A fault tolerant architecture for data fusion: A real application of Kalman filters for mobile robot localization

Kaci Bader, Benjamin Lussier, Walter Schön

A Fault Tolerant Architecture For Data Fusion: a real Application on Kalman Filters for Mobile Robot Localization

Kaci Bader Benjamin Lussier
Walter Schön
Sorbonne Universités, Université de Technologie de Compiégne ,
CNRS, UMR 7253, Heudiasyc
CS 60 319, 60203 Compiégne, France

Abstract

Multi-sensor perception have an important role in robotics and autonomous systems, as inputs for critical functions such as obstacle detection, localization, etc. This Multi-sensor perception begins to appear in critical applications, such as drones and ADAS (Advanced Driver Assistance Systems). However such complex systems are difficult to validate entirely. In this paper we study these systems under an alternative dependability method: fault tolerance. We propose an approach to tolerate faults in multi-sensor data fusion based on the more classical method of duplication-comparison, and offering detection and recovery services. We detail an example implementation using Kalman filters data fusion for mobile robot localization. We demonstrate its effectiveness in this case study using real data and fault injection.

Keywords: Data fusion, Multi-sensor Perception, Dependability, Fault tolerance

1 Introduction

Perception is a fundamental input to any robotic system. However, data perceived by such systems are often complex and subject to significant uncertainties and inaccuracies.

To overcome these problems the multi-sensor approach resorts to data from multiple and complementary sensors, and uses their redundancy to filter noises, eliminate some aberrant data, increase the precision of perception, and extract complex knowledge about the environment. But multiplying sensors and the underlying data fusion algorithms consequently increases the risks of hardware and software faults. Moreover, the validation of this approach encounters two major problems:

- First, fusion algorithms are part of the declarative programming paradigm, which consists in describing a problem and the system in way to reach a decision. This paradigm is often used in artificial intelligence applications such as planning, but is harder to understand and validate than imperative programming (which is the description of successive steps to execute). Generally, declarative programming approaches are nowadays forbidden in critical system standard. For example, the EN 50l28[1] railway standards states that artificial intelligence software are not recommended in critical applications, whereas procedural programming (that is part of imperative paradigm) is highly recommended. For this reason the behavior of fusion algorithms is hard to predict, making them difficult to validate by formal approaches, such as formal model and proof checking.

- Second, the open environment which complex robotic systems are confronted to also generates a near-infinite execution context. In validation, execution context refers to the different possible situations that the system may be confronted to. In open environment, this context is deemed near infinite because obstacles may appear at any moment in many different ways, lighting and wind conditions may vary, etc. As such, validation of automobile systems require thousand and thousand of hours passed driving on roads, with no certainty that all possible situations have been encountered. For this reason testing is a difficult, long and costly operation.

An alternative to this validation is the development of fault tolerance mechanisms: since it is difficult to remove all faults in the system, we will seek to limit their impact on its functions.

In this paper, we thus focus on the issues of fault tolerance in multi-sensor perception systems, which is a fundamental input for any autonomous robotic system. In our approach we propose an architecture based on duplication / comparison to detect and diagnose faults in a data fusion mechanisms. We illustrate it by an example application for mobile robot localization using Kalman filter data fusion, and we detail its fault tolerance services such as faults detection and system recovery suitable for multi-sensor perception to ensure their reliability.

This paper is organized as follows: after this introduction, section 2 summarizes concepts and related works on fault tolerance in data fusion. Section 3 describes our proposed architecture, and details its fault tolerance services such as fault detection and system recovery. Section 4 describes an intelligent vehicle localization application that implements our duplication-comparison approach, and section 5 presents our experimental study validating our approach using real data and fault injection. Finally the paper ends with conclusions and perspectives for future works.

2 Concepts and related work

This paper studies two different domains, each using specific terminologies and concepts. Dependability centers on the notion of fault, as a potential cause of system failures, and offers various means, including fault tolerance, to deal with it. Data fusion intends to merge different sensor outputs to better perceive the system's environment. Both fault tolerance and data fusion use redundancy, but the former tries to detect and tolerate internal faults, while the later focuses on aleas in an open and shifting environment. This section introduces the concepts of both these domains. It first describes the concepts related to dependability, that encompasses fault tolerance, then presents Kalman filters data fusion. It finally proposes a state of the art regarding fault tolerance mechanisms in data fusion.

2.1 Dependability

A system's dependability is its ability to deliver a service that can justifiably be trusted[2][3]. This notion encompasses three different concepts: *(a) its attributes,* the expected properties of the system, *(b) its threats,* unacceptable behaviors of the system that are causes or consequences of a lack of dependability, *(c) its means,* methods that allow a system to dependably perform its function (that is by placing a justified confidence in the service it delivers). For more information on general concepts in dependability, the reader may refer to [2] and [3].

- **The attributes** of dependability are properties that a system must verify. Six main attributes are defined: *(a) Availability:* readiness for correct service. *(b) Reliability:* continuity of correct service. A system is reliable if it delivers continuously and correctly its service for a specified period. *(c) Maintainability:* ability to undergo modifications and repairs. It characterizes the ability of a system to be returned quickly from a failure to an operational state in which it can perform its required functions. *(d) Safety:* absence of catastrophic consequences on the user(s), the system, and the environment. *(e) Confidentiality:* absence of unauthorized disclosure of information. *(f) Integrity:* absence of improper system alterations.

  In this work we particularly focus on two attributes of dependability: *safety* and *reliability*. Note that trying to achieve both of these attributes may be contradictory. Indeed, for an autonomous vehicle in a dangerous situation, safety may require to stop and assess the situation, while reliability would ask to continue the performed service.

- **Threats** are undesirable behaviors of the system. They are of three types: *faults*, *errors* and *failures*. These *threats* are linked by a causal relationship: the fault is the adjudged or hypothesized cause of an error, while the error is likely to result in a failure as shown in Figure 1. In this figure the decisional system A takes as input the output of the perception system B,

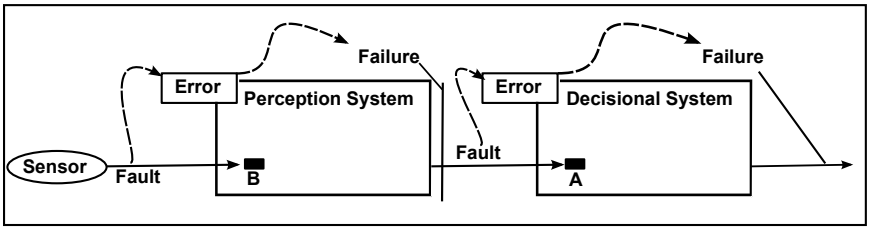

Figure 1: Causal chain

so the correct service of A depends on the correct service of B. For A, the failure of B is considered as an external fault, which in turn can cause an internal error in A, then finally the failure of A.

Perception in a robotic system being the basis upon which the system takes action, it is vital to avoid incorrect behavior, as failures could lead to the failure of the system as a whole.

- **Means** have been designed to counter the previous threats. They are classified into four types: *(a) Fault prevention,* that is how to prevent the occurrence or introduction of faults in the system, *(b) Fault tolerance,* that is how to allow a system to properly fulfill its function in the presence of faults, *(c) Fault removal,* that is how to reduce the presence (both number and severity) of faults in a system, *(d) Fault forecasting,* that is how to estimate the presence, the creation and consequences of faults in a system.

2.1.1 Fault tolerance

One of the four means of dependability, *fault tolerance* aims to ensure proper delivery of a system's services despite the presence of faults. Fault tolerance is principally carried out via *error detection* and *system recovery.*

- **Error detection** is a prerequisite for the implementation of fault tolerant solutions. It aims to detect the erroneous state of the system before it propagates to system failures. There are three main methods for error detection.

  1. *Duplication-comparison:* it consists to compare the results from at least two redundant units that are independent to the faults to tolerate and provide the same service.
  2. *Temporal watchdog:* it consists to check a temporal error in a system by controlling its response time which should not exceed a maximum value (time out).
  3. *Likelihood checks:* it seeks to detect errors by checking against aberrant values in the system state.

- **System recovery** allows the substitution of an error-free state to the erroneous state. This substitution can be made in three ways:

    1. *Recovery:* the system is restored to a correct state that occurred before the occurrence of the error. This correct state must have been previously saved by the system.
    2. *Pursuit:* a new state is found from which the system can function properly (eventually in a degraded mode).
    3. *Error compensation:* the erroneous state contains enough redundancy to allow its transformation into a correct state.

2.2 Data fusion

Information fusion consists in combining information from multiple sources to improve decision making[4]. Data fusion systems are widely used in various fields especially robotics for different applications, such as navigation, obstacle detection, object tracking, etc.

In localization applications, the most used data fusion approach is the Kalman filtering. In the following section we present the main concepts of the Kalman filter data fusion algorithm, which we use in our application to locate a mobile robot in its environment.

2.2.1 Kalman filters data fusion algorithm

Kalman filter in data fusion consist in estimating the unknown state of the system, and systematically correcting this estimation through observation. This is achieved through sets of sequential calculations to provide a best estimate of the system's state variables, with at each step a correction proportional to the error between the current prediction and sensors outputs. This method has been widely applied in many robotic applications [5] [6] [7] [8] (such as autonomous navigation, target tracking and localization).

We consider a discrete-time *system model* (equation 1) and a discrete-time *observation model* (equation 2) :

$$X_k = AX_{k-1} + BU_{k-1} + w_k \tag{1}$$

where X_k is the state vector, U_{k-1} the command vector, w_k a zero mean white gaussian noise with an assumed known covariance matrix Q_k, A the known state transition matrix, and B the known control transition matrix.

$$Z_k = HX_k + v_k \tag{2}$$

where Z_k is the observation vector of the sensor, and v_k the white gaussian observation noise for the sensor with zero mean and an assumed known covariance matrix R_k.
A multi-sensor Kalman filter for data fusion can be computed using these two models as a prediction step (equation 3) and a correction step (equation 4).

1. **Prediction step**: The corrected estimated state of the previous time step $\hat{X}_{k-1}^{+}$, and the command input U_{k-1} are used to produce an estimate of the current state $\hat{X}_{k}^{-}$ and its predicted uncertainty P_{k}^{-}:

$$\begin{cases} \hat{X}_{k}^{-} = A\hat{X}_{k-1}^{+} + BU_{k-1} \\ P_{k}^{-} = AP_{k-1}^{+}A^{T} + Q \end{cases} \tag{3}$$

2. **Correction step:** Once the observation Z_k is available, the predicted state $\hat{X}_{k}^{-}$ can be corrected by introducing $\hat{S}_k$ (and its associated covariance S_k) weighted by the Kalman filter gain K_k, thus obtaining the more accurate estimation state $\hat{X}_{k}^{+}$ and its covariance matrix P_{k}^{+}.

$$\begin{cases} \hat{S}_k = Z_k - H\hat{X}_{k}^{-} \\ S_k = HP_{k}^{-}H^{T} + R \\ K_k = P_{k}^{-}H^{T}S_{k}^{-1} \\ \hat{X}_{k}^{+} = \hat{X}_{k}^{-} + K_k\hat{S}_k \\ P_{k}^{+} = (Id - K_kH)P_{k}^{-} \end{cases} \tag{4}$$

Note that $\hat{S}_k$ is the residual of the Kalman filter, thereafter known as *Res*. Also note that, in our opinion, A, B, Q and R are particularly susceptible to design faults, as Q and R are generally determined empirically after some experiments, and A and B are part of a model that may be hard to validate exhaustively.

2.3 Fault tolerance in data fusion

To our knowledge, few studies exist on fault tolerance in data fusion. The approaches we have found in the literature mainly use fault tolerance by duplication / comparison, that we have categorized in two different classes: duplication based on an analytical model of the fusion process, and duplication based on hardware redundancies.

An example of duplication based on an analytical model of the fusion process is [9], where a Kalman filter is used as a mathematical model to propose an hybrid multi sensor data fusion architecture integrating Kalman filtering and fuzzy logic techniques. The main characteristics of this architecture is its tolerance to permanent and transient sensor faults. This tolerance is obtained by using the optimality of the Kalman filter and the capability of fuzzy systems to deal with imprecise information using fuzzy sets and common sense rules. In this architecture the transient faults are detected using a residual technique, and permanent faults are detected by a voting method [10]. In [11] the authors propose an architecture for fault-tolerant sensors. This approach uses an abstract sensor system [12] which is embedded in a virtual sensor to improve the sensor characteristics in the presence of noise and failures. By using a mathematical model to evaluate redundant sensor data, this approach achieves a more reliable position estimation. [13] proposes a state of the art concerning fault detection and isolation techniques based on analytical redundancy. These techniques are based on the generation of residuals using a mathematical model, and their

evaluation to detect and isolate faults in a dynamic system. It details the different residual generation techniques, including those based on Kalman filter, Thereafter a decision logic generally based on thresholding technique is applied to evaluate the residuals to detect and isolate faults. The authors noted that when the analytical model is uncertain, these techniques become problematic. In this case, they suggest to use knowledge based methods.

The approaches based on hardware redundancies use the analysis of some internal parameters (such as the conflict) to ensure fault tolerance. In [14] the authors discuss the detection of a malfunction by the temporal analysis of the conflict resulting from the fusion of data sources, based on the Transferable Belief Model (TBM) of Smets [15] [16]. The authors propose to merge data sources pairwise and take into account the evolution of conflict (as defined by Smets) resulting from the fusion. A source is considered defective if its conflict with others are high. After recognition of the failed source the authors suggest to weaken its influence on the final decision, by associating reliability indices and using the discounting technique proposed by Shafer [17]. This weakens the source's masses and increases their discounting factor without discarding conflicting sources completely. In [18], an algorithm to detect a defective source by analyzing the reliability of each source is proposed. To each source are associated two reliabilities: static reliability represents the quality of the source whereas dynamic reliability indicates conflict between this source and the others. Global reliability of the source is then based on the two previous reliabilities. This global reliability serves as a discounting factor to weight belief masses given by the sources before they are combined. Assuming that the conflict comes from a defective source, the authors analyze the discounting factor (global reliability of sources) to detect the erroneous source using a thresholding method. In [19] a similar approach was proposed by the same authors in the specific context of the possibility theory. In [20] D-S theory is used to resolve the interval estimation problem in the context of wireless sensor networks. In this method the data fusion merges N intervals to estimate the smallest one tolerating f hardware faults. In [21] an adaptive technique was used to weight the sensor outputs in multisensor data fusion process. This technique uses the standard deviation of each sensor (estimated by statistics and a temporal factor related to previous data) to calculate a discounting factor of the sensor outputs and uses it in data fusion process to ensure fault tolerance and weaken the negative effects of aging hardware. [22] presents approaches of data fusion for inertial network systems. In these systems, the authors use redundant nodes that are structured hierarchically. Most nodes play the role of slaves while one plays the role of master, which will realize the global fusion. Different transformation matrices (rotation and translation) are developed to ensure the alignment of measures. A Kalman filter is used as fusion algorithm. The authors use the IMU redundancy to ensure fault masking and to have more precise outputs. In [23] Peng et al propose a diagnosis method in analog circuits combining neural network and evidential reasoning. This method follows two steps. First, a preliminary diagnosis is implemented using a neural network on a preselected set of test signals to construct a mapping between sensors output and erroneous states.

Second, an evidential reasoning using D-S theory is employed to merge different possible fault patterns to get the correct one. In our opinion such a diagnosis could not be used on critical systems, because the mapping in the first step is based on training, that could be not exhaustive or deciding wrongly. Moreover remains the problem of the trustworthiness of the data fusion mechanism, that we explicit in the following paragraph. Other similar examples of fault tolerance in data fusion are mentioned in section 3.C of [24], an article that reviews fault tolerance practicies in wheeled robotic systems.

In our opinion, those papers focus only on hardware fault tolerance. Moreover, they detect and tolerate faults related to physical sensors using data fusion processes that are difficult to design and validate, thus being rather untrustworthy. Fault tolerance through data fusion can only be done to our knowledge by trying to tolerate software faults in the data fusion process, or guaranteeing with formal methods that this data fusion process is sound. We only found two papers in the literature addressing this problem, both using formal analysis to raise the trustworthiness of data fusion mechanisms. In [25] where an approach of data fusion based on Covariance Intersection CI and Covariance Union CU is proposed to ensure consistency and fault tolerance in the management of information in distributed networks. CI based solution ensures that the fusion result of consistent estimates is consistent in any situation. The CU completes the CI by ensuring that the fusion result of inconsistent estimates remains consistent and thus tolerating sensor faults through masking. Such mechanism is validated by a formal analysis of its properties. In [26] another approach for binary data fusion from distributed sensors is proposed. It provides fault tolerance by analyzing formally sensor confidence levels associated with each decision outcome, which serves to enhance performance by improving robustness of the Boolean Fusion and eliminate its weaknesses in order to reduce or even eliminate false alarms.

In our opinion a significant flaw in those different techniques is that they focus only on hardware faults, relying on the fusion mechanisms to detect and tolerate the errors. But fusion mechanisms are hard to design and validate, often incorporating values that have been determined empirically (such as gains) or models that are difficult to validate formally. Software faults in data fusion are thus for us a big concern in data fusion, and the proposed architecture aims to tolerate both hardware and software faults.

3 Fault tolerance with Duplication-Comparison

We propose in this section an approach that offers detection and recovery services for data fusion using duplication-comparison. Figure 2 details this generic architecture. It implements two parallel and independent branches, each running a data fusion mechanism using redundant or diversified sensors to perceive a system state and its environment.

Under a single fault hypothesis (no more than one activated fault is present at the same time in the system), this architecture can tolerate one hardware

fault in the sensor blocks, or detect one software fault in the data fusion blocks. These services could be extended to multiple faults by increasing the level of hardware and software redundancy. Such redundancy can also be exploited to tolerate the detected software faults using diversified data fusion mechanisms and the voting method [27].

The general principle of our architecture is presented in Figure 2 and consists of comparing the results of the two fusion blocks to detect a significant deviation between the data fusion outputs. Such difference indicates the presence of an error. The sensor outputs and residual values from the fusion are then used to determine whether this error is due to a hardware fault in the sensors or a software fault in the fusion algorithms, then to diagnose the detected error and determine the correct output of the perception system.

Details of our approach are presented in the following subsections. An application example for vehicle localization is presented in section 4.

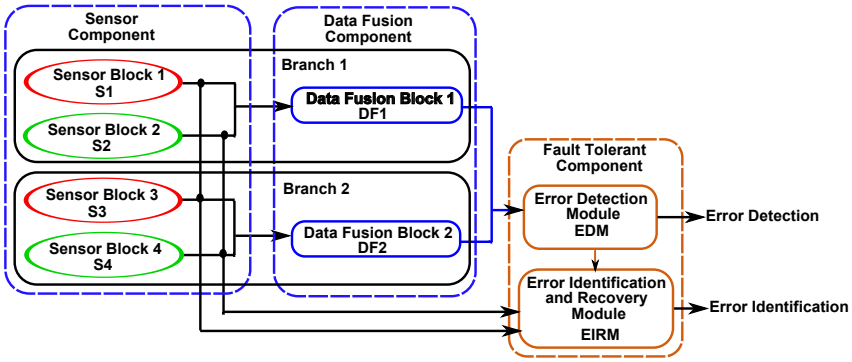

Figure 2: Duplication-Comparison Architecture for fault tolerance in multi-sensor perception

3.1 Fault detection and recovery services

Our architecture uses two parallel branches, each running one data fusion block with two sensor blocks : the first branch (S_1, S_2, DF_1) combines the outputs of sensor blocks S_1 and S_2 in the fusion block DF_1, and the second branch (S_3, S_4, DF_2) combines the outputs of the sensor blocks S_3 and S_4 in the fusion block DF_2. The outputs of each duplicated component are compared with its diversified redundant one : for example, in Figure 2 the output of DF_1 is compared with the output of DF_2, the output of S_1 is compared with the output of S_3, and the output of S_2 is compared with the output of S_4. The outputs of fusion blocks are compared to detect an eventual error in the system, and the outputs of the sensor blocks are used to diagnose the detected error and determine the correct output of the perception system. Our architecture is thus composed of two modules:

1. *Error Detection Module (EDM)* : it compares the outputs of DF_1 and DF_2 to detect possible faults in the system. When there is no activated fault in the system, the fusion block outputs will be similar. The outputs of DF_1 and DF_2 will be significantly different when a relevant error is present. The algorithm 1 describes the process of this module.

Algorithm 1: Error Detection Algorithm

Data: DF_1,DF_2 : fusion blocks outputs
Result: E : Error indicator
Calculate Δ_{DF_1,DF_2}
if $\Delta_{DF_1,DF_2} > Thrs_{Det}$ **then**
    E $\leftarrow$ {YES};
    `/* Compare sensor blocks outputs and the residual values for diagnosis and recovery */`
else
    E $\leftarrow$ {NO};
end

2. *Error Identification and Recovery Module (EIRM)* : it carries out the diagnosis of a previously detected fault, and ensure the system recovery. It is composed of two steps:

   - *Sensors blocks comparison:* it identifies the type of the fault, *Hardware* in the sensor blocks or *software* in the fusion blocks. This is achieved trough comparison of different sensor block's outputs (S_1 with S_3 on one hand, and S_2 with S_4 on the other hand).
     - If one sensors blocks output deviates from its dual, then a hardware fault is diagnosed on one of these two sensors, as an erroneous sensor will have a different output than a correct one. The faulty sensor will be identified in the *residual comparison step*.
     - If the sensors outputs are similar, then a software fault is diagnosed. With no more redundancy, we can do nothing but put the erroneous system in a safe state. However another diversified data fusion block DF_3 using two outputs amongst S_1, S_2, S_3 and S_4 can be implemented to diagnose the faulty data fusion block, and recovers the system from the detected error [27].
   - *Residual comparison:* it identifies the faulty sensor in the case of an hardware fault. As we have already checked in the *sensors outputs comparison* that the current discrepancy between the two branches is due to the sensors and not the fusion mechanisms, we trust here the data fusion algorithms and compare its residual outputs to identify the erroneous branch. The highest residual value indicates conflict in the data sources, and thus the faulty data fusion block.

Knowing now the type of the faulty sensor and its erroneous branch, the system has identified the erroneous sensor and can recover by using output values of the error-free branch.

3.2 Qualitative Analysis

The proposed architecture provides the following fault tolerance services:

- *Hardware fault detection and recovery*: it detects and recovers from one hardware error related to the perception sensors block (S_1, S_2, S_3 or S_4).
- *Software fault detection*: it detects one software fault related to data fusion algorithms (DF_1, DF_2). Such fault can be tolerated by a third diversified block and a voting method.

To ensure these services, our architecture requires some comparison between some sensor blocks:

- the comparison can be done on the sensors output or on processed data. For example in our case study (section4), a derivative of rear wheels speed is computed before comparing it to the acceleration of an INS (Inertial Navigation System).
- the comparison can be done between two sensor blocks, each encompassing one or more sensors, and where all used outputs of one block have a counterpart (raw or processed) in the second block.

4 Application: Kalman filters data fusion for vehicle localization

In this section we present an application of the generic architecture detailed in Section 3. We apply this architecture to a vehicle component localization using Kalman filters for data fusion. To implement our architecture two filters are used to fuse the input data from various sensor blocks. The first Kalman filter uses data of an accelerometer, a gyrometer and an odometer. The second one uses data of another odometer and a GPS. From these two branches, the EDM and EIRM modules described in section 3.1 can be implemented. Details of this implementation are shown in figure 3.

4.1 Tricycle model for front and rear wheel odometry

Computations of the vehicle position from odometry sensors (F-odo and R-WSS in Figure 3) are based on the tricycle model [28] presented in Figure 4. In this model, the vehicle is composed of two fixed rear wheels on the same axis and an orientable centered front wheel located on the longitudinal axis of the robot. The motion of the vehicle can be inferred from two variables: the longitudinal velocity and the direction of the steerable wheel. We establish the

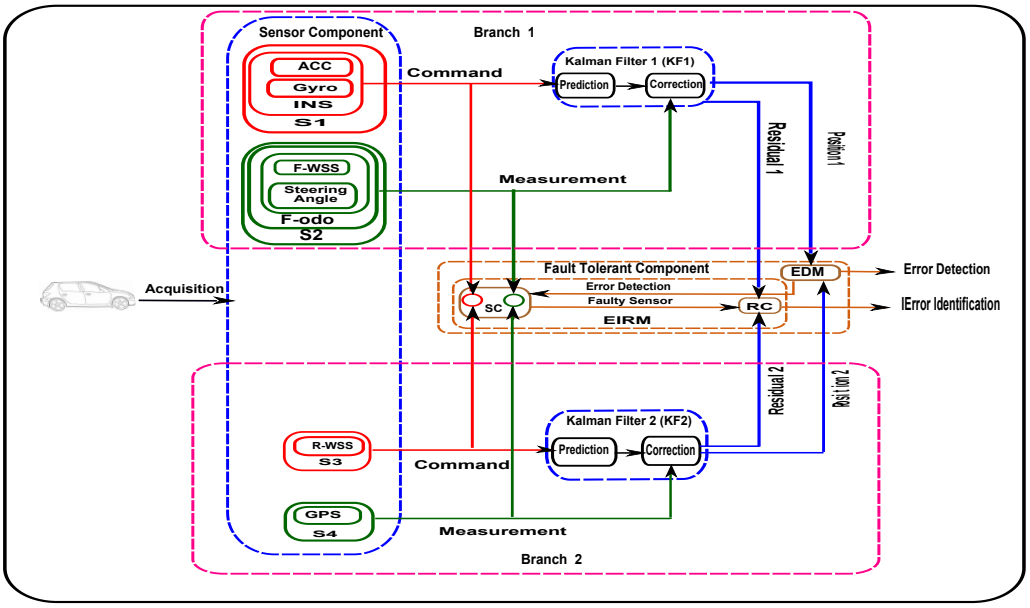

Figure 3: Architecture implementation for robot localization using Kalman filter

kinematic equations that we use in the prediction of KF_1 and the correction of KF_2 respectively in sections 4.3 and 4.4.

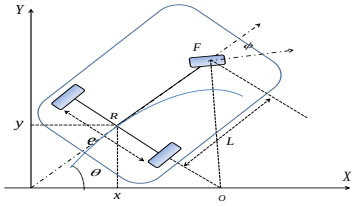

Figure 4: Tricycle model

4.2 Terminology and notations

We introduce in Table 1 some terminologies and notations used in the following section.

Notation	Description
$X_{KF_{i_k}}(x_{KF_{i_k}}, y_{KF_{i_k}}, \theta_{KF_{i_k}})$ With $i \in \{1,2\}$	State of the Kalman filter i at time k
$\Delta t_k = t_k - t_{k-1}$	Timestamps
$Z_{KF_{i_k}}$	measurement vector used in observation model of Kalman filter i a time k
$w_{KF_{i_k}}$	noise vector of the system model of Kalman filter i at time k
$v_{KF_{i_k}}$	noise vector of the observation model of Kalman filter i at time k
Q_{KF_i}	covariance matrix of the system noise w_{KF_i}
R_{KF_i}	covariance matrix of the measurement noise v_{KF_i}

Table 1: Notations

4.3 Kalman filter for INS-Front Odometer data fusion

The first Kalman filter applied to the first branch of our architecture (KF_1) consists of the coupling an inertial system and a front wheel odometry system: We use an INS (accelerometer and gyrometer) to predict the system state, and the front wheel odometry to correct it. The state variables are the positions of the vehicle (x, y), its orientation (θ) and its longitudinal speed (V). Implementation details of this configuration are as follows:

- *System model*: we use equation (5) as a system model in KF_1, taking as input $U_{INS}(\gamma_{INS}, \omega_{INS})$, the longitudinal acceleration measured by the accelerometer and the angular speed measured by the gyrometer.

$$\underbrace{\begin{bmatrix} x_{KF_1} \\ y_{KF_1} \\ \theta_{KF_1} \\ V_{KF_1} \end{bmatrix}_k}_{X_{KF_1 k}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \Delta t_k cos(\theta_{KF_1 k-1}) \\ 0 & 1 & 0 & \Delta t_k sin(\theta_{KF_1 k-1}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} x_{KF_1} \\ y_{KF_1} \\ \theta_{KF_1} \\ V_{KF_1} \end{bmatrix}_{k-1}}_{X_{KF_1 k-1}} + \underbrace{\begin{bmatrix} \frac{1}{2}\Delta t_k^2 & 0 \\ \frac{1}{2}\Delta t_k^2 & 0 \\ 0 & 1 \\ \Delta t_k & 0 \end{bmatrix}}_{B} \cdot \underbrace{\begin{bmatrix} \gamma_{INS} \\ \omega_{INS} \end{bmatrix}_{k-1}}_{U_{INS_{k-1}}} + w_{KF_1 k} \tag{5}$$

- *Observation model*: The observation model is based on front wheel odometry sensors and a tricycle model [28]. It is implemented as follows (equation 6):

$$Z_{KF_1 k} = \begin{bmatrix} x_{Fodo} \\ y_{Fodo} \\ \theta_{Fodo} \\ V_F \end{bmatrix}_k = H_{KF_1} . X_{KF_1 k} + v_{KF_1 k} \tag{6}$$

where $(x_{Fodo}, y_{Fodo}, \theta_{Fodo})$ is the vehicle pose estimated by the front odometry system, and V_F is the longitudinal speed given by the front encoder. The matrix H_{KF_1} that defines the observation model is the identity matrix because the measurements are the same than the system state variables, and we have thus decided not to make it appear in equations derived from 6.

4.4 Kalman filter for Rear odometry- GPS data fusion

The Kalman filter used in the second branch of our application implements a tricycle kinematic model [28] using the rear wheel odometry coupled with a GPS sensor.

- *System model*: the system model equations are as follows (equation (7)):

$$\underbrace{\begin{bmatrix} x_{KF_2} \\ y_{KF_2} \\ \theta_{KF_2} \\ V_{KF_2} \end{bmatrix}_k}_{X_{KF_2 k}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} x_{KF_2} \\ y_{KF_2} \\ \theta_{KF_2} \\ V_{KF_2} \end{bmatrix}_{k-1}}_{X_{KF2_{k-1}}} + \underbrace{\begin{bmatrix} \Delta t_k cos(\theta_k) & 0 \\ \Delta t_k sin(\theta_k) & 0 \\ 0 & \Delta t_k \\ 1 & 0 \end{bmatrix}}_{B} \cdot \underbrace{\begin{bmatrix} V_R \\ \omega_R \end{bmatrix}_{k-1}}_{U_{R_{k-1}}} + w_{KF_2 k} \tag{7}$$

where V_R and ω_R are respectively the rear wheels linear speed and their angular speed. These values are calculated from the rear wheels encoders:

- V_R is the mean of the rear left and rear right wheel speeds
- ω_R is the elementary rotations of the two rear wheels

$$\begin{cases} V_R = \frac{V_{RR}+V_{RL}}{2} \\ \omega_R = \frac{V_{RR}-V_{RL}}{e} \end{cases} \tag{8}$$

where V_{RR} and V_{RL} are respectively the rear right wheel speed and left wheel speed, and e is the entrax of the vehicle (the distance between the two wheels on the axle).

- *Observation model*: The GPS observations are defined in the following way:

$$Z_{KF2_k} = \begin{bmatrix} x_{GPS} \\ y_{GPS} \end{bmatrix}_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{H_{KF_2}} . X_{KF2_k} + v_{KF2_k} \tag{9}$$

Note that both process / observation models are nonlinear, so an extended Kalman filter is indeed much more appropriate for such nonlinear system. We used a simple Kalman Filter to facilitate the development process, and used a linearization interpolation to synchronize the different sensors data.

4.5 Fault tolerance services

As we see in figure 3, our architecture implements two independent positioning systems (both Kalman filters) using diversified sensors, thereby providing a software and hardware redundancy.

Our goal is to exploit the hardware redundancy in sensors and software diversification in the Kalman filter algorithms to provide fault tolerance services using the duplication- comparison technique.

In the following sections, we detail these detection and recovery services by explaining how the *EDM* and *EIRM* are implemented.

4.5.1 Error detection service:

To detect possible errors in the system, we compare the estimated positions of the two implemented Kalman filters. For this we calculate the euclidean distance $\Delta Pos_{(KF_1,KF_2)}$ (equation 10) between the output Pos_{KF_1} of the first filter and the output Pos_{KF_2} of the second filter. Then we compare this distance to a specific detection threshold $Thrs_{Det}$. If the distance $\Delta Pos_{(KF_1,KF_2)}$ is greater than the threshold $Thrs_{Det}$, this implies that a significant difference between the two positions (Pos_{KF_1}, Pos_{KF_2}) has occurred, which we identify as an error.

Otherwise no error is detected. The diagram 5 schematizes the operation of the fault detection.

$$\Delta Pos_{(KF_1,KF_2)} = \sqrt{(x_{KF_1} - x_{KF_2})^2 + (y_{KF_1} - y_{KF_2})^2} \tag{10}$$

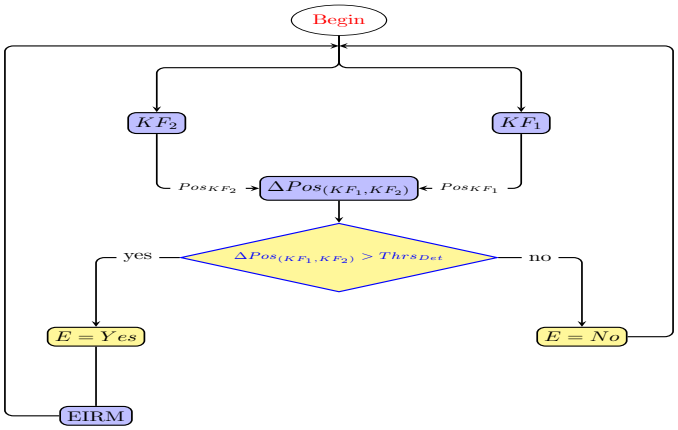

Figure 5: Fault detection organigram

4.5.2 Diagnostic and recovery services

When an error is detected, we proceed to the fault diagnosis in order to isolate the faulty component in the decision-making, and therefore deduce the correct output of the system. This diagnosis is made in two successive steps: *sensor output comparison (SC)*, followed by *residual comparison (RC)*:

- **Sensor outputs comparison (SC)** : in case of an hardware fault, it identifies the wrong sensor type by comparing the outputs of similar sensors. In our case study, we compare in one hand the output of the rear wheel speed sensor (*R-WSS*) with those of the inertial navigation system (Acc, Gyro), both sensors being used in the prediction step of the two Kalman filters. ω_{INS} is compared with ω_R, and γ_{INS} compared with the derivative of the linear speed exit from *R-WSS* $\frac{\partial V_R}{\partial t}$. On the other hand, the position given by the GPS (x_{GPS}, y_{GPS}) is compared with the one of the front odometric system (x_{Fodo}, y_{Fodo}), both sensors blocks being used in the correction step of our Kalman filters. These comparisons are performed using thresholding methods, and three thresholds are therefore necessary: ($Thrs_{\omega_{INS,RWSS}}$, $Thrs_{\gamma_{INS,RWSS}}$, $Thrs_{Pos_{GPS,Fodo}}$).
  - If the output of one sensor block deviates significantly from its dual, the system diagnoses an hardware fault on one of the two sensors. The faulty sensor will be identified in the residual comparison. Comparison of the sensors outputs is done by calculating the three distances given in equations (11) and determining whether they exceed

the aforementioned thresholds ($Thrs_{\omega_{INS,RWSS}}$, $Thrs_{\gamma_{INS,RWSS}}$, and $Thrs_{Pos_{GPS,Fodo}}$).

* $Thrs_{\omega_{INS,RWSS}}$: threshold for comparison between the angular speed given by the gyro ω_{INS} with those deduced from the rear wheel speed sensors (*R-WSS*) ω_R,
* $Thrs_{\gamma_{INS,RWSS}}$: threshold for comparison between the longitudinal acceleration provided by the accelerometer γ_{INS} with the derivative of the linear velocity calculated from the rear encoders (*R-WSS*) $\frac{\partial V_R}{\partial t}$,
* $Thrs_{Pos_{GPS,Fodo}}$: threshold for comparison between the position from the GPS $Pos_{GPS} = (x_{GPS}, y_{GPS})$ with the estimated position by the front odometry system $Pos_{Fodo} = (x_{Fodo}, y_{Fodo})$.

$$\begin{cases} \Delta Pos_{(GPS,F-odo)} = \sqrt{(x_{GPS} - x_{Fodo})^2 + (y_{GPS} - y_{Fodo})^2} \\ \Delta\gamma_{(INS,R-WSS)} = \mid \gamma_{INS} - \frac{\partial V_R}{\partial t} \mid \\ \Delta\omega_{(INS,R_WSS)} = \mid \omega_{INS} - \omega_R \mid \end{cases} \tag{11}$$

– If the sensor outputs are similar (i.e. the distances calculated in equations 11 are less than the chosen thresholds), the system diagnoses a software fault. As mentioned in the previous section we can recover from it with one more data fusion block.

- **Residual comparison (CR)** : in the case of a hardware fault, this step identifies the wrong branch according to the residual values of the two Kalman filters as described in section 3.

Knowing now the type of faulty sensor and the erroneous branch, the system has identified the faulty sensor, and can be restored using the position values of the error free branch.

5 Fault tolerance evaluation

In this section, we propose an experimental study to evaluate the fault tolerance mechanisms described previously. This study relies on real data and fault injection. First, we present the experimental environment (Activity, Faults, Measures) that we used to perform these experiments. Second we present the results of our fault injection campaign.

5.1 Hardware Environment

The Kalman filters and fault tolerance mechanisms detailed in this paper have been validated with real data acquired in the city of Compiègne (France) by the experimental vehicle CARMEN shown in Figure 6. The vehicle is equipped with several sensors: an inertial system measuring longitudinal and lateral accelerations and the yaw rate, a GPS for absolute navigation purpose, and finally

a CAN-bus gateway used to access the wheel speed sensors (WSS), a yaw rate gyrometer, and the steering angle.

Figure 6: Experimental vehicle (CARMEN)

5.2 Software Environment

Our evaluation environment is based on real data acquisition using the software platform PACPUS[1]. Data replay and fault injections were realized with Matlab: the acquired experimental data has been used offline to validate the presented architecture. Each data sample is time stamped, which allows offline reruns of the vehicle experiment. This is particularly useful to automate the experiments, which could not have been automated otherwise. We used fault injection as it is necessary to stress fault tolerance mechanisms. Hardware faults are simulated by the alteration and modification of sensor output values, while software faults are simulated through mutation[2] by modifying the models and gains of the Kalman filters.

5.3 Fault injection campaign

Our fault injection campaign is based on the definition of four essential parameters : *(a) Activity,* the mission performed by the robot and its environment, *(b) Faults,* the domain of hardware and software faults injected in the system, *(c) Results,* the logs and data generated during an experiment, *(d) Measures,* data aggregated from several experiments as a significative measures.

5.3.1 Activity:

We call *activity* the mission requested to the robot in a given environment. The activity in our experiments is driving of the circuit shown in Figure 7. This experimental circuit is composed of a straight line and two roundabouts. In our opinion, it allows to simulate various maneuvers that may be requested to the system.

[1]PACPUS : The Heudiasyc laboratory platform PACPUS aims to federate tools and resources to carry out research and experiments in the intelligent vehicles field. One of the main goals of the platform is to develop, integrate and test driving help and ADAS functions (Advanced Driver Assistance Systems) and autonomous navigation.

[2]A mutation is a basic syntactic change in an existing code.

Figure 7: Experimental Circuit

5.3.2 Sets of faults:

We introduce both hardware and software faults in the system: hardware faults on the sensors outputs ans software faults in the Kalman filters.

- *Hardware faults:* based on feedback from our engineers, we identified three different hardware faults categories on the embedded sensors in our vehicle: *bias faults*, *frozen data faults*, and *null faults*, modeled as follows:

  Let S_{iout} be the sensor output used by our perception mechanism and S_{in} its normal output when the sensor is functional (i.e. before injection).

  - for *bias faults*, the sensor output is biased by a constant Δ.

$$\underset{t_i \leq t_k}{S_{iout}(t_k)} = S_{in}(t_k) + \Delta \tag{12}$$

  - for *frozen data faults*, the sensor always sends the same output from a specific date t_i.

$$\underset{t_i \leq t_k}{S_{iout}(t_k)} = S_{in}(t_i) \tag{13}$$

  - for *null fault*, the sensor is blocked on its initial value, so after time t_i it always delivers the same initial output.

$$\underset{t_i \leq t_k}{S_{iout}(t_k)} = S_{in}(t_0) \tag{14}$$

- *Software faults* are mainly introduced during the software development phase (specification, design, coding, etc). These faults are mainly classified as development faults ; they are generally human made, unintentional and without malicious intent (although intentional sabotage is also possible). Some environmental external faults (such as solar flare or electrical storm) may also in a system cause faults that may be classified as software faults during operation by modifying its executive code (by bit-flips, for

example). Classical examples of software faults include: Assignment - a value is assigned incorrectly or not at all ; Checking - absence or incorrect validation of data or loop or conditions ; Algorithm - incorrect or missing implementation; Function - incorrect or missing parameters. In our experimentations we target any software faults in the source code (Kalman filter code).

We use the mutation technique (i.e. an elementary modification of the basic code) to generate mutated Kalman filters. Studies show that the mutation technique is representative of faults in imperative programs [29]. As sets of good practices are not as mature for declarative programming than for imperative programming, and as they have behavior defining values that are determined empyrically (such as gains for command laws, and noise matrices for Kalmna Filters), declarative programs are particularly prone to software faults. As no studies have been realized to characterize the representativity of software faults in declarative programming, we hypothesized that they are somewhat similar to faults in imperative programming. The mutation technique specifically aims to simulate software faults and validate fault tolerance mechanisms: from an original program (in our experiments the Kalman filters), we create a set of variants called mutants that differ from the original program by a single elementary modification. The behavior of the faults tolerance mechanisms is then analyzed by confronting experimental results of the nominal program and various mutants whose fault were activated during the experiment. But as no such studies exist for declarative programs, we decided to use this technique in our Kalman filters. After analyzing the process and observation models of both implemented Kalman filters, we injected several types of mutations:

- *numerical values substitution*: a numerical value may be replaced by a specific value of its domain (such as 0, 1, and -1 for integers).
- *unary operator*: an unary operator may be replaced by an other one (cos become sin for instance ...).
- *binary operator*: a binary operator may be replaced by a specific operator set (- instead of + and vice versa, * instead of /).
- *multiplication by a real value*: as the matrices Q, R of (process and observation noises, respectively) are initialized empirically through experimentation, faults in their values are highly possible and can affect the Kalman filters outputs. In our mutations we simulated some of these mistakes to reveal their effect on the system output.

5.3.3 Results and measures:

Our experiments produce several intermediates logs. From these various results, we try to establish meaningful measures to quantify the effectiveness of the proposed architecture. These results and measures are listed as follows, and summarized in table 2:

1. **Results**: are parameters that concern one experiment.

   - *Detection delay* (Del_{Det}): it is the difference between the fault injection time t_{Inj} and its detection time t_{Det}.

$$\begin{cases} Del_{Det} = t_{Det} - t_{Inj} \\ \text{With: } t_{Det} = \underset{\Delta Pos_{KF_1,KF_2} \geq Thrs_{Det}}{t_k} \end{cases} \tag{15}$$

   To determine whether the injected fault is detected, identified, and correctly recovered from, we define three boolean variables for each experiment:

   - B_d characterizes if an error is detected by the system ($B_d = 1$ if the detection occurs, $B_d = 0$ otherwise),
   - B_i represents the fact that the type of the fault is correctly diagnosed (hardware in sensors or software in data fusion algorithms) ($B_i = 1$ if the diagnosis is correct, $B_i = 0$ otherwise),
   - B_r determines whether the system recovers after detection (whether it diagnosis correctly the erroneous component and selects the correct output) ($B_r = 1$ if the system recovers correctly, $B_r = 0$ otherwise).

   Before introducing other measures, we need to define the grandeur $diff_{CF,CN}$, which represents the difference between the nominal system behavior and its behavior with an injected fault. This quantity thus determines the difference between the nominal position and the erroneous position.

   Let $(x_{KF_{1N}}, y_{KF_{1N}})$, $(x_{KF_{2N}}, y_{KF_{2N}})$ be the outputs of both Kalman filters KF_1, KF_2 in the absence of faults, and $(x_{KF_{1F}}, y_{KF_{1F}})$, $(x_{KF_{2F}}, y_{KF_{2F}})$ their outputs with an injected fault. Due to the single fault assumption, only one branch is affected by the fault. In the following we will assume that it is the branch 2 (this is an arbitrary choice, since both are symmetrical). In this case $(x_{KF_{1F}}, y_{KF_{1F}}) = (x_{KF_{1N}}, y_{KF_{1N}})$.

   In Figure 8, we define $P_N(x_N, y_N)$ the middle of the segment $[(x_{KF_{1N}}, y_{KF_{1N}})$, $(x_{KF_{2N}}, y_{KF_{2N}})]$, and $P_F(x_F, y_F)$ the middle of the segment $[(x_{KF_{1F}}, y_{KF_{1F}})$, $(x_{KF_{2F}}, y_{KF_{2F}})]$ as defined in equation 16.

$$\begin{cases} x_F = \frac{x_{KF_{1F}} + x_{KF_{2F}}}{2} \ , \ y_F = \frac{y_{KF_{1F}} + y_{KF_{2F}}}{2} \\ x_N = \frac{x_{KF_{1N}} + x_{KF_{2N}}}{2} \ , \ y_N = \frac{y_{KF_{1N}} + t_{KF_{2N}}}{2} \end{cases} \tag{16}$$

   The quantity $diff_{CF,CN}$ is thus the euclidean distance between the two points P_N and P_F, and its value is according to the equation 17:

$$diff_{CF,CN} = \sqrt{(x_F - x_N)^2 + (y_F - y_N)^2} \tag{17}$$

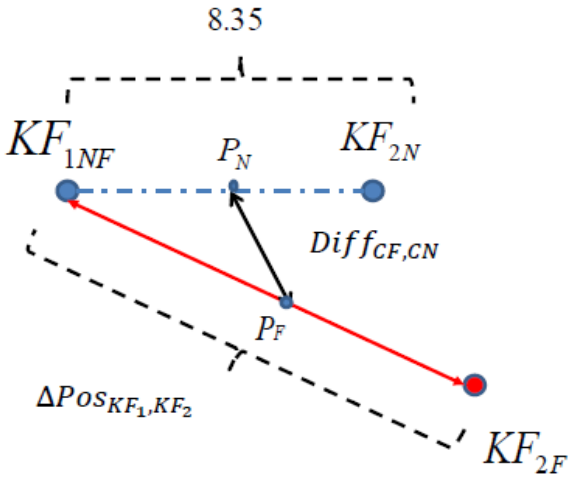

Figure 8: P_N and P_F definition

To characterize our system in terms of *false positives*[3] and *absence of detection*[4] we define two boolean variables B_{Err} and B_{Fail}:

- B_{Err} determines whether the injected fault causes an error in the system, without it being necessarily considered as a failure. We consider here that the system contains a significant error if the difference between the nominal behavior and the erroneous behavior of the system is greater than 2 meters. This variable ($B_{Err} = 1$ if the system contains an error, $B_{Err} = 0$ otherwise) is than defined as:

$$\begin{cases} B_{Err} = 1 \text{ if } diff_{CF,CN} >= (Thrs_{Err} = 2m) \\ B_{Err} = 0 \text{ if } diff_{CF,CN} < (Thrs_{Err} = 2m) \end{cases} \quad (18)$$

- B_{Fail} determines whether the injected fault causes a system failure. We consider that the localization system is failed if the difference between the nominal behavior and the behavior with the injected fault is greater than 10 meters. This variable ($B_{Fail} = 1$ if the system fails, $B_{Fail} = 0$ otherwise) is then defined as:

$$\begin{cases} B_{Fail} = 1 \text{ if } diff_{CF,CN} >= (Thrs_{Fail} = 10m) \\ B_{Fail} = 0 \text{ if } diff_{CF,CN} < (Thrs_{Fail} = 10m) \end{cases} \quad (19)$$

The two thresholds $Thrs_{Err}$ and $Thrs_{Fail}$ were defined here from experimental values from the system's nominal behavior. However, in an

[3] false positives: alerts with absence of errors in the system.
[4] absence of detection: absence of alerts in the presence of errors in the system.

industrial application, these thresholds would be given as system specification.

In our experiments we have also defined three temporal result to characterize the detection performance:

- Del_{Fail} determines how long it takes for the injected fault to cause a system failure:

$$\begin{cases} Del_{Fail} = t_{Fail} - t_{Inj} \\ \text{With: } t_{Fail} = \underset{Diff(CF,CN) \geq Thrs_{Fail}}{t_k} \end{cases} \tag{20}$$

- Del_{Err} determines how long it takes for a significant error to be detected.

$$\begin{cases} Del_{Err} = t_{Det} - t_{Err} \\ \text{With: } t_{Err} = \underset{Diff(CF,CN) \geq Thrs_{Err}}{t_k} \end{cases} \tag{21}$$

- $Del_{Fail,Det}$ determines the time remaining after detection and before the system is led to failure. Failures will be inevitably detected due to our threshold values choice.

$$\begin{cases} Del_{Fail,Det} = t_{Fail} - t_{Det} \\ \text{With: } t_{Det} = \underset{\Delta Pos_{KF_1,KF_2} \geq Thrs_{Det}}{t_k} \end{cases} \tag{22}$$

2. **Measures**: From the previous results, we define for each type of injected fault (hardware or software) measures characterizing the detection, the identification and the recovery rate in our experiments:

- P_{FP} represents the rate of "*false positive*" in the system. A *false positive* is declared if the system detects an error ($B_d = 1$) while its behavior was near the nominal behavior ($B_{Err} = 0$):

$$\begin{cases} P_{FP} = \frac{\text{Number of false positive}}{\text{Number of detected faults}} \\ \text{With: } B_d \& \overline{B_{Err}} \quad \Rightarrow \text{false positive} \end{cases} \tag{23}$$

- P_{ND} represents the rate of "*absence of detection*" in the system. An *absence of detection* is declared if the error generated by the injected fault is not detected ($B_d = 0$) while the system failed ($B_{Fail} = 1$):

$$\begin{cases} P_{ND} = \frac{\text{Number of non detected faults}}{\text{Number of faults causing system failure}} \\ \text{With: } \overline{B_d} \& B_{Fail} \quad \Rightarrow \text{absence of detection} \end{cases} \tag{24}$$

- P_i is the percentage of detected faults correctly diagnosed and identified.

$$P_i = \frac{\text{Number of correctly identified faults}}{\text{Number of detected faults}} \tag{25}$$

- P_r: is the percentage of detected faults correctly recovered from.

$$P_r = \frac{\text{Number of times that the system is successfully recovered}}{\text{Number of detected faults}} \tag{26}$$

- P_{EWF}: is the percentage of detected faults that would have caused the system's behavior to deviate significantly ($B_{Err} = 1$) without failing ($B_{Fail} = 0$). Thus, it is the rate of faults for which the logical expression $B_d \& B_{Err} \& \overline{B_{Fail}}$ is true. For these faults, detection is correct (there was an error), but these errors were not significant enough to cause a system failure. Depending on the system specification, they could be considered as false detection.
- $P_{Det,Err}$: is the percentage of detected faults that would have caused a significant error in the system:

$$P_{Det,Err} = \frac{\text{Injections}/ < B_d = 1 \quad \& \quad B_{Err} = 1 >}{\text{Injections}/ < B_{Err} = 1 >} \tag{27}$$

- $P_{Det,Fail}$: is the percentage of detected faults that causes the system's failures:

$$P_{Det,Err} = \frac{\text{Injections}/ < B_d = 1 \quad \& \quad B_{Fail} = 1 >}{\text{Injections}/ < B_{Fail} = 1 >} \tag{28}$$

Table 2 summarizes the various results and measures employed in our performance evaluation campaign.

5.4 Experimental example:

In this section we present a sample of our experimental campaign. We first present the nominal behavior of the system, then results in the presence of two injected faults.

5.4.1 Nominal behavior without fault injection

To evaluate the developed fault tolerant mechanisms, we performed data acquisition with the instrumented vehicle (figure 6). The vehicle travels a distance of 420 meters on the test circuit shown in Figure 7. For each used sensor, a dedicated software component, time-stamps and records its data. Data from the different sensors are then synchronized through a linear interpolation to use as synchronized time-discretized inputs for the Kalman filters.

		Designation	Unit
Results	Del_{Det}	Detection delay	Second
	B_d	Detection indicator	Boolean
	B_i	Identification indicator	Boolean
	B_r	Recovery indicator	Boolean
	B_{Err}	Error indicator	Boolean
	B_{Fail}	Failure indicator	Boolean
	Del_{Fail}	Failure delay	Second
	Del_{Err}	Error delay	Second
	$Del_{Fail,Det}$	Delay between detection and failure	Second
Measures	P_{FP}	False positives rate	%
	P_{ND}	No detection rate	%
	P_i	Identification rate	%
	P_r	Recovery rate	%
	P_{EWF}	Errors without failure rate	%
	$P_{Det,Err}$	Detected Errors rate	%
	$P_{Det,Fail}$	Detected Failures rate	%

Table 2: Different considered results and measures

A faultless experiment was performed to characterize the nominal system behavior. These results were useful to define the different detection and diagnostic thresholds, and to serve as a reference sample and ground truth for comparison with behaviors in the presence of faults.

Figure 9 represents the vehicle position estimated by the two implemented Kalman filters (Pos_{KF_1}, Pos_{KF_2}) and the output of our architecture Pos_S, which is the average of these two estimations. Figure 10 represents the distance $\Delta Pos_{(KF_1,KF_2)}$. In the nominal case, this distance is less than the chosen detection threshold $Thrs_{Det}$(10 m).

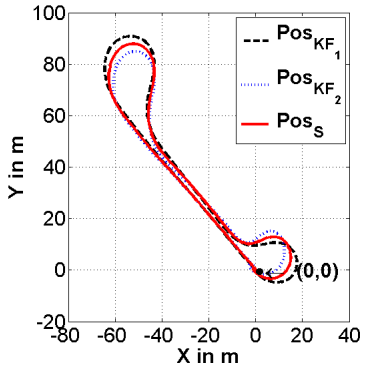

Figure 9: Nominal Behavior: Estimated position by both Kalman filters and the system

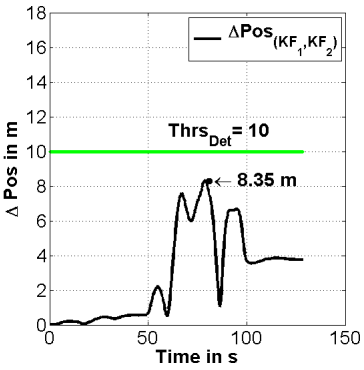

Figure 10: Nominal Behavior: Distance between the position of both Kalman filters

Three other thresholds are needed in our implementation:

- One threshold to compare the GPS position with the front wheel odometry system (F-odo) position: $Thrs_{Pos_{GPS,F-odo}} = 10$ meters.
- Two thresholds to compare the inertial navigation system outputs (INS) with the values processed of the rear wheel speeds sensor (R-WSS): $Thrs_{\gamma_{(INS,R-WSS)}} = 0.45\ m/s^2$, and $Thrs_{\omega_{(INS,R-WSS)}} = 0.23\ rad/s$.
- One threshold to ensure, in the hardware error detection case, that the conflict in one Kalman filters is considerably higher than the other $Thrs_{Res} = 9$ meters.

All these thresholds were chosen to be significantly higher than the values observed in the nominal behavior.

5.4.2 System's behavior in the presence of faults

Here we present the results of two fault injections: one hardware faults on the GPS sensor, and one software fault in the Kalman filter KF_1. These two two faults have been presented in [30].

- **Additive faults on GPS**

  The first injected fault that we consider is an additive permanent fault in GPS output. This external fault simulates a typical jump in the GPS position due to satellite signals rebounds. This fault is usually not permanent, but can occur for a significant time. At time $t_i = 60.5$ seconds, we added a jump of 10 meters on the x_{GPS}, as described in equation (29).

$$\underset{t_i \leq t_k}{x_{GPS}(t_k) = x_{GPS}(t_i) + 10} \tag{29}$$

  In Figure 11, we see that the difference between the positions estimated by the two Kalman filters $\Delta Pos_{(KF_1,KF_2)}$ exceeds the threshold $Thrs_{Det}$, so the system detects an error at time $t_{Det} = 62.5$ seconds.

  We see in Figure 12 that the INS and R-WSS outputs are similar, while the difference between the GPS position and F-odo position exceeds the threshold $Thrs_{Pos_{GPS,F-odo}}$. This indicates that the detected error is a hardware fault in the GPS or F-odo.

  Figure 13 confirms that the absolute value of the residual generated from KF_2 ($\mid Res_{KF_2} \mid$) is significantly higher than the absolute value of the residual generated from KF_1 ($\mid Res_{KF_1} \mid$) indicating that the branch KF_2 contains an error.

  Knowing now the couple (GPS, F-odo) containing the faulty sensor and the wrong branch (Branch 2), the system can identify the GPS as a faulty

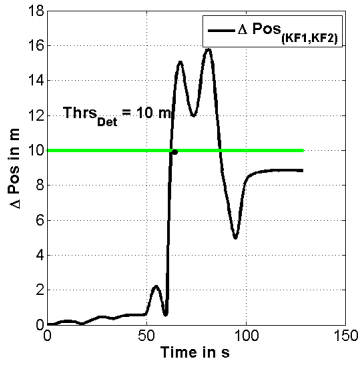

Figure 11: Additive Fault in GPS: Distance between the position of both Kalman filters

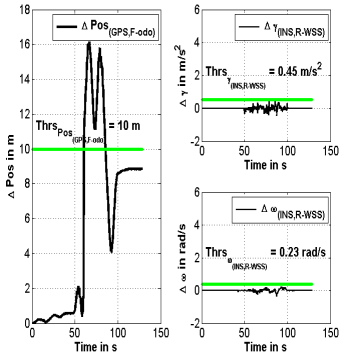

Figure 12: Additive Fault in GPS: Sensor Output Comparison

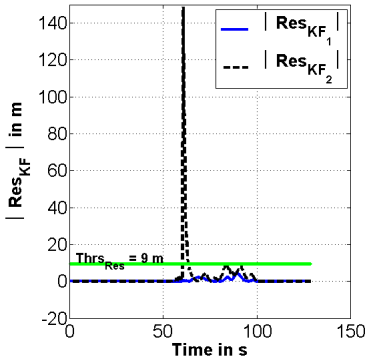

Figure 13: Additive Fault in GPS: Residue Comparison

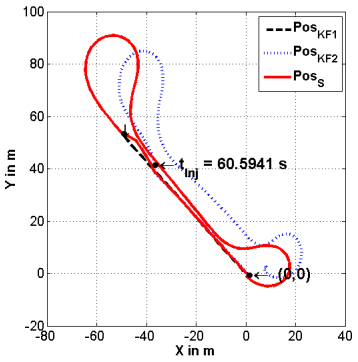

Figure 14: Additive Fault in GPS: Estimated position by both Kalman filters and our architecture

sensor, and take the correct output Pos_{KF_1} at t = 62.5 seconds, as shown in Figure 14, 2 seconds after the injection.

Note that the residual Res_{KF_2} has significant value only for a short period of 3 seconds. After that, it takes a value below our diagnostic threshold because the gaps between the erroneous GPS and the correct F-odo have been smoothed by the filter. It is therefore vital to detect the error in that window, and the determination of thresholds is confirmed to be very important.

Note that these thresholds depends on the application, and their optimization may be a costly work for the design of a new system.

- **Software fault in the Kalman filter KF_1**

  This injection is a software fault in the first Kalman filter KF_1. As mentioned above, Kalman filters mechanisms appear as particularly sensitive to design faults. For example, Covariance noises Q and R have a big impact on the filters output estimation, and are usually determined empirically. In this scenario, we put $Q = R/10^3$ from the beginning of the experiment ($t_{Inj} = 0$) to give more confidence to the INS system than the F-odo system.

  As shown in Figure 15, the estimated position by KF_1 slowly drift than from the one estimated by KF_2, before diverging abruptly and exceeding our detection threshold at time t = 87.1 seconds.

  Figure 16 shows that the detected error is not diagnosed as a hardware fault in the perception sensors. This indicates a software fault in one of the data fusion algorithms (KF_1 or KF_2) without distinguishing which one, for lack of software redundancy.

  After the detection of this software fault at time $t_{Det} = 87.1$ seconds, our architecture generates an alert indicating that the localization system has failed as shown in Figure 17. It is particularly interesting to see that such fault can not be detected for up to 80 seconds because the results of the failed block are similar to those of the non-failing block. This reinforces our opinion that data fusion mechanisms are difficult to validate and their faults should be taken into account by fault tolerance mechanisms.

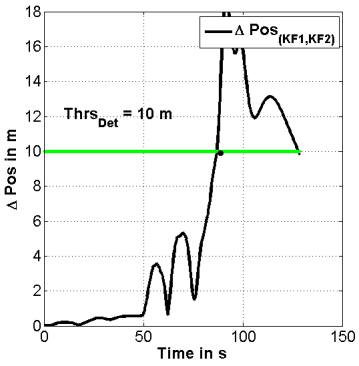

Figure 15: Software Fault in Kalman filter 1:Distance between the position of both Kalman filter

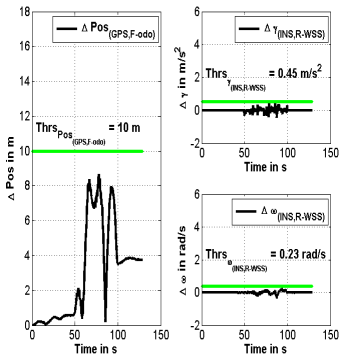

Figure 16: Software Fault in Kalman filter 1:Sensor Output Comparison

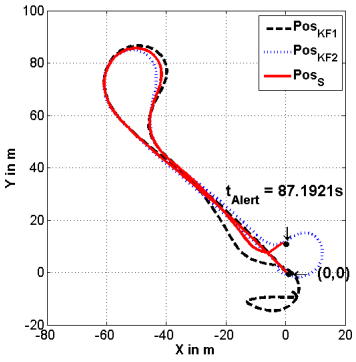

Figure 17: Software Fault in Kalman filter 1:Estimated position by both Kalman filters and our architecture

5.5 Global results

In our experiments, 90 faults on the sensors and 20 faults in Kalman filter algorithms were injected. For each type of component, we injected a set of different faults with different values, at different times, in order to validate as widely as possible their impact on the final result of the architecture, and measure the ability of our approach in terms of detection and recovery. Although the number of faults that we injected appears limited for the validation of a critical system, the injection and experiments are automated. Time is thus the only factor to limit the validation of our fault tolerant mechanisms. Exhaustive validation is of course impossible, but a large number of injecting faults in diverse environments and activities is crucial to justify trust in the fault tolerance of the system.

This section presents the overall results of our experimental campaign.

5.5.1 Hardware faults

- **Faults on GPS:** the GPS suffer from several external errors due to diverse environmental factors. we injected jumps in the position provided by the GPS, simulating four different jumps (2, 5, 10, 20) in four directions (+ X, -X, + Y, -Y). These faults were injected at four different dates. These different combinations make sixty-four (64) the number of injected faults on the GPS. A sample results of these injections is shown in Table 3.

  The experimental results show that all the injected faults that causes a system's failure are correctly detected, diagnosed, and recovered from.

- **Faults on the odometry sensors:** To simulate the faults that may occur on the encoders, we injected the three fault types (null, frozen, bias) described in 5.3.2. The null and frozen data faults were injected at

Comp	hop	Dir	t_{Inj}	t_{Det}	Del_{Det}	Del_{Fail}	$Del_{Fail,Det}$	Del_{Err}
GPS	2	+X	60.59	X	X	X	X	X
	5	-X	87.47	90.27	2.80	X	X	1.96
	10	-Y	72.35	72.91	0.56	X	X	0.28
	20	+Y	82.15	83.27	1.12	45.36	44.24	1.12

Table 3: Sample of additive faults in GPS

four different times making the eight fault injection scenarios presented in Table 4. The bias faults (1, 2, 3, and 4) were also injected at four different times, making sixty (16) fault injection scenarios (a sample is shown in Table 4). Thus twenty-four (24) faults in total are injected on the F-WSS. The table 4 give the results found in these cases.

Comp	Type	t_{Inj}	t_{Det}	Del_{Det}	Del_{Fail}	$Del_{Fail,Det}$	Del_{Err}
F-WSS	Hard	60.59	64.51	3.92	13.44	9.52	0.56
		87.47	99.79	12.32	13.72	1.40	5.04
	Null	60.59	62.27	1.68	2.52	0.84	1.12
		87.47	90.83	3.36	X	X	2.52
	Bias(1)	72.35	85.23	12.88	33.88	21.00	10.08
	Bias(4)	87.47	89.15	1.68	8.12	6.44	0.84

Table 4: Sample of faults injected on F-WSS

Note that all the injected frozen data faults are detected, diagnosed, and recovered from.

Note that the injected null faults are detected, and diagnosed correctly. However in the last injected fault, recovery has not occurred since the residuals generated by the two filters KF_1 and KF_2 do not exceed the chosen threshold $Thrs_{Res}$. This is an example of an error without failure (EWF).

Finally note that all the injected bias faults are detected and diagnosed correctly. However for a bias of 1 meter the system recovery is not done because this bias does not allow the residual generated by the Kalman filter to be significant.

- **Fault on the Steering angle sensor:** we injected only two *Null* faults on the steering angle sensor. The two injection times correspond to the instant when the vehicle begins driving on through respectively the first and the second straight line of the experimental circuit. On the straight lines, the fault is not detected because it has no significant impact on the system. It is detected as soon as the vehicle turns. Table 5 summarizes the results for these faults.

Comp	**Type**	t_{Inj}	t_{Det}	Del_{Det}	Del_{Fail}	$Del_{Fail,Det}$	Del_{Err}
Steering angle	Null	59.75	67.31	7.56	9.52	1.96	1.40
		82.15	88.03	5.88	13.16	7.28	1.12

Table 5: Null Faults in steering angle sensor

5.5.2 Software faults:

For software faults, many mutations are possible. We considered a sample of twenty (20) different mutations, simulating errors in the process and observation models of the Kalman filter. We injected the following mutations: 8 substitutions of numerical values, 4 substitutions of geometrical expressions, 2 variables sign exchange, 4 variable substitutions, 2 mutations modeling errors in covariance matrices Q and R estimation.

These mutations were generated manually by changing the Kalman filters codes. Table 6 show the effects of these software errors, and the ability of our approach in terms of detection of such errors.

Original	Mutant	t_{Inj}	t_{Det}	Del_{Det}	Del_{Fail}	$Del_{Fail,Det}$	Del_{Err}
Q1	Q1 = 10^{-5}.R1	0.00	87.19	87.19	X	X	25.76
$A^1_{1,4} : cos(\theta)$	$A^1_{1,4} : -cos(\theta)$	0.00	72.07	72.07	X	X	18.76
$B^1_{1,1} : \frac{1}{2}dt^2$	$B^1_{1,1} : dt^2$	0.00	X	X	X	X	X
$H^1_{1,1} : 1$	$H^1_{1,1} : 0$	0.00	88.03	88.03	89.99	1.96	21.28
Q2	Q2 = $10^{-5}.R2$	0.00	84.67	84.67	X	X	24.92
$A^2_{1,1} : 1$	$A^2_{1,1} : -1$	0.00	55.83	55.83	57.79	1.96	3.08
$B^2_{1,1} : cos\theta$	$B^2_{1,1} : sin\theta$	0.00	57.79	57.79	X	X	5.60
$H^2_{1,2} : 0$	$H^2_{1,2} : 1$	0.00	55.55	55.55	58.35	2.80	2.52

Table 6: Sample of software faults in Kalman filters

5.6 Global Measures

This section present the measures discussed in section 5.3.3 for our experimental campaign.

5.6.1 Hardware faults:

On the ninety (90) hardware injected faults on GPS sensors, F-WSS, and the steering angle sensor, all faults are correctly detected. However only 62.32 % are diagnosed and recovered from. This performance gap is due primarily to the choice of the used recovery thresholds. Indeed, we have empirically chosen all thresholds with only criterion being that they exceed nominal values. To improve the performance of our architecture, a better study of these thresholds is necessary. We also note that no *false positive* is generated by our architecture

on this set of hardware faults, and no *absence of detection* is reported. 44 % of faults injection correctly detect a significant error ($Diff_{CF,CN} \geq Thrs_{Err}$) while the system is not considered as failed ($Diff_{CF,CN} \geq Thrs_{Fail}$). These errors could be considered as false detection that can be treated using two solutions:

- to take into account the significant external disturbance that may occur on the sensors (eg: reflection of the sun on a camera, GPS jumping, measurement noise on inertial sensors) by raising the detection threshold.
- to consider them temporary faults, which would disappear from the system at the same time than the disturbances.

Table 7 provides an overview of global measures on all different hardware injected faults in our experimental campaign.

Type	P_d (%)	P_i (%)	P_r (%)	P_{FP} (%)
hardware	100.00	100.00	62.32	0.00
	P_{ND} (%)	P_{EWF}(%)	$P_{Det,Err}$(%)	$P_{Det,Fail}$(%)
	0.00	44	91.89	100.00

Table 7: Measures on hardware faults

5.6.2 Software faults:

On the 20 simulated mutations, 90 % of the faults affecting the system ($B_{Err} = 1$) were detected by our architecture. As mentioned previously, the level of software redundancy in our architecture do not allow the detected software faults to be recovered. To ensure this recovery service a diversified third branch is needed. For the injected software faults no *false positive* was generated and no *absence of detection* was reported. 44 % of realized mutations generate an error in the system without failure. These errors, as previously mentioned in the hardware faults, could be considered as false detection.

Table 8 summarizes the measures on the various performed mutations simulating software faults.

Type	P_d (%)	P_i (%)	P_r (%)	P_{FP} (%)
Software	90.00	100.00	0.00	0.00
	P_{ND} (%)	P_{EWF}(%)	$P_{Det,Err}$(%)	$P_{Det,Fail}$(%)
	0.00	44	88.89	100.00

Table 8: Measures on software faults

6 Conclusion and perspectives

The use of multi-sensor perception begins to spread in critical applications, which raises more and more the problem of their dependability. We presented in this paper a fault tolerance architecture based on duplication-comparison. This architecture provides the fault detection, and system recovery services, but requires means to compare the outputs of redundant sensors and to also compare the outputs of fusion mechanisms.

We presented an example of real application for our proposed approach, that consists of vehicle localization using Kalman filters. We have presented its fault tolerance evaluation using real experimental data and fault injection, demonstrating the effectiveness of our mechanisms in terms of fault detection and system recovery.

In future work, we intend to implement a third Kalman filter to ensure software fault tolerance in our vehicle localization application. We also plan to carry out a comparative study of our real application results according to different thresholds,seeking a method to determine optimal thresholds. Finally, we intend to implement this approach in addition of transient faults, where we assume that by maintenance or time, an activated fault may disappear.

7 Acknowledgment

This work was carried out and funded under the LABEX MS2T and ROBOTEX TEAM. It was supported by the French government, through the programs "Future Investments" managed by the French National Research Agency (References: ANR-11-IDEX-0004-02 and ANR-10-44 EQPX).

References

[1] E. CENELEC, 50128: Railway applications-communication, signalling and processing systems-software for railway control and protection systems, 2011.

[2] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 1 (1) (2004) 11–33.

[3] J.-C. Laprie, Dependability: Basic Concepts and Terminology, Springer-Verlag New York, Secaucus, NJ, USA, 1992.

[4] I. Bloch, H. Maître, Data fusion in 2d and 3d image processing: an overview, in: 10th Brazilian Symposium on Computer Graphics and Image Processing, 1997, pp. 127–134.

[5] L. Freeston, Applications of the kalman filter algorithm to robot localisation and world modelling, Electrical engineering final year project. University of Newcastle, Australia.

[6] C. Hu, W. Chen, Y. Chen, D. Liu, Adaptive kalman filtering for vehicle navigation, Journal of Global Positioning Systems 2 (1) (2003) 42–47.

[7] O. L. Casanova, et al., Robot position tracking using kalman filter, in: Proceedings of the World Congress on Engineering,London UK, Vol. II, 2008.

[8] S.-G. Kim, J. L. Crassidis, Y. Cheng, A. M. Fosbury, J. L. Junkins, Kalman filtering for relative spacecraft attitude and position estimation, Journal of Guidance, Control, and Dynamics 30 (1) (2007) 133–143.

[9] P. Escamilla-Ambrosio, N. Mort, A hybrid kalman filter-fuzzy logic multisensor data fusion architecture with fault tolerant characteristics, in: Proceedings of the international conference on artificial intelligence, 2001, pp. 361–367.

[10] P. Caspi, R. Salem, Threshold and bounded-delay voting in critical control systems, in: Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, 2000, pp. 327–337.

[11] S. Zug, J. Kaiser, An approach towards smart fault-tolerant sensors, in: International Workshop on Robotic and Sensors Environments. ROSE, 2009, pp. 35–40.

[12] K. Marzullo, Tolerating failures of continuous-valued sensors, ACM Transactions on Computer Systems (TOCS) 8 (4) (1990) 284–304.

[13] P. M. Frank, Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: A survey and some new results, Automatica 26 (3) (1990) 459–474.

[14] V. Ricquebourg, M. Delafosse, L. Delahoche, B. Marhic, A. Jolly-Desodt, D. Menga, Fault detection by combining redundant sensors: a conflict approach within the tbm framework, Cognitive Systems with Interactive Sensors, COGIS 2007.

[15] P. Smets, The transferable belief model for quantified belief representation, Handbook of defeasible reasoning and uncertainty management systems 1 (1998) 267–301.

[16] P. Smets, Data fusion in the transferable belief model, in: Proc of the third International Conference on Information Fusion, Vol. 1, 2000, pp. PS21–PS33.

[17] G. Shafer, A mathematical theory of evidence, Vol. 1, Princeton university press Princeton, 1976.

[18] F. Delmotte, G. Gacquer, Detection of defective sources with belief functions, in: Proceedings of 12th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Malaga, Spain, 2008, pp. 337–344.

[19] F. Delmotte, Detection of defective sources in the setting of possibility theory, Fuzzy sets and systems 158 (5) (2007) 555–571.

[20] B. Li, Y. Zhu, X. Rong Li, Fault-tolerant interval estimation fusion by dempster-shafer theory, in: Proceedings of the fifth International Conference on Information Fusion, Vol. 2, 2002, pp. 1605–1612.

[21] L. Shu-qing, Z. Sheng-xiu, A congeneric multi-sensor data fusion algorithm and its fault-tolerance, in: International Conference on Computer Application and System Modeling (ICCASM), Vol. 1, 2010, pp. V1–339.

[22] D. J. Allerton, H. Jia, Distributed data fusion algorithms for inertial network systems, Radar, Sonar & Navigation, IET 2 (1) (2008) 51–62.

[23] M. Peng, J. He, M. Shen, K. Xie, Analog fault diagnosis using decision fusion, in: 7th International Conference on Computer Science & Education (ICCSE), 2012, pp. 17–20.

[24] D. Zhuo-hua, C. Zi-xing, Y. Jin-xia, Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey, in: Proceedings of the International Conference on Robotics and Automation, ICRA, 2005, pp. 3428–3433.

[25] J. K. Uhlmann, Covariance consistency methods for fault-tolerant distributed data fusion, Information Fusion 4 (3) (2003) 201–215.

[26] N. N. Okello, S. C. Thomopoulos, Design of data fusion system for multiradar target detection, in: International Conference on Systems, Man and Cybernetics,'Systems Engineering in the Service of Humans', Vol. 3, 1993, pp. 672–677.

[27] K. Bader, B. Lussier, W. Schön, Functional diversification for software fault tolerance in data fusion: a real application on kalman filters for mobile robot yaw estimation, in: The annual European Safety and Reliability Conference ESREL, 2015.

[28] A. De Luca, G. Oriolo, C. Samson, Feedback control of a nonholonomic car-like robot, in: Robot motion planning and control, Springer, 1998, pp. 171–253.

[29] M. Daran, Modélisation des comportements érronés du logiciel et application á la validation des tests par injection de fautes, Ph.D. thesis, Institut National Polytechnique de Toulouse (1996).

[30] K. Bader, B. Lussier, W. Schön, A fault tolerant architecture for data fusion targeting hardware and software faults, in: The 20th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2014.