Sensor Fault Detection and Diagnosis for Autonomous Systems

Eliahu Khalastchi, Meir Kalech, Lior Rokach
Information Systems Engineering, Ben-Gurion University of the Negev, Israel
Deutsche Telekom Laboratories at Ben-Gurion University, Israel
{khalastc,kalech,liorrk}@bgu.ac.il

ABSTRACT

Autonomous systems are usually equipped with sensors to sense the surrounding environment. The sensor readings are interpreted into beliefs upon which the robot decides how to act. Unfortunately, sensors are susceptible to faults. These faults might lead to task failure. Detecting these faults and diagnosing a fault's origin is an important task that should be performed quickly online. While other methods require a high fidelity model that describes the behavior of each component, we present a method that uses a structural model to successfully detect and diagnose sensor faults online. We experiment our method with a laboratory robot Robotican1 and a flight simulator FlightGear. We show that our method outperforms previous methods in terms of fault detection and provides an accurate diagnosis.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics – *Autonomous vehicles, Sensors.*

General Terms

Reliability, Experimentation

Keywords

Fault detection, Model-Based Diagnosis, Robotics, UAV, sensors.

1. INTRODUCTION

The use of robots in our daily civilian and military life is increasing. Robots can replace humans in certain tasks that are too boring or too dangerous. However, these sophisticated and sometimes very expensive machines are susceptible to faults. A fault has the potential to cause mission failure or even to endanger the system itself or its surrounding e.g. a UAV (unmanned aerial vehicle) can crash due to a fault.

Faults are not restricted just to hardware wear and tear. Long before a robot tries to activate some actuator it should sense the ever changing dynamic environment and compute its beliefs over the world. It then needs to make choices of how to behave, and send the command to the relevant controllers to activate the actuators. Based on its actions, the world changes; thus the described operation proceeds iteratively. For example, a robot's laser distance sensor returns a reading. This reading derives a belief – the distance to a target object. The decision making process of the robot might decide to move towards the target object. This decision is translated to the execution of a set of commands from the robot's API. Each command activates some actuators, like the robot's wheels. As the robot is getting closer to the target the sensors react accordingly and the belief is updated.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

At each step of this cycle a fault might occur. Either due to false sensing or runtime errors (in the operating system of the robot) or due to a hardware failure. These faults need to be detected quickly and diagnosed. For example, if the code that computes the distance to the target crashes, then the robot might continue to move forever. The same can happen if the laser sensor returns a constant value or deviated from the target, or if the wheels are spinning in place.

Steinbauer *et al.* [12] conducted a survey on the nature of faults of autonomous robot systems. The survey participants are the developers competing in the different leagues of the Robocup competition [10]. Steinbauer *et al.* concluded that internal hardware components such as batteries and motors are most affected by faults to connectors or communication, and are critical to mission success. They categorize these faults as platform faults. Furthermore, configuration problems greatly affect sensors, and sensors faults have similar frequency but a higher negative impact than platform faults. We propose a Model-Based approach that detects and diagnoses sensors faults, and relates sensors to internal hardware components to enable diagnosis of these platform related faults.

In this paper we propose a **fault detection** method in which an autonomous system can detect that there is a failure in the system. In addition, we propose a **diagnosis** method which isolates the faulty component/sensor. In our fault detection method we combine a Data Driven approach ([7],[9],[13]) with a Model-Based approach [6]. We recognize correlations between data readings online and track them to detect correlation breaks along time where suspicious patterns are detected. To accurately determine a fault has occurred, and for the diagnosis process, we use a structural model that indicates sensors dependency on hardware components, and thus enables the isolation of the faulty sensor or component.

In our previous work [7] we introduced a successful model free, unsupervised and online approach for anomaly detection. In this work we use a structural model to isolate the faulty component or sensor and thus provide diagnosis. In addition, we address faults that their symptoms appear only over time.

There are two faults in particular that are hard to detect [13]: (1) Stuck – the sensor returns the same reading regardless the real state, and (2) Drift – the sensor returns values which continually drift upwards (or downwards) from the real state.

The stuck fault may indicate data that is in a range of the truthful readings, and the drift may change very slowly maintaining the correct range of the sensor. Both kinds of fault express abnormal behavior. On the other hand, even a healthy sensor can sometimes produce values that appear to be stuck or drifting as a reaction to

the robot's current action. Thus, these faults are challenging to detect.

We evaluate our method in two domains. The first is a laboratory robot [11] that applied different behaviors upon which faults were injected to its sensor readings. The second domain is a high fidelity flight simulator – FlightGear [2]. This simulator presents a more rich and complex environment to test our method. The system and instrumental faults which are already built-in and realistically mimic faults that occur in real flights present a very suitable domain to test our fault detection and diagnosis methods.

We compare our fault detection method to the algorithm presented in [7] and to the Local Outlier Factor (LOF) [9] and show that the method proposed in this paper is more accurate. In addition, we show the high success rates of detecting and diagnosing faults in both domains.

2. RELATED WORK

Steinbauer *et al*. conducted a survey on the nature of faults of autonomous robot systems [12]. The survey participants are developers competing in different leagues of the Robocup competition [10]. The reported faults were categorized as hardware, software, algorithmic and interaction related faults. The survey concludes that hardware faults such as sensors, actuators and platform related faults have a high negative impact on mission success. In this paper we focus on diagnosing sensor related faults as well as internal hardware components that sensors are dependent on.

There are diverse fault detection approaches as analytical methods, data-driven or knowledge-based systems [6]. Analytical approaches use mathematical models to compare expected outputs with observed outputs and derive a residual that is used to determine whether or not a fault has occurred. The limitation of this approach is by its requirement to express all the behavioral laws of every component in mathematical equations, which is a very hard task [13].

Data-driven approaches are model-free statistical methods. These methods face the challenge of dimension reduction and a dependency in the existence of quality information that can be extracted from the data ([6],[13]).

We propose the use of a structural model which depicts sensors dependencies on internal hardware components. As opposed to other analytical models, the structural model does not include a mathematical representation of components behavior and thus is easier to construct. On the other hand, the proposed approach is not driven by data alone, and is not dependent on the existence of quality information and has no need for dimension reduction.

We put our focus on one-dimensional sensors. Faults to these types of sensors may appear in a variety of forms. For example, the Advanced Diagnostics and Prognostics Testbed (ADAPT) [1] depicts the following faults to sensors on an electrical circuit: "stuck" where all values produced by the sensor are the same, "drift" where the values show a movement towards higher (or lower) values, and "abrupt" where there is a sudden large increase (or decrease) in the sensor's values. This testbed uses for the diagnosis competition [5] industrial track (DXC). Hashimoto *et al.* [4] use kalman filters along with kinematical models to diagnose "stuck" and "abrupt" faults to sensors of a mobile robot, as well as "scale" faults, where the (gain) scale of the sensor output differs from the normal. Our diagnosis algorithm relies on a function that returns the state of the sensor (i.e. abrupt, drift, stuck, scale etc.).

When a sensor has a state such as stuck or drift it might be the result of the robot's action and not a fault (e.g. a UAV climb might appear as an altitude drift). To conclude whether the sensor readings behave correctly or faulty we apply a similar technique to that of our previous work. [7]. Since only the robot's perception is available, we use correlated sensors for comparison. In our previous work, we determined which sensors are correlated, and per each correlated set of sensors we measured their current-input's degree of being an outlier with respect to previous inputs using the Mahalanobis Distance [8]. The approach suggested here, compares the state of correlated sensors which do not share component dependency. The same logical assumption is applied in both approaches. If two sensors are correlated they should react in the same manner to the robot's behavior. However, if the two sensors show different behaviors then it might be due to a fault.

3. PROBLEM DESCRIPTION

We define an autonomous system with a structural model that represents the sensors dependency of internal hardware components. The most fundamental entities are the sensors and components. The set of the sensors is $S = \{s_1, \dots, s_n\}$. Each sensor s_i reports online readings – a single value that is changed over time as the system operates. The second set represents hardware components which we denote as $C = \{c_1, \dots, c_k\}$. The given structural model maps internal components to their dependent sensors denoted as M:

Definition 1: [dependency set] *M is a set of tuples of the form* $< \boldsymbol{c_i}, \boldsymbol{S'} >$ *where* $\boldsymbol{c_i} \in \boldsymbol{C}, \boldsymbol{S'} \subseteq \boldsymbol{S}$. $\boldsymbol{S'}$ *considers sensors that* ***are*** *dependent* ***on the hardware component*** $\boldsymbol{c_i}$. *Given the healthy predicate* $\boldsymbol{h(x)}$ *denotes the health of* $\boldsymbol{x}, \forall\, \boldsymbol{s_j} \in \boldsymbol{S'}: \neg \boldsymbol{h}(\boldsymbol{c_i}) \rightarrow \neg \boldsymbol{h}(\boldsymbol{s_j})$.

If component c_i is faulty then all of its dependent sensors ($\forall s_j \in S'$) will report faulty data. However, if sensor s_j is faulty it does not imply that c_i is faulty; s_j can be faulty itself.

To formally represent the mapping between components and sensors we define the sensor mapping function and its inverse component function:

Definition 2: [mapping functions] *Given a component* $c_i \in C$, $\tau(c_i) = S'$ *is a function that returns the set of sensors that are dependent on the component* c_i. *Given a sensor* $s_i \subseteq S$ $\varphi(s_i) = C_i$ *is a function that returns the set of the components that the sensor* s_i *is dependent on.*

Figure 1 illustrates our model. It presents a partial structural model of the cockpit panel of a Cessna 172p airplane as it modeled by FlightGear simulator. The dark rectangles represent the components and the bright rectangles represent sensors. For instance, to enable the speed indicator to return a correct reading both the pitot system and the static system need to be operating correctly. The altimeter is dependent only on the static system. The altimeter returns two data readings - altitude and pressure, each is considered as a one dimensional sensor that is dependent on the static system. The same is applied for the attitude indicator that returns the values of the Pitch, Roll and Yaw, which are all dependent on the vacuum system. The GPS is a redundant sensor that besides position values it also returns the speed and the altitude of the aircraft. Since the GPS is dependent only on the electrical system, it will still work in case of a static system failure.

The goal is to report, for each online reading of the sensors, whether the reading indicates a fault (fault detection), and upon a detected fault to diagnose which of the internal components or sensors caused the fault (diagnosis).

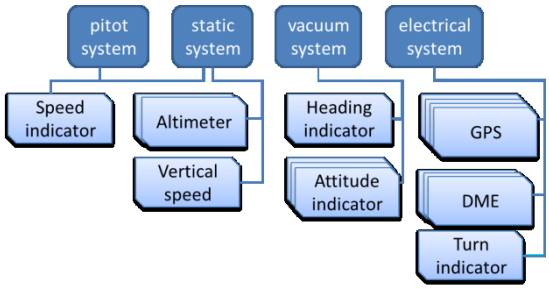

Figure 1: Partial structural model of a Cessna 172p airplane.

4. FAULT DETECTION AND DIAGNOSIS

We present an online and unsupervised approach for fault detection and diagnosis. Besides consuming the input in an online fashion, the approach is applicable online; meaning that (1) fault detection and diagnosis are returned quickly after the fault occurs, and that (2) all calculations are applicable on a computationally weak robot. Furthermore, only the online consumed data is available; no other past records of offline data are used. Supervised approaches require labeled data of both normal and faulty observations. These labeled observations are not always available and cannot hope to cover every possible scenario. Our unsupervised approach has no need for labeled data of any sort.

Our approach proceeds as follow: (1) The consumed inputs of the sensors readings are subjected to a correlation test that determines which sensors are correlated to each other. (2) Each sensor is tested for showing predefined suspicious patterns; sensors that show at least one of those patterns are marked as uncertain. (3) The fault detection and diagnosis use the structural model to verify that the uncertain sensors reacted to a fault and to diagnose the root cause of the fault.

Parts (1) and (2) are a data driven approach and will be described in the following subsection. Part (3) is a model-based approach and will be described in sections 4.2 and 4.3. The contribution of the model-based approach to the fault detection and to the diagnosis processes is described in the results section.

4.1 Online Preprocess

In our approach we propose to store the online consumed data in a sliding window.

> ***Definition 3: [sliding window]*** *A sliding window of size m is an $m \times n$ matrix denoted as H^t, stores the latest m readings of n sensors ending at time t. A cell $H^t_{i,j}$ stores the value of sensor s_i at time step $t-j$.*

With each incoming input, H is updated, keeping the current data of the last m time steps for each sensor. The data of H is used to both check which sensors are correlated as well as which sensors display predefined suspicious patterns.

We expect that redundant sensors that measure the same thing or sensors that are affected by the same action of the robot will show the same behavior during their last m values and return a high rate of correlation. Therefore, sensors that used to be correlated and now show a different behavior might indicate that a fault has happened. Since there is no external observation to compare to in the domain of autonomous systems but only the system's perception is available, the knowledge of which sensors are correlated is very important.

We divide H^t to two parts. The first (oldest) $(1,\dots,\frac{m}{2})$ rows and the second (newest) $(\frac{m}{2}+1,\dots,m)$ rows. The first $\frac{m}{2}$ values of each sensor s_i denoted as $H_i^{t_1} = (v_{i_1},\dots,v_{i_{m/2}})$ are used for a correlation test. The last $\frac{m}{2}$ values (newer) of sensor s_i denoted as $H_i^{t_2} = (v_{i_{\frac{m}{2}+1}},\dots,v_{i_m})$ are subjected to predefined suspicious patterns recognition test, as will be described later.

The correlation detection algorithm uses the *Pearson Correlation Coefficient* calculation with respect to every two sensors in H $(\forall s_i, s_j \in S\ (i \neq j), H_i^{t_1}, H_j^{t_1})$, thereby determining their rate of correlation.

> ***Definition 4: [correlated sensors set]*** *given sensor s_i and sliding window H^t, the set $P_{i,H^t} \subseteq S$ contains the sensors that are correlated to s_i based on H^t: $P_{i,H^t} = \{s_j | pearson(H_i^{t_1}, H_j^{t_1}) > threshold\}$[1]. We define $\mathcal{P}_{H^t} = \{P_{1,H^t},\dots,P_{n,H^t}\}$.*

The knowledge of the correlated sensors will be used to determine whether a suspicious pattern in a sensor is due to a fault or it is a normal reaction to the system's behavior. A suspicious pattern for a single dimension sensor is an observable pattern of the sensor values over time, which might indicate a fault. Notice that the pattern may be a normal reaction to the system's behavior.

Suspicious patterns are predefined and sensor specific. The appearance of a suspicious pattern in a sensor is associated with a sensor state. We are given a function that recognizes these suspicious patterns and returns the sensor state. To demonstrate it in this paper, we focus on three sensor states:

> ***Definition 5 [sensor state]*** *a sensor s_i can be in one of the following states:*
> *"stuck" - $\forall v_{i_x}, v_{i_y} \in H_i^{t_2}\ v_{i_x} = v_{i_y}$*
> *"drift" – the values in $H_i^{t_2}$ show a movement towards higher values or a movement towards lower values.*
> *"ok" – otherwise.*
> *The function $\sigma: (H^{t_2}, s_i) \rightarrow \{stuck, drift, ok\}$ returns the state of s_i according to its latest $m/2$ values (H^{t_2}).*

Since sensors are noisy, a drift is not necessarily a smooth movement towards higher or lower values. A simple linear regression can be used to indicate the slope of the drift. The function σ returns a drift state if the slope is higher than a threshold value.

The reason we focus on stuck and drift faults is that these are common to single dimensional sensors ([1][5][13]) and yet are hard to be classed as faults. For example, the altimeter gage in a UAV might appear to be stuck when the UAV is maintaining its altitude, or appear to be drifting when the UAV is gaining

[1] The Pearson function returns a value (-1..1), the threshold is a user defined value (0..1) e.g. 0.9.

altitude; both are reactions to the UAV's normal behavior. However, if the altimeter gage appears to be drifting while the UAV is maintaining altitude or the altitude gage is stuck while the UAV is gaining altitude, then these are the expressions of a fault.

If a sensor's state is changed to "stuck" or to "drift" we cannot conclude that it is due to a fault, since the values could still be expressing a reaction to the system's normal behavior. However, the sensor is considered as uncertain.

Definition 6: [uncertain sensor] *given the sensor state of* s_i *at time* t $\sigma:(H^{t_2}, s_i)$, *then if* $\sigma:\left(H^{(t+1)_2}, s_i\right) \neq$ *"ok" and* $\sigma:(H^{t_2}, s_i) \neq \sigma:(H^{(t+1)_2}, s_i)$ *then* s_i *is declared as an uncertain sensor. We use the set* L^t *to denote the set of uncertain sensors at time* t. *In addition* $\mathcal{L}^t = \bigcup_{i=t-x}^{t} L^i$ *contains all sensors that were marked as uncertain in the last* x *time steps.*

By extracting information out of the correlations between sensors in $\mathcal{P}_H$ and the system's structural model M, we can conclude whether or not the recognized suspicious pattern of a sensor is due to a fault.

In the next subsection we describe how the online preprocessing described here is used in the fault detection and diagnosis procedures.

4.2 Fault Detection

Following the previous subsection, a detection of a suspicious pattern in a sensor is not sufficient to implicate the sensor as faulty. We should still investigate whether it reflects a normal behavior or a fault. We propose to use the correlated sensor set to indicate a failure. A high correlation rate between two sensors dependent on two different components in the structural model can be the result of: (1) Redundant sensors (dependent on different internal components) that measure the same thing. For example, the altimeter and the GPS indicated altitude. If one system fails the other can be used as a backup. (2) Sensors that react to the same action of the robot. For example the *Pitch angle* and the *climb rate indicator* are correlated as the UAV's elevator is invoked.

In these two cases one sensor can either implicate or clear an uncertain correlated sensor of suspicion. If one sensor is faulty or displays a faulty behavior due to a dependency on a faulty component, then it is reasonable to assume that the other sensor was not affected by the fault and still reflects the robot's behavior. The same cannot be said about two correlated sensors that share a component dependency, since both sensors can be affected by the same fault.

For instance, consider that the altimeter shows a drifting state. If it is the result of the UAV's climbing then the GPS indicated altitude (which is dependent on a different component and was determined as correlated to the altimeter) also changed its state to drift and the altimeter is cleared of suspicion (see Fig.2). However, if the drift was a result of a fault, and not of the UAV's behavior, then every other correlated sensor from another internal component dependency should not be affected by the fault, and therefore poses a different state than the altimeter. In this case, we declare a fault (see Fig.3).

It is important to guarantee first that there is no possibility to clear the uncertain sensor of its suspicion (i.e. look for another correlated sensor that do not share component dependency but has the same state). Only then we look for an implicating sensor (i.e. a correlated sensor that does not share component dependency and has a different state) in order to verify the failure of the uncertain sensor.

Consider, for instance, that a UAV is taking off the runway. Before it gained altitude, the altimeter which is dependent on the static system, and the heading indicator which is dependent on the

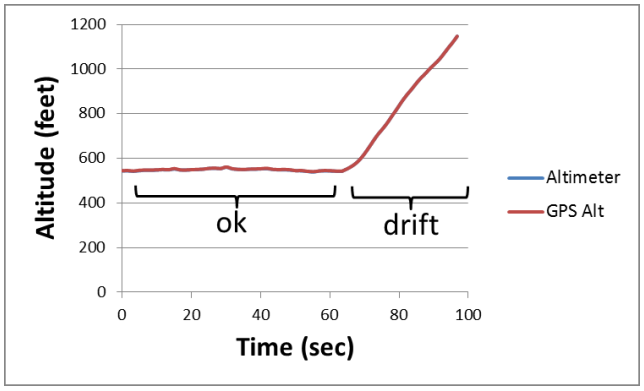

Figure 2: both Altimeter and GPS altitude are "drifting"

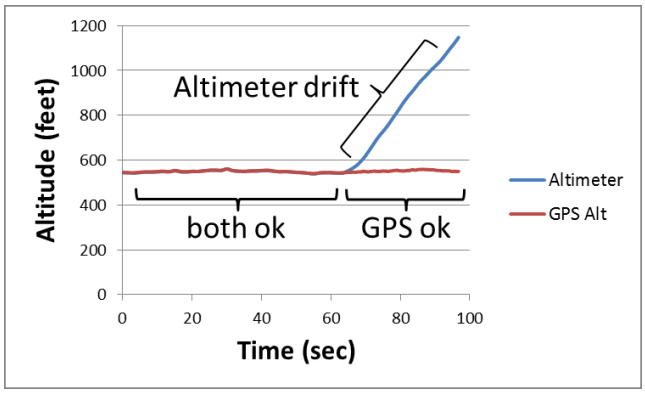

Figure 3: Altimeter "drifts" while GPS is not "drifting"

vacuum system, were both correlated (since both were idle). But then while taking off the altimeter appears to drift, while the heading indicator does not. This case would have implicated the altimeter unless the GPS indicated altitude was also found drifting, thereby clearing the altimeter of suspicion.

Algorithm 1 presents a fault detection algorithm that determines whether a sensor with a suspicious pattern state is the result of a fault. The fault detection algorithm is invoked with each input reading of the sensors. The algorithm obtains as an input the updated sliding window H^t, the updated set $\mathcal{P}_{H^t}$ that contains per each sensor s_i a set of s_i's correlated sensors P_{i,H^t}, and the updated uncertain sensors set L^t.

The algorithm iterates through every uncertain sensor (line 3). C_i is the set of internal components that s_i is dependent on (line 4). C_i is extracted from the structural model M by using the mapping function φ (Definition 2). Lines 5-8 try to find a sensor that clears the uncertainty of s_i, i.e. whether there exist a sensor s_j correlated to s_i that share the same state but is dependent on different components. We check every sensor s_j in s_i's correlated set P_{i,H^t} (line 5). C_j is the set of components that s_j is dependent on (line 6). If s_i and s_j share the same state ($\sigma:(H^{t_2}, s_i) = \sigma:\left(H^{t_2}, s_j\right)$) and do not share dependent components ($C_i \cap C_j = \phi$) then s_i is cleared of suspicion (lines 7-8).

If the algorithm did not return, then lines 9-14 check if an implicating sensor exists, i.e. whether exists a sensor s_j correlated to s_i that does not share the same state and is dependent on different components. In the same manner we iterate through each one of s_i's correlated sensors (line 10). This time we search a

Algorithm 1: Fault Detection Algorithm

1. **Input:**
   $H^t - m \times n$ sliding window at time t
   $\mathcal{P}_{H^t}$ - the sets of correlated sensors at time t
   L^t- the uncertain sensors set at time t
2. **Output:** a fault detection report
3. **For each** sensor $s_i \in L^t$
4. $C_i \leftarrow \varphi(s_i)$
5. **For each** sensor $s_j \in P_{i,H^t}$
6. $C_j \leftarrow \varphi(s_j)$
7. **If** $\sigma\colon(H^{t_2}, s_i) = \sigma\colon\left(H^{t_2}, s_j\right) \wedge C_i \cap C_j = \phi$
8. **Return**;
9. $suspected \leftarrow false$
10. **For each** sensor $s_j \in P_{i,H^{t+1}}$
11. $C_j \leftarrow \varphi(s_j)$
12. **If** $\sigma\colon(H^{t_2}, s_i) \neq \sigma\colon\left(H^{t_2}, s_j\right) \wedge C_i \cap C_j = \phi$
13. $suspected \leftarrow true$
14. **Break**;
15. **If** $suspected$
16. **Report** "fault detected, " , s_i, " is suspected"

sensor s_j that shares the same state as s_i but does not share component dependency (line 12). If such a sensor is found then the search is stopped (line 14) and a fault is reported (lines 15,16).

Upon fault detection, the diagnosis procedure is invoked. The diagnosis procedure is described next.

4.3 Diagnosis

In the previous subsection we described how an uncertain sensor s_i becomes suspected for a fault. The following diagnosis algorithm (Algorithm 2) is invoked upon fault detection. The algorithm is invoked with the suspected sensor s_i as an input. The fact that s_i is suspected and not any other s_j that was correlated to s_i is because s_i was the sensor that changed its state to drift or stuck while the other correlated sensors did not reflect the same state. Since we concluded that the change of s_i's state was not a reaction to the robot's behavior then we determine that s_i is suspected for a fault.

When a sensor is reported of having a stuck or drift state, it can either be a single sensor fault, or an expression of a fault of an internal component that the sensor is dependent on. If an internal component is faulty then all of its dependent sensors should display a faulty behavior (Definition 1). Therefore, we check for each of the other sensors that are dependent on the same internal component if they are suspected (i.e. changed their state into drift or stuck). If so, we can include the internal component in the report.

For example, when the heading indicator was reported as suspected for a fault, we check if the attitude indicator's *pitch, roll* and *yaw* changed their state as well (see figure 1). If so, then we also report the *vacuum system* as suspected of having a fault.

Algorithm 2: Diagnosis Algorithm

1. **Input:**
   s_i – the suspected sensor
   σ_i- the state of s_i
   $\mathcal{L}^t$ – the set of all uncertain sensors from the last x time steps
2. **Output**: a diagnosis report
3. **Report** s_i is a candidate with a fault state σ_i
4. $C_i \leftarrow \varphi(s_i)$
5. **For each** $c_k \in C_i$
6. $S_d \leftarrow \tau(c_k)$
7. $p \leftarrow \frac{|S_d \cap \mathcal{L}^t|}{|S_d|}$
8. **Report** c_k is a candidate with probability p
9. **Return;**

The diagnosis process reports s_i as a faulty sensor (line 3). Then, it extracts from the structural model the internal components that s_i is dependent on (line 4). For each of those internal components (line 5) the diagnosis process determines their probability of being faulty according to the number of their dependent sensors that are uncertain (are in the suspect set $\mathcal{L}^t$). S_d is a set containing the component c_k's dependent sensors (line 6). We report c_k as having the probability of being faulty as the ratio between the number of its dependent sensors that are suspected $|S_d \cap \mathcal{L}^t|$ and the total number of c_k's dependent sensors $|S_d|$ (line 7).

Since an internal component fault might be expressed by its dependent sensors in different time intervals, we use the suspected set $\mathcal{L}^t$, and return a probability of being fault. If we were to use L^t rather than $\mathcal{L}^t$, then only sensors that changed their state during this particular time frame could have implicate the component c_k. However, some of the sensors might have already changed their state and hence are not suspected during this particular time frame. This would result in a low probability of c_k faultiness. Therefore, we use $\mathcal{L}^t$ where suspected sensors remain for several time frames.

For example, a static system failure causes the altimeter to be stuck immediately and the vertical speed indicator to drift downwards a few seconds later. Since both sensors are in $\mathcal{L}^t$, the static system is reported at a probability of 1. If we were to use L^t, the static system would have a probability of 0.5 for being faulty.

The reason for returning a probability rather than determine c_k faultiness only if all its dependent sensors are suspected is due to the fact that some sensors might take very long time to react to c_k's failure while all others are already suspected. In this case we wish to report c_k and give an indication about the degree of its faultiness.

For example, when the electrical system fails, some instruments such as the GPS fail immediately, but the turn indicator in particular, will take 30-60 seconds to start drifting downwards. This is due to the fact that the unpowered gyro still spins, though slowly loosing speed. Hence the turn indicator's drift effect is yet

to show. In this case we would not like to clear the electrical system from all suspicion just because one instrument is yet to show suspicion. Therefore, we return the probability, which in this particular example is above 0.9.

5. EVALUATION

5.1 Experimental Setup

To evaluate our fault detection and diagnosis algorithms we use two domains. The first domain is a laboratory robot called Robotican1 (see Figure 4) [11]. The robotican1 has two 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 (aerials, ground underwater etc.) in which fault tolerance is very important for mission successful completion. When a sensor is damaged then another sensor can be used to fulfill the perception. Such is the case with UAVs where a set of different sensors measure the UAV's 3D location. If the GPS fails, other altitude gages, accelerometers and attitude gages can be used.

Robotican1 also has 5 degrees of freedom arm. Each joint is held by two electrical engines. These engines provide a sensed reading of the voltage applied by their action. To mimic some internal component depths we defined 3 abstract internal components: 1) sonar power supplier, 2) infrared power supplier, 3) arm power supplier.

Figure 4: Robotican1

We devised 17 different scenarios, which included a scenario without injected faults and scenarios that included different injected faults while the robot performed different tasks. Faults were injected to each type of sensor (motor voltage, infrared and sonar). The injected faults to the sensors were of type stuck or drift. These faults were injected to one or more sensors in different time intervals. We covered cases of faults to sensors that are dependent on the same components and on different internal components. Failing one of the three power suppliers described above causes each of the dependent sensors to fail. The robot's behavior was either to move, to stand still, or to move its arm to a given position.

The second domain is the *FlightGear* [2] high fidelity flight simulator (see Fig.5). This open source simulator is built for and used in academic research [3]. Furthermore, it realistically simulates flight instrumental faults such as an altimeter stuck, or system faults. For example, if the vacuum system fails, then the gyros responsible for the attitude indicator and the heading indicator slowly lose their spin speed, causing the indicators to drift slowly and deviate from the readings of the turn indicator and compass. These features make the FlightGear simulator to be very suitable to test our method.

We implemented an autopilot, which flies the aircraft according to its sensor readings. We used 16 flights that included 4 to 6 instrumental failures at different times while the UAV takes off and makes a few turns. We failed the altimeter, airspeed indicator and compass.

Figure 5: FlightGear Simulator Screenshot

On this data set of flights we evaluated (1) the contribution of using the structural model as an addition to the suspicious pattern recognition for fault detection (2) the accuracy of our algorithm comparing to similar competing fault detection approaches.

(1) As described in section 4, our fault detection technique consists of two parts, the Data Driven part which reports suspicious patterns in sensors, and the Model-Based part which uses a structural model to determine whether a suspicious pattern is a fault. We compared our fault detection algorithm to the same algorithm without the Model-Based part (i.e fault is reported whenever a suspicious pattern was recognized).

(2) We compared our approach to our previous fault detection approach presented in [7] (denoted here m.distance) and to another competing method for fault detection Local Outlier Factor (LOF) algorithm [9]. m.distance approach also utilizes a sliding window technique and Pearson correlation. However, to detect faults it uses Mahalanobis Distance [8] to compare the online input to the current data in the sliding window. The LOF algorithm is also an online density based outlier detection algorithm which uses the *K*-nearest neighbor to compare local density to the expected density and calculate the data instance measure of being an outlier accordingly.

To evaluate each approach, we calculated the detection rate and the false alarm rate. The best possible result for the detection rate is 1 indicating that all faults were detected. The best possible result for the false alarm rate is 0 indicating that no false alarms were raised.

In addition, we tested two scenarios for diagnosis – a takeoff and a free flight. For each scenario we tested every possible instrumental or system failure, and a combination of multiple faults. There are 8 types of instruments and 4 types of systems that can be failed. In total, we examined 16 flights for each scenario. We used these two data sets to test the diagnosis accuracy of our current approach.

5.2 Results

The evaluation of the fault detection over the 16 flights produced the following results: (1) without the use of the structural model in our fault detection algorithm the false alarm rate is very close to 1 whereas with the use of the structural model the false alarm rate is very close to 0, indicating the contribution of using the structural model. (2) All competing algorithms had a detection rate of 1 – all faults were detected. (3) Our proposed approach also diagnosed the failing sensor correctly. (4) The false alarm rates of the competing approaches are shown in Figure 6.

The proposed approach has a false alarm rate of 0.028. This rate is less than half of the false alarm rate of the m.distance approach. Moreover, most of the false reports of our proposed approach were produced by the same sensor. When this sensor is suppressed there are virtually no false alarms. The lower false

alarm rate is explained by the fact that the proposed approach uses the additional knowledge of component dependency to clear or implicate an uncertain sensor.

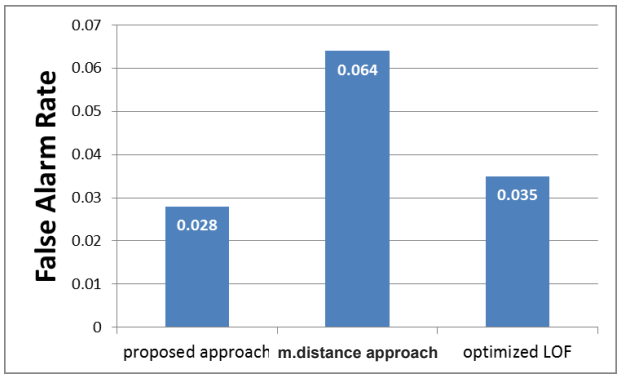

Figure 6: The False Alarm Rates for the approaches

The LOF algorithm returns the degree of a data instance being an outlier. Thus, a threshold is needed to label a data instance as faulty or healthy. Since the LOF algorithm does not have a policy to set these thresholds, we chose these thresholds in such a way that the results of the LOF algorithm will be optimized. Offline, considering the reports of an entire flight, we set the threshold as high as possible such that all faults are detected and thus false positives are minimized. The false alarm rate of our proposed approach is lower than the optimized false alarm rate of the LOF algorithm.

In the Robotican domain we tested 17 diagnosis scenarios. The results are a fault detection rate of 0.96 and a false alarm rate of 0.013. One fault out of 26 faults was not detected. The faulty sensor was suspected but was cleared after another correlated sensor of a different component dependency shared the same state. All detected faults were diagnosed correctly, i.e. the sensors and internal components that were reported matched the injected faults and the report was given at the time of the fault injection.

Figure 7 illustrates the proposed approach results of the two domains in an ROC chart[2]. an *ROC* chart describes the entire space of fault detection: the *X*-axis is the false alarm rate and the *Y*-axis is the detection rate. A classifier is expressed as a *2D* point. The perfect anomaly detector is located at point (0,1). In both domains our proposed approach is very close to the perfect fault detector (theoretically with a detection rate of 1 and a false alarm rate of 0).

In the FlightGear domain we evaluated two more scenarios, a takeoff and a free flight, 16 flights per each scenario. These scenarios were used to further evaluate the diagnosis aspect of the proposed approach. All instrumental failures and system failures were diagnosed. We would like to elaborate on the following study cases which show the need for various aspects of the diagnosis algorithm, advantages and disadvantages.

[2] Note that to produce a better view the scale of the false alarm rate reach 0.1 (and not 1)

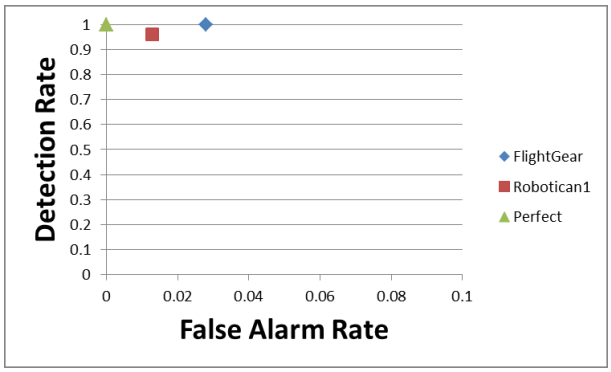

Figure 7: ROC chart of the two domains

Case 1: a static system failure causes the altimeter to be stuck, and the airspeed indicator to drift down to 0 a few seconds later. The static system was suspected at a probability of 1 due to the fact that suspected sensors remain suspected for a given time (a few seconds). The drift of the airspeed indicator caused the pitot system to be suspected as well and included in the diagnosis.

Case 2: a failure to the pitot system causes the airspeed indicator to drift upwards, unless there is a failure to the static system as well, which causes the airspeed indicator to be stuck. Note that the proposed approach is unaware of these rules, but still recognizes these effects as suspected faults and reports the pitot system as suspect when needed.

Case 3: a failure to the electrical system causes many instruments to fail immediately. But the turn indicator starts to drift downwards only after 30 seconds to 1 minute and is yet to be detected. This case justifies the use of probability to determine a suspected internal component. The electrical system is suspected in a probability greater than 0.9.

Case 4: a failure to the attitude indicator (and not the vacuum system) causes some sensors to fail i.e. pitch, roll and yaw angles. The proposed approach reported that the vacuum system is suspected at a probability of 0.8. The heading indicator which is also dependent on the vacuum system is healthy. If the proposed approach did not use a probability then the heading-indicator's health would have cleared the vacuum system suspicion. This case does not justify the use of a probability. However, the heading indicator may yet fail as in case 3 and thus a probability is used.

6. Discussion

Redundancy and correlation: since the approach tries to detect faults to single-dimensional sensors, it relies heavily on the additional data provided by redundant sensors. The correlation is used to indicate which sensors are redundant. When the knowledge of redundant sensors is present, we suggest using it instead.

However, some correlations between irredundant sensors might also help in fault detection. For example, a climbing rate sensor is usually correlated to the pitch sensor even though they are not redundant with respect to each other. Yet, possessing different states (e.g. pitch is "ok", climbing rate is "drifting" down) might indicate a fault (e.g. climbing rate is faulty or worse, an aircraft stall).

If attributes that calculate the expected value of a sensor are present, then they can be used as "redundant sensors" as well. For instance, the attribute *speed* calculated from GPS position samples can be used as a redundant sensor to a speed sensor.

Suspicious patterns: as described in the paper the fault detection algorithm is provided with a function σ that recognizes known suspicious patterns in a sensor data. Therefore, σ is domain specific and should be implemented according to the expression of known faults in the system's sensors. Since σ is a part of the approach's input the approach is still general and not contained strictly to patterns such as "drift" and "stuck". σ also handles noise issues. For instance, in the tested domains the "drift" state was calculated with a linear regression slope. The only concern of the fault detection approach is to determine whether a suspicious pattern is a reaction to a behavior of the system or a reaction to a fault.

Algorithm parameters: the fault detection algorithm and the diagnosis algorithm used different parameters. These parameters are domain specific, and should be adjusted according to a labeled data set.

Setting the correlation threshold too low could result in more false positives as well as false negatives since irrelevant sensors are used to implicate or clear uncertain sensors of suspicion. Setting the threshold too high could result in a higher rate of false negatives due to approaches reliance of redundancy.

The sliding-window size affects the time of calculation as well as the false negatives rates. Setting it too small could result in insufficient data size to determine a good correlation by *Pearson* or recognize a suspicious pattern by the σ function. We suggest using the largest size which is in the capabilities of online calculation in the system.

The time (x) a sensor remains suspected (in $\mathcal{L}^t$) affects the implication of an internal component. The longer the time is, the more likely it is for an internal component to be included in the diagnosis. Since past suspicions become, in time, irrelevant to current events then x should be limited. The setting of x is domain specific and should be learned from labeled diagnosis data set.

Structural model and diagnosis: the structural model presented in this paper may appear as simple partition to subsystems. In reality, a structural model contains several levels of depth describing component dependency (e.g. a sensor is dependent on a component that is dependent on another component). The diagnosis algorithm can be applied recursively, implicating each component at level i, if all its dependent components at level $i+1$ are suspected of having a fault. Thus, a component level fault is unmasked.

The proposed approach gives sensors an even weight when implicating a suspected component. Different weights can be used in the model according to the sensor's ability to indicate that the internal component is failing. In the FlightGear domain we did not monitor sensors that would have made the fault detection and diagnosis too easy. The vacuum intake sensor could easily implicate the vacuum system, and the voltmeter and ampermeter sensors could easily implicate the electrical system. We suggest modeling these kinds of sensors with a high weight.

7. CONCLUSION

We showed an approach that when given a structural model and sensor readings it can detect sensor related faults that occur over time and diagnose them online with high precision. We showed how a structural model is used diagnose internal components. We evaluated the approach on physical and simulated domains. We described study cases which show the advantages and disadvantages of the different aspects of the proposed approach.

The advantages of the approach are the ability to distinguish a fault from a normal behavior when a suspicious pattern is recognized in a sensor, the ability to detect faults that occur over a period of time, and the ability to successfully diagnose root causes. The disadvantages are the reliance on redundancy, the reliance on predefined suspicious patterns, and the return of the degree of faultiness of internal components rather than a deterministic answer.

We believe that further optimizations can be made as described in the discussion section.

8. REFERENCES

[1] ADAPT system http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/adapt-diagnostics/

[2] FlightGear, http://www.flightgear.org/

[3] FlightGear in Research - http://www.flightgear.org/Projects/

[4] Hashimoto M. (2005) A multi-model based fault detection and diagnosis of internal sensors for mobile robot. Intelligent Robots and Systems ,pp.3787- 3792.

[5] International Diagnostic Competition - http://sites.google.com/site/dxcompetition2011/

[6] Isermann R. (2005). Model-based fault-detection and diagnosis—Status and applications. Annual Reviews in Control, 29(1), 71–85.

[7] Khalastchi E., Kalech M., lin R. and Kaminka G. Online Anomaly Detection in Unmanned Vehicles. The Tenth International Conference on Autonomous Agents and Multi-Agent Systems, p. 115-122 (2011).

[8] Mahalanobis P. C. (1936). On the generalized distance in statistics. The National Institute of Science, pages 49–55.

[9] Pokrajac D. (2007) Incremental local outlier detection for data streams. IEEE Symposium on Computational Intelligence and Data Mining.

[10] Robotcup, http://www.robocup.org/

[11] Robotican http://www.robotican.net/

[12] Steinbauer G. a survey on the nature of faults of autonomous robot systems http://www.ist.tugraz.at/rfs/index.php/Main_Page

[13] Varun C., Arindam B. *Anomaly detection: A survey*. The Association for Computing Machinery, Computing Surveys, 41(3):1–58. (2009)