Self organizing distributed state estimators

Self organizing distributed state-estimators

Joris Sijs and Zoltan Papp

March 10, 2012

Distributed solutions for signal processing techniques are important for establishing large-scale monitoring and control applications. They enable the deployment of scalable sensor networks for particular application areas. Typically, such networks consists of a large number of vulnerable components connected via unreliable communication links and are sometimes deployed in harsh environment. Therefore, dependability of sensor network is a challenging problem. An efficient and cost effective answer to this challenge is provided by employing runtime reconfiguration techniques that assure the integrity of the desired signal processing functionalities. Runtime reconfigurability has thorough impact both on system design, implementation, testing/validation and deployment. The presented research focuses on the widespreaded signal processing method known as state estimation with Kalman filtering in particular. To that extent, a number of distributed state estimation solutions that are suitable for networked systems in general are overviewed, after which robustness of the system is improved according to various runtime reconfiguration techniques.

1 Introduction

Many people in our society manage their daily activities based on knowledge and information about, for example, weather conditions, traffic jams, pollution levels, oil reservoirs and energy consumptions. Sensor measurements are the main source of information when monitoring these surrounding processes. Moreover, a trend is to increase the amount of sensors, as they have become smaller, cheaper and easier to use, so that large-area processes can be monitored with a higher accuracies. To that end, sensors are embedded in a communication network creating a so called *sensor network*, which typically consists of *sensor nodes* linked via a particular network topology (Figure 1). Each sensor node combines multiple sensors, a central processing unit (CPU) and a (wireless) communication radio on a circuit board. Sensor networks have three attractive properties for system design: they require low maintenance, create "on-the-fly" (ad-hoc) communication networks and can maintain large amounts of sensors.

Nowadays, sensor nodes are commercial off-the-shelve products and give system designers new opportunities for acquiring measurements. Although they make sensor measurements available in large quantities, solutions for processing

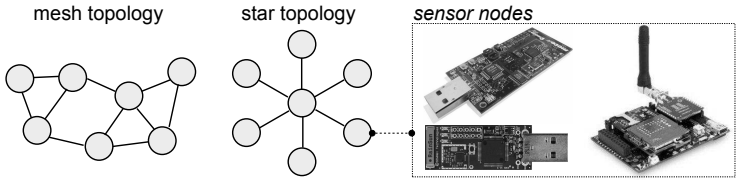


Figure 1: Sensor nodes in a mesh and star network topology with some examples of nodes: Tmote-Sky (top-left), G-node (bottom-left) and Waspmote (right).

these measurements automatically are hampered by limitations in the available resources, such as energy, communication and computation.

Energy plays an important role in remotely located processes. Such processes are typically observed by severely energy-limited sensor nodes (e.g. powered by battery or energy scavenging) that are not easily accessible and thus should have a long lifetime. Some applications even deploy sensor nodes in the asphalt of a road to monitor traffic, or in the forest to collect information on habitats. See, for example, the applications described in [44, 57] and recent surveys on sensor networks in [27, 1, 9, 37]. To limit energy consumption, one often aims to minimize the usage of communication and computational resources in sensor nodes. However, there are other reasons why these latter two resources should be used wisely.

Limited communication mainly results from upper bounds on the network capacity, as it was established in the Shannon-Hartley theorem for communication channels presented in [50]. It shows that the environment in which nodes communicate influences the amount of data that can be exchanged without errors. In addition, communication is affected by package loss as well, which occurs due to message collision (i.e. simultaneous use of the same communication channel by multiple transmitters). Hence, a suitable strategy for exchanging data is of importance to cope with the dynamic availability of communication resources.

Computational demand is related to the algorithms performed in sensor networks for processing the measurements. The established centralized solutions, where measurements are processed by a single node, fail for large-scale networks even when communication is not an issue: with an increasing amount of sensor nodes the computational load of a centralized solution will grow polynomially, up to a point that it is no longer feasible or highly inefficient. To that extent, non-centralized solutions are explored that aim to make use of local CPUs that are already present in each node.

A straightforward consequence of the resource limitation, the scale and the often hostile embedding environment is that fault-tolerance and/or graceful

degradation are critical requirements for large-scale distributed systems. This means that the sensor network should be able to cope with situations that emerge from common operational events, such as node failures, sensor degradation and power loss. Building in redundancy to cover the anticipated failure modes may result in complex, prohibitively expensive implementations. Instead, dynamical system architectures are to be realized via runtime reconfiguration, as it realizes a networked system that can follow the changes in the internal and external operational conditions and assure optimal use of available resources.

Limitations of the above resources are important design parameters. Depending on the sensor network application at hand, suitable trade-offs must be made to enable a feasible and practical deployment. One of these trade-offs is the local processing - communication trade-off. This encourages the local processing of the sensor measurements rather then communicating them, since exchanging one bit typically consumes much more energy than processing one bit. Hence, *centralized* methods for processing measurements are unpractical, due to their significant impact on the communication requirements. To solve this issue, *distributed* signal processing methods are increasingly studied. Such methods seek for a more efficient use of the spatially distributed computation and sensing resources according to the network topology. The signal processing method addressed in this chapter is state estimation.

Well studied state estimation methods are the Kalman filter (KF) for linear processes, with extensions known as the extended KF and unscented KF for nonlinear processes, see, e.g., [29, 48, 26]. Apart from their centralized solutions, some distributed implementation are found in [14, 13, 46, 42, 33, 53, 54, 47, 16]. Typically, these distributed solutions perform a state estimation algorithm locally in each node and thereby, compute a local estimate of the global state vector. Note that these distributed solutions can thus be regarded as a network of state-estimators. However, they were not designed to cope with the unforeseen operational events that will be present in the system, nor address deliberate reconfigurations of a sensor network during operation[1].

Therefore, the contribution of this chapter is to integrate solutions on distributed Kalman filtering with a framework of self organization. To that extent, each node not only employs a state estimator locally but additionally performs a management procedure that supports the network of state-estimators to establish self organization. The outline of this chapter is as follows. First we address the used notation, followed by a problem description in Section 3. Section 4 then presents several existing solutions on distributed Kalman filtering, with its required resources in Section 5, for which a supportive management procedure is designed in Section 6. The proposed network of self organizing state-estimators is further analyzed in Section 7 in an illustrative example, while concluding remarks are summarized in Section 8.

[1] For example, a reduction of the sampling time of nodes that run out of battery power, so to save energy and increase their lifetime.

2 Notation and preliminaries

$\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$ and $\mathbb{Z}_+$ define the set of real numbers, non-negative real numbers, integer numbers and non-negative integer numbers, respectively. For any $\mathcal{C} \subset \mathbb{R}$, let $\mathbb{Z}_{\mathcal{C}} := \mathbb{Z} \cap \mathcal{C}$. The notation 0 is used to denote either zero, the null-vector or the null-matrix of appropriate dimensions. The transpose, inverse (when it exists) and determinant of a matrix $A \in \mathbb{R}^{n\times n}$ are denoted as $A^\top$, A^{-1} and $|A|$, respectively. Further, $\{A\}_{qr} \in \mathbb{R}$ denotes the element in the q-th row and r-th column of A. Given that $A, B \in \mathbb{R}^{n\times n}$ are positive definite, denoted with $A \succ 0$ and $B \succ 0$, then $A \succ B$ denotes $A - B \succ 0$. $A \succeq 0$ denotes that A is positive semi-definite. For any $A \succ 0$, $A^{\frac{1}{2}}$ denotes its Cholesky decomposition and $A^{-\frac{1}{2}}$ denotes $(A^{\frac{1}{2}})^{-1}$. The Gaussian function (Gaussian in short) of vectors $x, \mu \in \mathbb{R}^n$ and matrix $\Sigma \in \mathbb{R}^{n\times n}$ is denoted as $G(x, \mu, \Sigma)$, for which $\Sigma \succ 0$ holds. Any Gaussian function $G(x, \mu, \Sigma)$ can be illustrated by its corresponding ellipsoidal sub-level-set $\mathcal{E}_{\mu,\Sigma} := \{x \in \mathbb{R}^n | (\mu - x)^\top \Sigma^{-1}(\mu - x) \leq 1\}$. See, Figure 2 for a graphical explanation of a sub-level-set.

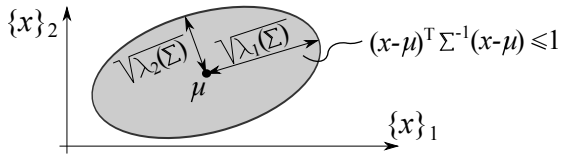


Figure 2: An illustrative interpretation of the sub-level-set $\mathcal{E}_{\mu,\Sigma}$.

3 Problem formulation

Let us consider a linear process that is observed by a sensor network with the following description.

Networked system The network consists of N sensor nodes, in which a node $i \in \mathcal{N}$ is identified by a unique number within $\mathcal{N} := \mathbb{Z}_{[1,N]}$. The set $\mathcal{N}_i \subseteq \mathcal{N}$ is defined as the collection of all nodes $j \in \mathcal{N}$ that have a direct network connection with node i, i.e., node i exchanges data with node j.

Process Each node $i \in \mathcal{N}$ observes a perturbed, dynamical process according to its local sampling time $\tau_i \in \mathbb{R}_{>0}$. Therefore, the discrete-time process model of node i, at the k_i-th sampling instant, yields

$$\begin{aligned} x[\mathsf{k}_i] &= A_{\tau_i} x[\mathsf{k}_i{-}1] + w[\mathsf{k}_i{-}1], \\ y_i[\mathsf{k}_i] &= C_i x[\mathsf{k}_i] + v_i[\mathsf{k}_i]. \end{aligned} \tag{1}$$

The state vector and local measurement are denoted as $x \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^{m_i}$, respectively, while process-noise $w \in \mathbb{R}^n$ and measurement-noise $v_i \in \mathbb{R}^{m_i}$ follow the Gaussian distributions $p(w[\mathsf{k}_i]) := G(w[\mathsf{k}_i], 0, Q_{\tau_i})$ and $p(v_i[\mathsf{k}_i]) := G(v_i[\mathsf{k}_i], 0, V_i)$, for some $Q_{\tau_i} \in \mathbb{R}^{n\times n}$ and $V_i \in \mathbb{R}^{m_i\times m_i}$. A

method to compute the model parameters A_{τ_i} and Q_{τ_i} from the corresponding continuous-time process model $\dot{x} = Fx + w$ is the following:

$$A(\tau_i) := e^{F\tau_i} \quad \text{and} \quad Q_{\tau_i} := B_{\tau_i} cov\big(w(t-\tau_i)\big) B_{\tau_i}^{\top},$$
$$\text{with} \qquad B(\tau_i) := \int_0^{\tau_i} e^{F\eta} d\eta.$$

The goal of the sensor network is to compute a local estimate of the global state x in each node i. Note that the process model is linear and both noises are Gaussian distributed. As such, it is appropriate to assume that the local estimate is Gaussian distributed as well, i.e., $p_i(x[\mathsf{k}_i]) := G(x[\mathsf{k}_i], \hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i])$ for some *mean* $\hat{x}_i[\mathsf{k}_i] \in \mathbb{R}^n$ and *error-covariance* $P_i[\mathsf{k}_i] \in \mathbb{R}^{n \times n}$. This further implies that one can adopt a distributed KF solution in the sensor network for state estimation, e.g., [13, 46, 42, 33, 53, 54, 47, 16]. Such solutions typically compute a local estimate of x in each node i based on y_i and on the data exchanged by its neighboring nodes $j \in \mathcal{N}_i$. Existing methods on distributed Kalman filtering present an a priori solution on what data should be exchanged, at what time and with which nodes. Hence, for a given sensor network, a matched (static) estimation procedure is derived per node under predefined conditions. Such static estimation procedures are infeasible when deploying large-scale networked systems. Broken communication links, newly added nodes to an existing network, node failures and depleted batteries are just a few examples of operational events likely to occur in large-scale sensor networks. Solutions should thus be in place that enables the (data processing) sensor network to cope with these configuration changes by reconfiguring its own operation in runtime. These topics are often addressed by methods that establish a self organizing network, in which a feasible solution for unforeseen system changes is sought for during the operation of a network rather than during its design time.

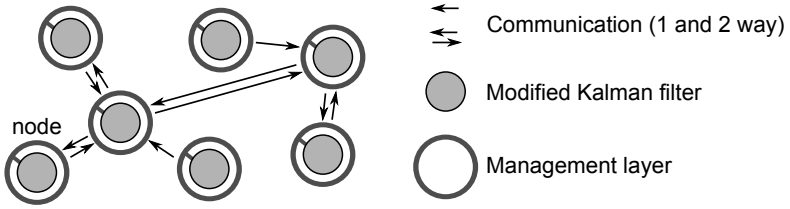


Figure 3: A network of Kalman filters with supporting management layer to realize the self-organizing property of the network.

Therefore, this chapter investigates a self-organization sensor network with the purpose of estimating the state vector of large-area processes. More specifically, the problem addressed is to integrate state-of-art results in distributed Kalman filtering with applicable solutions for establishing a self-organizing networked system. The (modified) Kalman filtering algorithms performed in the different nodes interact with each other via a management layer "wrapped around" the KF. The management layer is responsible for parametrization and

topology control, thus assuring coherent operational conditions for its corresponding estimator. Note that this warrants a two-way interaction between the modified KF and the management layer. Let us present the state-of-art in distributed Kalman filtering, next, before addressing the solutions that establish a self-organizing networked system.

4 Distributed Kalman filtering

The linear process model of (1) is characterized by Gaussian noise distributions. A well known state estimator for linear processes with Gaussian noise distributions is the KF, formally introduced in [29]. Since many distributed implementations of the KF make use of its original algorithm, let us define the Kalman filtering function $f_{\text{KF}} : \mathbb{R}^n \times \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}^m \times \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times m} \to \mathbb{R}^n \times \mathbb{R}^{n\times n}$. Different nodes will employ this function. Therefore, let us present a generalized characterization of f_{KF} independent of the node index i. To that end, let $y[\mathsf{k}] \in \mathbb{R}^m$ denote a measurement sampled at the synchronous sampling instants $\mathsf{k} \in \mathbb{Z}_+$ with a sampling time of $\tau \in \mathbb{R}_{>0}$ according to the following description:

$$y[\mathsf{k}] = Cx[\mathsf{k}] + v[\mathsf{k}], \qquad p(v[\mathsf{k}]) = G(v[\mathsf{k}], 0, V). \tag{2}$$

Then, a characterization of the Kalman filtering function, which computes updated values of the state estimates $\hat{x}[\mathsf{k}]$ and $P[\mathsf{k}]$ based on $y[\mathsf{k}]$ in (2), yields

$$(\hat{x}[\mathsf{k}], P[\mathsf{k}]) = f_{\text{KF}}(\hat{x}[\mathsf{k}-1], P[\mathsf{k}-1], A_\tau, Q_\tau, y[\mathsf{k}], C, V), \tag{3}$$

$$\begin{aligned} \text{with } M &= A_\tau P[\mathsf{k}-1] A_\tau^\top + Q_\tau; \\ K &= MC^\top (CMC^\top + V)^{-1}; \\ \hat{x}[\mathsf{k}] &= A_\tau \hat{x}[\mathsf{k}-1] + K(y[\mathsf{k}] - CA_\tau \hat{x}[\mathsf{k}-1]); \\ P[\mathsf{k}] &= (I_n - KC)M. \end{aligned} \tag{4}$$

The KF is a successful and well studied state-estimator. See, for example, some assessments presented in [2, 18, 59]. Its success is based on three aspects:

- Measurements are included iteratively;
- The estimation error $x - \hat{x}$ is asymptotically unbiased and attains the minimal quadratic value of the error-covariance P;
- The Kalman filtering algorithm is computationally tractable.

Therefore, when distributed solutions for state estimation became apparent, the Kalman filtering strategy was often the starting point for any novel distributed state-estimator. Moreover, many of the ideas explored in distributed Kalman filtering are easily extendable towards distributed state estimation in general. A summary of these ideas is given in the next sections, as it facilitates in the decision on how to compute a node's local estimate $p_i(x)$.

The overview on distributed Kalman filtering distinguishes two different approaches. In the first approach nodes exchange their local measurement, while in the second approach nodes share their local estimate (possibly additional to exchanging local measurements). This second approach was proposed in recent solutions on distributed Kalman filtering, as it further improves the estimation results in the network. For clarity of exposition, solutions are initially presented with synchronized sampling instants $\mathsf{k} \in \mathbb{Z}_+$, i.e., each node i has the same sampling instant $\tau \in \mathbb{R}_+$. After that, modifications are given to accommodate asynchronous sampling instants $\mathsf{k}_i \in \mathbb{Z}_+$ and local sampling times $\tau_i \in \mathbb{R}_+$.

4.1 Exchange local measurements

4.1.1 Synchronized sampling instants

First solutions on distributed KFs proposed to share local measurements. See, for example, the methods presented in [22, 55, 21, 13]. Local measurements are often assumed to be independent (uncorrelated). Therefore, they are easily merged with any existing estimate in a particular node. To reduce complexity even further, most methods do not exchange the actual measurement but rewrite y_i, C_i and V_i into an *information form*, i.e.,

$$z_i[\mathsf{k}] := C_i^\top V_i^{-1} y_i[\mathsf{k}] \quad \text{and} \quad Z_i[\mathsf{k}] := C_i^\top V_i^{-1} C_i, \quad \forall i \in \mathcal{N}. \tag{5}$$

Established terms for $z_i[\mathsf{k}] \in \mathbb{R}^n$ and $Z_i[\mathsf{k}] \in \mathbb{R}^{n\times n}$ are the *information vector* and *information matrix*, respectively. They are used in an alternative KF algorithm with equivalent estimation results but different computational complexity, known as the *Information filter*. To that extent, let us introduce the Information filtering function $f_{\text{IF}} : \mathbb{R}^n \times \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}^m \times \mathbb{R}^{m\times m} \to \mathbb{R}^n \times \mathbb{R}^{n\times n}$, for $z[\mathsf{k}] := C^\top V^{-1} y[\mathsf{k}]$ and $Z[\mathsf{k}] := C^\top V^{-1} C$ as the information form of the generalized measurement $y[\mathsf{k}]$ expressed in (2), i.e.,

$$(\hat{x}[\mathsf{k}], P[\mathsf{k}]) = f_{\text{IF}}(\hat{x}[\mathsf{k}-1], P[\mathsf{k}-1], A_\tau, Q_\tau, z[\mathsf{k}], Z[\mathsf{k}]), \tag{6}$$

$$\begin{aligned} \text{with } M &= A_\tau P[\mathsf{k}-1] A_\tau^\top + Q_\tau; \\ P[\mathsf{k}] &= (M^{-1} + Z[\mathsf{k}])^{-1}; \\ \hat{x}[\mathsf{k}] &= P[\mathsf{k}](M^{-1} A_\tau \hat{x}[\mathsf{k}-1] + z[\mathsf{k}]). \end{aligned} \tag{7}$$

Notice that a node i can choose between f_{KF} and f_{IF} for computing a local estimate of x. This choice depends on the format in which nodes share their local measurement information, i.e., the normal form (y_i, C_i, V_i) or the information form (z_i, Z_i), as well as the computational requirements of f_{KF} and f_{IF}. In addition, note that when the original KF is employed by a node i, i.e., $(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}]) = f_{\text{KF}}(\hat{x}_i[\mathsf{k}-1], P_i[\mathsf{k}-1], A_\tau, Q_\tau, \bar{y}_i[\mathsf{k}], \bar{C}_i, \bar{V}_i)$, then $\bar{y}_i[\mathsf{k}]$ is constructed by stacking $y_i[\mathsf{k}]$ with the received $y_j[\mathsf{k}]$ column wise[2], for all $j \in \mathcal{N}_i$. However, the distributed KF proposed in [13] showed that the administration required to construct $\bar{y}_i$, $\bar{C}_i$ and $\bar{V}_i$ can be simplified into an addition when local

[2]The parameters $\bar{C}_i$ and $\bar{V}_i$ can be constructed similar to $\bar{y}_i$.

measurements are exchanged in their information form instead. This implies that each node i performs the following function, which is also schematically depicted in Figure 4, i.e.,

$$\begin{aligned}(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}]) &= f_{\text{IF}}(\hat{x}_i[\mathsf{k}-1], P_i[\mathsf{k}-1], A_\tau, Q_\tau, \bar{z}_i[\mathsf{k}], \bar{Z}_i[\mathsf{k}]),\\ \text{with } \bar{z}_i[\mathsf{k}] &= z_i[\mathsf{k}] + \sum_{j\in\mathcal{N}_i} z_j[\mathsf{k}] \quad \text{and} \quad \bar{Z}_i[\mathsf{k}] = Z_i[\mathsf{k}] + \sum_{j\in\mathcal{N}_i} Z_j[\mathsf{k}].\end{aligned} \tag{8}$$

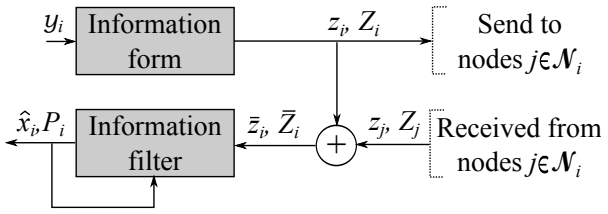


Figure 4: Schematic set-up of a node's local algorithm for estimating the state x according to a distributed KF where local measurements are exchanged in their information form.

This simple, yet effective, distributed KF triggered many novel extensions. For example, to reduce communication requirements by quantization of the measurement values, as presented in [47], or to estimate only a part of global state vector x in a node i, e.g., [40, 32]. However, a drawback when exchanging measurements is that node i receives localized data from the neighboring nodes $j \in \mathcal{N}_i$. Hence, only a part of the measurements produced by the sensor network is used for computing $\hat{x}_i$ and P_i. A solution to exploit more measurement information, as it is proposed in [42, 33], is to attain a consensus on local measurements. This means that, before f_{IF} is performed, each node i first employs a distributed consensus algorithm on $z_i[\mathsf{k}]$ and $Z_i[\mathsf{k}]$, for all $i \in \mathcal{N}$. Some popular consensus algorithms are found in [24, 61, 62, 58]. However, they require that neighboring nodes exchange data multiple times in between two sampling instants. Due to this demanding requirement, distributed KFs with a consensus on local measurements are not very popular. Other extension of the distributed KF presented in (8) take into account that the sampling instants of individual nodes can differ throughout the network. As this is also the case for the considered network, let us discuss the extension for asynchronous measurements, next.

4.1.2 Asynchronous sampling instants

The assumed sensor network of Section 3 has different sampling instants per node. This means that the k_i-th sample of node i, which corresponds to its local sampling instant $t_{\mathsf{k}_i} \in \mathbb{R}_+$, will probably not be equal to the time $t \in \mathbb{R}_+$ at which a neighboring node $j \in \mathcal{N}_i$ sends $\big(z_j(t), Z_j(t)\big)$. To address this issue, let us assume that node i received $\big(z_j(t), Z_j(t)\big)$ at time instant $t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}$.

Then, this received measurement information is first "predicted" towards the local sampling instant t_{k_i}, so that it can be used when node i runs its local estimation function f_{IF}. The results of [23] characterize such a prediction, for all $j \in \mathcal{N}_i$ and $t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}$ as follows:

$$\begin{aligned} z_j[\mathsf{k}_i|t] :=& \big(A^{-\top}_{t_{\mathsf{k}_i}-t}\big) z_j(t) + \big(\Phi^\top \Sigma^{-1} A^{-\top}_{t_{\mathsf{k}_i}-t}\big)\hat{x}_i(t) \\ & - \Phi(\Phi_{P_i} + Q^{-1}_{t_{\mathsf{k}_i}-t} + \Phi_{Z_j})^{-1} A^{-\top}_{t_{\mathsf{k}_i}-t}\big(\hat{x}_i(t) - z_j(t)\big), \\ Z_j[\mathsf{k}_i|t] :=& \Phi_{Z_j} + \Phi_{P_i}\big(\Phi_{P_i} + Q^{-1}_{t_{\mathsf{k}_i}-t}\big)\Phi_{P_i} - \Phi\big(\Phi + Q^{-1}_{t_{\mathsf{k}_i}-t}\big)^{-1}\Phi^\top, \end{aligned} \tag{9}$$

in which $\Phi_{P_i} := A^{-\top}_{t_{\mathsf{k}_i}-t} P_i^{-1}(t) A^{-1}_{t_{\mathsf{k}_i}-t}$, $\Phi_{Z_j} := A^{-\top}_{t_{\mathsf{k}_i}-t} Z_j^{-1}(t) A^{-1}_{t_{\mathsf{k}_i}-t}$ and $\Phi := \Phi_{P_i} + \Phi_{Z_j}$. Further, note that a node $j \in \mathcal{N}_i$ may have send multiple data packages in between t_{k_i-1} and t_{k_i} with local measurement information, for example, when node j has a smaller sampling time than node i.

The (predicted) measurement of (9) in information form can directly be used by an Information filter. This means that the values of $\hat{x}_i[\mathsf{k}_i]$ and $P_i[\mathsf{k}_i]$ are updated at the local sampling instant t_{k_i} of node i according to an algorithm that is similar to the one presented in (8), i.e.,

$$\begin{aligned} (\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) &= f_{\text{IF}}(\hat{x}_i[\mathsf{k}_i-1], P_i[\mathsf{k}_i-1], A_{\tau_i}, Q_{\tau_i}, \bar{z}[\mathsf{k}_i], \bar{Z}[\mathsf{k}_i]), \\ \text{with } \bar{z}[\mathsf{k}_i] &= z_i[\mathsf{k}_i] + \sum_{j\in\mathcal{N}_i} z_j[\mathsf{k}_i|t], \quad \forall t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}, \\ \bar{Z}[\mathsf{k}_i] &= Z_i[\mathsf{k}_i] + \sum_{j\in\mathcal{N}_i} Z_j[\mathsf{k}_i|t], \quad \forall t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}. \end{aligned} \tag{10}$$

Note that the above information filter assumes that local measurement are exchanged in the information form. A solution when nodes exchange local measurements in their normal form, i.e., (y_i, C_i, V_i), is to employ the Kalman filtering function f_{KF} for each time instant $t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}$ at which a new measurement is received. Such a procedure could reduce the computational demands of a node, since the prediction formulas of (9) are complex. Nonetheless, incorporation of local measurements $y_j(t)$ that are not sampled at the predefined sampling instants t_{k_i} requires much attention from the management layer of the individual node i. A more natural solution to this problem is obtained in distributed KFs that exchange local estimates instead of local measurements, which are presented, next.

4.2 Exchange local estimates

4.2.1 Synchronous sampling instants

The main advantage of exchanging local estimates is that measurement information spreads through the entire network, even under the condition that nodes exchange data only once per sampling instant. However, since local estimation results are exchanged, note that nodes require a method that can merge multiple estimates of the same state x into a single estimate. Various solutions of such

methods are found in literature. However, before addressing these methods, let us start by presenting the generalized estimation algorithm performed by each node i that corresponds to this type of distributed KF solutions.

Typically, solutions of distributed KF that exchange local estimates first merge the local measurement $y_i[\mathsf{k}]$ with the previous local estimate $p_i(x[\mathsf{k}{-}1])$ via a Kalman filter and thereby, compute the updated estimate $p_i(x[\mathsf{k}])$. This updated local estimate is then shared with neighboring nodes, due to which node i will receive the local estimate of nodes $j \in \mathcal{N}_i$. It will be shown that not every solution requires to share both the locally estimated mean as well as its corresponding error-covariance. Therefore, let us introduce set of received means at node i as $\mathcal{X}_i \subset \mathbb{R}^n$ and a corresponding set of received error-covariances as $\mathcal{P}_i \subset \mathbb{R}^{n\times n}$, i.e.,

$$\mathcal{X}_i[\mathsf{k}] := \{\hat{x}_j[\mathsf{k}] \in \mathbb{R}^n | j \in \mathcal{N}_i\}\,, \tag{11}$$

$$\mathcal{P}_i[\mathsf{k}] := \left\{P_j[\mathsf{k}] \in \mathbb{R}^{n\times n} | j \in \mathcal{N}_i\right\}. \tag{12}$$

The above information of the local estimation results at neighboring nodes, together with the node's own local estimate, i.e., $\hat{x}_i[\mathsf{k}]$ and $P_i[\mathsf{k}]$, will be used as input to a merging function. More precisely, let us introduce this merging function $\Omega : \mathbb{R}^n \times \mathbb{R}^{n\times n} \times \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}^n \times \mathbb{R}^{n\times n}$, which results in the merged Gaussian estimate $p_{i^+}(x[\mathsf{k}]) := G(x[\mathsf{k}], \hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}])$, as follows:

$$(\hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}]) = \Omega(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}], \mathcal{X}_i[\mathsf{k}], \mathcal{P}_i[\mathsf{k}]). \tag{13}$$

Then, the generalized local algorithm performed by a node $i \in \mathcal{N}$ for estimating the state, which is also depicted in the schematic set-up of Figure 5, yields

$$\begin{aligned}(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}]) &= f_{\text{KF}}(\hat{x}_{i^+}[\mathsf{k}{-}1], P_{i^+}[\mathsf{k}{-}1], A_\tau, Q_\tau, y_i[\mathsf{k}], C_i, V_i);\\ &\text{share } (\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}]) \text{ with all } j \in \mathcal{N}_i;\\ &\text{collect } (\hat{x}_j[\mathsf{k}], P_j[\mathsf{k}]) \text{ for all } j \in \mathcal{N}_i;\\ (\hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}]) &= \Omega(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}], \mathcal{X}_i[\mathsf{k}], \mathcal{P}_i[\mathsf{k}]).\end{aligned} \tag{14}$$

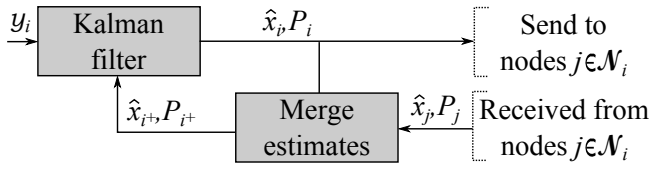


Figure 5: Schematic set-up of a node's local algorithm for estimating the state x according to a distributed KF where local estimates are exchanged.

Note that a suitable strategy for the merging function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is yet to be determined. Literature indicates that one can choose between three types of strategies, i.e., consensus, fusion and a combination of the two. A detailed account on these three strategies is presented next, by starting with consensus.

Consensus strategies aim to reduce conflicting results of the locally estimated means $\hat{x}_i$, for all $i \in \mathcal{N}$. Such an objective makes sense, as $\hat{x}_i$ in the different nodes i of the network is a local representative of the same global state x. Many distributed algorithms for attaining a consensus (or the average) were proposed, which all aim to diminish the difference $\hat{x}_i[\mathsf{k}] - \hat{x}_j[\mathsf{k}]$, for any two $i, j \in \mathcal{N}$. See, for example, the distributed consensus methods proposed in [24, 61, 62, 58]. The general idea is to perform a weighted averaging cycle in each node i on the local and neighboring means. To that extent, let $W_{ij} \in \mathbb{R}^{n \times n}$, for all $j \in \mathcal{N}_i$, denote some weighting matrices. Then, a *consensus* merging function $\Omega(\cdot, \cdot, \cdot, \cdot)$ is typically characterized as follows:

$$\begin{aligned}(\hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}]) &= \Omega(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}], \mathcal{X}_i[\mathsf{k}], \mathcal{P}_i[\mathsf{k}]),\\ \text{with } \hat{x}_{i^+}[\mathsf{k}] &= \Big(I_n - \sum_{\hat{x}_j[\mathsf{k}] \in \mathcal{X}_i[\mathsf{k}]} W_{ij}\Big)\hat{x}_i[\mathsf{k}] + \sum_{\hat{x}_j[\mathsf{k}] \in \mathcal{X}_i[\mathsf{k}]} W_{ij}\hat{x}_j[\mathsf{k}],\\ P_{i^+}[\mathsf{k}] &= P_i[\mathsf{k}].\end{aligned} \tag{15}$$

Note that the above consensus merging function is limited to the means and that the error-covariance of a node is not updated, due to which $\mathcal{P}_i[\mathsf{k}]$ can be the empty set. Further, most research on consensus methods concentrates on finding suitable values for the weights W_{ij}, for all $j \in \mathcal{N}_i$. Some typical examples of *scalar* weights were proposed in [24, 62], where $d_i := \sharp\mathcal{N}_i$ (number of elements within the set $\mathcal{N}_i$) and $\epsilon < \min\{d_1, \ldots, d_N\}$, i.e.,

Nearest neighboring weights	$W_{ij} := (1 - d_i)^{-1},$	$\forall j \in \mathcal{N}_i;$
Maximum degree weights	$W_{ij} := (1 - \epsilon)^{-1},$	$\forall j \in \mathcal{N}_i;$
Metropolis weights	$W_{ij} := (1 + \max\{d_i, d_j\})^{-1},$	$\forall j \in \mathcal{N}_i.$

An analysis on the effects of these weights, when they are employed by the consensus function in (15), was presented in [24, 62]. Therein, it was shown that employing *nearest neighboring weights* in (15) results in a bias on $\lim_{\mathsf{k}\to\infty} \hat{x}_{i^+}[\mathsf{k}]$. This is prevented by employing *maximum degree weights* or *metropolis weights*. However, *maximum degree weights* require global information to establish ϵ in every node, which reduces its applicability in sensor networks.

Employing a consensus strategies for merging the local estimates of neighboring nodes is very popular in distributed KFs. As a result, many extension of the above solution are found in literature. A common extension is to perform the averaging cycle not only on the means $\hat{x}_i[\mathsf{k}]$ and $\hat{x}_j[\mathsf{k}]$, as characterized in (15), but also on the error-covariances $P_i[\mathsf{k}]$ and $P_j[\mathsf{k}]$ of neighboring nodes. See, for example, the distributed KF proposed in [45] and a related solution presented in [7]. It is worth to point out that an in depth study on distributed KFs with a consensus on local estimates is presented in [5]. Therein, it is shown that minimization of the estimation error by jointly optimizing the Kalman gain K of f_{KF} and the weights W_{ij} of Ω is a non-convex problem. Hence, choosing the value of the Kalman gain K affects the weights W_{ij}, for all $j \in \mathcal{N}_i$, which raised new challenges. A solution for joint optimization on K and W_{ij} was

introduced in [43] as the *distributed consensus information filter*. However, a drawback of any consensus method is that the local error-covariance $P_i[\mathsf{k}]$ is not taken into account when deriving the weights W_{ij}, for all $j \in \mathcal{N}_i$. The error-covariance is an important variable that represents a model for the estimation error $cov(x[\mathsf{k}] - x_i[\mathsf{k}])$. Therefore, merging two local estimates $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$ in line with their individual error-covariance, implies that one can choose the value of W_{ij} such that the result after merging, i.e., $p_{i^+}(x[\mathsf{k}])$, is mainly based on the on the local estimate with the least estimation error. This idea is in fact the fundamental difference between a consensus approach and a fusion strategy. In fusion, both error-covariances $P_i[\mathsf{k}]$ and $P_j[\mathsf{k}]$ are explicitly taken into account when merging $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$, as it is indicated in the next alternative merging function based on fusion.

Fusion-consensus strategies is a label for characterizing some initial fusion solutions that are based on the fusion strategy *covariance intersection*, which was introduced in [25]. Fusion strategies typically define an algorithm to merge two prior estimates $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$ into a single, "fused" estimate. Some fundamental fusion methods presented in [55, 4] require that correlation of the two prior estimates is available. In (self-organizing) sensor networks one cannot impose such a requirement, as it amounts to keeping track of shared data between all nodes in the network. Therefore, this overview considers fusion methods that can cope with unknown correlations. A popular fusion method for unknown correlations is *covariance intersection*. The reason that this method is referred to as a fusion-consensus strategy, is because the fusion formula of *covariance intersection* is similar to the averaging cycle of (15) in consensus approaches. The method characterizes the fused estimate as a convex combination of the two prior ones. As an example, let us assume that node i has only one neighboring node j. Then employment of *covariance intersection* to characterize $\Omega(\cdot,\cdot,\cdot,\cdot)$ of (14) as a fusion function, for some $W_{ij} \in \mathbb{R}_{[0,1]}$, yields

$$
\begin{aligned}
P_{i^+}[\mathsf{k}] &= ((1-W_{ij})P_i^{-1}[\mathsf{k}] + W_{ij}P_j^{-1}[\mathsf{k}])^{-1}, \\
\hat{x}_{i^+}[\mathsf{k}] &= P_{i^+}[\mathsf{k}]((1-W_{ij})P_i^{-1}[\mathsf{k}]\hat{x}_i[\mathsf{k}] + W_{ij}P_j^{-1}[\mathsf{k}]\hat{x}_j^{-1}[\mathsf{k}]).
\end{aligned}
$$

Note that the above formulas indicate that the error-covariance $P_i[\mathsf{k}]$ and $P_j[\mathsf{k}]$ are explicitly taken into account when merging $\hat{x}_i[\mathsf{k}]$ and $\hat{x}_j[\mathsf{k}]$. Moreover, even the weight W_{ij} is typically based on these error-covariances, e.g., $W_{ij} = \mathrm{tr}(P_j[\mathsf{k}])(\mathrm{tr}(P_j[\mathsf{k}])+\mathrm{tr}(P_{i(l-1)}))^{-1}$ with some other examples found in [19, 41, 15]. As a result, the updated estimate $p_{i^+}(x[\mathsf{k}])$ computed by the merging function Ω will be closer to the prior estimate $p_i(x[\mathsf{k}])$ or $p_j(x[\mathsf{k}])$ that is "the most accurate one", i.e., with a smaller error-covariance. An illustrative example of this property will be given later on. For now, let us continue with the merging function in case node i has more than one neighboring node. Fusion of multiple estimates can be conducted recursively according to the order of arrival at a node. Therefore, the merging function $\Omega(\cdot,\cdot,\cdot,\cdot)$ based on fusion method

covariance intersection has the following characterization:

$$
\begin{aligned}
&(\hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}]) = \Omega(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}], \mathcal{X}_i[\mathsf{k}], \mathcal{P}_i[\mathsf{k}]),\\
&\text{with:} \qquad \text{for each received estimate } (\hat{x}_j[\mathsf{k}], P_j[\mathsf{k}]), \text{ do}\\
&\qquad\qquad \Sigma_i = \big((1-W_{ij})P_i^{-1}[\mathsf{k}] + W_{ij}P_j^{-1}[\mathsf{k}]\big)^{-1};\\
&\qquad\qquad \hat{x}_i[\mathsf{k}] = \Sigma_i\big((1-W_{ij})P_i^{-1}[\mathsf{k}]\hat{x}_i[\mathsf{k}] + W_{ij}P_j^{-1}[\mathsf{k}]\hat{x}_j^{-1}[\mathsf{k}]\big); \qquad (16)\\
&\qquad\qquad P_i[\mathsf{k}] = \Sigma_i;\\
&\qquad \text{end for}\\
&\qquad \hat{x}_{i^+}[\mathsf{k}] = \hat{x}_i[\mathsf{k}], \quad P_{i^+}[\mathsf{k}] = P_i[\mathsf{k}].
\end{aligned}
$$

Although *covariance intersection* takes the exchanged error-covariances into account when merging multiple estimates, it still introduces conservatism. Intuitively, one would expect that $p_{i^+}(x[\mathsf{k}])$ is more accurate than $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$, for all $j \in \mathcal{N}_i$, as prior estimates of neighboring nodes are merged. A formalization of this intuition is that $P_{i^+}[\mathsf{k}] \preceq P_i[\mathsf{k}]$ and $P_{i^+}[\mathsf{k}] \preceq P_j[\mathsf{k}]$ should hold for all $j \in \mathcal{N}_i$. One can prove that *covariance intersection* does not satisfy this property, due to which an alternative fusion method is presented, next.

Fusion strategies aim to improve the accuracy after fusion, for which the basic fusion problem is the same as previously mentioned, i.e., merge two prior estimates $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$ into a single, "fused" estimate $p_{i^+}(x[\mathsf{k}])$, when correlations are unknown. Some existing fusion methods are found in [63, 52, 51]. In this survey the *ellipsoidal intersection* fusion method of [52, 51] is presented, since it results in algebraic expressions of the fusion formulas. In brief, *ellipsoidal intersection* derives an explicit characterization of the (unknown) correlation a priori to deriving algebraic fusion formulas that are based on the independent parts of $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$. This characterization of the correlation, for any two prior estimate $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$, is represented by the *mutual covariance* $\Gamma_{ij} \in \mathbb{R}^{n\times n}$ and the *mutual mean* $\gamma_{ij} \in \mathbb{R}^n$. Before algebraic expressions of these variables are given, let us first present the resulting merging function $\Omega(\cdot,\cdot,\cdot,\cdot)$ when *ellipsoidal intersection* is employed in this function for fusion, i.e.,

$$
\begin{aligned}
&(\hat{x}_{i^+}[\mathsf{k}], P_{i^+}[\mathsf{k}]) = \Omega(\hat{x}_i[\mathsf{k}], P_i[\mathsf{k}], \mathcal{X}_i[\mathsf{k}], \mathcal{P}_i[\mathsf{k}]),\\
&\text{with:} \qquad \text{for each received estimate } (\hat{x}_j[\mathsf{k}], P_j[\mathsf{k}]), \text{ do}\\
&\qquad\qquad \Sigma_i = (P_i^{-1}[\mathsf{k}] + P_j^{-1}[\mathsf{k}] - \Gamma_{ij}^{-1})^{-1};\\
&\qquad\qquad \hat{x}_i[\mathsf{k}] = \Sigma_i\big(P_i^{-1}[\mathsf{k}]\hat{x}_i[\mathsf{k}] + P_j^{-1}[\mathsf{k}]\hat{x}_j^{-1}[\mathsf{k}] - \Gamma_{ij}^{-1}\gamma_{ij}\big); \qquad (17)\\
&\qquad\qquad P_i[\mathsf{k}] = \Sigma_i;\\
&\qquad \text{end for}\\
&\qquad \hat{x}_{i^+}[\mathsf{k}] = \hat{x}_i[\mathsf{k}], \quad P_{i^+}[\mathsf{k}] = P_i[\mathsf{k}].
\end{aligned}
$$

The mutual mean γ_{ij} and mutual covariance Γ_{ij} are found by a singular value decomposition, which is denoted as $(S, D, S^{-1}) = svd(\Sigma)$ for a positive definite $\Sigma \in \mathbb{R}^{n\times n}$, a diagonal $D \in \mathbb{R}^{n\times n}$ and a rotation matrix $S \in \mathbb{R}^{n\times n}$. As such, let

us introduce the matrices $D_i, D_j, S_i, S_j \in \mathbb{R}^{n\times n}$ via the singular value decompositions $(S_i, D_i, S_i^{-1}) = svd(P_i[\mathsf{k}])$ and $(S_j, D_j, S_j^{-1}) = svd(D_i^{-\frac{1}{2}} S_i^{-1} P_j[\mathsf{k}] S_i D_i^{-\frac{1}{2}})$. Then, an algebraic expression of γ_{ij} and Γ_{ij}, for some $\varsigma \in \mathbb{R}_+$ while $\{A\}_{qr} \in \mathbb{R}$ denotes the element of a matrix A on the q-th row and r-th column, yields

$$\begin{aligned} D_{\Gamma_{ij}} =& \mathrm{diag}_{q\in\mathbb{Z}_{[1,n]}}\big(\max[1, \{D_j\}_{qq}]\big), \\ \Gamma_{ij} =& S_i D_i^{\frac{1}{2}} S_j D_{\Gamma_{ij}} S_j^{-1} D_i^{\frac{1}{2}} S_i^{-1}, \\ \gamma_{ij} =& \big(P_i^{-1} + P_j^{-1} - 2\Gamma^{-1} + 2\varsigma I_n\big)^{-1} \times \\ & \big((P_j^{-1} - \Gamma^{-1} + \varsigma I_n)\hat{x}_i + (P_i^{-1} - \Gamma^{-1} + \varsigma I_n)\hat{x}_j\big). \end{aligned}$$

A suitable value of ς follows: $\varsigma = 0$ if $|1 - \{D_j\}_{qq}| > 10\epsilon$, for all $q \in \mathbb{Z}_{[1,n]}$ and some $\epsilon \in \mathbb{R}_{>0}$, while $\varsigma = \epsilon$ otherwise. The design parameter ϵ supports a numerically stable result of *ellipsoidal intersection*.

This completes the three alternatives that can be employed by the merging function $\Omega(\cdot,\cdot,\cdot,\cdot)$. Before is continued with an extension of this merging function towards asynchronous sampling instants, let us first present an illustrative comparison of the two fundamentally different approaches. An illustration of this comparison is depicted in Figure 6, which is established when $p_i(x[\mathsf{k}])$ and $p_j(x[\mathsf{k}])$ are either the result of a fusion or a consensus approach. The consensus result is computed with the averaging cycle of (15) and $W_{ij} = 0.1$. Recall that only the means $\hat{x}_i[\mathsf{k}]$ and $\hat{x}_j[\mathsf{k}]$ are synchronized and not their error-covariances. The fusion result is computed with *ellipsoidal intersection* of (17). Let us further point out that Figure 6 is not included to decide which method is better. It is merely an example to illustrate the goal of consensus (reduce conflicting results) with respect to the goal of fusion (reduce uncertainty).

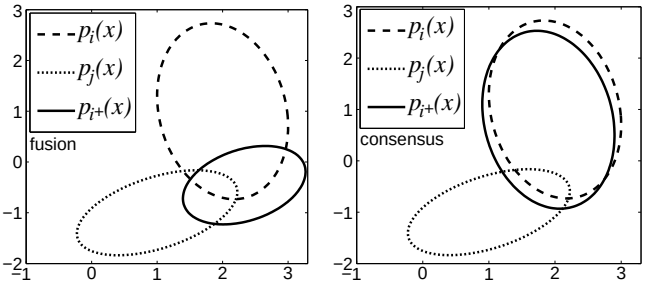


Figure 6: A comparison of consensus versus fusion. Note that PDFs are represented as ellipsoidal sub-level-set, i.e., $G(\theta, \mu, \Sigma) \to \mathcal{E}_{\mu,\Sigma}$. A graphical characterization of such a sub-level-set is found in Figure 2, though let us point out that a larger covariance Σ implies a larger area-size of $\mathcal{E}_{\mu,\Sigma}$.

4.2.2 Asynchronous sampling instants

The assumed networked system of Section 3 has different sampling instants per node. This implies that the k_i-th sample of node i, which corresponds to the sampling instant $t_{\mathsf{k}_i} \in \mathbb{R}_+$, will probably not be equal to the time $t \in \mathbb{R}_+$ at which a neighboring node $j \in \mathcal{N}_i$ sends $(\hat{x}_j(t), P_j(t))$. Compared to exchanging measurements, asynchronous sampling instants can be addressed more easily for distributed KF solutions that exchange local estimates. More precisely, the received variables $(\hat{x}_j(t), P_j(t))$ should be predicted from time t towards the sampling instant t_{k_i}, i.e.,

$$\begin{aligned} \hat{x}_j[\mathsf{k}_i|t] &:= A_{t_{\mathsf{k}_i}-t}\hat{x}_j(t), && \forall j \in \mathcal{N}_i, t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}, \\ P_j[\mathsf{k}_i|t] &:= A_{t_{\mathsf{k}_i}-t}P_j(t)A^\top_{t_{\mathsf{k}_i}-t} + Q_{t_{\mathsf{k}_i}-t}, && \forall j \in \mathcal{N}_i, t \in \mathbb{R}_{(t_{\mathsf{k}_i-1}, t_{\mathsf{k}_i}]}. \end{aligned} \tag{18}$$

Then, solutions of distributed Kalman filtering that are in line with the set-up depicted in (14) can cope with asynchronous sampling instants by re-defining $\mathcal{X}_i[\mathsf{k}_i]$ and $\mathcal{P}_i[\mathsf{k}_i]$ as the collection of the above predicted means $\hat{x}_j[\mathsf{k}_i|t]$ and error-covariances $P_j[\mathsf{k}_i|t]$, for all $j \in \mathcal{N}_i$.

This completes the overview on distributed Kalman filtering, in which nodes can adopt a strategy that exchanges local measurements or local estimates. Next, existing self-organization methods are presented, though an analysis of the required resources for estimation is studied first.

5 Required resources

The distributed KFs presented in the previous section are typically proposed for static sensor networks. However, the focus of this chapter is to extent those methods for sensor networks that have to deal with changes in the networked system. To cope with these changes, nodes must be able to adapt the conditions of their local estimation algorithm, or even choose a local algorithm that is based on a different type of distributed KF. In order to carry out these reconfiguration processes certain design decisions should be made in runtime depending on the available resources (e.g. how to reassign the KF tasks in case of node failures, what type of KF algorithms are feasible to run under given communication constraints, etc.) Therefore, this section presents a summary of the required resources for the different distributed KF strategies. Important resources in sensor networks are communication and computation. Let us start by addressing the communication demand of a node i. Section 4 indicates that there are three different types of data packages that a node can exchange, i.e., the local measurement $y_i \in \mathbb{R}^{m_i}$ in normal form or information form, and the local estimate of $x \in \mathbb{R}^n$. The resulting communication demands of node i that correspond to these different data packages are listed in Table 1.

Next, let us indicate the computational demand of a node i by presenting the algorithm's complexity of the different functionalities that can be chosen to compute $p_i(x)$. This complexity involves the number of floating points operations

Table 1: The communication demand in the amount of elements (floating points) that is exchanged by each node depending on the data that is shared.

exchanged data	communication demand
(y_i, C_i, V_i)	$m_i^2 + 2m_i + n$
(z_i, Z_i)	$n^2 + n$
$(\hat{x}_i, P_i)$	$n^2 + n$

depending on the size of local measurements $y_i \in \mathbb{R}^{m_i}$ and state vector $x \in \mathbb{R}^n$. To that extent, the following properties on the computational complexities of basic matrix computations are used:

- The summation/subtraction of $A \in \mathbb{R}^{q\times r}$ with $B \in \mathbb{R}^{q\times r}$ requires $O(qr)$ operations;
- The product of $A \in \mathbb{R}^{q\times r}$ times $B \in \mathbb{R}^{r\times p}$ requires $O(qrp)$ operations;
- The inverse of $A \in \mathbb{R}^{q\times q}$ invertible matrix requires $O(q^3)$ operations;
- The singular value decomposition of $A \in \mathbb{R}^{q\times q}$ requires $O(12q^3)$ operations;

Then, the resulting computational complexity of the Kalman filtering functions f_{KF} and f_{IF} and of the three merging functions Ω, i.e., characterized by a consensus, fusion-consensus and fusion strategy, are listed in Table 2.

Table 2: The computational demand in the amount of floating points operations depending on the employed functionality, where $M_i := m_i + \sum_{j\in\mathcal{N}_i} m_j$.

functionality	computational demand
f_{KF}	$\approx O(4n^3 + 3M_i n^2 + 2nM_i^2 + M_i^3)$
f_{IF}	$\approx O(3n^3 + M_i n^2 + nM_i^2)$
Ω consensus of (15)	$\approx O(3n + M_i + 1)$
Ω fusion-consensus of (16)	$\approx O(3n^3 + 9n^2)$
Ω fusion of (17)	$\approx O(31n^3 + 7n^2)$

The next section makes use of the above tables to decide what type of data should be exchanged between neighboring nodes and which functionalities should be followed in the local estimation algorithm of a node.

6 Self-organizing solutions

The design challenge of any embedded system is to realize given functionalities, in this case the ones of the local estimation algorithm, on a given hardware

platform while satisfying a set of non-functional requirements, such as response times, dependability, power efficiency, etc. Model-based design has been proven to be a successful methodology for supporting the system design process. Model-based methodologies use multiple models to capture the relevant properties of the design (when the required functionalities are mapped onto a given hardware configuration), e.g., a model of the required functionalities, temporal behavior, power consumption and hardware configuration. These models can then by used for various purposes, such as automatic code generation, architecture design, protocol optimization, system evolution and so on. Important for the design process are the interactions between the different models, which can be expressed as constraints, dependencies, etc. In this section a model-based design methodology is followed to assure *dependability* for state estimation in a sensor network via runtime reconfiguration.

To illustrate the model guided design process for distributed signal processing let us consider an example. Two fundamental models for system design are emphasized here: the task model (capturing the required functionalities) and the physical model (capturing the hardware configuration of the implementation). For the sake of simplicity a particular hardware configuration and communication topology is assumed; the question to answer is how the required functionalities can be realized on the given configuration, as shown in Figure. 7.

The task model in this figure is represented as directed graph wherein the signal processing components (tasks) are represented by the vertices of the graph, while their data exchange (interactions) are represented by the edges. Both the tasks as well as the interactions are characterized by a set of properties, which typically reflect non-functional requirements or constraints. These properties are used to determine system level characteristics and thus the feasibility of certain design decisions can be tested (see details later). The tasks run on a connected set of processors, represented by the physical model of the system. The components of the physical model are the computing nodes, i.e., consisting of processor, memory, communication and perhaps other resources, and the communication links. During the system design the following steps are carried out (typically it is a iterative process with refinement cycles [3] - but the iterations are not considered here):

- Select the algorithms for the processing realized by the tasks;
- Compose the task model;
- Select the hardware components for the physical model;
- Select a communication topology;
- Establish the mapping between the task model and the physical model.

The design process involves a particular mapping that defines the assignment of a task T_r to a processor P_q, i.e., it determines which task runs on which node. [3] Obviously the memory and execution time requirements define

[3]We assume that nodes are equipped with a multitasking runtime environment, consequently multiple tasks can be assigned to a single node.

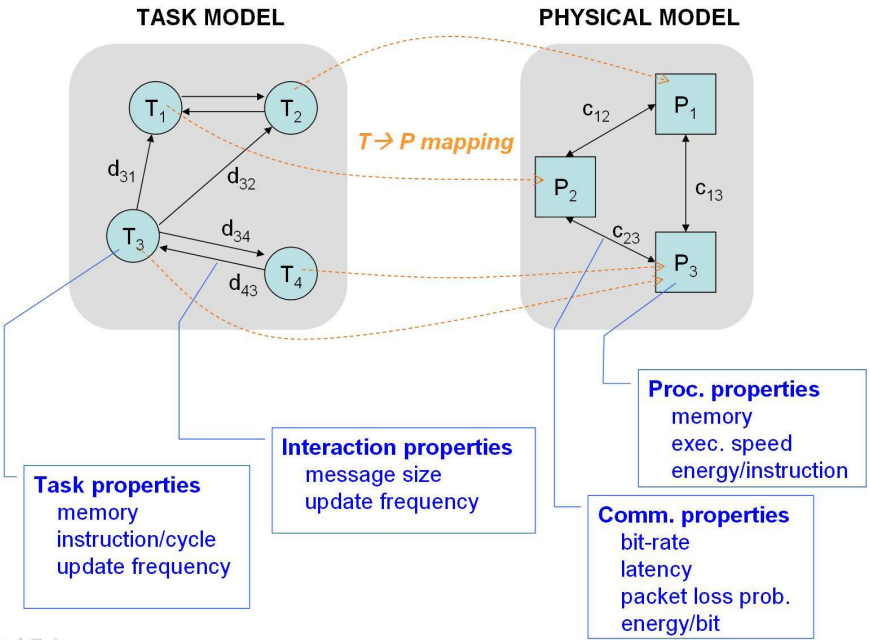


Figure 7: Modeling of signal processing and implementation.

constraints when assigning the tasks to nodes. Further, data exchange between tasks makes the assignment problem more challenging in distributed configurations, as a task assignment also defines the use of communication links - and the communication links have limited capabilities (indicated by the attached property set in Figure 7). After every refinement cycle, according to the steps listed above, the feasibility of a resulting design should be checked. For example, an assignment of T_3 to P_3 and T_4 to P_1 may yield an unfeasible design if the interaction d_{34} imposes too demanding requirements on the communication link c_{13}, i.e., high data exchange rate or large data size. On the other hand, assigning both T_3 and T_4 to P_3 may violate the processing capability constraint on P_3. Changing the hardware configuration and/or using less demanding algorithms (and eventually accepting the resulting lower performance) for implementing T_3 or T_4 could be a way out.

Note that the design process results in a sequence of decisions, which lead to a feasible system design. Traditionally the design process is "offline" (*design time*), i.e., it is completed before the implementation and deployment of the system itself. The task model, the hardware configuration and their characteristics are assumed to be known during this design time and the design uncertainties are assumed to be low. Under these conditions a model-based optimization can be carried out, delivering an optimal architecture ready for implementation. Unfortunately these assumptions are overly optimistic in a wide spectrum of application cases.

(Wireless) sensor networks deployed for monitoring large-scale dynamical processes are especially vulnerable. Sensor deterioration, node failure, unreliable communication, depleted batteries, etc., are not exceptions but common events in normal operation. These events result in changes in the system configuration, as it is captured by the physical model, due to which implementations relying on static designs may fail to deliver according to the specifications. A possible work-around is to build redundancy into the system and thereby, to implement fault-tolerance. In this case the top-level functionalities remain intact until a certain level of "damage" is reached. This approach usually leads to complex and expensive implementations - unacceptable for the majority of applications. The components are "under-utilized" in nominal operation, while power consumption is increased due to the built-in redundancy. The other approach is to accept the fact that maintaining a static configuration is not feasible and make the system such that it "follows" those changes and "adjusts" its internals to assure an implementation of the assigned functionalities as far as it is feasible. The resulting behavior typically manifests "graceful degradation" property, i.e., until damage reaches a certain level the set of functionalities and their quality can be kept; beyond that level the system loses non-critical functionalities and/or the quality of running functionalities is reduced due to a shortage of resources. Realizing this latter approach has significant impact both on system design and on the runtime operation of the system. Conceptually the system design process is not completely finished in design time, instead a set of design alternatives are provided for execution. During operation - depending on the health state of the configuration and the conditions of the embedding environment - a selection is made automatically to assure an optimal use of available resources, i.e., providing the highest level of the functionalities under the given circumstances. In the next section typical solutions for implementing this latter approach are overviewed.

6.1 Approaches to runtime adaptivity

Evolution of large-scale networked embedded systems in general and (wireless) sensor networks in particular poses a number of technical challenges on the design, implementation, testing, deployment and operation processes [30]. Considering the reconfiguration as a "vehicle" to implement such evolution, the reconfiguration of the functionalities on the available hardware can be carried out at four different stages of the system's life-cycle:

(i) design time - configuration redesign, new code base, etc.,

(ii) load time - new functionalities are implemented via code update,

(iii) initialization time - during system (or component) startup the optimal design alternative is selected and parameterized depending on a "snapshot" of the context,

(iv) runtime - reconfiguration is performed while the system is in use.

Here only the *runtime* reconfiguration variant of the evolution is considered with special emphasis on the needs of distributed Kalman filtering.

In case of runtime reconfiguration the reconfiguration process is triggered by observation of changes in the embedding environment of the system or in the system itself, e.g., realizing node failure or a low battery status. The "trajectory" for reconfiguration is not predefined but is a result of an optimization process attempting to maximize the "usefulness" of the system as defined by a performance criterion. The concept of the reconfiguration process is illustrated in Figure 8.

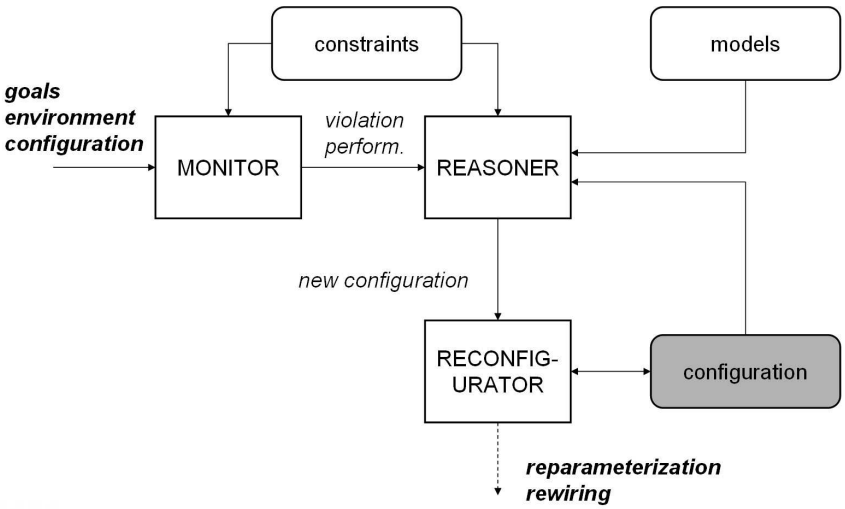


Figure 8: The reconfiguration process.

The process relies on the model-based approach as introduced above. The relevant models of the system, such as the task model, physical model, temporal model, etc., are formalized and stored in an efficiently accessible way in a database represented by the *models* block. The *constraints* block represents the dependencies in the models and between models. During operation of the signal processing systems the $MONITOR$ collects information about several aspects of the operation. Goals of the operation may change depending on, for example, different user needs. Changes in the observed phenomenon may cause that the models assumed in design time have become invalid. Similarly, internal changes in the system configuration should be recognized, such as broken communication and sensor failure. The $MONITOR$ functionality checks if the observed changes result in violating certain constraints of the systems or a significant drop in performance. If the $MONITOR$ concludes that under current circumstances the system cannot perform as requested, then the reconfiguration process is initiated. The central component is the $REASONER$, which based on the models, constraints and the actual findings, determines a new configu-

ration that satisfies all constraints and provides an acceptable performance. It should be emphasized the $REASONER$ may not carry out pure logical reasoning but also other types of search and optimization functions depending on the representation used to describe the models, goals and so on. The new configuration is passed to the $RECONFIGURATOR$ functionality to plan and execute the sequence of operations for "transforming" the old into the new configuration in runtime.[4]

Note that the reconfiguration process of Figure 8 runs on the same embedded monitoring system that is used for signal processing. An efficient implementation of this runtime reconfiguration should address three challenges:

- Representation: What are the right formalisms to describe the models and their interaction? To what extent should the models be made part of the running code? What is an efficient model representation in runtime?
- Monitoring: How can we collect coherent information about the health state of the system, even in case of failures? How can we deduct the potentially disruptive situations, i.e., which should trigger reconfiguration actions, from the raw observation set?
- Reasoning: What are the efficient algorithms, which are matching with the model representation, to resolve the conflicts rising from changes in the environment and/or in the system configuration? What are the chances for a distributed solution of the reasoning process?

There are no ultimate answers to these questions. The application domains have crucial impact on the optimal representation and reasoning, as well as on the resources that are required to run the reconfiguration process itself. Consequently, a thorough analysis of the application in hand, its typical failure modes, the dependability requirements and other relevant aspects of the system in its environment jointly identify the proper selection of techniques for setting up a suitable runtime reconfiguration process.

The research area of runtime reconfigurable systems design is quickly evolving. Established domains as self-adaptive software systems [8] and dynamically reconfigurable hardware systems [20, 6] provide fundamental contributions. In the following a few characteristic approaches are briefly addressed. A reconfiguration methodology based on model integrated computing (MIC) was introduced in [31]. Therein, the designer describes all relevant aspects of the system as formal models. A meta-modeling layer supports the definition of these relevant aspects that are to be modeled and generates the necessary model editors, i.e., carries out model analysis, verification, etc. The program synthesis level

[4]The operations for "transforming" the configuration act on the program modules implementing the task graph and on a "switchboard" realizing the flexible connections among the tasks. Consequently, the program modules should implement a "standard" API, which allows for a function independent, unified configuration interface to software components. This way parameter changes in the signal processing functions and in the connections between these functions can be carried out irrespective to the actual functions involved in the processing.

consists of a set of model interpreters, which according to the supplied models and constraints generate program code. The reconfiguration is triggered by changes in the models or constraints, which initiates a new model interpretation cycle. Though MIC provides a flexible way to describe and implement reconfigurable systems, the model interpretation is a computationally demanding step and may seriously limit the applicability in real-time cases. Alternatively, a model-oriented architecture with related tools for runtime reconfigurable systems was presented in [39]. This approach uses variability, context, reasoning and architecture models to capture the design space. In runtime the interactions among the event processor, goal-based reasoner, aspect model weaver and the configuration checker/manager components will carry out the reconfiguration. The approach is well suited for coping with high number of artifacts but the real-time aspect is not well-developed. A formalization of the reconfiguration as a constraint satisfaction problem was proposed in [56, 34, 38]. The design space is (at least partially) represented and its design constraints are explicitly stated. These methodologies implement a "constraint guided" design space exploration to find feasible solutions under the observed circumstances. In parallel a suitable performance criterion is calculated to guide the reconfiguration process to optimal solution. The method described in [56] is also capable of hardware/software task migration and morphing. Different reconfiguration solutions were developed for service oriented architectures (SOAs). For example, the reconfiguration method introduced in [28] extends the "traditional" discover - match - coordinate SOA scheme with a hierarchical service overlay mechanism. This service overlay implements a composition functionality that can dynamically "weave" the required services from the available service primitives. In [35] a solution is proposed that follows an object centric paradigm to compose the compound services. By modeling the service constraints, an underlying constraints satisfaction mechanism implements the dynamic service configuration. A different approach was presented in [60], which describes a model-based solution to validate at runtime that the sensor network functionalities are performed correctly, despite of changes in the operational conditions. It models the application logic, the network topology and the test specification, which are then used to generate diagnostic code automatically. Though the solution does not address the $REASONING$ functionality of Figure 8, it delivers low false negative detection rates, i.e., it covers the $MONITOR$ functionality effectively.

6.2 Implementation of runtime reconfiguration

The runtime reconfiguration brings in an extra aspect of complexity, which is "woven" into the functional architecture of the system and thus makes the testing and validation extremely challenging. To keep the development efforts on a reasonable level both design and implementation support is needed. Many of the runtime reconfiguration approaches cited propose an architectural methodology, design tool set and runtime support, e.g., [17, 31, 30, 56, 28, 60, 34, 35]. A common feature of these efforts is to support the system developer with application independent reconfiguration functionalities, which can be parameterized

according to the concrete needs of the application at hand. They also attempt to "separate concerns" when feasible, i.e., try to make the design of the functional architecture and the reconfiguration process as independent as possible, while still maintaining clear interactions between them. Typically, the corresponding reconfiguration functionalities manifest themselves in an additional software layer between the "nominal" real-time executive layer, such as TinyOS [36] or Contiki [12], and the application layer. See Figure 9 for more details. The (application independent) monitoring and reconfiguration functionalities in this figure receive the application specific information from "outside" in the form of models. Conceptually they are "interpreters". As such, they realize a virtual machine dedicated to a certain type of computational model, e.g., rule based inference, finite sate machine, constraint satisfaction, and so on. They read-in the application specific "program", which is represented by the *reconfiguration rules* component in Fig. 9, and interpret its code in the context of the data received from the *MONITOR* function. For example, if the reconfiguration process is based on a rule-based representation of the application specific knowledge[5], then the *REASONER* implements a forward changing (data driven) inference engine [49], in which using the *actual configuration* and the data received from *MONITOR* as fact base. The inference process results in derived *events* and *actions*, which define the reconfiguration commands issued for the application layer. The application program is characterized by (multi-aspect) models, requirements and constraints that are created by the designer according to, for example, [60]. For efficiency reasons the models created by a designer are rarely used directly by the reconfiguration process. Instead, after thorough compile-time checking, these models are translated to a "machine friendly" format to enable resource-aware access and transformations. The models can also be used for automatic code generation and synthesis to create the application code if the appropriate tools are available [31]. The monitoring functionality of Figure 9 (equivalent to the one in Figure 8) defines the set of observations that a reconfiguration process should take into consideration. Typically, this monitoring should cover the operational characteristics of an application, e.g., sensor noise level and estimator variance, combined with the health state of its execution platform, e.g. battery energy level and quality of communication channels. The reconfiguration rules then define the "knowledge base" of the reasoner/reconfigurator, which is also depicted in Figure 9, i.e., they determine how the recognized changes in operational conditions are handled. Note that reconfiguration rules do not necessarily refer to rule-based knowledge base but that the format and content of the "rules" is determined by a reasoning procedure, e.g., constraint satisfaction, graph matching, first-order logic, etc. Further note that the application layer of Figure 9 uses a number of reconfigurable components (c_1 ... c_n) to implement the required application level functionality. These components should implement a unified application programming interface (API), so that the middleware layer is able to retrieve information for its monitoring purposes and for executing its reconfiguration commands.

[5]This type of formalism will be used in the case study later in the chapter.

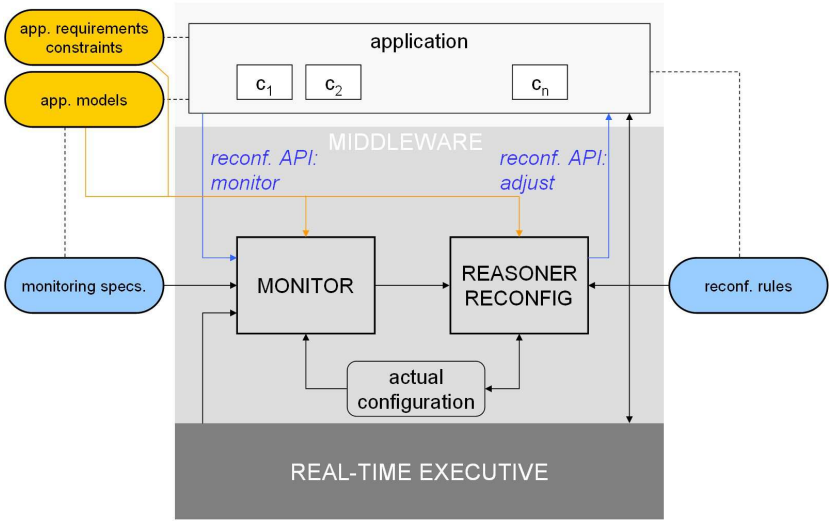


Figure 9: Middleware for runtime reconfiguration.

It should be emphasized that in certain applications the reconfiguration decisions could rely on system-wide information, In these cases the monitoring and reconfiguration activities inherently involve communication, resource scheduling, etc. This adds an extra layer of complexity to the systems, e.g., implementing distributed snap-shot algorithms, leader election and distributed reasoning/planning, that may demand resources beyond the capabilities of the nodes. A work-around is to give up the fully distributed implementation of the reconfiguration and assign the most demanding functionalities to (dedicated) powerful nodes, as proposed in [31, 34, 56]). The monitoring information is then forwarded to the reconfiguration node(s) where a new configuration is determined. The reconfiguration commands are transferred back to the nodes for synchronized execution.

In the next section the role of runtime reconfiguration will be demonstrated. It follows from the inherent network topology properties assumed in distributed state estimation that reconfiguration decisions are based on the information from local and neighboring nodes. As such, a distributed implementation of runtime reconfiguration is feasible, even on nodes of moderate computing capabilities.

7 Case study on a diffusion process

The results of the presented self-organizing sensor network for state estimation are demonstrated and evaluated in a spatio-temporal 2D diffusion process. The goal of the sensor network is to follow the contaminant's distribution profile in time (i.e. the concentration distribution in space and time of a particular

chemical compound) in the presence of wind. To that extent, let us consider an area of 1200×1200 meters containing a contaminant source. As time passes, the contaminant spreads across the area due to diffusion and wind. To simulate the spread, let us divide the area into a grid with a grid-size of 100 meters. The center of each grid-box is defined as a grid-point. Then, the spread of the contaminant is represented by the concentration level $\rho^{(q)} \in \mathbb{R}_+$ at the q-th grid-point $q \in [1, 144]$. This concentration level $\rho^{(q)}$ depends on the corresponding levels at neighboring grid-points, which are denoted as q_n for north, q_s for south, q_e for east and q_w for west. See Figure 10 for a graphical representation of these grid-points relative to the q-th grid-point. Further, the *continuous-time* process model of $\rho^{(q)}$, for some $a, a_n, a_s, a_e, a_w \in \mathbb{R}$, yields

$$\dot{\rho}^{(q)} = a\rho^{(q)} + a_n\rho^{(q_n)} + a_s\rho^{(q_s)} + a_e\rho^{(q_e)} + a_w\rho^{(q_w)} + u^{(q)}, \quad \forall q \in \mathbb{Z}_{[1,144]}.$$

The variable $u^{(q)} \in \mathbb{R}_+$ in (7) parameterizes the production of chemical matter by a source at grid-point q and follows $u^{(18)} = 75$, $u^{(29)} = 75$, $u^{(30)} = 100$, $u^{(31)} = 100$ and $u^{(42)} = 175$ for all time $t \in \mathbb{R}_+$, while $u^{(q)} = 0$ for all other $q \in \mathbb{Z}_{[1,144]}$. The remaining parameters are chosen to establish a northern the wind direction, i.e., $a = \frac{-12}{800}$, $a_n = \frac{1}{800}$, $a_s = \frac{2}{800}$, $a_e = \frac{7}{800}$ and $a_w = \frac{2}{800}$.

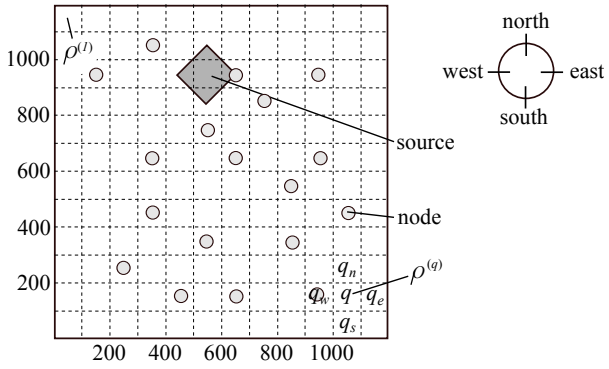


Figure 10: The monitored area is divided into a grid. Each grid-point q has four neighbors q_n, q_s, q_e and q_w, i.e., one to the north, south, east and west of grid-point q, respectively. The chemical matter produced by the source spreads through the area due to diffusion and wind.

A sensor network is deployed in the area to reconstruct the concentration levels at each grid-point based on the local measurements taken by each node.

Communication The network consists of 18 sensor nodes that are randomly distributed across the area, see also Figure 10. It is assumed that the sensor nodes communicate only with their direct neighbors, i.e. nodes with a 1-hop distance, and that their position is available.

Process Neither the wind direction nor values of the contaminant source are available to the nodes. Therefore, the process model that is used by the

local estimation algorithms of the different nodes is a simplified diffusion process in *continuous-time*, i.e.,

$$\dot{\rho}^{(q)} = \alpha\rho^{(q)} + \alpha_n\rho^{(q_n)} + \alpha_s\rho^{(q_s)} + \alpha_e\rho^{(q_e)} + \alpha_w\rho^{(q_w)} + w^{(q)},$$

with $\alpha = \frac{-12}{800}$, $\alpha_n = \frac{3}{800}$, $\alpha_s = \frac{3}{800}$, $\alpha_e = \frac{3}{800}$ and $\alpha_w = \frac{3}{800}$. The unknown source and model uncertainties are represented by process-noise $w^{(q)} \in \mathbb{R}$, for all $q \in \mathbb{Z}_{[1,144]}$. A suitable characterization of this noise, i.e., to cover unknown source values $u^{(q)}$ in between -150 and 150, is given by the *continuous-time* PDF $p(w^{(q)}(t)) = G(w^{(q)}(t), 0, 2\cdot 10^3)$. Further, the state is defined as the collection of all concentration levels, i.e., $x := \begin{pmatrix}\rho^{(1)} & \rho^{(2)} \cdots \rho^{(144)}\end{pmatrix}^\top$. The model parameters A_{τ_i} and Q_{τ_i} of the *discrete-time* process model in (1) are characterized with the initial sampling time of $\tau_i = 10$ seconds, for all nodes $i \in \mathcal{N}$. To determine the other process model parameters, i.e., C_i and V_i, it is assumed that each sensor node i measures the concentration level at its corresponding grid-point, i.e., $y_i[\mathsf{k}_i] = \rho^{(q)}[\mathsf{k}_i] + v_i$, for some $q \in \mathbb{Z}_{[1,144]}$ and $p(v_i[\mathsf{k}_i]) = G(v_i[\mathsf{k}_i], 0, 0.5)$, for all $i \in \mathbb{Z}_{[1,18]}$. The real concentration levels at the three time instants $t = 140$, $t = 240$ and $t = 340$ are illustrated in Figure 11.

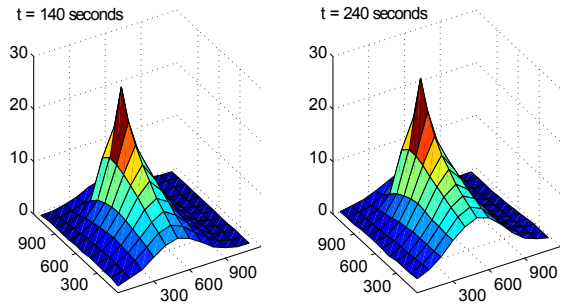


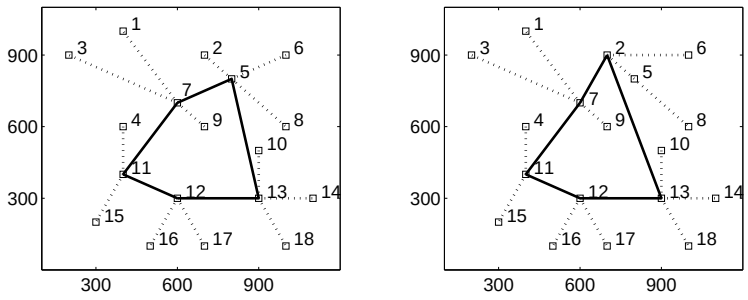
Figure 11: The simulated concentration levels at the different grid-points for two instances of the time $t \in \mathbb{R}_+$.

The objective of the sensor network is to determine the contaminant distribution by estimating the state x in multiple nodes of the network. This is carried out in two types of sensor networks, a hierarchical network and a fully distributed one. In each configuration unforseen events occur indicating node break-down and batteries depleting below critical energy level. The nodes must adapt their local state estimating functionalities to recover from lost neighbors and/or to reduce their energy consumption so that batteries do not get depleted. Let us start this analysis with the hierarchical network.

7.1 A hierarchical sensor network

In a *hierarchical* sensor network nodes are given specific tasks prior to its deployment. Basically, the network consists of multiple subnetworks, as it is illustrated

in Figure 12(a). In each subnetwork nodes exchange their local measurements with the center node of that particular subnetwork (denoted with dashed lines). The center node computes a local estimate based on these received measurements via f_{KF}, after which this estimate is shared with the center nodes of other subnetworks (denoted with the solid lines). The received estimates are then fused with the local estimate according to the merging function f_{ME} and the fusion method *ellipsoidal intersection* of (17).



(a) Initial topology (b) Topology after 250 seconds

Figure 12: Network topology in a hierarchical network.

Two events will occur in this network, followed by the corresponding action as it is implemented in the reconfiguration process of each node. The reconfiguration is local: operational events are monitored locally and the reconfiguration actions influence only the node issued the request for action. A rule-based representation formalism is used to define the "knowledge base" of the reconfiguration functionality. As such, the $REASONER$ component of the middleware implements a forward chaining rule interpreter, i.e., if *event* then *action* ([49, 10, 11]). For clarity of the illustrative example, we do not attempt a rigorously formal description of the knowledge base but only the "style" of the rule-based representation is shown.

- At $t = 150$ seconds nodes 1, 3 and 8 will cross their critical energy level;
  If *the critical energy level is crossed,* then *lower the node's local sampling time from* 10 *seconds to* 20 *seconds.*

- At $t = 250$ seconds node 5 will break down. To detect whether a state estimating nodes brakes down, nodes within each subnetwork exchange acknowledgements or heartbeat messages are used to indicate normal operational mode;
  If *the acknowledgement of the state estimating node is not received,* then *check the energy levels of all other nodes in the corresponding subnetwork. The node with the largest energy level takes over the responsibility for estimating the state, according to an algorithm that is similar to the node that broke down. Also, re-establish the connection with the other subnetworks.*

As an example, the rule set below shows the handling of the #2 event

```
Rule_2a:
    IF   NEIGHBOR(?x) & TIMEDOUT(?x) & ?x.function = centerfun
    THEN set(go_for_newcenter,TRUE)

Rule_2b:
    IF   go_for_newcenter & NEIGHBOR(?x) &
         !TIMEDOUT(?x) & max(?x.power) = self.power
    THEN exec(assign,centerfun), exec(broadcast,centerfun_msg)
```

Figure 12(a) depicts the network topology prior to the event that node 5 brakes down, while Figure 12(b) illustrates this topology after the event (assuming that the battery of node 2 has the highest energy level). This figure indicates that node 2 has become responsible for estimating the state and thereby, replaces node 5 that broke down at $t = 250$ seconds. Further, Figure 13 depicts a particular estimation error, for which the estimation error of single node i is defined as $\Delta_i := \big(x - \hat{x}_i\big)^\top \big(x - \hat{x}_i\big)$. More specifically, the figure presents the difference in the estimation error of a network *not* effected by operational event with the estimation error in a network that *is* effected by the previously presented operational event. The reason that the figure depicts the results of node 7, is because this node affected by both events.

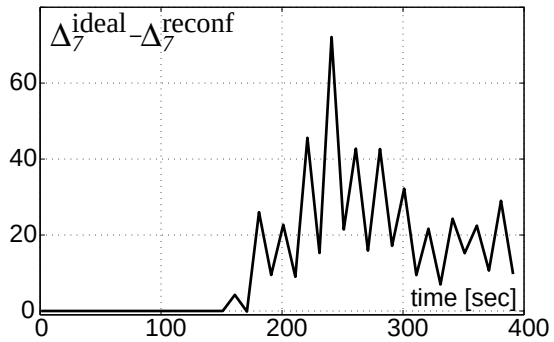


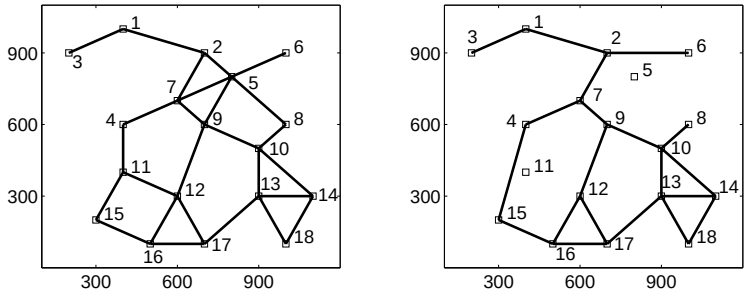
Figure 13: The difference in the estimation error of node 7 for a network that is *not* effected by operational events (Δ_7^{ideal}) with the estimation error in a network that *is* effected by the previously presented operational event (Δ_7^{reconf}).

Before Figure 13 is analyzed, let us denote the hierarchical network in the *presence* of the above mentioned operational events as the reconf-case and the hierarchical network in the *absence* of operational event as the ideal-case. Then the figure indicates that the results of the reconf-case and the ideal-case are equivalent until the two operational events occur, which is expected as both network cases are similar until 150 seconds. After that time, the estimation error of node 7 in the reconf-case increases with respect to the ideal-case. This is due to the fact that nodes 1, 3 and 8 double their local sampling times from

$t = 150$ on and thus, node 7 will receive twice as less measurement information from nodes 1 and 3. This leads to an increase in estimation error of node 7 compared to the ideal-case. Further, note that this error decreases when local measurement information from nodes 1 and 3 is received, i.e., at the time instants 170, 190, 210, ..., 370, 390. At these instances node 7 receives two more local measurements, i.e., y_1 and y_3, which is not the case at the other sampling instants as nodes 1 and 3 doubled their local sampling time. After the second operational event, i.e., node 5 breaks down at $t = 250$, the difference in the estimation error of node 7 for the reconf-case with respect to the ideal-case decreases (on average). This behavior can be explained from the fact that node 2 has become a direct neighbor of node 7, while this node 2 was indirect neighbor via node 5 prior to $t = 250$. Since node 2 is closer to the contaminant source, node 7 obtains an improved estimation result when node 2 is its direct neighbor rather than an indirect one.

7.2 A distributed sensor network

The distributed sensor networks reflects an ad-hoc networked system. This means that the nodes establish a mesh-network-topology, as it is depicted in Figure 14(a). Since there is no hierarchy in this network, each node estimates the local state by performing the distributed KF of (14): the local measurement is processed by f_{KF} to compute a local estimate of the state, which are then shared with neighboring nodes as input to the merging function Ω employing the state fusion method *ellipsoidal intersection* of (17).



(a) Initial topology (b) Topology after 250 seconds

Figure 14: Network topology in a distributed network.

Two events will occur in this network, followed by the corresponding action as it is implemented in the management layer of each node.

- At $t = 150$ seconds nodes 1, 3 and 8 will cross their critical energy level; If *the critical energy level is crossed,* then *lower the node's local sampling time from* 10 *seconds to* 20 *seconds.*

- At $t = 250$ seconds nodes 5 and 11 will break down. Nodes detect that another node has broken down, since no new local estimates are received from that node;
  If *a nodes brakes down and the network has lost its connectivity,* then *establish a network connection with other nodes until this connectivity is re-established. In case this means to increase the communication range to larger distances, decrease the sampling time accordingly.*

Figure 14(a) depicts the network topology prior to the event that nodes 5 and 11 brake down, while Figure 14(b) illustrates the topology and after the event. This figure indicates that the sensor network establishes connectivity, also after the event of a node braking down. However, the nodes 6 and 15 will have to exchange data with nodes that are far away. Therefore, these node will lower their local sampling time to 20 seconds. Further, Figure 15 depicts the same estimation error as Figure 13, only then for a distributed network. This means that the figure presents the difference in the estimation error of a network *not* effected by operational event with the estimation error in a network that *is* effected by the previously presented operational event. The reason that the figure depicts the results of node 7, is because this node affected by both events.

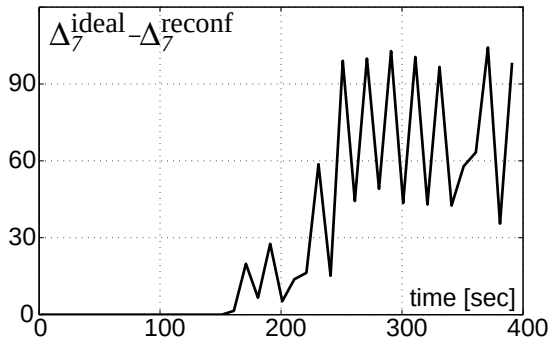


Figure 15: The difference in the estimation error of node 7 for a network that is *not* effected by operational events (Δ_7^{ideal}) with the estimation error in a network that *is* effected by the previously presented operational event (Δ_7^{reconf})

Before Figure 15 is analyzed, let us denote the distributed network in the *presence* of the above mentioned operational events as the reconf-case and the distributed network in the *absence* of operational event as the ideal-case. Then, the figure indicates a similar behavior compared to the hierarchical network that was previously discussed, i.e., the results of the reconf-case and the ideal-case are equivalent until the first operational event occurs, after which the error of the reconf-case increases with respect to the ideal-case. Also, the estimation results of node 7 have an "up-down" type of behavior, which is due to the action undertaken by nodes 1 and 3 to double their local sampling times. As such, node 7 receives an updated estimate from nodes 1 and 3 after every other of its

local sampling instants. The difference between the estimation error of node 7 in the reconf-case increases even further with respect to the ideal-case after the second operational event, i.e., nodes 5 and 11 break down at $t = 250$.

Both the illustrative case studies of a hierarchical and a distributed sensor network indicate that the state is estimated by multiple nodes in the network, even in the presence of unforeseen operational events. As such, adopting a self-organizing method in large-scale and ad-hoc sensor networks improves the robustness of state estimation within the network.

8 Conclusions

Ad-hoc sensor networks typically consist of a large number of vulnerable components connected via unreliable communication links and are sometimes deployed in harsh environment. Therefore, dependability of networked system is a challenging problem. This chapter presented an efficient and cost effective answer to this challenge by employing runtime reconfiguration techniques additional to a particular signal processing method (Kalman filtering). More precisely, a distributed Kalman filtering strategy was presented in a self-organizing sensor networks. This means that each node computes a local estimate of the global state based on its local measurement and on the data exchanged by neighboring nodes. The self-organizing property was implemented via a runtime reconfiguration process, so to have a sensor network that is robust to external and internal system changes, e.g., nodes that are removed or added to an existing network during operation.

Firstly, a brief overview of existing solutions for distributed Kalman filtering was presented. The corresponding algorithms were described with equivalent input and output variables. As a result, nodes could choose which of the algorithms is currently best suitable for estimating the state vector, while taking into account the available communication and computational resources. This further enabled nodes to select what information is to be shared with other nodes, i.e., local measurements or local estimates, and how the received information is merged with the local estimate. Secondly, the system architecture was addressed, such that challenging design issues could be separated from the actual implementation of a (self-organizing) distributed Kalman filter. To that extent, an overview of typical reconfiguration approaches was given with an emphasize on the interactions between the signal processing and hardware/communication aspects of system design. After that, the self-organizing property of the proposed distributed Kalman filter was assessed in a diffusion process for two types of sensor networks, i.e., a hierarchical network and a fully distributed one. In both cases, the network was able to cope with unforeseen events and situations. Or differently, employing runtime reconfiguration in the nodes of the sensor network implements a kind of self-awareness with the ability to create corrective actions and thus assuring that data processing functionalities are never used beyond their scope of validity.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: a survey. *Elsevier, Computer Networks*, 38:393–422, 2002.

[2] B. D. O. Anderson and J. B. Moore. *Optimal filtering*. Prentice-Hall, 1979.

[3] A.T. Bahill and B. Gissing. Re-evaluating systems engineering concepts using systems thinking. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 28(4):516 –527, nov 1998.

[4] Y. Bar-Shalom and L. Campo. The effect of the common process noise on the two-sensor fused-track covariance. *IEEE Trans. on Aerospace and Electronic Systems*, AES-22(6):803–805, 1986.

[5] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri. Distributed Kalman filtering based on consensus strategies. *IEEE journal in selected areas in communications*, 26(4):622–633, 2008.

[6] Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. Dynamic task mapping for mpsocs. *IEEE Des. Test*, 27:26–35, September 2010.

[7] D. W. Casbeer and R. Beard. Distributed information filtering using consensus filters. In *Proc. of the American Control Conf.*, pages 1882 – 1887, St. Louis, USA, 2009.

[8] Betty H. C. Cheng, Rogrio de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Mller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In Betty H. C. Cheng, Rogrio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[9] C. Y. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities and challenges. In *Proc. of the IEEE*, volume 91, pages 1247–1256, 2003.

[10] David Chu, Lucian Popa, Arsalan Tavakoli, Joseph M. Hellerstein, Philip Levis, Scott Shenker, and Ion Stoica. The design and implementation of a declarative sensor network system. In *SenSys'07*, pages 175–188, 2007.

[11] Adriaan de Jong, Matthias Woehrle, and Koen Langendoen. Momi: model-based diagnosis middleware for sensor networks. In *Proceedings of the 4th*

International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks, MidSens '09, pages 19–24, New York, NY, USA, 2009. ACM.

[12] Adam Dunkels, Bjrn Grnvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[13] H.F. Durant-Whyte, B.Y.S. Rao, and H. Hu. Towards a fully decentralized architecture for multi-sensor data fusion. In *1990 IEEE Int. Conf. on Robotics and Automation*, pages 1331–1336, Cincinnati, USA, 1990.

[14] S.C. Felter. An overview of decentralized Kalman filters. In *IEEE 1990 Southern Tier Technical Conf.*, pages 79–87, Birmingham, USA, 1990.

[15] D. Franken and A. Hupper. Improved fast covariance intersection for distributed data fusion. In *Proc. of the 8-th Int. Conf. on Information Fusion*, pages WbA23:1–7, Philidalphia, PA, USA, 2005.

[16] F. Garin and L. Schenato. *Networked Control Systems*, volume 406 of *Lecture Notes in Control and Information Sciences*, chapter A survey on distributed estimation and control applications using linear consensus algorithms, pages 75–107. Springer, 2011.

[17] J.C. Georgas, A. van der Hoek, and R.N. Taylor. Using architectural models to manage and visualize runtime adaptation. *Computer*, 42(10):52 –60, oct. 2009.

[18] M. S. Grewal and A. P. Andrews. *Kalman filtering: theory and practise.* Rootledge, 1993.

[19] U. D. Hanebeck, K. Briechle, and J. Horn. A tight bound for the joint covariance of two random vectors with unknown but constrained cross-correlation. In *Proc. of the IEEE Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 85–90, Baden-Baden, Germany, 2001.

[20] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 642 –649, 2001.

[21] H.R. Hashmipour, S Roy, and A.J. Laub. Decentralized structures for parallel Kalman filtering. *IEEE Trans. on Automatic Control*, 33(1):88–93, 1988.

[22] M.F. Hassan, G. Salut, M.G. Sigh, and A. Titli. A decentralized algorithm for the global Kalman filter. *IEEE Trans. on Automatic Control*, 23(2):262–267, 1978.

[23] Vesa Hasu and Heikki Koivo. Decentralized kalman filter in wireless sensor networks - case studies. In Khaled Elleithy, Tarek Sobh, Ausif Mahmood, Magued Iskander, and Mohammad (eds) Karim, editors, *Advances in Computer, Information, and Systems Sciences, and Engineering: Proc. of IETA 2005, TeNe 2005 and EIAE 2005*, pages 61–68, The Netherlands, 2006. Springer.

[24] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. on Automatic Control*, 48(6):988–1001, 2003.

[25] S. J. Julier and J. K. Uhlmann. A non-divergent estimation algorithm in the presence of uknown correlations. In *Proc. of the American Control Conf.*, pages 2369–2373, Piscataway, USA, 1997.

[26] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Prceedings of AeroSense: The 11-th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pages 182–193, Orlando, Florida, 1997.

[27] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *Proc. of the 5-th ACM/IEEE Int. Conf. on Mobile Computing and Networking*, pages 271 – 278, Seattle, USA, 1999.

[28] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Seamless service composition (sesco) in pervasive environments. In *Proceedings of the First ACM International Workshop on Multimedia Service Composition*, MSC '05, pages 11–20, New York, NY, USA, 2005. ACM.

[29] R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. of the ASME Journal of Basic Engineering*, 82(D):35–42, 1960.

[30] G. Karsai, F. Massacci, L.J. Osterweil, and I. Schieferdecker. Evolving embedded systems. *Computer*, 43(5):34 –40, may 2010.

[31] G. Karsai and J. Sztipanovits. A model-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE*, 14(3):46 –53, may/jun 1999.

[32] U.A. Khan and J.M.F. Moura. Distributed Kalman filters in sensor networks: Bipartite fusion graphs. In *IEEE 14-th Workshop on Statistical Signal Processing*, pages 700–704, Madison, USA, 2007.

[33] S. Kirti and A. Scaglione. Scalable distributed Kalman filtering through consensus. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pages 2725 – 2728, Las Vegas, USA, 2008.

[34] Sachin Kogekar, Sandeep Neema, Brandon Eames, Xenofon Koutsoukos, Akos Ledeczi, and Miklos Maroti. Constraint-guided dynamic reconfiguration in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04, pages 379–387, New York, NY, USA, 2004. ACM.

[35] Xenofon D Koutsoukos, Manish Kushwaha, Isaac Amundson, Sandeep Neema, and Janos Sztipanovits. *OASiS: A service-oriented architecture for ambient-aware sensor networks*, volume 4888 LNCS, pages 125–149. 2007.

[36] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[37] F. L. Lewis. Wireless sensor networks. In *Smart environments: Technologies, protocols, applications (Chapter 2)*, Lecture Notes in Computer Science. Wiley, New York, 2005.

[38] Pragnesh J. Modi, Hyuckchul Jung, Milind Tambe, Wei-Min Shen, and Shriniwas Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, CP '01, pages 685–700, London, UK, UK, 2001. Springer-Verlag.

[39] Brice Morin, Olivier Barais, Jean-Marc Jzquel, Franck Fleurey, and Arnor Solberg. Models at runtime to support dynamic adaptation. *IEEE Computer*, pages 46–53, October 2009.

[40] A.G.O. Mutambara and Duranth-Whyte H.F. Fully decentralized estimation and control for a modular wheeled mobile robot. *Int. Journal of Robotic Research*, 19(6):582–596, 2000.

[41] W. Niehsen. Information fusion based on fast covariance intersection filtering. In *Proc. of the 5-th Int. Conf. on Information Fusion*, pages 901–905, Annapolis, USA, 2002.

[42] R. Olfati-Saber. Distributed Kalman filtering for sensor networks. In *Proc. of the 46-th IEEE Conf. on Decision and Control*, pages 5492 – 5498, New Orleans, USA, 2007.

[43] R. Olfati-Saber. Kalman-Consensus filter: Optimality, stability, and performance. In *Proc. of the 48-th IEEE Conf. on Decision and Control*, pages 7036 – 7042, Shanghai, China, 2009.

[44] Z. Papp, J. Sijs, and M. Lagioia. Sensor network for real-time vehicle tracking on road networks. In *Proc. of the 5-th Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing*, pages 85 – 90, Melbourne, Australia, 2009.

[45] W. Ren, R. Beard, and D. Kingston. Multi-agent Kalman consensus with relative uncertainty. In *Proc. of the American Control Conf.*, pages 1865 – 1870, Portland, USA, 2005.

[46] A. Ribeiro, G. B. Giannakis, and S. I. Roumeliotis. SOI-KF: Distributed Kalman filtering with low-cost communications using the sign of innovations. *IEEE Trans. on Signal Processing*, 54(12):4782 – 4795, 2006.

[47] A. Ribeiro, I. D. Schizas, S. I. Roumeliotis, and G. B. Giannakis. Kalman filtering in wireless sensor networks: Reducing communication cost in state-estimation problems. *IEEE Control Systems Magazine*, 4:66–86, 2010.

[48] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman filter: Particle filter for tracking applications*. 2002.

[49] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[50] C. E. Shannon and W. Weaver. *The mathematical theory of communication*. The University of Illinois Press, Urbana, Illinois, 1949.

[51] J. Sijs and M. Lazar. Distributed Kalman filtering with global covariance. In *Proc. of the American Control Conf.*, pages 4840 – 4845, San Francisco, USA, 2011.

[52] J. Sijs and M. Lazar. State fusion with unknown correlation: Ellipsoidal intersection. *Automatica (*in press*)*, 2012.

[53] J. Sijs, M. Lazar, P.P.J. Van de Bosch, and Z. Papp. An overview of non-centralized Kalman filters. In *Proc. of the IEEE Int. Conf. on Control Applications*, pages 739–744, San Antonio, USA, 2008.

[54] A. Speranzon, C. Fischione, K.H. Johansson, and A. Sangiovanni-Vincentelli. A distributed minimum variance estimator for sensor networks. *IEEE Journal on Selected Areas in Communications*, 26(4):609–621, 2008.

[55] J.L. Speyer. Computation and transmission requirements for a decentralized Linear-Quadratic-Gaussian control problem. *IEEE Trans. on Automatic Control*, 24(2):266–269, 1979.

[56] Thilo Streichert, Dirk Koch, Christian Haubelt, and Jrgen Teich. Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems. *EURASIP JOURNAL ON EMBEDDED SYSTEMS*, page 15, 2006.

[57] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *ACM Communications*, 47:34–40, 2004.

[58] A. Tahbaz Salehi and A. Jadbabaie. Consensus over ergodic stationary graph processes. *IEEE Trans. on Automatic Control*, 55:225–230, 2010.

[59] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[60] Yafeng Wu, Krasimira Kapitanova, Jingyuan Li, John A. Stankovic, Sang H. Son, and Kamin Whitehouse. Run time assurance of application-level requirements in wireless sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 197–208, New York, NY, USA, 2010. ACM.

[61] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Systems and Control Letters*, 53(1):65–78, 2004.

[62] L. Xiao, S. Boyd, and S. Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Proc. of the 4-th Int. Symp. on Information processing in sensor networks*, pages 63 – 70, Los Angelos, California, USA, 2005.

[63] Y. Zhuo and J. Li. Data fusion of unknown correlations using internal ellipsoidal approximations. In *Proc. of the 17-th IFAC World Congress*, pages 2856–2860, Seoul, Korea, 2008.