Knowl Inf Syst (2015) 43:657–688
DOI 10.1007/s10115-014-0754-y

REGULAR PAPER

Online data-driven anomaly detection in autonomous robots

Eliahu Khalastchi · Meir Kalech · Gal A. Kaminka · Raz Lin

Received: 9 September 2013 / Revised: 10 February 2014 / Accepted: 3 May 2014 /
Published online: 21 May 2014

Abstract The use of autonomous robots is appealing for tasks, which are dangerous to humans. Autonomous robots might fail to perform their tasks since they are susceptible to varied sorts of faults such as point and contextual faults. Not all faults can be known in advance, and hence, anomaly detection is required. In this paper, we present an online data-driven anomaly detection approach (*ODDAD*) for autonomous robots. *ODDAD* is suitable for the dynamic nature of autonomous robots since it declares a fault based only on data collected online. In addition, it is unsupervised, model free and domain independent. *ODDAD* proceeds in three steps: data filtering, attributes grouping based on dependency between attributes and outliers detection for each group. Above a calculated threshold, an anomaly is declared. We empirically evaluate *ODDAD* in different domains: commercial unmanned aerial vehicles (UAVs), a vacuum-cleaning robot, a high-fidelity flight simulator and an electrical power system of a spacecraft. We show the significance and impact of each component of *ODDAD*. By comparing *ODDAD* to other state-of-the-art competing anomaly detection algorithms, we show its advantages.

Keywords Anomaly detection · Robotics · UAV · UGV · Unmanned vehicles · Autonomous agents · Unsupervised · Model free · Online · Data driven · ODDAD · AI · Fault detection

1 Introduction

Autonomous robots operate in dynamic environments, where it is impossible to foresee and impractical to account for all possible faults. Instead, the control systems of the robots must be complemented by anomaly detection systems that can detect anomalies in the robot's systems and trigger diagnosis. To be useful, such a system has to be computationally light so

E. Khalastchi (✉) · M. Kalech
Department of Information Systems Engineering, Ben-Gurion University of the Negev, Be'er Sheva, Israel
e-mail: khalastc@bgu.ac.il

G. A. Kaminka · R. Lin
The MAVERICK Group, Department of Computer Science, Bar Ilan University, Ramat Gan, Israel

that it does not create a computational load on the robot, which itself can cause failures, and has to detect faults with high degrees of both precision and recall. High rates of false positives will lead operators to ignore the system [15]; low rates of true alarms make it ineffective. Moreover, the faults must be detected quickly after their occurrence, so that they can be dealt before they become catastrophic. In addition, an anomaly detector should be able to detect *contextual failure* [2]. A contextual failure occurs when a faulty sensor reports valid readings, which are invalid given some operational or sensory context. For instance, a sensor can get physically stuck such that it no longer reports the true value of its reading, but does report a value which is in the range of valid readings.

The field of anomaly detection has been widely researched. Anomaly detection approaches are typically divided into three categories: model based, knowledge based and data driven (machine learning). Model-based algorithms are potentially very accurate. However, these methods heavily rely on the fidelity of the underlying model which is very hard to construct for complex autonomous systems. Knowledge-based algorithms associate symptoms with diagnosis by rules (IF-THEN sentences). This approach is usually good in detecting predefined faults but not unknown ones. Data-driven algorithms usually rely on statistical information (taken out of the set of the training data) to detect outliers and label them as faults. These statistical methods rely on the existence of faults which are expressed as outliers in the training data and usually have to use some dimension reduction techniques to handle all the available data [2].

In this paper, we apply an online data-driven anomaly detection approach (hereinafter *ODDAD*). By online, we mean that with each consumed sampled input, we look at the latest m samples of input (sliding window) and quickly decide whether or not this online data present a fault. To determine the occurrence of a fault, we proceed in three steps: (1) The input is filtered to reduce noise, (2) we apply dimension reduction by splitting the data into sets of correlated attributes and (3) we use the Mahalanobis Distancecalculation to each set in order to return the degree of a data instance being an outlier. Above a calculated threshold, we declare an anomaly.

ODDAD includes the important properties that are necessary for a quick and accurate anomaly detector in the domain of autonomous systems: (a) model free—no analytical description of the system is needed (like model-based approaches require), (b) domain independent, (c) unsupervised—no training set is needed and (d) completely online—no processing of old and offline large data sets is needed. The last property is especially important for the following reasons: firstly, current data are used for comparison (rather than obsolete data); secondly, there is a consideration of the dynamic nature of correlations which are built and destroyed as the robot operates; and lastly, the computation is kept light since only a portion of the current online data is used.

ODDAD differs from previous data-driven approaches. For example, in a similar way to our approach, Lin et al. [16] use the Mahalanobis Distanceto detect faults. However, the correlations are calculated offline and thus do not consider the dynamic nature of these correlations. It is important to detect correlated attributes online as we show in the results section.

The contributions we show in this paper are as follows: (1) We present a novel accurate anomaly detection algorithm which is lightweight, unsupervised, online, domain independent and model free. (2) We enable the use of the Mahalanobis Distancecalculation as an anomaly detector in the domain of autonomous systems. (3) We present a comprehensive set of experiments which show the success of *ODDAD* in various domains of autonomous systems.

To evaluate *ODDAD*, we conduct experiments in four different domains:

- Actual flight data from a commercial unmanned aerial vehicle (UAV), in which simulated faults were injected by the manufacturer.
- Data from a RV-400 vacuum-cleaning robot that was subjected to physical faults.
- The *FlightGear* flight simulator, which is widely used for research [4,7,26] and is able to simulate real faults.
- An electrical power system (EPS), which simulates the functions of a typical aerospace vehicle power system, provided by the Advanced Diagnostics and Prognostics Testbed (ADAPT) laboratory at the NASA Ames Research Center [13].

In all, we conducted experiments that show that *ODDAD* is superior to other state-of-the-art anomaly detectors. In addition, we show the impact of each one of the components of *ODDAD* (sliding window, filtering and online correlation detection). Finally, we show that the filtering we use also improves other competing approaches.

This paper proceeds as follows: In the next section, we present related work. In Sect. 3, we present the online anomy detection problem, and in Sect. 4, we describe our online data-driven anomaly detection (ODDAD) approach. Section 5 specifies the experimental setup of our domains and presents the results. Section 6 concludes.

2 Related work

Anomaly detection has generated substantial research over past years. Applications include intrusion and fraud detection, medical monitoring, robot behavior novelty detection, fault detection in physical systems, etc. (see [2] for a comprehensive survey).

Steinbauer et al. [27] conducted a survey about faults of autonomous robots in the context of RoboCup [25]. He presents an adapted fault taxonomy suitable for autonomous robots and gives information on the nature, the relevance and impact of faults in robot systems. All the surveyed types of faults and failures are expressed as anomalies in the monitored data. An anomaly detection can be used as a fault detection.

A known approach to anomaly detection relies on a model of the system. Model-Based approaches for fault detection and diagnosis are studied by two distinct and parallel research communities, the FDI community [11] and the DX community, e.g., in the work of [33]. FDI approaches are derived from control theory and statistical decision making. DX approaches are derived from Computer Science and Artificial Intelligence. Both approaches rely on a model that describes the system in order to detect faults or anomalies. However, the concepts, assumptions and techniques of the two approaches are very different.

Trav-Massuys [32] presents in her extensive survey fault detection and diagnosis as it is understood in the Control (FDI) and Artificial Intelligence (DX) fields, and exemplifies how different theories of these fields can be synergistically integrated to provide better diagnostic solutions and to achieve improved fault management in different environments.

Daigle et al. [5] propose an event-based approach for diagnosis of parametric faults in continuous systems. Their approach is based on a qualitative abstraction of deviations from the nominal behavior, i.e., expected behavior. Their technique is based on a finite automaton under the assumption that it is feasible to create a model that captures all relevant system behavior. Yet, in contrast to *ODDAD*, their approach isolates only a single fault. Another model-based approach for anomaly detection is model-based reasoning (e.g., [10,28]). As most model-based approaches, model-based reasoning requires having a model of the robot and its interactions with the environment. Such models are complex to build.

One main direction in anomaly detection uses machine learning methods to model what constitutes nominal behavior and derive from the representation of the nominal behavior the abnormal behavior. For example, Eski et al. [6] presented an experimental investigation on an industrial robot manipulator, using neural network for analyzing the vibration condition on joints. The use of a neural network requires a training set of samples which is already classified, i.e., the approach is a supervised approach. Because of the dynamic context in which autonomous robots operate, classified data sets are not common. In addition, the adaptation of machine learning algorithms to different contexts requires fine tuning of many parameters and thresholds. For this reason, an anomaly detection approach should be unsupervised. We present an unsupervised approach and demonstrate that it can be easily adapted to different contexts, while preserving high anomaly detection rates.

One-class classification-based anomaly detection methods assume all training data instances to be of one class label. Such methods learn a discriminative boundary around the nominal instances using a *one-class classification algorithm* [2,9]. Any test instance that falls outside the learnt boundary is considered as anomalous. Support vector machines (SVMs) [17,29], as other machine learning techniques, need additional computation to calculate this boundary in the one-class setting [19,24].

However, as we show in Sect. 5.4, contextual anomalies are undetected even by a successful and well-known classifier such as *SVM*. Even under unrealistic favoring conditions, where both nominal and anomalous data samples are available for training, our proposed unsupervised method detects such anomalies. When labeled data are scarce both the unlabeled and labeled data instances are utilized to train and update the classification model [20]. Our approach is unsupervised and hence has no need for updating such a classification model.

The large amount of data of monitored Unmanned Vehicles (UVs) is produced from a large number of system components comprising of actuators, internal and external sensors, odometry and telemetry, that are separately monitored at high frequency. The separated monitored components can be thought of as dimensions, and thus, a collection of monitored readings, at a given point in time, can be considered a multidimensional point. Therefore, statistical methods that produce an anomaly score for each given point can use calculations that consider the points' density, such as Mahalanobis Distance [16] or K-Nearest Neighbors [23]. We use such methods in this paper. Statistical approaches usually assume that the data are generated from a single distribution, which is not the case for high-dimensional real data sets [2].

To distinguish the inherent noisy data from anomalies, Kalman filters are usually applied (e.g., [3,8,30]). Since simple Kalman filters, when used alone, might produce a large number of false positives, additional computation is used to determine an anomaly. For example, Cork and Walker [3] present a nonlinear model, which together with Kalman filters tries to compensate for malfunctioning sensors of UAVs. We use a simpler filter that significantly improved the results of our approach. The filter normalizes values using a Z-score transformation.

Laurikkala et al. [14] propose the use of Mahalanobis Distanceto reduce the multivariate observations to univariate scalars. In this work, we reduce dimensionality with dependency detection techniques and use the Mahalanobis Distanceto return a scalar which depicts the degree of anomaly for a given data instance.

Brotherton and Mackey [1] use the Mahalanobis Distanceas the key factor for determining whether signals measured from an aircraft are of nominal or anomalous behavior. They describe a limitation arose as a result of the number of dimensions used due to run-time issues. We address this challenge in this paper by dividing the number of dimensions into correlated groups.

Although Mahalanobis Distanceseems fitting for anomaly detection, it is not commonly used for that aim. The challenge of reducing run-time and false positives must be handled. In this work, we meet this challenge by applying dimension reduction and noise filtering. Thus, making the Mahalanobis Distancea successful anomaly detector.

This work is an extension of a previous paper [12]. One extension in this work is in evaluating our methods on an additional domain of electrical power system (EPS) described in [13]. In addition, we compare the online approach to the *incremental Local Outlier Factor* (*LOF*) algorithm [23]. We chose to compete with the incremental *LOF* algorithm since it is one of the leading online outlier detectors which handles multivariate data and considers the data distribution in a different way than our proposed online approach. Finally, we compare *ODDAD* to the offline anomaly detector proposed in [16].

3 The problem of online anomaly detection

We deal with the problem of online anomaly detection. Let $A = \{a_1, \ldots, a_n\}$ be the set of attributes that are monitored. The attributes can be collected by internal or external sensors (e.g., odometry, telemetry, speed, heading, GPS_x, GPS_y, etc.) The data are sampled at some frequency f. An input vector $\vec{i_t} = \{i_{t,1}, \ldots, i_{t,n}\}$ is given online, where $i_{t,j} \in \mathbb{R}$ denotes the value of attribute a_j at current time t.

In our model, past data H are also accessible and considered as nominal. H is an $m \times n$ matrix where the columns denote n monitored attributes and the rows maintain the values of these attributes over m time steps. H can be recorded from a complete operation of the UV that is known to be nominal (e.g., a flight with no known failures), or it can be created from the last m inputs that were given online, that is $H = \{\vec{i}_{t-m}, \ldots, \vec{i}_{t-1}\}$. The online anomaly detection problem is to decide for each given $\vec{i_t}$, whether or not $\vec{i_t}$ is anomalous with respect to H.

The definition of anomalous varies, but is typically given in the form of statistical outlying. Three common categories of anomalies are described in the literature [2]:

1. *Point anomalies*: invalid data instances, corresponding to invalid values in $\vec{i_x}$.
2. *Contextual anomalies*: data instances that are only invalid with respect to a specific context but not otherwise. In our approach, the context is provided by the data of H which changes over time.
3. *Collective anomalies*: related data instances that are valid apart, but create an invalid collection. In our approach, H holds a timeseries of values for each monitored attribute. An invalid collection may contradict other collections in H.

We demonstrate the anomalies using examples. Consider a UAV that collects and monitors n attributes, such as: air-speed, heading, altitude, roll pitch and yaw, and other telemetry and sensors data. The input is provided in a given frequency (usually 10 Hz), when suddenly a fault occurs.

A point anomaly exists when a data instance shows an invalid value, for instance, a fault may cause the air-speed indicator to produce values above the maximum air-speed that is achievable by the UAV. *A contextual anomaly* exists when a data instance, though valid on its own, is considered invalid with respect to a certain context. For instance, the air-speed indicator produces the value of 0, which is a valid value on the ground but not expected while in flight. *A collective anomaly* exists when values which may be valid on their own create together an invalid collection. For instance, the altimeter (measuring altitude) is suddenly stuck while the UAV is taking off. Each value is valid on its own and is also legitimate within the context of the take-off behavior of the UAV. However, the series of unchanged values is anomalous when the values are expected to grow. Our goal is to detect these failures by flagging them as anomalies

4 Online data-driven anomaly detection (ODDAD)

In this section, we describe the different components of the our *ODDAD* approach. Each subsection describes a different component. We start by describing the flow of the *ODDAD* approach in Sect. 4.1. The rest of the Sects. (4.2–4.7) are placed in an order that best describes the motivation for each component of *ODDAD* , not necessarily according to *ODDAD*'s flow. Finally, in Sect. 4.8, we describe the whole process of *ODDAD* as an anomaly detector.

4.1 *ODDAD* approach overview

Figure 1 illustrates the flow of *ODDAD*. In the first stage, the current online consumed input is filtered in order to reduce noise that might be falsely interpreted as an anomaly. The filtering is described in Sect. 4.7.

Then, the last m filtered inputs are kept in a sliding window H, which is described in Sect. 4.4. We assume that the robot starts by operating normally and an anomaly might occur after some point in time. Therefore, we assume the data of the sliding window are nominal and we wish to compare it against the current input to detect anomalies as they occur. For this comparison, we use the Mahalanobis Distancecalculation because of its multidimensional nature and consideration of distribution (see Sect. 4.2).

The Mahalanobis Distancecannot be used on its own as an anomaly detector. In order for the Mahalanobis Distanceto be a successful anomaly detector, it should be applied on correlated dimensions as we discuss in Sect. 4.2. Therefore, the data in the sliding window H are split into sets of correlated attributes as explained in Sect. 4.5. In Fig. 1, columns with the same color are correlated (i.e., attributes {1,4,6} , {2,5} and {3,7} are correlated). Then, each set is associated with a calculated anomaly threshold as described in Sect. 4.6. These thresholds are later used for singling out anomalies.

The data of the current input vector are split into subvectors to match the partitioning of H. In Fig. 1, the input values of attributes {1,4,6}, {2,5}, {3,7} form the subvectors that match the first, second and third sets of correlated attributes, respectively. Each subvector is compared to the data of its matching set of correlated attributes using the Mahalanobis Distancecalculation. Any result that crosses a set's anomaly threshold triggers an anomaly alarm. This process of anomaly detection is described in Sect. 4.8.

4.2 Mahalanobis distance as an anomaly detector

Mahalanobis Distanceis an n dimensional Z-score. It calculates the distance between an n dimensional point to a group of others in units of standard deviations [18]. In contrast to the common n dimensional Euclidean distance, Mahalanobis Distancealso considers the points' distribution. Therefore, if the group of points represents a distribution of observed sampled data, then the Mahalanobis Distanceindicates whether a new point is an outlier with respect to this observation. A point with similar values to the observed points is located in the multidimensional space, within a dense area and will have a lower Mahalanobis Distance. However, an outlier will be located outside the dense area and will have a larger Mahalanobis distance.

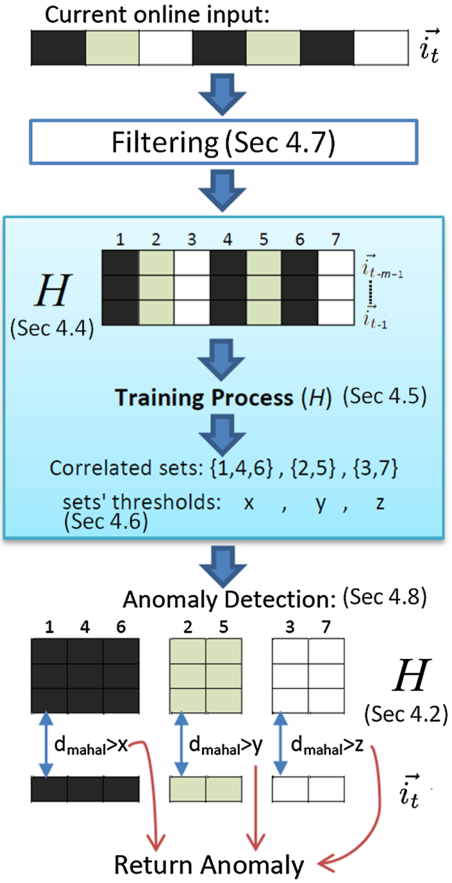


Fig. 1 The outline of the approach

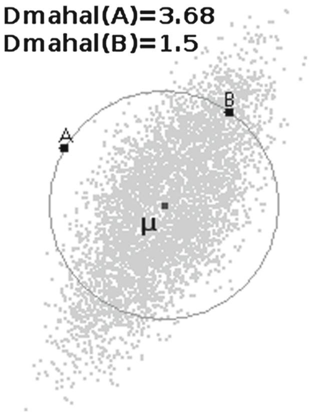


Fig. 2 Euclidean versus Mahalanobis Distanceof points A, B with respect to the *gray points*

An example is depicted in Fig. 2. We can see that while A and B have the same Euclidean distance from the centroid μ, A's Mahalanobis Distance(3.68) is greater than B's (1.5), because an instance of B is more probable than an instance of A with respect to the other points.

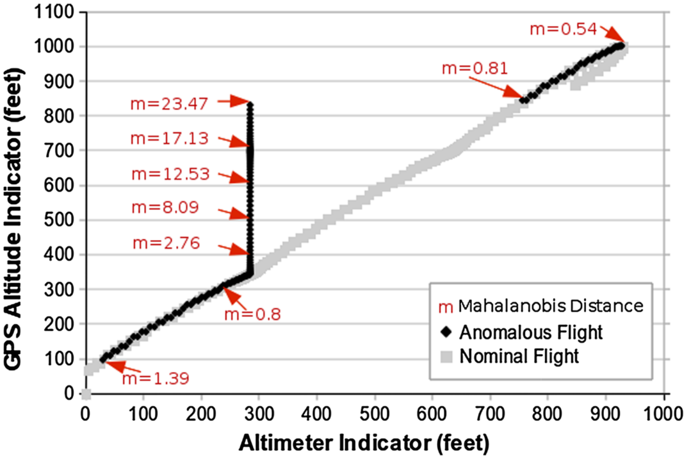


Fig. 3 Example of Mahalanobis Distanceas an Anomaly Detector. m = how anomalous a *black point* is with respect to the distribution of *gray points*. When the altimeter gets stuck the value of m gets higher

Formally, the Mahalanobis Distanceis calculated as follows: Let $\vec{i'_t}$ be a vector containing the current input values of k correlated attributes at time step t. Let H' be an $m \times k$ matrix which contains the values for these k correlated attributes over the last m time steps (i.e., $t - m, \ldots, t - 1$); each $H'_{s,a}$ is the nominal value of attribute a in time step s. We define the mean of H' by $\mu = (\mu_1, \mu_2, \ldots, \mu_k)$ where μ_j is the average of the m values of attribute j stored in H'. We define S as the covariance matrix of H'. The Mahalanobis Distance, D_{mahal}, from $\vec{i'_t}$ to H', is defined as:

$$D_{mahal}(\vec{i'_t}, H') = \sqrt{(\vec{i'_t} - \vec{\mu})S^{-1}(\vec{i'_t}^{T} - \vec{\mu}^{T})}$$

Using the Mahalanobis Distance, we can detect the three common categories of anomalies discussed above (point, contextual and collective).

A sudden *point anomaly* of an attribute yields an invalid value in $\vec{i'_t}$ which was not observed in the attribute's previous valid values stored in H. The invalid value creates a multidimensional point that is a part of the cluster of valid points stored in H resulting in a high Mahalanobis Distanceand an alarm.

A *contextual anomaly* yields a multidimensional point where each dimension may be in the valid scope but the multidimensional point is yet a part of the cluster of points stored in H. The occurrence of the values in $\vec{i'_t}$ together was not observed in H and thus is anomalous with respect to the context provided by H. For instance, contradicting values for the once correlated *Altimeter* and the *GPS indicated altitude* attributes yields a two-dimensional point, which is a part of the cluster stored in H where all the two-dimensional points have correlated dimensions. This results in a high Mahalanobis Distanceand an alarm.

A *collective anomaly* yields a series of values for an attribute in which their occurrence together as a collection is anomalous. Given such a series of values, the latest values in the series create a multidimensional point in $\vec{i'_t}$ which is getting farther and farther away from the cluster of the stored points in H due to the correlation break of the anomalous dimension (see Fig. 3). This results in a high Mahalanobis Distanceand an alarm.

Figure 3 presents an example of the Mahalanobis Distanceas an anomaly detector. The data were taken from simulated flights of *FlightGear fight simulator* [7]. Recall the running example (Sect. 3), where the *Altimeter* was stuck on a legal value, but the *GPS Altitude* indicated that the UAV kept on rising. The values of these two attributes supposed to change together. However, when one of the attributes is stuck, even on a legal value, it is a *collective anomaly* with respect to the other attribute. These two attributes are depicted as the two dimensions in Fig. 3. The square (gray) points present an observation of the values of these attributes, taken from the sliding window, which we consider to present a nominal flight. The diamond (black) points present the values of current inputs which are injected with an anomaly. For a period of time, the *GPS Altitude* attribute's values kept on rising while the *Altimeter* attribute's values stayed the same. The Mahalanobis Distanceof several points from the anomalous flight with respect to the nominal observation is depicted in Fig. 3 as m. Before the anomaly occurs, the Mahalanobis Distanceis 0.8. During the anomaly time, the *GPS Altitude*'s values keep rising while the *Altimeter*'s values remain the same. The 2D point formed of the current values of these two attributes is being located further away from a dense area, rising the Mahalanobis Distanceup to 23.47 standard deviations. After the anomaly time, the *Altimeter*'s values are nominal again, placing the current 2D point back in a dense area, decreasing the Mahalanobis Distanceto 0.81.

Using the Mahalanobis Distanceas an anomaly detector is prone to errors without guidance. In this paper, we show that the success of Mahalanobis Distanceas an anomaly detector depends on whether the dimensions inspected are correlated or not. When the dimensions are expected to be correlated, a large Mahalanobis Distancemay indicate the result of *point*, *contextual* or *collective* anomalies. However, when the dimensions are not expected to be correlated, the large Mahalanobis Distancemay simply indicate this expected lack of correlation rather than an unexpected anomaly. An alarm in such case would be a false alarm.

Therefore, it is imperative to use a *preprocess* that determines correlated dimensions, prior to the usage of the Mahalanobis Distance. Instead of using the Mahalanobis Distanceonce on all the n dimensions, the Mahalanobis Distanceis applied several times, once per each set of k correlated dimensions where $k < n$. The selection of correlated dimensions (instead of all dimensions):

1. Reduces false positives and thus enables the use of Mahalanobis Distanceas anomaly detector.
2. Reduces run-time. It acts as a dimension reduction, which is essential for the Mahalanobis Distancecalculation to be feasible online.

4.3 The challenge of finding correlated attributes

Finding correlated attributes automatically is a difficult task. Some attributes may be statically correlated with others, but the correlation may change dynamically. For example, the *elevator* value of an aircraft's stick is correlated with the aircraft's *pitch* and with the change of height, measured in the differences of the values of the *altitude* attribute (see Fig. 4). However, this is not always true. There is another dependency on the value of the *roll* attribute, which is influenced by the *aileron* value of the aircraft's stick. As the aircraft is being rolled, the *pitch* axis becomes more vertical. This, in turn, makes the *elevator* value correlate with the *heading* value, rather than the height (see Fig. 5). In the same manner, the *rudder* is correlated with the aircraft's *yaw*, which usually affects the heading. However, as the aircraft is being rolled, the *yaw* axis becomes more horizontal. This, in turn, makes the *rudder* value correlate with the *altitude* value, rather than the heading.

Figure 6 shows a visualization of a correlation change between attributes over time. The figure shows a matrix sized 71×71, where each cell $< i, j >$ depicts the correlation strength

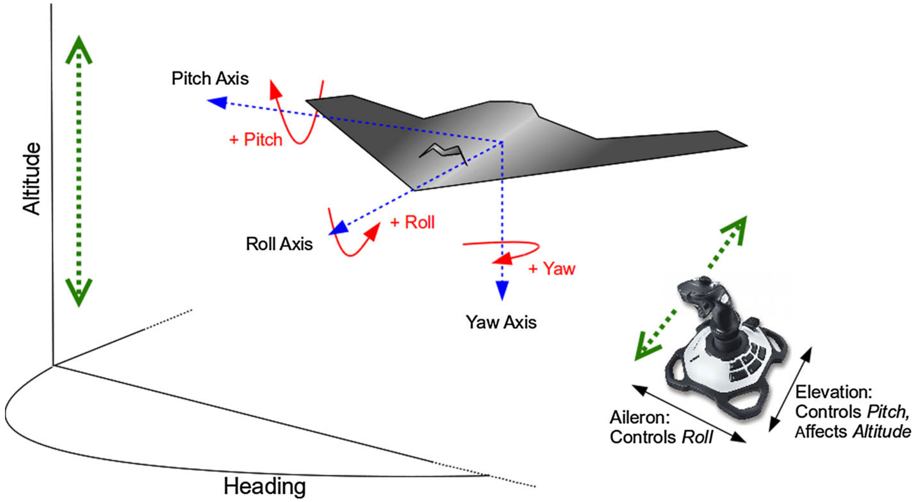


Fig. 4 The *Altitude* is affected by the *Elevator*. The *Elevator* affects the *Pitch* that affects the *Altitude* when the UAV is leveled

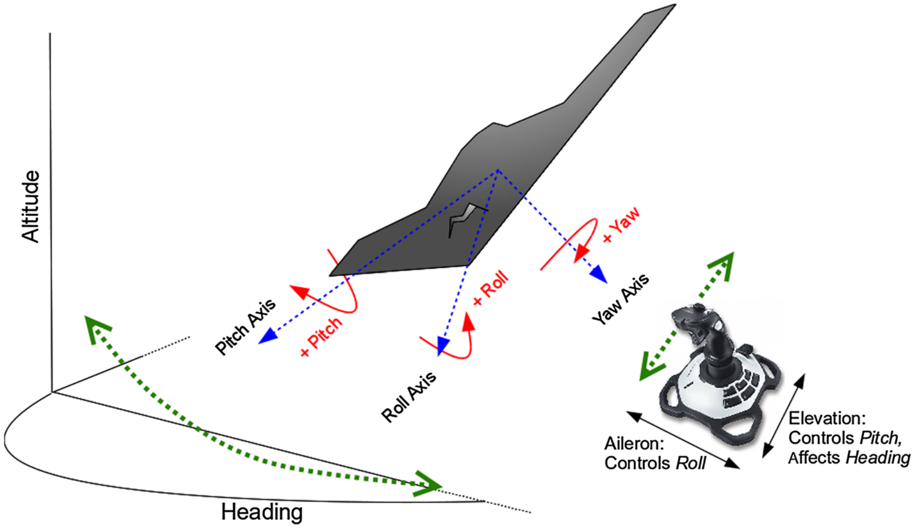


Fig. 5 The *Heading* is affected by the *Elevator*. The *Elevator* affects the *Pitch* that affects the *Heading* when the UAV is rolled

between attributes a_i, a_j. The stronger the correlation, the darker the color of the cell. Figure 6 displays three snapshots taken from different time periods of a simulated flight in FlightGear, where 71 attributes were monitored. The correlation change is apparent.

To handle the dynamic nature of correlations between attributes, we consume the online data in a sliding window fashion as described in Sect. 4.4 and apply a quick correlation detection on the consumed data as described in Sect. 4.5.

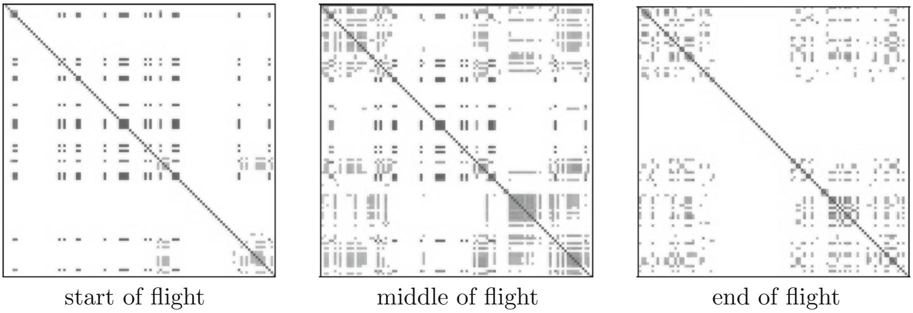
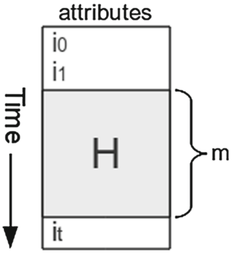


Fig. 6 Visualization of correlation change during a flight. *Darker* cells present stronger correlation between attributes

Fig. 7 An illustration of the sliding window



4.4 The sliding window

To meet the challenge of detecting dynamically correlated attributes, we utilize a *sliding window* technique to maintain H—the historical data—online. The sliding window (see Fig. 7) is a dynamic window of predefined size m which governs the size of past data taken into account. Thus, every time a new input $\vec{i_t}$ is received, H is updated as the last m online inputs: $H \leftarrow \{\vec{i}_{t-m}, \ldots, \vec{i}_{t-1}\}$. The data in H are always assumed to be nominal. Based on H, we evaluate the anomaly score for the current input $\vec{i_t}$ using the *Mahalanobis Distance*.

Considering the need to reduce run-time and false positives, there are two advantages for using a sliding window approach for the historical data rather than a complete record of the past data. First, it allows achieving reduced computation time, which makes it feasible to be used online. Second, as newer data enter the sliding window, older data are removed and ignored. Thus, only time-relevant data are maintained in H. If past data were not ignored, then more false positives would have been derived due to the changing nature of the data over time[1].

4.5 Online correlation detection

We use a fast online correlation detection (see Algorithm 1) that is based on *Pearson correlation coefficient* calculation [22].

Algorithm 1 calculates the n sets of correlated attributes, one set per each of the n attributes ($\forall a_i \in A$). Each set is contained in a tuple $< C \subseteq A, at \in \mathbb{R} >$ that includes an anomaly threshold value at for each correlated set C. The algorithm returns CS, the set of all tuples created by the algorithm.

[1] Note that even a data instance that was determined as anomalous automatically enters the sliding window in the next time step. Otherwise, H will not reflect changes over time.

Algorithm 1 Correlation_Detector(H, A, ct)

```
Require: $H$ - $m \times n$ matrix, the historical data made of the last $m$ online inputs.
Require: $A$ - the set of attributes
Require: $ct$ - a threshold value between 0 and 1
Ensure: $CS$ - a set of tuples $< C \subseteq A, at \in \mathbb{R} >$
1: $CS \leftarrow \emptyset$
2: for each $a_i \in A$ do
3:   $C \leftarrow \{a_i\}$
4:   for each $a_{j \neq i} \in A$ do
5:     if $|\rho_{i,j}(H_i^T, H_j^T)| > ct$ then
6:       $C \leftarrow C \cup \{a_j\}$
7:   $CS \leftarrow CS \cup < C, 0 >$
8: return $CS$
```

The correlation calculation is done as follows: The vectors of the last m values of each two attributes a_i, a_j are extracted from H and denoted as H_i^T,H_j^T (where T denotes transpose). We then apply the *Pearson correlation* on them denoted as $\rho_{i,j}$. If the absolute value of the result $|\rho_{i,j}|$ is larger than a correlation threshold parameter $ct \in [0, 1]$, then the two attributes are declared as correlated and a_j is added to a_i's correlated set C.

The ct parameter governs the size of the correlated attributes set. The higher the value of ct, the less the attributes are deemed correlated, thereby decreasing the dimensions and the total amount of calculations. This might also prevent attributes from being deemed correlated and affect the flagging of anomalies. On the other hand, the lower the value of ct, the more the attributes are considered correlated, thereby increasing the dimensions and also increasing the likelihood of false positives, as less correlated attributes are selected.

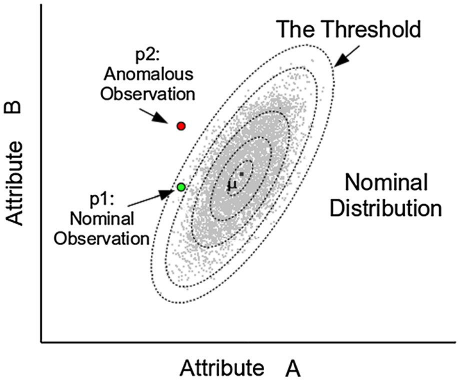
ct is determined offline by running the anomaly detection algorithm on data known to be nominal (e.g., nominal flights), where the value of ct is set, once no anomalies are returned. We use the same correlation threshold ct to construct the different sets of correlated attributes.

Each set C is associated with an anomaly threshold which is set (for now) to 0 and added to CS (line 7). The threshold per each set is later calculated as described in the next section.

4.6 Anomaly threshold calculation

Once the correlated group is established, an anomaly-threshold value per each set of correlated attributes should be determined. This is done using the Mahalanobis Distance. Figure 8

Fig. 8 Threshold setting example. The outer ring is the threshold for the distribution of gray points



demonstrates the threshold setting for the anomaly detector with respect to a set of two correlated attributes. The gray (2D) points represent the nominal observations of the values of the two attributes. Each ring around the centroid μ is a standard deviation boundary. Each point within a ring has an equal or smaller Mahalanobis Distancethan the boundary. The outer ring is the highest Mahalanobis Distanceof all the nominal points. Thus, the ring acts as a threshold. Points $p1$ and $p2$ have the same value in attribute A, but different values in attribute B. The Mahalanobis Distanceof $p1$ is lower than the threshold, and thus, it is considered as a nominal observation. The Mahalanobis Distanceof $p2$ is higher than the threshold, and thus, $p2$ will cause a declaration of an anomaly.

Algorithm 2 sets a threshold value per each set of correlated attributes. These thresholds are later used by the *Anomaly Detector* (Algorithm 3) to declare an anomaly if the anomaly score of a given input crossed a threshold value.

Algorithm 2 Threshold_Setter(H, CS)

```
Require: $H$ - $m \times n$ matrix, the historical data made of the last $m$ online inputs.
Require: $CS$ - a set of tuples $< C \subseteq A, at \in \mathbb{R} >$
Ensure: sets an anomaly-threshold value for each tuple in $CS$
for each $< C, at > \in CS$ do
  $at \leftarrow 0$
  let $H_c$ be a $m \times |C|$ matrix, containing the last $m$ values of the attributes in $C$ taken from $H$
  for each $\vec{p} \in H_c$ do
    if $at < D_{mahal}(\vec{p}, H_c)$ then
      $at \leftarrow D_{mahal}(\vec{p}, H_c)$
```

Each anomaly-threshold at in each tuple $< C, at >$ is set by the algorithm to be the highest Mahalanobis Distanceobserved. This observation is based on a distribution of m points in a $|C|$-dimensional space representing the last m values of the attributes in C. These points are held in the rows of a $m \times |C|$ matrix denoted as H_c. The values of H_c are taken from the columns of H that are corresponding to the attributes in C (e.g., for each $a_i \in C$ take H_i^T).

Each $|C|$-dimensional point $\vec{p} \in H_c$ represents a data instance that is considered to be nominal with respect to the distribution of the other points in H_c. Hence, any lower Mahalanobis Distancethan the Mahalanobis Distanceof $\vec{p}$ should not cause an alarm. Therefore, a threshold is set to be the highest Mahalanobis Distanceobserved.

4.7 Data filtering

Monitoring in robots has special characteristics. Robots monitor their environment using sensors and create a world's state. They decide how to act in a way that would indirectly affect the world's state such that some goal state will be achieved. In other words, the expected changes in the environment are a function of the actions selected by the robot.

We, therefore, propose to monitor the difference in the values measured by the sensors which originates from the robot's actions, rather than the absolute values. The raw readings of the sensors usually do not correspond directly to the agent's actions. For example, an increase of *speed* should be correlated with the loss of *height* generated by the UAV's action, rather than correlating a specific *speed* value with a specific *height* value. Formally, we use the difference between the last two samples of each attribute, denoted as $\triangle(\vec{i_t}) = \vec{i_t} - \vec{i_{t-1}}$. We also denote the history of the differential data as $\triangle(H)$.

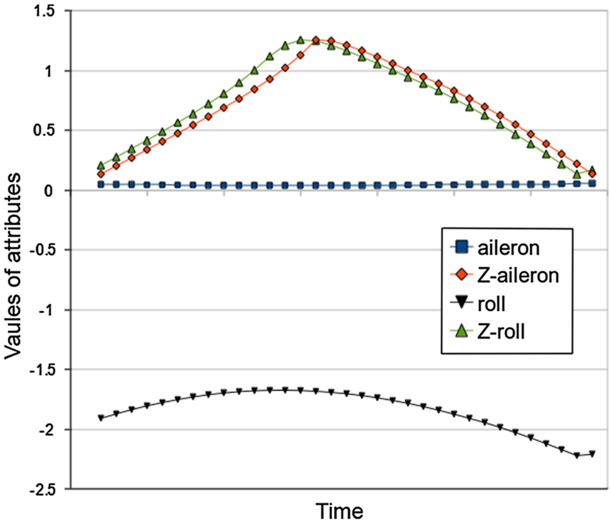


Fig. 9 Illustration of the Z-transformation. Any change is expressed as a "normalized ripple"

In the domain of robotics, the data are sampled at high frequency and might include noise. When the raw data are sampled at high frequency, the differential data contain values which are close to 0 because the raw values have not changed very much between samples. A noise, such as an abrupt high value spike, causes a dramatic change between samples, which is very anomalous with respect to the other differential data samples which are close to 0. Yet, we would not want to raise an alarm each time there is a momentary noise. Therefore, we use a filter that "slows" the changes in the differential data.

We apply a smoothing function using a z-transform. This filter measures changes in terms of standard deviations based on the sliding window and normalizes all values to use the same standard deviation units. A Z-score is calculated for a value x and a vector $\vec{x}$ using the vector's mean value $\bar{x}$ and its standard deviation σ_x, that is, $Z(x, \vec{x}) = \frac{x-\bar{x}}{\sigma_x}$.

We apply z-transform to transform each value $i_{t,j}$ to its Z-score based on the last m values extracted from the sliding window H (H_j^T). Formally, $Z_{raw}(\vec{i_t}, H) = \{Z(i_{t,1}, H_1^T), \ldots, Z(i_{t,n}, H_n^T)\}$. The Z_{raw} transformation can be also applied on the differential data $\triangle(\vec{i_t})$, $\triangle(H)$, i.e., $Z_{\triangle}(\vec{i_t}, H) = Z_{raw}(\triangle(\vec{i_t}), \triangle(H))$. If a single raw value is affected by noise, its Z-value is smoothed by the rest of the raw values.

In our simulations, we experimented with two types of filters that use the Z-transformations: Z_{raw} and $Z_{\triangle}$.

When an actuator is idle, its Z-values are all 0s, since each incoming raw value is equal to the last m raw values. However, as the actuator's reading changes, the raw values become increasingly different from one another, increasing the actuator's Z-values until the actuator is idle again (possibly on a different raw value). The last m raw values are filled again with constant values, lowering the actuator's Z-values. This way, a change is modeled by a "ripple effect," causing other attributes that correspond to the same changes to be affected by that effect too.

Figure 9 illustrates the Z-transformation technique. The data are taken from a segment of a simulated flight in the FlightGear domain. The figure presents values of two attributes (Y-axis) through time (X-axis). The *aileron* attribute stores the left and right movement of the UAV's stick. There movements control the UAV's *roll* which is sensed using gyros and stored in the *roll* attribute. We say that the *aileron* and *roll* attributes are correlated if they share the same effect of change. The *aileron*'s raw data are represented by the square points, which remain almost constant. The *roll*'s raw data, represented by an upside-down triangle, differ significantly from the *aileron*'s data. However, they share a similar ripple effect, illustrated by their Z-transformed values, shown in the right-side up triangle points and the diamond points. We can see that even the smallest change is normalized to become a "slow" ripple.

4.8 Anomaly detector engine

This section summarizes the whole process and presents our online anomaly detection engine (Algorithm 3). It includes four steps: (1) applying filters (Sect. 4.7), (2) correlation detection (Algorithm 1 in Sect. 4.5), (3) finding an anomaly threshold (Algorithm 2 in Sect. 4.6) and (4) executing the Mahalanobis Distanceoutlier detector.

Algorithm 3 Anomaly_Detector($\vec{i_t}, A, ct$)

```
Require: $\vec{i_t}$ - the current input vector received online
Require: $A$ - the set of attributes
Require: $ct$ - a threshold value between 0 and 1
Ensure: "Anomaly" if $\vec{i_t}$ is anomalous with respect to previous inputs
1: maintain $H_{raw}$ as the last $m$ inputs of $\vec{i_t}$
2: $\vec{i_{tz}} \leftarrow Z_{\triangle}(\vec{i_t}, H_{raw})$
3: maintain $H_z$ as the last $m$ filtered vectors - $\vec{i_{tz}}$
4: $CS \leftarrow$ Correlation_Detector($H_z, A, ct$)
5: Threshold_Setter($H_z, CS$)
6: for each $< C, at > \in CS$ do
7:    let $H_c$ be a $m \times |C|$ matrix, containing the last $m$ values of the attributes in $C$ taken from $H_z$
8:    let $\vec{p}$ be the values of the attributes in $C$ taken from $\vec{i_{tz}}$
9:    if $at < D_{mahal}(\vec{p}, H_c)$ then
10:       return "Anomaly"
```

The algorithm is invoked with every new consumed online input $\vec{i_t}$. The last m values of all attributes are maintained in H_{raw}—sliding window (line 1). Each input vector that is obtained online, $\vec{i_t}$, is transformed to $Z_{\triangle}(\vec{i_t}, H_{raw})$ (line 2). The sliding window of the filtered values H_z is updated as well (line 3). The *online preprocess* takes place in lines 4–5. The set of all tuples CS is returned by the $Correlation\_Detector$ algorithm (Algorithm 1); one tuple per each attribute $a \in A$, containing a's set of correlated attributes C and the anomaly-threshold at for that set (line 4). The anomaly thresholds are calculated in line 5 by the $Threshold\_Setter$ algorithm (Algorithm 2). For each tuple $< C, at >$, we extract from H_z the columns that correspond to the attributes in C and store them in a $m \times |C|$ matrix denoted as H_c (line 7). H_c represents the filtered data of a correlated set of attributes that is considered to be nominal. We extract the current filtered values of the same attributes from $\vec{i_{tz}}$ and store them in $\vec{p}$ (line 8). If the Mahalanobis Distanceof $\vec{p}$ and H_c is greater than the anomaly-threshold at, then this means that $\vec{p}$ represents a data instance that is anomalous with respect to the nominal data H_c. Therefore, anomaly is declared (lines 9–10).

5 Evaluation

In this section, we describe the evaluation of *ODDAD*. We start by describing the different domains and anomaly detection scenarios (Sect. 5.1). We continue with a description of the

Fig. 10 An RV-400 robot

evaluation criteria (Sect. 5.2). Next, we evaluate the impact of each component of *ODDAD* and show their significance (Sect. 5.3). Finally, we compare *ODDAD* to other competing anomaly detection approaches and show that *ODDAD* is more accurate (Sect. 5.4).

5.1 Domains and scenarios

We use four different domains to evaluate our approach: two different unmanned vehicles, a flight simulator and a physical electric power supplier of a spacecraft. **The first** set of data came from a commercial unmanned aerial vehicle. The UAV is equipped with 55 sensors and actuators, as well as a communication system. The communication system transmits the information, along with monitoring information, to the ground station.

The different attributes can be categorized to the following data groups: *air data* include telemetry data that the UAV measures, *inertial data* include information about the inertial navigation system (INS), *engine data* include information about the engine's air and water temperature, *servo* information, and *other* information, including the UAV mass, the air temperature and other information. The data are measured by the sensors at either 1 Hz or 10 Hz frequencies, yet the whole data set is downloaded from the UAV at a frequency of 10 Hz.

For the UAV, the following errors were recorded:

- *Descend*: In this recording, one of the sensors is malfunctioning and thus causes the sensor reading to decrease rapidly from a valid input to a constant value of zero. This is a *point anomaly* as described in Sect. 4.2.
- *Constant*: In this recording, one of the sensors is malfunctioning and reports a constant value for a given period. This is a *contextual anomaly* as described in Sect. 4.2.

The second set of experiments was conducted on a commercial vacuum-cleaning mobile robot (the Friendly Robotics RV-400, Fig. 10), used in our laboratory and fitted with our own control software.

The RV-400 robot (hereinafter UGV) is equipped with fewer sensors and actuators than the UAV. It has 22 attributes measured, including: ten sonar sensors which measure range, four bumper sensors and various other measurements including the target velocity and the actual velocity (based on wheel motor encoder data), etc. The data itself are recorded at 10 Hz frequency.

For the UGV, the following errors were recorded:

- *Weight Drag Halt*: In this recording, the robot was attached to a cart with a string which was loose. Then, the robot started its movement away from the cart, causing the string to

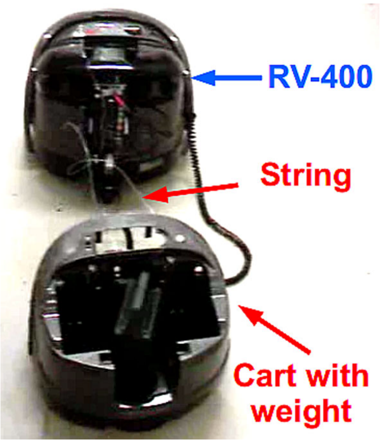


Fig. 11 RV-400 tangled with a string connected to a heavy cart

Table 1 Description of experimental data in the UAV and UGV

Data Type	Description
Nominal UAV A	Nominal flight behavior
Nominal UAV B	Nominal flight behavior
Descend UAV C	An error in a sensor, which rapidly decreases its value until a constant zero. The error is between seconds 1,599 and 1,605
Constant UAV D	An error in a sensor, which value is stuck constant. The error is between seconds 810 and 820
Nominal UGV A	Nominal driving behavior
Weight drag halt UGV	An error in the nominal driving behavior: The UGV attempts to drag a heavy load, which causes to comes to a complete halt at 10 s
Direction deviation UGV	An error in the nominal driving behavior: the UGV has an object stuck in one of its wheels, causing it to bounce every 5 s

stretch, until it was completely stretched. This caused the robot to completely halt (see Fig. 11). This scenario also presents the challenge of having little data (only 96 s of data).

- *Direction Deviation*: In this recording, a coin was attached to one of the robot's wheels. This caused the robot to divert from nominal behavior every time the coin touched the floor (which was about every 5 s). It also changed its heading.

The experimental data sets of the UAV and UGV are summarized in Table 1.

To further test our approach, on more types of faults and on various conditions, we use a **third domain**, the *FlightGear* flight simulator (see Fig. 12). *FlightGear* models real-world behavior and provides realistic noisy data. "Instruments that lag in real life, lag correctly in *FlightGear*, gyro drift is modeled correctly, the magnetic compass is subject to aircraft body forces"[7]. Furthermore, *FlightGear* also accurately models many instrument and system faults that can be injected into a flight. For example, "if the vacuum system fails, the HSI gyros spin down slowly with a corresponding degradation in response as well as a slowly increasing bias/error" [7].

In the *FlightGear* simulation, we programmed an autonomous UAV based on the simulated platform of the Cessna 172p aircraft. We planed the following behaviors: a take-off, an altitude maintenance, a turn and eventually a landing. During a flight, 4–6 faults were injected into three different components: the *airspeed-indicator, altimeter* and the *magnetic compass*. The

File View Location Autopilot Environment Equipment ATC/AI Network Debug Help Cessna C172P

Fig. 12 FlightGear flight simulator

faults and their time of injection were both randomly selected. Each fault could be a contextual anomaly [2] with respect to the UAV's behavior, and a collective anomaly [2] with respect to the measurements of different instruments such as the *GPS airspeed, altitude indicators* and the *Horizontal Situation Indicator*.

The measured attributes of a robotic system are typically low grained, in the sense that a single attribute can express the state of multiple components of a subsystem. For example, the values of the attribute *power supply* are affected by the work of the complex components of the electrical power system. Each of these components might be faulty. To test our approach on a complex subsystem which is more fine grained, and where all the components affect each other, we use an additional domain. **The fourth** domain is an electrical power system (EPS), which simulates the functioning of a typical aerospace vehicle power system (see Fig. 13). The data set was generated from an EPS in the Advanced Diagnostics and Prognostics Testbed (ADAPT) laboratory at the NASA Ames Research Center [13]. Eighty-one attributes are monitored in 2 Hz; they store data from sensors that measure system variables such as voltages, currents, temperatures and switch positions. Faults were injected into the EPS using physical or software means. Some components were stuck on legal values or drifted, switches failed to open or close [13].

The lines in Table 2 summarize the characteristics of each domain (by order): the type of data used, the type of anomalies (i.e., real or simulated), the number of faults and nominal scenarios, the duration of each scenario in seconds, the number of attributes sampled, the frequency the data were sampled, the number of anomalies per scenario and the duration of each anomaly in seconds. In total, the experimental setup covers a comprehensive and varied range of domains and characteristics of both simulated and real-world data and anomalies.

5.2 Evaluation criteria

We evaluate different anomaly detectors by the detection rate and false alarm rate. To this aim, we define four counters, which are updated for every input $\vec{i_t}$. A "True Positive" (TP) refers to the flagging of an anomalous input as anomalous. A "False Negative" (FN) refers to the flagging of an anomalous input as nominal. A "False Positive" (FP) refers to the flagging of a nominal input as anomalous. A "True Negative" (TN) refers to the flagging of a nominal input as nominal. Table 3 summarizes how these counters are updated.

For each evaluated approach, we calculate the detection rate = $\frac{tp}{tp+fn}$ and the false alarm rate = $\frac{fp}{fp+tn}$. An efficient anomaly detector should maximize the detection rate and minimize

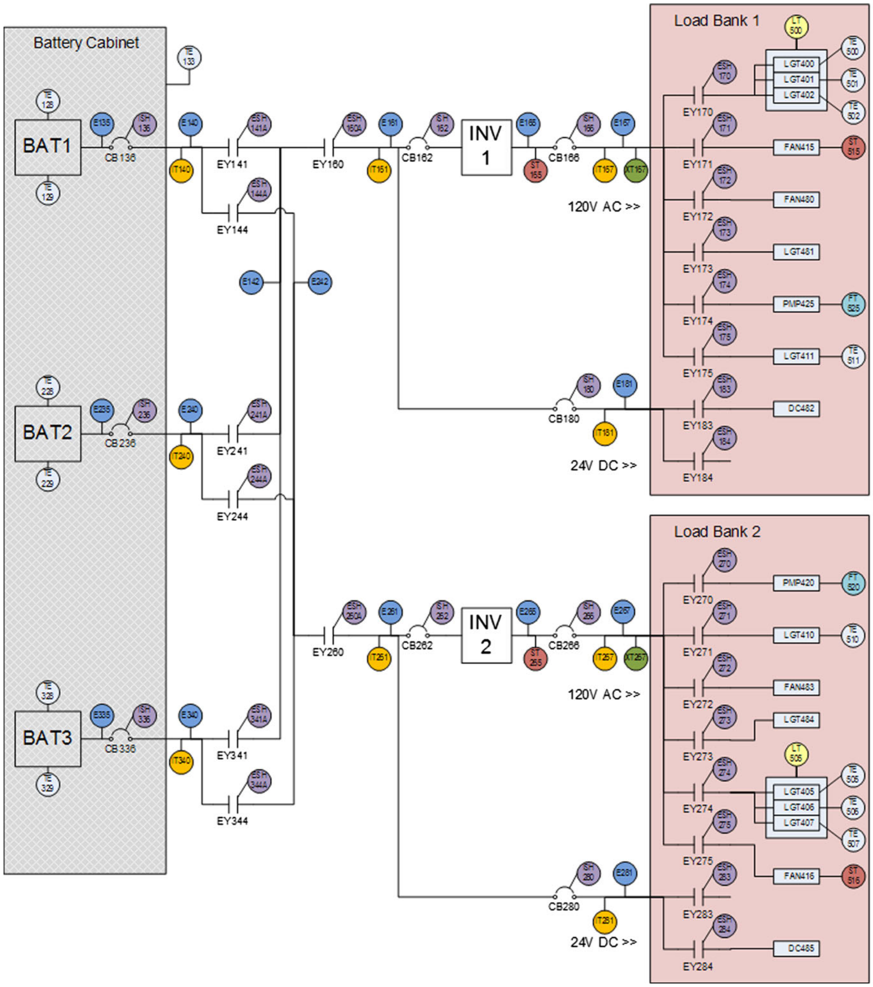


Fig. 13 The electrical power system diagram. The system contains batteries (on the *left* side), circuit breakers, invertors, capacitors and other electrical components. The circles in the different colors are sensors for current, voltage and temperature. In the *right*, there are two instruments that act as load banks

Table 2 Tested domains and their characteristics

Domain	UAV	UGV	FlightGear	EPS
Data	Real	Real	Simulated	Real
Anomalies	Simulated	Real	Simulated	Real + simulated
Fault scenarios	2	2	15	16
Nominal scenarios	2	1	1	1
Scenario duration (s)	2,100	96	660	120 to 300
Attributes	55	25	23	81
Frequency	4 Hz	10 Hz	4 Hz	2 Hz
Anomalies per scenario	1	1	4 to 6	1 to 3
Anomaly duration (s)	100, 64	30	35	Until the end of the input

Table 3 Scoring an anomaly detector

Score	Description
TP	Counts 1 if at least one "anomalous" flagging occurred during a fault time
FN	Counts 1 if no "anomalous" flagging occurred during a fault time
FP	Counts every "anomalous" flagging during nominal time
TN	Counts every "nominal" flagging during nominal time

the false alarm rate. The perfect anomaly detector has a detection rate of 1 and a false alarm rate of 0.

Some competing anomaly detection approaches only return an anomaly score and have no policy for determining whether or not the returned score is high enough (i.e., above a threshold) to declare an anomaly. Yet, we wish to evaluate these approaches with terms of false alarm and detection rates. Therefore, we use an optimization algorithm (hereinafter *OPT*) as follows: After an anomaly detector returns, the anomaly scores for the entire input, *OPT* retrospectively calculates the highest threshold possible such that all anomalies would have been detected by the anomaly detector. Thus, *OPT* selects the theoretical best threshold that minimizes the false alarm rate given a detection rate of 1. In other words, *OPT* applied on an anomaly detector represents the theoretical best anomaly detector. With *OPT* applied to *ODDAD* and to other competing approaches which only return anomaly scores, we can compare the approaches in an unbiased way. To be successful, an anomaly detector must return a high anomaly score for true anomalies and a low score for none anomalies, i.e., to be very distinctive. An optimized anomaly detector which is originally distinctive yields a low rate of false positives. Thus, the optimized anomaly detector with the lowest false alarm rate is the winner.

Figure 14 illustrates an example of the *OPT* threshold selection. After an anomaly detector has returned the anomaly scores for the entire input, and an oracle reported that the anomalies were injected in time steps 6, 8 and 15 that are marked in red in the figure, *OPT* selects the highest threshold such that all anomalies would have been detected by the anomaly detector. *OPT* selects the threshold to be just below the anomaly score of time step 6. Thus, all anomalies can be detected, and now we can count the number of false positives. Since the highest threshold was chosen, *OPT* did its best to minimize the false alarm rate given the requirement of a detection rate of 1.

5.3 The impact of the components of *ODDAD*

ODDAD is based on three key features: first, a comparison to a *sliding window*, rather than a complete record of past data. Second, the use of an *online preprocess* to find pairs of correlated attributes instead of offline full n-attribute dependencies. Third, the use of *differential filtered data*. To demonstrate the independent contribution of each feature, we test each feature in the *FlightGear* domain. We chose the *FlightGear* domain since it is the richest domain (see Table 2). We begin by testing the following online anomaly detectors that are described by three parameters (*Nominal Data, Preprocess and Filter*), as summarized in Table 4. The bold line is our recommended *ODDAD* approach when using Z_Δ as the $filter$.

The $filter$ can be *raw*, Δ, *Zraw*, Z_Δ as described in Sect. 4.7. CD denotes the use of a complete record of past data. SW denotes the use of a Sliding Window. For instance, *(SW,Tcd,Zraw)* uses data that were filtered using the Z-transformation on the raw val-

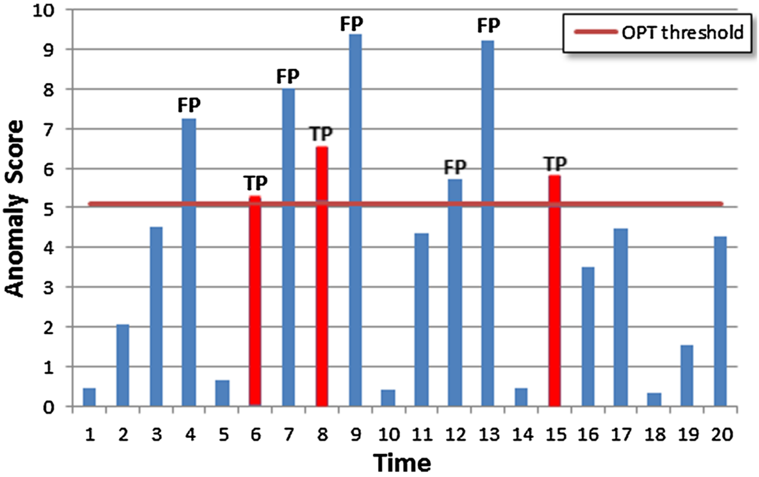


Fig. 14 OPT threshold selection example. The *Red bars* indicate anomalies. We set the highest threshold such that every anomaly will be detected (TP). Then, we can count the number of false alarms, marked as "FP" for false positives

Table 4 Tested anomaly detectors

Name	Nominal Data	Correlation detector	Threshold setter	Online detection
(CD, none, $filter$)	Complete past data	Not applied	Applied offline	On one set of attributes
(SW, none, $filter$)	Sliding window	Not applied	Applied online	On one set of attributes
(CD, Tcd, $filter$)	Complete past data	Applied offline	Applied offline	on n sets of correlated attributes
(SW, Tcd, $filter$)	Sliding window	Applied offline	Applied online	On n sets of correlated attributes
(SW, Tsw, $filter$)	Sliding window	Applied online	Applied online	on n Sets of correlated attributes

ues (Zraw), to compare the input to the data of a sliding window (SW), applied on sets of correlated attributes that were determined offline (Tcd). Following the same principle, $((SW, Tsw, Z_\Delta))$ uses data that were filtered using the Z-transformation on the differential values (Z_Δ) to compare the input to the data of a sliding window (SW), applied on sets of correlated attributes that were determined online—on the data of the sliding window (Tsw).

Each of these algorithms detects anomalies online. However, there is a difference when and if the correlation detector is applied and when the threshold setter is applied. Whenever the correlation detector is applied, the anomaly detector is applied on n sets of correlated attributes, otherwise only one set containing all attributes is used. When the correlation detector is applied online (on the sliding window's data), the correlations are built and destroyed dynamically, otherwise they are fixed. The threshold setter is always applied on the nominal data; if it is the sliding window's data, then thresholds are found online, otherwise offline.

$(SW, Tsw, filter)$ is our proposed *ODDAD* approach when the filter is Z_Δ (see Algorithm 3 in Sect. 4.8). $(CD, Tsw, filter)$ is *not* displayed in Table 4. This anomaly detector executes the preprocess on the sliding window, and thus, the thresholds are calculated online each time that different correlated sets are returned. However, the comparison of the online input is made against a complete record of past data, and thus, thresholds are calculated on

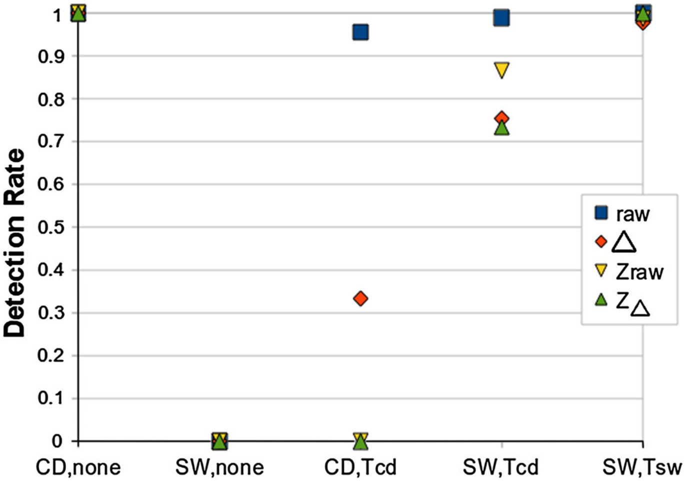


Fig. 15 FlightGear: detection rate. Higher is better

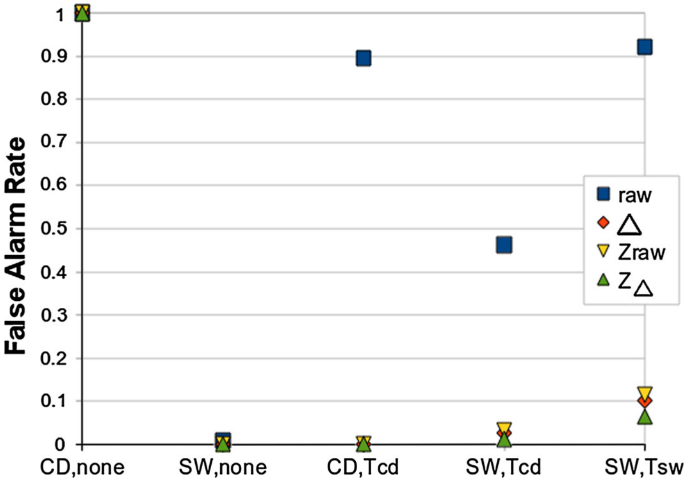


Fig. 16 FlightGear: false alarm Rate. Lower is better

the data of CD, which is considerably larger than the data of SW. Therefore, the anomaly detection of $(CD, Tsw, filter)$ is not feasible online.

In the first test suite, we present the influence of the different filters on our algorithm. Figures 15 and 16 present the average detection rate and false alarm rate, respectively, of 15 flights in the *FlightGear* simulator. The scale ranges from 0 to 1, where 0 is the best possible score for a false alarm rate and 1 is the best possible score for a detection rate.

We begin with the first anomaly detector, *(CD,none)*. Both Figs. 15 and 16 show a value of 1, indicating a constant declaration of an anomaly. In this case, no improvement is achieved by any of the filters. This accounts for the fact that the comparison is made to a complete record of **past** data. Since once a new point is sampled from a *different* flight, it is very unlikely for it be observed in the past data, resulting in a higher Mahalanobis Distancethan the threshold, and the declaration of an anomaly.

Table 5 Feature contributions

Feature	Contribution	Reason
Sliding window	Decreases FP	Similarity of $\vec{i_t}$ to H
Preprocessing	Increases TP	Correlated dimensions → more conspicuous anomalies
Online preprocessing	Increases TP	Correspondence to dynamic correlation changes
Filters	Decreases FP Increases TP	Better correlations are found

The next anomaly detector we examine is *(SW,none)*. This detector decisions are based on a sliding window. Since data are collected at a high frequency, the values of $\vec{i_t}$ and the values of each vector in H are very similar. Therefore, the Mahalanobis Distanceof $\vec{i_t}$ is not very different from the Mahalanobis Distanceof any vector in H. Thus, the threshold is very rarely crossed. This explains the very low false alarm rate for this algorithm shown in Fig. 16. However, the threshold is not crossed even when anomalies occur, resulting in a very low detection rate as Fig. 15 shows. The reason is the absence of a correlation detection preprocess. The Mahalanobis Distanceof a contextual or collective anomaly is not higher than Mahalanobis Distances of points with uncorrelated dimensions in H.

The next two anomaly detectors introduce the use of offline preprocessing. The first *(CD,Tcd)* uses a complete record of past data, while the second *(SW,Tcd)* uses a sliding window. In both anomaly detectors, the preprocessing is done offline on a complete record of past data. Yet, *(CD,Tcd)* compares the input to the past offline record, while *(SW,Tcd)* compares the input to the data of the sliding window. When no filter is used, *(CD,Tcd)* declares an anomaly most of the times. This is illustrated in the square dots in Figs. 15 and 16. When filters are used, more false negatives occur, expressed in the extremely low false alarm rates and the decreased detection rate. However, when a sliding window is used, even with no filters, *(SW,Tcd)* gets better results: a detection rate of 1, and less than 0.5 false alarm rate, which is lower than *(CD,Tcd)*'s false alarm rate. The filters used with *(SW,Tcd)* decrease the false alarm rate to almost 0, but the detection rate, though decreased, remains high. Comparing *(SW,Tcd)* to *(CD,Tcd)* shows the importance of a sliding window, while comparing *(SW,Tcd)* to *(SW,none)* shows the crucial need of preprocessing.

The final anomaly detector is *(SW,Tsw)* which differs from *(SW,Tcd)* by the preprocessing mechanism. *(SW,Tsw)* applies an online preprocess on the sliding window. This allows achieving a very high detection rate. Each filter used allows increasing of the detection rate closer to 1, until Z_Δ gets the score of 1. The false alarm rate is very high when no filter is used. When using filters, we are able to reduce the false alarm rate closely to 0. (SW, Tsw, Z_Δ) achieves a detection rate of 1, and a low false alarm rate of 0.064.

These results show the main impact of each feature variant, summarized in Table 5. The sliding window causes the decrease of false positives since H and $\vec{i_t}$ are kept similar when H is updated. The correlation detection preprocessing phase increases the true positives since when correlated dimensions are used, outliers present a break in the expected correlation and hence are found to be anomalous; the anomalies are more conspicuous. The online preprocessing further increases the true positives since the challenge of finding dynamically correlated attributes is met. The use of filters increases the true positives and decreases the false positives since differential data can now be used, free of noise, and thus, better correlations are found.

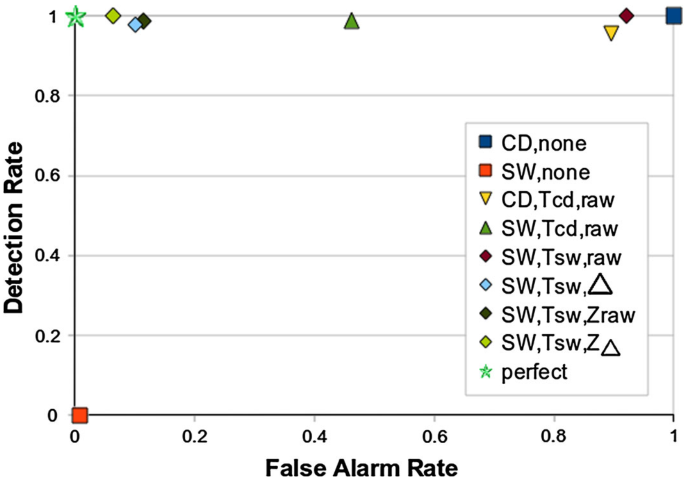


Fig. 17 The Anomaly Detection ROC chart. The configuration (SW,Tsw,Z_Δ) is the closest to the theoretical perfect score

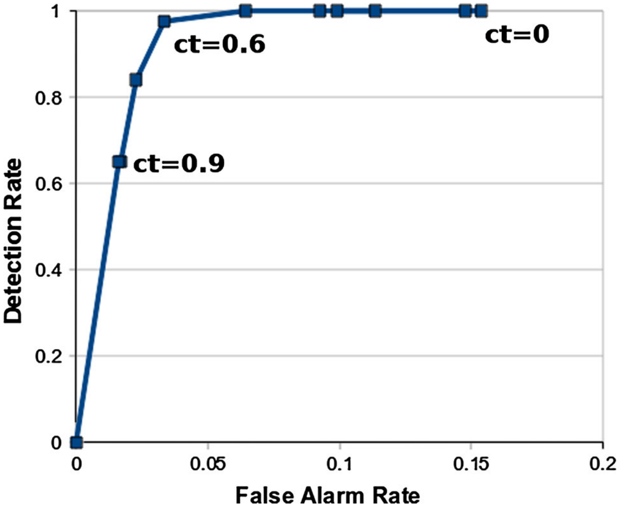


Fig. 18 The influence of the correlation threshold

Figure 17 demonstrates the receiver operating characteristic (*ROC*) chart describing the entire space of our anomaly detectors. The X-axis is the false alarm rate, and the Y-axis is the detection rate. An anomaly detector is expressed as a $2D$ point. The perfect anomaly detector is located at point (0,1), representing a point that has no false positives and detects all the anomalies. Figure 17 illustrates that when the features of our approach are applied, they allow the results to approximate the perfect anomaly detector.

Figure 18 focuses on the best anomaly detector in the *ROC* space—(SW, Tsw, Z_Δ) (*ODDAD*), the one with the highest detection rate and lowest false alarm rate. Note that the X-axis scales differently than in Fig. 17, it ranges between [0, 0.2] in order to zoom in on the effect. To evaluate the impact of the correlation threshold $ct \in \{0..1\}$ (described in Sect. 4.5), we test a variation of values for ct.

When ct equals 0, all the attributes are selected for each correlated set, resulting in false alarms. As ct increases, fewer uncorrelated attributes are selected, reducing the false alarms, until a peak is reached. The average peak of the 15 *FlightGear*'s flights was reached when ct equals 0.5. *ODDAD* averaged a detection rate of 1 and a false alarm rate of 0.064. As ct increases above that peak, fewer attributes that are crucial for the detection of an anomaly are selected, thereby increasing the false negatives, which in turn lowers the detection rate. When ct reaches 1, no attributes are selected, resulting in a constant false negative.

5.4 Comparing *ODDAD* against competing methods

In this section, we evaluate *ODDAD*s accuracy against other anomaly detection approaches in all the domains. We first compare it against the support vector machine (*SVM*) algorithm [29] since it is considered to be a very successful classifier. Next, we compare *ODDAD* to the anomaly detector presented by Lin et al. [16] because of its use of Mahalanobis Distanceand offline dependency detection. Finally, we compare *ODDAD* against the Incremental Local Outlier algorithm (incremental *LOF*) [23] because of its similarity to *ODDAD* by being online and density based.

Support vector machines are considered very successful classifiers when examples of all categories are provided [29]. We ran the *SVM* algorithm [31] on the data from *FlightGear* domain. We generated a training set of samples from both categories, i.e., normal and anomalous. All of the 23 monitored attributes were used as features since no manual feature selection was made for any of the competing algorithms. We tested linear, polynomial and radial kernels for the SVM, and yet, to our surprise, the SVM did not detect any anomaly, i.e., it produced a detection rate of 0.

The ODDAD and LOF algorithms on the other hand had a detection rate of 1. Apart of their automatic feature selection, i.e., correlation or neighborhood based, we believe that the main reason is that these algorithms compare recent online inputs while the SVM is based on an offline training set. The tested anomalies can only be regarded as anomalies with respect to the dynamic context of the current stage of the active flight. ODDAD and LOF create a boundary around the current context and thus are able to accurately detect these anomalies. On the other hand, an offline training data set may include a lot of these contexts, which makes the SVM to set high boundaries which do not allow it to detect anomalies online. This shows how elusive these contextual anomalies are. Even under unrealistically favorable conditions, where both nominal and anomalous data samples are available for the training, the SVM is unable to detect anomalies.

Hence, we continue evaluating with other competing approaches that are able to detect anomalies. The previous approach (hereinafter *(CD,MSDD,none)*) presented in [16] uses an offline dependency detection preprocess that selects dependent attributes. Each online consumed input is compared to a past record of nominal data by using the Mahalanobis Distancecalculation on sets of dependent attributes. The *(CD,MSDD,none)* approach is similar to *(CD,Tcd,none)* described in Sect. 5.3. However, the dependency detection is different from the correlation detection procedure of *ODDAD*. While *ODDAD* uses the *Pearson Correlation* calculation on pairs of attributes, the competing *(CD,MSDD,none)* approach uses the *MSDD* algorithm [21] to detect dependencies between values of groups of attributes. For example, the *MSDD* can find that at a given time during a flight, the values of attitude-related attributes (e.g., pitch, yaw, roll, etc.) are dependent on the previous values of control-related

Table 6 Optimized results *(CD,MSDD,none)* versus *ODDAD*

Domain	*OPT(CD,MSDD,none)*		*OPT(ODDAD)*	
	Detection rate	False alarm rate	Detection rate	False alarm rate
UAV	1	0.044	1	0.0014
UGV	1	0	1	0.005
FlightGear	1	1	1	0.0013
EPS	1	1	1	0.001

attributes (e.g., elevator, rudder, aileron, etc.). The level of dependencies found by the *MSDD* is governed by the search depth, which needs to be deep enough to find good dependencies.

The main drawback of the *(CD,MSDD,none)* approach is the fact that the *MSDD* is heavy on system resources (CPU and Memory) and impractical to be run online. In theory, the *MSDD* is a stronger tool than the *Pearson Correlation* in finding attributes' dependencies. However, since it must be run offline, it does not address the dynamic nature of correlation between attributes, which has a great impact on the anomaly detection accuracy.

(CD,MSDD,none) returns an anomaly score and has no policy on how to determine a threshold above which an anomaly is declared. Therefore, we compare *OPT(CD,MSDD,none)* with *OPT(ODDAD)* and present the best theoretical results for each approach. Table 6 summarizes these results on the four domains described in Sect. 5.1. The detection rate of both algorithms is 1. The false alarm rate measure is more challenging. Except for the UGV domain, where there is a slight difference in the false alarm rates, in all other domains, *ODDAD* outperforms *(CD,MSDD,none)*. In the FlightGear and EPS domains, the *(CD,MSDD,none)* approach failed. Every input was declared as anomalous. This shows the problem of not selecting attributes while they are dynamically correlated but rather rely on previously recorded data.

The two approaches have advantages and disadvantages that affect the results. *(CD,MSDD, none)* finds dependencies between n-dimension attributes while *ODDAD* finds correlations between pairs. On the other hand, this enables *ODDAD* to apply the correlation detection process online and thus address the dynamic nature of correlation between attributes while *(CD,MSDD,none)* cannot. Moreover, *ODDAD* compares the online input to recent data (sliding window) while the *(CD,MSDD,none)* compares the online input to past recording of data. The comparison to recent data makes any anomaly to become more obvious, and thus, it gives an advantage for *ODDAD*. Finally, *ODDAD* uses filters that reduce noise and therefore help in reducing false positives.

We continue with a run-time comparison. While the *ODDAD* approach has no offline phase, the *(CD,MSDD,none)* approach's offline run-time is significantly long. It is determined by the search depth of the MSDD algorithm, which needs to be deep enough to return good results. The offline run-time can take *hours* and even *days* depending on the search depth, the number of attributes and the data size. The run-time of the online preprocess of the *ODDAD* approach scales in the number of attributes being measured; it is usually a matter of a *few milliseconds*—faster than the frequency of the input. If the number of attributes is too large, and causes the run-time of the online preprocess to be slower than the frequency of the input, then the data can be sampled in a lower frequency or the size of the sliding window can be reduced.

We further evaluate *ODDAD* in the context of an online and a density-based anomaly detector. We compared it to the *incremental Local Outlier Factor algorithm* [23]. As in our

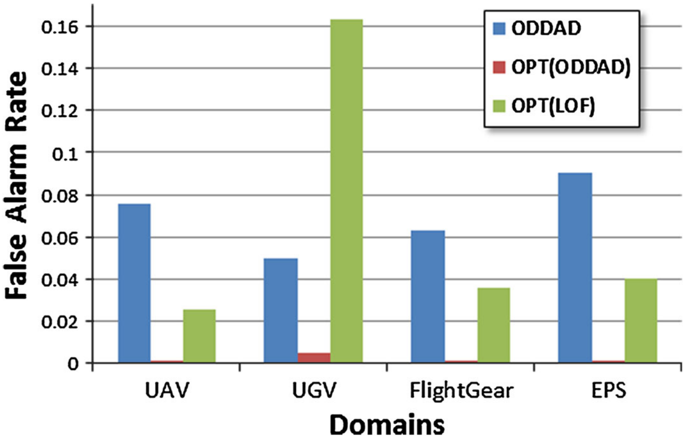


Fig. 19 False alarm rates of *ODDAD* versus LOF in different domains

approach, the *incremental LOF* returns a density-based anomaly score in an online fashion. The *incremental LOF* uses *K*-Nearest Neighbors technique to compare the density of the input's "neighborhood" against the average density of the nominal observations [23]. A detection rate of 1 was the result for both *ODDAD* and the *incremental LOF algorithm*, making the *incremental LOF algorithm* a better competitive approach to ours than the *SVM*.

Since *ODDAD* has a detection rate of 1 in every domain we tested, and since the *incremental LOF* returns an anomaly score rather than an anomaly label, we again compare the two approaches using the *OPT* comparison. Figure 19 shows, for every domain tested, the false alarm rate of the following:

1. *ODDAD*
2. *OPT(ODDAD)*—theoretical best for *ODDAD*.
3. *OPT(LOF)*—theoretical best for the incremental LOF approach.

The comparison between *ODDAD* and *OPT(LOF) does not* indicate which approach is better in anomaly detection. The comparison between *OPT(ODDAD)* and *ODDAD* indicates how much better *ODDAD* can theoretically get. The comparison between *OPT(ODDAD)* and *OPT(LOF)* does indicate which approach is better, since both are optimized.

In all the domains, *OPT(ODDAD)* achieves the lowest false alarm rate. Naturally, *OPT(ODDAD)* has a lower false alarm rate than *ODDAD*. But more significantly, it presents a lower false alarm rate than *OPT(LOF)*, making our approach a better anomaly detector than the *incremental LOF algorithm*.

In the UGV domain, there is a surprising result. *ODDAD*, which is not optimized, had a lower false alarm rate than *incremental LOF*, although the latter is optimized. This is explained by the fact that in the UGV domain, there are very little data, and therefore, each datum has only a few neighbors. Recall that *incremental LOF* utilizes a *K*-Nearest Neighbors approach which usually fails when nominal or anomalous instances do not have enough close neighbors [2]. On the other hand, the Mahalanobis Distanceuses all the points in the distribution, which is enough data to properly detect the anomalies.

The *ODDAD* approach uses a sliding window for comparison against the online input. The size of the sliding window affects the false alarm rate. The anomaly threshold chosen by *ODDAD* is set to be the highest Mahalanobis Distancewhen applied to all points in the sliding window. Thus, the bigger the window size, the more the points there are, and the

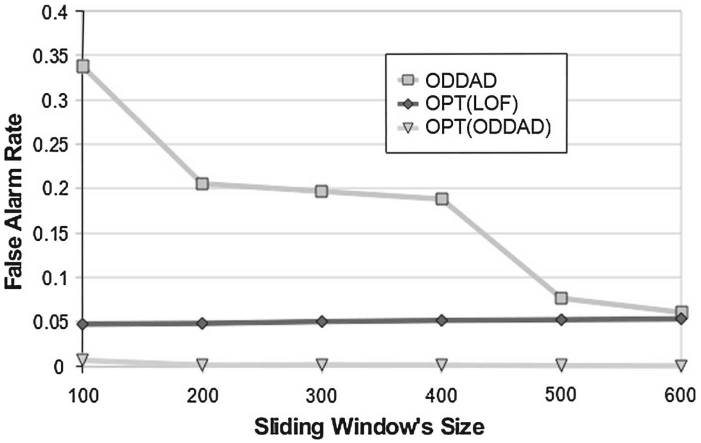


Fig. 20 The effect of the sliding window's size. As the size gets bigger, the ODDAD gets fewer false alarms. Not optimized ODDAD can match the low false alarm rate of the optimized LOF. This demonstration was made in the *FlightGear* domain

higher the anomaly threshold is likely to be. As the anomaly threshold gets higher, fewer false alarms are returned by *ODDAD*. Figure 20 depicts *ODDAD*'s reduction of the false alarm rate as the sliding window size gets bigger. An unlimited sliding window size would contain all data points from the beginning of the robot's (or system's) operation, and the anomaly threshold would have been so high such that no input would have been detected as an anomaly: a detection rate of 0 as well as a false alarm rate of 0.

In addition, Fig. 20 depicts the effect of the sliding window size on the *LOF* approach. While Mahalanobis Distanceuses the distribution of all the points in the sliding window, *incremental LOF* uses only a neighborhood within the window, thus unaffected by its size. Note that there is a window size where the false alarm rates of *ODDAD* and *OPT(LOF)* converge, even though the LOF is optimized.

Figure 20 also depicts the effect of sliding window size on the *OPT(ODDAD)*. There is very little effect since the anomaly-threshold choice is done independently of the sliding window—offline, by an oracle selection of the highest threshold possible such that all anomalies would have been detected. However, a small window size has some effect on *OPT(ODDAD)* since the size does affect the correlation detection. The smaller the size is, the more likely it is to choose uncorrelated attributes.

Finally, to evaluate the influence of the filters used in the second approach on another density-based technique, we also implemented the *incremental LOF algorithm* with those filters. Figure 21 shows the decrease of the false alarm rate of *OPT(LOF)*, when the filters are used, averaged over 15 flights of the *FlightGear*'s domain. While the raw data produce a false alarm rate of 0.037, *diff* produces only a half, and *Zraw* and *Zdiff* produce a rate of 0.027. The last results are very close to the optimal results of *ODDAD* which had only 3 false positives. This shows the benefit of a differential filtered data.

6 Conclusions and future work

In this paper, we presented a novel approach for detecting anomalies in autonomous robots. The approach uses the Mahalanobis Distanceto detect anomalies and thus benefits from its

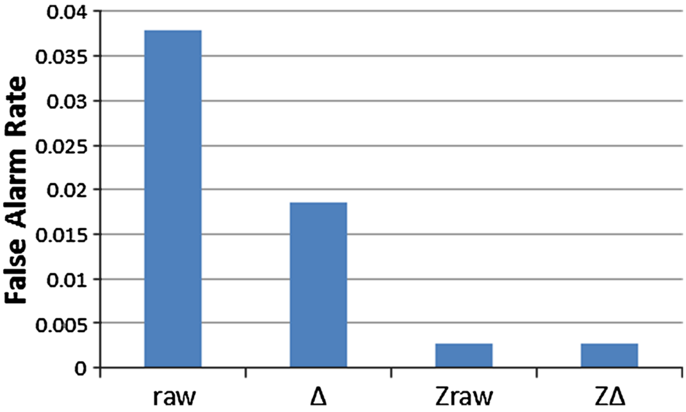


Fig. 21 False alarm rate of *OPT*(*LOF*) when our filter are applied

model-free nature. Experiments in both simulated and physical domains show *ODDAD*'s domain independent quality and its ability to accurately detect anomalies in the real world. *ODDAD* uses an online preprocess that makes the approach work completely as an online anomaly detector for robots. Therefore, the *ODDAD* approach has the qualities of a "plug and play" mechanism for different robotic platforms. We showed how essential a preprocess is to the success of the online anomaly detector. Moreover, the experiments also show the benefits of:

– The comparison of the current data input to the data of the sliding window.
– Finding dynamic correlations between attributes (online).
– Filtering the data.

We showed that this approach is superior to the incremental *LOF* algorithm and that the approach succeeds where other well-known classifiers, such as *SVM*, have failed even under unrealistic favorable conditions. We compared *ODDAD* with our previous approach that used the *MSDD* algorithm offline to detect dependencies between attributes and found that *ODDAD* is more effective. Finally, we showed that filtering can improve other anomaly detection techniques, thereby showing its independent contribution.

In the future, we plan to add a diagnostic process once a fault has detected. A diagnosis will try to isolate the root causes of the fault. In addition, the *ODDAD* approach works well where there are attributes of redundant systems. There are systems where the number of redundant sensors is very low. This arises a new challenge to our algorithm.

Acknowledgments This research was funded in part by ISF Grant #1511/12 and by Kamin program. As always, thanks to K. Ushi and K. Ravit.

References

1. Brotherton T, Mackey R (2001) Anomaly detector fusion processing for advanced military aircraft. In: IEEE proceedings of aerospace conference, vol 6, IEEE, pp 3125–3137
2. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey, association for computing machinery (ACM) computing surveys 41(3):15:1–15:58. doi:10.1145/1541880.1541882
3. Cork L, Walker R (2007) Sensor fault detection for UAVs using a nonlinear dynamic model and the imm-ukf algorithm. In: Information, decision and control (IDC'07), IEEE, pp 230–235

4. Craighead J, Murphy R, Burke J, Goldiez B (2007) A survey of commercial open source unmanned vehicle simulators. In: Proceedings of the IEEE international conference on robotics and automation, pp 852–857
5. Daigle MJ, Koutsoukos XD, Biswas G (2009) A qualitative event-based approach to continuous systems diagnosis. IEEE Trans Control Syst Technol 17(4):780–793
6. Eski I, Erkaya S, Savas S, Yildirim S (2011) Fault detection on robot manipulators using artificial neural networks. Robot Comput Integr Manuf 27(1):115–123
7. FlightGear (2013) http://www.flightgear.org/.
8. Goel P, Dedeoglu G, Roumeliotis SI, Sukhatme GS (2000) Fault detection and identification in a mobile robot using multiple model estimation and neural network. In: Proceedings of the IEEE international conference on robotics and automation, pp 2302–2309
9. Hido S, Tsuboi Y, Kashima H, Sugiyama M, Kanamori T (2011) Statistical outlier detection using direct density ratio estimation. Knowl Inf Syst 26(2):309–336
10. Hofbaur M, Köb J, Steinbauer G, Wotawa F (2007) Improving robustness of mobile robots using model-based reasoning. J Intell Robot Syst 48(1):37–54. doi:10.1007/s10846-006-9102-0
11. Hwang I, Kim S, Kim Y, Seah CE (2010) A survey of fault detection, isolation, and reconfiguration methods. IEEE Trans Control Syst Technol 18(3):636–653
12. Khalastchi E, Kaminka GA, Kalech M, Lin R (2011) Online anomaly detection in unmanned vehicles. In: Proceedings of the 10th international conference on autonomous agents and multiagent systems, AAMAS '11, International foundation for Autonomous Agents and Multiagent Systems, pp 115–122. http://dl.acm.org/citation.cfm?id=2030470.2030487
13. lab A (2013). http://c3.nasa.gov/dashlink/projects/3/
14. Laurikkala J, Juhola M, Kentala E, Lavrac N, Miksch S, Kavsek B (2000) Informal identification of outliers in medical data. In: Proceedings of the 5th international workshop on intelligent data analysis in medicine and pharmacology, pp 20–24. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.36.7407
15. Leveson NG, Turner CS (1993) An investigation of the therac-25 accidents. Computer 26(7):18–41
16. Lin R, Khalastchi E, Kaminka GA (2010) Detecting anomalies in unmanned vehicles using the mahalanobis distance. In: Proceedings of the IEEE international conference on robotics and automation, IEEE, pp 3038–3044
17. Lishuang X, Tao C, Fang D (2011) Sensor fault diagnosis based on least squares support vector machine online prediction. In: IEEE conference on robotics, automation and mechatronics (RAM), IEEE, pp 275–279
18. Mahalanobis PC (1936) On the generalized distance in statistics. In: Proceedings of the National Institute of Sciences, vol 2, ppp 49–55
19. Manevitz LM, Yousef M (2002) One-class svms for document classification. J Mach Learn Res 2, pp 139–154. http://portal.acm.org/citation.cfm?id=944790.944808
20. Masud M, Woolam C, Gao J, Khan L, Han J, Hamlen K, Oza N (2012) Facing the reality of data stream classification: coping with scarcity of labeled data. Knowl Inf Syst 33(1): 213–244. doi:10.1007/s10115-011-0447-8
21. Oates T, Schmill MD, Gregory DE, Cohen PR (1995) Detecting complex dependencies in categorical data. In: Fisher D, Lenz H (eds) Learning from data: artificial intelligence and statistics, Springer, pp 185–195
22. Pearson K (1894) Contributions to the mathematical theory of evolution. Philos Trans R Soc Lond A 185:71–110
23. Pokrajac D, Lazarevic A, Latecki LJ (2007) Incremental local outlier detection for data streams. In: The symposium on computational intelligence and data mining, CIDM'07, IEEE, pp 504–515
24. Ratsch G, Mika S, Scholkopf B, Muller K-R (2002) Constructing boosting algorithms from SVMs: an application to one-class classification. IEEE Trans Pattern Anal Mach Intell 24(9):1184–1199
25. Ruiz-del Solar J, Chown E, Ploeger PG (2011) RoboCup 2010: robot soccer world cup XIV, vol. 6556, Springer, Berlin
26. Sorton EF, Hammaker S (2005) Simulated flight testing of an autonomous unmanned aerial vehicle using flight-gear. American Institute of Aeronautics and Astronautics 2005–7083, Institute for Scientific Research
27. Steinbauer G (2013) A survey about faults of robots used in robocup. In: RoboCup 2012: robot soccer world cup XVI', Springer, Berlin, pp 344–355
28. Steinbauer G, Mörth M, Wotawa F (2006) Real-time diagnosis and repair of faults of robot control software. In: Bredenfeld A, Jacoff A, Noda I, Takahashi Y (eds) RoboCup 2005: robot soccer world cup IX', Springer, pp 13–23. http://dl.acm.org/citation.cfm?id=2124160.2124164
29. Steinwart I, Christmann A (2008) Support vector machines. Springer, Berlin

30. Sundvall P, Jensfelt P (2006) Fault detection for mobile robots using redundant positioning systems. In: ICRA, pp 3781–3786
31. SVM light (2010). http://svmlight.joachims.org
32. Travé-Massuyès L (2014) Bridging control and artificial intelligence theories for diagnosis: a survey. Eng Appl Artif Intell 27:1–16
33. Zaman S, Steinbauer G, Maurer J, Lepej P, Uran S (2013) An integrated model-based diagnosis and repair architecture for ros-based robot systems In: IEEE international conference on robotics and automation (ICRA), IEEE, pp 482–489

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 is a Ph.D student 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.

Gal A. Kaminka is a professor at the Computer Science Department and the brain sciences research center, at Bar Ilan University (Israel), where he chairs the Bar Ilan University Robotics Consortium, and his MAVERICK research group. His research expertise includes multiagent and multirobot systems, teamwork and coordination, behavior and plan recognition, and modeling social behavior. He received his PhD from the University of Southern California (2000), spent time as a postdoctorate fellow at Carnegie Mellon University (until 2002), and a year as a Radcliffe Fellow at Harvard University's Radcliffe Institute for Advanced Study (2012). Prof. Kaminka was awarded an IBM faculty award and top places at international robotics competitions. He is the 2013 recipient of the Israeli national Landau Prize in exact sciences.

Raz Lin received the B.Sc. degree (summa cum laude) in mathematics and computer science, the M.Sc. degree (magna cum laude) in computer science and the Ph.D. degree from the Bar-Ilan University, Ramat-Gan, Israel, in 2001, 2002 and 2008, respectively. He is currently a Postdoctoral Fellow with the Department of Computer Science, Bar-Ilan University, where he investigates issues of automated negotiations, personalization, training and learning, and where he also received a four-year Presidents scholarship for outstanding students.