Autonomous Robots (2018) 42:1231–1248
https://doi.org/10.1007/s10514-017-9688-z

A sensor-based approach for fault detection and diagnosis for robotic systems

Eliahu Khalastchi[1] · **Meir Kalech**[1]

Received: 11 February 2016 / Accepted: 18 November 2017 / Published online: 7 December 2017

Abstract
As we rely more on robots, thus it becomes important that a continuous successful operation is maintained. Unfortunately, these sophisticated, machines are susceptible to different faults. Some faults might quickly deteriorate into a catastrophe. Thus, it becomes important to apply a fault detection and diagnosis (FDD) mechanism such that faults will be diagnosed in time, allowing a recovery process. Yet, some types of robots require an FDD approach to be accurate, online, quick, able to detect unknown faults, computationally light, and practical to construct. Having all these features together challenges typical model-based, data-driven, and knowledge-based approaches. In this paper we present the SFDD approach that meets these requirements by combining model-based and data-driven techniques. The SFDD utilizes correlation detection, pattern recognition, and a model of structural dependencies. We present two different implementations of the SFDD. In addition, we introduce a new data set, to be used as a public benchmark for FDD, which is challenging due to the contextual nature of injected faults. We show the SFDD implementations are significantly more accurate than three competing approaches, on the benchmark, a physical robot, and a commercial UAV domains. Finally, we show the contribution of each feature of the SFDD.

Keywords Model-based · Data-driven · Fault detection · Fault diagnosis · Sensors · Robotics

1 Introduction

The use of robots in our daily lives is increasing. Recent reports by the International Federation of Robotics (IFR 2016a, b) describe yearly increases of 15% and 25% in sales of service and industrial robots respectively. Robots are used for tasks that are too dangerous, too dull, too dirty or too difficult to be done by humans. Among such tasks we find surveillance and patrolling (Agmon et al. 2008), aerial search (Goodrich et al. 2008), rescue (Birk and Carpin 2006) and mapping (Thrun 2002).

Robots are complex entities comprising both physical (hardware) and virtual (software) components. They operate in different dynamic physical environments with a varying degree of autonomy e.g., satellites, Mars rovers, Unmanned Aerial, Ground or Underwater Vehicles (UAV, UGV, UUV), Robotic arm in an assembly line, etc. These sophisticated, and sometimes very expensive machines, are susceptible to a wide range of physical and virtual faults such as wear and tear, noise, or software-control failures (Steinbauer 2013). If not detected in time, a fault can quickly deteriorate into a catastrophe; endangering the safety of the robot itself or its surroundings (Dhillon 1991). For instance, an undetected engine fault in a UAV (Unmanned Aerial Vehicle) can cause it to stall and crash. Thus, robots should be equipped with Fault Detection and Diagnosis (hereinafter FDD) mechanisms that will allow recovery processes to take place.

In some aspects, robots are not that different from other physical systems in the context of FDD; sensors and actuators can be found in non-robotic systems as well. Yet, the autonomy and complex behaviors of robots do yield requirements that challenge typical FDD approaches—even at the level of sensors and actuators FDD.

An autonomous robot may be required to continue operating in the presence of faults. Thus, an FDD mechanism must be **quick** enough to allow recovery before a catastrophe occurs, and **online**, i.e., detect faults as they occur given

✉ Eliahu Khalastchi
khalastc@bgu.ac.il

Meir Kalech
kalech@bgu.ac.il

[1] Information Systems Engineering, Ben-Gurion University of the Negev, Beersheba, Israel

the data at hand, rather than after the operation is over where all the data is available. For instance, in an altitude of X, an engine fault must be detected and diagnosed within Y milliseconds, to allow an engine restart.

In addition, an autonomous robot may be independent of remote supporting systems. Thus, FDD is carried out on board the robot, burdening system resources (CPU, Memory), and thus should be kept **computationally-light**. For instance, a rover on Mars cannot wait 22 min for a server on Earth to communicate the fact that the rover has a critical fault. Thus, the FDD process should be executed onboard the robot without interfering other mission oriented processes.

Another challenge resides in the fact that an autonomous robot may not have an external source to compare with its own perception, e.g. a human operator, radar station, other robots. If a fault leads to a wrong perception then the entire behavior of the robot is compromised. To detect faults, an FDD mechanism must generate an expectation based on no other than the perception of the robot, which in itself might be faulty.

Complex behaviors, in particular, interaction with the dynamic and uncertain physical environment, yield two additional requirements from FDD: the ability to detect **unknown faults**, and begin **practical to construct**. Due to the dynamic and complex nature of a robot's behavior, one cannot account for every possible fault in advance. In addition, approaches which rely on a high fidelity model of such interactions might prove to be impractical.

Unfortunately, traditional model-based, knowledge-base, and data-driven approaches for FDD in the literature are challenged to meet all of these requirements at once, as we elaborate in the related work section. Basically, some models are hard to construct, unknown faults are hard to extrapolate with knowledge-based approaches and data-driven approaches may not be quick and online.

In this paper, we present a Sensor based approach for Fault Detection and Diagnosis for robotic systems—the SFDD. The approach is designed to meet the FDD requirements for robots by combining data-driven and model-based techniques. In particular, sensor correlation detection and pattern recognition are used to generate an expectation for fault detection purposes. The fault detection heuristic and the diagnosis process use a practical model of structural dependencies. Multiple faults can be diagnosed. In this work we present two implementations: the basic SFDD and the extended SFDD which includes additional features that significantly improve the basic SFDD. These algorithms serve as our main contribution.

Our second contribution lies in the introduction of a new data set which can be used as a public benchmark to evaluate FDD approaches. This data set is very challenging since it contains (a) different types of faults, (b) varying fault durations, (c) concurrent occurrence of double faults, and (d) contextual faults, i.e., data instances that possess values which are valid under one context but are invalid under another.

This work is an extension of our previous work (Khalastchi et al. 2013) which depicted the basic implementation. In this work we expand both the theoretical and empirical parts of the research. In particular, in the theoretical aspect we analyze the advantages and drawbacks of the basic SFDD. These findings pave the way for the extended SFDD implementation presented here. We greatly expand the empirical evaluation as well. We empirically show that the SFDD implementations are significantly more accurate than three competing approaches. For this aim we use the new benchmark, a physical robot, and a commercial UAV. In addition, we show the contribution of each feature that was added to the basic SFDD, resulting in the extended implementation.

2 Related work

Steinbauer conducted a survey on the nature of faults of autonomous robots (Steinbauer 2013). The survey participants are developers competing in different leagues of the Robocup competition (Anon 2013). Along with software faults (Steinbauer et al. 2005; Kleiner et al. 2008), the survey concludes that hardware faults such as sensors, actuators and platform related faults have a deep negative impact on mission success. In this paper we focus on the detection and diagnosis of such faults.

To handle such faults, one may apply an FDD approach. Three general approaches are usually used for FDD: Knowledge-Based systems, Model-Based, and Data-Driven approaches. Knowledge-Based systems (Akerkar and Sajja 2010) typically associates recognized behaviors with predefined known faults. Thus, potentially all requirements can be met apart of the ability to detect unknown faults.

Model-Based approaches (Friedrich et al. 1999; Isermann 2005; Travé-Massuyès 2014) on the other hand, are very equipped to detect unknown faults. Instead of modeling different faults, the expected (normal) behavior of each component is modeled analytically. During operation, the system output is compared to the output given by the model and a high residual indicates a fault. For example, in (Steinbauer and Wotawa 2005) and recently in (Wienke 2016) a model based approach is used for detecting failures in the control software of a robot, including of an unknown type.

Model-Based approaches can meet all requirements but a model might not be available, or impractical to construct due to the dynamic context of the environment. Steinbauer and Wotawa (2010) the authors emphasize the importance of the robot's belief management and fault detection with respect to the real-world dynamic environment. In Akhtar and Kuestenmacher (2011) the authors address this challenge by the use of naive physical knowledge. The naive physical knowledge is represented by the physical properties of objects which are formalized in a logical framework. Yet, formalizing the physical laws demands great a priori knowledge about the context in which the robot operates (environment and task).

A diagnosis model can be automatically generated (Zaman and Steinbauer 2013; Zaman et al. 2013). The authors utilize the Robot Operating System (ROS) (Quigley et al. 2009) to model communication patterns and functional dependencies between nodes. A node is a process that controls a specific element of the robot, e.g., a laser range finder or path planning. The model used by our proposed approach is different. First, it is general and not based on ROS, and thus can work on other platforms. Second, our model is structural rather than behavioral. Even though it is manually constructed, it is practical to construct since the challenge of describing expected behaviors with respect to the environment is avoided. Another key difference is that our model is used for fault detection and not just for diagnosis.

Data-Driven approaches (Golombek et al. 2011; Hodge and Austin 2004; Kodratoff and Michalski 2014) are model free and have a natural appeal for detecting unknown faults. Data-Driven approaches can be divided to outlier detection and machine learning approaches. Outlier detection, e.g., (Pokrajac et al. 2007), uses available data to construct a statistical model and to decide whether or not an outlier is the expression of a fault (Chandola et al. 2009). Being online restricts the FDD approach from having the complete data of the entire operation; data is available only up to the current point in time where it is challenging to differentiate fault expressions from yet-to-be-seen normal behaviors and thus accuracy is compromised. Online model creation may produce dynamic models that fit very well to the dynamic nature of the robot. Yet, such approaches face the challenge of producing the dynamic model within the constraints of time and computational-load.

On the other hand, machine learning approaches, e.g., support vector machines (SVM) (Steinwart and Christmann 2008), may involve offline preprocessing that reduces online computations. An FDD model is learned from offline data and applied online. Supervised learning approaches rely on data that is already labeled for diagnosed faults (Khalastchi et al. 2014). Such data is typically unavailable in the domain of robots. Some approaches, e.g., (Leeke et al. 2011; Christensen et al. 2008), depend on the injection of fault expressions to the training data prior to the learning. Such approaches face the challenge of mimicking the faults propagation through the system to correctly inject the fault's expressions in the data.

Another possibility is to use unsupervised or one-class classification techniques. Hornung et al. (2014) utilize a data set of a fault-free operation. They apply two clustering techniques to produce a two-classed labeled data. First, Mahalanobis-Distance (Mahalanobis 1936) is utilized to cluster data instances depicting normal behavior. Then, negative selection is used to cluster the unknown data instances, i.e., an infinite amount of possible data instances that do not appear in the data and depict the abnormal behavior of the robot. Having the two labels, an SVM learning algorithm is applied offline, and used online as an anomaly detector. For efficiency the high dimension of the data is reduced by techniques of projection and re-projection. In this paper we compare our approach to an algorithm based on Hornung *et al's* algorithm.

For both supervised and unsupervised approaches it is a challenging task to select general features such that overfitting is avoided, and thus the learned model is general enough to handle previously unobserved behaviors. In addition, the offline produced model is static. It is challenging to capture the online dynamic behavior of the robot with static models. In this paper we describe and analyze two implementations of the SFDD: the basic implementation creates the model online, and the extended implementation creates the model offline.

Our arguments regarding knowledge-based, model-based, and data-driven approaches are consistent with the arguments of Pettersson (2005) which presents a survey about execution monitoring in robotics. His survey particularly focuses on mobile robots. There is a very good discussion about the advantages and disadvantages of each approach. Our research is focused on hybrid FDD approaches that by combining model-based and data-driven techniques we can avoid these disadvantages when applied for robotic systems.

The proposed approach in this paper has evolved from different insights learned from previous works. Lin et al. (2010) we showed that dependent data streams, learned offline, form the clusters needed by outlier detection. Khalastchi et al. (2011, 2015) two interesting insights are shown: (a) linear correlation detection is good enough to form clusters of the dependent data streams, and (b) online temporal correlation detection is responsible for the detection of some faults, and thus are important. The proposed approach, presented in this paper, utilizes the Pearson based correlation detection. In addition, we provide further analysis about temporal versus prevailing correlations and their contribution for fault detection.

Hornung et al. (2014) and Birnbaum et al. (2015) discussed some disadvantages of Khalastchi et al. (2011). Namely, they discuss (a) the additional processing time and how it might prevent the processing of high multidimensional streams which are sampled in a high frequency, and (b) the fact that some faults may be left undetected for a long period of time, e.g., an error resulted of a slow wear that increases very slowly and can only be detected after it has grown immensely. Identifying these disadvantages alongside other gained insights from Khalastchi et al. (2011) we introduced the (basic) SFDD approach (Khalastchi et al. 2013), which is extended in this paper. The SFDD utilizes pattern recognition which is computationally lighter than the Mahalanobis-Distance calculation used in Khalastchi et al. (2011), and additionally, it allows the quick detection of slow drifts. Thus, the two disadvantages of Khalastchi et al. (2011) are avoided.

3 Definitions

Let $C = \{c_1, \ldots, c_n\}$ be a set of components of the robot. For example, a *gyro* is a component.

Definition 1 (*sensor*) Let $C_s \subset C$ be a set of sensors. Each sensor c_i expresses an observation $v \in \mathrm{R}$ we can observe about the robot. $\langle v_i^{j,k} \rangle = \langle v_i^j, v_i^{j+1}, \ldots, v_i^k \rangle$ represents a vector of the values sensed by sensor c_i from time j to time k, i.e., the sampled data stream of c_i.

For an ease of reading, we distinguish between the data stream of an ongoing operation, denoted as $\langle d_i^{j,k} \rangle$, and the data stream of an historical completed operation, denoted as $\langle h_i^{j,k} \rangle$. We refer to the vector of the sensed values $\langle v_i^{j,k} \rangle$ as an **observation** of the sensor component c_i.

Implementation note as defined above, each $c_i \in C_s$ is one-dimensional. If a robot's sensor provides multiple dimensions of data, then we treat each dimension as a single component in C_s. For example, an *attitude indicator* is a physical senor which has two degrees of freedom and provides the *pitch* and *roll* angles of a UAV. In this case, we add *attitude pitch* and *attitude roll* as components in C_s. The *attitude indicator*, on the other hand, is kept in C, yet it is not treated as a data providing sensor, but rather, as a component which the *attitude pitch* and *attitude roll* are dependent on. This dependency is defined next.

Given $c_i, c_j \in C$, the predicate $\delta\left(c_i, c_j\right)$ indicates that c_i is dependent on c_j, i.e., the input of c_i is affected by the output of c_j. Note that the definition above contains implicit transitivity, i.e., $\delta\left(c_i, c_j\right) \wedge \delta\left(c_j, c_k\right) \rightarrow \delta\left(c_i, c_k\right)$. For example, assume the following dependencies. A *vacuum pump* c_v pushes air to spin a *gyro* c_g which is used to indicate the *heading* c_h of a UAV. Then the following predicates are true: $\delta\left(c_h, c_g\right), \delta\left(c_g, c_v\right), \delta\left(c_h, c_v\right)$, i.e., the *heading* is also dependent on the *vacuum pump*.

Given $c_i \in C$, the predicate $h\left(c_i\right) \rightarrow true$ if the component c_i is healthy (none faulty).

Given $c_i \in C_s$, the predicate $n\left(c_i\right) \rightarrow true$ if the output of c_i is normal, i.e., the values of $v_i^{j,k}$ do not depict an anomaly. Later we will discuss how an anomaly can be inferred. Armed by these three predicates we can define the system description:

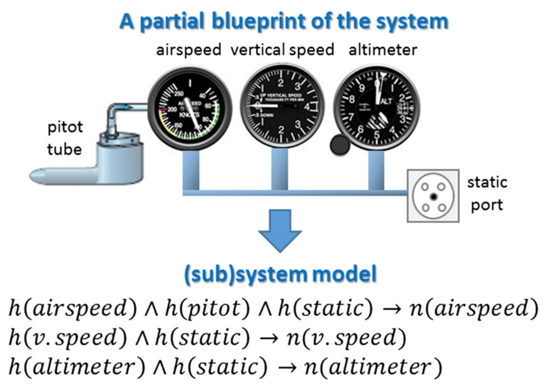
A partial blueprint of the system
airspeed
vertical speed
altimeter
pitot tube
static port
(sub)system model
h(airspeed) ∧ h(pitot) ∧ h(static) → n(airspeed)
h(v.speed) ∧ h(static) → n(v.speed)
h(altimeter) ∧ h(static) → n(altimeter)

Fig. 1 From blueprint (top diagram) to system model (bottom formulas)

Definition 2 (*system description*) The system description SD is given as follows:

$$\forall c_i \in C_s : \left(h\left(c_i\right) \wedge \bigwedge_{c \in C, s.t. \delta(c_i, c)} h\left(c\right) \right) \rightarrow n\left(c_i\right)$$

That is, for every sensor c_i we add a rule which states that if the sensor is healthy and all the components that the sensor is dependent on are healthy as well, then the sensor must display a normal output. As an immediate consequence, $\neg n\left(c_i\right)$ implies that at least c_i or one of the components that c_i is dependent on must be faulty; any of which can explain the anomaly of c_i. Note that $n\left(c_i\right) = true$ does not imply that c_i and all the components that c_i is dependent on are all healthy; masking and intermittent faults might still occur while $n\left(c_i\right)$ is true. Yet, $n\left(c_i\right)$ may indicate that, at the very least, it is probable that c_i and the components c_i is dependent on are healthy. We elaborate on this aspect in the diagnosis section (Sect. 4.5).

It is important to note that the construction of such a structural model is relatively simple when compared to analytical models (required by other model-based approaches) which mathematically or logically describe the expected behavior of every component and their interactions.

The hardware dependencies can be extracted from the "blueprints" of the robot. For example, Fig. 1 depicts a partial pitot-static system of a UAV. At the bottom of the figure the derived (sub)system model is shown. For instance, the *airspeed indicator* is dependent on the input (air pressure) of both the *pitot tube* and the *static port* components. Thus, the rule states that if *airspeed indicator*, *pitot tube*, and *static port* are all healthy, then the *airspeed indicator* should output normal readings. On the other hand, if the *airspeed indicator* is anomalous then a fault to any of these components may explain the anomaly.

A fundamental building block of the fault detection algorithms, proposed in the next section, is the ability to detect correlated sensors.

Definition 3 (*correlated sensors*) Given $c_i, c_j \in C_S$, the predicate $\sigma\left(c_i, c_j\right) \rightarrow true$ indicates that sensors c_i and c_j are correlated, that is, if their data streams are correlated.

The correlation function may vary. For instance, in our implementation we use the Pearson correlation calculation. The threshold above which the data streams are deemed as correlated may vary as well; in our experiments, for instance, we used 0.9. We address these issues in the description of the approaches, and in the evaluation section.

A data stream may depict certain patterns, e.g., a "stuck" pattern where all the values in the data stream are the same. We define Π as the set of patterns. In our implementations $\Pi = \{stuck, drift\}$, i.e., patterns that might express fault symptoms.

Definition 4 (*pattern detection*) Let the predicate $\pi\left(\langle v_i^{j,k}\rangle, x\right) \rightarrow true$ if the data stream $\langle v_i^{j,k}\rangle$ is depicting a pattern $x \in \Pi$.

For instance, $\pi\left(\langle d_i^{t-m,t}\rangle, \text{“}stuck''\right)$ is true if all the values in the data stream of component c_i in the last m time steps are the same. $\pi\left(\langle d_i^{t-m,t}\rangle, \text{“}drift''\right)$ is true if, after a point in time, the derivative of the values in the data stream appears to increase (or decrease) towards a steeper angle. To make the calculation simple and robust to noise in the data, we simply checked if all the values are linearly arranged within a bound in an angle that is steeper than the one before (see Fig. 2).

Note that the patterns are domain specific and serve as inputs to the fault detection algorithm and are not the focus of this paper. Similarly, other patterns can be added, such as "abrupt change", "drop to zero", "static noise", etc. if they might express fault symptoms in other domains, e.g., (Nasa 2009; Hashimoto et al. 2003; Sharma et al. 2010).

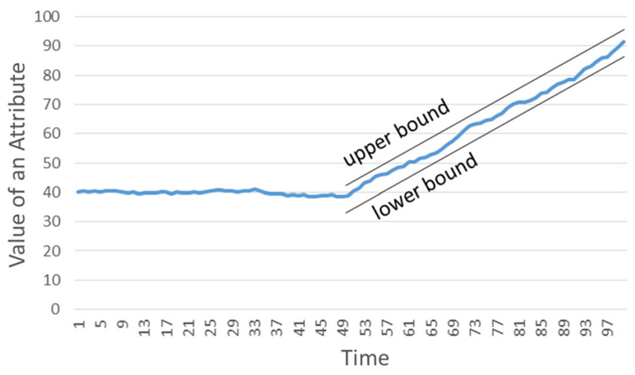

Fig. 2 An example of a "drift" detected. At time step 50 the values stated to drift w.r.t the values of time steps 1–49

Definition 5 (*fault detection problem*) Given a set of components C, an observation over C_s and the system description, the fault detection problem is to raise an alarm upon the occurrence of a fault

4 The sensor-based fault detection and diagnosis

In this section we describe the SFDD and the extended SFDD approaches. First, the fault detection process is discussed. In particular, we describe the SFDD, discuss its drawbacks and continue with the description of the extended SFDD fault detection approach. Then, we describe the diagnosis process, which is the same for both approaches. Finally, we summarize by comparing the two approaches and hypothesize which should be more accurate.

4.1 The basic SFDD approach

The SFDD (Khalastchi et al. 2013) is an online approach. The sensor readings are sampled at a fixed rate and stored in a 'sliding window'. That is, for every time step t we store the observation of the last m readings of every sensor—$\left[\langle d_1^{t-m,t}\rangle, \ldots, \langle d_n^{t-m,t}\rangle\right]$. Sensors may allow different sampling rates. For sensors with a slower sampling rate the previous value is repeated until a new value is sampled. For sensors with a quicker sampling rate we simply neglect the data in between the fixed samples. We advise to use the highest sampling rate your robot can handle.

We assume that the robot starts with a non-faulty state, and a fault might occur after a given point in time. Thus, each instance of the sliding window (time step t) is divided into two halves. The first half, $\left[\langle d_1^{t-m,t-m/2}\rangle, \ldots, \langle d_n^{t-m,t-m/2}\rangle\right]$, which is made of older data samples, is assumed to contain no faults. The second half $\left[\langle d_1^{t-m/2,t}\rangle, \ldots, \langle d_n^{t-m/2,t}\rangle\right]$, which is made of newer data samples, is checked for faults.

The underlying assumption is that correlated sensors should stay correlated unless a fault has occurred or the robot has changed its behavior; the correlation between sensors is dynamic in nature. For instance, elevating the UAV typically gains altitude, but if the UAV is rolled on its side then it will accelerate its heading change rather than its altitude (see Fig. 3). In this behavior the heading rate of change, rather than the altitude climb, is correlated to the force applied on the elevators.

Correlation detection Thus, on the first half of the window we check for correlations between each two sensors. That is, the correlation calculation is applied on dynamic content, and thus provides the information of dynamic correlations. To this end we used the Pearson correlation calculation.

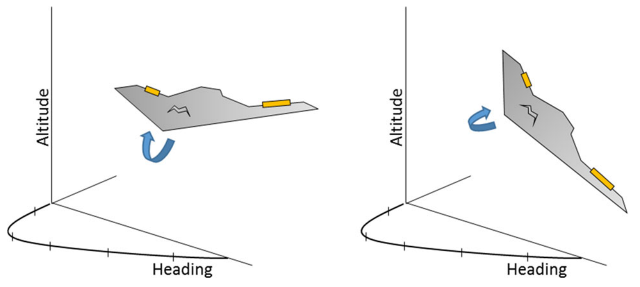

Fig. 3 On the left: elevators affect the altitude. On the right: elevators affect the heading

$$\forall t, \forall c_i, c_j \in C_s \sigma\left(c_i, c_j\right) \longleftrightarrow \left| pearson\left(\langle d_i^{t-m,t-\frac{m}{2}}\rangle, \langle d_j^{t-m,t-\frac{m}{2}}\rangle\right)\right| > \theta$$

The Pearson correlation is a quick calculation that finds linear correlations. The returned value of the Pearson calculation ranges from -1, which indicates a high negative correlation, through 0, which indicates no correlation, to 1, which indicates a high positive correlation. Thus, if the absolute returned value is greater than a given threshold variable θ, then the sensors are deemed as correlated $\sigma\left(c_i, c_j\right) \rightarrow true$. We discuss the role of the threshold in the evaluation section.

In the domain of robots, many sensors, though not linear in their nature, typically possess linear relations amongst each other. This is typically the case for sensors which depict different observables of the same action applied by the robot, and for controlled actuator-sensor relationships. For instance, the *elevators, pitch, vertical speed,* and *altitude*, all have linear ties amongst each other since they depict different aspects of the climbing behavior of a UAV as governed by the *elevator* actuator.

Fault detection: On the second half of the window, we apply the following fault detection heuristic:

$$\forall t, \forall c_i, c_j \in C_s, s.t.\, \sigma\left(c_i, c_j\right) \wedge \nexists c \in Cs.t.\, \delta\left(c_i, c\right) \wedge \delta\left(c_j, c\right), \exists x \in \Pi \left(\pi\left(\langle d_i^{t-\frac{m}{2},t}\rangle, x\right) \wedge \nexists c_j s.t.\, \pi\left(\langle d_j^{t-\frac{m}{2},t\rangle}, x\right)\right) \rightarrow \neg n\left(c_i\right)$$

Meaning, given an instance of a sliding window (time step t), for each two correlated sensors c_i and c_j, which are not dependent on the same component $c \in C$, if c_i is detected for having a pattern x and there does not exist another component c_j that is depicting the same pattern, then c_i is deemed as anomalous, i.e. $\neg n\left(c_i\right)$.

When a pattern is detected $\pi\left(\langle d_i^{t-\frac{m}{2},t}\rangle, x\right)$, it can be the result of the robot's action, or the result of a fault. We assume that a robot's action causes other correlated sensors to depict the same pattern. Thus, if any correlated sensor c_j is depicting the same pattern we do not want to raise an alarm, but if no such sensor can be found, i.e., $\nexists c_j s.t.\, \pi\left(\langle d_j^{t-\frac{m}{2},t}\rangle, x\right)$, then we assume the detected pattern is the result of a fault. Yet, we need to be careful, if c_i and c_j are dependent on the same component c, then a fault to c might cause c_i and c_j to depict the same pattern, thereby leading the fault detection heuristic not to raise an alarm. Hence, we use the constraint $\nexists c \in Cs.t.\, \delta\left(c_i, c\right) \wedge \delta\left(c_j, c\right)$. The underlying assumption is that a fault to one component does not affect other independent components. Since the readings of c_i and c_j come from independent sources, then we can use the above heuristic to detect faults.

Example Assume the *altimeter* is "stuck". It can be the result of the UAV maintaining its altitude or the result of a fault. Assume two correlated sensors: the *GPS indicated altitude* and the *air-pressure*. The *air-pressure* and *altimeter* are both dependent on the *static port* component and the readings of one are a linear transformation of the other, i.e., they will always depict the same patterns, in this case, both are "stuck". On the other hand, the *GPS indicated altitude* is dependent on the electrical system. The *air-pressure* being "stuck" does not imply that the UAV is maintaining its altitude because a fault to the *static port* can lead to the same result. However, since the *GPS* is independent, we can assume that if the *GPS indicated altitude* is "stuck" as well, then this is a reaction to the altitude-maintenance action. If we cannot find any sensor, such as the *GPS,* which depicts the same pattern, then the detected pattern is assumed to be an expression of a fault, e.g., $\neg n\left(altimeter\right)$. There is a contradiction in the information this sensor provides with respect to the other sensors, which implies a fault.

To summarize, the flow of the basic SFDD is depicted in the left part of Fig. 7. The online data is used for correlation and pattern detection. If correlated (and independent) sensors do not show the same pattern then a fault is declared.

Next we describe the advantages and drawbacks of the basic SFDD which have led the development of the extended SFDD.

4.2 The advantages and drawbacks of the basic SFDD approach

The main advantages of the basic SFDD are its: (a) relative computational lightness, (b) reliance on a practical-to-construct model, (c) independence of offline preprocessing, i.e., not additional historical data is needed, (d) domain independence, and (e) high accuracy (Khalastchi et al. 2013). Yet, the basic SFDD approach can be made more accurate and computationally lighter.

The basic SFDD applies correlation detection on the dynamic contents of the sliding window. Thus, temporal correlation can be detected. However, this method has two major

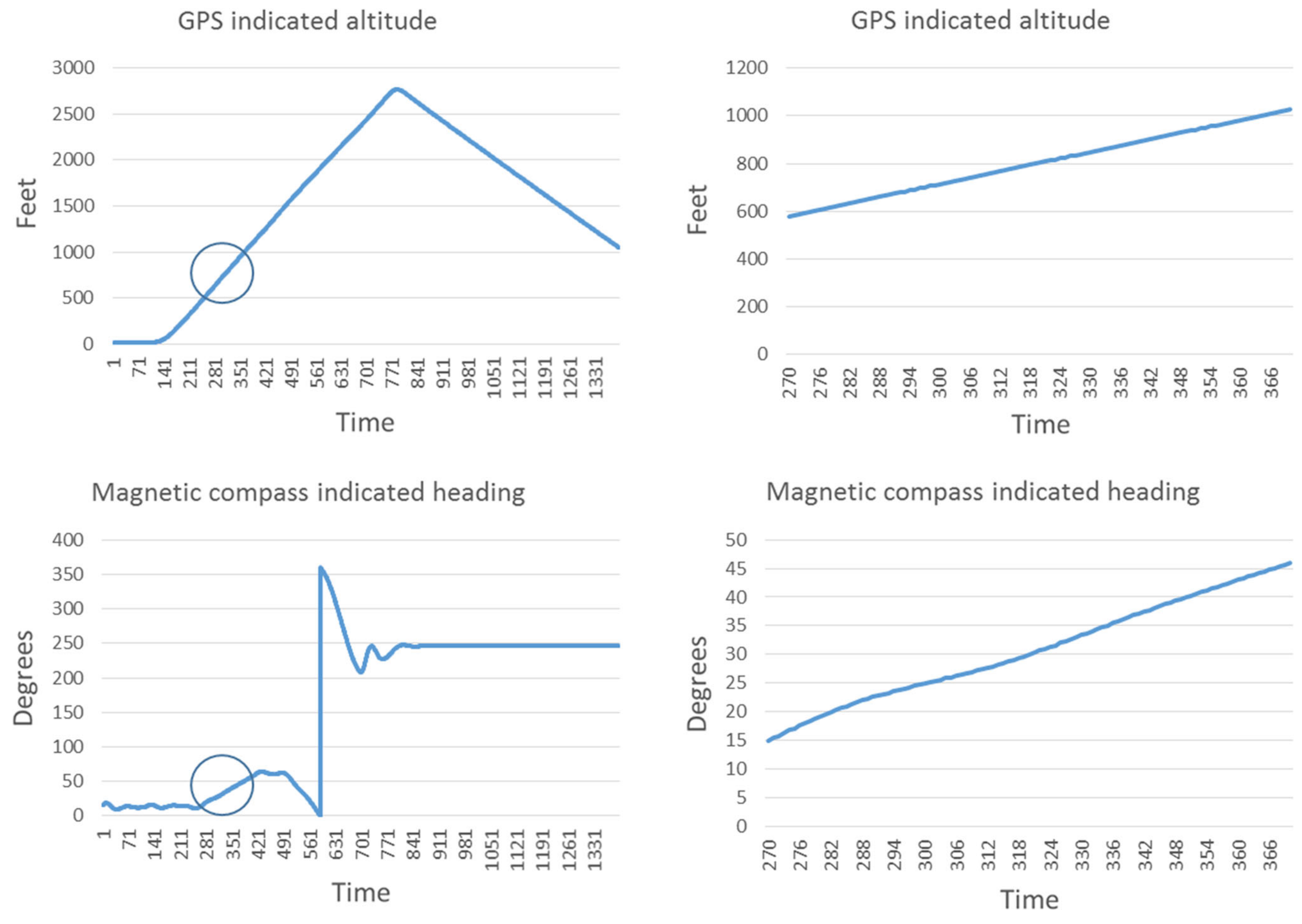

Fig. 4 The GPS indicated altitude and the compass heading are not correlated. Yet, they are temporaly correlated

disadvantages. First, sensors that should not be correlated may be temporally correlated. The existence of an independent, yet falsely correlated, sensor c_j, which depicts the same pattern as c_i, leads to false negatives. Second, sensors that should be correlated may be temporally uncorrelated. This might lead to false positives when there is no fault but still every correlated sensor displays uncorrelated patterns and thus a fault is reported by the basic SFDD.

Example Figure 4 depicts the values of two very different sensors, the *GPS indicated altitude* and the *magnetic-compass indicated heading*. On the left charts we can clearly see that they are not correlated. However, if we zoom in to time steps 270–370, as shown in the right charts, we can clearly see that these two sensors are temporally correlated. The Pearson correlation score for this instance is 0.998 which is a very high score. On the other hand, as we can see in Fig. 5 on the left charts, the *vertical speed indicator* is fairly correlated to the *flight elevator*. This is due to the fact that the *flight elevator* typically control the *pitch*, and hence, the *altitude* and *vertical speed*. However, if we zoom in to time steps 407–457 as the right charts depict, we can clearly see that the two sensors are temporally uncorrelated. Furthermore, this is a typical relationship between controllers and controlled components. The controllers fluctuate in order to keep the controlled components steady. Thus, these sensors are temporally uncorrelated.

Another disadvantage of the basic SFDD can be found in its heuristic. Consider the fault-free case depicted in the charts of Fig. 5. At time steps 407–457 the *vertical speed* is drifting. The SFDD fault detection heuristic searches for correlated sensors which depict the same exact pattern. The correlated sensors do not display a drifting pattern. In particular, the *flight elevator* depicts a fluctuating pattern as we can see at the bottom right chart. The basic SFDD determines this case as a fault even though it is fault-free. The strict demand of the heuristic to display the same pattern should be relaxed and allow a range of patterns to be considered legitimate.

Next we discuss how the extended SFDD approach overcomes these issues.

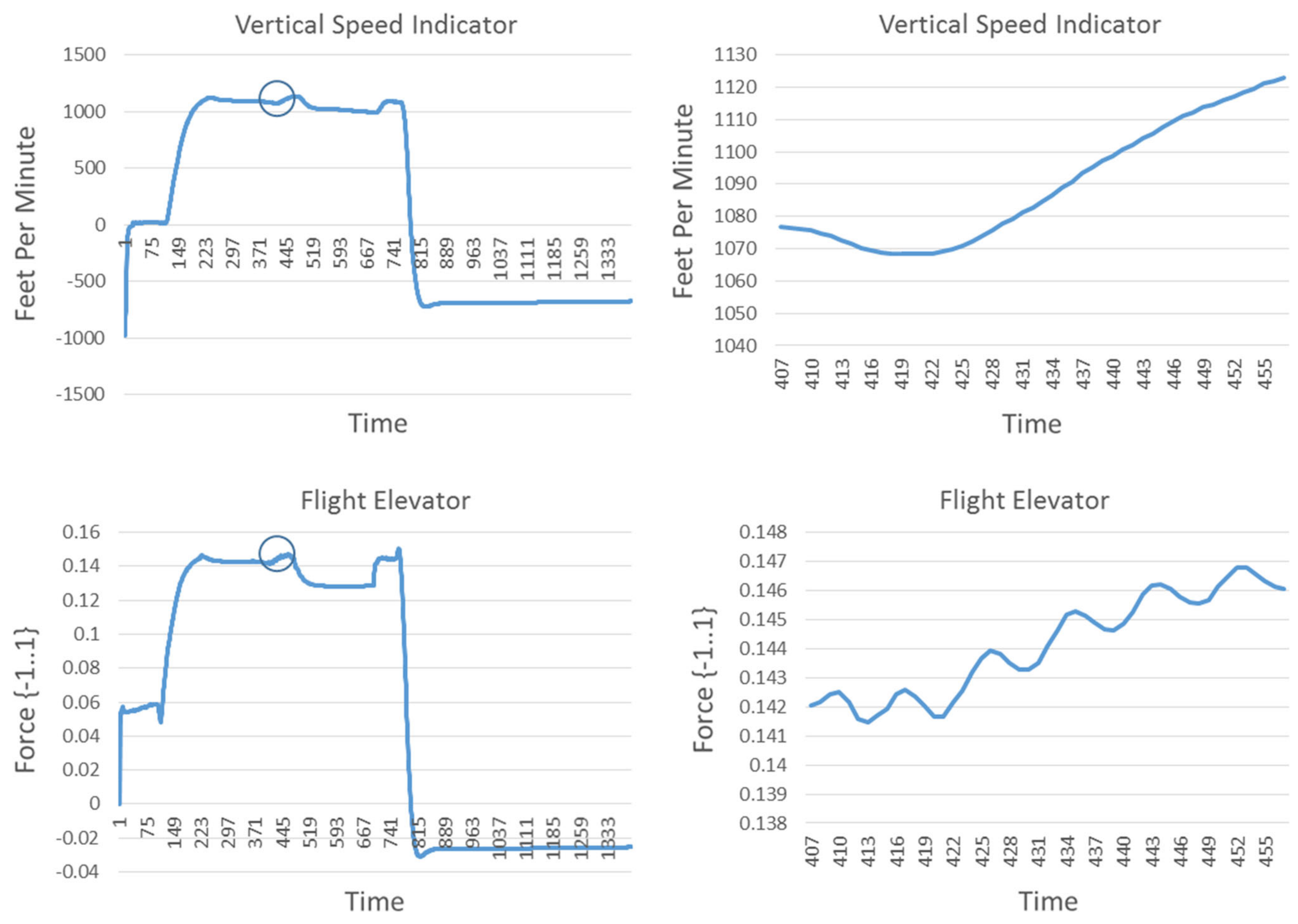

Fig. 5 The vertical speed indicator and the fligjt elevator ar correlated. Yet, they are temporaly not correlated

4.3 The extended SFDD approach

Understanding the drawbacks of the basic SFDD has paved the way for the addition of extended features that improve the basic SFDD. We denote this implementation as the "extended SFDD". The following features make the extended SFDD more accurate and computationally lighter than the basic SFDD.

Feature extraction The extended SFDD applies a simple process of differential feature extraction which allows greater sensitivity for the occurrence of faults.

$$\forall c_i \in C_s, C_s \leftarrow C_s \cup \{c_d\}\, s.t. \langle v_d^{j,k} \rangle \\ = \langle v_i^{j+1} - v_i^{j}, v_i^{j+2} - v_i^{j+1}, \ldots, v_i^{k+1} - v_i^{k} \rangle$$

For each sensor c_i we add to the set of sensors C_s a virtual sensor c_d which is comprised from the differentials of the raw readings of c_i.

The SFDD approach is only sensitive to sensors that are correlated to other sensors. The use of both raw and differential streams allow the discovery of additional correlations, specifically actuator-sensor relationships, thereby increasing the extended SFDD's sensitivity for faults.

The sensors of a robot continuously sense the state of the robot and its environment—before and after an actuator is applied. Thus, the differential readings of a sensor may be correlated to the raw readings of an actuator, indicating the actuator's effect. Figure 6 depicts an example. The *flight elevator* (top left chart) is not correlated to any other sensor, and in particular, the *altimeter* (right chart). Yet, the *flight elevator* is one of the most significant actuators that governs the altitude. Thankfully, the differential data stream of the *altimeter* (bottom chart) is correlated to the *flight elevator*, exposing the relationship between the two. Now, the extended SFDD is sensitive to faults that might occur to the *flight elevator*.

Correlation detection The correlation detection process is performed offline, on a fault-free complete historical record, i.e.

$$\sigma\left(c_i, c_j\right) \longleftrightarrow \left| pearson\left(\langle h_i^{0,end}\rangle, \langle h_j^{0,end}\rangle\right)\right| > \theta$$

This solves the problems of unwanted temporal correlations between uncorrelated sensors, and unwanted temporal mis-

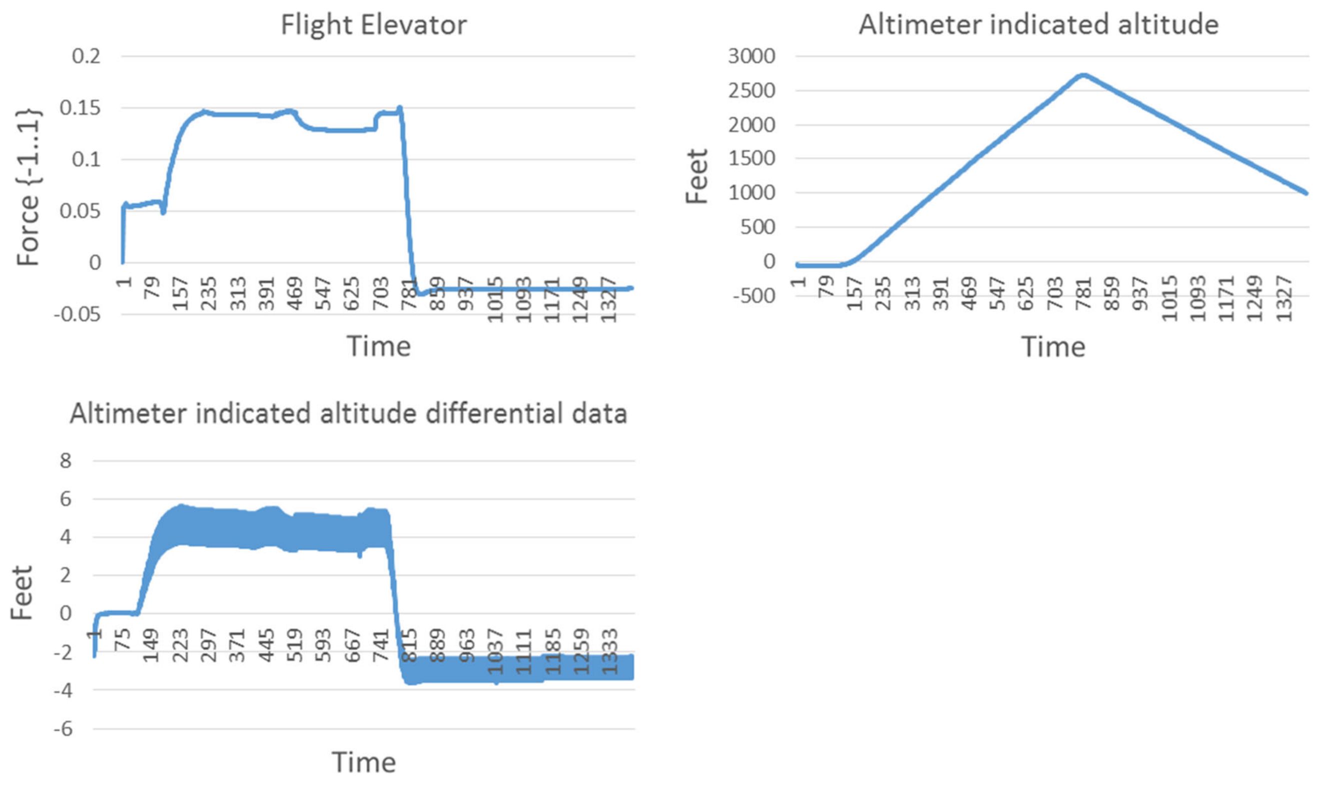

Fig. 6 The Flight elevator actuator is uncorrelated to the altimeter, but is correlated to the altimeter's differential data

correlations between correlated sensors. Sensors are deemed as correlated only if they are correlated in the large scale of an entire operation; small scale temporal correlations or miscorrelations become insignificant in the large scale.

In addition, since the correlation detection is an offline process, the online computational load is decreased. However, the extended SFDD neglects the notion of dynamic correlations, and thus which approach is more accurate becomes an interesting question, which we address this paper.

Fault detection The fault detection heuristic is relaxed, and allows the co-appearance of different patterns to indicate a normal behavior.

We define M to be a set of observations where each observation is a tuple of the form $\langle c_i, x, c_j, y\rangle$ where $c_i, c_j \in C_s$ and $x, y \in \Pi$.

The denotation of the tuple is that the occurrence of pattern $x$to component c_i and pattern $y$to component c_j were observed at the same time. For instance, c_i is the *altimeter* sensor, c_j is the *GPS indicated altitude*, x and y are the "stuck" pattern. Both sensors were observed to be stuck at the same time.

Offline, we process the fault-free historical record in a sliding window fashion (sized m), and for each instance of the window (time step t) we examine correlated sensor components. For each two correlated sensors c_i and c_j, which are not dependent on the same component c, if patterns x, y are detected for c_i, c_j respectively, then the observation $\langle c_i, x, c_j, y\rangle$ is added to M.

Formally:

$$\forall t, \forall c_i, c_j \in C_s, s.t. \sigma\left(c_i, c_j\right) \wedge \nexists c \in C\, s.t. \delta\left(c_i, c\right) \wedge \delta\left(c_j, c\right),$$
$$\pi\left(\langle h_i^{t-m,t}\rangle, x \in \Pi\right) \wedge \pi\left(\langle h_j^{t-m,t}\rangle, y \in \Pi\right) \rightarrow$$
$$M \leftarrow M \cup \left\{\langle c_i, x, c_j, y\rangle\right\}$$

We treat M as a set of legitimate observations. Online, we apply a similar process, yet if the detected observation does not exist in M then we raise a fault alarm.

Formally:

$$\forall t, \forall c_i, c_j \in C_s, s.t. \sigma\left(c_i, c_j\right) \wedge \nexists c \in C s.t. \delta\left(c_i, c\right) \wedge \delta\left(c_j, c\right),$$
$$\left(\pi\left(\langle d_i^{t-m,t}\rangle, x \in \Pi\right) \wedge \pi\left(\langle d_j^{t-m,t}\rangle, y \in \Pi\right) \wedge \langle c_i, x, c_j, y\rangle \notin M\right)$$
$$\rightarrow \neg n\left(c_i\right) \wedge \neg n\left(c_j\right)$$

In other words, if a combination of patterns appears online to correlated and independent sensors c_i, c_j, then if it was observed in the fault-free record then this case is deemed as normal, otherwise, both sensors are deemed as anomalous, i.e., $\neg n\left(c_i\right) \wedge \neg n\left(c_j\right)$.

Example The *rudder* controls the *yaw* axis of the UAV and thus its heading. Offline, we observed a behavior of maintaining a steady heading. The state of the *rudder* was observed

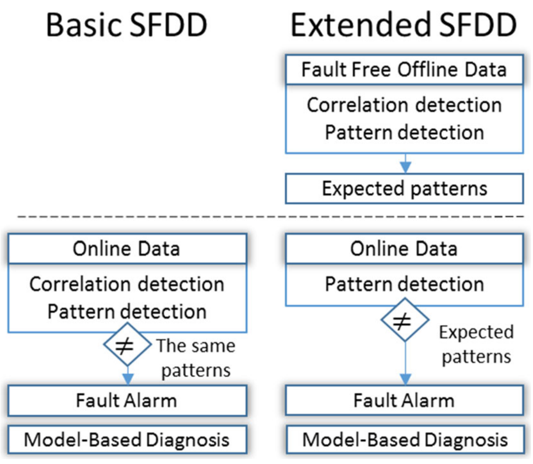

Fig. 7 The flow of the basic versus extended SFDD

to be "fluctuating" or "stuck" while the *heading indicator* was observed to be "stuck". Online, we observe the *heading indicator* to be "stuck". It might be due to a fault or may be due to the behavior of maintaining a steady heading. We look at the current pattern of the *rudder*, if it is observed to be "fluctuating" or "stuck" then we can determine that this case expresses normal behavior. However, if the *rudder* is depicting a "drift" pattern then this previously unobserved case is deemed as the result of a fault.

To summarize, the flow of the extended SFDD is depicted in the right part of Fig. 7. The fault-free offline data is used for correlation and pattern detection. This process generates an expectation for the co-appearance of different patterns to correlated (and independent) sensors. Online, if the expected patterns are not observed then a fault is declared.

Next we compare the basic and extended implementations side by side and summarize.

4.4 Basic versus extended SFDD: a summary

Table 1 summarizes the different parts of these approaches, one against the other. The basic SFDD has no feature extraction, while the extended SFDD considers differential streams as well as raw streams, making it more sensitive for faults.

The correlation detection of the basic SFDD is online. Dynamic correlations are considered, yet they might contribute to false negatives and false positives. The extended SFDD applies offline large scale correlation detection, overcoming the drawbacks of the basic SFDD, yet neglecting the notion of dynamic correlations.

The basic SFDD has no offline processing. The extended SFDD, on the other hand, stores which patterns had occurred together in a fault-free historical record of the robot's operation. This allows the extended SFDD to apply a relaxed version of the fault detection heuristic applied by the basic SFDD.

The basic SFDD expects correlated sensors to depict the same exact patterns to indicate a normal behavior. The extended SFDD expects correlated sensors to depict patterns which were observed offline.

4.5 The diagnosis process

To isolate the faulty components that caused the failure we use Model-Based Diagnosis (MBD). The input to classical MBD algorithms, as described in Reiter (1987) and de Kleer and Williams (1987), is a tuple SD, $COMPS$, OBS, where SD is a formal description of the diagnosed system's behavior, $COMPS$ is the set of components in the system that may be faulty, and OBS is a set of observations. A diagnosis problem arises when SD and OBS are inconsistent with the assumption that all the components in $COMPS$ are healthy. The output of an MBD algorithm is a set of diagnoses.

Definition 6 (*diagnosis*) A set of components $\Delta \subseteq COMPS$ is a diagnosis if

$$\bigwedge_{c \in|\Delta} (\neg h(c)) \wedge \bigwedge_{c' \notin \Delta} (h(c)) \wedge SD \wedge OBS$$

is consistent, i.e., if assuming that the components in Δ are faulty, then SD is consistent with OBS.

Table 1 Basic SFDD versus extended SFDD implementations

	Basic SFDD	Extended SFDD
Feature extraction	None	$\forall c_i \in C_s, C_s \leftarrow C_s \cup \{c_d\}\, s.t. \langle v_d^{j,k} \rangle = \langle v_i^{j+1} - v_i^j, \ldots, v_i^{k+1} - v_i^k \rangle$
Correlation detection	$Pearson\left(d_i^{t-m,t-\frac{m}{2}}, d_j^{t-m,t-\frac{m}{2}}\right)$	$Pearson\left(h_i^{0,end}, h_j^{0,end}\right)$
Offline processing	None	$\pi\left(h_i^{t-m,t}, x \in \Pi\right) \wedge \pi\left(h_j^{t-m,t}, y \in \Pi\right) \rightarrow M \leftarrow M \cup \left\{\langle c_i, x, c_j, y \rangle\right\}$
Fault detection heuristic	$\left(\pi\left(d_i^{t-\frac{m}{2},t}, x \in \Pi\right) \wedge \nexists c_j s.t. \pi\left(d_j^{t-\frac{m}{2},t}, x\right)\right) \rightarrow \neg n\,(c_i)$	$\pi\left(d_i^{t-m,t}, x \in \Pi\right) \wedge \pi\left(d_j^{t-m,t}, y \in \Pi\right) \wedge \langle c_i, x, c_j, y \rangle \notin M \rightarrow \neg n\,(c_i) \wedge \neg n\left(c_j\right)$

The set of components ($COMPS$) in our domain is C. The system description (SD) has been formally defined in Definition 2. The observation is the data stream of the sensor components (Definition 1). However, to use MBD algorithms we should express the observation as Boolean variables. In fact, we have defined the predicate $n(c)$ which returns true if the output of component c is normal. Thus, we can define OBS with this predicate on the output of the sensors: $OBS = \bigwedge_{c \in C_S} n(c)$. The values of $n(c)$ for the sensor components will be computed before the diagnosis process by the fault detection process described in the previous section.

A known approach to compute diagnoses is conflict directed. A conflict directed approach identifies first conflict sets, each of which includes at least one fault. Then, it applies a hitting set algorithm to compute sets of multiple faults that explain the observation (de Kleer and Williams 1987; Williams and Ragno 2007; Stern et al. 2012). Intuitively, since at least one component in every conflict is faulty, only a hitting set of all conflicts can explain the unexpected observation. Note that inferring conflicts in our model is straightforward: for each observation that returns $false$, a conflict includes all the components that their output influence that observation. Obviously, it is logically inferred from the model.

Barinel is a recently proposed MBD algorithm (Abreu et al. 2009, 2011) based on exactly this concept. The main drawback of using Barinel is that it often outputs a large set of diagnoses, thus providing weaker guidance to the operator that is assigned to solve the observed fault.

To address this problem, Barinel computes a score for every diagnosis it returns, estimating the likelihood that it is true. This serves as a way to prioritize the large set of diagnoses returned by Barinel. The exact details of how this score is computed is given by Abreu et al. For the purpose of this paper, it is important to note that the score computation used by Barinel is Bayesian: it computes for a given diagnosis the posterior probability that it is correct given the n and $\neg n$ observations. In this work we also consider the results of the observations to rank the diagnoses.

For example, consider the subsystem depicted in Fig. 1, and that the *airspeed indicator* readings are stuck. According to the model $\neg n(altimeter) \rightarrow \neg h(altimeter) \vee \neg h(pitot) \vee \neg h(static)$. Thus, the conflict is $\{altimeter, pitot, static\}$. However, since the readings of the *vertical speed* and *altimeter* are normal, the probability that the *static port* is faulty is decreased, thereby returning $\Delta = \{\{airspeed\}, \{pitot\}\}$.

5 Evaluation

In this section we provide our evaluation: the experimental setup, competing approaches, evaluation measurements and the results.

5.1 Experimental setup

We created a new data set which we contribute as a public benchmark for general fault detection and diagnosis approaches as well as approaches designed more specifically for the domain of robots. We utilized the FlightGear (Perry 2004) flight simulator. FlightGear is an open source high fidelity flight simulator designed for research purpose and is used for a variety of research topics. FlightGear is a great domain for robotic FDD research, particularly for UAVs, and yet it is not used enough for this topic of research. We thoroughly introduce this domain, and use it as our main testbed. In addition to this simulated domain, we experiment with two real-world domains as well: a commercial UAV and a laboratory robot. These domains are not as intricate as the FlightGear domain, but they do provide the different aspects of real-world data, and thus, appliance of the FDD approaches to the real-world domains.

5.1.1 The FlightGear benchmark domain

The most important aspect of FlightGear as a domain for FDD is the fact that FlightGear has built-in realistically simulated sensors, internal components, engine, and actuators faults. For example, if the *vacuum* fails, the *HSI* (Horizontal Situation Indicator) gyros spin down slowly with a corresponding degradation in response as well as a slowly increasing bias/error (drift) (Perry 2004), and in turn, if not detected, lead the aircraft miles away from its desired position. Thus, the first advantage is that FDD approaches may solve a real-world problem. More importantly, while in other domains faults' expressions are injected into the recorded data after the flight is done, in FlightGear the simulated faults are built-in and can be injected during the flight. First, there is no bias; the faults were not modeled by the scientist who created and tested the FDD approach. Second, built-in faults which are injected during the flight propagate and affect the whole simulation. Hence, fault expressions are more realistic.

We created a control software to fly a Cessna 172p aircraft as an autonomous UAV. The flight instruments are used as sensors which sense the aircraft state with respect to its environment. These sensor readings are fed into a decision making mechanism which issues instructions to the flight controls (actuators) such that goals are achieved. As the UAV operates, its state is changed and again being sensed by its sensors. The desired flight pattern consists of seven steps: a takeoff, 30 s of a strait flight, a right turn of 45°, 15 s of a strait flight, a left turn of 180°, 15 s of a strait flight, and a decent to one kilofeet. In total, the flight pattern duration is 6 min of fight.

During a flight, 23 sensors are sampled in a frequency of 4 Hz. These sensors present 5 flight controls feedback (actua-

Table 2 Summary of injected faults

	Fault to	Type	Effect
1	Airspeed indicator	Sensor	Stuck
2	Altimeter		
3	Compass		
4	Turn indicator		Quick drift to minimum value
5	Heading indicator		Stuck
6	Vertical speed indicator		
7	Slip skid ball		
8	Pitot tube	Internal component	Airspeed drifts up
9	Static port		Airspeed drifts Altimeter and Encoder are stuck
10	Electrical		Turn indicator slowly drifts down
11	Vacuum		Attitude and heading indicators slowly drift
12	Elevator	Actuator	Stuck

tors), and 18 flight instruments (sensors). The sampled flight controls are the ailerons, elevators, rudder, flaps, and engine throttle. The flight instruments are the airspeed indicator, altimeter, horizontal situation indicator, encoder, GPS, magnetic compass, slip skid ball, turn indicator, vertical speed indicator, and engine's RPM. Each instrument yields 1 to 4 streams of data which together add up to 18 sensors. Note that we did not sample sensors that would have made the fault detection easy. For instance, sampling the current and voltage of the aircraft would have made the detection of an electrical system failure unchallenging.

The data set contains one flight which is free of faults, 5 subsets that each contains 12 recorded flights in which different faults were injected, and one special "stunt flight". In total, the data set contains 62 recorded flights with almost 90,000 data instances. We injected faults to 7 different sensors, 4 different internal components, and an actuator. Table 2 depicts the different faults and their effect. For instance, a fault injection of type 9 fails the *static port* which, in turn, leads the *airspeed indicator* to drift upwards or downwards, and the *altimeter* and *encoder* to be stuck.

Four subsets represent a single fault scenario. Each of the 12 flights in a subset corresponds to an injected fault in Table 2, i.e., flight 1 was injected with an airspeed indicator failure, flight 2 was injected with an altimeter failure, etc. Flights 1–9 were injected 6 times with faults. Each injection occurred at a random time of the flight, and for a random duration of 5–30 s. Flights 10, 11 were injected once, at a random time of the flight, with a duration of 180 s. These flights depict a slow and shallow drift which develops over time. The twelfth flight was injected three times with a fault to the flight elevator actuator, where each fault occurs at a random time and has a duration of 30 s.

The fifth subset represents a double-fault scenario. In each of the 12 flights we injected 3 times double faults. Each time a fault was injected to two different components at the same time. The double injection occurred at a random time of the flight, and for a random duration of 10–30 s. In total, the data set contains 72 injected faults.

In addition to these five subsets of flights we recorded a special "stunt flight" which has a significantly different flight pattern than the other flights in the benchmark. The stunt flight includes high-G maneuvers, i.e., sharp turns, climbs, dives, and loops where axis change is highly frequent. While the aircraft is rolled to its side, turning sharply, a fault was injected to the altimeter which causes the indicated altitude to be stuck. The quick transitions between states make this flight very challenging for FDD approaches, especially for approaches which are depended on offline learning, like the extended SFDD, where state transitions were mild.

5.1.2 Real-world domains

In addition to the FlightGear simulated domain, we experiment with two physical robots: a commercial UAV and a laboratory robot. These domains are not as complex as the simulated domain, but they serve to show the domain independence of the SFDD and its ability to handle real-world data.

Commercial UAV domain The real UAV domain consists of 6 recorded real flights of a commercial UAV. 53 sensors were sampled in 10 Hz. The sensors consists of telemetry, inertial, engine and servos data. Flights duration varies from 37 to 71 min. The UAV manufacture injected a synthetic fault to two of the flights. The first scenario is a value that drifts down to zero. The second scenario is a value that remains frozen (stuck). The detection of these two faults were challenging for the manufacture since in both scenarios the values are in normal range. These two flights were used to test the FDD approaches. In total, the test flights contain 65,741 data instances out of which 1,593 are expression of faults.

Laboratory robot domain *Robotican* is a laboratory robot that has 2 wheels, 3 sonar range detectors in the front, and 3 infrared range detectors which are located right above the sonars, making the sonars and infrareds redundant systems to one another. This redundancy reflects real world domains such as unmanned vehicles. In addition, Robotican1 has 5 degrees of freedom arm. Each joint is held by two electrical motors. These motors provide a feedback value depicting the voltage that was actually applied.

The following scenario was repeated 10 times: the robot slows its movement as it approaches a cup. Concurrently, the robot's arm is adjusted to grasp the cup. In 9 out of the 10 times faults were injected. Faults of type *stuck* or *drift* were injected to different type of sensors (motor voltage, infrared and sonar). We sampled 15 sensors in 8 Hz. Scenarios duration lasted only 10 s where 1.25 s expressed a fault. In total, the test set contains 800 instances out of which 90 are expression of faults.

5.1.3 Competing fault detection approaches

We carefully chose three competing approaches, where each approach is similar to the SFDD in some aspect, but emphasizes the difference of other aspects. Table 3 depicts the competing approaches, their similarities and differences form the SFDD, and the hypothesis for their ability to detect faults accurately.

The extended SFDD uses an offline fault-free data set to "learn" which patterns may co-appear. Classification algorithms can utilize such information as well. The SVM is a well-known and typically successful classification algorithm. In addition, we provide the SVM based approach with both faulty and normal examples. Yet, we expect the SVM to fail, i.e., to report every fault as normal. Our experiment contains injected faults of contextual nature, and thus, domain knowledge may be needed. The SVM is domain-independent, while the SFDD's fault detection heuristic utilizes domain knowledge (component dependencies). In addition, the context is implicitly provided to the SFDD in the form of the sliding window and correlated sensors. The SVM approach does not cluster correlated sensors together. In turn, uncorrelated data instances form a scattered distribution that require large margins from the support vectors. These large margins reduce the fault detection sensitivity to the point where almost every data instance would be considered as normal.

Another aspect of the SFDD is the dependency on linear correlations. If two sensors depict a linear correlation then the readings of one sensor can be used to predict the readings of the other. To this end, linear regression can be used offline to learn these linear ties from a fault-free record. However, online, the observed values are distributed around the model's predicted values. A threshold needs to be set such that all normal data instances fall within this threshold. Since the readings of one sensor are actually affected by several others, then large deviations from the predicted value are expected. For instance, the *vertical speed* cannot be accurately predicted by the readings of the *pitch angle* since the *vertical speed* is affected by the current *altitude* and *airspeed* as well. As a result, higher thresholds are set, leading to undetected faults. On the other hand, the SFDD avoids this problem since it concerns patterns and not expected values.

Our previous approach (Khalastchi et al. 2011), Online Data Driven Anomaly Detection (ODDAD), is very similar to the SFDD. The online data is processed via a sliding window and sensors are grouped together according to their linear correlations. Yet, the fault detection heuristic is different. ODDAD uses the model-free Mahalanobis-Distance calculation (Mahalanobis 1936) to indicate anomalies. The SFDD applies a fault detection heuristic that utilizes domain-knowledge (component dependencies). Thus, we expect the SFDD approaches to be more accurate than ODDAD.

5.1.4 Evaluation measurements

The main measurement we are interested in is how accurate an FDD approach is. For that, we define the following common parameters about a fault report, which Table 4 depicts.

Every detected fault counts as one true positive. Every false alarm counts as one false positive. Every undetected fault counts as a false negative, and the rest counts as true negatives, i.e., no faults and no alarm. The fault detection rate (a.k.a. recall, true positive rate, sensitivity) is defined as $= \frac{TP}{TP+FN}$ where *TP* is the amount of true positives, and *FN* is

Table 3 The competing approaches, theirs similarities, differences and hypothesis compared to the SFDD

	SVM	Linear Regression	ODDAD
Similarity	Offline learning	Linear relations	Correlation based
Difference	Classification versus fault detection heuristic	Value prediction versus pattern prediction	Model free versus model based
Hypothesis	Undetected contextual faults	Threshold setting problem	Lower fault detection rate

Table 4 Evaluation measures

Evaluation measure	Description
Fault detection rate	The rate of faults detected out of all injected faults
False alarm rate	The rate of incorrect fault reports out of all none-faults
FP_{time}	The average duration of incorrect fault reports
Detection time lag	The time it takes an FDD approach to detect a fault
Diagnosis	Precision and true positive rate

the amount of false negatives. The false alarm rate (a.k.a. false positive rate, 1—specificity) is defined as $= \frac{FP}{TN+FP}$ where *FP* is the amount of false positives and *TN* is the amount of true negatives. A perfect FDD approach has a detection rate of 1 (all faults are detected) and a false alarm rate of 0 (no false alarms).

Knowing the false alarm rate of an approach is not enough. An approach might theoretically have only one incorrect fault report, but this report has a duration of the entire operation. Hence, we measure the average duration of incorrect fault reports denoted as FP_{time}. An FDD approach with a lower score is better.

As we have mentioned in the introduction, an FDD approach can benefit from being quick, i.e., detect the fault as quickly as possible as it occurs. Thus, another important analysis is the (average) time lag between the occurrence of a fault and its detection by an FDD approach. In particular, this is interesting for the flights 10 and 11 in the FlightGear benchmark, where the expression of the fault progresses slowly.

For the evaluation of the diagnosis we first need to define which diagnosis is returned. Since the diagnoses are ranked according to their probabilities, we simply choose the first diagnosis that is ranked first. The selected diagnosis is a set of components which supposed to be faulty. For each faulty component that is not included in this set we count one false negative (FN). For each faulty component that is included in the set we count one true positive (TP). For each component included in the set, which is not faulty we count one false positive (FP). Hence, we are interested in two measurements: The diagnosis precision defined as $\frac{TP}{TP+FP}$, and the diagnosis true positive rate defined as $\frac{TP}{TP+FN}$.

For example, assume that the diagnosis for injected faults to the *vertical speed indicator* and the *static port* is {*static port*}. Thus, TP=1, FN=1. In a case that the diagnosis for a *pitot tube* failure is {*airspeed indicator*}, TP=0, FP=1.

5.2 Results and analysis

In this section we provide our analysis and results. We start with the comparison of the competing approaches. We continue with an analysis of the contribution of each feature of the SFDD. We conclude with a summary of conclusions from the results.

5.2.1 The results of the competing approaches

Table 5 depicts the detection rate and false alarm rate results of the competing approaches taken from the FlightGear benchmark. The test column depicts the test flights from which the results were taken.

We present results for (1) all the five subsets, (2) the fifth subset—the multiple fault scenarios, and (3) the stunt flight. For instance, the false alarm rate of the extended SFDD measured over the entire 60 test flights is 0.008. As expected, the SVM based approach has failed to detect the faults. Even though the SVM was provided with both normal and faulty examples, every instance was classified as normal resulting in a detection and a false alarm rates of 0. The different parameters of the SVM (e.g. SVM kernels) has little to none effect on this result. The main reason for this result is that the dimension is constructed of all sensors—even uncorrelated ones. Uncorrelated sensors yield a sparse distribution where the support vectors have to be broadly widen to include all normal samples. As a result, every new sample falls within the normal category.

Similarly, the linear regression based approach, LNR, finds broad thresholds around expected values in order to include all normal observations. As a result, small drifts and

Table 5 Fault detection and false alarm rates of the competing approaches over the FlightGear benchmark

Test	FDD approach	Detection rate	False alarm rate
All 60 test flights	SVM	0	0
	LNR	0.68	0.09
	ODDAD	0.88	0.13
	Basic	0.87	0.1
	Extended	0.96	0.008
Subset 5—double faults	SVM	0	0
	LNR	0.68	0.07
	ODDAD	0.87	0.12
	Basic	0.82	0.1
	Extended	0.97	0.004
Stunt flight	SVM	0	0
	LNR	1	1
	ODDAD	1	0.017
	Basic	1	0.009
	Extended	1	0.040

Table 6 Results on the real-world domains

Domain	Approach	Detection rate	False alarm rate
Commercial UAV	SVM	0	0
	LNR	0	0.040
	ODDAD	1	0.037
	Basic	1	0.032
	Extended	1	0.002
Robotican1	SVM	0	0
	LNR	0.44	0.0035
	ODDAD	1	0.017
	Basic	0.56	0
	Extended	1	0.012

stuck patterns fall within these thresholds and faults are left undetected. In turn, the detection rate decreases. The false alarm rate is quite high as well. Non-fault samples can still be outside of the calculated thresholds, as we cannot expect to observe every possible data instance in fault-free record.

For all test flights, the ODDAD approach and the basic SFDD have very similar scores, where the false alarm rate of the basic SFDD is a bit better. Compared to the SVM and LNR these scores are significantly better. The main reason is that the ODDAD and basic SFDD approaches do group correlated sensors together, and each avoids the threshold-setting problem of the SVM and LNR in a different way. ODDAD calculates for each group of correlated sensors a threshold based on the furthest Mahalanobis distance observed in the current instance of the sliding window. The basic SFDD applies pattern detection and thus it does not need to set a threshold above which a fault is detected. The reason the basic SFDD has a bit lower (better) false alarm rate is due to its use of the structural model in the fault detection heuristic, which only considers independent correlated sensors.

The extended SFDD is significantly better than the competing approaches in all test flights except the stunt flight; the ODDAD and basic SFDD show a lower false alarm rate. The stunt flight significantly differs from fault-free flight, which is quite leveled. The ODDAD and basic SFDD, which do not process an offline operation, have a natural advantage over the extended SFDD when the online operation significantly differs from the offline operation. It can be interesting to devise a way to automatically decide online which implementation to use. We leave that to future work.

Similar results were obtained from the real-world domains as Table 6 depicts. Once again the extended SFDD is significantly more accurate than the other approaches. The detection rate of the basic SFDD is surprisingly low for the Robotican1 domain—0.56. Since each operation lasted only a few seconds, the size of the sliding window is kept small. Recall that the basic SFDD divides the sliding window into two halves, providing even less data to work with. This may explain the low result.

Table 7 Contributions of different features of the SFDD (TPR,FPR)

	Online	Offline
Raw	0.87, 0.1	0.87, 0.005
Raw+Diff	0.89, 0.15	0.96, 0.008

The FP_{time} measure for the FlightGear benchmark of LNR, ODDAD, basic SFDD, and extended SFDD is 31, 20, 8.9, and 6.5 s respectively. The extended SFDD has the lowest (best) score. These results show that both SFDD implementations have small durations of incorrect fault reports as well as having low false alarm rates.

5.2.2 The contribution of the SFDD features

We wish to show the contribution of each feature of the extended SFDD approach presented here. Table 7 depicts the contribution of the following features. “Online” refers to online (temporal) correlation detection and the basic SFDD’s fault detection heuristic. “Offline” refers to offline (large scale) correlation detection and the extended SFDD’s relaxed fault detection heuristic, which relies on offline observations. “Raw” refers to the raw sensor readings, “Raw+Diff” refers to the addition of differential data streams as extracted features. Each cell represents a fault detection rate and a false alarm rate the approach scored. For example, the basic SFDD is represented by the cell for {Online, Raw} and it scored a detection rate of 0.87 and a false alarm rate of 0.1. The extended SFDD approach is depicted at the most bottom right cell with a detection rate of 0.96 and a false alarm rate of 0.008. These results were taken from the FlightGear data set. The other data sets produced similar results.

We can see from the table that for each instance the addition of the differential data streams contributed to the increase of both the detection rate and the false alarm rate. Monitoring more sensors increases the sensitivity of the SFDD approach, resulting in more fault reports. When we compare online against offline, we see the significant improvement of accuracy. In particular, the relaxed fault detection heuristic and the large scale correlation detection overcome the drawbacks of temporal correlations and the strict heuristic of the basic SFDD, which in turn, contribute to less false alarms.

An additional value which may govern the sensitivity of the SFDD is θ—the threshold above which sensors are deemed as correlated. The SFDD implementations consider only correlated sensors since an uncorrelated sensor has no other source to compare against. Unfortunately, there is no rule that can be said about θ. On the one hand, a low θ increases the number of “correlated” sensors, potentially increasing the sensitivity of the SFDD and thus contribut-

Table 8 The effects of the size (in s) of the sliding window

Sliding window size (s)	Detection rate	False alarm rate	Detection time
2	0.97	0.058	1.7
4	0.95	0.008	4.71
6	0.78	0.009	7.53
8–12	0.67	0.007	10.52
14	0.58	0.006	10.32
16	0.48	0.008	16

ing to an increase of true positives. On the other hand, such "correlated" sensors might contribute to false positives by displaying other patterns, or contribute to false negatives by displaying the same patterns. Hence we advise to simply use a high threshold, e.g., 0.9, such that enough sensors, which are correlated, are monitored by the SFDD.

Another feature that governs the sensitivity of the SFDD is the **size of the sliding window**; there is a tradeoff between fault detectability and false alarms. This boils down to the implementation of the pattern detection. For instance, in our implementation all values in a stream have to be the same in order to detect a stuck pattern. Consider that the readings of a sensor are stuck for a period of x seconds. A window sized equal or less than x will allow the detection of the stuck pattern, while a window sized larger than x will not allow the recognition of the stream as being stuck. Thus, we can choose x to be large or small enough to govern the sensitivity of the approach to small occurring patterns. Larger windows overlook small patterns, which in turn, decrease fault reports, thereby potentially decreasing the detection and the false alarm rates. A smaller size makes the approach sensitive to small patterns, which in turn, increases fault reports and potentially increases the detection and false alarm rates.

This notion is well supported by the results shown in Table 8. The second column depicts the detection rate (or sensitivity) of the extended SFDD approach, and the third depicts the false alarm rate. Results were measured over the entire FlightGear benchmark. We can see that the detection rate decreases as the size of the sliding window gets longer. The minimum duration of an injected fault in the benchmark is 5 s. As a result, a significant reduction in the fault detection rate is observed between 4–6 s size. The majority of the injected faults have a duration less than 15 s. As a result the detection rate decreases to less than 0.5 for a size of 16 s.

As expected, we observe a significant decrease of the false alarm rate as the size of the window increases. After a certain size the false alarm rate is unchanged. This is a case where in each experiment the same data instances persistently lead to incorrect reports, regardless of the size of the window. Hence, the false alarm rate remains objectively unchanged.

In addition to the sensitivity, the SFDD's detection time lag is also affected by the size of the sliding window. Assume a sliding window of the size of x seconds. At time t, a fault has occurred and a data stream begins to show a pattern. Only at time $t + x$ there will be an instance of the sliding window in which the pattern fills the entire stream. Only then will the pattern detection be able to return "true" and allow the fault to be detected. Hence, we expect the averaged detection time to be linearly correlated to the size of the sliding window, i.e., it will take in average x seconds to detect a fault with a sliding window size of x seconds. Indeed, the results shown the fourth column support this claim. We can see that there is a linear correlation between the size of the window and the detection time. Considering the required sensitivity and detection time in a given domain, one can set the size of the sliding window to meet these requirements.

The following diagnosis results are independent from fault detection, i.e., as if all faults have been detected without any false alarms. The single fault scenarios (FlightGear subsets 1–4) contain 12 different types of injected faults. There is only one case where the diagnosis process has returned a FP instead of a TP. The FP was caused due to the return of the *airspeed indicator* in the case of *pitot tube* failure. Since no other components are dependent on the *pitot tube*, the diagnosis process could not distinguished between these components and thus, both were returned as different diagnoses; the first one (*airspeed*) was evaluated. Thus, out of 12 injected faults TP = 11, FP = 1, FN = 1, leading to precision and a true positive rate of $\frac{11}{12} = 0.916$.

The double-fault scenarios (FlightGear subset 5) contain 12×3 different injected double-faults. FPs and FNs resulted when the *airspeed indicator* was returned instead of the *pitot tube*. In addition, one FN resulted from the return of the *static port* as a diagnosis for faults in the *static port* + *vertical speed indicator*. Due to the dependency on the *static port*, it was the most probable diagnosis. Thus, the diagnosis precision is $\frac{66}{66+5} = 0.93$ and the true positive rate is $\frac{66}{66+6} = 0.916$.

6 Discussion and future work

In this paper we have tackled the problem of FDD in the robotics domain. For our evaluation we used three different domains: two real-world physical robots and one simulated more intricate domain, which can be used as a public benchmark for FDD. Our proposed SFDD approach meets the requirements for robotic systems: it is accurate, online, quick, able to detect unknown faults, computationally light and practical to construct. We presented an extension of the SFDD and showed the contribution of each extended feature: the use of differential data, the offline large-scaled correlation detection, and the relaxed fault detection heuristic. These features contribute to the accuracy and computational lightness of the SFDD. Yet we showed a case where the basic SFDD can perform more accurately than the extended SFDD. In addition, we showed that the extended SFDD approach is more accurate than three other FDD approaches, which might have been considered as a reasonable choice.

For future work, we aim to investigate two things: (1) can we devise an approach that can automatically choose online between the decision of the basic and the extended SFDD based on how different the online data is from the fault free operation. (2) can the SFDD be used to detect software faults.

Note that a component does not have to be a physical sensor, it can also be an abstract (virtual) belief such as *estimated time to intercept*, or even parameter variables of high-level instructions such as *desired climbing rate*. Yet modeling the dependency between software components, e.g., a sensor fusing function and the sensors it is applied on, may prove to be more intricate than the model we have used.

References

Abreu, R., Zoeteweij, P. & Van Gemund, A. J. (2009). Spectrum-based multiple fault localization. In *24th IEEE/ACM international conference on automated software engineering, Auckland*.

Abreu, R., Zoeteweij, P., & Van Gemund, A. J. (2011). Simultaneous debugging of software faults. *Journal of Systems and Software*, *84*(4), 573–586.

Agmon, N., Kraus, S., & Kaminka, G. A. (2008). Multi-robot perimeter patrol in adversarial settings. In *IEEE international conference on robotics and automation (ICRA), Pasadena* (pp. 2339–2345).

Akerkar, R., & Sajja, P. (2010). *Knowledge-based systems*. Sudbury, MA: Jones & Bartlett Publishers.

Akhtar, N., & Kuestenmacher, A. (2011). Using Naive Physics for unknown external faults in robotics. In *The 22nd international workshop on principles of diagnosis (DX-2011), Murnau, Germany*.

Anon. (2013). *Robocup*. (Online) Available at:http://www.robocup.org/

Birk, A., & Carpin, S. (2006). Rescue robotics–A crucial milestone on the road to autonomous systems. *Advanced Robotics*, *20*(5), 595–605.

Birnbaum, Z., et al. (2015). Unmanned Aerial Vehicle security using behavioral profiling. In *International conference on unmanned aircraft systems (ICUAS), Denver, CO*.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*, 1–58.

Christensen, A. L., O'Grady, R., Birattari, M., & Dorigo, M. (2008). Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, *24*, 49–67.

de Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

Dhillon, B. S. (1991). *Robot reliability and safety*. Berlin: Springer.

Friedrich, G., Stumptner, M., & Wotawa, F. (1999). Model-based diagnosis of hardware designs. *Artificial Intelligence*, *111*, 3–39.

Golombek, R., Wrede, S., Hanheide, M., & Heckmann, M. (2011). Online data-driven fault detection for robotic systems. In *IEEE/RSJ international conference on intelligent robots and systems (IROS), San Francisco, CA*.

Goodrich, M. A., et al. (2008). Supporting wilderness search and rescue using a camera-equipped mini UAV. *Field Robotics*, *25*(1), 89–110.

Hashimoto, M., Kawashima, H., & Oba, F. (2003). A multi-model based fault detection and diagnosis of internal sensors for mobile robot. In *IEEE/RSJ international conference on intelligent robots and systems (IROS), Las Vegas, Nevada, USA*.

Hodge, V., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, *22*, 85–126.

Hornung, R., et al. (2014). Model-free robot anomaly detection. In *IEEE/RSJ international conference on intelligent robots and systems (IROS), Chicago*.

IFR. (2016). Executive summary world robotics 2016 industrial robots. In *The International Dederation of Robotics (IFR)*.

IFR. (2016). Executive summary world robotics 2016 service robot. In *The International Federation of Robotics (IFR)*.

Isermann, R. (2005). Model-based fault-detection and diagnosis-status and applications. *Annual Reviews in control*, *29*(1), 71–85.

Khalastchi, E., Kalech, M., Kaminka, G. A., & Lin, R. (2015). Online data-driven anomaly detection in autonomous robots. *Knowledge and Information Systems*, *43*(3), 657–688.

Khalastchi, E., Kalech, M., & Rokach, L. (2013). Sensor fault detection and diagnosis for autonomous systems. In *International conference on Autonomous agents and multi-agent systems (AAMAS), Saint Paul*.

Khalastchi, E., Kalech, M., & Rokach, L. (2014). A Hybrid Approach for Fault Detection in Autonomous Physical Agents. In *International conference on Autonomous agents and multi-agent systems (AAMAS), Paris*.

Khalastchi, E., Kaminka, G. A., Kalech, M., & Lin, R. (2011). Online anomaly detection in unmanned vehicles. In *International conference on Autonomous agents and multi-agent systems (AAMAS), Taipei*.

Kleiner, A., Steinbauer, G., & Wotawa, F. (2008). Towards automated online diagnosis of robot navigation software. In *International conference on simulation, modeling, and programming for autonomous robots, Venice, Italy*.

Kodratoff, Y., & Michalski, R. S. (2014). *Machine learning: An artificial intelligence approach*. Burlington: Morgan Kaufmann.

Leeke, M., Arif, S., Jhumka, A., & Anand, S. S. (2011). A methodology for the generation of efficient error detection mechanisms. In *IEEE/IFIP 41st international conference on dependable systems and networks (DSN), Hong Kong* (pp. 25–36).

Lin, R., Khalastchi, E., & Kaminka, G. A. (2010). Detecting anomalies in unmanned vehicles using the mahalanobis distance. In *International conference on robotics and automation (ICRA), Anchorage, AK*.

Mahalanobis, P. C. (1936). On the generalised distance in statistics. *The National Institute of Sciences of India*, *2*, 49–55.

Nasa. (2009). *ADAPT*. (Online) Available at: http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/diagnostic-algorithm-benchmarking/eps-testbed/

Perry, A.R. (2004). The flightgear flight simulator. In: *USENIX annual technical conference, Boston, MA*.

Pettersson, O. (2005). Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, *53*(2), 73–88.

Pokrajac, D., Lazarevic, A., & Latecki, L. J. (2007). Incremental local outlier detection for data streams (pp. 504–515). In *IEEE symposium on computational intelligence and data mining (CIDM), Honolulu, HI*.

Quigley, M., et al. (2009). ROS: An open-source Robot Operating System. In *ICRA workshop on open source software, Kobe, Japan*.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*, 57–95.

Sharma, A. B., Golubchik, L., & Govindan, R. (2010). Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)*, *6*(3), 23.

Steinbauer, G. (2013). A Survey about Faults of Robots Used in RoboCup. *RoboCup 2012: Robot Soccer World Cup XVI* (pp. 344–355). Berlin: Springer.

Steinbauer, G., Morth, M., & Wotawa, F. (2005). Real-time diagnosis and repair of faults of robot control software. *Robot Soccer World Cup,* (pp. 13–23).

Steinbauer, G., & Wotawa, F. (2005). Detecting and locating faults in the control software of autonomous mobile robots. In *International joint conference on artificial intelligence (IJCAI), Edinburgh, Scotland* (pp. 1742–1743).

Steinbauer, G., & Wotawa, F. (2010). On the Way to Automated Belief Repair for Autonomous Robots. In *The 21st international workshop on principles of diagnosis (DX), Portland, Oregon*.

Steinwart, I., & Christmann, A. (2008). *Support vector machines*. Berlin: Springer.

Stern, R. T., Kalech, M., Feldman, A., & Provan, G. M. (2012). Exploring the Duality in Conflict-Directed Model-Based Diagnosis. In *The 26th conference on artificial intelligence (AAAI), Toronto, Ontario, Canada*.

Thrun, S. (2002). Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, *1*, 1–35.

Travé-Massuyès, L. (2014). Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, *27*, 1–16.

Wienke, J. l. W. S., (2016). Autonomous fault detection for performance bugs in component-based robotic systems. In *IEEE/RSJ international conference on intelligent robots and systems (IROS), Daejeon, Korea*.

Williams, B. C., & Ragno, R. J. (2007). Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, *155*(12), 1562–1595.

Zaman, S., & Steinbauer, G. (2013). Automated generation of diagnosis models for ROS-based robot systems. In *The 24th international workshop on principles of diagnosis, Jerusalem, Israel*.

Zaman, S., et al. (2013). An integrated model-based diagnosis and repair architecture for ROS-based robot systems. In *IEEE international conference on robotics and automation (ICRA), Karlsruhe*.

Eliahu Khalastchi Received the B.Sc. degree and the M.Sc. degree (magna cum laude) in computer science, from the Bar-Ilan University, Ramat-Gan, Israel, in 2009, and 2010, respectively. Currently, Khalastchi has finished his Ph.D. studies under the advisement of Dr. Meir Kalech and Prof. Lior Rokach in the department of Information System Engineering at Ben-Gurion University of the Negev, Be'er Sheva, Israel. Khalastchis research interests lie in artificial intelligence and specifically in fault detection and diagnosis for single and multi robots.

Meir Kalech Completed his Ph.D. at the Computer Science Department of Bar-Ilan University in 2006. In 2008 he became a faculty member of the Department of Information System Engineering at Ben-Gurion University of the Negev. Kalechs research interests lie in artificial intelligence and specifically in anomaly detection and diagnosis. He is a recognized expert in model-based diagnosis (MBD) and has published dozens of papers in leading journals and refereed conferences. Kalech established the Anomaly Detection and Diagnosis Laboratory at Ben-Gurion University, which promotes research with the government and leading corporations such as General Motors and Elbit.