - y_{n,k} w_n) \end{aligned} \tag{3.27}$$

Setting $\partial E/\partial wi = 0 \ \ \forall i | 1 \leq i \leq n$, and rearranging the expression above results in

$$\begin{aligned}
\sum_{k=1}^{\ell} y_{1,k}^2 w_1 + \sum_{k=1}^{\ell} y_{1,k} y_{2,k} w_2 + \ldots + \sum_{k=1}^{\ell} y_{1,k} y_{n,k} w_n &= \sum_{k=1}^{\ell} x_k y_{1,k} \\
\sum_{k=1}^{\ell} y_{2,k} y_{1,k} w_1 + \sum_{k=1}^{\ell} y_{2,k}^2 w_2 + \ldots + \sum_{k=1}^{\ell} y_{2,k} y_{n,k} w_n &= \sum_{k=1}^{\ell} x_k y_{2,k} \\
&\vdots \\
\sum_{k=1}^{\ell} y_{n,k} y_{1,k} w_1 + \sum_{k=1}^{\ell} y_{n,k} y_{2,k} w_2 + \ldots + \sum_{k=1}^{\ell} y_{n,k}^2 w_n &= \sum_{k=1}^{\ell} x_k y_{n,k}
\end{aligned} \tag{3.28}$$

Expressed equation (3.28) in a matrix form results in

$$\begin{pmatrix}
\sum_{k=1}^{\ell} y_{1,k}^2 & \sum_{k=1}^{\ell} y_{1,k} y_{2,k} & \cdots & \sum_{k=1}^{\ell} y_{1,k} y_{n,k} \\
\sum_{k=1}^{\ell} y_{1,k} y_{2,k} & \sum_{k=1}^{\ell} y_{2,k}^2 & \cdots & \sum_{k=1}^{\ell} y_{2,k} y_{n,k} \\
\vdots & \vdots & & \vdots \\
\sum_{k=1}^{\ell} y_{1,k} y_{n,k} & \sum_{k=1}^{\ell} y_{2,k} y_{n,k} & \cdots & \sum_{k=1}^{\ell} y_{n,k}^2
\end{pmatrix}
\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} =
\begin{bmatrix} \sum_{k=1}^{\ell} x_k y_{1,k} \\ \sum_{k=1}^{\ell} x_k y_{2,k} \\ \vdots \\ \sum_{k=1}^{\ell} x_k y_{n,k} \end{bmatrix} \tag{3.29}$$

Consequently, values of weights that minimize the error between input signal x and the estimation can be computed by solving the following system of equations:

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} =
\begin{pmatrix}
\sum_{k=1}^{\ell} y_{1,k}^2 & \sum_{k=1}^{\ell} y_{1,k} y_{2,k} & \cdots & \sum_{k=1}^{\ell} y_{1,k} y_{n,k} \\
\sum_{k=1}^{\ell} y_{1,k} y_{2,k} & \sum_{k=1}^{\ell} y_{2,k}^2 & \cdots & \sum_{k=1}^{\ell} y_{2,k} y_{n,k} \\
\vdots & \vdots & & \vdots \\
\sum_{k=1}^{\ell} y_{1,k} y_{n,k} & \sum_{k=1}^{\ell} y_{2,k} y_{n,k} & \cdots & \sum_{k=1}^{\ell} y_{n,k}^2
\end{pmatrix}^{-1}
\begin{bmatrix} \sum_{k=1}^{\ell} x_k y_{1,k} \\ \sum_{k=1}^{\ell} x_k y_{2,k} \\ \vdots \\ \sum_{k=1}^{\ell} x_k y_{n,k} \end{bmatrix} \tag{3.30}$$

Here, the existence of the inverted matrix can be easily proved by means of the Lagrange's identity.

Then, for both scenarios the data fusion process is carried out by means of a weighted average, using equation (3.30) to compute the corresponding weights. However, as mentioned before, this is not directly applicable for the scenario where sensors are uncalibrated. For this scenario a previous step for the individual calibration of sensors using equation (3.24) is required. Once all sensors are calibrated, data fusion using the weighted average can be executed. For both implementations of the LSQ (calibration and weights' estimation) the same learning phase is used. Figure 3.5 illustrate this.

3.6.3 Simulations

LSQ method can be implemented only if inputs and outputs are known. This is the main disadvantage of this method, since in some applications it is really hard (and even sometimes impossible) to have knowledge about the measured input. This requirement is equivalent to the availability of a golden device without noise at least during the learning phase. For this exercise it is assumed that

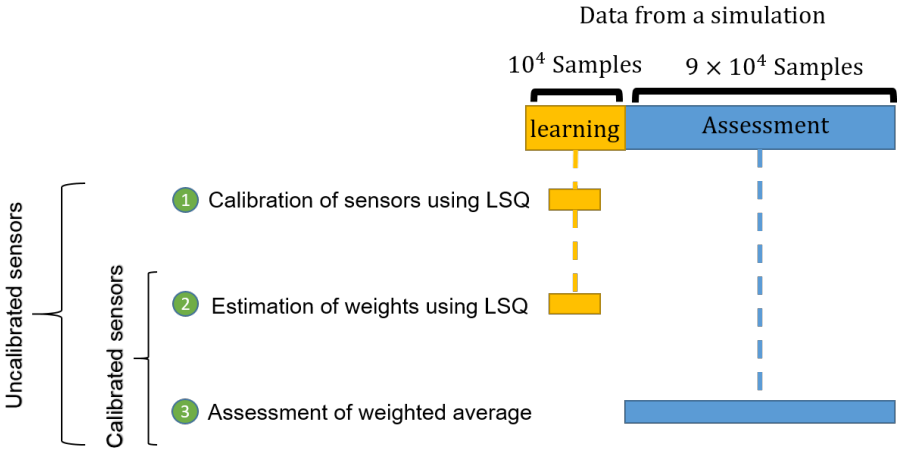


Figure 3.5 – Map of the implementation of the LSQ method.

inputs are known during the learning phase, and therefore, implementation of LSQ method can be carried out. Data from the learning phase is used for the individual calibration of sensors (equation (3.24)) and the estimation of weights (equation (3.30)), as illustrated in Figure 3.5.

Calibration for the scenario with uncalibrated sensors

First, we present the implementation of LSQ for the scenario with uncalibrated sensors. We start with the estimation and correction of systematic errors present in all sensors. Equation (3.24) is used for computing parameters a_i and b_i, as shown below:

$$\begin{matrix}\{(x_1, y_{1,1}), \ldots, (x_\ell, y_{1,\ell})\} \\ \vdots \\ \{(x_1, y_{n,1}), \ldots, (x_\ell, y_{n,\ell})\}\end{matrix} \xrightarrow[\text{sensor}]{\text{For each}} \begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{pmatrix} \sum_{k=1}^{\ell} x_k^2 & \sum_{k=1}^{\ell} x_k \\ \sum_{k=1}^{\ell} x_k & \ell \end{pmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{\ell} x_k y_{i,k} \\ \sum_{k=1}^{\ell} y_{i,k} \end{bmatrix} \tag{3.31}$$

Table 3.4 shows the values of a_i and b_i estimated with LSQ method. Note that a_i and b_i correspond to $(\alpha_T - \alpha_i)$ and β_i in equation (3.1). Comparing values a_i and b_i with parameters presented in Table 3.1 it can be seen that estimates given by LSQ method are (in general) accurate. Observe that accuracy in the estimation of these parameters increase as the tag-number of sensors increases. For example, parameters estimated for sensor 5 are more accurate than the ones estimated for sensor 1. This is because measurements reported by sensor 5 contain a lower level of noise than those measurements reported by sensor 1.

Data fusion using weighted average

Now, the LSQ is used to implement a weighted average for data fusion in both scenarios: 1) uncalibrated sensors once that measurements are corrected using (3.31), and 2) calibrated sensors. Implementation are carried out using equation (3.30), where individual sensors' weights are computed as follows:

Table 3.4 – Parameters a_i and b_i obtained using the LSQ method and their comparison with the true values.

Parameter	Parameter	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
Scale factor	a_i	0.9960	1.0084	0.9590	0.9309	0.9006
	A_i	1.0050	1.0100	0.9600	0.9300	0.9000
	$\lvert A_i - a_i \rvert$	0.0090	0.0016	0.0010	0.0009	0.0006
Bias	b_i	-0.0268	-0.1017	-0.3244	0.3475	-0.3959
	β_i	-0.1000	-0.1300	-0.2700	0.3500	-0.4000
	$\lvert \beta_i - b_i \rvert$	0.0732	0.0283	0.0544	0.0025	0.0041

$$\begin{matrix}\{(x_1, y_{1,1}), \ldots, (x_\ell, y_{1,\ell})\} \\ \vdots \\ \{(x_1, y_{n,1}), \ldots, (x_\ell, y_{n,\ell})\}\end{matrix} \xrightarrow[\text{data set}]{\text{For the whole}} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{pmatrix} \sum_{k=1}^{\ell} y_{1,k}^2 & \sum_{k=1}^{\ell} y_{1,k}y_{2,k} & \cdots & \sum_{k=1}^{\ell} y_{1,k}y_{n,k} \\ \sum_{k=1}^{\ell} y_{1,k}y_{2,k} & \sum_{k=1}^{\ell} y_{2,k}^2 & \cdots & \sum_{k=1}^{\ell} y_{2,k}y_{n,k} \\ \vdots & \vdots & & \vdots \\ \sum_{k=1}^{\ell} y_{1,k}y_{n,k} & \sum_{k=1}^{\ell} y_{2,k}y_{n,k} & \cdots & \sum_{k=1}^{\ell} y_{n,k}^2 \end{pmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{\ell} x_k y_{1,k} \\ \sum_{k=1}^{\ell} x_k y_{2,k} \\ \vdots \\ \sum_{k=1}^{\ell} x_k y_{n,k} \end{bmatrix} \tag{3.32}$$

Table 3.5 shows the obtained weights for both scenarios. Note that in both cases, assigned weights correspond to individual performance of sensors. Thus, sensor 1 (sensor with the highest noise level) has the lowest weight, while sensor 5 (sensor with the lowest noise level) obtains the highest weight.

Table 3.5 – Estimated weights using LSQ method.

Parameter	Scenario	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
Weight	Calibrated sensors	0.0058	0.0301	0.0614	0.2903	0.6123
	Uncalibrated sensors	0.0088	0.0324	0.0658	0.3064	0.5806

Once individual weights are estimated, estimation of signal x at time step k is computed as follows:

$$\hat{x}_k = \sum_{i=1}^{n} w_i \cdot y_{i,k} \tag{3.33}$$

RMS error due to the estimation of x is presented in Table 3.6. Observe that error in the scenario with calibrated sensors is slightly lower than the one from the scenario with uncalibrated sensors. This was expected since calibration by means of LSQ is not perfect. Also, note that for both scenarios the estimation error is lower than the one obtained with the previous algorithm, and even is lower than the error presented by the less noisy calibrated sensor, i.e. sensor 5.

Discussion

From the implementation of the LSQ algorithm in both scenarios we noted that, in both cases, the error obtained was lower than the obtained with the blind calibration algorithm. This was expected

Table 3.6 – Assessment of LSQ method.

Sensor / Data fusion method	RMS Error (Units)
	Scenario: Calibrated sensors
Sensor 1	1.7694
Sensor 2	0.7831
Sensor 3	0.5820
Sensor 4	0.2680
Sensor 5	0.1957
	Scenario: Calibrated sensors
Average	0.4095
	Scenario: Calibrated sensors
Weighted average	0.1360
	Scenario: Uncalibrated sensors
Weighted average	0.1491

since in this case we used a weighted average instead of a simple average for the data fusion process. Even so, the estimation of systematic errors using the LSQ was more accurate than the estimation using the blind calibration, however, unlike blind calibration, for the LSQ the knowledge of the input is required. Moreover, for the LSQ once the parameters or weights of the sensors have been estimated, they cannot be updated unless a new learning phase is executed. This was not the case for the blind calibration method, whose estimates are dynamically updated.

Nevertheless, Table 3.6 shows how by implementing LSQ method in both scenarios (calibrated and uncalibrated sensors) an improvement in the input estimate can be obtained. In fact, it is important to point out that LSQ method is a Best Linear Unbiased Estimator (BLUE) [68]. Therefore, assuming unbiased measurements (without the presence of systematic errors), the implementation of this method obtains the best possible estimate, and hence, the best system performance.

Finally, a question that arises is whether it is possible to unify the two processes carried out with the LSQ (calibration and estimation of weights) in a single one. From equation (3.30) it seems that this cannot be done with the LSQ method. However, there are some other methods that can be used for this, such as the next algorithm: artificial neural networks.

3.7 Artificial Neural Network (ANN)

The Artificial Neural Networks (ANNs) are an inspiration from the human brain. To our knowledge, biological neural networks work as it is described in [69]: *"human brain comprises of as many as* 10^{11} *biological neurons. Each of the biological neuron is connected to the other neurons making massive* 10^{22} *connections between the various neurons"*. Each of the biological neuron is an information processing unit in itself, where all neurons operate in parallel [70]. They take their input from the body inputs or from the other neurons. This input is processed, and then, the resulted information is transmitted to other neurons. The information receipted by the body is continuously processed by the various neurons one after the other. Then, after a large number of iterations, the final output is given.

In the case of ANN, the main objective is to reproduce as faithfully as possible the process

carried out by biological neural networks. ANNs consist of a set of artificial neurons. As explained in [69], the various artificial neurons are joined or connected to each other in order to transmit the flow of information or data between the neurons. In this manner the information or data given to the system as inputs is processed again and again by the various neurons and the results are exchanged. The output of one neuron becomes the input for the other neurons. This process goes on and finally the computed answer is returned by the network.

The task of any fundamental artificial neuron may be divided into two parts. The first part calculates the weighted average of its inputs. Here each connection has its own weight. As the input arrives through the connection, it is multiplied by the corresponding weight. Then, the addition of all such inputs is performed. The second part of the neuron consists of an activation function. The weighted average of the first part is passed through the activation function. This is the final output of the system. The activation function is usually non-linear to enable the ANNs to solve non-linear problems. Figure 3.6 exemplifies procedure described above.

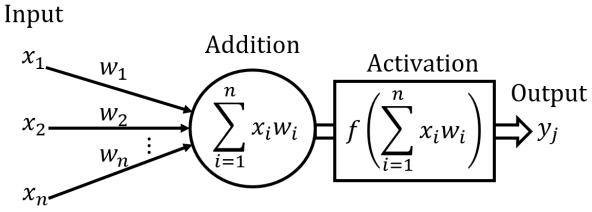


Figure 3.6 – Example of an artificial neuron.

Numerous models of ANNs exist in literature. They all use different fundamentals of problem solving. Many of these models use supervised or reinforcement learning whereas the others use unsupervised learning. Some are suited to the functional prediction problems, whereas the other are more suited towards the classification problems. In this work, the multi layer perceptron network has been chosen for data fusion in our defined sensor array system. This network architecture is widely used in applications of all kinds, due to its simplicity. There are more complex network models, however, the use of this ANN permits a first analysis of the feasibility of implementing this kind of algorithms in embedded sensor systems. Next, the theoretical foundation of this architecture is presented.

3.7.1 Multi Layer Perceptron (MLP)

From our analogy of the biological neuron we know that an artificial neuron (also known as perceptron) does the task of taking weighted sum of the inputs and passing it through an activation function [69]. A single neuron is usually not able to solve the problem, especially if the problem is non-linear in nature. This requires the use of multiple neurons in a layered architecture one after the other. This model is called as Multi Layer Perceptron (MLP) and is presented in Figure 3.7. This architecture is also known as backpropagation, this is because of the algorithm used for the learning technique.

A MLP is a feed forward network, i.e., flow of information has a single way. In this model, perceptrons are arranged in the form of layers. The first layer is called the input layer, where the number of neurons of this layer corresponds to the number of inputs to the network. The input layer is passive, i.e. it does not implement any kind of information processing. Similarly, the last layer in the network is called the output layer, where the number of neurons in this layer corresponds

to the number of outputs. Unlike the input layer, the output layer is active, i.e. it does the task of information processing. All the layers between the input and the output layer are called hidden layers. Each neuron in these layers perform the task of processing. Neurons from a hidden layer m are connected with all neurons from the $m+1$ and $m-1$ layers. They take the inputs from the $m-1$ layer and give the computed output to the $m+1$ layer. In this way, computation of the information takes place starting from the input layer to the output layer through the hidden layers.

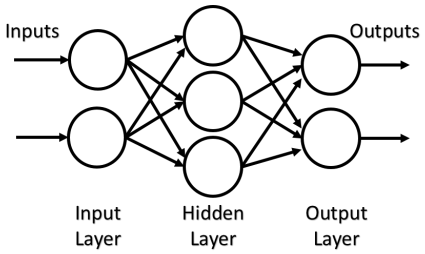


Figure 3.7 – Example of a multilayer perceptron network.

Each circle in the output and hidden layers showed in Figure 3.7 is an artificial neuron that does the task of information processing. Each connection between two neurons has a weight associated with it. Each neuron first computes the weighted average of all its inputs and then applies the activation function. Equation that describes this procedure is the following:

$$y_i = f\Big(\sum_{i=1}^{n} x_i w_i + b\Big) \tag{3.34}$$

Here, n denotes the number of inputs to the neuron, x_i denotes the values of those inputs, f is the activation function, b is the bias added as an extra input, and y_i is the output of this neuron. For performance reasons, it is always preferred to keep all input and outputs of the ANN in the range of ± 1.

3.7.2 Data fusion in sensor array systems using MLP

Let's take again the definition of the sensor array system S given above. To implement a MLP neural network to merge data coming from all sensors in S, the first step is to define inputs and outputs of our network. Here, inputs will be the vector of sensors' measurements at time step k, i.e., $y_k = [y_{1,k}, \dots, y_{n,k}]^T$, while the network output is the estimation of signal at the same time step, i.e., $\hat{x}_k$.

Next, it is necessary to define the data set used for training the network. Data obtained during the learning phase of the simulation is used for this purpose. On the one hand, inputs for the learning algorithm are sensors' measurements $Y = [y_1, \dots, y_\ell]^T$. Here, ℓ denotes the size of the learning phase data, which is equal to 10^4 samples. On the other hand, target vector will be true values of measured signal, i.e., $X = [x_1, \dots, x_\ell]$.

There are many different parameters to consider for a MLP network. For our implementation, we have decided to establish most of these parameters as constants. The reason is that all these parameters are strongly related to the characteristics of each application, and therefore, to obtain a better performance it would be necessary to specify more about the array system and the type of application. Then, the established parameters are: a single hidden layer; the transfer function for neu-

rons in the hidden layer is the hyperbolic tangent sigmoid; for training the Levenberg-Marquardt's algorithm is used [71]; and for evaluating the training process it is used the mean square error. Just the number of neurons in the hidden layer is left free for evaluating the best option.

3.7.3 Simulations

Simulations of this implementation are carried out under the model described in section 3.3. The first step is to train the MLP network using data from the learning phase. Once trained, the network is evaluated using the remaining data (from sample $10^4 + 1$ till the last sample). This ANN is implemented for both scenarios: uncalibrated and calibrated sensors.

For both scenarios, results are presented in Table 3.7. Here, it can be seen that RMS error in both scenarios are very similar, as expected. However, the estimation error using calibrated sensors is slightly lower than the one obtained with uncalibrated sensors. Observe that the ANN's performance is good even if the number of neurons is smaller than the number of sensors. Even, it can be seen that (in both scenarios) performance is affected by the increment in the number of neurons in the hidden layer. This effect is caused by what is usually called as the network overfeeding. Finally, it is important to highlight how error in estimation in both scenarios is lower than the one obtained by the calibrated sensor with highest precision in S (sensor 5) and by the simple average of all calibrated sensors (see Table 3.6). Observe that performance obtained with the ANN with calibrated sensors is very similar that performance obtained by the weighted average via LSQ method. This result makes sense, since under the conditions used, the neural network performs the same procedure as the weighted average via the LSQ method.

Table 3.7 – Assessment of MLP neural network.

Size of hidden layer	RMS Error (Units)	
	Uncalibrated sensors	Calibrated sensors
3 Neurons	0.1492	0.1370
5 Neurons	0.1492	0.1370
10 Neurons	0.1496	0.1373
20 Neurons	0.1494	0.1372
30 Neurons	0.1498	0.1378

3.7.4 Discussion

Here, we have analyzed the implementation of a MLP network in our generic sensor array system. This network was implemented for the two proposed scenarios: calibrated and uncalibrated sensors. Unlike LSQ, ANNs implements both calibration and weights estimation in a single step, which simplifies the process. Another advantage of this method is the good performance offered, even when MLPs are one of the simplest existing neural network architectures. It is important to note that this does not mean that using a more complex network performance can improve. For example, another type of networks widely used for signal processing are the Radial Basis Function (RBF). Roughly, a RBF network is the same as a MLP, with the difference that activation function is a radial function, hence its name. This type of networks are used for inference of functions at the input. This is useful for non-dynamic applications (also known as off-line applications). However, if the system is required for a real-time (on-line) application, then this network cannot be used.

As a disadvantage we have the fact that neural networks are strongly related to the characteristics of each implementation. In addition, there are a large number of variables that determine the performance in each application. For example, performance of this method is linked to the size and content of the data training. This hinders the idea of genericity that was defined at the beginning of this chapter.

As a partial conclusion, under the proposed scenarios, LSQ and ANN allows the same level of performance (Table 3.6 and Table 3.7) with the same drawbacks, i.e., both methods requires a learning phase with a known input or a golden device and do not comply with an evolution with time of the system. Next, we present the analysis of the Kalman filter, an algorithm that seeks to overcome these limitations.

3.8 Kalman filter

From [72]: *"the Kalman filter is named in honor to Rudolph E. Kalman, who in 1960 published his famous paper describing a recursive solution to the discrete-data linear filtering problem"*. This method consists of a set of mathematical equations that, based on Bayesian inference, implement a predictor-corrector type estimator. Being based on Bayes's rule, this method seeks to minimize error covariance, thus achieving an improvement in the estimation of the input signal. In fact, in some cases, such improvement is optimal. However, this can be done only when some presumed conditions are met. Here, we highlight this last fact: the estimation of the signal can be done when some features about the measured signal are known. Usually, this features are the frequency, dynamics, amplitude or phase. This is different from the previous two algorithms, where both required the complete knowledge of the input signal. Nevertheless, this small relaxation in the requirements has allowed a great diffusion by means of its implementation in a great variety of applications. For example, [72]: *"the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation"*.

The Kalman filter addresses the general problem of estimating the state $x \in \mathcal{R}^m$ governed by the following equation [72]:

$$x_k = Fx_{k-1} + Bu_k + w_{k-1} \tag{3.35}$$

Here, F is the $m \times m$ state transition matrix which relates the state at the previous time step $k-1$ to the state at the current step k, in the absence of either a driving function or process noise. Note that in practice F might change with each time step, but here we assume it is constant. The $m \times l$ matrix B is known as control matrix, which relates the optimal control input $u \in \mathcal{R}^l$ to the state x. Finally, u_k is the control input vector at time step k, and w_k represents the process noise. The measurement vector $y \in \mathcal{R}^n$ at time step k is described as:

$$y_k = Hx_k + v_k \tag{3.36}$$

H represents the state-to-measurement matrix, i.e. H describes the relationship between what is observed by sensors and what is estimated by the filter. This $n \times m$ matrix relates the state vector x_k to the measurement vector y_k. For some implementations H might change with each time step or measurement, but here we assume it is constant. Observe that (3.36) assumes the use of calibrated sensors. From now on, we will use the Kalman filter only for the scenario with calibrated sensors. Finally, v_k is the measurement noise. Both w_k and v_k are assumed to be white noise processes,

independent of each other, and with normal probability distributions:

$$p(w) \sim \mathcal{N}(0, Q) \tag{3.37}$$

$$p(v) \sim \mathcal{N}(0, R) \tag{3.38}$$

In practice, the process noise covariance Q and the measurement noise covariance R matrices might change each time step or measurement. However, for simplicity, hereafter we assume they are constant.

The Kalman filter performs the estimation process by using a form of feedback control [72]: *"the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback, i.e., for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate"*. The time update equations are usually called predictor equations, while the measurement update equations are called corrector equations. Figure 3.8 offers a schematic view of the operation of a Kalman filter.

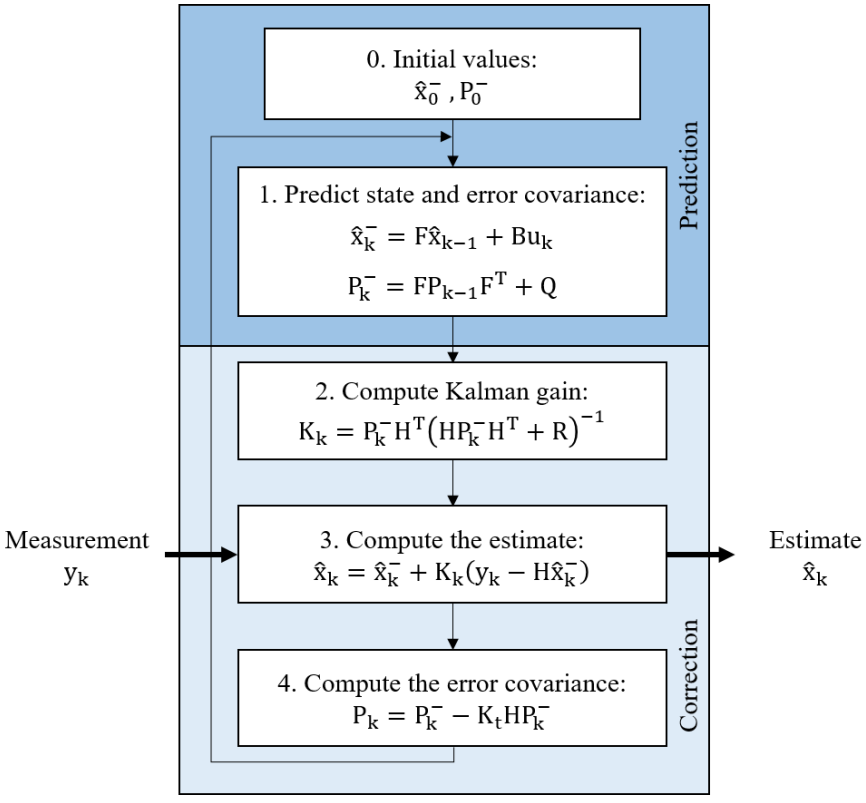


Figure 3.8 – A complete picture of the operation of Kalman filter.

Note that Figure 3.8 includes matrices P_k^- and P_k, which are used to indicate the error covariance in the prediction and estimation at time step k. To see this more clearly, definitions of these matrices are presented below:

$$P_k^- = E[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T] \tag{3.39}$$

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \tag{3.40}$$

Here $\hat{x}_k^-$ denotes the prediction and $\hat{x}_k$ denotes the estimation of the measured signal x. Note that both P_k^- and P_k may change with each time step. Next, we present the implementation of a Kalman filter for the sensor array system described in section 3.2 under the scenario where all sensors are calibrated.

3.8.1 Kalman filter for data fusion in sensor array systems

The implementation of a Kalman filter requires the partial knowledge of the input signal. Usually, a Kalman filter is used in applications where dynamics of the system (i.e., its behavior) are known [73]. In other cases, the filter takes advantage of the use of complementary sensors, for example, combining the measurements of acceleration and angular velocity to estimate the displacement of an object [32].

In this work, we seek to give a generic solution to the problem of data fusion in a sensor array system, such as the one presented in section 3.2. For this system it is assumed that there is no information about its dynamics. In addition, all devices in the system are assumed to be same-type sensors, which makes it impossible to implement solutions where different measured quantities are combined, such as the one presented in [32]. In [74], an on-line algorithm to blindly calibrate sensor drift using signal space projection and Kalman filter is proposed. This solution is very similar to the one presented with the blind calibration algorithm, and therefore, we have decided not to include it in this review. A different implementation is presented in [75], where a model-free filter is introduced based on the filtering equations of Kalman and the data-driven modeling of Takens. This nonparametric algorithm replaces the model with dynamics reconstructed from delay coordinates, while using the Kalman update formulation to reconcile new observations.

Let's consider the multi sensor system S described in section 3.2. To implement the Kalman-Takens filter in S, we start with the definitions of the state vector and measurement vector for this system:

$$x_k = F(x_{k-1}) + w_{k-1} \tag{3.41}$$

$$y_k = \begin{bmatrix} y_{1,k} \\ \vdots \\ y_{n,k} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} x_k + \begin{bmatrix} v_{1,k} \\ \vdots \\ v_{n,k} \end{bmatrix} \tag{3.42}$$

Here, the model function F is unknown, and therefore, it must be obtained from other methods. It is here where an estimation of the dynamic model of the system is carried out by using the Takens theorem explained in [76]. The idea presented in [75] is to replace the system evolution, traditionally done through application of F, with advancement of dynamics non parametrically using delay-coordinate vectors. The implementation of this variant of the Kalman filter is explained below by using the following example taken from [76]:

"Let $\bar{y}_{k-1} = \frac{1}{n}\sum_{i=1}^{n} y_{i,k-1}$ *be the average of measurement vector at time step* $k-1$*, and let* $\tilde{x}_{k-1} = [\bar{y}_{k-d}, \ldots, \bar{y}_{k-1}]$ *its corresponding delay-coordinate vector, where* d *is the number of delays (in terms of time steps). This delay vector* $\tilde{x}_{k-1}$ *is used to predict the state* x_k*. Thus, given*

a delay-vector $\tilde{x}_{k-1}$, computation of $F(\tilde{x}_k)$ is carried out by locating its M nearest neighbors $[\bar{y}'_{k-d}, \ldots, \bar{y}'_{k-1}]$, $[\bar{y}''_{k-d}, \ldots, \bar{y}''_{k-1}]$ $\ldots$ $, [\bar{y}^M_{k-d}, \ldots, \bar{y}^M_{k-1}]$ withing the set of data dictionary. Once that neighbors are found, the known $\bar{y}'_k, \bar{y}''_k, \ldots, \bar{y}^M_k$ values are used to predict $\hat{x}^-_k$. For example, the estimation can be done by computing the weighted average of predictions obtained by the neighbors, as follows:"

$$\hat{x}^-_k = F(\tilde{x}_{k-1}) = \left[w_1 \bar{y}'_k + w_2 \bar{y}''_k + \ldots + w_M \bar{y}^M_k\right] \tag{3.43}$$

Figure 3.9 shows an example of the use of a data dictionary for predicting the next state. It is important to remark that this algorithm requires the storage of a data dictionary. This set works as a database from which the closest historical neighbors of a vector are obtained. This data dictionary can be formed during the execution time by including at each time step a new delay-vector. Then, the search of the closest neighbors is done using the already stored delay-vectors. At the beginning of the execution, the performance of this filter is low, and it increases as the size of the data dictionary increases. Once that a defined number of delay-vectors are stored, we stop to add new delay-vectors to the dictionary. Observe that, unlike the learning phase used in previous algorithms, the knowledge of the input signal is not required. Moreover, it is possible to increase the size of the data dictionary as much as required just by including at each step the new formed delay vector. However, by increasing the size of the data dictionary, computational work required for this algorithm increases. To get an idea of the computational complexity, the following analysis is performed.

Figure 3.9 – Example of Kalman-Takens algorithm.

Consider a data set composed of m measurement vectors. From this set, we can build $m - d$ delay vectors. The set of these $m - d$ delay vectors is known as a data dictionary. During the prediction phase, having measurement vector y_{k-1}, a new delay vector $\tilde{x}_{k-1}$ is created. To find the M closest neighbors to $\tilde{x}_{k-1}$, it is required to consult the $m - d$ delay vectors that composed the data dictionary. So, at each time step, at least a number of $\mathcal{O}(m - d)$ operations should be done in order to find the predicted state. Because $m \gg d$, the computational complexity of this algorithm can be approximate to $\mathcal{O}(m)$.

One possible solution to the high computational cost of this algorithm consist in dividing the data dictionary into clusters, and then, looking for the M closest neighbors inside of a specific cluster, instead of the whole data dictionary. This can reduce the number of computational steps, and therefore, the execution time of this algorithm. For example, Figure 3.10 shows a clustering

of size 3 in a data dictionary with $d = 3$. Thus, search for the M nearest neighbors is reduced to $\mathcal{O}(m/3)$, where factor $\frac{1}{3}$ is due to the number of clusters.

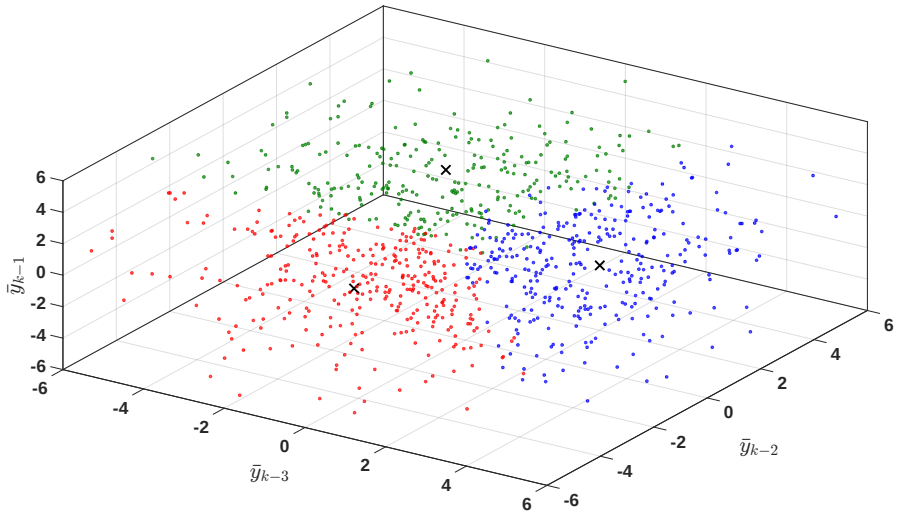


Figure 3.10 – Example of a 3-clustering in a data dictionary with $d = 3$.

As mentioned before, this algorithm will be applied only for the scenario with calibrated sensors. Finally, equation (3.42) shows that H results in a $n \times 1$ vector where all its elements are ones. This is due to the assumption that all sensors measure the same input.

The implementation of Kalman-Takens filter can be carried out using equations presented in Figure 3.8, with the difference that model function F is computed using the d nearest delay vectors found inside the data dictionary, as described in (3.43). Different versions of Kalman filter can be used for implementations (for example the extended Kalman filter or the ensemble Kalman filter), but for this analysis the linear version of the Kalman filter will be used. The Matlab code used for this implementation can be found in Appendix.

3.8.2 Simulations

Next, we present the result of implementing the Kalman-Takens algorithm under the scenario where all sensors are calibrated. For this implementation, the first 10^3 samples are used for building the data dictionary. To reduce the computational complexity, data dictionary is divided into 5 clusters using Matlab function *kmeans*, where the determining parameter for classification is the euclidean distance between delay vectors. Assessment of this implementation is carried out from the beginning to the end of the simulation.

For Kalman equations, control signals are omitted. Matrix R results in a diagonal matrix of size $n \times n$, which is parametrized using the white noise power level of each sensor. This information is taken from Table 3.1. Matrix Q results in a single value which is set up with values $\{10^{-3}, 10^{-2}\}$. These values were selected after performing several simulations and evaluating the relationship between the value of Q and the RMS error obtained, as illustrated in Figure 3.11. Parameters $\hat{x}_0^-$ and P_0^- are initialized with values 0 and 10^{-3}, respectively. For Takens method, the weighted average presented in equation (3.43) is used to compute the predicted state $\hat{x}_k^-$, where each weight is estimated using distance as parameter. For example, weight of predicted value coming from

neighbor i is computed as:

$$w_i = \frac{\text{distance}_i^{-1}}{\sum_{j=1}^{M} \text{distance}_j^{-1}} \tag{3.44}$$

where distance_i denotes the euclidean distance between neighbor i and $\bar{y}_k$. Parameters d and M are set up with values $\{2, 4, 6\}$ and $\{5, 10, 15, 20, 25\}$, respectively. Results of these simulations are presented in Figure 3.12, Figure 3.13, and Table 3.8.

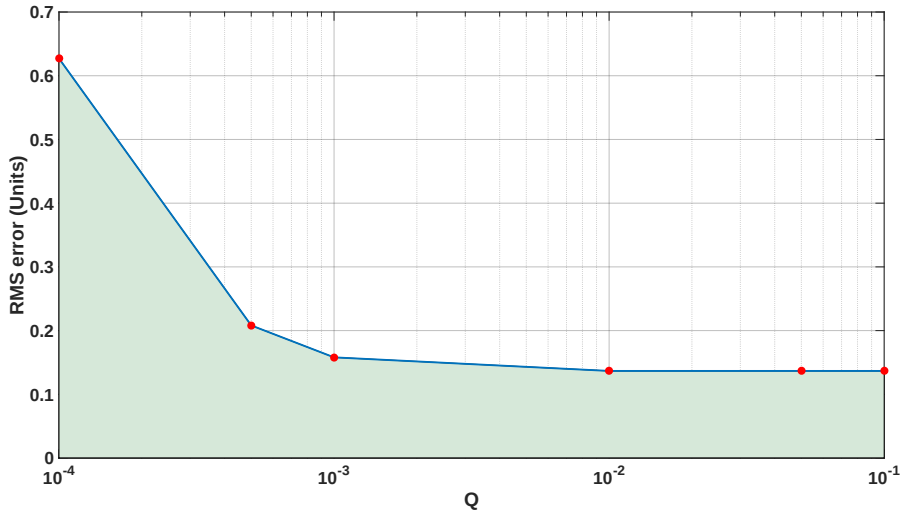


Figure 3.11 – Relationship between RMS error and parameter Q. For this graph, values of d and M were fixed to 6 and 20, respectively.

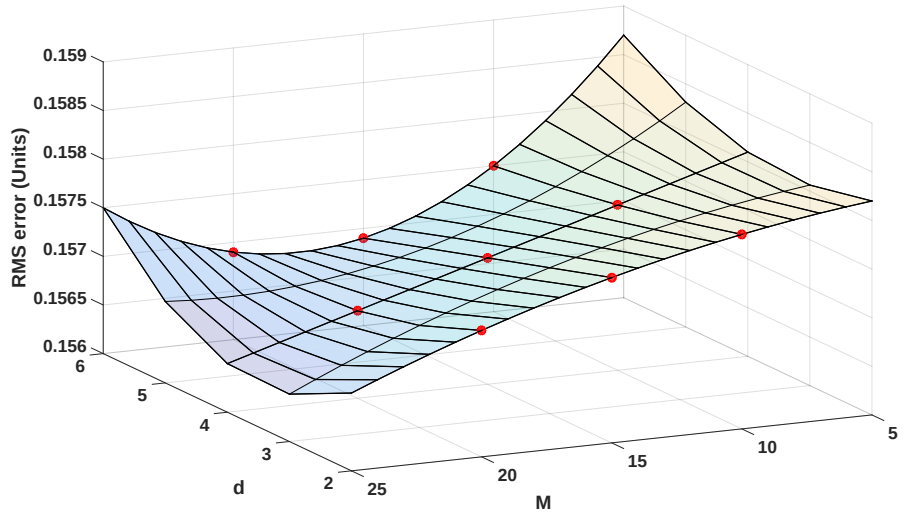


Figure 3.12 – RMS error obtained from different executions of Kalman-Takens algorithm. Parameter Q is fixed to 10^{-3}.

Table 3.8 – Best performance obtained in the assessment of Kalman-Takens.

Signal estimator	Size of delay vector d	Number of neighbors M	Value of Q	RMS Error (Units)
Kalman - Takens algorithm	4	20	10^{-3}	0.1569
	6	20	10^{-2}	0.1363

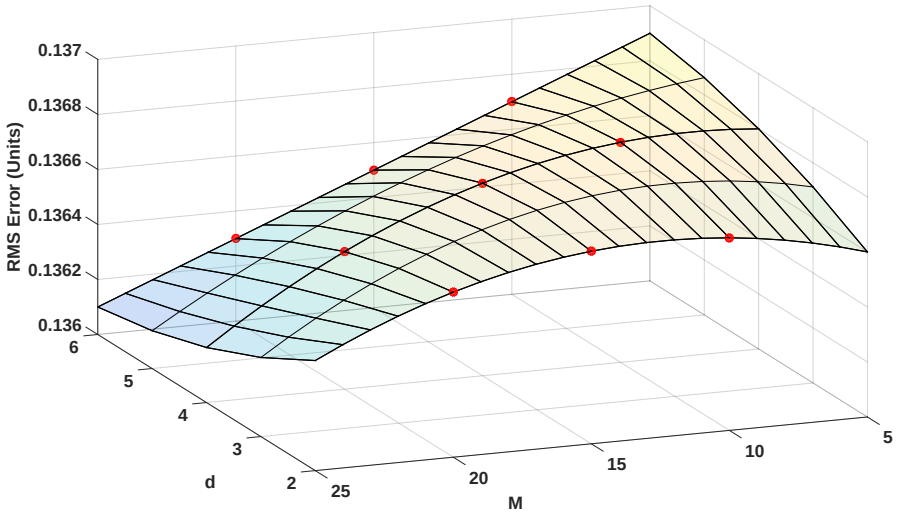


Figure 3.13 – RMS error obtained from different executions of Kalman-Takens algorithm. Parameter Q is fixed to 10^{-2}.

From Table 3.8 we observe that RMS error reached with this implementation is almost equivalent that errors obtained through the LSQ method and MLP network. Nevertheless, it should be taken into account that for this implementation the knowledge of input is not required. This fact allows its implementation in many more applications than the two previous algorithms.

3.8.3 Discussion

Under the proposed assumptions of our generic sensor array system, dynamics, shape, amplitude and frequency of the signal are unknown. Only the noise spectral density of each sensor is known. This scenario is frequently found in real applications, especially for implementation in poorly controlled or unknown environments.

As a solution for the lack of information on the measured signal, the variant of the Kalman filter presented in [75] was used. In this algorithm, an attempt is made to approach the dynamics of the system through the use of delay vectors. Broadly speaking, the main idea is to look for a similarity between the current and the previous measurements. Thus, a prediction of the next measurement can be done by considering the predictions given by the previous measurements.

Two important parameters to consider in the implementation of this algorithm are the size of the delay vector and the size of the data dictionary. By increasing these parameters, it is sought to increase the probability of finding the repetition of a pattern inside the behavior of a signal. Nevertheless, these two parameters have a direct impact on the performance of the algorithm, since by increasing the number of delays and/or the size of the data dictionary the memory and

computational steps required to carry out this algorithm increase dramatically.

As a partial conclusion, under the given assumptions (including the pre-calibration of sensors), Kalman-Takens filtering allows the same level of performance that LSQ and MLP methods, without a learning phase but at the price of a higher computational workload. The next algorithm can reach the same level of performance with a lower computational cost.

3.9 Random weighting

The last algorithm to analyze is the random weighting method proposed in [64]. This algorithm achieves the same level of performance that the LSQ under the scenario where all sensors are calibrated, with the difference that random weighting does not require input knowledge. In addition, the computation workload required for the random weighting is considerably lower than that required for the Kalman-Takens filter. Next, we present a brief summary about this algorithm.

Consider the weighted average of system S at time step k given by:

$$\hat{x}_k = w_{1,k} y_{1,k} + w_{2,k} y_{2,k} + \cdots + w_{n,k} y_{n,k} \tag{3.45}$$

Here, $w_{i,k}$ denotes the weight of sensor i at time step k, where $\forall k$ we have that:

$$\sum_{i=1}^{n} w_{i,k} = 1 \tag{3.46}$$

Assume that all sensors are calibrated, where stochastic errors between different sensors are independent and no related with the input. Thus, expression (3.45) can be expressed as:

$$\hat{x}_k = x_k + w_{1,k}\varepsilon_{1,k} + w_{2,k}\varepsilon_{2,k} + \cdots + w_{n,k}\varepsilon_{n,k} \tag{3.47}$$

Where $\varepsilon_{i,k}$ denotes stochastic error of sensor i at time step k. Then, the estimation error is:

$$error(\hat{x}_k) = w_{1,k}\varepsilon_{1,k} + w_{2,k}\varepsilon_{2,k} + \cdots + w_{n,k}\varepsilon_{n,k} \tag{3.48}$$

Each ε_i is a set of independent and identically distributed random variables with $\varepsilon_i \sim \mathcal{N}(0, \sigma^2(\varepsilon_i))$, where $\sigma^2(\varepsilon_i) = \frac{1}{k}\sum_{j=1}^{k}(\varepsilon_{i,j} - \mu)^2$, and $\mu = \frac{1}{k}\sum_{j=1}^{k}\varepsilon_{i,j}$. Then, variance of error described in (3.48) can be written as:

$$\sigma^2(error(\hat{x}_k)) = {w_{1,k}}^2\sigma^2(\varepsilon_1) + \cdots + {w_{n,k}}^2\sigma^2(\varepsilon_n) \tag{3.49}$$

The best estimation of the signal x is obtained by minimizing the variance in (3.49), which can be done using the method of Lagrange multipliers, as presented in [77]. First, functions f and g are defined as:

$$\begin{aligned} f(w_{1,k}, \ldots, w_{n,k}) &= {w_{1,k}}^2\sigma^2(\varepsilon_1) + \cdots + {w_{n,k}}^2\sigma^2(\varepsilon_n) \\ g(w_{1,k}, \ldots, w_{n,k}) &= w_{1,k} + \cdots + w_{n,k} \end{aligned} \tag{3.50}$$

To find the critical points of function f subjected to the constraint given by g it is necessary to

solve the following system of equations:

$$\begin{aligned} \nabla f(w_{1,k}, \dots, w_{n,k}) &= \lambda \nabla g(w_{1,k}, \dots, w_{n,k}) \\ g(w_{1,k}, \dots, w_{n,k}) &= 1 \end{aligned} \tag{3.51}$$

It is possible to rewrite this as a collection of $n+1$ equations with the $n+1$ unknowns, $w_{1,k}, \dots, w_{n,k}$ and λ:

$$\begin{aligned} f_{w_{1,k}}(w_{1,k}, \dots, w_{n,k}) &= \lambda g_{w_{1,k}}(w_{1,k}, \dots, w_{n,k}) \Rightarrow 2w_{1,k}\sigma^2(\varepsilon_1) = \lambda \\ &\vdots \\ f_{w_{n,k}}(w_{1,k}, \dots, w_{n,k}) &= \lambda g_{w_{n,k}}(w_{1,k}, \dots, w_{n,k}) \Rightarrow 2w_{n,k}\sigma^2(\varepsilon_n) = \lambda \\ g(w_{1,k}, \dots, w_{n,k}) &= 1 \Rightarrow w_{1,k} + \cdots + w_{n,k} = 1 \end{aligned} \tag{3.52}$$

Rearranging expressions presented above, the values of variables $w_{1,k}, \dots, w_{n,k}$ are:

$$\begin{aligned} w_{1,k} &= \frac{1}{1 + \frac{\sigma^2(\varepsilon_1)}{\sigma^2(\varepsilon_2)} + \frac{\sigma^2(\varepsilon_1)}{\sigma^2(\varepsilon_3)} + \cdots + \frac{\sigma^2(\varepsilon_1)}{\sigma^2(\varepsilon_n)}} \\ &\vdots \\ w_{n,k} &= \frac{1}{\frac{\sigma^2(\varepsilon_n)}{\sigma^2(\varepsilon_1)} + \frac{\sigma^2(\varepsilon_n)}{\sigma^2(\varepsilon_2)} + \frac{\sigma^2(\varepsilon_n)}{\sigma^2(\varepsilon_3)} + \cdots + 1} \end{aligned} \tag{3.53}$$

Generalizing (3.53), the weight $w_{i,k}$ of sensor i is given by:

$$w_{i,k} = \frac{1}{\sum_{j=1}^{n} \frac{\sigma^2(\varepsilon_i)}{\sigma^2(\varepsilon_j)}} \tag{3.54}$$

Finally, it is necessary to ensure that the found solution is a minimum. For this, the Hessian matrix is built:

$$\begin{aligned} F(w_{1,k}, \dots, w_{n,k}) &= {w_{1,k}}^2\sigma^2(\varepsilon_1) + \cdots + {w_{n,k}}^2\sigma^2(\varepsilon_n) \\ &\quad + \lambda(w_{1,k} + \cdots + w_{n,k} - 1) \\ \frac{\partial^2 F}{\partial^2 w_{i,k}} &= 2\sigma^2(\varepsilon_i) \\ \frac{\partial^2 F}{\partial w_{i,k} w_{j,k}} &= 0 \quad ; \forall i \neq j \\ H(F) &= \begin{bmatrix} 2\sigma^2(\varepsilon_1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 2\sigma^2(\varepsilon_n) \end{bmatrix} \end{aligned} \tag{3.55}$$

The resulting matrix is always a diagonal matrix where all elements are positives, therefore the determinant is always positive implying that the solution is a minimum.

Thus, weight $w_{i,k}$ corresponding to sensor i at time step k is obtained as follows:

$$w_{i,k} = \left(\sum_{j=1}^{n} \frac{\sigma^2(\varepsilon_i)}{\sigma^2(\varepsilon_j)} \right)^{-1} \tag{3.56}$$

It can be seen from (3.56) that to obtain the optimal weighting factors, the variance of each sensor in S has to be calculated. In [64], these variances are calculated according to the sensor observations as follows.

For any two sensors i and j, their observations at time step k are $y_{i,k}$ and $y_{j,k}$, respectively. The corresponding observation errors are $\varepsilon_{i,k}$ and $\varepsilon_{j,k}$. Assuming that $y_{i,k}$ and $y_{j,k}$ are independent of each other, and both $\varepsilon_{i,k}$ and $\varepsilon_{j,k}$ are white noise processes with zero mean, then $y_{i,k} = x + \varepsilon_{i,k}$, and similarly, $y_{j,k} = x + \varepsilon_{j,k}$. Let's denote the variance of sensor i as follows:

$$\sigma^2(\varepsilon_i) = E[\varepsilon_i^2] \tag{3.57}$$

The cross-covariance function of y_i and y_j is:

$$\gamma_{i,j} = E[y_i y_j] = E[x^2] \tag{3.58}$$

And the self-covariance function of y_i is:

$$\gamma_{ii} = E[y_i^2] = E[x^2] + E[\varepsilon_i^2] \tag{3.59}$$

Therefore, from (3.57)-(3.59), variance of sensor i can be computed as follows:

$$\sigma^2(\varepsilon_i) = E[\varepsilon_i^2] = \gamma_{ii} - \gamma_{i,j} \tag{3.60}$$

Thus, random weighting algorithm is based in equations (3.60) (for sensors' variances) and (3.56) (for individual weights). Next, we present an example of implementation of this algorithm for data fusion in a sensor array system.

3.9.1 Random weighting for data fusion in sensor array systems

The implementation of random weighting method in system S consists in the following steps: each time step k, the measurement vector $y_k = [y_{1,k}, \dots, y_{n,k}]^T$ is used to update the self-covariance and the cross-covariance of each sensor. Then, individual variances of sensors are updated. Once this is done, weights of all sensors are computed. Finally, the estimation of the measured signal is calculated by means of a weighted average. This is illustrated below:

$$\begin{bmatrix} y_{1,k} \\ \vdots \\ y_{n,k} \end{bmatrix} \xrightarrow[\text{sensor } i]{\text{For each}} \text{Update } \gamma_{ii}, \gamma_{i,j} \rightarrow \text{ Update } \sigma^2(\varepsilon_i) \rightarrow \text{ Compute } w_{i,k} \rightarrow \text{ Compute } \hat{x}_k = \sum_{i=1}^{n} w_{i,k} y_{i,k}$$

Note that for computing $\gamma_{i,j}$ it is required to select a sensor reference. The performance of this algorithm is not linked to the selection of this sensor, however, it must be ensured that such reference is free from failures at any time step. For our implementation, sensor 1 has been selected as the reference. The code of this algorithm can be consulted in Appendix.

One important point to remark is that estimation of variances is carried out each time step. This allows a dynamic update in weights of the sensors, which is ideal for scenarios where low-frequency noises are strongly present on sensors measurements. The computational complexity of this algorithm is $\mathcal{O}(n^2)$, which is given by the sum executed in (3.56). This low computational cost makes this algorithm a viable option for real-time implementations.

3.9.2 Simulations

For its evaluation, the random weighting method described above is implemented into the generic system described in section 3.2. Note that this algorithm does not require a learning phase, and therefore, its evaluation is carried out from the beginning to the end of the simulation. This implementation can only be applied for the scenario where all sensors are calibrated. The reason of this is the following: if systematic errors are present in sensors, then they will affect in the computation of γ_{ii} and $\gamma_{i,j}$, which completely breaks the correctness of the algorithm.

Random weighting method requires the on-line computation of average in (3.60). At the beginning, accuracy of this on-line estimation is very low; however, after a while, the estimate of the average converges to the expected value. This is illustrated in Figure 3.14 and Figure 3.15. In the first figure we observe how estimation of $\gamma_{1,1}$ converges after 100 samples to the value obtained by the off-line value computed on all the samples. On the second figure, it is shown how at the beginning, estimation of individual variances is unstable; even zero and negative values are obtained. Once the on-line computation of (3.60) is stable (after 10^4 samples), estimation of $\sigma^2(\varepsilon_i)$ starts to be accurate. As a result of the instability in estimation of individual variances, estimation of measured signal given by the random weighting algorithm presents a high error at the beginning. Once that estimation of individual variances converges, the algorithm presents its best performance. This is illustrated in Figure 3.16, where an analysis of the RMS error in the signal estimation is presented. Here, on the horizontal axis we present the sample from which we start the computation of error, i.e., RMS error is calculated from this sample until the last sample of the simulation. On the vertical axis, value of the RMS error corresponding to this data vector is found. Through this graph it is possible to observe how, once the behavior of the on-line computation of (3.60) is stable, optimal performance of the algorithm is obtained (after 10^4 samples). A synthesis of the results obtained in this graph are shown in Table 3.9.

Table 3.9 – Assessment of random weighting algorithm.

Parameter	Scenario	From the beginning to the end of simulation	From Sample 1×10^3 to the end of simulation	From Sample 1×10^4 to the end of simulation
RMS Error (Units)	Calibrated Sensors	1.6753	1.6720	0.1375

3.9.3 Discussion

The random weighting algorithm could only be implemented for the scenario with calibrated sensors. Without strong modifications, this algorithm cannot be applied to the scenario with uncalibrated sensors. This algorithm was able to reach a similar level of performance that the Kalman-Takens filter, the LSQ and the MLP network. However, unlike such algorithms, this method does not require the storage of big quantities of data (as the Kalman-Takens method) or the implementation of a learning phase (as the LSQ and the MLP network).

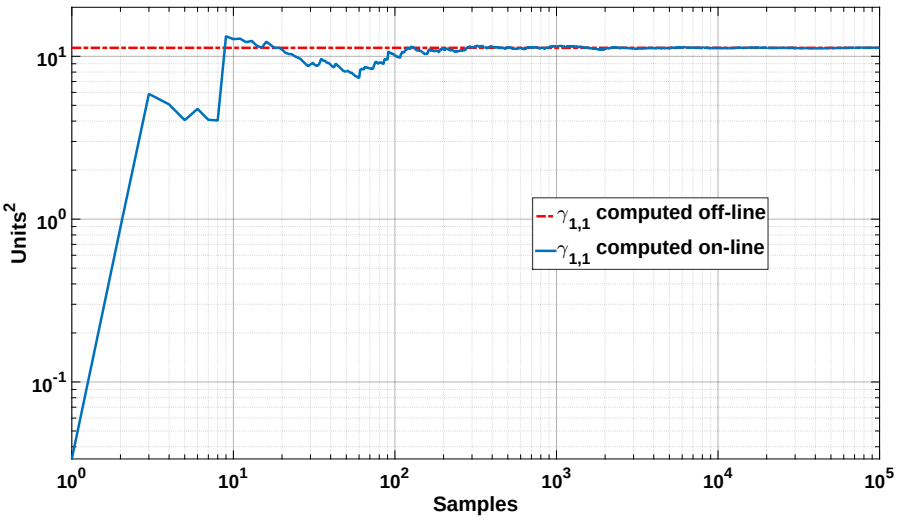


Figure 3.14 – Difference in on-line and off-line estimation of $\gamma_{1,1}$.

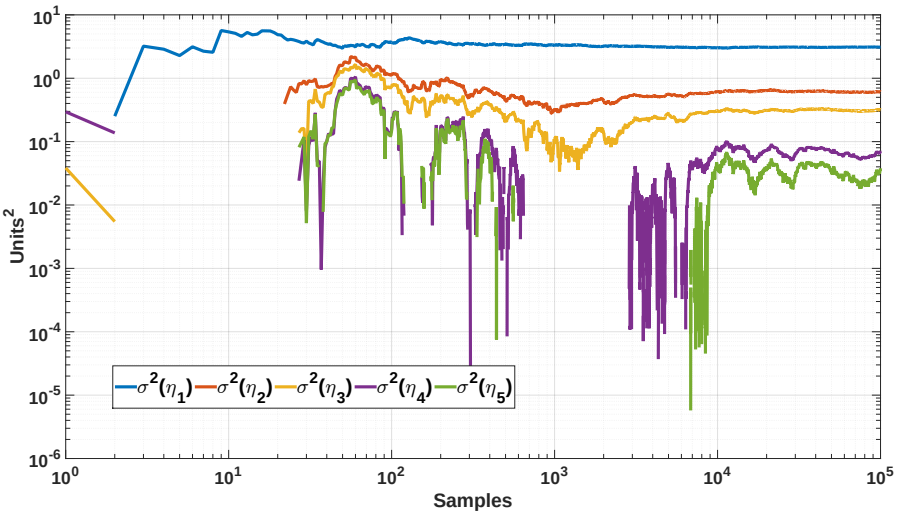


Figure 3.15 – Estimation of $\sigma^2(\varepsilon_i)$.

In addition, variances of sensors (and therefore, weights) can be updated at each time step. This advantage allows its implementation in more dynamic scenarios, where the presence of low-frequency noises are considered. It should be noted that such dynamism is limited by the on-line computation of (3.60), which is cumulative, and therefore, the more measurements taken into account it will be more difficult to detect any change in the variance of any sensor. To counteract this, a different function can be implemented to estimate the arithmetic mean, such as a low pass filter. This will be fully illustrated in chapter 4.

Also, it is important to highlight that this algorithm does not require the knowledge of the input, which makes it practical for applications where controlled environments are not possible.

On the other hand, a huge disadvantage of this algorithm is its dependence on a reference sensor for the estimation of variances. Because of this, this algorithm must be implemented under certain

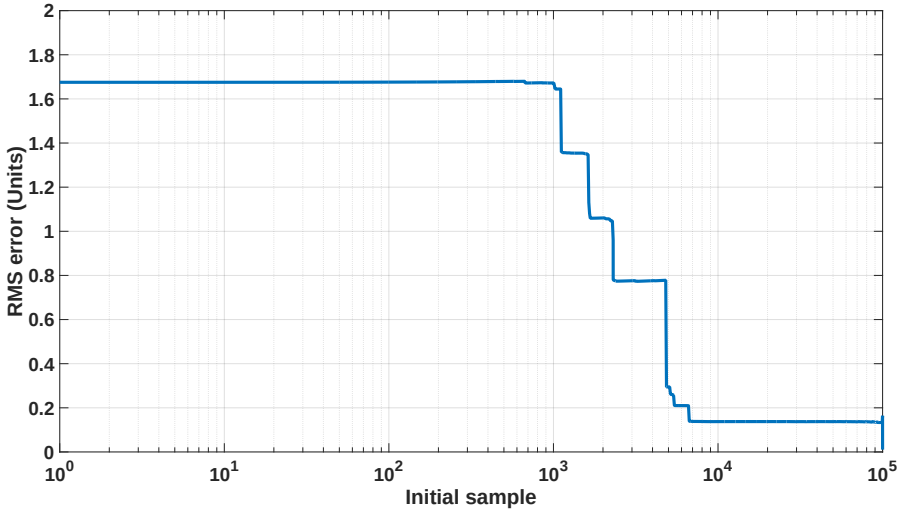


Figure 3.16 – RMS error obtained with the random weighting algorithm.

assumptions, such as ensuring that at any time step the reference is a fault-free sensor.

3.10 Summary

Throughout this chapter we have evaluated the implementation of five different algorithms for data fusion in sensor array systems. For the assessment, simulations of environment presented in section 3.3 were carried out. All algorithms were evaluated using the same information. To measure the performance, the RMS of the difference between the signal estimate obtained by the algorithm and the measurement made by an ideal sensor (without noise) was used as a metric. The performance obtained in all cases is summarized in Table 3.10.

Table 3.10 – Comparison of all implemented algorithms.

Sensor / Data fusion method	RMS Error (Units)	
	Uncalibrated Sensors	Calibrated Sensors
Sensor 1	1.7616	1.7694
Sensor 2	0.8004	0.7831
Sensor 3	0.6298	0.5820
Sensor 4	0.4736	0.2680
Sensor 5	0.5242	0.1957
Average	0.4303	0.4095
Blind calibration + Simple average	0.4228	—
LSQ method	0.1491	0.1360
MLP ANN	0.1492	0.1370
Kalman-Takens filter	—	0.1363
Random weighting	—	0.1375

Table 3.10 shows that LSQ and ANN may be used for uncalibrated sensors. The blind calibration algorithm shows a bad level of performance comparable to the one given by the simple average of uncalibrated sensors. This algorithm is not dedicated to sensor arrays with different noise levels.

For the scenario with calibrated sensors, the performance obtained using the random weighting algorithm is almost the same as the one obtained with the LSQ, Kalman-Takens and MLP ANN. However, there are some advantages of using random weighting method such as it does not require any knowledge about the input signal, it is dynamic, and it has a low computational complexity. In Table 3.11 we present some other properties of these algorithms.

Table 3.11 – Comparison of all presented algorithms.

Requirements/Characteristics	Blind calibration	LSQ	MLP ANN	Kalman - Takens	Random weighting
Knowledge of input signal (learning phase)	—	✓	✓	—	—
Sensor reference	✓	—	—	—	✓
Storage of big data (data dictionary)	—	—	—	✓	—
Supports in-run changes in sensors noise levels (omitting changes in sensor reference)	✓	—	—	✓	✓
Complexity during the learning phase	—	$\mathcal{O}(n^3)$	High	—	—
Complexity during the evaluation	$\mathcal{O}(n^3)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(m)$ m -size of the data dictionary	$\mathcal{O}(n^2)$

Based on the previous study, in next chapter a new algorithm for data fusion in a sensor array system is proposed. The design of such algorithm was based on the best features of algorithms presented above. For example, the use of a weighted average at each time step was taken as a reference. This offers an advantage in terms of dynamism as well as a possibility of carrying out fault detection. Also, for this new algorithm the idea of orthogonal projection using in the blind calibration method was taken, as well as the idea of minimization of variances used in the LSQ. In other cases, thanks to the analysis it was concluded to avoid the use of other elements, such as data dictionaries like shown in Kalman-Takens filter, which require high memory resources, as well as a high computational work.

Chapter 4

An adaptive algorithm based on MINQUE for data fusion and fault detection in sensor array systems

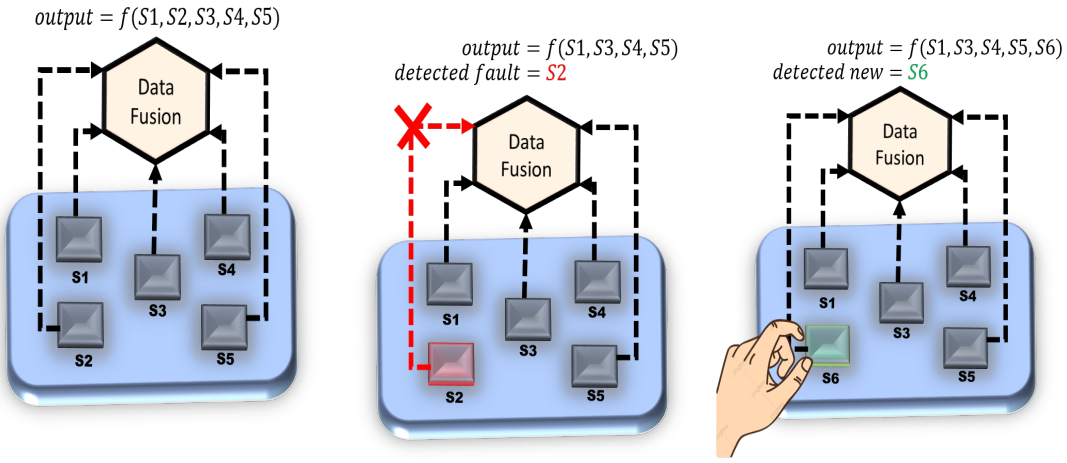
Low cost and small size integrated sensors bring a significant interest to redundant sensing systems such as sensor array systems and sensor networks. Redundancy allows to increase both performance and dependability of individual sensors at the system level. However, to fully exploit the benefits of these redundant systems it is necessary to design data fusion algorithms that are able not only to offer a better performance than the one of each element of the system, but also to adapt to possible failures in such elements. In this chapter, a new adaptive algorithm for sensor array systems is presented. This approach is generic, which means that it can be implemented for different types of sensor systems. The proposal is an on-line method based on the MInimum Norm Quadratic Unbiased Estimation (MINQUE), which: 1) estimates the variances of sensors at each time step, 2) detects and identifies faulty / uncalibrated sensors, and 3) reincorporates new / recovered sensors during the runtime. In addition, a proof that MINQUE algorithm requires that the number of sensors is strictly greater than two times the number of signals to measure is presented. Consequently, the proposed method is able to manage the presence of $m - 1$ faulty devices, where m is the number of sensors less two times the number of measured signals.

4.1 Introduction

A sensor array system is defined as a collection of same-type or different-type sensors, usually deployed in a fixed topological pattern and used to measure one or several physical inputs. Once measurements are obtained, a data fusion method is applied to leverage redundancy in measurements. These methods allow us to exploit the benefits of redundant information, such as offering better performance than that achieved individually by the elements of the system and adapting to possible failures / uncalibrations of such elements.

These two characteristics are essential for the development of autonomous multi-sensor systems. For example, consider sensor array shown in Figure 4.1. Data fusion algorithms look for an optimal way to merge measurements from all sensors, taking advantage of the redundant data. In addition, in many applications where the implementation lifetime is long, failures or uncalibrations of sensors may occur. However, due to the redundancy of elements, the presence of these faults

could be overcome by identifying and discarding damaged (or uncalibrated) sensors, as illustrated in Figure 4.1.



(a) Higher performance than that achieved individually by the elements of the system.

(b) Detection and omission of faulty / uncalibrated sensors.

(c) Detection of new / recovered sensors.

Figure 4.1 – Sought advantages when using data fusion algorithms in redundant multi-sensor systems.

Array signal processing and measurement data fusion have been extensively studied in the literature [9], and proposed methods have been applied to various types of sensor arrays such as antenna arrays [10], inertial sensor arrays [32], magnetic sensor arrays [11], acoustic sensor arrays [12], and chemical sensor arrays [13].

One example of this can be found in [31], where a fusion method for measurements from several triads of accelerometers and gyroscopes is presented. Simulations assuming arrays of four accelerometer/gyroscope triads are done, and for experiments, an embedded system of 32 MEMS-based accelerometer/gyroscope/magnetometer triads is used to show that redundancy in data can overcome low-cost sensor problems.

Another example can be found in [78], where a data fusion algorithm for arrays of inertial MEMS is developed. The general idea is to apply a maximum likelihood method combined with a motion model for estimating the specific force, angular velocity, and angular acceleration of the array at each time-step. This algorithm exploits the advantage of having different types of sensors into a single IMU, restricting it only for applications in inertial sensor arrays where accelerometers and gyroscopes are included.

Finally, an algorithm based on random weighting for sensor data fusion is presented in [64]. In this paper, a weighted average based on the analysis of variances of individual sensors is implemented for data fusion. An exhaustive study of this algorithm was presented in chapter 3, where it was concluded that the main disadvantage of this algorithm lies on the individual estimation of variances: the variance of a given sensor is estimated using its self-covariance and the cross-covariance with other sensors. This approach is interesting in terms of genericity: it can be applied to any sensor array system. However, this solution is unreliable in scenarios where sensors of a system are susceptible to the presence of faults. This is a consequence of the use of a golden device, i.e., a trustable sensor; a condition that cannot be fulfilled in many real applications.

From the study of state of the art, we learn that the use of weighted average presents two main interests: i) the idea of estimating sensors' variances without the knowledge of the input and, ii) genericity of the algorithm. By studying previous work in that field [64], we were able, on the one hand, to improve the process for estimating individual variances of sensors to avoid the use of a golden device. On the other hand, we introduce reliability constraints into the algorithm to handle dynamically the presence of faulty sensors.

As a result, a variation of the MInimum Norm Quadratic Unbiased Estimation (MINQUE) algorithm [79] for sensor array systems is presented in this chapter. This is an on-line algorithm whose main advantages are:

- Estimation of sensors' variances can be done at each time step, permitting to track changes of these variances caused principally by the low-frequency noise effects,
- Knowledge of the input signal is not required,
- Unlike in [64], it does not require the use of a golden device for the estimation of the variance,
- Genericity, which means that it can be implemented for different types of sensor arrays,
- Detection of faults / uncalibrations of sensors, as well as the reincorporation of recovered / replaced sensors,
- Fault tolerance, as long as the number of fault-free sensors is greater than two times the number of measured signals.

This chapter is organized as follows: in section 4.2 the problem statement is introduced. In section 4.3 adaptation of MINQUE algorithm for estimating variances in sensor array systems is explained in detail, where some results about the number of sensors required for such algorithm as well as the number of tolerated faulty sensors are presented. In section 4.4 is explained the procedure to carry out fault detection and reincorporation of recovered devices. Finally, the pseudo-code of the proposed algorithm that summarizes the entire process is presented in section 4.5.

4.2 Problem statement

Consider a discrete-time sensor array system S composed of n same-type sensors, all of them measuring a linear combination of the same r physical quantities denoted as X. Then, measurement reported by sensor i at time step k is defined as:

$$y_{i,k} = (\alpha_T + \alpha_{i,k})h_i\mathbf{X}_k + \beta_{i,k} + \varepsilon_{i,k} \tag{4.1}$$

where:

- α_T is the typical value for the scale factor (usually normalized to one)
- $\alpha_{i,k}$ is the scale factor error in sensor i at time step k
- h_i is the observation row vector of size $1 \times r$
- $\mathbf{X}_k$ is the $r \times 1$ vector of physical quantities to measure
- $\beta_{i,k}$ is the bias of sensor i at time step k

- $\varepsilon_{i,k}$ denotes the noise present in sensor output (white noise, flicker noise, brown noise, etc.)

Observe that $\alpha_{i,k}$ and $\beta_{i,k}$ were previously defined as constants, however, it is now assumed that they can change over the lifetime of an implementation. Throughout this chapter, we use the term "uncalibration" to refer to these changes where $\alpha_{i,k}$ and $\beta_{i,k}$ (equal to zero after calibration) are different from zero.

On the other hand, it is important to note that observation vector h_i is not the same that the scale factor $(\alpha_T + \alpha_{i,k})$; h_i corresponds to the intensity with which $i-$th sensor receives inputs (usually associated with the topology of the array), while $(\alpha_T + \alpha_{i,k})$ is linked to the physical properties of the sensor. Vector h_i is usually known, while the value of $\alpha_{i,k}$ can only be known if sensor i is calibrated. From here on, it will be assumed that h_i is known.

For the sake of simplicity, assume that all sensors are calibrated. Then, systematic errors are normalized to $(\alpha_T + \alpha_{i,k}) = 1$ and $\beta_{i,k} = 0$. Measurement reported by calibrated sensor i at time step k is defined as:

$$y_{i,k} = h_i \mathbf{X}_k + \varepsilon_{i,k} \tag{4.2}$$

Accordingly, assuming that all sensors in S are calibrated and synchronized, vector of n measurements reported at time step k is defined as:

$$\mathbf{Y}_k = \mathbf{H}\mathbf{X}_k + \mathbf{E}_k \tag{4.3}$$

where:

$$\begin{aligned} \mathbf{Y}_k &= [y_{1,k}, \dots, y_{n,k}]^T \\ \mathbf{H} &= [h_1, \dots, h_n]^T \\ \mathbf{E}_k &= [\varepsilon_{1,k}, \dots, \varepsilon_{n,k}]^T \end{aligned} \tag{4.4}$$

When $r = 1$ (i.e., the system measures one single input), $\mathbf{X}$ is a scalar whose best estimate is obtained by computing a weighted average of all sensors, since it has been observed that sensors have different noise magnitudes, even if they have the same manufacturer reference [80].

Assuming that stochastic errors between different sensors are independent and not related with the input, a weighted average of S at time step k is described by:

$$\hat{\mathbf{X}}_k = w_{1,k} y_{1,k} + \cdots + w_{n,k} y_{n,k} \tag{4.5}$$

where $w_{i,k}$ denotes the weight of sensor i at time step k. Besides, weights are subject to the constraint:

$$\sum_{i=1}^{n} w_{i,k} = 1 \tag{4.6}$$

For simplicity, assume that all sensors perceive $\mathbf{X}$ with the same intensity. Then $\mathbf{H} = \mathbf{1}_n$, where $\mathbf{1}_n$ denotes all-ones vector of size n. Thus, using (4.2), equation (4.5) can be re-expressed as:

$$\hat{\mathbf{X}}_k = \mathbf{X}_k + \sum_{i=1}^{n} w_{i,k} \varepsilon_{i,k} \tag{4.7}$$

where the estimation error is:

$$\text{error}_k = \sum_{i=1}^{n} w_{i,k}\varepsilon_{i,k} \tag{4.8}$$

and hence, the variance of the estimation error can be expressed as:

$$\sigma^2(\text{error}_k) = \sum_{i=1}^{n} w_{i,k}^2\sigma^2(\varepsilon_{i,k}) \tag{4.9}$$

Here, $\sigma^2(\varepsilon_{i,k})$ denotes the variance of sensor i at time step k, defined as:

$$\sigma^2(\varepsilon_{i,k}) = \frac{1}{k}\sum_{j=1}^{k}(\varepsilon_{i,j} - \mu)^2 \tag{4.10}$$

where:

$$\mu = \frac{1}{k}\sum_{j=1}^{k}\varepsilon_{i,j} \tag{4.11}$$

Note that it is assumed that variances of sensors may change with time. This assumption allows to give a more realistic approach to the sensor system behavior since it captures the dynamism of low-frequencies noises present in sensors' measurements.

Finally, the best estimation of $\mathbf{X}_k$ is obtained by minimizing (4.9). Such minimization results in an expression that relates weights and variances of all sensors in the system, as presented below [64, 77, 80]:

$$w_{i,k} = \left(\sum_{j=1}^{n}\frac{\sigma^2(\varepsilon_{i,k})}{\sigma^2(\varepsilon_{j,k})}\right)^{-1} \tag{4.12}$$

Equation (4.12) represents the well known weighted average method, where all weights are computed each time step by means of the variances of all sensors. However, when $r \geq 2$ (i.e., the system S measures two or more signals) using this method becomes extremely complicated and sometimes even impossible, making it necessary to implement of a different method for the data fusion process. One of the most implemented methods for multiple signal estimation is the Kalman filter [81]. The Kalman filter is a set of mathematical equations that use a predictor-corrector type estimator that address the general problem of estimating vector $\mathbf{X}$ by combining vector of measurements described in (4.3) and a prediction of $\mathbf{X}_k$ given by [72]:

$$\mathbf{X}_k^- = \Phi\hat{\mathbf{X}}_{k-1} + \nu_{k-1} \tag{4.13}$$

Here, Φ is the $r \times r$ state transition matrix which relates $\mathbf{X}_{k-1}$ with $\mathbf{X}_k$, and ν_{k-1} is the r size process noise vector. Estimation of $\mathbf{X}_k$ is carried out by relating the measurement described in (4.3) and prediction given in (4.13):

$$\hat{\mathbf{X}}_k = \mathbf{X}_k^- + \mathbf{K}_k\left(\mathbf{Y}_k - \mathbf{H}\mathbf{X}_k^-\right) \tag{4.14}$$

where:

$$\mathbf{K}_k = \mathcal{P}_k^- \mathbf{H}^T \left(\mathbf{H}\mathcal{P}_k^- \mathbf{H}^T + \mathbf{R}\right)^{-1}$$
$$\mathcal{P}_k^- = \Phi \mathcal{P}_{k-1} \Phi^T + \mathbf{Q}$$
$$\mathcal{P}_{k-1} = \mathcal{P}_{k-1}^- - \mathbf{K}_{k-1}\mathbf{H}\mathcal{P}_{k-1}^-$$
$$\mathbf{Q} = diag(\sigma^2(\mathbf{X}))$$
$$\mathbf{R} = diag([\sigma^2(\varepsilon_{1,k}), \ldots, \sigma^2(\varepsilon_{n,k})]) \tag{4.15}$$

Here, diag(υ) is a square diagonal matrix with the elements of vector υ on the main diagonal. In [82], it is shown that accurate estimates of $\mathbf{Q}$ and $\mathbf{R}$ are crucial to obtain a good estimate of $\mathbf{X}$. Moreover, if an implementation has a poor approximation of Φ or the implemented is a nonparametric method, then the knowledge of $\mathbf{R}$ becomes decisive for a good performance of the filter.

To illustrate this, we retake the Kalman-Takens filter presented in chapter 3. This procedure replaces the parametric model described in (4.13) with dynamics reconstructed from delay vectors formed from previous measurements. Quality in the estimation of $\mathbf{R}$ depends entirely on the reconstruction of Φ, which at the same time, depends on the size of the dictionary of delay vectors. Consequently, to obtain a good estimation of $\mathbf{R}$ a big amount of recorded data is needed.

For both, the weighted average and the Kalman filter, the estimation of sensors' variances is essential. A straightforward way of obtaining such information is by using parameters extracted from sensors datasheets. However, by doing this, a poor performance is obtained, especially if all sensors in an array have the same manufacturer reference [80].

In [64], sensors' variances are estimated at each time step using a golden device (i.e., a reference sensor). The major drawback of using a golden device it the need of ensuring that the reference sensor is fault-free at any time step, which is not suitable for implementations where faults in the elements of the system may occur.

A better option to compute sensors' variances is the MINQUE algorithm presented for the first time in [79] and more recently extended in [83]. This algorithm allows the estimation of all variances without the need of a reference sensor. Necessary conditions for the use of this algorithm are 1) all sensors must be calibrated, 2) all devices must measure the same signals regardless of their intensities (all of them different from zero), and 3) the number of sensors must be higher than the number of measured signals. It is straightforward to note that sensor array systems fulfill such conditions.

Further, we will demonstrate that by implementing the MINQUE method in a sensor array system, detection of faults and uncalibrations in sensors can be carried out under certain conditions.

Based on this, in this chapter we propose a MINQUE-based algorithm that, in addition to computing variances of sensors, is capable of detecting the presence of failures in the array elements. The proposal is an on-line method that is able to follow changes in sensors' variances caused by different factors. We hereby assume that malfunction of a sensor will appear as an abnormal noise level in low- or high-frequencies ; this includes saturations, biases, blurring, scale factor errors, and so on. Under this assumption, the proposed algorithm is able of detecting all these malfunctions.

This proposal is divided into 3 parts: 1) estimation of sensor variances, 2) detection and identification of faults, and 3) detection and reincorporation of recovered sensors. Next, we present the theoretical bases for the estimation of sensors' variances.

4.3 Estimation of sensors variances

To carry out the estimation of sensors' variances, an online version of the MINQUE algorithm presented in [79] is proposed. This approach leverages correlation in the collection of sensors in a system that meets the three following constraints: 1) all sensors must be calibrated, 2) all devices must measure the same signals regardless of their intensities (all of them different from zero), and 3) the number of sensors must be higher than the number of measured signal. The general idea is to project measurement vector $\mathbf{Y}_k$ onto the orthogonal complement of the observation matrix $\mathbf{H}$. Once this is done, the remaining is a linear combination of stochastic errors coming from the n sensors, which are quantified each time step using an on-line variance estimator. Finally, taking advantage of some linear algebra properties, individual variances of sensors are estimated. This procedure is detailed below.

Consider a sensor array system S consisting of n calibrated sensors that measure the same r physical inputs, with $n > r$. Then, measurement vector $\mathbf{Y}_k$ is described by equation (4.3) where it is assumed that matrix $\mathbf{H}$ is known. Due to the redundancy on measurements, $\mathbf{Y}_k$ lies in a subspace of dimension r. Let $\mathbf{P}$ be the orthogonal projection matrix onto the orthogonal complement of $\mathbf{H}$ (the method to obtain this matrix is described later (4.3.1)). Hence, $\mathbf{P}$ is a $n \times n$ matrix with rank $n - r$ that satisfies the constraint:

$$\mathbf{PH} = 0 \tag{4.16}$$

and therefore, multiplication of matrix $\mathbf{P}$ with measurement vector $\mathbf{Y}_k$ results in:

$$\mathbf{PY}_k = \mathbf{PE}_k \tag{4.17}$$

Now, let $\sigma^2(v_k)$ be an on-line function that computes the variance of each element in a vector v from the first to the $k-$th time step, i.e. $\sigma^2(v_k) = \frac{1}{k}\sum_{j=1}^{k}(v_j - \bar{v})^2$, where $\bar{v} = \frac{1}{k}\sum_{j=1}^{k} v_j$. If this function is applied to (4.17), then the resulting variance is:

$$\sigma^2(\mathbf{PY}_k) = \mathbf{P}^{\circ 2}\, \sigma^2(\mathbf{E}_k) \tag{4.18}$$

where $\mathbf{P}^{\circ 2}$ denotes the Hadamard product of $\mathbf{P}$ with itself (i.e., $\mathbf{P}^{\circ 2} = \mathbf{P} \circ \mathbf{P}$), this results in a $n \times n$ matrix with full rank (proof in section 4.3.2), and therefore, invertible. Consequently, variances of sensors at time step k can be estimated as follows:

$$\hat{\sigma}^2(\mathbf{E}_k) = (\mathbf{P}^{\circ 2})^{-1}\, \sigma^2(\mathbf{PY}_k) \tag{4.19}$$

Expression above is an on-line variant of the MINQUE algorithm. Observe that matrix $\mathbf{P}$ does not need to be calculated each time step unless the topology of the array changes, or sensors are added (or removed) to the data fusion process. This last case is analyzed in detail in section 4.4. Next, it is explained how to compute matrix $\mathbf{P}$.

4.3.1 Orthogonal projection P

The construction of matrix $\mathbf{P}$ depends entirely on the definition of observation matrix $\mathbf{H}$. It is worthwhile noting that the presence of scale factor errors modifies the definition of such matrix $\mathbf{P}$. An analysis when scale factor errors are present during the implementation is given in section 4.4.1. Here, it is assumed that all sensors are calibrated before deployment and that no systematic errors occur during the execution of the algorithm.

Considering that $\mathbf{H}$ is the $n \times r$ known matrix with rank equal to r, computation of $\mathbf{P}$ is carried out as follows [65]:

$$\mathbf{P} = \mathbf{I}_n - \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \tag{4.20}$$

where $\mathbf{I}_n$ is the identity matrix of size n. Observe that if one sensor is removed (or added) from $\mathbf{Y}_k$, then the recalculation of matrix $\mathbf{P}$ is required. This action is carried out when a fault or uncalibration in a sensor is detected and needs to be discarded (in the same way, it is carried out when a sensor recovers from a fault or is replaced and requires being (re)incorporated to the data fusion process). The computation of $\mathbf{P}$ can be performed straightforwardly as long as $\mathbf{H}$ is known.

4.3.2 Existence of $(\mathbf{P}^{\circ 2})^{-1}$

To show that the Hadamard product $\mathbf{P}\circ\mathbf{P}$ is invertible, some known results from linear algebra are used. If required, the reader can consult [65, 84] to verify any of the lemmas or theorems presented below. First, two important properties of $\mathbf{P}$ are highlighted:

Lemma 1. *The orthogonal projection matrix $\boldsymbol{P}$ is symmetric and idempotent.*

This leads us to the next lemma:

Lemma 2. *A symmetric and idempotent matrix is positive semi-definite with the eigenvalues 0 and 1.*

Then, the following theorem shows that the Hadamard product of two semi-definite matrices results in a positive semi-definite matrix.

Theorem 3 (Schur product theorem)**.** *Suppose $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two square positive semi-definite matrices of size n. Then, $\boldsymbol{M}_1\circ\boldsymbol{M}_2$ is also positive semi-definite.*

Remember that only positive definite matrices are invertible. Although theorem 3, $\mathbf{P}^{\circ 2}$ is still invertible under certain conditions. For example, it is known that for $n \leq 2$, the Hadamard power $\mathbf{P}^{\circ 2}$ cannot be positive definite [79], but when $n \geq 3$ the situation is different. In [85], some interesting results related to Hadamard products are presented, from which the next sufficient condition for the non-singularity of $\mathbf{P}^{\circ 2}$ is taken:

Lemma 4. *Let $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ be two positive semi-definite matrices of size $n\times n$, with $n \geq 3$. If $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ have no zero main diagonal entries, then $\boldsymbol{M}_1\circ\boldsymbol{M}_2$ is positive definite if there is a $q \in \{1, \ldots, n\}$ such that $rank(\boldsymbol{M}_2) > n - q$ and every principal minor of $\boldsymbol{M}_1$ of size q is positive.*

Thus, for sensor array systems with 2 sensors it is not possible to implement the MINQUE method [79]. For systems composed of 3 or more sensors, to ensure that $\mathbf{P}^{\circ 2}$ is non-singular, it is sufficient to verify the next two conditions: 1) each element in the main diagonal of $\mathbf{P}$ is different from zero, and 2) all the principal minors (hereinafter denoted as PM) of $\mathbf{P}$ of order q are positives. This last condition is equivalent to find a constant q such that, every list of q distinct columns of $\mathbf{P}$ is linearly independent. At this point, an interesting consequence of lemma 4 can be highlighted:

Lemma 5. *To ensure the non-singularity in $\boldsymbol{P}^{\circ 2}$, it is required that $2r < n$.*

Proof. Due to the fact that $rank(\mathbf{P}) = n - r$, and $rank(\mathbf{P}) > n - q$, then $r < q$. In addition, it should be noted that $q \leq n - r$ (otherwise, the list of q elements contains dependent columns), this implies that $2r < n$. □

Significantly, this result shows that to carry out the estimation of sensors' variances by means of the MINQUE method, the number of sensors used for this process needs to be strictly greater than twice the number of measured signals.

4.4 Presence of faults and recoveries

As mentioned earlier, this proposal is divided into 3 parts: 1) estimation of sensors' variances, 2) detection and identification of faults, and 3) detection and reincorporation of recovered sensors. It was shown in previous sections how the estimation of sensors' variances is carried out by computing (4.19). In this section, we will explain how detection and identification of faults, as well as the detection and reincorporation of recovered sensors are carried out.

4.4.1 Detection of faults

We hereby assume that malfunction of a sensor will appear as an abnormal noise level in low- or high-frequencies ; this includes saturations, biases, scale factor errors, and so on. Under this assumption, all the aforementioned errors can be detected by the proposed algorithm. In this work, we focused on the detection of changes in systematic errors during the runtime. These changes can be seen as uncalibrations of sensors due to different internal and/or external factors. Although this sounds restrictive, it actually covers more points than just detecting uncalibrated sensors. For example, a critical damage in a sensor can be seen as an uncalibrated sensor with a scale factor equal to zero (i.e., $(\alpha_T + \alpha_{i,k}) = 0$) and bias equal to some constant. From here on, terms *faulty sensor* and *uncalibrated sensor* will be used as synonyms to refer an uncalibrated sensor.

As mentioned before, it is assumed that all sensors are calibrated before their implementation. Therefore, at the beginning of the execution, only the presence of stochastic phenomena affects sensors' measurements. However, failures and uncalibration of sensors may appear during the runtime. The estimation of individual variances carried out with (4.19) is sensitive to the presence of systematic errors, which allows their detection. To show this, first let's rewrite equation (4.3) as follows:

$$\mathbf{Y}_k = diag(\mathbf{A}_k)\mathbf{H}\mathbf{X}_k + \mathbf{B}_k + \mathbf{E}_k \tag{4.21}$$

where:

$$\mathbf{A}_k = [(\alpha_T + \alpha_{1,k}), \ldots, (\alpha_T + \alpha_{n,k})]^T \tag{4.22}$$

$$\mathbf{B}_k = [\beta_{1,k}, \ldots, \beta_{n,k}]^T \tag{4.23}$$

Therefore, equation (4.17) results as below:

$$\mathbf{P}\mathbf{Y}_k = \mathbf{P}\, diag(\mathbf{A}_k)\mathbf{H}\mathbf{X}_k + \mathbf{P}\mathbf{B}_k + \mathbf{P}\mathbf{E}_k \tag{4.24}$$

Assuming $\alpha_T = 1$, if all sensors are calibrated then $\mathbf{B}_k = 0$ and $\mathbf{A}_k = \mathbf{I}_n$, resulting in the same equality as that presented in equation (4.17). However, if one or more sensors are uncalibrated, then:

$$\mathbf{P}\mathbf{Y}_k = \mathbf{P}\, diag(\alpha_k)\, \mathbf{H}\mathbf{X}_k + \mathbf{P}\mathbf{B}_k + \mathbf{P}\mathbf{E}_k \tag{4.25}$$

where

$$\mathbf{P}\, diag(\alpha_k)\, \mathbf{H} = \mathbf{P}\, diag(\mathbf{A}_k)\mathbf{H} \tag{4.26}$$

$$\alpha_k = [\alpha_{1,k}, \ldots, \alpha_{n,k}]^T \tag{4.27}$$

From (4.25) we note that $\mathbf{B}_k$ and $\mathbf{E}_k$ can be represented as a single phenomenon. This simplifies such equation as follows:

$$\mathbf{PY}_k = \mathbf{P}\, diag(\alpha_k)\mathbf{HX}_k + \mathbf{PE}_k \tag{4.28}$$

This simplification from (4.25) to (4.28) shows that, if there is an uncalibration in sensors related only to the bias error (no scale factor error), then it is possible to detect directly such uncalibration by means of the estimation of sensors' variances computed with equation (4.19). This means that changes in constant biases are detected as low-frequency noise effects, which makes sense since this parameter is no longer constant.

On the other hand, equation (4.26) allows to note that, when uncalibrated sensors are present in the system, the result of the projection $\mathbf{P}\, diag(\mathbf{A}_k)\mathbf{H}$ is a linear combination of columns of $\mathbf{P}$ and rows of $\mathbf{H}$ corresponding to all uncalibrated sensors. For example, if sensor 1 and 3 were uncalibrated, then the result would be the product of the first and third columns of $\mathbf{P}$ multiplied by the scale factor errors of sensors 1 and 3 (accordingly) and by the first and third rows of $\mathbf{H}$. Thus, when one or more sensors are uncalibrated, the variance of $\mathbf{PY}_k$ results in:

$$\sigma^2(\mathbf{PY}_k) = \left(\mathbf{P}\, diag(\alpha_k)\, \mathbf{H}\right)^{\circ 2} \sigma^2(\mathbf{X}_k) + \mathbf{P}^{\circ 2}\, \sigma^2(\mathbf{E}_k) \tag{4.29}$$

When there is a single uncalibrated sensor, the expression above is simplified to:

$$\sigma^2(\mathbf{PY}_k) = \mathbf{P}^{\circ 2}\, diag(\alpha_k^2)\, \mathbf{H}^{\circ 2}\, \sigma^2(\mathbf{X}_k) + \mathbf{P}^{\circ 2}\, \sigma^2(\mathbf{E}_k) \tag{4.30}$$

and therefore, estimation of sensors' variances given by $(\mathbf{P}^{\circ 2})^{-1}\sigma^2(\mathbf{PY}_k)$ results as follows:

$$\hat{\sigma}^2(\mathbf{E}_k) = diag(\alpha_k^2)\, \mathbf{H}^{\circ 2}\, \sigma^2(\mathbf{X}_k) + \sigma^2(\mathbf{E}_k) \tag{4.31}$$

From expression (4.31) it can be noted that, when a single sensor is uncalibrated, the estimation of sensors' variances remains undisturbed, except for the estimation that corresponds to the uncalibrated sensor, i.e.

$$\begin{cases} \hat{\sigma}^2(\mathbf{E}_{i,k}) = \sigma^2(\varepsilon_{i,k}) & \text{calibrated sensors} \\ \\ \hat{\sigma}^2(\mathbf{E}_{i,k}) = \alpha_k^2 h_i^{\circ 2} \sigma^2(\mathbf{X}_k) + \sigma^2(\varepsilon_{i,k}) & \text{uncalibrated sensor} \end{cases} \tag{4.32}$$

From the previous equation, it is straightforward to see that if the variance of the input signal is high then the uncalibrated sensor could be easily identify by detecting the inconsistent increment in the variance of one of the sensors.

Similarly, the detection of the uncalibrated sensor can also be carried when the variance of the input is low or equal to zero, even if from (4.32) is not easy to see it. To explain this, we retake equation (4.29) with a different arrangement:

$$\sigma^2(\mathbf{PY}_k) = \sigma^2(\mathbf{P}\, diag(\alpha_k)\mathbf{HX}_k + \mathbf{PE}_k) \tag{4.33}$$

Remember that it is assumed that at the beginning of the runtime, there is a period of time in which all the sensors are calibrated. During this period, the term $\mathbf{P}\, diag(\alpha_k)\mathbf{HX}_k$ will be equal to zero (since $\alpha_k = 0$). However, once an uncalibration occurs, this term will be different from

zero, which will cause a change in the variance computed with equation (4.33). Even more, if it is assumed that $\mathbf{X}_k \gg \mathbf{E}_k$, then this change will be considerably large.

In this way, a sensor can be identified as uncalibrated using the estimation of its variance. For example, it can be established that if the variance of a sensor exceeds a threshold, it will be marked as uncalibrated:

$$\begin{cases} \hat{\sigma}^2(\mathbf{E}_{i,k}) \leq \text{threshold} & \text{calibrated sensors} \\ \\ \hat{\sigma}^2(\mathbf{E}_{i,k}) > \text{threshold} & \text{uncalibrated sensor} \end{cases} \tag{4.34}$$

To set this threshold, variances computed at the beginning of the runtime can be used. This is exemplified in chapter 5. Once marked, the uncalibrated sensor is removed from the variance estimation process. This is done by removing measurement reported by sensor i from measurement vector $\mathbf{Y}_k$, as well as removing row corresponding to sensor i from matrix $\mathbf{H}$, assuming that sensor i is the uncalibrated device.

As demonstrated in lemma 5, this algorithm is capable of working under the presence of m faulty sensors, as long as $m < n - 2r$. So, if it is assumed that: 1) variances are estimated using only sensors that were not marked as uncalibrated in the previous time step, and 2) at any time step there is only one "new" uncalibrated sensor, then the proposed algorithm is able to give a correct estimation of variances of calibrated sensors, as well as to detect and discard the uncalibrated devices.

4.4.2 Detection and reincorporation of recovered sensors

As mentioned before, once an uncalibrated sensor is found it is removed from $\mathbf{Y}_k$ and $\mathbf{H}$. This means that its variance is no longer computed using equation (4.19). However, to detect recovered sensors (sensors that have been replaced or re-calibrated during the runtime) the estimation of variances of all sensors is required. Variances of uncalibrated sensors can still be computed using (4.19), however, it must be ensured the presence of at most one uncalibrated measurement in $\mathbf{Y}_k$ to ensure the correctness of this process. This is done as follows:

Let $\bar{S} \subset S$ be the set of sensors marked as uncalibrated. For each sensor $j \in \bar{S}$:

- Reincorporate measurement of sensor j to $\mathbf{Y}_k$
- Reincorporate the corresponding row of sensor j to $\mathbf{H}$
- Compute matrix $\mathbf{P}$
- Compute $\hat{\sigma}^2(\mathbf{E}_{k,j})$ using (4.19)

Once estimated, the variance of each uncalibrated sensor is used to determine if there is a recovery or not:

- Evaluate expression (4.34), if sensor j is detected as uncalibrated then remove it from $\mathbf{Y}_k$ and $\mathbf{H}$, otherwise mark it as calibrated (sensor $j \notin \bar{S}$)

These steps allow to estimate variances of all uncalibrated sensors by reincorporating them (one by one) to the set of calibrated devices. Once these variances are estimated, detection of recoveries is carried out similarly as the detection of uncalibrated sensors, using expression (4.34).

Next, the complete process for estimation of sensors' variances, fault detection, and reincorporation of recovered sensors are summarized.

4.5 Overview of the algorithm

The proposed algorithm based on MINQUE method is summarized in Figure 4.2, as well as the pseudo-code shown in algorithm 1. This method is used to: 1) estimation of sensors' variances, 2) detection and identification of faults, and 3) detection and reincorporation of recovered / replaced sensors. It is able to work under the presence of faults, as long as the number of faulty sensors is less than $n - 2r$. This proposed algorithm can be combined with the weighted average described in (4.12) or Kalman filter depicted in (4.15) for carrying out a data fusion process.

In Algorithm 1, once that a sensor is marked as uncalibrated, it is removed from the data fusion process. However, a different approach can be carried out using some on-line calibration method such as those proposed in [66] and [86]. This point is beyond the scope of this thesis.

On the other hand, function runVar(υ) denotes the on-line algorithm that estimates the variance of each element in a vector υ, assuming that these elements are independent samples that belong to different discrete sequences. Note that this function is different from variance(υ), which estimates the variance between the elements of υ. For function runVar(υ), Welford's algorithm can be used [87, p. 232], but it requires the implementation of variables that increment each time step, making it impractical for real applications. Thus, for the implementation of such function, a Welford's algorithm with an exponential smoothing instead of an incremental counter is used. The pseudo code of this modified function is presented in algorithm 2 in the Appendix. For this, the simplest form of exponential smoothing is implemented, but it can be replaced with a more complex function. Matlab code of both algorithms (algorithms 1 and 2) are included in the Appendix.

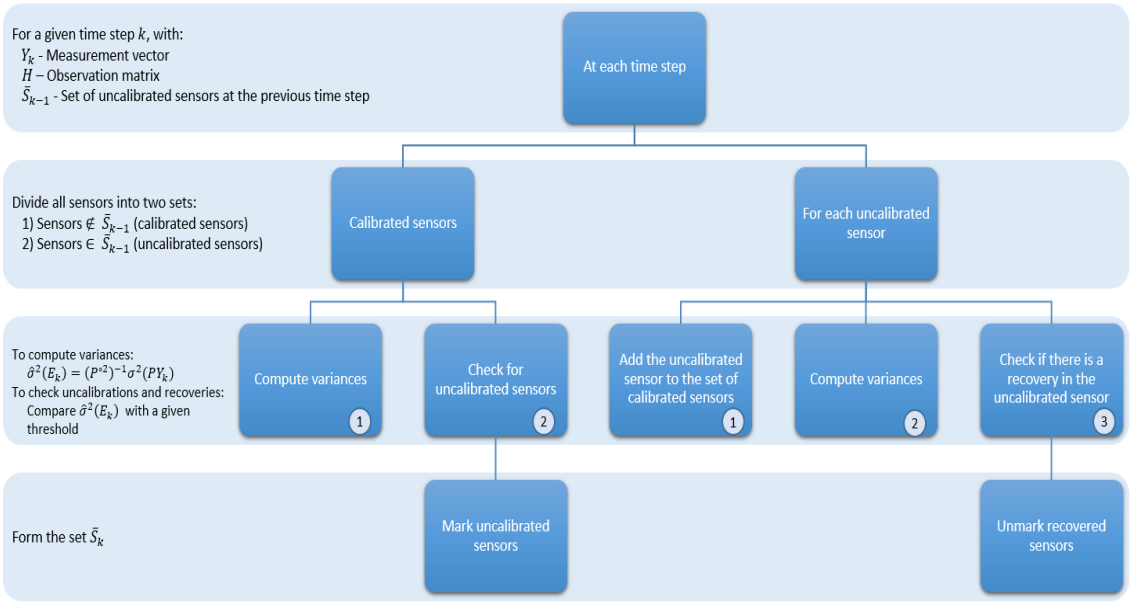


Figure 4.2 – Schematic of proposed algorithm.

Algorithm 1 Estimation of sensors' variances

Require:
$\mathbf{Y}_k = [y_{1,k}, \dots, y_{n,k}]^T$ - Measurements reported by all sensors in S at time step k,
$\mathbf{H}$ - Observation matrix corresponding to all sensors in S,
$\bar{S}_{k-1}$ - set of uncalibrated sensors at time step $k-1$
ξ - Threshold for the identification of uncalibrated sensors (equation (4.34))
Ensure:
$\hat{\sigma}^2(\mathbf{E}_k)$ - Estimation of sensors' variances at time step k ,
$\bar{S}_k$ - set of uncalibrated sensors at time step k

Remove from $\mathbf{Y}_k$ measurements coming from sensors $\in \bar{S}_{k-1}$
Remove from $\mathbf{H}$ rows corresponding to sensors $\in \bar{S}_{k-1}$
Compute $\mathbf{P}$ using $\mathbf{H}$
$\sigma^2(\mathbf{PY}_k) = \text{runVar}(\mathbf{PY}_k, \mu(\mathbf{PY}_{k-1}), \sigma^2(\mathbf{PY}_{k-1}))$
$\hat{\sigma}^2(\mathbf{E}_k) = (\mathbf{P}^{\circ 2})^{-1}\ \sigma^2(\mathbf{PY}_k)$ ▷ Sensors' variances
if any $\hat{\sigma}^2(\mathbf{E}_{i,k})) > \xi$ **then** ▷ Detection of uncalibrated sensor
    Mark sensor i as uncalibrated ($\in \bar{S}_k$)
    Remove from $\mathbf{Y}_k$ the row corresponding to sensor i
    Remove from $\mathbf{H}$ the row corresponding to sensor i
end if
for each uncalibrated sensor $j \in \bar{S}_{k-1}$ **do**
    Add measurement given by sensor j to $\mathbf{Y}_k$
    Add row corresponding to sensor j to $\mathbf{H}$
    Compute $\mathbf{P}$ and $\sigma^2(\mathbf{PY}_{i,k})$
    if $\hat{\sigma}^2(\mathbf{E}_{j,k})) \leq \xi$ **then** ▷ Detection of recovered sensors
        Mark sensor j as calibrated ($j \notin \bar{S}_k$)
    else
        Remove sensor j from $\mathbf{Y}_k$ and $\mathbf{H}$
    end if
end for
return $\hat{\sigma}^2(\mathbf{E}_k)$, $\bar{S}_k$

4.6 Extension of this work for two or more physical inputs

In previous sections, we presented the proposed algorithm under scenarios where a sensor array measured a single input, however, this algorithm can be used in scenarios where two or more different inputs are measured. For example, the standard model equation of the three-axis accelerometers can be expressed as follows [31, 52]:

$$a = a_{\text{body}} + \mathring{\omega} \times d + \omega \times (\omega \times d) + \varepsilon \tag{4.35}$$

For the sake of simplicity, the subscript k that indicates the time step has been omitted. Rewriting equation (4.35) in matrix form, we get the following [52]:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} a_{\text{body}_x} \\ a_{\text{body}_y} \\ a_{\text{body}_z} \end{bmatrix} + \begin{bmatrix} \mathring{\omega}_x \\ \mathring{\omega}_y \\ \mathring{\omega}_z \end{bmatrix} \times \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \left(\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \right) + \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} \tag{4.36}$$

Here a_{body_x}, a_{body_y} and a_{body_z} denote the body frame acceleration in axis x, y and z respectively. Now, expanding equation (4.36) we get:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} a_{\text{body}_x} \\ a_{\text{body}_y} \\ a_{\text{body}_z} \end{bmatrix} + \begin{bmatrix} \mathring{\omega}_y d_z - \mathring{\omega}_z d_y \\ \mathring{\omega}_z d_x - \mathring{\omega}_x d_z \\ \mathring{\omega}_x d_y - \mathring{\omega}_y d_x \end{bmatrix} + \begin{bmatrix} \omega_y(\omega_x d_y - \omega_y d_x) - \omega_z(\omega_z d_x - \omega_x d_z) \\ \omega_z(\omega_y d_z - \omega_z d_y) - \omega_x(\omega_x d_y - \omega_y d_x) \\ \omega_x(\omega_z d_x - \omega_x d_z) - \omega_y(\omega_y d_z - \omega_z d_y) \end{bmatrix} + \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} \tag{4.37}$$

Let's consider the situation where we have a two-axis sensor (without loss of generality, axes x and y) placed at a distance d_x from the center, and along the x-axis. This implies that d_y and d_z are zero. Now, assume that the sensor is exposed to an angular motion only along the central axis (z-axis). This makes ω_x, $\mathring{\omega}_x$, ω_y and $\mathring{\omega}_y$ zero. Substituting these values in the above equation, we get the following:

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} a_{\text{body}_x} - \omega_z^2 d_x + \varepsilon_x \\ a_{\text{body}_y} + \mathring{\omega}_z d_x + \varepsilon_y \end{bmatrix} \tag{4.38}$$

Expression (4.38) is given in [52] for the angular rate estimation around the z-axis through the data fusion of outputs coming from two accelerometers and one gyroscope. Let's use this configuration to show how our proposed algorithm can be used for measuring two or more signals.

For this, first observe from equation (4.38) that both axes (axes x and y) measure two different inputs: x-axis measures a_{body_x} and ω_z^2, while y-axis measures a_{body_y} and $\mathring{\omega}_z$. Distance d_x is assumed to be known as the topology of the system is usually known. To simplify the analysis, let's take just x-axis. Now, instead of two accelerometers (as in [52]) let's assume the use of three accelerometers under the same configuration. Then, measurements reported by the three accelerometers would be:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & -d_{x_1} \\ 1 & -d_{x_2} \\ 1 & -d_{x_3} \end{bmatrix} \begin{bmatrix} a_{\text{body}_x} \\ \omega_z^2 \end{bmatrix} + \begin{bmatrix} \varepsilon_{x_1} \\ \varepsilon_{x_2} \\ \varepsilon_{x_3} \end{bmatrix} \tag{4.39}$$

Comparing equation (4.39) with the general formula of a sensor's output given in (4.3) we obtain the following parameters:

$$\mathbf{Y} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & -d_{x_1} \\ 1 & -d_{x_2} \\ 1 & -d_{x_3} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} a_{\text{body}_x} \\ \omega_z^2 \end{bmatrix} \tag{4.40}$$

In this case, the data fusion process can be carried out by the weighted average presented in (4.12). Note that if we use the two axes (x and y-axis) at the same time in the data fusion process, the number of measured inputs will be 4, however, the resulted matrix $\mathbf{H}$ will have zeros in its construction, which increases the probability that matrix $\mathbf{P}$ has no inverse, and therefore the algorithm cannot be used. This can be solved by using the Kalman filter to estimate the inputs, and then the proposed algorithm will be used as a subroutine to estimate the variances in the measurements.

As the reader can appreciate, the algorithm shows to be useful in the estimation of multiple entries, however, due to the limited time of this thesis, it was decided to leave this as future work.

Chapter 5

Assessment of the proposed algorithm

In chapter 4, an algorithm for estimation of sensors' variances in array systems was presented. Then, data fusion in such systems can be done by combining this algorithm with other methods such as a weighted average or a Kalman filter (described in chapter 4). In this chapter, we exemplified this process of data fusion by means of simulations into the Matlab Simulink environment, as well as a real implementation in a system composed of 12 MEMS accelerometers.

5.1 Simulations

Hereafter, the validity of Algorithm 1 is assessed by numerical simulations. These simulations are carried out into the MATLAB Simulink environment, where sensors are modeled using the simulink block presented in chapter 2 and [39].

For this simulations, a sensor array system composed of 6 three-axis gyroscopes is assumed. The sensed angular velocity is independent of the gyroscope location within the array [31]. Consequently, measurements reported by the $i-$th triad at time step k are

$$\begin{bmatrix} \omega_{i,\vec{x},k} \\ \omega_{i,\vec{y},k} \\ \omega_{i,\vec{z},k} \end{bmatrix} = \begin{bmatrix} \omega_{\vec{x},k} \\ \omega_{\vec{y},k} \\ \omega_{\vec{z},k} \end{bmatrix} + \begin{bmatrix} \varepsilon_{i,\vec{x},k} \\ \varepsilon_{i,\vec{y},k} \\ \varepsilon_{i,\vec{z},k} \end{bmatrix} \tag{5.1}$$

where $\omega_{\vec{x},k}$, $\omega_{\vec{y},k}$ and $\omega_{\vec{z},k}$ denote angular velocities in $\vec{x}$, $\vec{y}$ and $\vec{z}$ axes. For the sake of simplicity, the next analysis will be done only for the $\vec{x}$ axis. The analysis is the same for $\vec{y}$ and $\vec{z}$. Thus, measurement given by the $i-$th sensor is reduced to

$$\omega_{i,k} = \omega_k + \varepsilon_{i,k} \tag{5.2}$$

Consequently, variables used in the proposed algorithm are:: $\mathbf{Y}_k = [\omega_{1,k}, \ldots, \omega_{6,k}]^T$, $n = 6$, $\mathbf{X} = \omega_k$ (and therefore $r = 1$), matrix $\mathbf{H} = \mathbf{1}_n$, and $m < 4$. Due to the definition of $\mathbf{H}$, $\mathbf{P}$ results in a $n \times n$ matrix with $rank(\mathbf{P}) = n - 1$ and defined as follows:

$$\mathbf{P} = \begin{cases} \frac{n-1}{n} & \text{, for } \mathbf{P}_{i,i} \\ \frac{-1}{n} & \text{, for } \mathbf{P}_{i,j} \mid i \neq j \end{cases} \tag{5.3}$$

Thus, the existence of $(\mathbf{P}^{\circ 2})^{-1}$ is ensured by the following facts:

- all diagonal entries in $\mathbf{P}$ are non zero,
- it exists a constant $q = 2$ such that $rank(\mathbf{P}) > n - q$,
- every PM of $\mathbf{P}$ of size q is equal to $\left(\frac{n-1}{n}\right)^2 - \left(\frac{1}{n}\right)^2$, which is positive.

This analysis is valid for all cases when $\mathbf{H} = \mathbf{1}_n$, as long as $n \geq 3$ (see section 4.3). Therefore, if m sensors are removed from $\mathbf{H}$, and $\mathbf{P}$ is recomputed, the proposed algorithm is still valid as long as $n - m \geq 3$.

For the detection of uncalibrated sensors, the condition used to determine when a sensor is uncalibrated (described in (4.34)) is set as follows:

$$\text{Uncalibrated sensor} = \begin{cases} \hat{\sigma}^2(\varepsilon_{i,k}) > 2\,\sigma^2(\varepsilon_{max}) & \text{True} \\ \\ \text{Otherwise} & \text{False} \end{cases} \tag{5.4}$$

where $\hat{\sigma}^2(\varepsilon_{max})$ denotes the highest variance in S found during a pre-calibration step. For these simulations $\sigma^2(\varepsilon_{max}) = \sigma^2(\varepsilon_6)$.

For the function runVar(v) a smoothing factor $\gamma = 0.999$ is empirically selected. The effect on changing this factor has been left as future work.

As mentioned earlier, an initial calibration in all sensors is assumed. Simulink block presented in [39] is used for simulation of system S. Parameters used to configure sensor models are shown in Table 5.1, where Q, B and K denote the white noise, $1/\mathrm{f}$ noise, and $1/\mathrm{f}^2$ noise magnitudes, respectively. To show the result of this parametrization, an Allan DEviation (ADEV) graph resulting from a time domain analysis is presented in Figure 5.1. This graph is obtained using the AVAR Matlab function [59] and sets of 10^6 samples coming from sensors' outputs, setting a zero input and an output sample time of 5 ms in all the sensors. Note that all parameters Q, B and K were set up in such a way that all simulated sensors belong to the tactical grade [52].

Table 5.1 – Parameters used for sensor model simulations.

	$Q\left(\mathrm{rad/s}\times\sqrt{\mathrm{s}}\right)$	$B\left(\mathrm{rad\,s^{-1}}\right)$	$K\left(\mathrm{rad/s}/\sqrt{\mathrm{s}}\right)$
Sensor 1	3×10^{-3}	8×10^{-4}	9×10^{-5}
Sensor 2	4×10^{-3}	9×10^{-4}	1×10^{-4}
Sensor 3	5×10^{-3}	1×10^{-3}	1×10^{-4}
Sensor 4	6×10^{-3}	1.5×10^{-3}	1.3×10^{-4}
Sensor 5	7×10^{-3}	1.8×10^{-3}	1.5×10^{-4}
Sensor 6	8×10^{-3}	1.7×10^{-3}	1.6×10^{-4}

For the measured angular velocity ω_k, a sine wave signal of $8\,\mathrm{rad\,s^{-1}}$ (460 $^\circ/\,\mathrm{sec}$) and frequency of 20Hz is generated with the *Signal generator* Matlab Simulink block. For the sensor array system, a sampling frequency of 200 Hz is used. The considered simulation time is 1000 s (2×10^5 samples).

Finally, to illustrate the usefulness of the proposed algorithm, and taking advantage of the fact that $r = 1$, the estimation of $\mathbf{X}_k$ will be done using the weighted average method described in equation (4.12).

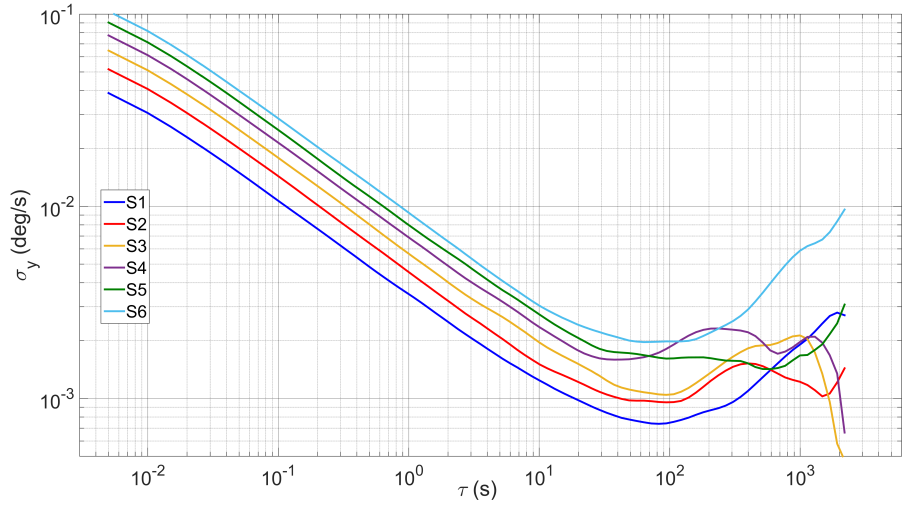


Figure 5.1 – Allan deviation graph of 6 similar sensors with different noise levels.

5.1.1 Assessment of calibrated devices

Next, the performance of the proposed algorithm is evaluated under a fault-free scenario. Figure 5.2 shows sensors' variances estimated with the proposed algorithm (equation (4.32)), which correspond to the noise levels shown in Figure 5.1.

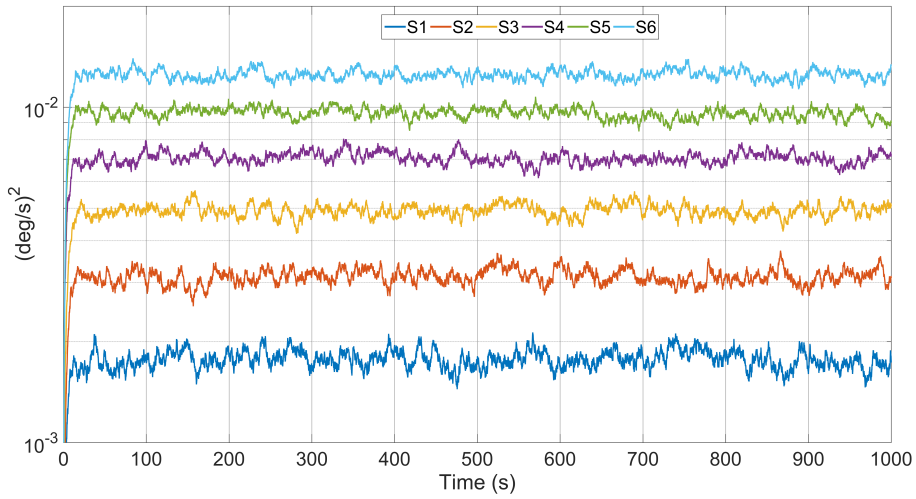


Figure 5.2 – Individual variance estimation of 6 fault-free sensors.

Figure 5.3 and Figure 5.4 show the computation of weights for the data fusion process and the error resulting from the estimation of the input signal, respectively. In Figure 5.3, weights correspond to the noise levels shown in Figure 5.1 and Figure 5.2, with the expected inverse relationship. For Figure 5.4, a window Root Mean Square (window RMS) of the error in the estimation of the input signal is performed, which allows to capture the dynamic change in the estimation error. This is done using a window size of 200 samples (equivalent to 1s of data). In this figure,

error obtained with the proposed algorithm (by means of a weighted average defined in (4.12)) is compared against the error obtained with a simple average and the Least SQuare (LSQ) method. For the LSQ, inputs and outputs of sensors corresponding to the first 400s of simulation are used. Figure 5.4 shows how the performance obtained with the proposed algorithm reaches that obtained with LSQ method, in real-time and without input knowledge. Remember that the LSQ method is not a real-time algorithm and requires input knowledge.

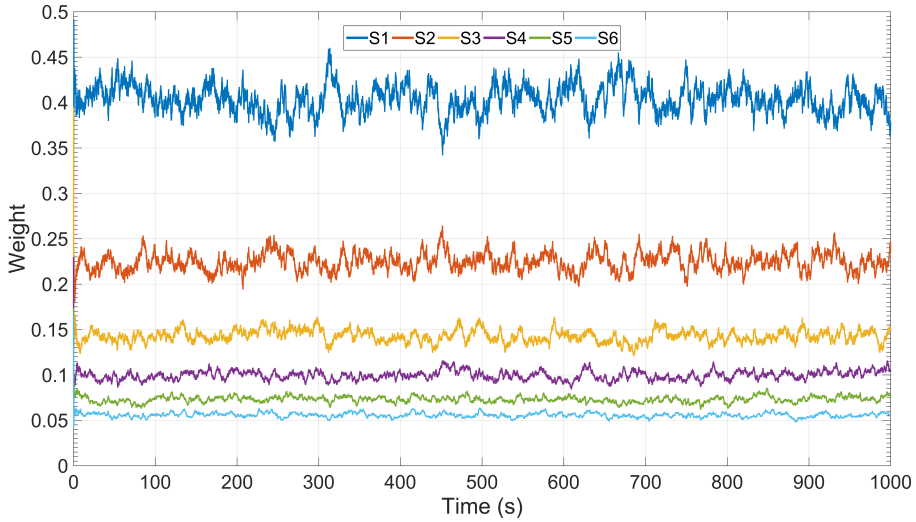


Figure 5.3 – Assignment of weights resulting from estimation of sensors' variances for a 6 fault-free sensor system.

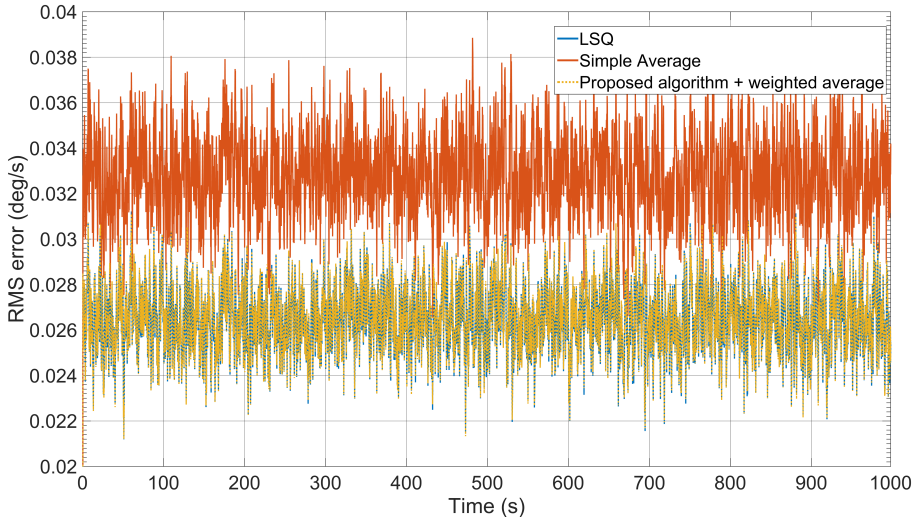


Figure 5.4 – A window RMS error for assessment of the proposed algorithm (by means of a weighted average), and its comparison with a simple average, and a least square method (LSQ).

5.1.2 Presence of faults and recoveries

Now, to show how the proposed algorithm is able to carry out fault detection, scenarios where different faults affect the system are presented. First, a critical damage in sensor 1 is simulated at 400s. As mentioned earlier, critical damage in a sensor can be seen as a device with a scale factor equal to zero (i.e., $\alpha_{1,k} = -1$). Second, a scale factor error in sensor 2 is induced at 500s, where $\alpha_{2,k} = -0.20$. Finally, a scale factor error of $\alpha_{3,k} = -0.15$ is simulated in sensor 3 from 600s to 700s. Note how it is assumed that the error in sensor 3 is corrected at 700s. The intention of this is to show that the proposed algorithm is capable of reincorporating recovered devices. For all these faults, constant biases are set up to zero.

Figure 5.5 shows the estimation of sensors variances. Here, the horizontal dotted line in black is the threshold (denoted as ξ) used to discriminate uncalibrated sensors. Initially, all sensors are free of faults, therefore estimated variances are below this threshold. If the estimated variance of a sensor goes above this threshold, then this sensor is marked as uncalibrated and it is removed from the measurement vector $\mathbf{Y}_k$ and the observation matrix $\mathbf{H}$. On the other hand, if the estimated variance of a faulty sensor goes below the threshold, then this sensor is marked as a recovery and it is reincorporated to $\mathbf{Y}_k$ and $\mathbf{H}$. For example, in Figure 5.5 the estimated variance corresponding to sensor 1 goes above ξ at 400s, where it is marked as uncalibrated. Similarly, at 500s and 600s sensors 2 and 3 are marked as uncalibrated once that their estimated variances go above the threshold ξ. Finally, recovery of sensor 3 is detected at 760s when the estimated variance of such sensor goes below ξ.

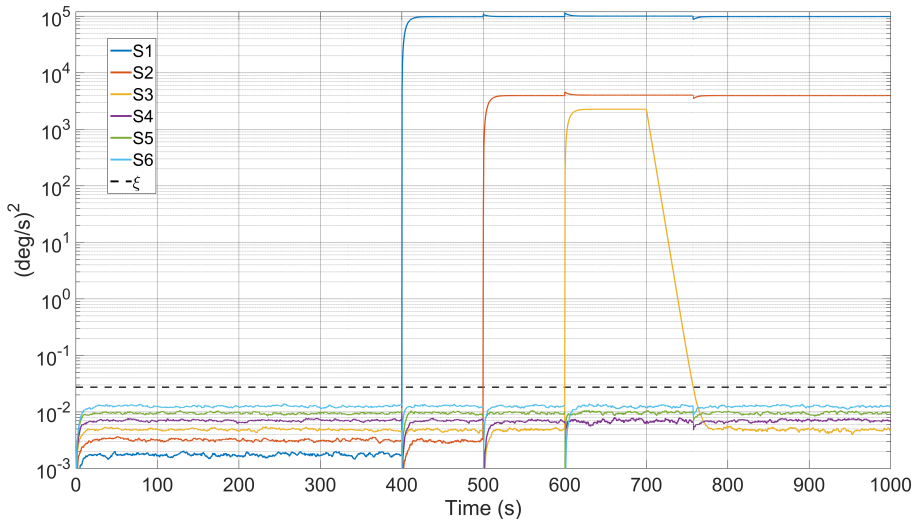


Figure 5.5 – Individual variance estimation of a system of 6 sensors. Sensor 1, 2 and 3 are identified and marked as uncalibrated at 400s, 500s and 600s, respectively. Recalibration of sensor 3 is detected at 760s. Horizontal dotted line in black shows the threshold used to determine when sensors are uncalibrated.

The reason the algorithm takes 60 seconds to detect the recalibration of sensor 3 is due to the smoothing factor (called γ) using in the runVar(u) function. We have observed that this time can be decreased by decreasing the value of γ, however, this increments the error in the estimation of the input. As previously stated, for the moment the analysis of the impact of γ in the performance of the proposed algorithm has been left as future work, however, for the moment we conclude that

the choice of the value of this variable is a trade off between the error in the input estimation and the speed in the error detection.

Once identified faulty sensors are removed from the data fusion process by assigning them a zero weight, as shown in Figure 5.6. In the case of sensor 3, once it is recovered, it is reincorporated to the data fusion process as soon as its estimated variance becomes less than the threshold established in (5.4), as illustrated in figures 5.5 and 5.6.

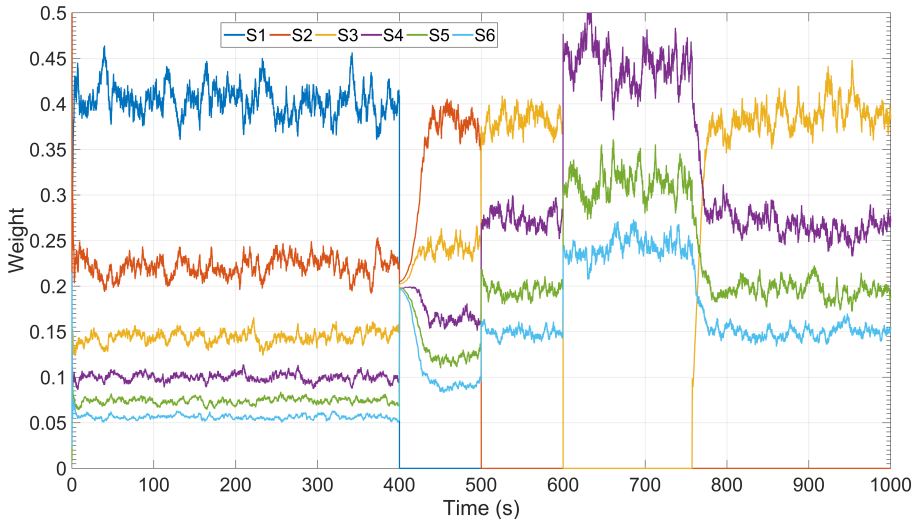


Figure 5.6 – Assignment of weights computed by proposed algorithm for a 6 sensor system. Faults due to critical damage, and scale factor errors are well suppressed once faulty sensors are identified.

Finally, Figure 5.7 shows the window RMS error obtained through LSQ, a simple average and the proposed algorithm (by means of a weighted average) when faults are present in the sensing system. Like in section 5.1.1, for LSQ data collected during the first 400s of simulation is used to execute the method. Observe that once faults are induced in sensors, the error obtained with the proposed algorithm is lower than those obtained with the simple average and the LSQ. In fact, we observed that under the presence of faults, the performance achieved by the proposed algorithm is equivalent to a "sectioned LSQ", i.e. an implementation of the LSQ by sections using only fault-free sensors. To show this, in Figure 5.7 we present the performance reached by an implementation of the LSQ in the following way: 1) using data coming from all sensors for the time period 0s to 400s, 2) using data coming from sensors 2 to 6 for the time period 400s to 500s, 3) using data coming from sensors 3 to 6 for the time period 500s to 600s, 4) using data coming from sensors 4 to 6 for the time period 600s to 700s, and 5) using data coming from sensors 3 to 6 for the time period 700s to 1000s. Note that in real applications, sectioned LSQ cannot be implemented because it requires the knowledge of the input for the complete execution time. However, this is not the case of the proposed algorithm, which is able to reach the same RMS error as the sectioned LSQ without any input knowledge, as shown in Figure 5.7.

5.2 Experimental results

Now, the proposed algorithm is implemented in an in-house embedded system composed of 12 MEMS based sensors (Figure 5.9). The system is composed of eight LIS2DH, one MPU9250, one

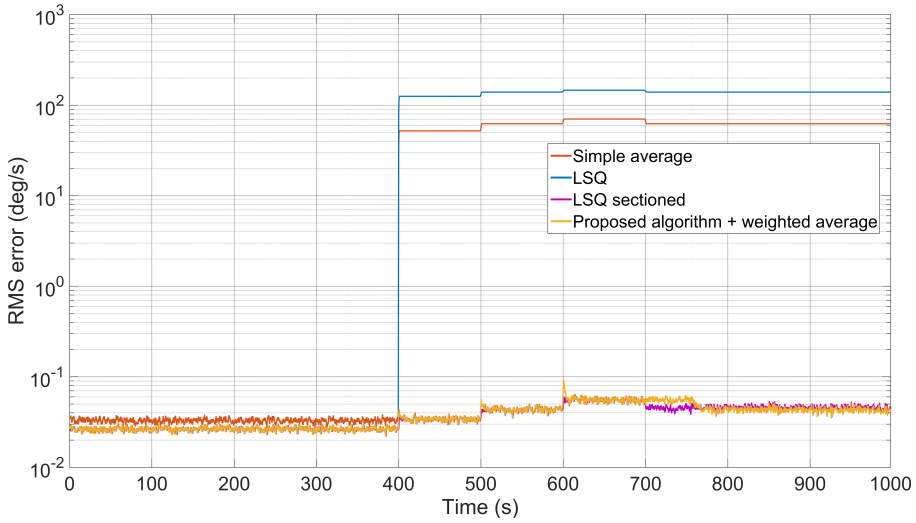


Figure 5.7 – RMS error obtained under the presence of faults.

BMA280, one ADXL343, and one MMA8653. Each chipset is a three-axis accelerometer whose selectable full scales are $\pm 2g/\pm 4g/\pm 8g/\pm 16g$, and output data rates go from 1Hz to 5.3kHz. For this implementation, a full scale of $\pm 2g$ and an output data rate of 200Hz are set up for all the devices.

Figure 5.8 shows the ADEV graph corresponding to the four different models of accelerometers included in the system: MPU9250, BMA280, ADXL343, and MMA8653; as well as the graph corresponding to the mean of the 8 similar sensors (LIS2DH). These graphs are obtained using the AVAR Matlab function [59] and sets of 10^6 samples coming from sensors' outputs, setting a zero input and a sampling time of 5 ms.

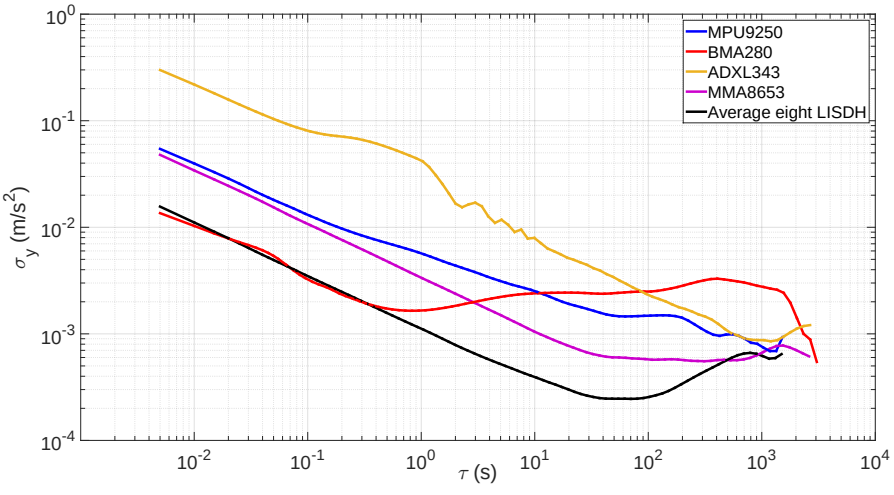


Figure 5.8 – Allan deviation graph of 4 different sensors, and the mean of 8 similar sensors.

To evaluate the performance of the proposed algorithm, an implementation using sensors MPU9250, BMA280, ADXL343, and MMA8653 is carried out. From Figure 5.8 it is observed that, the aver-

age of the 8 similar sensors gives the best performance in the system. Hence, for the assessment of the data fusion process, the average of the eight LIS2DH is used as reference to compute the errors in the estimation of the input obtained with the different methods (simple average, LSQ and the proposed algorithm).

Assuming that all sensors are calibrated and that their axes are aligned with the axes of the array frame, the standard model equation of accelerometers can be expressed as follows [31, 52]:

$$a_{i,k} = a_{\text{body},k} + \mathring{\omega}_k \times d_i + \omega_k \times (\omega_k \times d_i) + \varepsilon_{i,k} \tag{5.5}$$

where $a_{i,k}$ is the acceleration measured by the $i-$th triad at time step k, $a_{\text{body},k} = [a_{\vec{x},k}, a_{\vec{y},k}, a_{\vec{z},k}]^T$ is the acceleration of the body frame in axes $\vec{x}$ $\vec{y}$ and $\vec{z}$, ω_k and $\mathring{\omega}_k$ are the angular velocity and angular acceleration of the body frame, d_i is the position of $i-$th triad in the array coordinate frame, and $\varepsilon_{i,k}$ is the vector of noise present in the triad.

For this experiment, a rail that allows a linear displacement in only 2 directions (forward and backward) was designed (see Figure 5.10). By aligning the array coordinate frame with the rail, the sensor array system is exposed to accelerations only along one of the axes (hereinafter the $\vec{x}$ axis). This makes $a_{\vec{y},k}$, $a_{\vec{z},k}$, ω (and therefore, $\mathring{\omega}$) equal to zero. Thus, the measurement reported by $i-$th sensor at time step k is simplified as follows:

$$a_{i,k} = a_{\vec{x},k} + \varepsilon_{i,k} \tag{5.6}$$

Thus, variables used in the proposed algorithm are: $\mathbf{Y}_k = [a_{1,k}, \ldots, a_{4,k}]^T$, $n = 4$, $\mathbf{X} = a_{\vec{x},k}$ (and therefore $r = 1$), matrix $\mathbf{H} = \mathbf{1}_n$, and $m < 2$. The definition of $\mathbf{P}$ and the proof of the existence of $(\mathbf{P}^{\circ 2})^{-1}$ remain the same as the presented in section 5.1.

The condition used to find uncalibrated sensors described in (4.34) is set as follows:

$$\text{Uncalibrated sensor} = \begin{cases} \hat{\sigma}^2(\varepsilon_{i,k}) > 2\,\sigma^2(\varepsilon_{max}) & \text{True} \\ \\ \text{Otherwise} & \text{False} \end{cases} \tag{5.7}$$

where $\sigma^2(\varepsilon_{max})$ denotes the highest variance in S found during a pre-calibration phase, which resulted as $\sigma^2(\varepsilon_{max}) = \hat{\sigma}^2(\varepsilon_{\text{ADXL343}})$. In addition, for the function runVar(υ) a smoothing factor $\gamma = 0.999$ is empirically selected.

Finally, for the input, a signal with amplitude $< 2g$ is manually generated. The duration of the experiment is 1000 s (2×10^5 samples).

5.2.1 Assessment of calibrated devices

Figure 5.11 shows the estimation of sensors' variances for the first 400s of measurements using the proposed algorithm. The corresponding weights are presented in Figure 5.12, where its inverse relationship with the estimated variances can be appreciated.

Figure 5.13 presents a performance comparison for this 400s. This figure shows the window RMS error obtained with a simple average using the four sensors, a LSQ computed from inputs and outputs corresponding to the first 400s of simulation, and the estimation obtained with the proposed algorithm by means of a weighted average defined in (4.12). The size of the window is set up to 2×10^4 samples (equivalent to 100s of data) in order to present a figure where the transient behavior of the estimates can be appreciated. This figure shows that the error in the estimation of the input

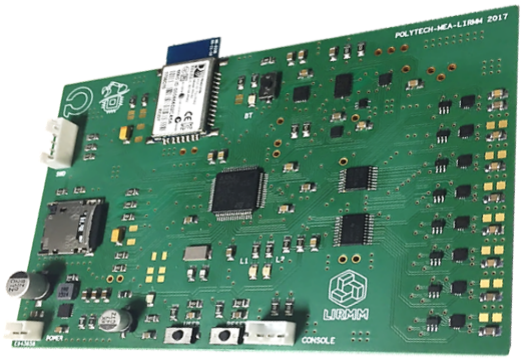


Figure 5.9 – An in-house designed embedded system with an inertial sensor array. The array consists of 12 inertial sensor chipsets: eight LIS2DH, one MPU9250, one MMA8653, one ADXL343 and one BMA280. Each chipset contains an accelerometer triad.

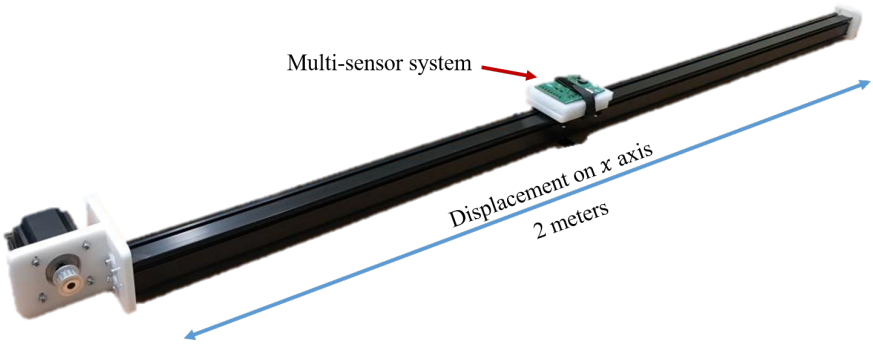


Figure 5.10 – Rail built for acceleration measurement using the card with 12 MEMS accelerometers presented in Figure 5.9. This rail allows the measurement of acceleration on a single axis (in both directions) without other magnitudes (apart from gravity) interfering with the experiment.

obtained with the proposed algorithm is slightly larger than the one obtained by the LSQ. However, unlike LSQ, the proposed algorithm does not use any knowledge about the input. Furthermore, for the whole 400s, the error obtained with the proposed algorithm presents a better performance than the one obtained with the simple average.

5.2.2 Presence of faults and recoveries

Now, in order to show how the proposed algorithm is capable to detect and identify uncalibrated sensors, two different scenarios where two of the sensors exhibit calibration errors are presented. In the first scenario, a scale factor error $\alpha_{\text{MPU9250}} = -0.25$ is induced on sensor MPU9250 from 500s to 600s, then a recovery of this sensor is assumed. For the second scenario, a critical damage on sensor BMA280 is generated by setting up a scale factor error $\alpha_{\text{BMA280}} = -1$, from 700s to 800s. Then, after 800s a recovery of this sensor is assumed.

Figure 5.14 presents the estimation of sensors' variances given by the proposed algorithm. Vertical dotted lines in gray represent the detection of the uncalibration in sensor MPU9250 at 570s, its recovery at 625s, the detection of a fault on sensor BMA280 at 745s, and its recovery at

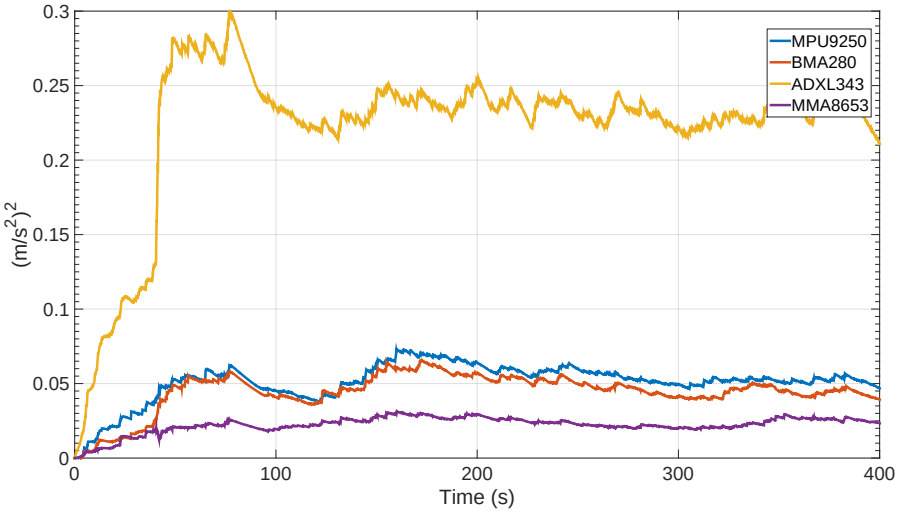


Figure 5.11 – Variance estimation of an array of 4 MEMS accelerometers.

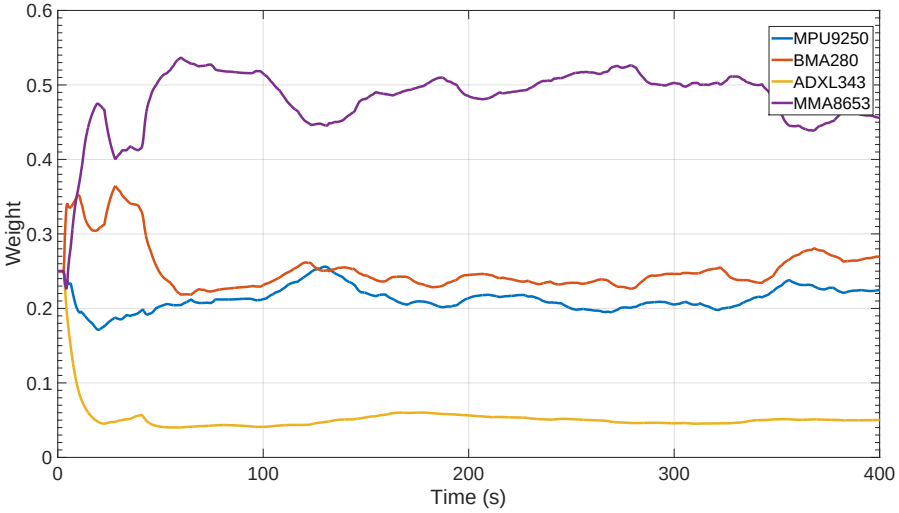


Figure 5.12 – Assignment of weights resulting from estimation of sensors' variances.

850s.

Faults and uncalibrations on sensors are detected by means of expression (5.7). The horizontal dotted line represent the value of ξ, which is set up to $\xi = 2 * \sigma^2(\epsilon_{max,t})$ using the value of the highest variance (variance of sensor ADXL343) estimated at $t = 100$s. The value of this threshold is $\xi = 0.4702$. After 100s, once the variance of any sensor exceeds this threshold, the algorithm assumes the presence of a new uncalibrated sensor. In the same way, if the value of any uncalibrated sensor falls below the threshold, then the algorithm assumes the recovery of this sensor. Vertical dotted lines exemplify this.

Once identified, faulty sensors are removed from the data fusion process by assigning them a zero weight, as shown in Figure 5.15. After its recovery, each faulty sensor is reincorporated to the data fusion process as soon as its value resulting from the evaluation of (5.7) becomes lower than

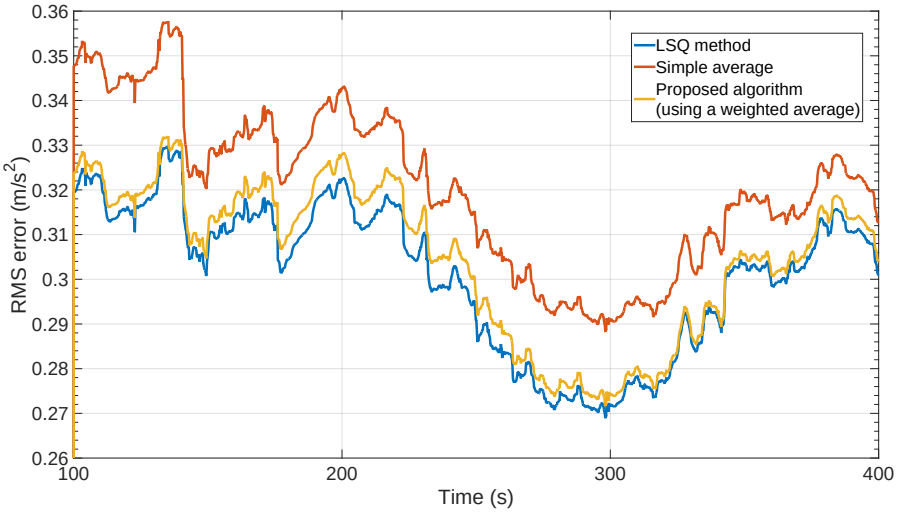


Figure 5.13 – A window RMS error for the assessment of the proposed algorithm (by means of a weighted average), and its comparison with a simple average, and the LSQ method.

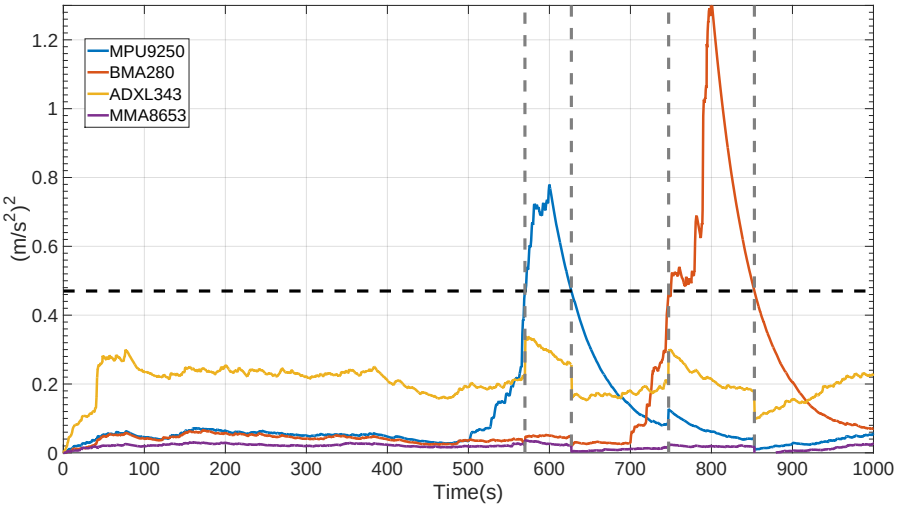


Figure 5.14 – Individual variance estimation of a system composed of 4 MEMS based accelerometers. Sensor MPU9250 is marked as uncalibrated at 570s and unmarked at 625s. Similarly, sensor BMA280 is marked at 745s and unmarked at 850s. Vertical dotted lines show the time at which such identifications are carried out. Horizontal dotted line indicates the threshold for the identification.

ξ. Figure 5.14 and Figure 5.15 illustrate this.

Finally, Figure 5.16 shows the window RMS error obtained with LSQ, a simple average and the proposed algorithm (by means of a weighted average) for the total time of the experiment. For the execution of LSQ, data obtained during the first 400s of the experiment are used. Note that, when there are no faults in the system ($t < 400$s), error obtained with the proposed algorithm is lower than the one obtained with a simple average, and close to the one obtained with the LSQ

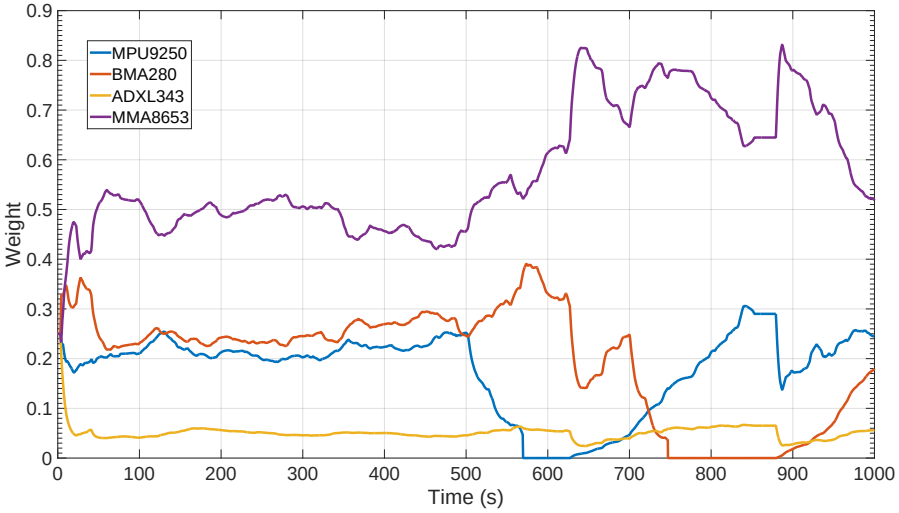


Figure 5.15 – Assignment of weights computed by the proposed algorithm for a 4 sensor system.

method. Once faults in sensors MPU9250 and BMA280 occur, the level of performance offered by the proposed algorithm becomes better than performances obtained with the other two methods. This is due to the removal of faulty sensors from the data fusion process.

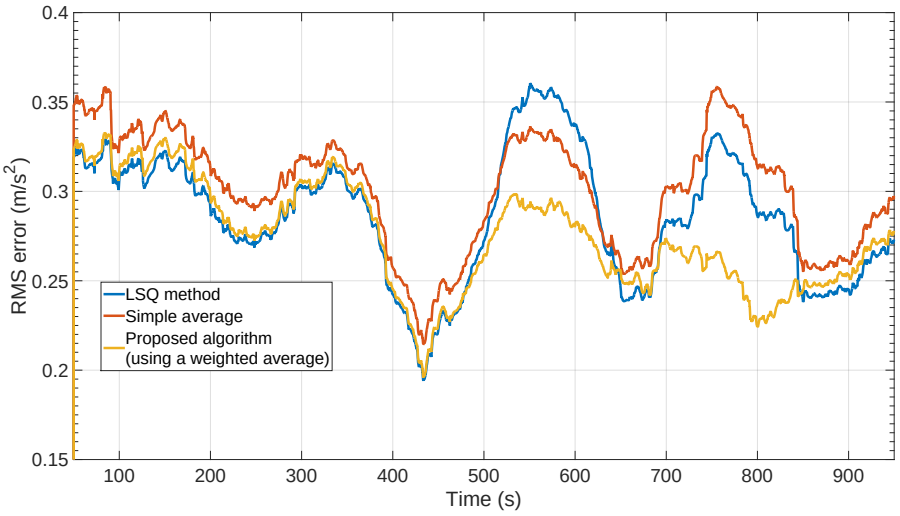


Figure 5.16 – RMS error obtained from the complete time of the experiment.

5.3 Conclusions

Assuming a set of calibrated sensors, the proposed algorithm is able to carry out: 1) estimation of sensors' variances, 2) detection of faults, and 3) detection and reincorporation of recovered / replaced sensors. In this chapter, we tested these three points through simulations into Matlab Simulink and a real implementation of a 12 MEMS-sensor system. In both cases, a single in-

put signal was measured. To evaluate the usefulness of this algorithm, it was combined together with the weighted average method to estimate the input. For comparison, the LSQ and the simple average of all the sensors were used.

For both, simulations and the physical implementation, it was proven that the level of performance obtained with the proposed algorithm was able to reach the same level of performance offered by the LSQ method, however, the proposed method has some advantages over LSQ, such as not requiring input knowledge and updating of weights at each time step. Moreover, it was proven through different scenarios that the proposed algorithm is adaptable to the presence of faults, which allows a certain degree of autonomy and extends the lifetime of the system.

Although some examples were presented, not all the advantages of this algorithm were evaluated. For example, consider an array consisting of sensors with different full scales. A saturation error can be seen as a scale factor error, where: saturated value = input * scale factor. Then, the proposed algorithm must be able to handle computation of variances as well as the treatment of failures in the array. The different full scales will result in different noise levels for which the proposed algorithm will assign the corresponding weights. Furthermore, if one or more sensors present saturation errors, these will be detected by the proposed algorithm.

In conclusion, assuming a single input, the proposed algorithm combined with the weighted average method is able to reach the same level of performance that the LSQ, without input knowledge, and with the advantage of being adaptive in the presence of different faults in the elements of the system. Those characteristics make this algorithm suitable for an implementation in embedded systems.

Chapter 6

Summary and conclusions

The seed of this research work was the general idea stating that n identical sensors measuring the same signal will improve the noise level by $\sqrt{\mathrm{n}}$. In the case of sensors with different level of noise it has been proven that a weighted average can improve the performance with respect to a simple average. The algorithm proposed in this work intends to adapt dynamically weights in order to improve adaptability, robustness, and dependability of a sensor array.

In the first chapter, a brief introduction about the measurement process, sensors, and multi-sensor systems was presented. Characteristics such as systematic errors and stochastic noises were defined. Finally, concepts of redundant measurements and data fusion were introduced at the end of this chapter.

During the literature study, it was observed that, in order to carry out this work, a simulation tool was required for analyzing the main phenomena present in sensor systems. Thus, a generic sensor model for simulations in Matlab Simulink was developed and presented in Chapter 2. This model can be used for emulating the behavior of a sensor at a system level. The parameters considered by this model are scale factor, bias, nonlinearity (and thermal and random variations of their nominal values), white noise, $1/f$ noise, and $1/f^2$ noise. It is possible to configure such model by extracting parameters from a sensor datasheet, a power spectral density graph, or an Allan deviation graph. By using this sensor model, it is possible to carry out simulations of a single sensor or a multi-sensor system, allowing to test data fusion algorithms for different applications. In section 2.4, examples of the configuration of the sensor model by using PSD and ADEV graphs are presented. In both cases, the sensor model showed its ability to reproduce the requested noise behavior. Finally, in section 2.5, an example of implementation of a multi-sensor system is presented, and it is used to compare two different data fusion algorithms for tilt estimation based on gyroscope and accelerometer measurements, showing the usefulness of this tool for assessing different data fusion algorithms for a given application.

Using such sensor model, an analysis of five different algorithms for data fusion in sensor array systems was presented in chapter 3. The selected algorithms for the study were: blind calibration, LSQ, MLP ANN, Kalman filter, and the random weighting method. Our conclusions were that LSQ and ANN may be used for uncalibrated sensors. For calibrated sensors, the level of performance obtained using the random weighting is almost the same as the one obtained with the LSQ, Kalman-Takens, and MLP ANN. However, the random weighting shows some advantages such as not input knowledge, dynamism, and low computational complexity. Through this study, properties, pros and cons of each algorithm were emphasized to generate a comprehensive comparison.

Based on this analysis, a new adaptive algorithm for sensor array systems was presented in

chapter 4. This algorithm is presented as a proposal to overcome some drawbacks observed in the analysis aforementioned. The proposal is an on-line method based on the MInimum Norm Quadratic Unbiased Estimation (MINQUE), besides being able to follow changes in sensors' variances caused principally by the low-frequency noise effects, can detect and point out sensors affected by different faults. Throughout this chapter, it was shown that MINQUE algorithm requires that the number of sensors is strictly greater than two times the number of measured signals. Consequently, the presented variant of MINQUE method can detect faults as long as the number of faulty sensors is smaller than the difference between the number of sensors and two times the number of measured signals.

Finally, in chapter 5, the proposed algorithm is assessed using simulations inside the Matlab Simulink environment as well as a real implementation in a system composed of 12 MEMS accelerometers. As mentioned before, the MINQUE method can detect faults as long as the number of faulty sensors is smaller than the difference between the number of sensors and two times the number of measured signals. This was exemplified in both, simulations and the physical implementation. Moreover, it was verified that, when the system does not present uncalibrated sensors, the algorithm is capable of achieving the level of performance obtained with LSQ without any knowledge of the physical input.

Future work

One important point left for future work is the assessment of the proposed algorithm (chapter 4) under the presence of errors different from those proposed in chapter 5. We believe that this algorithm is able to handle detection of failures caused by saturation and blurring errors. However, due to lack of time, this was not verified.

On the other hand, it is proposed at the end of chapter 4 to extend the proposed algorithm to multiple physical inputs. This would require to further assess the reliability of the algorithm.

Finally, in chapter 3 an implementation of a basic neural network was presented for the data fusion process in a sensor array system. However, such implementation was considered static because once the ANN is parameterized, it cannot be updated. In [89], architecture is presented that offers a possible solution for this limitation.

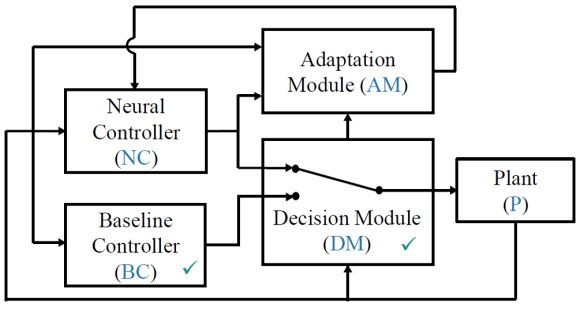


Figure 6.1 – The neural simplex architecture presented in [89].

This architecture is designed to update a neural network in real time. It uses a baseline controller (BC) that guarantees the correct operation of the control module but with a low performance, a deep neural network controller (NC) that does not guarantee security but that offers a high performance,

a decision module (DM) which is responsible for judging the outputs of the NC, and an adaptation module (AM) formed by a reinforcement learning algorithm which is responsible for retraining the neural network. The general picture of this architecture is shown in Figure 6.1. Broadly, this architecture works as follows: at the beginning, the NC is the algorithm in charge of controlling the system (denoted as Plant in Figure 6.1). The DM determines when NC's outputs are wrong. Once this happens, the DM switches the control to the BC and informs to the AM. The AM retrains "in shadow mode" the NC using available information in order to prevent the NC from making the same mistake. At each time step that the BC is in control, the AM retrains the NC. Also, at each time step, the DM evaluates NC's outputs. Once it is determined that the NC works correctly again, the DM returns the control to the NC. The idea of implementing this architecture for data fusion in a sensor array system seems interesting.

Appendix A

Appendix - Introduction

Colored noise using an IIR filter

```
function [x] = colored_noise_IIRFilter(n_pts, alpha, psd_white_noise, fs, num_coeff)
    %% Colored noise generated using a IIR filter with -num_coeff- coefficients
    % n_pts - number of samples of colored noise generated
    % alpha - exponent of colored noise - 1/f^{alpha}
    % psd_white_noise - power of white noise at the input of the IIR filter
    % fs - frequency of sampling at the output of the filter
    % num_coeff - number of coefficients used inside the IIR filter
    % x - Output colored noise

    % vector definition
    a = zeros(1,n_pts);
    % define psd of white noise
    variance_white_noise = (psd_white_noise * fs) / 2  ;
    % generate white noise, the input of the filter
    w = normrnd(0, sqrt(variance_white_noise), 1, n_pts);
    % Generate coefficients
    % a0
    a(1,1) = 1;
    % generate coefficients ak, 1 <= k <= num_coeff
    for k = 1 : 1 : num_coeff
        a(1,k+1) = ((k - 1 - (alpha/2)) * a(1,k))/k;
    end
    % discrete fourier transform
    a_dft = fft(a);
    w_dft = fft(w);
    % divide the two complex vectors
    for k = 1: 1 : n_pts/2 + 1
        factor_real = (real(w_dft(1,k)) * real(a_dft(1,k))) + (imag(w_dft(1,k)) * imag(a_dft
      (1,k)));
        factor_imag = (imag(w_dft(1,k)) * real(a_dft(1,k))) - (real(w_dft(1,k)) * imag(a_dft
      (1,k)));
        denominator = (real(a_dft(1,k)) * real(a_dft(1,k))) + (imag(a_dft(1,k)) * imag(a_dft
      (1,k)));
        w_dft(1,k) =  (factor_real/denominator) + 1i*(factor_imag/denominator);
    end
    % Copy the first half into the second half
    w_dft(1,(n_pts/2)+2:n_pts) = real(w_dft(1,n_pts/2:-1:2)) - 1i*imag(w_dft(1,n_pts/2:-1:2));
    % inverse Fourier transform
    x = ifft(w_dft);
end
```

Appendix B

Appendix - Generic sensor model for simulations at system level

Spectrum analysis

```
function [PSD,f] = power_spectrum_estimation(vector_of_time_series,frequency_of_sampling,
    type_of_plot,ideal_response,psd_colored_noise)
%
%
%input:  power_spectrum_estimation(vector_of_time_series,frequency_of_sampling,type_of_plot,
    ideal_response)
%output: desired plot of psd
%
%   type_of_plot:
%       * 'PSD' for Power Spectral Density
%       * 'PS' for Power Spectrum
%       * 'LSD' for Linear Spectral Density
%       * 'LS' for Linear Spectrum
%       * 'LOG' for Log vs Log PSD graph
%
%   ideal_response:
%       * 'PINK' for Pink Ideal PSD (1/f)
%       * 'BROWN' for Brown Ideal PSD (1/f^2)
%
%   Argument management:
%       vector_of_time_series and frequency_of_sampling are mandatory
%       type_of_plot and ideal_response are optional
%
%
if nargin < 3
   type_of_plot = 'PSD';
end
if nargin < 4
   ideal_response = 'N';
end
if nargin < 5
   psd_colored_noise = 1e0;
end


% -------- Parameters

%vector of time series
    x = vector_of_time_series;
%fs the sampling frequency, Hz  - Note: Nyquist frequency Nf = fs/2
    fs = frequency_of_sampling;
%FFT frame size
```

```
41      nfft = floor(length(x)/4);
42      %nfft = max(256,2^nextpow2(length(10)));
43  %frequency of resolution, Hz
44      fres = fs / nfft;
45  %vector holding the window weights w_j, nfft should have the same length as the length of the
        window vector nwin
46      nwin = hanning(nfft);
47  %an integer(a fraction of the FFT length) that indicates the desired overlap, for Hanning
        window 1/2*nfft
48  %if we don't specify a value,  the default number of overlapped samples is 50% of the window
        length.
49      noverlap = nfft/2;
50
51
52  % -------- Estimation
53
54  %PSD estimation by Welch's method
55  %The results are Pxx, a vector with the power spectral density (PSD)and f, a vector with the
        corresponding frequencies in Hz.
56  %If y is the unit of the input time series, the output Pxx has the unit y2/Hz.
57      [PSD,f] = pwelch(x,nwin,noverlap,nfft,fs,'oneside');
58  %For obtaining Power Spectrum - units^2
59      %We define the following two sums for normalization purposes
60      s1 = sum(hanning(nfft));
61      s2 = sum(hanning(nfft).^2);
62      %normalized equivalent noise bandwidth
63      nenbw = nfft * (s2 / s1^2);
64      %Effective noise bandwidth
65      enbw = nenbw * fres;
66      %Power Spectrum
67      PS = PSD .* enbw;
68  %For obtaining Linear spectral density - units / sqrt(Hz)
69      LSD = sqrt(PSD);
70  %For obtaining Linear Spectrum - units
71      LS = sqrt(PS);
72
73
74  % -------- Plotting
75
76  %Plotting ideal responses
77  %White Noise
78  PSDWhite = f(2:end)./f(2:end);
79  PSDWhite = PSDWhite.*psd_colored_noise;
80  %Pink Noise
81  PSDPink = 1./f(2:end);
82  PSDPink = PSDPink./psd_colored_noise;
83  %Brown Noise
84  PSDBrown = 1./(f(2:end)).^2;
85  PSDBrown = PSDBrown./psd_colored_noise;
86
87  if strcmp(type_of_plot,'PS')
88      %plot Power Spectrum (PS) - un^2
89      %figure();
90      semilogy(f(2:end),PS(2:end));
91      hold on;
92      xlabel('Hz'); ylabel('Units^2');
93      grid on;
94      head = sprintf('Power Spectrum \nUnits^2 vs Hz');
95      title(head);
96      legend('PS');
97      hold off;
98
99  elseif strcmp(type_of_plot,'LSD')
100     %plot Linear Spectral Density (LSD) - un/sqrt(Hz)
101     %figure();
102     semilogy(f(2:end),LSD(2:end));
103     hold on;
104     xlabel('Hz'); ylabel('Units/sqrt(Hz)');
105     grid on;
106     head = sprintf('Linear Spectral Density \nUnits/sqrt(Hz) vs Hz');
107     title(head);
108     legend('LSD');
109     hold off;
110
111 elseif strcmp(type_of_plot,'LOG-LSD')
112     %plot LOG Linear Spectral Density (LSD) - un/sqrt(Hz)
```

```
    %figure();
    loglog(f(2:end),LSD(2:end));
    hold on;
    xlabel('log_{10}(Hz)'); ylabel('log_{10}(Units/sqrt(Hz))');
    grid on;
    head = sprintf('Log vs Log Linear Spectral Density \nlog(Units/sqrt(Hz)) vs log(Hz)');
    title(head);
    legend('LSD');
    hold off;

elseif strcmp(type_of_plot,'LS')
    %plot Linear Spectrum (LS) - units
    %figure();
    semilogy(f(2:end),LS(2:end));
    hold on;
    xlabel('Hz'); ylabel('Units');
    grid on;
    head = sprintf('Linear Spectrum \nUnits vs Hz');
    title(head);
    legend('LS');
    hold off;

elseif strcmp(type_of_plot,'LOG')
    %plot Log vs Log PSD - log(un^2/Hz)
    %figure();
    %loglog(f(2:floor((length(f)/2))),PSD(2:floor((length(PSD)/2))));
    loglog(f(2:end),PSD(2:end));
    hold on;
    if strcmp(ideal_response,'PINK')
        loglog(f(2:end),PSDPink);
    elseif strcmp(ideal_response,'BROWN')
        loglog(f(2:end),PSDBrown);
    elseif strcmp(ideal_response,'WHITE')
        loglog(f(2:end),PSDWhite);
    end
    xlabel('log_{10}(Hz)'); ylabel('log_{10}(Units^2/Hz)');
    grid on;
    head = sprintf(' Log vs Log Power Spectral Density \nlog(Units^2/Hz) vs log(Hz)');
    title(head);
    legend('PSD');
    hold off;

elseif strcmp(type_of_plot,'PSD')
    %plot Power Spectral Density (PSD) - un^2/Hz
    %figure();
    semilogy(f(2:end),PSD(2:end));
    hold on;
    xlabel('Hz'); ylabel('Units^2/Hz');
    grid on;
    head = sprintf('Power Spectral Density \nUnits^2/Hz vs Hz');
    title(head);
    legend('PSD');
    hold off;
end

end
```

Appendix C

Appendix - Assessment of different data fusion algorithms for sensor array systems

Kalman - Takens Filter

```
%% "Ensemble Kalman Filtering without a Model"
% presented by Franz Hamilton, Tyrus Berry, and Timothy Sauer in 2016
%
%  total_of_measurements - total of measurements in the simulation
%  n - number of sensors in the system
%  sensors_measurements - measurements reported by sensors
%  xhat - estimation of the input signal
%  d - size of delay vector, (number of time steps)
%  M - Number of nearest neighbors
%  size_training_data - the training data set is from the first sample to
%                       the sample indicated for size_training_data
%  manifold - dictionary that contains all dalay vectors
%
%  Kalman Filter - Equations
%   xk = F(xk_minus_one) + wk
%   zk = H * xk + vk
%
%  Used variables:
%   xk - state vector
%   zk - measurement vector
%   wk - process noise
%   vk - measurement noise
%   F - transition function
%   H - state to measurement matrix
%   Q - process noise covariance matrix
%   R - measurement noise covariance matrix
%   P - Prediction on the error covariance

function [xhat] = Kalman_Takens_algorithm(sensors_measurements,d,M,size_training_data)

% inital variables
[total_of_measurements,n] = size(sensors_measurements);
training_data_set = sensors_measurements(1:size_training_data,:);
x_pred = zeros(total_of_measurements,1);
xhat = zeros(total_of_measurements,1);
Q = 1e-2;
% R - Matrix that contains power of white noise of each sensor
R = diag([5e-3,1e-3,5e-4,1e-4,5e-5]);
```

```
P_est_ant = 1e-3;
H = ones(n,1);

% avg - average of measurement vector at each time step
avg = sum(training_data_set')'./n;
% generate manifold (data dictionary)
% manifold(k,:,:) = [ id_manifold , avg(measurement_vector_timestep_k-d), ... , avg(
    measurement_vector_timestep_k-1) ]
for k=1:1:size_training_data-d
    manifold(k,:) = [k,avg(k:k+d-1,:)'];
end
% classify dictionary of data into clusters
[SS,C] = kmeans(manifold(:,2:end),5,'MaxIter',1000);
% create subdictionaries according to clusters
cluster1 = [];
cluster2 = [];
cluster3 = [];
cluster4 = [];
cluster5 = [];
for i=1:size(manifold,1)
    if SS(i,1) == 1
        cluster1 = [cluster1;manifold(i,:)];
    elseif SS(i,1) == 2
        cluster2 = [cluster2;manifold(i,:)];
    elseif SS(i,1) == 3
        cluster3 = [cluster3;manifold(i,:)];
    elseif SS(i,1) == 4
        cluster4 = [cluster4;manifold(i,:)];
    elseif SS(i,1) == 5
        cluster5 = [cluster5;manifold(i,:)];
    end
end
% during the learning phase, estimation is done using simple average
for k=1:1:size_training_data
    xhat(k,1) = sum(sensors_measurements(k,:))/n;
end

% after the learning phase
for k=size_training_data+1:1:total_of_measurements
    % Takens - reconstruction of state variable
    % delay vector of time step k - 1
    % x_k_minus_1 = [ avg(measurement_vector_timestep_k-d), ... , avg(
     measurement_vector_timestep_k-1) ]
    x_k_minus_1 = (sum(sensors_measurements(k-d:k-1,:)')'./n)';
    % Classify delay vector x_k_minus_1 into a cluster
    [D,cluster_x_k_minus_1] = pdist2(C,x_k_minus_1,'cityblock','Smallest',1);
    if cluster_x_k_minus_1 == 1
        cluster = cluster1;
    elseif cluster_x_k_minus_1 == 2
        cluster = cluster2;
    elseif cluster_x_k_minus_1 == 3
        cluster = cluster3;
    elseif cluster_x_k_minus_1 == 4
        cluster = cluster4;
    elseif cluster_x_k_minus_1 == 5
        cluster = cluster5;
    end
    % find the M nearest neighbors of x_k_minus_1
    % neighbors(j,:) = [ distance , id_delay vector , avg(measurement_vector_timestep_k+1) ];
    neighbors = zeros(M,3);
    num_neighbor = 0;
    max_distance = 0;
    for i=1:size(cluster,1)-1
        % distance between measurement and i-th manifold
        distance =  pdist2(x_k_minus_1,cluster(i,2:end),'cityblock');
        if num_neighbor == 0
            num_neighbor = num_neighbor + 1;
            neighbors(num_neighbor,:) = [distance,cluster(i,1),manifold(cluster(i,1)+1,d+1)];
            max_distance = distance;
        elseif num_neighbor > 0 && num_neighbor < M
            num_neighbor = num_neighbor + 1;
            neighbors(num_neighbor,:) = [distance,cluster(i,1),manifold(cluster(i,1)+1,d+1)];
            if distance > max_distance
                max_distance = distance;
            end
        elseif num_neighbor == M && distance < max_distance
```

```
            old_max = find(neighbors(:,1) == max(neighbors(:,1)));
            neighbors(old_max,:) = [distance,cluster(i,1),manifold(cluster(i,1)+1,d+1)];
            max_distance = neighbors(find(neighbors(:,1) == max(neighbors(:,1))),1);
        end
    end
    % normalization of weights
    total_of_distance = sum(1./neighbors(:,1));
    weight = zeros(M,1);
    % weighted average
    weight(:,1) = (1./(neighbors(:,1)))./total_of_distance;
    x_pred(k,1) = weight(:,1)'*neighbors(:,3);
    % Kalman Filter
    zk = sensors_measurements(k,:)';
    % 1.- Predict state and error covariance
    F = x_pred(k,1)/xhat(k-1,1);
    P_pred = F^2*P_est_ant + Q;
    % 2.- Compute Kalman gain
    Kk = ( P_pred * H' ) / ( H * P_pred * H' + R );
    % 3.- Compute the estimate
    xhat(k,1) = x_pred(k,1) + Kk * ( zk - H * x_pred(k,1) );
    % 4.- Compute the error covariance
    P_est_ant = P_pred - Kk * H * P_pred;
end

end
```

Blind calibration method

```
%% Blind calibration algorithm
% This algorithm was proposed by Laura Balzano and Robert Nowak in "Blind
% Calibration of Sensor Networks", in 2007.
%
%  total_of_measurements - total of measurements in the simulation
%  n - number of sensors in the array system
%  sensors_measurements - measurements reported by sensors
%  xhat - estimation of the input signal

function [estimation,inv_alpha_hat,betahat] = blind_calibration_algorithm(sensors_measurements
    )

% Initial variables
% vector of measurements
Yk = sensors_measurements';
[total_of_measurements,n] = size(sensors_measurements);
Ybar = zeros(n,total_of_measurements);
% gamma is the smoothing constant
gamma = 0.999999;
% build orthogonal projection matrix - P
P = ones(n,n);
for i=1:n
    for j=1:n
        if i ~= j
            P(i,j) = -1/(n-1);
        end
    end
end

% for each time step
for k=1:1:total_of_measurements
    % estimate Y_bar
    if k == 1
        Ybar(:,k) = Yk(:,k);
    else
        Ybar(:,k) = ((k-1)/k)*Ybar(:,k-1) + (1/k)*Yk(:,k);
    end
    P_times_Yk_minus_Ybar = P * (diag(Yk(:,k)) - diag(Ybar(:,k)));
    A = [ones(1,n).*(1/n);P_times_Yk_minus_Ybar];
    B = [1;zeros(n,1)]; % assume typical value for average of alpha_i
    % if the system has a solution then solve it
    if rank(A) == n
        if k == 1
            inv_alpha_hat(:,k) = lsq(A,B); % least square solution to Ax=B
        else
            % low-pass filter used for alpha^-1 ; it avoids spurious estiamtions
            inv_alpha_hat(:,k) = inv_alpha_hat(:,k-1).*gamma + lsq(A,B).*(1-gamma);
        end
    % if the system has not solution then assume typical values
    else
        inv_alpha_hat(:,k) = ones(n,1);
    end
    % to estimate beta
    A = [ones(1,n).*(1/n);P];
    B = [0;P*Ybar(:,k)];  % assume typical values
    % if the system has a solution then solve it
    if rank(A) == n
        betahat(:,k) = lsq(A,B);
    % if the system has not a solution then assume typical values
    else
        betahat(:,k) = zeros(n,1);
    end
    % signal estimation
    xhat(k) = sum((Yk(:,k)-betahat(:,k)).*inv_alpha_hat(:,k))/n;
end

estimation = xhat';

end
```

Random weighting algorithm

```
%% "Random Weighting Method for Multisensor Data Fusion"
% Proposed by Shesheng Gao, Yongmin Zhong, and Wei Li in 2011
%
%  total_of_measurements - total of measurements in the simulation
%  sensors_measurements - measurements reported by sensors
%  gamma_ii - self covariance of sensor i
%  gamma_ij - cross-covariance between sensors i and j
%  xhat - estimation of the input signal

function [xhat, wi, individual_variance, gamma_ii, gamma_ij] = random_weighting(
    sensors_measurements)
[total_of_measurements,n] = size(sensors_measurements);
gamma_ii = zeros(total_of_measurements,n);
gamma_ij = zeros(total_of_measurements,n);
individual_variance = zeros(total_of_measurements,n);
wi = zeros(total_of_measurements,n);
xhat = zeros(total_of_measurements,1);
% sensor used for cross-covariances
sensor_leader = 1;
%sensor_leader = 5;
% sensor used for cross-covariance of sensor leader
second_sensor_leader = 2;

% for each measurement
for k=1:1:total_of_measurements
    % Update of gamma_{i,i} , gamma_{i,j} and sigma_square_{i}
    for i=1:1:n
        % for the first iteration there is not average
        if k == 1
            gamma_ii(k,i) = sensors_measurements(k,i)^2;
            if (i == sensor_leader)
                gamma_ij(k,i) = sensors_measurements(k,i).*sensors_measurements(k,
    second_sensor_leader);
            else
                gamma_ij(k,i) = sensors_measurements(k,i).*sensors_measurements(k,
    sensor_leader);
            end
        else
            gamma_ii(k,i) = run_mean(sensors_measurements(k,i)^2,gamma_ii(k-1,i),k);
            if (i == sensor_leader)
                % on-line mean
                gamma_ij(k,i) = ((k-1)/k)*gamma_ij(k-1,i) + (1/k)*(sensors_measurements(k,i).*
    sensors_measurements(k,second_sensor_leader));
            else
                gamma_ij(k,i) = ((k-1)/k)*gamma_ij(k-1,i) + (1/k)*(sensors_measurements(k,i).*
    sensors_measurements(k,sensor_leader));
            end
        end
        individual_variance(k,i) = gamma_ii(k,i) - gamma_ij(k,i);
    end
    % Compute weights
    for i=1:1:n
        wi(k,i) = inv(sum(individual_variance(k,i)./individual_variance(k,:)));
    end
    % Estimation of the signal
    xhat(k,1) = sum(wi(k,:).*sensors_measurements(k,:));
end
```

Appendix D

Appendix - An adaptive algorithm based on MINQUE for data fusion and fault detection in sensor array systems

Window RMS error

```
%% Window RMS - Computes the Root Mean Square of data inside the window
%       data - vector of data to analize
%       size_window - size of the window used for the analysis

function output  = window_rms(data,size_window)
iterations = size(data,1);
data_squared = data.^2; % square of the data
output = zeros(iterations,1);
% move the window step by step
for k=size_window/2:1:iterations - size_window/2
    % mean of square data inside the window
    avg = sum(data_squared(k-size_window/2+1:1:k+size_window/2))/size_window;
    % square root of mean
    output(k,1) = sqrt(avg);
end
end
```

Estimation of sensors' variances

```
function [uncalibrated_sensor, xi, PY, varPY, est_E] = estimation_of_variances_of_sensors(
      sensors_measurements, H)
%% Algorithm for estimation of sensors' variances
% Required:
%             sensors_measurements - Measurements taken by sensors
%                                H - Observation matrix
%
% Output:
%                            est_E - Estimation of variances of sensors

%% VARIABLES
% total of measurements taken by the system
total_of_measurements = size(sensors_measurements,1);
% number of sensors in the system
n = size(sensors_measurements,2);
% Y - vector of measurements
Y = sensors_measurements';
% number of signals to measure
r = size(H,2);
% PY - multiplication of Y and orthogonal projection matrix P
PY = zeros(n,total_of_measurements);
% run mean and variance of PY
avgPY = zeros(n,total_of_measurements);
varPY = zeros(n,total_of_measurements);
% est_E - Estimate of sensors' variances
est_E = zeros(n,total_of_measurements);
% t - initial lapse of time in which there aren't faults, this lapse goes from 0 to t
t = 100 * 200; %200 - sampling frequency in Hz
% list of detected uncalibrated sensors
uncalibrated_sensor = zeros(n,total_of_measurements);
% lapse of time required to the stabilization of function variance
%recovery_time = 2e3;
recovery_time = 30 * 200; %200 - sampling frequency in Hz
%recovery_time = 1; %200 - sampling frequency in Hz
% smoothing constant used in the low-pass filter for the function var( )5
ALPHA = 0.999;
%ALPHA = 0.9;
%ALPHA = 0.7;
%ALPHA = 0.5;
%ALPHA = 0.3;
%ALPHA = 0.1;

%% ALGORITHM
%  For each time step
for i=1:1:total_of_measurements
    % for the first (xi * frequency of sampling) samples, assume that all sensors are
      calibrated
    if i <= t
        calibrated_sensors = [1:1:n]';
    % for the rest of samples, find the set of uncalibrated sensors
    else
        calibrated_sensors = find(uncalibrated_sensor(:,i-1) == 0);
    end
    % Computation of P - orthogonal projection of H
    P = eye(size(calibrated_sensors,1)) - H(calibrated_sensors,:)*inv(H(calibrated_sensors,:)
      '*H(calibrated_sensors,:))*H(calibrated_sensors,:)';
    % PY
    PY(calibrated_sensors,i) = P * Y(calibrated_sensors,i);
    % Run variance of PY
    if i == 1
        avgPY(:,i) = PY(:,i);
    else
        avgPY(:,i) = run_mean(PY(:,i),avgPY(:,i-1),ALPHA);
        varPY(:,i) = run_variance(PY(:,i),avgPY(:,i),avgPY(:,i-1),varPY(:,i-1),ALPHA);
    end
    % estimation of variances of calibrated sensors
    est_E(calibrated_sensors,i) = inv(P.^2) * varPY(calibrated_sensors,i);
    % detection of uncalibrated sensors and recoveries only after time = t
    if i > t
        % xi - threshold used for the detection of uncalibrated sensors
        if ~(exist ('xi')) && i > t
            %xi = 2*max(max(est_E(:,1:t)));
```

```
            xi = 1e4;
        end
        % detection of uncalibrated sensors
        if any(est_E(calibrated_sensors,i) > xi)...
            && all(sum(uncalibrated_sensor(:,i-recovery_time-1:i-1)')==0 | sum(
    uncalibrated_sensor(:,i-recovery_time-1:i-1)')==recovery_time+1)
            % find the position of the highest variance
            posicion_highest_variance = find(est_E(calibrated_sensors,i) == max(est_E(
    calibrated_sensors,i)));
            % find the tag-number of the uncalibrated sensor (tag-number: s_1, s_2, ...)
            faulty_sensor = calibrated_sensors(posicion_highest_variance);
            % mark sensor as uncalibrated
            if (sum(uncalibrated_sensor(faulty_sensor,i-recovery_time-1:i-1))==0)
                uncalibrated_sensor(faulty_sensor,i) = 1;
            end
        end
        % Recovery
        if sum(uncalibrated_sensor(:,i-1)) > 0
            recovered_sensors = [];
            for j=1:1:n
                if uncalibrated_sensor(j,i-1) == 1
                    % assume sensosr j as calibrated
                    calibrated_sensors = [find(uncalibrated_sensor(:,i-1) == 0 &
    uncalibrated_sensor(:,i) == 0) ; j];
                    % compute P
                    P = eye(size(calibrated_sensors,1)) - H(calibrated_sensors,:)*inv(H(
    calibrated_sensors,:)'*H(calibrated_sensors,:))*H(calibrated_sensors,:)';
                    % compute PY
                    temp_PY = [P * Y(calibrated_sensors,i),calibrated_sensors];
                    PY(j,i) = temp_PY(temp_PY(:,2)==j,1);
                    % mean and variance
                    avgPY(j,i) = run_mean(PY(j,i),avgPY(j,i-1),ALPHA);
                    varPY(j,i) = run_variance(PY(j,i),avgPY(j,i),avgPY(j,i-1),varPY(j,i-1),
    ALPHA);
                    % estimation of sensors' variances
                    temp_est_E = [inv(P.^2) * varPY(calibrated_sensors,i),calibrated_sensors];
                    est_E(j,i) = temp_est_E(temp_est_E(:,2)==j,1);
                    % verify if there is a recovery
                    if est_E(j,i) <= xi && sum(uncalibrated_sensor(j,i-recovery_time:i)) >=
    recovery_time
                        recovered_sensors = [recovered_sensors, j];
                    end
                    uncalibrated_sensor(j,i) = 1;
                end
            end
            uncalibrated_sensor(recovered_sensors,i) = 0;
        end
    end
end
end
```

Weighted average

```
function [w, weighted_average] = weighted_average_data_fusion(sensors_measurements,est_E,
    uncalibrated_sensor)
%%  Data fusion of sensor measurements usign weighted average
% Require:
%                   est_E - individual variance estimation
%     uncalibrated_sensor - status of a sensor (calibrated/uncalibrated)
%    sensors_measurements - total of measurements
% Output:
%                       w - estimated weights
%        weighted_average - estimate of input at time step i using weighted average

%% VARIABLES
% total_of_measurements - total of measurements taken by the system
total_of_measurements = size(sensors_measurements,1);
% n - number of sensors in the system
n = size(sensors_measurements,2);
% Y - Vector of measurements
Y = sensors_measurements';
% w - estimated weights
w = zeros(n,total_of_measurements);
% estimate of input at time step i using weighted average
weighted_average = zeros(total_of_measurements,1);
% to avoid picks in weights when a sensor is added, a low-pass filter is used for update of
% weights
%ALPHA = 0.999;
% time during which there are not faults
%time_free_of_faults = 8e4;

%%  For each time step
for i=1:1:total_of_measurements
    % Computation of individual weights
    % find calibrated sensors
    calibrated_sensors = find(uncalibrated_sensor(:,i) == 0);
    % simple average if any variance of calibrated sensors is <= 0 or > 1
    if any(est_E(calibrated_sensors,i) <= 0)
        if i == 1
            w(:,i) = (1/sum(~uncalibrated_sensor(:,i))) * ~uncalibrated_sensor(:,i);
        else
            %w(:,i) = w(:,i-1);
            w(:,i) = (1/sum(~uncalibrated_sensor(:,i))) * ~uncalibrated_sensor(:,i);
        end
    else
        for j=1:1:n
            if (uncalibrated_sensor(j,i)==0)
                for k=1:1:n
                    if (uncalibrated_sensor(k,i)==0)
                        w(j,i) = w(j,i) + est_E(j,i)/est_E(k,i);
                    end
                end
                if i == 1
                    w(j,i) = inv(w(j,i));
                else
                    w(j,i) = inv(w(j,i));
                    %w(j,i) = ALPHA*w(j,i-1) + (1-ALPHA)*inv(w(j,i));
                end
           end
        end
    end
%     % ignorar un sensor
%     if any(w(:,i) < 0.15)
%         sensores_a_ignorar = find(w(:,i) < 0.15);
%         a_dividir = sum(w(sensores_a_ignorar,i));
%         for j=1:1:n
%             if any(sensores_a_ignorar == j) || w(j,i) == 0
%                 w(j,i) = 0;
%             else
%                 w(j,i) = w(j,i)/(1-a_dividir);
%             end
%         end
%     end
    % Weighted average
    for j=1:1:n
```

```
        weighted_average(i) =  sum(w(:,i).*Y(:,i));
    end
end
end
```

Algorithm 2 Funtion runVar( )

Require:

v_k - Vector at time step k,

$\mu(v_{k-1})$ - mean at $k-1$,

$\sigma^2(v_{k-1})$ - variance at $k-1$

Ensure:

$\sigma^2(v_k)$ - Variance at time step k,

$\mu(v_k)$ - mean at k

define ζ ▷ Smoothing factor, with $0 \leq \zeta \leq 1$

if $k = 1$ **then**

$\mu(v_k) = v_k$

$\sigma^2(v_k) = 0$

else

$\mu(v_k) = \zeta\ \mu(v_{k-1}) + (1-\zeta)(v_k - \mu(v_{k-1}))$

$\sigma^2(v_k) = \zeta\ \sigma^2(v_{k-1}) + (1-\zeta)(v_k - \mu(v_{k-1}))(v_k - \mu(v_k))$

end if

return $\sigma^2(v_k)$, $\mu(v_k)$

Algorithm 3 Weighted average

Require:

$\mathbf{Y}_k = [y_{1,k}, \dots, y_{n,k}]^T$ - Measurements reported by all sensors at time step k

$\bar{S}_k$ - Set of uncalibrated sensors at time step k

$W_{k-1} = [w_{1,k-1}, \dots, w_{n,k-1}]^T$ - Weights of all sensors at time step $k-1$

Ensure:

$W_k = [w_{1,k}, \dots, w_{n,k}]^T$ - Weights for all sensors at time step k

$w_{i,k} = 0, \forall i | 1 \geq i \geq n$ ▷ To start, set to zero all weights

if Any estimated variance is ≤ 0 **then**

Set $W_k = W_{k-1}$ ▷ To avoid wrong weights

else

for Each calibrated sensor i **do**

for Each calibrated sensor j **do**

$w_{i,k} = w_{i,k} + \frac{1}{\sigma^2(\epsilon_{j,k})}$

end for

$w_{i,k} = (w_{i,k})^{-1}$

end for

end if

return W_k

Bibliography

[1] J. G. Webster, *The measurement, instrumentation and sensors handbook*. CRC Press., 3 ed., 1999.

[2] V. Singh, "Smart sensors: Physics, technology and applications," *Indian Journal of Pure and Applied Physics*, vol. 43, pp. 7–16, 02 2005.

[3] W. C. Crone, *A Brief Introduction to MEMS and NEMS*. Boston, MA: Springer US, 2008.

[4] S. E. Lyshevski, *Nano- and micro-electromechanical systems : fundamentals of nano- and microengineering*. Boca Raton, Fla. : CRC Press, 2nd ed., 2005. Includes bibliographical references and index.

[5] E. Mounier, "Status of the mems industry 2018." YOLE developpement.

[6] D. K. Shaeffer, "Mems inertial sensors: A tutorial overview," *IEEE Communications Magazine*, vol. 51, pp. 100–109, April 2013.

[7] V. Garg, I. Santamaria, D. Ramírez, and L. L. Scharf, "Subspace averaging and order determination for source enumeration," *IEEE Transactions on Signal Processing*, vol. 67, pp. 3028–3041, June 2019.

[8] N. Verma and D. Singh, "Data redundancy implications in wireless sensor networks," *Procedia Computer Science*, vol. 132, pp. 1210 – 1217, 2018. International Conference on Computational Intelligence and Data Science.

[9] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE Signal Processing Magazine*, vol. 13, pp. 67–94, July 1996.

[10] E. D. D. Claudio, R. Parisi, and G. Jacovitti, "Space time music: Consistent signal subspace estimation for wideband sensor arrays," *IEEE Transactions on Signal Processing*, vol. 66, pp. 2685–2699, May 2018.

[11] I. Skog, "Inertial and magnetic-field sensor arrays - capabilities and challenges," in *2018 IEEE SENSORS*, pp. 1–4, Oct 2018.

[12] K. Han and A. Nehorai, "Nested vector-sensor array processing via tensor modeling," *IEEE Transactions on Signal Processing*, vol. 62, pp. 2542–2553, May 2014.

[13] A. Jeremic and A. Nehorai, "Landmine detection and localization using chemical sensor array processing," *IEEE Transactions on Signal Processing*, vol. 48, pp. 1295–1305, May 2000.

[14] *International Vocabulary of Metrology – Basic and General Concepts and Associated Terms*. 2012.

[15] K. Kalantar-zadeh, *Sensors: An introductory course*. Springer, Boston, MA, 11 2013.

[16] H. Unbehauen, *CONTROL SYSTEMS, ROBOTICS AND AUTOMATION - Volume XXI: Elements of Automation*. eolss Publishers Company, Limited, 2009.

[17] J. Fraden, *Handbook of Modern Sensors: Physics, Designs, and Applications*. Springer International Publishing, 5th ed., 2016.

[18] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*. Springer Publishing Company, Incorporated, 1st ed., 2007.

[19] G. Secer and B. Barshan, "Improvements in deterministic error modeling and calibration of inertial sensors and magnetometers," *Sensors and Actuators A: Physical*, vol. 247, pp. 522 – 538, 2016.

[20] P. Gründler, *Chemical Sensors: An Introduction for Scientists and Engineers*. Springer, Berlin, Heidelberg, 01 2007.

[21] N. El-Sheimy, H. Hou, and X. Niu, "Analysis and modeling of inertial sensors using allan variance," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, pp. 140–149, Jan 2008.

[22] L. Callegaro, "Unified derivation of johnson and shot noise expressions," *AMERICAN JOURNAL OF PHYSICS*, vol. 74, pp. 4218 – 4227, March 2006.

[23] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global Positioning Systems, Inertial Navigation, and Integration*, vol. 4 of *10*. Country: WILEY, 3 ed., April 2007.

[24] N. J. Kasdin, "Discrete simulation of colored noise and stochastic processes and 1/f alpha; power law noise generation," *Proceedings of the IEEE*, vol. 83, pp. 802–827, May 1995.

[25] G. Heinzel, A. Rudiger, and R. Schilling, "Spectrum and spectral density estimation by the discrete fourier transform (dft), including a comprehensive list of window functions and some new at-top windows," 2002.

[26] J. Li and J. Fang, "Not fully overlapping allan variance and total variance for inertial sensor stochastic error analysis," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, pp. 2659–2672, Oct 2013.

[27] "Mathworks® documentation colored noise," 2015. Available at `http://www.mathworks.com/help/dsp/ref/colorednoise.html`.

[28] N. Xiong and P. Svensson, "Multi-sensor management for information fusion: issues and approaches," *Information Fusion*, vol. 3, no. 2, pp. 163 – 186, 2002.

[29] H. F. Durrant-Whyte, "Sensor models and multisensor integration," *The International Journal of Robotics Research*, vol. 7, no. 6, pp. 97–113, 1988.

[30] E. AZIMIRAD, J. Haddadnia, and A. Izadipour, "A comprehensive review of the multi-sensor data fusion architectures," *Journal of Theoretical and Applied Information Technology*, vol. 71, 01 2015.

[31] I. Skog, J. Nilsson, P. Händel, and A. Nehorai, "Inertial sensor arrays, maximum likelihood, and cramér–rao bound," *IEEE Transactions on Signal Processing*, vol. 64, pp. 4218–4227, Aug 2016.

[32] G. Chatterjee, L. Latorre, F. Mailly, P. Nouet, N. Hachelef, and C. Oudea, "Smart-mems based inertial measurement units: gyro-free approach to improve the grade," *Microsystem Technologies*, vol. 23, pp. 3969–3978, Sep 2017.

[33] D. Bellot, A. Boyer, and F. Charpillet, "A new definition of qualified gain in a data fusion process: application to telemedicine," in *Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002. (IEEE Cat.No.02EX5997)*, vol. 2, pp. 865–872 vol.2, July 2002.

[34] R. Boudjemaa and A. Forbes, *Parameter estimation methods in data fusion*. Mathematics and Scientific Computing, 02 2004.

[35] Y. Zheng, "Methodologies for cross-domain data fusion: An overview," *IEEE Transactions on Big Data*, vol. 1, pp. 16–34, March 2015.

[36] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[37] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*. Springer Publishing Company, Incorporated, 1st ed., 2007.

[38] P. Nouet, J. M. R. Velazquez, and F. Mailly, "A generic model for sensor simulation at system level," in *DTIP: Design, Test, Integration and Packaging*, (Roma, Italy), IEEE, May 2018.

[39] J. M. Rivera Velázquez, F. Mailly, and P. Nouet, "System-level simulations of multi-sensor systems and data fusion algorithms," *Microsystem Technologies*, Oct 2018.

[40] P. Petkov and T. Slavov, "Stochastic modeling of mems inertial sensors," *Cybernetics and Information Technologies*, vol. 10, pp. 31–40, 01 2010.

[41] P. Cheng, M. Nazar, and B. Oelmann, "Torque sensor based on differential air pressure using volumetric strain," *IEEE Sensors Journal*, vol. 17, pp. 3269–3277, June 2017.

[42] D. Wu, C. Gao, Y. Zhai, Y. Shen, and Z. Ji, "Fault diagnosis of pitch sensor bias for wind turbine based on the multi-innovation kalman filter," in *2016 35th Chinese Control Conference (CCC)*, vol. 3, pp. 6403–6407, July 2016.

[43] P. Asolkar, S. Gajre, Y. Joshi, and A. Das, "Simulation of colored and non-gaussian wind noise for tropical shallow waters," in *OCEANS 2016 MTS/IEEE Monterey*, vol. 3, pp. 1–5, Sept 2016.

[44] J. Yan, C. Li, G. Xu, and Y. Xu, "A novel on-line self-learning state-of-charge estimation of battery management system for hybrid electric vehicle," in *2009 IEEE Intelligent Vehicles Symposium*, pp. 1161–1166, June 2009.

[45] Y. Wang, A. Yang, X. Chen, P. Wang, Y. Wang, and H. Yang, "A deep learning approach for blind drift calibration of sensor networks," *IEEE Sensors Journal*, vol. 17, pp. 4158–4171, July 2017.

[46] I. Skog, J. Nilsson, P. Handel, and A. Nehorai, "Inertial sensor arrays, maximum likelihood, and cramer rao bound," *IEEE Transactions on Signal Processing*, vol. 64, pp. 4218–4227, Aug 2016.

[47] A. V. Constantin and G. I. Gheorghe, "Study of piezoresistive and capacitive tactile sensors modeling and simulation for the best linearity with applications in modern microelectronics and walking analysis," in *2017 International Semiconductor Conference (CAS)*, pp. 241–244, Oct 2017.

[48] C. Simon, T. Ludwig, and M. Kruse, "Extracting sensor models from a scene based simulation," in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 259–264, Sept 2016.

[49] S. Tripathi, P. Wagh, and A. B. Chaudhary, "Modelling, simulation sensitivity analysis of various types of sensor errors and its impact on tactical flight vehicle navigation," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 938–942, March 2016.

[50] J. Deng, K. Ghosh, and H. . P. Wong, "Modeling carbon nanotube sensors," *IEEE Sensors Journal*, vol. 7, pp. 1356–1357, Sep. 2007.

[51] G. Chatterjee, L. Latorre, F. Mailly, P. Nouet, N. Hachelef, and C. Oudea, "Smart-mems based inertial measurement units: gyro-free approach to improve the grade," *Microsystem Technologies*, vol. 23, 12 2015.

[52] G. Chatterjee, *Développement d'une unité de mesure inertielle à base de Smart-MEMS*. PhD thesis, 2016. Thèse de doctorat dirigée par Nouet, Pascal Systèmes automatiques et microélectroniques Montpellier 2016.

[53] S. Kouda, Z. Dibi, and F. Meddour, "Modeling of a smart humidity sensor," in *2008 International Conference on Microelectronics*, pp. 135–138, Dec 2008.

[54] C. Ferreira, "Designing neural networks using gene expression programming," in *Applied Soft Computing Technologies: The Challenge of Complexity* (A. Abraham, B. de Baets, M. Köppen, and B. Nickolay, eds.), (Berlin, Heidelberg), pp. 517–535, Springer Berlin Heidelberg, 2006.

[55] "Mathworks® documentation band-limited white noise." Available at `https://mathworks.com/help/simulink/slref/bandlimitedwhitenoise.html`.

[56] J. Rivera, "Mathworks®, sensor model for simulations at system level," 2020. Available at `https://fr.mathworks.com/matlabcentral/fileexchange/77819-sensor-model-for-simulations-at-system-level`.

[57] "Freescale semiconductor. data sheet: Fxln83xxq," 2014. Available at `http://www.nxp.com/docs/en/data-sheet/FXLN83xxQ.pdf`.

[58] M. Miroslav and S. Mikulas, "Computation and evaluation allan variance results," in *2016 New Trends in Signal Processing (NTSP)*, pp. 1–9, Oct 2016.

[59] "Avar, file exchange, mathworks®," 2018. Available at `https://fr.mathworks.com/matlabcentral/fileexchange/55765-avar`.

[60] "Phidgetspatial precision 3/3/3 high resolution: Product specifications." Available at `https://www.phidgets.com/?tier=3&catid=10&pcid=8&prodid=1038`.

[61] A. Alessandro, G. Vitale, S. Scudero, R. Anna, A. Constanza, A. Fagiolini, and L. Greco, "Characterization of mems accelerometer self-noise by means of psd and allan variance analysis," in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pp. 159–164, June 2017.

[62] P. Gui, L. Tang, and S. Mukhopadhyay, "Mems based imu for tilting measurement: Comparison of complementary and kalman filter based data fusion," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 2004–2009, June 2015.

[63] M. Clifford and L. Gomez, "Measuring tilt with low-g accelerometers," 2005. Available at `http://www.nxp.com/docs/en/application-note/AN3107.pdf`.

[64] S. Gao, Y. Zhong, and W. Li, "Random weighting method for multisensor data fusion," *IEEE Sensors Journal*, vol. 11, pp. 1955–1961, Sept 2011.

[65] G. Strang, *Introduction to Linear Algebra*. Wellesley, MA: Wellesley-Cambridge Press, fifth ed., 2016.

[66] L. Balzano and R. Nowak, "Blind calibration of sensor networks," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, (New York, NY, USA), pp. 79–88, ACM, 2007.

[67] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[68] S. J. Miller, "The method of least squares," 2006. Available at `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=7F676FCC3603F751C1CC98A48A533E63?doi=10.1.1.710.4069&rep=rep1&type=pdf`.

[69] A. Shukla, R. Tiwari, and R. Kala, *Artificial Neural Networks*, pp. 31–58. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[70] A. Konar, *Artificial Intelligence and Soft Computing*. CRC Press.

[71] H. Yu and B. Wilamowski, *Industrial Electronics Handbook, Capter 12: Levenberg-Marquardt Training*, vol. 5, pp. 1–15. 2 ed., 01 2011.

[72] G. Welch and G. Bishop, "An introduction to the kalman filter," 1995.

[73] C. K. Chui and G. Chen, *Kalman Filtering with Real-time Applications*. Berlin, Heidelberg: Springer-Verlag, 1987.

[74] Yuzhi Wang, Anqi Yang, Zhan Li, Pengjun Wang, and Huazhong Yang, "Blind drift calibration of sensor networks using signal space projection and kalman filter," in *2015 IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1–6, April 2015.

[75] F. Hamilton, T. Berry, and T. Sauer, "Ensemble kalman filtering without a model," *Physical Review X*, vol. 6, 03 2016.

[76] L. NOAKES, "The takens embedding theorem," *International Journal of Bifurcation and Chaos*, vol. 01, no. 04, pp. 867–872, 1991.

[77] J. M. Rivera Velázquez, F. Mailly, and P. Nouet, "Dynamic weighted average in multisensory systems," in *2019 Symposium on Design, Test, Integration Packaging of MEMS and MOEMS (DTIP)*, pp. 1–4, May 2019.

[78] J. Wahlstrom, I. Skog, and P. Handel, "Inertial sensor array processing with motion models," in *2018 21st International Conference on Information Fusion (FUSION)*, pp. 788–793, July 2018.

[79] C. R. Rao, "Estimation of heteroscedastic variances in linear models," *Journal of the American Statistical Association*, vol. 65, no. 329, pp. 161–172, 1970.

[80] A. Waegli, J. Skaloud, S. Guerrier, M. E. Parés, and I. Colomina, "Noise reduction and estimation in multiple micro-electro-mechanical inertial systems," *Measurement Science and Technology*, vol. 21, p. 065201, apr 2010.

[81] S. Akhlaghi, N. Zhou, and Z. Huang, "Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation," in *2017 IEEE Power Energy Society General Meeting*, pp. 1–5, July 2017.

[82] B. Tyrus and S. Timothy, "Adaptive ensemble kalman filtering of non-linear systems," *Tellus A: Dynamic Meteorology and Oceanography*, vol. 65, no. 1, 2013.

[83] Z. C. Yu, "A universal formula of maximum likelihood estimation of variance-covariance components," *Journal of Geodesy*, vol. 70, pp. 233–240, Jan 1996.

[84] C. Rao and M. Rao, *Matrix Algebra and Its Applications to Statistics and Econometrics*. 01 1998.

[85] R. A. Horn and Z. Yang, "Rank of a hadamard product," *Linear Algebra and its Applications*, vol. 591, pp. 87 – 98, 2020.

[86] A. Yang, P. Wang, and H. Yang, "Blind drift calibration of sensor networks using multi-output gaussian process," in *2018 IEEE SENSORS*, pp. 1–4, Oct 2018.

[87] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[88] N. El-Sheimy, H. Hou, and X. Niu, "Analysis and modeling of inertial sensors using allan variance," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, pp. 140–149, Jan 2008.

[89] D. Phan, N. Paoletti, R. Grosu, N. Jansen, S. A. Smolka, and S. D. Stoller, "Neural simplex architecture," 2019.