Data fusion fault tolerant strategy for a quadrotor UAV under sensors and software faults

Hussein Hamadi, Benjamin Lussier, Isabelle Fantoni, Clovis Francis

Data Fusion Fault Tolerant Strategy for a Quadrotor UAV under Sensors and Software Faults

Hussein Hamadi[a,b,*], Benjamin Lussier[a], Isabelle Fantoni[c], Clovis Francis[b]

[a] *Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253 Heudiasyc, 60200 Compiègne, France*
[b] *Université Libanaise, Faculté de Génie, Centre de Recherche Scientifique en Ingénierie (CRSI), Liban*
[c] *Laboratoire des Sciences du Numérique de Nantes (LS2N), UMR CNRS 6004, 1 rue de la Noë, 44321 Nantes, France*

Abstract

This article presents the design and implementation of a fault tolerant architecture for sensor fusion that tolerates faults on a quadrotor unmanned aerial vehicle (UAV). It aims to tolerate both hardware sensors faults (GPS jamming, IMU lock or freezing, magnetometer sensitivity to high power magnetic fields...) and software faults (faults in the Kalman filter, bad parameters initialization....). The proposed architecture uses data fusion with Kalman filters in order to estimate the states (position and orientation) of the UAV. It includes an analytical redundancy using the dynamic model of the system. The estimations of the defined Kalman filters and the dynamic model feed a weighted average voter, which increases the accuracy of the outputs and the error detection process. The proposed architecture allows multiple recovery solutions to a faulty system and thus increasing its flexibility. The architecture is validated using numerical simulations and experimental flights in real outdoor environment using a quadrotor.

Keywords: Fault tolerance, data fusion, sensor faults, software faults, Kalman filter, Unmanned Aerial Vehicle

1. Introduction

Nowadays, Unmanned Aerial Vehicles (UAV) show a growing interest for applications such as infrastructures surveillance (such as power grid or railways), hazardous terrain exploration (such as snowslide or landslide), monuments modeling and restoration, etc. During their operations, multirotor UAVs are subject to different technical and operational constraints such as limited battery, weight, altitude, possible faults, and flight over populated area. Thus, it is important to increase their technical reliability to ensure a safe flight in critical areas despite faults in the system.

However, in order to allow such critical systems to operate in an open environment where their failure could cause catastrophic consequences their fault tolerance must be improved. Moreover, these fault tolerance mechanisms must not be too costly in order to be implemented on UAVs with limited payloads. In this article, we particularly focus on failures in the data fusion system caused by sensor and software faults in the data fusion process.

In this work, we present a fault-tolerance architecture for data fusion targeting sensors and software faults on an outdoor quadrotor UAV. This architecture extends the duplication-comparison technique described in [1]. This architecture uses data fusion with Kalman filters in order to estimate the states (position and orientation) of the UAV.

This paper is organized as follows: first, we present a state of the art on data fusion systems for UAV in section 2. Then we describe the UAV dynamics and the relations between the virtual and real control inputs along with system's and motor's parameters identification in Section 3. These equations will be used in our architecture as a dynamic model that is diversified compared to the sensors outputs. In Section 4, we describe the well-known general equations of the Kalman filter and the Extended Kalman filter that we use in our architecture. Then, in Section 5, we propose our fault tolerance architecture for data fusion, which uses a voting system, the redundancy based approach (Duplication/Comparison) described in [1] and the redundant analytical dynamic model previously mentioned based on the quadrotor's equations of motions. In Section 6, we detail a case study on the quadrotor *Tarot*650 where we show the effectiveness of the proposed strategy through real experiments for hardware faults, and numerical simulation for software faults. Finally, Section 7 presents our conclusions and possible perspectives.

*Corresponding author
Email address: hussein.hamadi@altran.com (Hussein Hamadi)

2. State of the art on fault tolerance for data fusion

The field of data fusion plays a fundamental role for multi-sensor data fusion systems. In general, for dynamic systems, it is implemented using the Kalman filter technique that allows to fuse data from different sensors and reduces noise. We present here a state of the art for data fusion and then for fault tolerant techniques in data fusion.

Data Fusion [2] consists in joining or merging information obtained from several sources and exploiting that information in various tasks such as answering questions, making decisions, estimating numerical values, etc. These sources are, commonly in robotics, physical sensors observing the actual situation and providing different information on the possible events. Other definitions of the data fusion concept can be found in [3]. Multi-sensor data fusion combines several sensors measurements to form a better and easier representation to use of the observed environment: the world model. It seeks to take advantage of all available information on a given problem to counter each sources imperfections and improve robustness in data fusion. Generally three major theoretical frameworks are used to implement a data fusion mechanism: the probability theory, the possibility theory [4, 5], and the belief function theory [6]. In our work we focus on the probabilistic framework, particularly the Kalman filter method [7].

Data fusion has a long history in the robotic field. It has been a significant focus during the 80-90s in military applications [8, 9], and remains relevant in this area. Fusion methods have been adapted and developed for robotics applications (such as autonomous navigation, target tracking, and localization). In [10] a belief function theory data fusion technique is used and adapted to the robot localization problem using ultrasound measurements. As navigation is fundamental for mobile robots, Kalman filters have been used in system localization for a long time [11, 12]. As the original Kalman filter can only be applied to linear systems, the extended Kalman filter (EKF) has been proposed for non linear systems. This EKF has been successfully implemented for robot position estimation in [13], and [14].

2.1. State of the art on multi-sensor data fusion systems

By using data fusion the risk of software and hardware faults increases, in terms of sensor failures and processing failures, due to the rising number of sensors and the underlying data fusion mechanisms [15]. Hence, there is a need to apply a fault tolerant strategy in safety critical applications to overcome these issues and detect any failures, and to ensure more reliable performance outcomes with respect to autonomous systems. To our best knowledge, there exist only few works in the literature regarding fault tolerance in the field of data fusion. The approaches that can be found use the duplication and comparison techniques to tolerate physical faults [16]. These approaches can be categorized into two categories: duplication based on an analytical model, and duplication based on hardware redundancy.

The model based approaches, also known as analytical redundancy approaches [17], determine functional relationships between the measured states through a mathematical model. This mathematical model can either be developed from physics analyses or obtained from the measurements directly. Subsequently, a residual r_k is then generated between the actual sensor output y_k and the estimated modeled output $\hat{y}_k$, i.e.,

$$r_k = y_k - \hat{y}_k \tag{1}$$

A residual zero-mean, that is, $\sum_k \frac{r_k}{k} = 0$ means no fault and the mean deviation from zero means the existence of a fault. A Nadaraya-Watson statistical estimator and a priori observations are used in [18] to validate sensor measurements. Residuals or innovations generated by the Kalman filter (KF) were used in [19, 20, 21] to detect faults: statistical tests on residual whiteness, mean, and covariance,... In [21], a failure detection approach for a KF-based GPS integrity monitoring system was proposed. The idea is to process subsets of the measurements by an auxiliary KF component and to use the estimate generated as a reference for the detection of failures. The KF prediction was used as a reference for detecting inconsistencies in the measurement of sensors in [22]. An adaptive sensor/actuator detection and isolation scheme for a Unmanned Aerial Vehicle (UAV) based on KF have been proposed in [23]. The detection of a system failure in this method is done by applying statistical tests on the KF's innovation covariance. In [24], this method is used to improve the accuracy of personal outdoor positioning systems. Common tools for assessing residual statistical characteristics are generalized probability ratio tests [25], chi-square tests [26], and multiple hypothesis tests [27]. Some authors also proposed approaches based on Extended KF (EKF) [28, 29] and Unscented KF (UKF) [30] with the objective of detecting inconsistencies in the data fusion of non-linear systems. Multi-sensor data fusion for multi-robot system based on Kullback-Leibler Divergence (KLD) was proposed in [31]. The method calculates the KLD between an Informational Filter a *priori* and a *posteriori* distributions and uses the threshold of the Kullback-Leibler Criterion to detect and remove suspicious sensor data.

Considering hardware redundancy based approaches, two or more sensors usually measure the same critical state and then detect as well as isolate the faulty sensors by consistency checks and majority voting [17]. For example, in

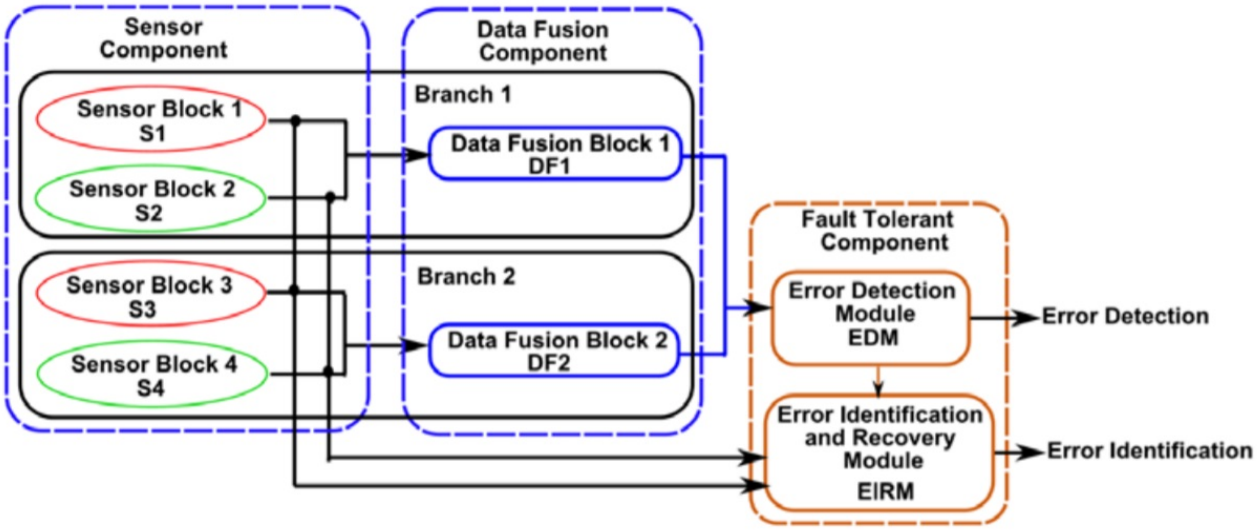


Figure 1: Duplication-comparison architecture for fault tolerance in multi-sensor data fusion, adopted from [1]

[32], the authors proposed a voter-based fault detection system for multiple sensors subsystems of inertial navigation system (INS), GPS, attitude sensor and heading reference system (DAHRS). A sensor voting algorithm was presented in [33] to manage three redundant sensors.

Based on redundant multi-sensor navigation systems,an inconsistency detection for hypersonic cruise vehicles (HCVs) was proposed in [34]. Two sensor blocks were involved: the first block consists of an inertial navigation system (INS) and a GPS, while the second block consists of an INS and a navigation system. The method uses chi-square test and sequential probability ratio test to detect inconsistencies in each block's local sensor estimates before sending their data to a central node for a global estimate. In another work, an application for failure detection and isolation on redundant aircraft sensors based on a fuzzy logic and a majority voting was proposed in [35]. A method for detecting spurious sensor data based on the Bayesian framework without any prior information was proposed in [36]. This method adds a term to a Bayesian probabilistic approach that increases the a *posteriori* distribution if measurement from one sensor is inconsistent with the other.

2.2. Related works based on duplication/comparison

In [1], the authors introduced an approach for tolerating faults using multi-sensor data fusion. This approach is based on the method of duplication/comparison and offer detection and diagnosis of faults in a data fusion mechanism and a subsequent system recovery. Error detection helps to detect the erroneous state of the system before the propagation of the error can cause the failure of the system. System recovery allows an error-free state to be substituted in place of an erroneous state. The Figure 1 illustrates the architecture for fault tolerance using duplication-comparison and multi-sensory data fusion. The major difference between this approach and the ones introduced earlier is that it offers the capability of taking into account the software faults in the system. Also we believe that this strategy is more reliable and accurate in identifying and dealing with faults in the system since it is less sensitive to uncertainties and has a reduced number of assumption to be considered in the conception of the architecture.

3. Quadrotor Dynamics and Motor Identification

In this section, the dynamic model of the quadrotor UAV is presented. The set of equations of motion given in this section and governing the system dynamics will be used in our fault tolerant architecture as a diversified branch called Dynamic model (DM) in the fusion process, which will be compared to the sensors outputs.

The modeling process is made under the following assumptions:

- **Assumption 1** The structure of the vehicle is supposed to be rigid and symmetrical. Specifically, the center of gravity is supposed to be fixed and the actuator's position are symmetrical with respect to the vehicle axes, which will allow much simpler equations concerning the forces applied on the system.

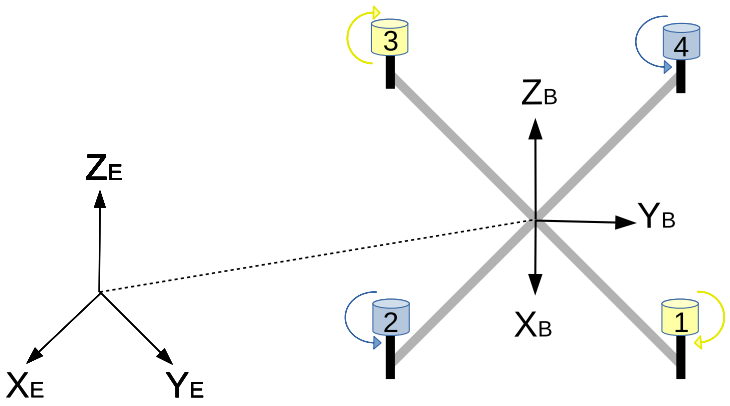


Figure 2: Quadrotor configuration

- **Assumption 2** The motor dynamics are ignored. This will allow to not consider the equations between the motor's rotational speed and the feeding current and voltage.
- **Assumption 3** The center of gravity and the body-fixed frame origin are assumed to coincide. This will allow the off-diagonal terms in the inertia matrix to be zero.
- **Assumption 4** The propellers are supposed to be rigid. Thus, we ignore the blade flapping (the up and down movement of a rotor blade).
- **Assumption 5** The thrust and the drag are proportional to the square of the rotors speed. This simplifies the identification procedure of the motor parameters.
- **Assumption 6** The system dynamics are limited to small angles and small variations of linear and angular velocities, and thus no acrobatic behavior of the UAV can occur.

3.1. Dynamic model

The dynamical model of the quadrotor (Fig. 2) can be found following the Newton-Euler formalism. First, by defining the Earth frame $\{X_E, Y_E, Z_E\}$ and the body frame $\{X_B, Y_B, Z_B\}$, the translation motion of the UAV can be obtained using Newton-Euler's law:

$$m\dot{\mathbf{v}} = \mathbf{R}_{B\to E}.\mathbf{F} + \mathbf{F_w} - \mathbf{F_z} \tag{2}$$

where m is the mass of the UAV, $\mathbf{v}$ is the velocity vector in the Earth frame, $\mathbf{F} = \begin{bmatrix}0 & 0 & u_f\end{bmatrix}^T$ is the vector representing the sum of all the forces in the body frame, $\mathbf{F_z} = \begin{bmatrix}0 & 0 & mG\end{bmatrix}^T$ represents the weight of the vehicle with G the gravitational constant, $\mathbf{F_w} = \begin{bmatrix}F_{w_x} & F_{w_y} & F_{w_z}\end{bmatrix}^T$ is the vector of external wind perturbations acting on the UAV represented in the earth frame and u_f is the thrust force produced by the motors and $\mathbf{R}_{B\to E}$ is the rotation matrix from the body frame to the earth frame. Therefore, we obtain:

$$\begin{cases} m\ddot{x} &= (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)u_f + F_{w_x} \\ m\ddot{y} &= (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)u_f + F_{w_y} \\ m\ddot{z} &= (\cos\phi\cos\theta)u_f - mG + F_{w_z} \end{cases} \tag{3}$$

where ϕ, θ and ψ represent the Euler angles respectively (roll, pitch and yaw angles). Using Newton-Euler's law, we can model the dynamic of the UAV's attitude by the following:

$$I\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times I\boldsymbol{\omega} + \boldsymbol{\tau} \tag{4}$$

or

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{5}$$

where $\boldsymbol{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ represents the angular velocity vector in the body frame, $\boldsymbol{\tau} = \begin{bmatrix} \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$ is the vector of moments acting on the UAV and I_{xx}, I_{yy} and I_{zz} are the inertial constants of the UAV. A simplification is made by setting $\begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T = \begin{bmatrix} p & q & r \end{bmatrix}^T$. This assumption holds true for small angles of movement (**Assumption 6**). So, the full dynamic model of the UAV in the inertial frame is given by:

$$\begin{cases} m\ddot{x} &= (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)u_f + F_{w_x} \\ m\ddot{y} &= (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)u_f + F_{w_y} \\ m\ddot{z} &= (\cos\phi\cos\theta)u_f - mG + F_{w_z} \\ I_{xx}\ddot{\phi} &= \dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) - J_r\dot{\theta}\Omega_r + \tau_\phi \\ I_{yy}\ddot{\theta} &= \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) + J_r\dot{\phi}\Omega_r + \tau_\theta \\ I_{zz}\ddot{\psi} &= \dot{\phi}\dot{\theta}(I_{xx} - I_{yy}) + \tau_\psi \end{cases} \tag{6}$$

The virtual inputs of the system are related to the force and torque of each motor by the following expressions:

$$\begin{aligned} u_f &= F_1 + F_2 + F_3 + F_4 \\ \tau_\phi &= d(F_1 + F_4 - F_2 - F_3) \\ \tau_\theta &= d(F_1 + F_4 - F_2 - F_3) \\ \tau_\psi &= (\tau_2 + \tau_4) - (\tau_1 + \tau_3) \end{aligned} \tag{7}$$

where d is the distance from the motor's position to the corresponding axis which is equal for all motors, and the force F_i and the torque τ_i produced by each motor is proportional to the square of the angular speed :

$$\begin{aligned} F_i &= K_f\omega_i^2 \\ \tau_i &= K_t\omega_i^2 \\ i &= 1, ..., 4 \end{aligned} \tag{8}$$

where K_f and K_t represent respectively the thrust and drag coefficients of the actuators. The model presented in equations (6) is widely used in the literature for describing the quadrotor motion and is validated in many publications such as [37].

3.2. Motor model Identification Procedure

The relation between the generated aerodynamic forces and moments by the propellers are the following: $F_i = K_f\omega_i^2$ and $\tau_i = K_t\omega_i^2$. Since Round Per Minute (RPM) sensors are not yet compatible with the Cube flight controller used in our flight tests, it is difficult to measure the motor coefficients K_f and K_t directly. However, since the Pulse Width Modulation (PWM) signals communicated between the autopilot and the ESCs are measured and identified, it is possible to identify a polynomial relation between the PWM signals and the generated forces and moments. These relationships can be considered as an alternative solution for the identification of the motor coefficients and a replacement for the K_f and K_t gains. To do this, we have set up two experiments: the first is used to identify the relation between the input PWM signals and the force generated by the actuator, and the second is used to identify the relation between the PWM signals and the generated torque.

3.2.1. Relationship between PWM inputs and generated thrust force

The first experiment is shown in Figure 3. It aims to find the relation between the PWM control signal and the thrust generated. The actuator and its propeller are mounted on top of a single bar attached to a base support. When the actuator starts running, the weight scale under the support measures the added thrust force generated by the actuator. To identify the relation between the PWM signal and the thrust generated, we implemented a Matlab application to send PWM signals to the ESC, starting from 1000 up to 2000 with a step of 50. A value of 1000 means that no signal is sent to the motor, thus the motor is turned off, and a value of 2000 means that we run the motor at full power.

After collecting data, we obtain the curve of the thrust forces generated in Newton (N) with respect to each PWM signal sent to the motor u_{pwm}, as shown in Figure 4. Five tests of the same experiment were conducted to minimize the error due to the imperfections of the setup and the measurement noises, and to verify the repeatability of the results.

Using the *polyfit* function in Matlab, we found that a polynomial of degree three presents a good approximation of the collected data from the tests, with an approximation error of order 10^{-3}. The obtained relation is as follows:

$$f_i = (-1.4736u_{pwm}^3 + 11.0691u_{pwm}^2 - 16.7074u_{pwm} + 7.3007)/100 \tag{9}$$

Figure 3: Experimental setup for thrust identification of the Tarot 650

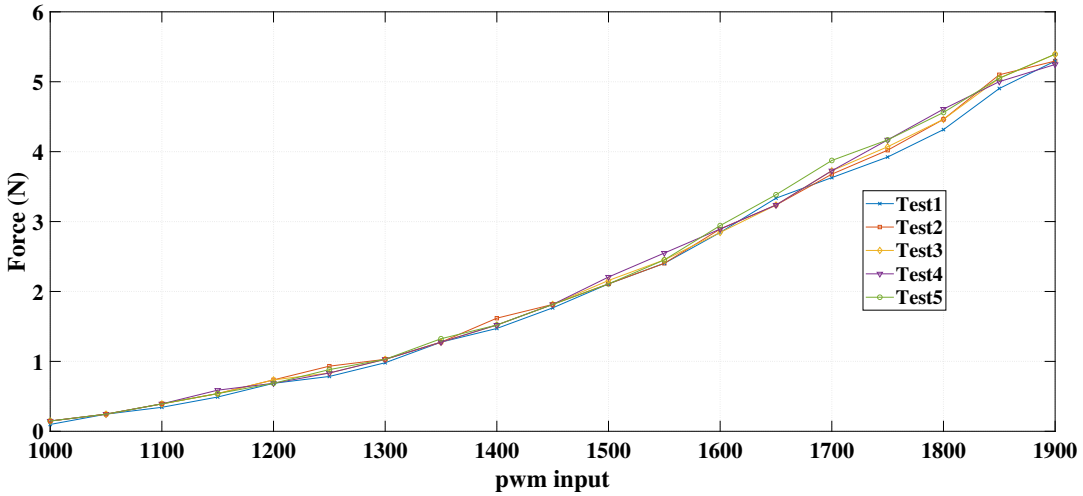


Figure 4: Thrust force as function of the applied PWM input

In hovering mode, the PWM values of motors is around 1500 and 1600 which corresponds to 2 N and 3 N of generated forces by the actuators. Note that, an approximation using a polynomial of second order gives an error of 2×10^{-2} which correspond to $1 - 2\%$ of percentage error in hovering mode. Even though this error is not huge, it adds up quickly with the passage of time, and we will get bad estimations after a short amount of time. Thus we consider this approximation unacceptable. Moreover, an approximation using a polynomial of fourth order gives an acceptable error of order 4×10^{-3}, while has a computation time significantly bigger than a polynomial of degree three which gives a similar error of 3×10^{-3}. Thus, we choose the approximation of the polynomial of third order in our model.

3.2.2. Relationship between PWM inputs and generated torque

The second experiment is shown in Figure 5. It aims to find the relation between the PWM control signal sent to the motor and the torque generated. In this setup, a small bar, with a known length of 13 cm, is attached to the motor arm and placed on top of the weight scale. When the actuator starts running, the motor arm starts to turn slightly oppositely to the rotor's direction due to the generated torque. This will induce a force on the weight scale transmitted by the small bar. Thus by multiplying this force by the length of the small bar, we obtain the torque generated by the motor.

To identify the relation between the PWM signal and the generated torque, we used the same Matlab application than in Section 3.2.1 to send PWM signals to the ESC, starting from 1000 up to 2000 with a step of 50.

After collecting data, we obtain the curve of the torque generated in Newton.meter (N.m) with respect to each PWM signal sent to the motor u_{pwm}, as shown in Figure 6. Again, five tests of the same experiment are realized to minimize errors due to measurements noises and the imperfections of the setup.

Figure 5: Experimental setup for torque identification of the Tarot 650

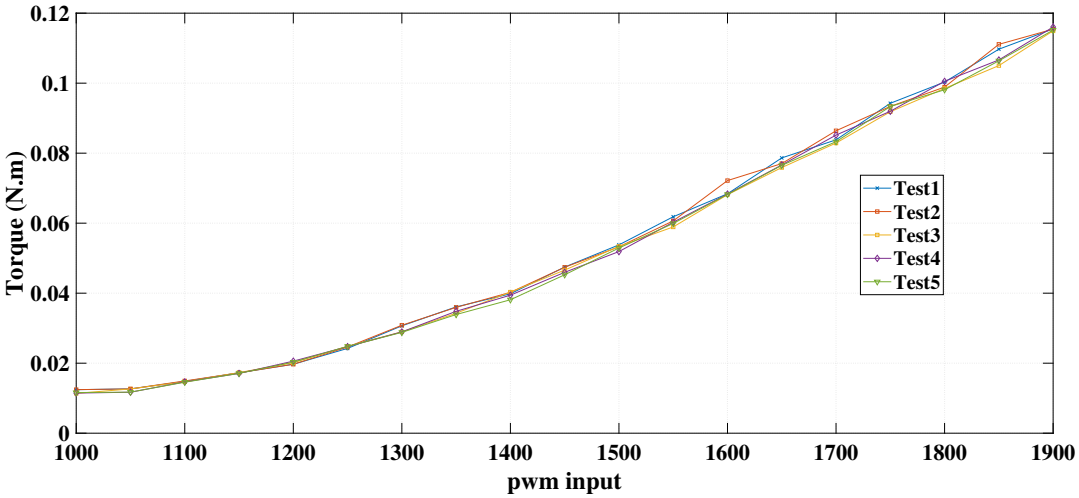


Figure 6: Motor torque as function of the applied PWM input

Using the *polyfit* function in Matlab, we found that a polynomial of degree three presents a good approximation

of the collected data from the tests, with an approximation error of order 10^{-3}. The obtained relation is as follows:

$$\tau_i = (-0.0905u_{pwm}^3 + 0.4771u_{pwm}^2 - 0.679u_{pwm} + 0.3045)/100 \tag{10}$$

Similarly as in Section 3.2.1, an approximation using a polynomial of second order gives an unacceptable error of 3×10^{-2}. An approximation using a polynomial of fourth order gives an acceptable error of 5×10^{-4}, but has a computation time bigger than a polynomial of degree three which gives a similar error of 2×10^{-4}. Thus, we choose the approximation of a polynomial of third order for our model.

3.2.3. Inertia Matrix

The moment of inertia (inertia matrix) I (see equation (11)) of a rigid body is required to calculate the torque needed to achieve a desired angular acceleration about a rotational axis. The calculation of this moment of inertia depends on the body mass distribution and the chosen axis. Assuming a perfect symmetry about the three rotational axis, the off-diagonal terms of the inertia matrix (I_{xy}, I_{xz}, I_{yz}) become zero, and the diagonal terms can be calculated using the following equations based on [38]

$$\begin{array}{c} I_{xx} = \Sigma_i m_i(d_{y_i}^2 + d_{z_i}^2) \\ I_{yy} = \Sigma_i m_i(d_{x_i}^2 + d_{z_i}^2) \\ I_{zz} = \Sigma_i m_i(d_{x_i}^2 + d_{y_i}^2) \end{array} \tag{11}$$

with $\Sigma_i m_i$ the mass of the UAV's arm (including the actuator, ESC and the arm itself) and d_{x_i}, d_{y_i} and d_{z_i} the perpendicular distances from the end of the arm to the specific axis, which are the same in each case since the structure is symmetrical. In the case of the Tarot 650, by applying the equation 11, we get the following values:

$$\begin{array}{c} I_{xx} = I_{yy} = 3.38 * 10^{-2}\ Kg.m^2 \\ I_{zz} = 2.25 * 10^{-2}\ Kg.m^2 \end{array} \tag{12}$$

4. Data fusion using Extended Kalman filter

The Kalman filter assumes that the system states and the measurements of the sensors can be described by a linear dynamic system. This dynamic system is divided into two parts:

- The linear model which describes the evolution of the system states over time.
- The measurement model which describes how the measurements are related to the states.

Thus, the Kalman filter assumes that the system can be represented by linear equations. When the system is nonlinear, we can use linearization techniques to transform the problem into a linear problem, using the Extended Kalman filter (EKF). A good introduction to the topic can be found in [39]. A more detailed description of the concept, derivation and properties is given in [40].

In the following, we recall the well known equations of Kalman filters that will be extensively used in our architecture.

4.1. Kalman filter for linear systems

The Kalman filter, also known as linear quadratic estimator, is an optimal estimator for discrete linear system of the form:

$$\boldsymbol{x_{k+1}} = \boldsymbol{A_k x_k} + \boldsymbol{w_k} \tag{13}$$

where $\boldsymbol{x_{k+1}} \in \mathbb{R}^n$, $\boldsymbol{x_k} \in \mathbb{R}^n$ represent the system states respectively at the instants $k+1$ and k, $\boldsymbol{A_k} \in \mathbb{R}^{n \times n}$ is the transition matrix between $k+1$ and k and $\boldsymbol{w_k} \in \mathbb{R}^n$ is the noise vector in the states model.

On the other hand, the observation model which describes the measurement of the sensors is described by:

$$\boldsymbol{z_k} = \boldsymbol{H_k x_k} + \boldsymbol{v_k} \tag{14}$$

where $\boldsymbol{z_k} \in \mathbb{R}^p$, $\boldsymbol{H_k} \in \mathbb{R}^{p \times n}$ and $\boldsymbol{v_k} \in \mathbb{R}^p$ are respectively the measurement vector at instant k, the observation matrix and the noise vector in the observation model.

The process noise $\boldsymbol{w_k}$ and the measurement noise $\boldsymbol{v_k}$ are assumed to be independent random variables with Gaussian probability density functions and zero mean value. The normal probability distributions p are as follows:

$$p(w) \backsim N(0, Q), \boldsymbol{Q} = diag(\sigma_{w1}^2, \sigma_{w2}^2, ...,) \tag{15}$$

$$p(v) \backsim N(0,R), \boldsymbol{R} = diag(\sigma_{v1}^2, \sigma_{v2}^2, ...,) \tag{16}$$

with σ^2 being the variance of the corresponding noise distribution.

Kalman filter process

The first step (prediction step) is to determine the predicted state $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-$ at time k, using the previous corrected state $\hat{\boldsymbol{x}}_{\boldsymbol{k-1}}$, and the associated error covariance matrix $\boldsymbol{P}_{\boldsymbol{k}}^-$ using:

$$\begin{aligned} \hat{\boldsymbol{x}}_{\boldsymbol{k}}^- &= \boldsymbol{A_k}\hat{\boldsymbol{x}}_{\boldsymbol{k-1}} \\ \boldsymbol{P}_{\boldsymbol{k}}^- &= \boldsymbol{A_k}\boldsymbol{P}_{\boldsymbol{k-1}}^+\boldsymbol{A}_{\boldsymbol{k}}^{\boldsymbol{T}} + \boldsymbol{Q_k} \end{aligned} \tag{17}$$

The second step (correction step) is to correct the predicted states and the covariance matrix using the measurements from the sensors $\boldsymbol{z_k}$ when they are available. The predicted state can be corrected using the innovation (or residual) $\hat{\boldsymbol{S}}_{\boldsymbol{k}}$ and the computed Kalman gain $\boldsymbol{K_k}$. Thus, the corrected state $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ and its associated covariance matrix $\boldsymbol{P}_{\boldsymbol{k}}^+$ can be estimated using the following equations:

$$\begin{aligned} \hat{\boldsymbol{S}}_{\boldsymbol{k}} &= \boldsymbol{z_k} - \boldsymbol{H_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^- \\ \hat{\boldsymbol{S}}_{\boldsymbol{k}} &= \boldsymbol{H_k}\boldsymbol{P}_{\boldsymbol{k}}^-\boldsymbol{H}_{\boldsymbol{k}}^{\boldsymbol{T}} + \boldsymbol{R_k} \\ \boldsymbol{K_k} &= \boldsymbol{P}_{\boldsymbol{k}}^-\boldsymbol{H}_{\boldsymbol{k}}^{\boldsymbol{T}}\hat{\boldsymbol{S}}_{\boldsymbol{k}}^{\boldsymbol{-1}} \\ \hat{\boldsymbol{x}}_{\boldsymbol{k}} &= \hat{\boldsymbol{x}}^- + \boldsymbol{K_k}\hat{\boldsymbol{S}}_{\boldsymbol{k}} \\ \boldsymbol{P}_{\boldsymbol{k}}^+ &= \boldsymbol{P}_{\boldsymbol{k}}^- \boldsymbol{K_k}\boldsymbol{H_k}\boldsymbol{P}_{\boldsymbol{k}}^- \end{aligned} \tag{18}$$

The innovation $\hat{\boldsymbol{S}}_{\boldsymbol{k}}$ represents the difference between the predicted states and the real measurement from the sensors.

4.2. Extended Kalman filter EKF for nonlinear systems

For the estimation of a nonlinear system, several extended versions of the Kalman filter exist. A widely used approach is to linearize the system dynamics in every step around the a priori estimation $\boldsymbol{x}_{\boldsymbol{k}}^-$, and proceed as for a linear system. This approach is known as the EKF. The fundamental imperfection of the EKF, as pointed out in [41], is that the distributions of the various random variables are no longer a normal distribution, after undergoing their respective non-linear transformations. Thus, the optimality of the estimation is only approximated by linearization. The stochastic system equations from (17) and (14) are now generalized to the nonlinear case as:

$$\boldsymbol{x_{k+1}} = \boldsymbol{f(x_k, w_k)} \tag{19}$$

again with the state vector $\boldsymbol{x_k} \in \mathbb{R}^n$, and the observation model is given by:

$$\boldsymbol{z_k} = \boldsymbol{h(x_k, v_k)} \tag{20}$$

with the measurements vector $\boldsymbol{z_k} \in \mathbb{R}^m$.

Extended Kalman filter process

By deriving the Jacobian matrix of the partial derivatives of $\boldsymbol{f(x_k, w_k)}$ and $\boldsymbol{h(x_k, v_k)}$ with respect to the state x_k and the noise vectors $\boldsymbol{w_k}$ and $\boldsymbol{v_k}$, we obtain the linearized approximation of the matrices:

$$\begin{aligned} \boldsymbol{A_k} &= (\tfrac{\boldsymbol{\partial f(x_k, w_k)}}{\boldsymbol{\partial x_k}})^{\boldsymbol{T}}|_{\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-} \\ \boldsymbol{W_k} &= (\tfrac{\boldsymbol{\partial f(x_k, w_k)}}{\boldsymbol{\partial w_k}})^{\boldsymbol{T}}|_{\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-} \\ \boldsymbol{H_k} &= (\tfrac{\boldsymbol{\partial h(x_k, v_k)}}{\boldsymbol{\partial x_k}})^{\boldsymbol{T}}|_{\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-} \\ \boldsymbol{V_k} &= (\tfrac{\boldsymbol{\partial h(x_k, v_k)}}{\boldsymbol{\partial v_k}})^{\boldsymbol{T}}|_{\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-} \end{aligned} \tag{21}$$

The EKF is implemented as shown in Figure 7, and a detailed process can again be found in [41].

5. Fault-tolerance architecture for data fusion system

In this section, we present a fault-tolerance architecture for data fusion targeting sensors and software faults on an outdoor quadrotor UAV. This architecture extends the duplication-comparison technique described in [1], with the weighted average voting system proposed in [42]. This architecture uses data fusion with Kalman filters in order to

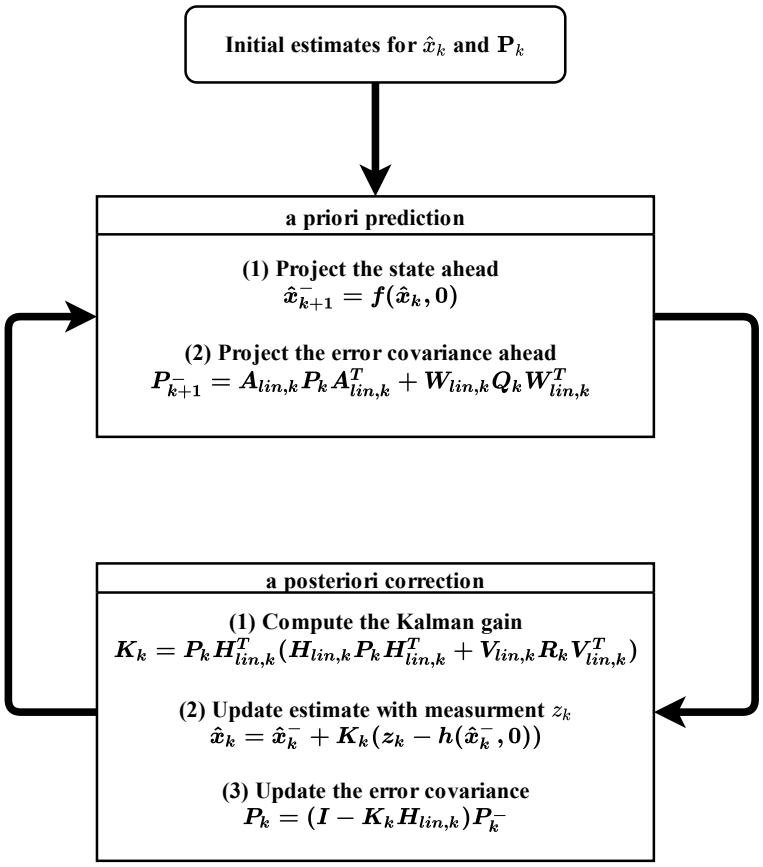


Figure 7: Data flow of the Extended Kalman filter operation

estimate the states (position and orientation) of the UAV. Four main additions have been made between our proposed architecture and the one proposed in [1]: (1) The first difference is that we add an analytical redundancy using the dynamic model of the system (DM). (2) The second difference is that we use a weighted average voter instead of a simple thresholding, which increases the accuracy of the outputs and the error detection process. (3) The third difference is that we propose multiple solutions which can be applied to recover a faulty system and this increases the flexibility of our architecture. (4) The fourth difference is that our architecture is experimentally validated on a closed loop quadrotor UAV during real outdoor experiments, instead of on an open loop using real data. The proposed architecture in our work is illustrated in Figure 8. All its modules will be detailed in the following sections: first the voting system used to calculate the final estimation of the redundant components, second the error detection and fault diagnosis mechanisms and finally the system recovery techniques. Please note that in our work, we consider only single or successive faults as simultaneous faults are always extremely difficult to accurately identify and even possibly detect. Note however that common cause faults, the most common simultaneous faults, can be avoided through diversification processes, which is applied both on hardware and software in our fault tolerant architecture (except for the GPS sensors, which should also be diversified as much as possible).

5.1. Weighted Average Voting System

As seen in Figure 1, the fault tolerance architecture for data fusion proposed in [1] uses redundant sensors blocks to produce diversified fusion outputs, and a voter to detect errors and produce a unique output. However, the voter is a fairly common threshold comparator that averages the output of the healthy sensors blocks and detects errors by a thresholding comparison. We propose to use an updated version of this voter in order to improve the system's behavior both without faults and with faults prior to the error detection module, by weighting the output of each sensors block with its consistency with the other's.

Considering a redundant system using multiple diversified but functionally identical modules operating in parallel, the weighted average voter [42] gives a weighted mean of values obtained from the redundant modules. Given a set of inputs from three diversified modules x_{m1}, x_{m2}, and x_{m3} for a particular cycle, the weighted average voter determines first a numerical distance of input pairs: $d_{12} = |x_{m1} - x_{m2}|$, $d_{13} = |x_{m1} - x_{m3}|$ and $d_{23} = |x_{m2} - x_{m3}|$. Out of these values, the weighting values of individual inputs, w_{m1}, w_{m2} and w_{m3} are obtained: in practice, a module's result far

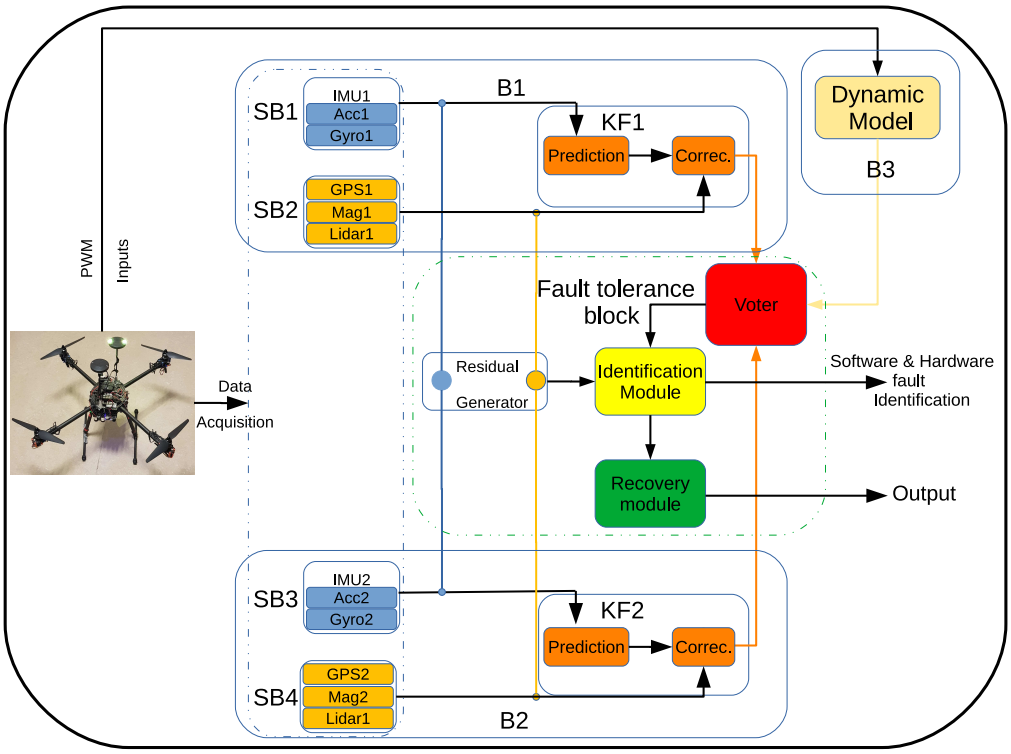


Figure 8: Proposed data fusion architecture for tolerating sensors and software faults

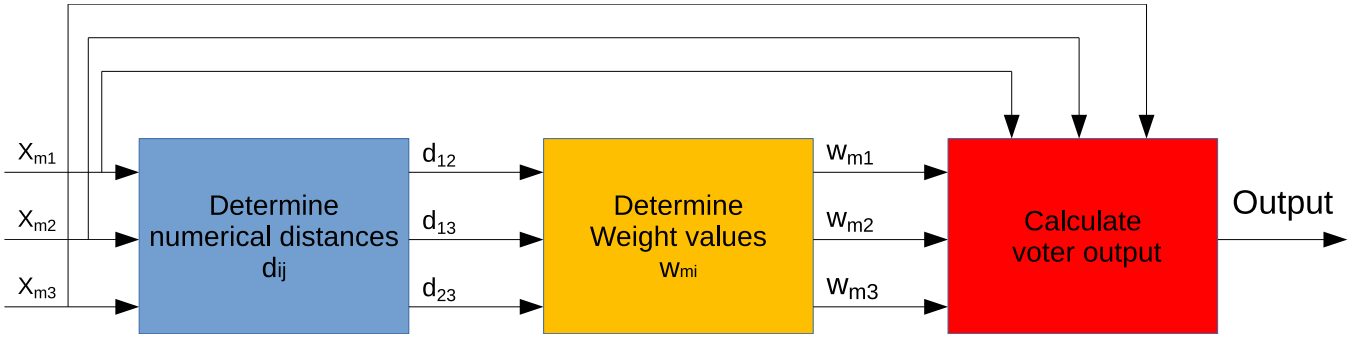


Figure 9: A 3-input weighted average voter

from the other modules results would be assigned a lower weight. The weight values are then used to calculate a single value as the voter output (Figure 9). We detail in the rest of this section how the weights and the final output are computed.

Voter implementation

In order to determine the consistency of all voter input pairs, the weighted average voter uses the concept of the soft threshold [42]. For any voter input pairs i and j, the agreement indicator s_{ij} is defined as follows:

$$s_{ij} = \begin{cases} 1, & \text{if } d_{ij} \leqslant a \\ (\frac{n}{n-1})(1 - \frac{d_{ij}}{na}), & \text{if } a < d_{ij} \leqslant na \\ 0, & \text{if } d_{ij} \geqslant na \end{cases} \tag{22}$$

where d_{ij} is the distance between the input pairs i and j, a is the fixed threshold of the voter, and n is another positive tuneable thresholding parameter.

If the distance d_{ij} of input pairs is less than the threshold a, the agreement indicator is $s_{ij} = 1$. This means that the measurements of the modules i and j are aligned and consistent for the specific application. Oppositely, if the distance d_{ij} of input pairs is more than the threshold $n \times a$, then the agreement indicator becomes $s_{ij} = 0$, which

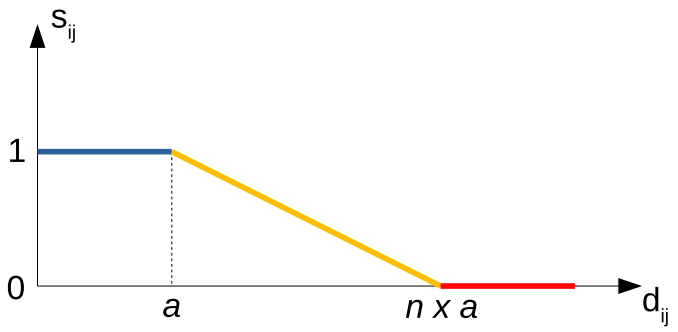


Figure 10: Agreement indicator in function of the distance between input pairs

means that the measurements of the modules i and j are inconsistent. For input pairs with a distance between a and $n \times a$, the agreement indicator varies in the range $[0, 1]$ as shown in Figure 10.

After calculating the agreement indicator for each input pairs, the next step is to calculate the weighted values w_{mi} for each module i based on the following equation:

$$w_{mi} = \frac{\sum_{j=1, j \neq i}^{k} s_{ij}}{k-1} \tag{23}$$

where k is the number of diversified modules.

The voter output is calculated as follows:

$$y_v = \frac{\sum_{i=1}^{k} x_{mi} \times w_{mi}}{\sum_{i=1}^{k} w_{mi}} \tag{24}$$

The tuneable parameter n has a significant impact on the behavior of the voter. In our case, we will also use the voter as a detection error mechanism, considering that an error is present in a sensor when an agreement indicator reaches zero. Thus, as n increases, the chances of getting undetected errors increases, while when n tends to 1, the voter behaves as a common voter with a fixed hard threshold value a, where the agreement indicator s_{ij} is either 0 or 1, which increases the chances of getting false positives.

5.2. Enhanced data fusion architecture for tolerating sensor and software faults

Data fusion can be used to reduce sensor noise and errors and to achieve and optimize a solution in case of multiple sensors. However, even though data fusion architectures are robust to some temporary outliers (like sensor noises) they can not usually tolerate prolonged sensor faults. They are also vulnerable to software faults, particularly on gains or covariances in the data fusion mechanism.

In this section, we present our proposed data fusion architecture for tolerating sensors and software faults which is based on duplication and comparison and was initially proposed in [1]. We also use a redundant predictive block based on a mathematical model of the UAV for both error detection and diagnosis and the weighted average voting system presented in section 5.1. This architecture will be validated in section 6 on a real outdoor UAV: the *Tarot*650 quadrotor.

The proposed fault tolerance architecture is shown in Figure 11. This architecture provides detection and recovery services adapted to multi-sensor data fusion systems to ensure their reliability and the safety of the UAV.

The main differences between our architecture (Figure 11) and the architecture proposed in [1] are summarized as follows:

- *Branches*: in our architecture we proposed to add an analytical redundancy using the dynamic model (DM) and the equations of motion of the UAV, whereas in [1], all the branches are formed only using available hardware (sensors).
- *Voter*: in our architecture we used an average weighted voter to calculate the output of the system, whereas in [1], a simple thresholding is used instead.

- *Recovery*: in our architecture, we propose several solutions to recover a faulty system by altering the outputs of the sensor blocks, whereas in [1], the solutions were limited to using the identified healthy branch.
- *Validation*: our architecture is validated in real outdoor flights, whereas the architecture in [1] was only validated in an open loop using data sets from real experiments.

In our generic architecture, we implement two parallel and independent data fusion branches, each containing a data fusion block (here KF1 and KF2) that estimates the state of the UAV from redundant and diversified sensor blocks. Each sensor block (SB1, SB2, SB3 and SB4) may contain one or several sensors. In the Figure 11, we consider that the sensor blocks SB1 and SB3 contain functionally similar sensors measuring the same state's variables. Similarly, the sensor blocks SB2 and SB4 are functionally equivalent. Note that more than two independent branches could be used if the required resources and space are available, which ultimately would allow to tolerate more successive faults. In the residuals generator block we calculate the residuals from the redundant sensors data which represents the differences between the output of these redundant sensors. Moreover, the dynamic model module in the Figure 11 represents the analytic redundancy in our architecture, where we use the equations of motion of the UAV and the control input (thrust and torques) of the motors. The output of all the mentioned modules are then fed into the final module which is called the software/hardware error detection, isolation and recovery module.

This architecture can tolerate one or more successive hardware faults related to sensor blocks (as long as we still have redundancies in functionally equivalent sensors) and allows to tolerate a software fault related to the fusion blocks, under the assumption of no simultaneous sensor's failure.

5.3. Fault detection

We propose in this section an implementation example of our architecture with three parallel branches, two of them executing a data fusion process and using each two sensor blocks similarly to Figure 11. The first branch (SB1, SB2, KF1) combines the outputs of sensor blocks SB1 and SB2 within the KF1 fusion block, the second branch (SB3, SB4, KF2) combines the outputs of sensor blocks SB3 and SB4 at the KF2 fusion block, and the third branch is the Dynamic model which uses the equations of motion of the quadrotor UAV. For the sensor blocks (SB1, SB2) and (SB3, SB4), the outputs of each component are compared with its redundant component: the outputs of SB1 are compared with the output of SB3, while the outputs of SB2 are compared with the output of SB4. Also, the outputs of the fusion blocks KF1 and KF2 and of the Dynamic model (DM) are compared and combined using the weighted average voter. The comparison between the fusion blocks allows to detect an error in the system, while the comparisons between the sensor blocks are used to diagnose the detected error. In case of no fault in the system, the output of the data fusion system is the same as the output of the voter which represents a combined solution of the three estimations of the state vector. In the following we explain the error detection and identification algorithm. Note that a third sensor branch could be used instead (or additionally) to the Dynamic Model and could allow to tolerate more hardware faults, but would obviously be more costly and limit the payload of the UAV.

Our architecture is made up of three modules (as shown in Figure 12). In the following, we define each module the two modules used respectively for error detection and identification:

1. *Voter and error detection*: The error detection is done by computing the values of the agreement indicators s_{ij} of the outputs of the two fusion blocks KF1 and KF2, and of the Dynamic model (DM). The indicators are computed using the equation (22), where the indices 1, 2 and 3 represent respectively the outputs of KF1, KF2 and DM. When the indicator s_{12} goes to zero, this implies the occurrence of an error in the system. This error is either due to a software fault of the data fusion blocks KF1 and KF2, or to a hardware fault in one of the sensor blocks. Using these agreement indicators, we can also identify the erroneous branch thanks to the redundancy introduced by the DM. When $s_{12} = 0$, we compare the values of s_{13} and s_{23}. If $s_{13} = min(s_{13}, s_{23})$, and $|s_{13} - s_{23}| > \epsilon$ (a threshold value delta), then the error is in the first branch. If $s_{23} = min(s_{13}, s_{23})$ and $|s_{13} - s_{23}| > \epsilon$, the error is in the second branch. Finally, if $|s_{13} - s_{23}| \leqslant \epsilon$, we can only conclude that the fault does not yet affect the system sufficiently to be identified, and we will continue checking this condition in the next cycles until we identify the erroneous branch.
   In case where the indicators s_{13} and s_{23} go to zero with $s_{12} = 1$, this implies that there is an error in the estimation of the DM. Such error can be due to high uncertainties in the system, which can happen because of unusual environmental conditions or divergence due the model imperfections. In this case, on possible solution is to reset the output of the DM to the average value of the outputs of KF1 and KF2, under the assumption that no other fault impacts the sensors. We could also remove the erroneous DM from the system, but we would then only be able to detect hardware faults, and no longer tolerate them as we would be unable to identify the correct branch.

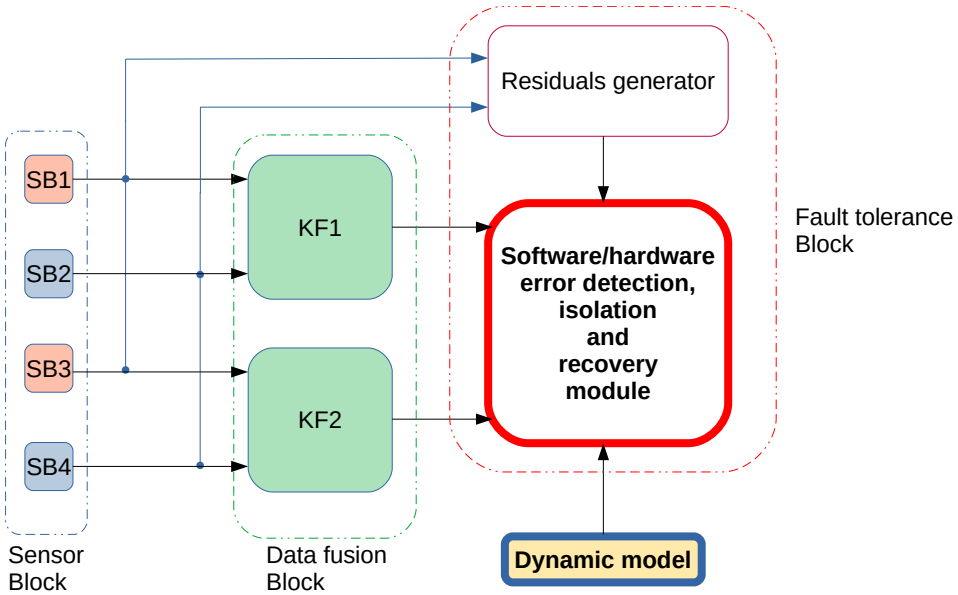


Figure 11: Fault tolerance architecture using Duplication comparison and analytical redundancy

2. *Software/hardware error identification module*: This module diagnoses the detected error, and identifies whether it is a software fault in the data fusion process or a hardware fault of a sensor block. In practice, the comparison of sensors outputs allow us to determine whether the detected error is hardware or software, and to identify the erroneous sensor in the case of a hardware fault. The residuals are computed as the differences between the outputs of the redundant sensors blocks (SB1/SB3 and SB2/SB4 in our implementation). If the output of a sensor block deviates significantly from its redundant block then the system diagnoses a hardware error on one of these two sensors. More precisely, a hardware error is detected if the value of the residual $\Delta_{SB1,SB3}$ or $\Delta_{SB2,SB4}$ exceeds the values of the fixed thresholds Th_{13} and Th_{24}, which are depending on the application. If the value of all residuals are below their corresponding threshold, a software error is diagnosed either in KF1 or KF2. Note that if these thresholds were too high or too low, this may cause respectively absences of error detection or false positives. The value of these thresholds are determined by taking into account the precision of the onboard sensors. For example, for the low cost GPS l used in our $Tarot650$, a value of 2 meters is acceptable. On the other hand, if the value of the residuals are below the predefined thresholds, we diagnose a software error in one of the data fusion blocks KF1 and KF2.
The operation of the fault tolerance algorithm is summarized in the algorithm 1 and illustrated in Figures 13 and 14

5.4. Recovery module

Once the error in the system is detected on the voter level and isolated by the identification module, an appropriate system recovery solution must be applied accordingly in order to re-stabilize the data fusion system. In the following, some of the possible solutions are presented.

5.4.1. Recovery for Hardware Fault

In the case of a hardware fault in component i of KF1, many solutions may be implemented in order to reconfigure the system after the error detection and identification process. However, we identify two main solutions as follows:

- **Sol 1**: the first solution consists in removing the component i from KF1 and replacing it with the equivalent component j from KF2. The newly defined block KF3 (KF1 with component j) is then integrated into the

Algorithm 1: Fault detection and identification algorithm

Data: $KF1, KF2$: data fusion output
$SB1, SB2, SB3, SB4$: sensor blocks outputs
Constant: $\epsilon, Th_{24}, Th_{13}$

```
begin
    Calculate s_12, s_13, s_23;
    Calculate Δ_{SB1,SB3} and Δ_{SB2,SB4};
    if s_12 = 0 then
        if |s_13 − s_23| > ε then
            if Δ_{SB1,SB3} > Th_13 then
                if s_13 = min(s_13, s_23) then
                    /* error in SB1, apply adapted hardware recovery mechanism */
                else
                    /*error in SB3, apply adapted hardware recovery mechanism */
            else if Δ_{SB2,SB4} > Th_24 then
                if |s_13 − s_23| > ε then
                    if s_13 = min(s_13, s_23) then
                        /* error in SB2, apply adapted hardware recovery mechanism */
                    else
                        /*error in SB4, apply adapted hardware recovery mechanism */
            else if s_13 = min(s_13, s_23) then
                /* error in KF1, apply adapted software recovery mechanism */
            else
                /* error in KF2, apply adapted software recovery mechanism */
        else
            /* cannot identify the faulty branch, keep running without modifications */
    else
        /* No error detection, system healthy */
```

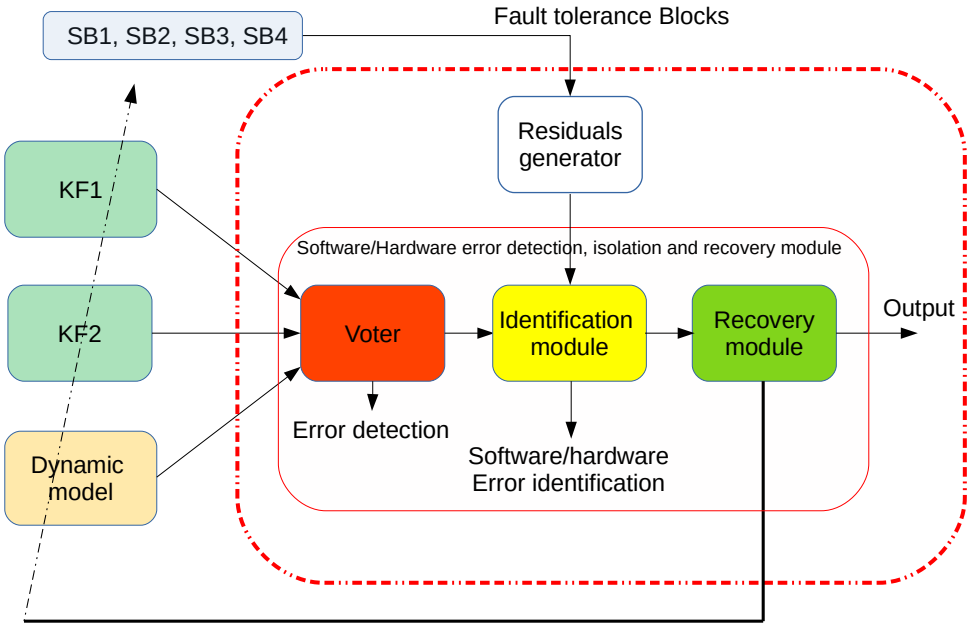


Figure 12: Software/hardware error detection, isolation and recovery module

voting mechanism along with KF2 and DM. Henceforth, using this technique, it is possible to tolerate all software faults and hardware faults except on component j which cannot be even detected anymore. This is because if the component j becomes faulty, it will affect KF2 and KF3 simultaneously, and by following the diagnostic logic of 1, we can see that a fault in DM will be detected because it's equivalent output will defer from the coherent estimations of KF2 and KF3 and a false alarm will be reported by the error detection process.

- **Sol 2**: the second solution consists in removing the branch containing KF1 from the architecture and using only the remaining healthy branches KF2 and DM. Moreover, the healthy components of KF1 as backup resources and keep comparing their output with those of KF2. Henceforth, it is possible to detect all possible faults although we can only tolerate the errors on the healthy components only (all components except component j).

- **Sol 3**: the third solution consists in removing the branch containing KF1 and replacing it by a virtual estimation branch KF3 where the output of KF3 is a weighted average of the outputs of KF2 and DM, where the weight of the KF2's output is greater than the DM's output. This may increase the precision of the estimation in case of increasing uncertainties in the DM calculations due to unmodeled effects. The same properties of sol 2 are available here, however the detection time becomes greater in this case after a new fault in the system as the third branch is a combination of the other two.

Note that if the hardware faults is in KF2, the same solutions mentioned above can be applied by removing KF2 instead of KF1. Once the appropriate solution for the intended application is selected, the recovery module can be easily developed as its code is quite simple.

5.4.2. Recovery for Software Fault

In the case of software fault in KF1, again many solutions may be proposed in order to reconfigure the system after the error detection and identification process. However, we identify two main solutions as follows:

- **Sol 1**: the first solution consists in initializing a new data fusion block KF3 with the current states values of the healthy block KF2. Also, the same code section of the KF2 is used in order to perform the estimations in KF3, since no software fault is yet activated in KF2. Henceforth, using this technique, it is possible to detect

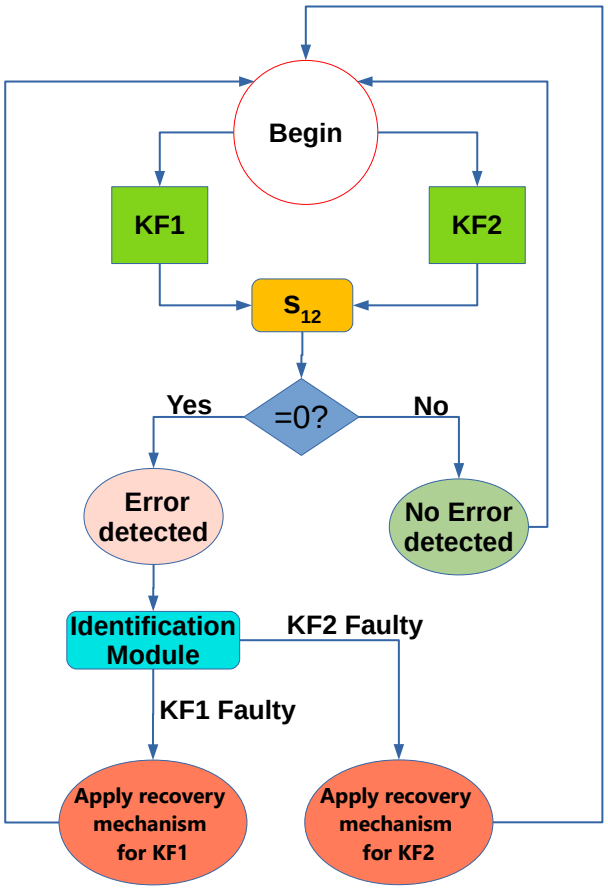


Figure 13: Proposed data fusion architecture for tolerating sensors and software faults

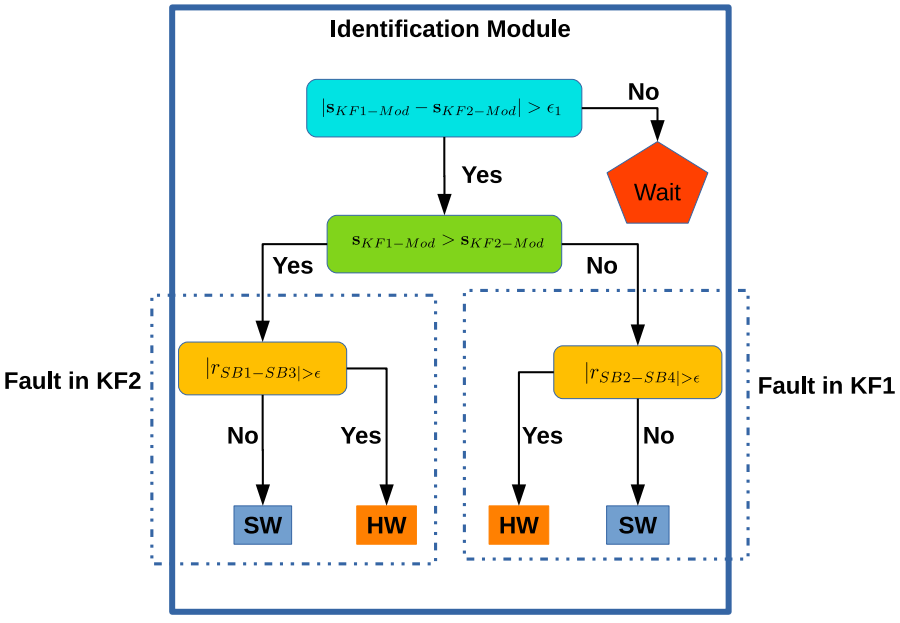


Figure 14: Proposed data fusion architecture for tolerating sensors and software faults

and tolerate all hardware faults in the system although software faults cannot be guaranteed to be detected anymore. This is because a new software fault may affect the shared code section between KF2 and KF3 and thus leads to a simultaneously distortion in both estimations, and by following the diagnostic logic of 1, we can see that a fault in DM will be detected because it's output will defer from the coherent estimations of KF2 and KF3 and a false alarm will be reported by the error detection module.

- **Sol 2**: the second solution consists in removing all the branch containing KF1 from the architecture and using only the remaining healthy branches KF2 and DM. Moreover, the healthy components of KF1 can be kept as backup resources to keep comparing their output with those of KF2. Henceforth, it is possible to detect and tolerate all the harware faults in the system, however we can only detect software faults without the ability to tolerate them. It is also possible to identify a virtual estimation branch KF3 by using the average of the outputs of KF2 and DM as the output of KF3. This may increase the precision of the estimation in case of increasing uncertainties in the DM calculations due to unmodeled effects.

- **Sol 3**: the third solution consists designing during the development phase a third diversified Kalman filter software block KF3. The added KF3 in the system is only performing mathematical calculations during operations without its outputs being used in the data fusion mechanism. Once a software fault in KF1 is detected, it is replaced by the diversified KF3. Using this solution, all the detecting and tolerance capacities in the system are retained after the first software fault. Note however that fault detection in KF3 must be done from the start of the operation even if its outputs are not immediately used, as it could become erroneous before KF1 or KF2 and should obviously not be used in this case.

Again once the appropriate solution for the intended application is selected, the recovery module can be easily developed accordingly.

6. Validation

To evaluate the proposed architecture for tolerating sensors and software faults developed during this work, we have conducted the following experiments:

- A real outdoor flight test with hardware fault injection (additive fault of 5m in the x and y direction) on the GPS1 during a U-path trajectory tracking
- A real outdoor flight test with hardware fault injection (additive fault of -1m) on the Lidar used in KF1 during a hovering at 3m altitude
- A simulation of a flight with software altitude fault injection in KF1 during a hovering at 3m altitude
- A simulation of a flight with software x, y fault injection in KF1 during a U-path trajectory tracking

The real experiments were conducted using the Tarot 650 which is a commercial hobby type quadrotor UAV. The implemented UAV using this frame is shown in Figure 15. The Tarot 650 is equipped with the Hex Cube Black (FMUv3). This is an updated version of the Pixhawk controller which is an open-source and open-hardware autopilot able to run the Arducopter flight stack used to build our software. This Cube has a 32-bit ARM Cortex M4 core processor with FPU with 168 Mhz/256 KB RAM/2 MB Flash and a 32-bit failsafe co-processor. It also includes three redundant IMUs and two redundant barometers. The Cube has been equipped with additional sensors, namely the Lidar Lite v3, two GNSS modules and an RTK module used as the reference for the position measurements to evaluate our experimental results.

To our best knowledge, the model parameters of the TAROT 650 were not identified elsewhere in the literature, and we couldn't obtain these information from the manufacturers. Therefore, we had to estimate these constants using mesurement tool (numerical balance for mass, and a meter for length) and analytical expressions for inertia values. The Tarot 650 parameters are given in Table 1.

m	Mass of the vehicle	$1.7\,kg$
l	Length of the arm	$0.23\,m$
I_{xx}, I_{yy}	Inertia	$3.38 * 10^{-2}\ Kg.m^2$
I_{zz}	Inertia	$2.25 * 10^{-2}\ Kg.m^2$

Table 1: The TAROT 650 model's parameters

Figure 15: Experimental Tarot 650 quadrotor

All the simulations and experiment configurations were done using a ground control station (Mission Planner) running the Arducopter flight stack. Note that the control law used to control the UAV is a smooth second order sliding mode control law based on the super-twisting algorithm proposed in [43]. It has been proven that this control law is capable of stabilizing the UAV despite the existence of external perturbations due to wind which increases the reliability of the states estimations of the dynamic model.

Also, it is worth mentionning that in our platform, the Ardupilot is used as an open source flight code to develop our architecture and the pixhawk is used as an autopilot which supports Ardupilot. By default, Ardupilot has a built-in Extended Kalman Filter algorithm (EKF2) to consolidate the sensors data in order to estimate the vehicle position, velocity and angular orientation based on rate gyroscopes, accelerometer, compass, GPS and Lidar. In our work, we created two instances of the Ardupilot EKF core (KF1 and KF2) along with the analytical model in order to build our architecture. In both instances, we used the default settings of the EKF parameters which are related to the accelerometer, gyro and magnetometer noises since these values are recommended when using the internal IMU of the Pixhawk (InvenSense MPU9250 in our case). However, we modified the Lidar and GPS parameters to get better response. These parameters are commonly updated in practice to take into consideration the type of the chosen device and its specifications. For example, we did the following modifications to certain parameters:

- $EK2\_GPS\_TYPE$: This parameter controls how GPS is used. We set to 1 in order to use 2D velocity & 2D position (GPS velocity does not contribute to altitude estimate).
- $EK2\_RNG\_NOISE$: This is the RMS value of noise in the range finder measurement. Increasing it reduces the weight on this measurement. We have chosen the value of 0.025 meters since this is the precision of the Lidar Lite v3 that we used in the experiments.
- $EK2\_POSNE\_NOISE$: This sets the GPS horizontal position observation noise. Increasing it reduces the weight of GPS horizontal position measurements. We have chosen the value of 2.5 meters since this is the precision of the "*Here* 3" Precision GNSS Module that we used in our experiments.

6.1. Implementation of the fault tolerance architecture

The details of the implemented architecture are shown in Figure 8. In this application we implement three parallel branches (B1, B2 and B3).

The first two branches B1 and B2 are the fusion blocks, each containing two sensors blocks: SB1 and SB2 in B1, and SB3 and SB4 in B2. The first sensors blocks SB1 and SB3 are the IMU blocks (IMU1 and IMU2), they are used to estimate the prediction of the state vector. Each IMU contains the following sensors:

Estimated State	a	n
x	0.5	6
y	0.5	6
z	0.5	2
ϕ	1	3
θ	1	3
ψ	2	3

Table 2: Tuneable parameters values for selecting the thresholds of the voter

- A gyroscope (Gyro1 and Gyro2): it is used to measure the angular velocity of the UAV in the body frame. The integration of this measure gives the attitude angles (roll, pitch and yaw) of the UAV.
- An accelerometer (Acc1 and Acc2): it is used to measure the linear accelerations of the UAV. The integration of this measure gives the linear velocity, and its double integration gives the position of the UAV.

Each of the other two blocks SB2 and SB4 contains the following sensors:

- A GPS (GPS1 and GPS2): it is used to measure the absolute position of the UAV.
- A magnometer (Mag1 and Mag2): it is used to measure the heading (yaw) of the UAV.
- A lidar (Lidar1): it is used to measure the altitude of the UAV.

Note that we only use one lidar in our experimental quadrotor. Of course we would need another redundant lidar to tolerate faults on this sensor according to our architecture, but we did not have the second one on the UAV at the time of the experiments. In fact, we intended first to use a barometer as a diversified sensor to the lidar, but its performances were too poor in practice to use it.

A data fusion block combines the outputs of these sensors blocks in a Kalman filter (KF1 for SB1 and SB2, and KF2 for SB3 and SB4). The sensors blocks SB2 and SB4 are used in the correction step of the Kalman filter process, while the sensors blocks SB1 and SB3 are used for the prediction step of the Kalman filter process. The third branch B3 is the Dynamic Model block. In this block the state vector of the UAV is estimated using the dynamic model of the vehicle and the relationships between the PWM inputs and the generated thrust and torques that we identified on our UAV as follows:

$$F_i = (-1.4736u_{pwm}^3 + 11.0691u_{pwm}^2 - 16.7074u_{pwm} + 7.3007)/100 \tag{25}$$

$$\tau_i = (-0.0905u_{pwm}^3 + 0.4771u_{pwm}^2 - 0.679u_{pwm} + 0.3045)/100 \tag{26}$$

The dynamic model that we use in this chapter is the model from (6), since the experimental validations are done in an outdoor environment where wind perturbations affect the UAV.

Finally, in order to test our architecture in real outdoor experiments, we need to choose all the values of the thresholds used in the Residuals Generator block and the minimum and maximum threshold's parameters used in the voter module. The values that we used were determined after conducting several real outdoor experiments in normal conditions and are chosen empirically in order to prevent the occurrence of false alarms and undetected errors. They are given in Tables 2 and 3. For example, in case of hardware fault in one of the GPS modules, the residuals between the measured distances d_1 and d_2 measured respectively by GPS1 and GPS2 will exceed the fixed threshold Th_{23} which can be found equal to 2 m from Table 3.

Note that these parameters and thresholds are determined based on the type and accuracy of the sensors and the precision of the Kalman filter algorithm and the analytical model used to build up the architecture. Based on these information, they can be tuned empirically during experiments in order to improve the consistency between the different estimations. One way to proceed is to begin by setting up initial values based on simulations results, and conducting several flight tests, and testing different values in data replay in order to better evaluate these parameters and to avoid false alarms as much as possible.

6.2. Additive fault on GPS1

In this experiment, an additive fault of 5m in the x and y direction has been injected in the first GPS (GPS1). The UAV is required to follow a U-path trajectory starting from the initial point $(x = 0, y = 0)$ to the destination

Sensor	Measured state	Threshold	Unit
GPS	d	2	m
Magnetometer	ψ	6	deg
Lidar	z	1	m
IMU	$\ddot{x}, \ddot{y}$	0.01	m/s^2
Gyro	$\dot{\phi}, \dot{\theta}$	0.1	deg/s

Table 3: Thresholds values used in the Residual Generator block

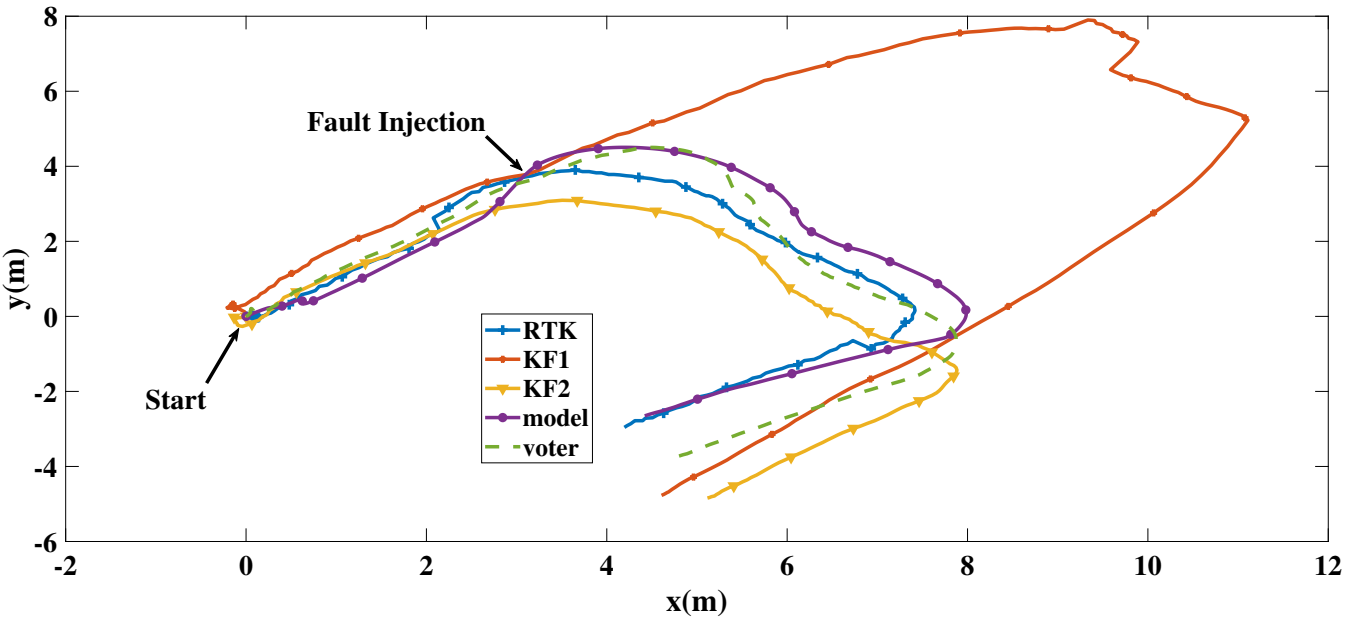


Figure 16: GPS additive fault: positions estimated by the two Kalman fi
lters, the dynamic model, the voter, and the ground truth given by a GPS RTK

point $(x = 4, y = -3)$ as shown in the Figure 16. This is the case of a trajectory tracking case, where the desired positions and velocities are time-dependent and the system is restricted to follow the U-path reference. Note that the GPS RTK is chosen as ground truth because of its precision (1cm) when receiving enough satellites data (four or more in our case).

The injected fault simulates an external fault such as a jump in the position provided by the GPS due to bounces from one of the satellite signals. This fault is generally not permanent, but it can occur for a significant period of time. Note that using the same type of GPS would usually not tolerate this fault, as it has a common cause. However, using diversified GPSs (such as Galileo, Glonass, or USA's GPS) would allow to tolerate faults due to a lack of satellites visibility in the constellation or signal rebounds. However some common cause faults, such as signal obtrusions due to a tunnel or a dense forest, would still be impossible to tolerate.

Between the instants $t_{inj} = 7s$ and $t_{inj_{end}} = 15.8s$, we added a 5 meter jump on the x_{GPS1} and y_{GPS1} components, as described in (27).

$$\begin{aligned} &x_{GPS1}(t_k) = x_{GPS1}(t_{inj}) + 5 \\ &y_{GPS1}(t_k) = y_{GPS1}(t_{inj}) + 5 \\ &\text{for } t_k \text{ such as } t_{inj_{end}} \geq t_k \geq t_{inj} \end{aligned} \tag{27}$$

Figure 16 shows the UAV positions given by the different localization systems during the first experiments: the ground truth as the output of the GPS RTK, the output of the branches B1 and B2 as the Kalman filter blocks KF1 and KF2, the output of the branch B3 given by the dynamic model, and the output of the data fusion component as the result of the voter on the three branches. The additive fault injected to the system causes the position estimated by KF1 to deviates significantly from all the other systems.

Figure 17 presents the agreement indicators of our voter during the experiment. We can see that after the fault injection at $t = 7s$ the agreement indicator between the two Kalman filter blocks $s_{dKF1-dKF2}$ starts to fall down from $s_{dKF1-dKF2} = 1$ and reaches $s_{dKF1-dKF2} = 0$ after 0.5s at $t_{det} = 7.5s$. A similar behavior can be seen for the agreement indicator between KF1 and the dynamic model. The agreement indicator between KF2 and the model

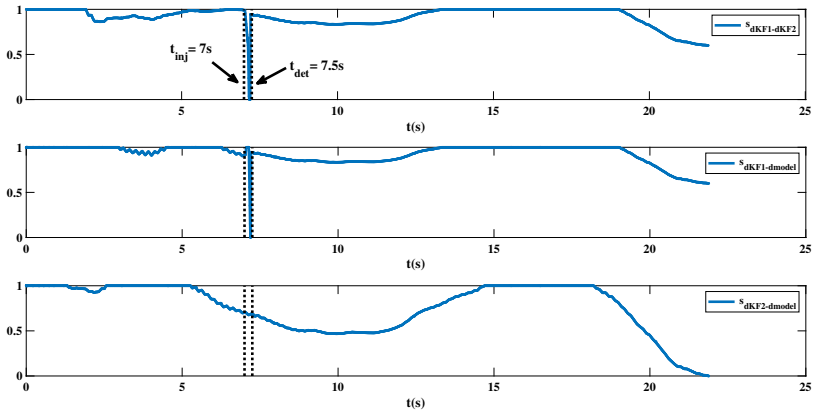


Figure 17: GPS additive fault: the agreement indicators of the Euclidean distance of the first and second Kalman filters and the model

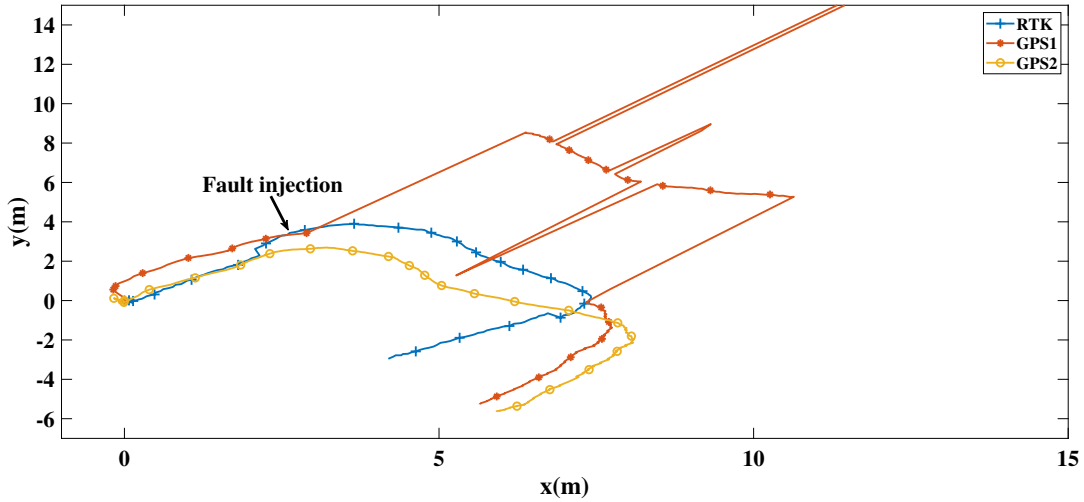


Figure 18: GPS additive fault: positions measurements of the two GPS and the RTK GPS

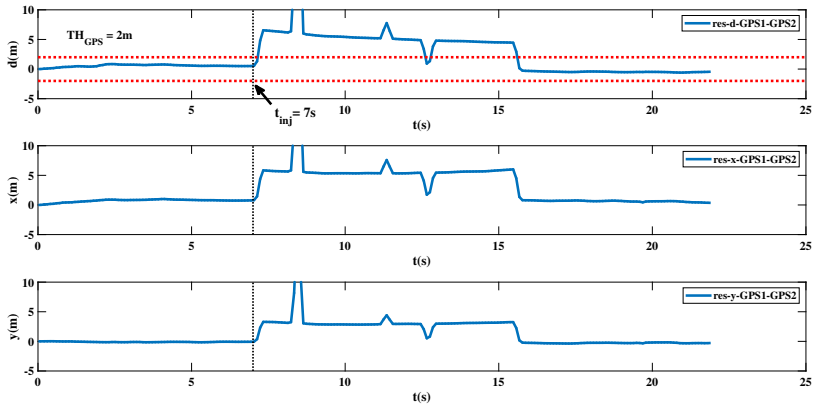


Figure 19: GPS additive fault: The residues of the x, y, d components between the first GPS and the second GPS

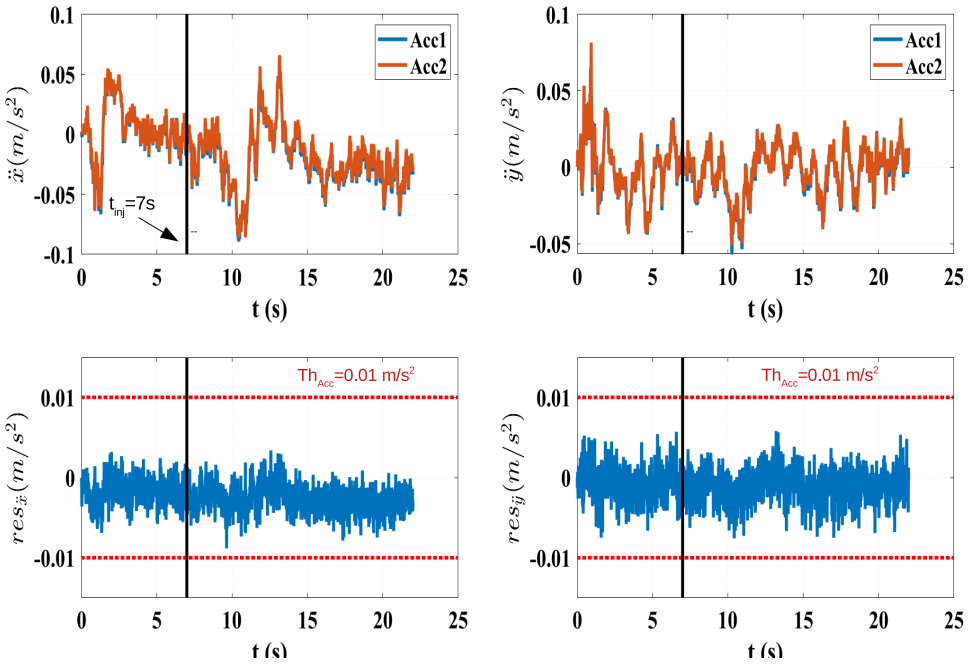


Figure 20: GPS additive fault: accelerations estimates by the two Accelerometers and their corresponding residues

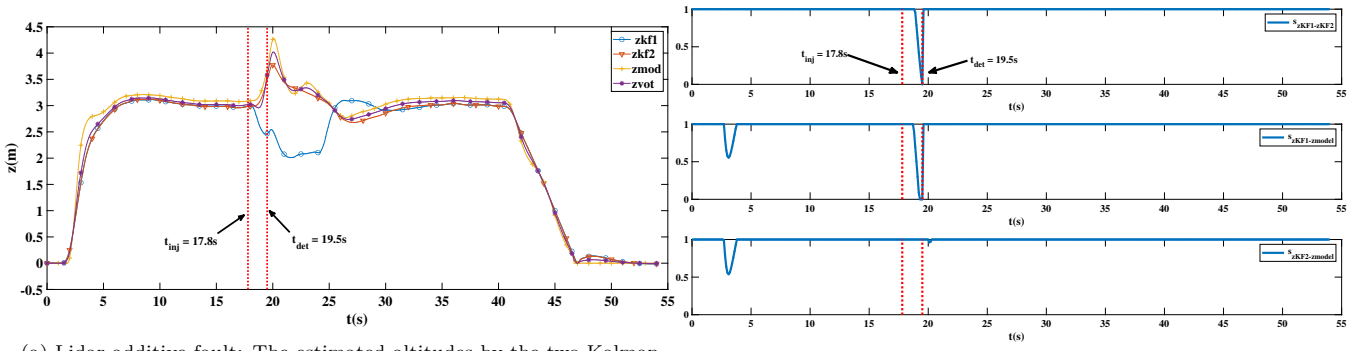
stays much higher during this period, although it drops from 1 to 0.5. This is due to the value of KF1 affecting the output of the voter until the error detection, and thus affecting the dynamic model that uses the voter's output to calculate its next position.

Following Algorithm 1, when the agreement indicator $s_{dKF1-dKF2}$ equals 0, we check which one is the more consistent to the dynamic model. Figure 17 clearly shows from the agreement indicators pertaining to the dynamic model that the erroneous one is KF1. Thus, the erroneous branch is B1 which contains KF1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. The Figure 18 shows the measured positions by GPS1 and GPS2, and for reference by the ground truth. As shown in this figure, the positions of GPS1 and GPS2 are highly different after the fault injection. Indeed, in Figure 19, we can see the residuals of the measured x, y, d components by the two GPS. In particular, the distance d between the two GPS exceeds the $Th_{GPS} = 2m$ threshold at $t_{det} = 7.5s$, thus a hardware fault is reported at $t_{det} = 7.5s$. Note that after the fault injection on GPS1, some outliers appear in the measurements of this sensor. In our opinion, we think that this is due to some corrections which are done automatically in the Ardupilot since the autopilot is reporting unusual output of the sensor which is not supposed to be unhealthy.

Since the detected error in our architecture involves the positions estimates, the outputs of the Acc1 and Acc2 are also compared since they are also used to predict the positions estimates in the Kalman Filter and could be another source of the error. The Figure 20 shows the measured accelerations $\ddot{x}$ and $\ddot{y}$ measured by Acc1 and Acc2 and their corresponding residues. It is clear that the values of the residues do not exceed the value of the predefined threshold $Th_{Acc} = 0.01m/s^2$, thus the two Acc are reported as healthy sensors. Since we have already identified the first branch as the erroneous branch in the previous step, we can conclude that the erroneous sensor is the first GPS (GPS1) in KF1.

Once the GPS1 is detected as the faulty sensor, the first branch is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the position estimated by KF2 and the Dynamic model. This recovery corresponds to the solution 3 in section 5.4.1. This explains why the agreement indicators $s_{dKF1-dKF2}$ and $s_{dKF1-dmodel}$ increases quickly after the identification of the fault in Figure 17, since KF1 has been recovered. Note that as we do not directly use the redundant sensors of SB1 and SB3, we can no longer directly tolerate another fault. However, we can still compare the outputs of the IMU, the lidars and the magnometers to detect hardware faults. We could also have used the other recovery mechanism proposed in 5.4.1, by using GPS2 instead of GPS1 in KF1.



(a) Lidar additive fault: The estimated altitudes by the two Kalman filters, the dynamic model and the voter

(b) Lidar additive fault: The agreement indicators of the altitude of the first and second Kalman filters and the model

Figure 21

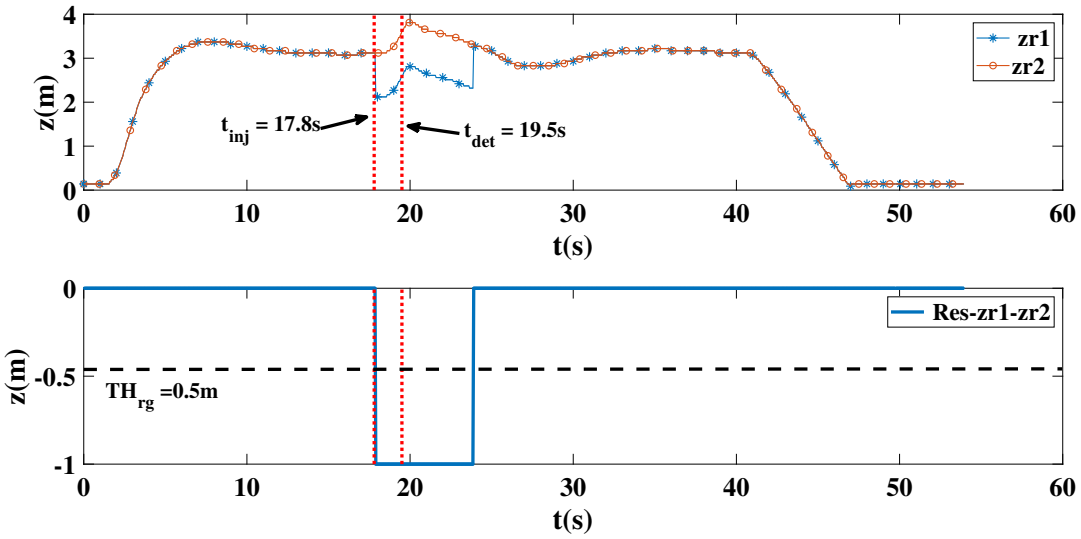


Figure 22: Lidar additive fault: The measured altitude and the residues of the Lidar's output used in KF1 and KF2

6.3. Hardware additive fault on Lidar1

In this experiment, the quadrotor is required to perform a hovering flight at a 3 meters altitude. In this experiment, the injected fault consists in adding a value of -1 meter to the Lidar1 output used in KF1. Note that because of time constraints, as previously stated in section 6.1, we only have one lidar on our UAV, which outputs are sent to both SB2 and SB4. To simulate a fault on a single Lidar, we thus only inject the fault on the data received by SB2. There is no need for an additional ground truth in this experiment since the precision of the Lidar Lite v3 equipped to the drone is below 15 cm, thus we consider it as a ground truth for the altitude estimation. The estimated altitudes by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are also depicted in Figure 21a. As can be seen, the additive fault injected to the system causes the estimated altitude by KF1 to deviate from all the other systems.

In Figure 21b, it can be noticed that after the fault injection at $t_{inj} = 17.8s$ the agreement indicator of the altitude between the two Kalman filter blocks $s_{zKF1-zKF2}$ starts to fall down from $s_{zKF1-zKF2} = 1$ and reaches $s_{zKF1-zKF2} = 0$ after 1.7s at $t_{det} = 19.5s$. As in the previous experiment, the agreement indicator between KF1 and the model behaves in the same way. However, here the agreement indicator between KF2 and the model stays at the maximum value of 1, probably because of a more conservative threshold value than for the position, thanks to the better precision of the lidar compared to the GPS.

As described in algorithm 1, an error has been detected due to the value of $s_{zKF1-zKF2}$. Figure 21b shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to diagnose if the error is due to a hardware or a software fault. Thus we compare the outputs of the Lidar used in KF1 and KF2 which are considered independant. The Figure 22 shows the measured altitudes of the Lidar $zr1$ and $zr2$ in KF1 and KF2 respectively, and their residues. We can see that the residuals of the $zr1$ and $zr2$ exceeds the $Th_{rg} = 1m$ threshold at $t_{det} = 19.5s$, thus a hardware

fault is reported at $t_{det} = 19.5s$. We do not need to compare other sensors results here, as the lidar is the only sensor determining the altitude in our data fusion. Since we have already identified that the first branch is the erroneous branch from the previous step, we can conclude that the faulty sensor is the $zr1$ in KF1.

Once $zr1$ is detected as the faulty sensor, the first branch is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the altitude estimated by KF1 and the Dynamic model. This corresponds to solution 3 in section 5.4.1. This explains why the agreement indicators $s_{zKF1-zKF2}$ and $s_{zKF1-zmodel}$ increases quickly after the identification of the fault in Figure 21b, since they no longer describe the difference between the faulty branch and the other branches in the system. As in the previous experiment, this solution has the flaws described in 5.4.1, and could have been replaced by the other method presented in the same section.

6.4. Software altitude fault

To evaluate the proposed architecture for tolerating software faults developed during this work, we present in this section the simulation results of a software fault injection described in the following.

In order to validate our architecture against a software fault in the altitude estimation, the altitude covariance term P_z is forced to a negative value of -0.1, starting from the beginning of the flight simulation described in the following. In this simulation, the quadrotor is required to perform a hovering flight at a 3 meters altitude. The estimated altitudes by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are depicted in Figure 23a. It can be noted that KF1 still gives an acceptable behavior for more than 15 seconds even if the software fault is injected before the takeoff. Indeed, development software faults are always present in the system but can still take time to be activated or cause errors. Here, the fault is immediately activated as it is in the lines of code executed by the Kalman filter, but it takes more than 15 seconds to cause an error in the system.

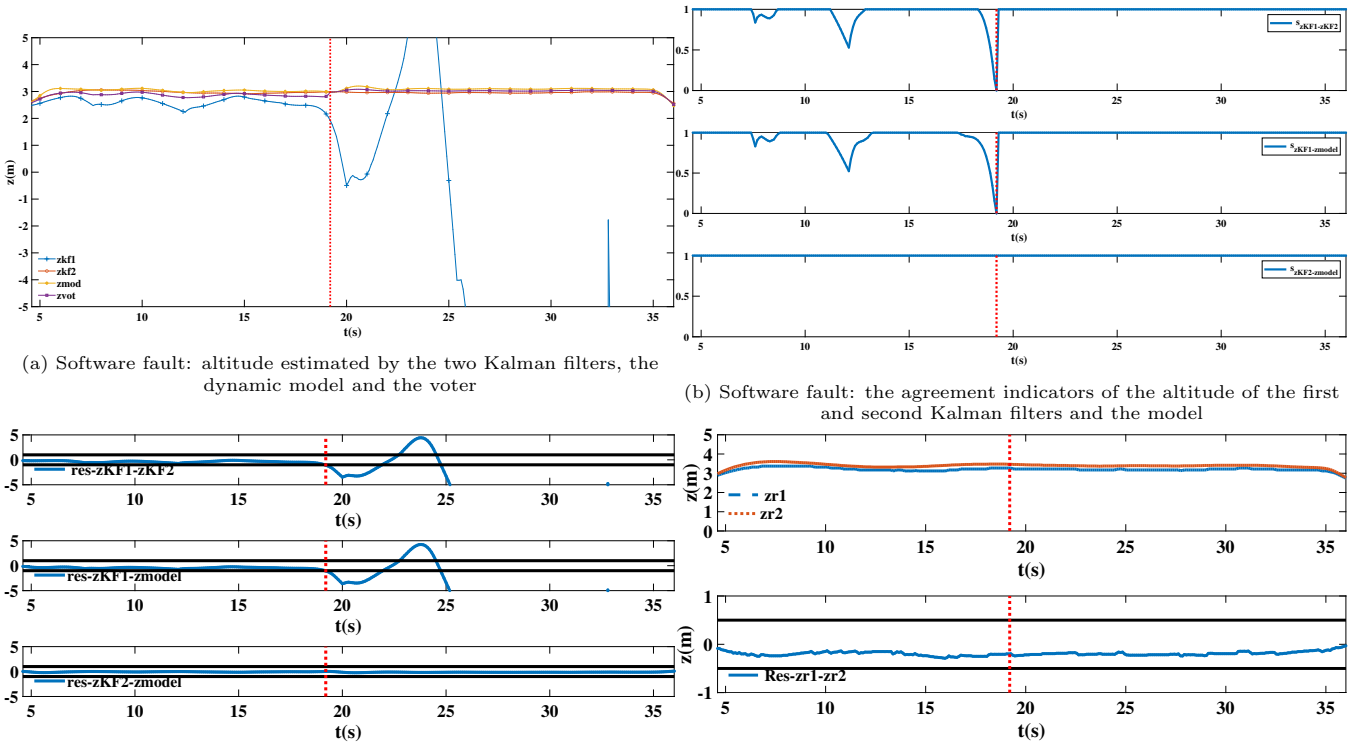
This injected fault can correspond to two real faults. First a development error in the value of this term. This can correspond to a programming error (an incorrect value due to some mistakes from the programmers) or a design error. Indeed, the matrices P and Q in Kalman filters have values that are not easy to determined and can thus be designed incorrectly. Second, it can simulate the propagation of other software errors during operation. Consider that the effect of rounding errors on the state estimation can be accounted for by calculation errors in the error covariance matrix P during the data fusion mechanism in the Kalman filter. The longer the Kalman filter has been running and the higher the iteration rate, the greater the distortion of the matrix becomes. The diagonal terms of the covariance matrix represent the estimation uncertainties relative to each state estimation and should always be positive in order to guarantee a stable estimation during the data fusion positive. However, distortions due to other software errors in the system could lead, the estimated covariance terms to become negative as in our injected fault, which will cause divergence in the estimation process.

In Figure 23b, we can see that the agreement indicator of the altitude between the two Kalman filter blocks $s_{zKF1-zKF2}$ starts to fall down from $s_{zKF1-zKF2} = 1$ and reaches $s_{zKF1-zKF2} = 0$ after 1.7s at $t_{det} = 19s$. As in the previous experiments, the agreement indicator between KF1 and the model behaves in the same way. However, here the agreement indicator between KF2 and the model stays at the maximum value of 1 as for the software attitude fault, probably because of a more conservative threshold value due to the reduced uncertainties in the simulation environment. These results are consistent with the values of the residues between the branches presented in Figure 23c, where only the residues of KF1 exceed the predefined threshold value of 0.5 m, thus indicating a fault in the first branch.

As described in algorithm 1, an error has been detected due to the value of $s_{zKF1-zKF2}$. Figure 23b shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. The Figure 23d show the measured altitudes by the first and the second Lidar. As shown in this figure, the altitude measurements of both Lidars show consistency during all the flight time. Since we do not need to compare other sensor results here, as the Lidar is the only sensor used to measure the altitude, this leads to the conclusion that no hardware fault is present in the system. Thus, a software fault is confirmed in KF1.

Once KF1 is detected as the faulty block, it is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the altitude estimated by KF1 and the dynamic model. This is another alternative solution for those presented in section 5.4.2. This explains why the agreement indicators $s_{zKF1-zKF2}$ and $s_{zKF1-zmodel}$ increases quickly after the identification of the fault in Figure 23b, since they no longer describe the difference between the faulty branch and the other branches in the system. As in the previous experiment, this solution has the flaws described in 5.4.2, and could have been replaced by the other method presented in the same section.



(a) Software fault: altitude estimated by the two Kalman filters, the dynamic model and the voter

(b) Software fault: the agreement indicators of the altitude of the first and second Kalman filters and the model

(c) Software fault: the residues of the altitude estimations between KF1; KF2 and the DM

(d) Software fault: The residues of the z components between the first simulated Lidar and the second simulated Lidar

Figure 23

6.5. Software position fault

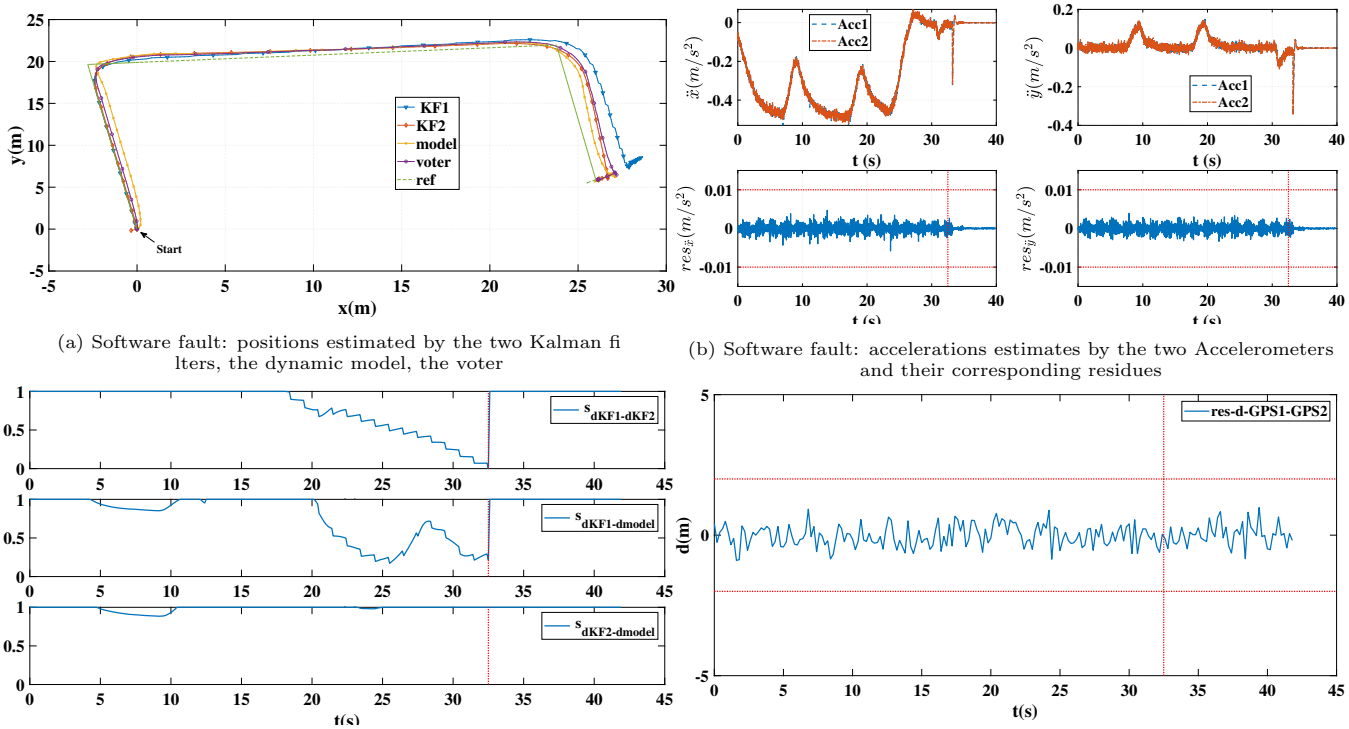
In order to validate our architecture against a software fault in the position estimation, the position covariances terms P_x and P_y representing P_x from equation 18 are forced to a small negative value of -0.01 starting from the beginning of the flight simulation described in the following. As previously said, this fault can be representative of several software development faults, particularly in the values of matrices P and Q. In this simulation, the UAV is required to follow a U-path trajectory starting from the initial point $(x = 0, y = 0)$ to the destination point $(x = 26, y = 5)$. The estimated positions by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are depicted in Figure 24a. It can be noted that KF1 still gives an acceptable behavior for more than 18 seconds even if the software fault is injected before the takeoff. As previously said, it is typical of software faults to take time to causes errors in a system, as otherwise they would be easily eliminated during validation.

In Figure 24c, we can see that the agreement indicator of the Euclidean distance between the two Kalman filter blocks $s_{dKF1-dKF2}$ starts to fall down from $s_{dKF1-dKF2} = 1$ and reaches $s_{dKF1-dKF2} = 0$ after 18s at $t_{det} = 32.5s$. As in the previous experiments, the agreement indicator between KF1 and the model behaves in the same way.

As described in algorithm 1, an error has been detected due to the value of $s_{dKF1-dKF2}$. Figure 24c shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. The Figures 24b and 24d shows that the diversified sensors couples contributing in the estimation of the position (Acc1/Acc2 for the prediction step and GPS1/GPS2 for the correction step in the Data fusion process) show consistency during all the flight time. Thus, a software fault is confirmed in KF1.

Once KF1 is detected as the faulty block, it is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the altitude estimated by KF1 and the dynamic model. This explains why the agreement indicators $s_{dKF1-dKF2}$ and $s_{dKF1-dmodel}$ increases quickly after the identification of the fault in Figure 24c, since they no longer describe the difference between the faulty branch and the other branches in the system.



(a) Software fault: positions estimated by the two Kalman fi lters, the dynamic model, the voter

(b) Software fault: accelerations estimates by the two Accelerometers and their corresponding residues

(c) Software fault: the agreement indicators of the Euclidean distance of the first and second Kalman filters and the DM

(d) Software fault: The residues of the Euclidean distance measured by the first GPS and the second GPS

Figure 24

7. Conclusion

In this paper, we have presented our proposed data fusion architecture for tolerating sensors and software faults which is based on duplication and comparison and was initially proposed in [1]. Four main differences exist between our proposed architecture and the one proposed in [1]: (1) The first difference is that we propose to add an analytical redundancy using the dynamic model (DM). (2) The second difference is that we use a weighted average voter instead of a simple thresholding, which increases the accuracy of the outputs and the error detection process. (3) The third difference is that we have multiple solutions which can be applied to recover a faulty system, increasing the flexibility of our architecture. (4) The fourth difference is that our architecture is experimentally validated in real outdoor environment using a quadrotor. The experiments show that our architecture is able to deal with hardware faults during real flights, and software faults during simulations.

For perspectives, redundancies in the proposed architecture could also be exploited to timely reinitialize state variables values in the dynamic model block. As the dynamic model is a simplification of reality, it will probably diverge slowly from it as time passes. Resetting its state variables values using sensors data when no faults are present in the system (typically when the diversified KF blocks give very similar outputs) would correct this problem. It would also be interesting to test the effectiveness of the architecture regarding successive and simultaneous hardware, or software or mixed hardware and software faults in the system. The proposed solutions for the recovery could then be analyzed further more in terms of robustness and sensitivity to different conditions of faulty situations. Also, this architecture could be tested on other applications such as autonomous cars or rovers to further guarantee its applicability.

References

[1] K. Bader, B. Lussier, W. Schön, A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization, Robotics and Autonomous Systems 88 (2017) 11 – 23.

[2] I. Bloch, A. Hunter, A. Appriou, A. Ayoun, S. Benferhat, P. Besnard, L. Cholvy, R. Cooke, F. Cuppens, D. Dubois, Fusion: General concepts and characteristics, International journal of intelligent systems 16 (10) (2001) 1107–1134.

[3] H. Boström, S. F. Andler, M. Brohede, R. Johansson, A. Karlsson, J. Van Laere, L. Niklasson, M. Nilsson, A. Persson, T. Ziemke, On the definition of information fusion as a field of research, Informatics Research Centre, University of Skovde (2007).

[4] D. Dubois, H. Prade, Possibility theory, Springer, 2012.

[5] D. Dubois, H. Prade, Possibility theory and data fusion in poorly informed environments, Control Engineering Practice 2 (5) (1994) 811–823.

[6] A. P. Dempster, Upper and lower probabilities induced by a multivalued mapping, in: Classic Works of the Dempster-Shafer Theory of Belief Functions, Springer, 2008, pp. 57–72.

[7] R. E. Kalman, A new approach to linear filtering and prediction problems, Journal of basic Engineering 82 (1) (1960) 35–45.

[8] D. L. Hall, S. A. McMullen, Mathematical techniques in multisensor data fusion, Artech House, 2004.

[9] D. L. Hall, J. Llinas, An introduction to multisensor data fusion, Proceedings of the IEEE 85 (1) (1997) 6–23.

[10] Z. Yi, H. Y. Khing, C. C. Seng, Z. X. Wei, Multi-ultrasonic sensor fusion for mobile robots, in: Proceedings of the IEEE Intelligent Vehicles Symposium, 2000, pp. 387–391.

[11] B.-S. Choi, J.-J. Lee, The position estimation of mobile robot under dynamic environment, in: IECON Annual Conference of the IEEE Industrial Electronics Society, 2007, pp. 134–138.

[12] L. Freeston, Applications of the kalman filter algorithm to robot localisation and world modelling, Electrical Engineering Final Year Project (2002).

[13] F. Dellaert, D. Fox, W. Burgard, S. Thrun, Monte carlo localization for mobile robots, ICRA 2 (1999) 1322–1328.

[14] 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.

[15] A. Aitouche, B. Ould-Bouamama, Sensor location with respect to fault tolerance properties, International Journal of Automation and Control 4 (3) (2010) 298–316.

[16] K. Bader, Tolérance aux fautes pour la perception multi-capteurs : application à la localisation d'un véhicule intelligent, Ph.D. thesis, University of technology of Compiegne (2014).

[17] L. Jiang, Sensor fault detection and isolation using system dynamics identification techniques, Ph.D. thesis, The University of Michigan (2011).

[18] S. Wellington, J. Atkinson, R. Sion, Sensor validation and fusion using the nadaraya-watson statistical estimator, in: Proceedings of the Fifth International Conference on Information Fusion., Vol. 1, 2002, pp. 321–326.

[19] R. Doraiswami, L. Cheded, A unified approach to detection and isolation of parametric faults using a kalman filter residual-based approach, Journal of the Franklin Institute 350 (5) (2013) 938–965.

[20] S. Huang, K. K. Tan, T. H. Lee, Fault diagnosis and fault-tolerant control in linear drives using the kalman filter, IEEE Transactions on Industrial Electronics 59 (11) (2012) 4285–4292.

[21] R. Da, C.-F. Lin, A new failure detection approach and its application to gps autonomous integrity monitoring, IEEE transactions on Aerospace and Electronic Systems 31 (1) (1995) 499–506.

[22] Q. Kai, Y. Hui, Y. X. Peng, R. Yan, An integrated fault detection scheme for the federated filter, in: Fourth International Conference on Digital Manufacturing & Automation, 2013, pp. 161–164.

[23] C. Hajiyev, H. E. Soken, Robust adaptive kalman filter for estimation of uav dynamics in the presence of sensor/actuator faults, Aerospace Science and Technology 28 (1) (2013) 376–383.

[24] E. P. Herrera, H. Kaufmann, J. Secue, R. Quirós, G. Fabregat, Improving data fusion in personal positioning systems for outdoor environments, Information Fusion 14 (1) (2013) 45–56.

[25] H. Jamouli, D. Sauter, A generalized likelihood ratio test for a fault-tolerant control system, in: International Conference on Advances in Computational Tools for Engineering Applications, IEEE, 2009, pp. 474–479.

[26] R. E. Walpole, R. H. Myers, S. L. Myers, K. Ye, Probability and statistics for engineers and scientists, Vol. 5, Macmillan New York, 1993.

[27] I. Hwang, S. Kim, Y. Kim, C. E. Seah, A survey of fault detection, isolation, and reconfiguration methods 18 (2010) 636–653.

[28] S. Matzka, R. Altendorfer, A comparison of track-to-track fusion algorithms for automotive sensor fusion, in: Multisensor Fusion and Integration for Intelligent Systems, Springer, 2009, pp. 69–81.

[29] D. Del Gobbo, M. Napolitano, P. Famouri, M. Innocenti, Experimental application of extended kalman filtering for sensor validation, IEEE Transactions on control systems technology 9 (2) (2001) 376–380.

[30] M. Sepasi, F. Sassani, On-line fault diagnosis of hydraulic systems using unscented kalman filter, International Journal of Control, Automation and Systems 8 (1) (2010) 149–156.

[31] J. Al Hage, M. E. El Najjar, D. Pomorski, Multi-sensor fusion approach with fault detection and exclusion based on the kullback–leibler divergence: Application on collaborative multi-robot system, Information Fusion 37 (2017) 61–76.

[32] T. Kerr, Decentralized filtering and redundancy management for multisensor navigation, IEEE Transactions on Aerospace and Electronic Systems (1) (1987) 83–119.

[33] S. Dajani-Brown, D. Cofer, G. Hartmann, S. Pratt, Formal modeling and analysis of an avionics triplex sensor voter, in: International SPIN Workshop on Model Checking of Software, Springer, 2003, pp. 34–48.

[34] R. Wang, Z. Xiong, J. Liu, J. Xu, L. Shi, Chi-square and sprt combined fault detection for multisensor navigation, IEEE Transactions on Aerospace and Electronic Systems 52 (3) (2016) 1352–1365.

[35] D. Berdjag, J. Cieslak, A. Zolghadri, Fault detection and isolation of aircraft air data/inertial system, Progress in Flight Dynamics, Guidance, Navigation, Control, Fault Detection, and Avionics 6 (2013) 317–332.

[36] M. Kumar, D. P. Garg, R. A. Zachery, A method for judicious fusion of inconsistent multiple sensor data, IEEE Sensors Journal 7 (5) (2007) 723–733.

[37] S. Bouabdallah, Design and control of quadrotors with application to autonomous flying, Ph.D. dissertation, Ecole Polytech Federale de Lausanne, Lausanne, Switzerland (2007).

[38] Q. Quan, Introduction to multicopter design and control, 2017.

[39] G. Welch, G. Bishop, An introduction to the kalman filter (1995).

[40] R. Faragher, et al., Understanding the basis of the kalman filter via a simple and intuitive derivation, IEEE Signal processing magazine 29 (5) (2012) 128–132.

[41] L. Ljung, Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems, IEEE Transactions on Automatic Control 24 (1) (1979) 36–50.

[42] G. Latif-Shabgahi, A novel algorithm for weighted average voting used in fault tolerant computing systems, Microprocessors and Microsystems 28 (7) (2004) 357–361.

[43] H. Hamadi, B. Lussier, I. Fantoni, C. Francis, H. Shraim, Observer-based super twisting controller robust to wind perturbation for multirotor uav (2019) 397–405.